ASPECT-ORIENTED SOFTWARE DEVELOPMENT

— A Little Guidance to Better Java Applications —

Hinrich E. G. Bonin[1]

Jan-2002 – Feb-2005

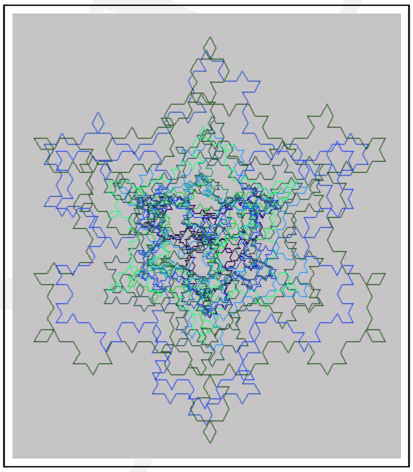
[1]Prof. Dr. rer. publ. Dipl.-Ing. Dipl.-Wirtsch.-Ing. Hinrich E. G. Bonin, University of Applied Sciences, Fachhochschule Nordostniedersachsen, Volgershall 1, D-21339 Lüneburg, Germany.

Contents

List of Figures

List of Tables

Abstract



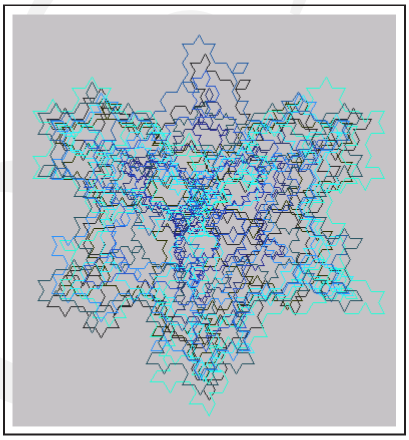
Post-object software development is based on the idea that a system is better designed and programmed by separately specifying the system's areas of interest. The aspect-oriented paradigm adresses *crosscutting concerns* through modularization, thus alleviating and controlling much of the code's tangling potential. There are two aspects of crosscutting concerns:

- *Design level*: concerns that crosscut
- *Implementation level*: a programming construct enables crosscutting concerns to be captured in modular units.

AspectJ is a powerful language that provides support for the implementation of crosscutting concerns through *join points* (collection of principle points in the execution of the base program), and *advices* (method-like structures attached to join points). Precedence rules are defined when more than one advice apply at a join point.

Chapter 1

Preface



"The choice of programming language is important
because it influences one's point of view."
The Gang-of-Four design patterns
↪ [Gamma+, 1994], p. 4

Since you have obtained a copy of ¨ASPECT-ORIENTED SOFTWARE DEVELOPMENT¨ and have started reading it, I assume that you already have an intrest in aspect-oriented software development (AOSD) and/or ascept-programming (AOP). Java^{TM} programmers are attracted by the the paradigm of object-oriented programming (OOP) and perhaps know the term *crosscutting concerns*.

Modularity helps when developing systems. Modules can be replace by other modules while the rest of the system stays intact. For instance, in the architecture of buildings, plans for rooms, water, gas, and electricity are specified separately. When an architect wants to exchange parts of the electricity support for a room, he never exchanges the complete room. Instead, he only modifies the electricity plan of the room, which does not affect the other plans. After all plans are finished, the construction process integrates them into the physical layout of the building and eliminates remaining conflicts. This principle of *aspect separation* can also be used in software engineering. Aspects of software, such as persistence, debugging, or animation, should be described separately and exchanged independently wihtout disturbing the modular structure of the system (↪ [Aßmann, 2003] p. 12,13).

ASPECT-ORIENTED SOFTWARE DEVELOPMENT has the great conceptual vision that we could provide independent specifications for each distinct *concern* and then *weave* them together to build a valid application. Aspects are abstractions capturing and localizing crosscutting concerns e. g. a code which cannot be encapsulated within one class but that is spread over many classes. Like objects, aspects may arise at any stage of a software lifecycle, including requirements specification, design, implementation,etc.

This book[1] will help to improve a complex Java application on the base of the aspect-oriented pradigm. The examples are coded in AspectJ[2], a simple and practical AO extension to Java, widely used in the AOP research community. AspectJ has been developed at the Xerox Palo Alto Research center (↪ p. 281) and provides a *static aspect weaver*, and other development tools. AspectJ makes both name- and property-based crosscutting possible. AspectJ programs are clean modular implementations of crosscutting concerns.

AspectJ is making the following key definitions ([Lieberherr+, 2001], p. 40):

Join point

- *Join points* are principled points in the execution of the program.

[1] Remark:This is a **draft document** and continues to be revised. The latest version can be found at `http://as.fhnon.de/publikation/anwdall.pdf`. Please send comments to `mailto:bonin@fhnon.de`

[2] exactly: `ajc` version 1.0.1 (built 18.12.2001 11:11 PST) running on java 1.3.1. or AspectJ Compiler 1.1.0 running on java version "1.4.0_01"; Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.0_01-b03) Java HotSpot(TM) Client VM (build 1.4.0_01-b03, mixed mode). `AspectJ`^{TM} is a trade mark of Xerox Cooperation.

Pointcut

Advice

- *Pointcuts* are a means of referring to collections of join points and certain values at those join points.
- *Advice* is a method-like construct that can be attached to pointcuts;
- *aspects* are modular units of crosscutting implentations, comprised of pointcuts, advice, and ordinary Java member declarations.

Aspect

POP

Examples of post-object programming (POP) technologies include: Domain-specific Languages, Gernerative Programming, Generic Programming, Constraint Languages, Reflection and Meta Programming , Feature-Oriented Development, Views & Viewpoints, and Asynchronous Message Brokering ([Elrad et al., 2001a], p. 30). AOP is one of the most promising alternatives to improve classical OOP ([Pace / Campo, 2001], p. 67).

concerns

AOP is based on the paradigm that computers are better programmed by separately specifing the various *concerns* (properties or areas of interest) of a system.[3]. A software engineering environment weaves or composes these concerns and the descriptions of their mutual relationships into the runnable program. While OOP tries to find communality among classes and pushes it up the inheritance tree, AOP attempts to realize scattered concerns as first-class elements, and ejects them horizontally from the object structure. ”AOP is focused on mechanisms for simplifying the realizations of such crosscutting concerns.” ([Elrad et al., 2001a], p. 30)

cross-cutting

Aspect-oriented languages have three critical elements ([Kiczales+, 2001], p. 60):

1. a join point model
2. a means of indentifying join points, and
3. a means of affecting implementation at join points.

Aspect-oriented software development introduces a new paradigm that complements existing ones (↪ Table 1.1 p. 16). A new paradigm brings new options, but also new problems, e. g. when several aspects have to compose to an application, a given aspect not only crosscuts the

inter-aspects composition

[3]“A *concern* is some functionality or requirement necessary in a system which has

	Structured programming	**Modular programming**	**Data abstraction**	**Object-oriented programming**
Paradigm	Explicit control constructs	Information hiding	Hiding the representation of data	Objects, with classification and specialization
Constructs of the programming language	Do, while and other loops, blocks, etc.	Modules with well-defined enforced interfaces	Types	Classes, objects, polymorphism

Legend:
Idea ↪ [Elrad+, 2001], p. 36.

Table 1.1: Software Engineering Paradigms compared.

Joint Point	**The aspect or the class knows** ...
Aspect-directional	The class knows about the aspect.
Class-directional	The aspect knows about the class.
Open	Both classes and aspects know about each other.
Closed	Neither the aspect nor the class know about the other.

Legend:
Classification ↪ [Kersten / Murphy, 1999]

Table 1.2: Classification of Join Points

application, but may also crosscut other aspects. This issue is called the *inter-aspects composition* aspect ([Pawlak+, 2001], p. 2). The solotion in AspectJ is based on precedence rules.

Weave-Time Issues The weave-time issues occur when the weaver weaves the aspects (or a particular aspect) into the base program. Generally speaking, weaving can be done at: *Weaving*

compile-time The source code from the primary and the aspect lagugages is weaved before being put through the phases of the compiler where byte code is produced.[4]

link-time The weaving occurs after the primary and aspect language code has been compiled into byte code.[5]

load-time The weaving occurs when classes are loaded by the classloader. Ultimately, the weaving is at the byte-code level.

run-time The virtual maschine is responsible for detecting join points and loading and execution aspects.

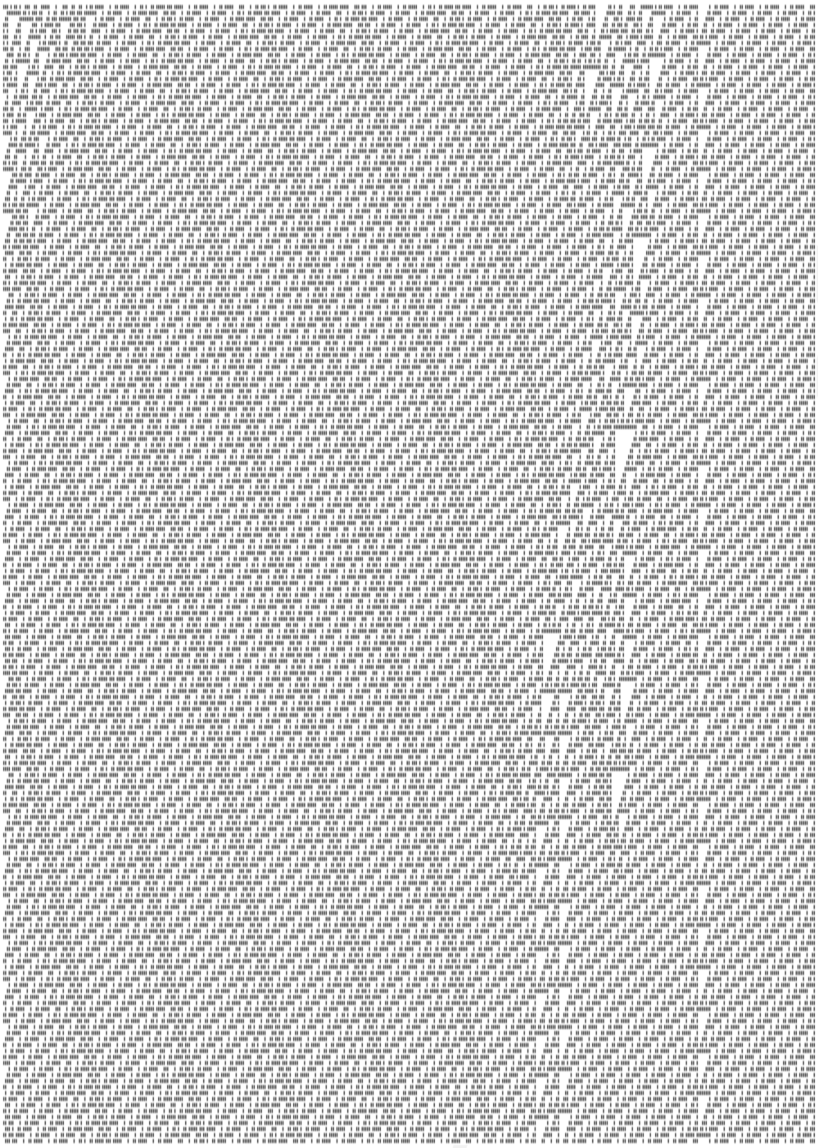
In *compile-time* weaving, the aspect code is analyzed, converted to the primary language if needed, and inserted directly into the base program. A *link-time* or *run-time* weaver waits until runtime to handle the weave. A processor is used to place *hooks* in the code of the base program. *hooks* When the hooks are executed, a modified runtime systems determines whether any aspects need to execute. A compile-time weaving system has some shortcomings. First it needs the source code of the base program for all aspects. For example, the convenience features like JAR files cannot be used. Second it is unable to perform a dynamically change of an aspect.

---

been implemented in a code structure. This definition allows a concern to not be limited to object-oriented systems — a structured system can also have concerns." ↪ [Gradecki+, 2003] p. 4

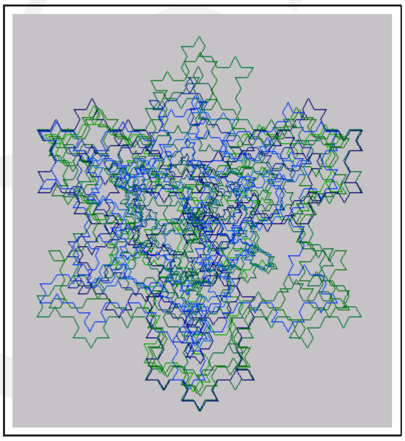
[4] `ajc version 1.0.x` uses this form of weaving.

[5] `ajc version 1.1.x` will use this form of weaving.



Chapter 2

From OOP to AOP



The term *aspect-oriented programming* (AOP) is attributed to Gregor Kiczales et. al. ([Kiczales+, 1997]).

> ”An aspect is a modular unit of crosscutting implementation. It is defined very much like a class, and can have methods, fields, constructors, intializers, named pointcuts, and advice.” ([Kiczales+, 2001], p. 61)

aspect

The application is obtained by weaving the primary structure with the crosscutting aspects ([Pace / Campo, 2001],p. 68).

Anyway, some programmers have always found the naming conventions of AOP confusing because *they refer to Aspects, Concerns, and*

Crosscuts (like the old saw?) and none of them means what it usually means (↪ [Cooper, 2003]).

2.1 OO Class: `HelloWorld`

The traditional choice for the first program in a new programming paradigm is a program that displays a simple greeting: ”Hello, World!”. We are following that tradition. The goal for AOP is to build on OOP by supporting the separation of those concerns that OOP handles poorly. That is why, we start to introduce an object-oriented example of ”Hello, World!”, compiled with AspectJ. We will just give a brief introduction of OOP covering the details that are necessary for understanding AOP.

class

instance

An object-oriented application is organized around *classes*. During execution of a *Java*^{TM}[1] program, objects (*instances*) of these classes are created, initialized, and then manipulated. A class must be defined before instances can be created. In the Java world, a new class is specified as an incremental modification of one other previously defined class; by default its *super* class is the class `java.lang.Object`. A new class is said to *inherit*, which means that its effective definition is a combination of what is explicit in its own definition and what is in those of the superclass it inherits from.

`package`

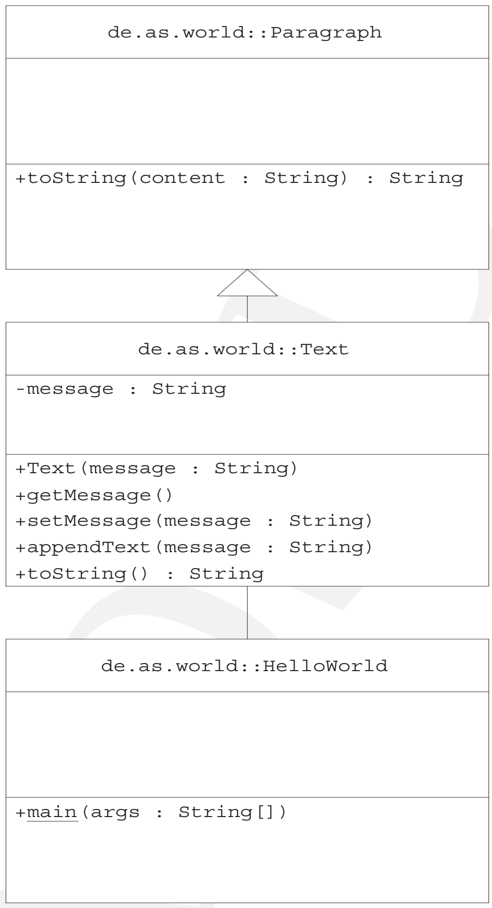
In Java, packages provide a structuring mechanism of classes. A Java package is a set of related classes. In our example we use the statement:

```
package de.as.world;
```

for the three classes (↪ class diagram figure 2.1 p. 21):

- `HelloWorld` (↪ p. 23)
  This class uses the contructor `Text(String message)` to build a instance of the class `Text`, referred by `myText`. To print out `myText`, we need a representation as a `String` object. The method `toString()`, implicit called, does this converting.

[1]Note that the use of the general term Java^{TM} implies in fact two meanings: on the one hand, *Java as a programming language*, on the other hand, the *Java Virtual Machine* (JVM), which is not necessarily targeted by the Java Language exclusively, but may be used by other languages as well.



Legende:
UML ≡ Unified Modeling Language
(UML2 Proposals 2002 ↪ `http://www.community-ML.org/`)

`Foo`	≡	Class `Foo`
——	≡	Association
`Foo` ◁— `Bar`	≡	Inheritance; A `Bar`-object has the attributes of the class `Foo` and can use the methods of the class `Foo`.

Figure 2.1: UML Class Diagram: `HelloWorld`-Example

- Text (↪ p. 23)
  This class has the slot[2] message and the get- and set-methods for this private slot. To distinguish the parameter message from the slot message the constructor and the set-method contain the this operator.

- Paragraph (↪ p. 24)
  This class adds the extensible markup language (XML[3]) tags <paragaph> and </paragraph> to the string, which is the argument of the method toString(String content). The class Text is subclass of the class Paragaraph. To convert myText into a String, the system calls the method toString() in the class Text and this method calls the toString(String content). The result is, we get the text Hello, World! in a <paragraph>-container.

To compile these Java files with AspectJ we start a command line interface and type:[4]

```
>ajc -argfile de/as/world/files.lst
```

The file files.lst in the path de/as/world is a line-delimited list of arguments. These arguments are inserted into the argument list. To produce a Java documentation we type:

```
>ajdoc -argfile de/as/world/files.lst
```

To execute the Java application we type:

```
>java de.as.world.HelloWorld
```

The string de.as.world is the name of the package, the string HelloWorld is the name of the class. The result is displayed on the standard output device:

```
<paragraph>Hello, World!</paragraph>
```

[2]Also called attribut or variable. Starting from the old good days of CLOS (Common Lisp Object System) we are used to say *slot* (↪ [Bonin, 1991]).

[3]For a XML tutorial and Handbook ↪ [Goldfarb / Prescod, 2002].

[4]see also the logging file HelloWorld.log (↪ p. 25)

Class `HelloWorld`

```
/**
 *  "Hello, World!" Application
 *
 *@author     Bonin
 *@version    1.1
 */

package de.as.world;

public class HelloWorld
{
   public static void main(String[] args)
   {
      Text myText;

      myText = new Text("Hello, ");
      myText.appendText("World!");

      System.out.println(myText);
   }
}
```

Class `Text`

```
/**
 *  Text for the "Hello, World!" Application
 *
 *@author     Bonin
 *@version    1.1
 */

package de.as.world;

public class Text extends Paragraph
{
   private String message;


   public Text(String message)
   {
```

```
      this.message = message;
   }


   public String getMessage()
   {
      return message;
   }


   public void setMessage(String message)
   {
      this.message = message;
   }


   public void appendText(String message)
   {
      this.setMessage(this.getMessage() +
            message);
   }


   public String toString()
   {
      return toString(this.getMessage());
   }
}
```

Class Paragraph

```
/**
 *  Mark Up the Text of the "Hello, World!" Application
 *
 *@author     Bonin
 *@version    1.1
 */

package de.as.world;

public class Paragraph
{
```

```
   public String toString(String content)
   {
      return "<paragraph>" +
            content +
            "</paragraph>";
   }
}
```

Argument list files.lst

```
HelloWorld.java
Paragraph.java
Text.java
```

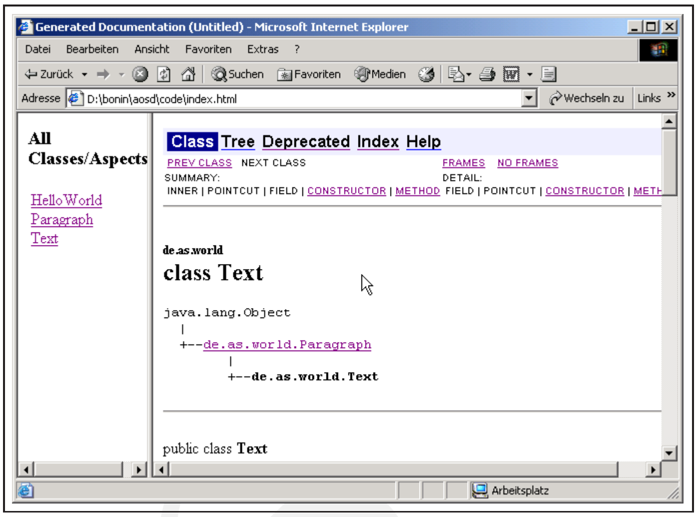
Protocol HelloWorld.log

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.3
    (built 08.02.2002 12:47 PST) running on java 1.3.1

D:\bonin\aosd\code>ajdoc -argfile de/as/world/files.lst

D:\bonin\aosd\code>ajdoc -argfile de/as/world/files.lst
Starting compile...
Loading source file D:\bonin\aosd\code\de\as\world\HelloWorld.java...
Loading source file D:\bonin\aosd\code\de\as\world\Paragraph.java...
Loading source file D:\bonin\aosd\code\de\as\world\Text.java...
Creating root...
Generating documentation...
Building tree for all the packages and classes...
Generating overview-tree.html...
Building index for all the packages and classes...
Generating index-all.html...
Generating deprecated-list.html...
Building index for all classes...
Generating allclasses-frame.html...
Generating index.html...
Generating packages.html...
Generating overview-summary.html...
Generating de\as\world\HelloWorld.html...
Generating de\as\world\Paragraph.html...
Generating de\as\world\Text.html...
Generating serialized-form.html...
Generating package-list...
```

```
Generating help-doc.html...
Generating stylesheet.css...

D:\bonin\aosd\code>java de.as.world.HelloWorld
<paragraph>Hello, World!</paragraph>

D:\bonin\aosd\code>cd de\as\world

D:\bonin\aosd\code\de\as\world>dir
            44 files.lst
           800 HelloWorld.class
         7.541 HelloWorld.html
           340 HelloWorld.java
         1.205 HelloWorld.log
           694 Paragraph.class
         7.752 Paragraph.html
           310 Paragraph.java
           954 Text.class
         9.587 Text.html
           686 Text.java

D:\bonin\aosd\code\de\as\world>
```

Java Documation File `index.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
"http://www.w3.org/TR/REC-html40/loose.dtd>
<!--NewPage-->
<HTML>
<HEAD>
<!-- Generated by javadoc on Fri May 31 14:43:42 CEST 2002-->
<TITLE>
Generated Documentation (Untitled)
</TITLE>
</HEAD>
<FRAMESET cols="20%,80%">
<FRAME src="allclasses-frame.html" name="packageFrame">
<FRAME src="de/as/world/HelloWorld.html" name="classFrame">
</FRAMESET>
<NOFRAMES>
<H2>
Frame Alert</H2>

<P>
```



Legende:
`D:\bonin\aosd\code\index.html` with Browser *Microsoft Internet Explorer Version 6*

Figure 2.2: Generated Documentation: `HelloWorld`-Example

```
This document is designed to be viewed using the frames feature.
If you see this message, you are using a non-frame-capable web client.
<BR>
Link to <A HREF="de/as/world/HelloWorld.html">Non-frame version.</A>
</NOFRAMES>
</HTML>
```

See result of `D:\bonin\aosd\code\index.html` with Browser *Microsoft Internet Explorer 6* ↪ figure 2.2, p. 27.

2.2 Computational Reflection

Reflection is thinking and computing about oneself with help of the metadata. Reflection enables a base level to reason about itself with the help of its metalevel. Hence, a refective architecture keeps a causal connection between base level and metalevel: every time the system changes, the metadata also changes, and every time the metadata changes, the system changes (↪ [Aßmann, 2003] p. 50).

2.2.1 Java's Reflection

A program property *reflection* enables the program to access its internal structure and behavior. *Computational reflection* enables the program to manipulate that structure, thereby modifying its behavior. Java's reflection based on the `java.lang.reflect`-package, which represents the members of a class with `Method`-, `Constructor`-, and `Field`-objects. Java's reflection is *read-only* ([Sullivan, 2001], p. 96). For example, a program can query the methods of a class, but a program cannot dynamically change the methods of a class. Full reflection allows modification of any *meta*-information that can be reified.

Java can be regarded as a simple reflective system. It provides a partial metamodel with types, such as `Class`, `Method`, `Constructor`, and `Field`, but not *Statement*. Java permits metaprogramming, since it is possible to contstruct, to compile, to load, and to execute classes.

The following example `Foo`(↪ p. 29) invoke both methods `getSlot()` and `setSlot()`, without knowing their names (↪ figure 2.3 p. 29).

In the first step we create an instance of the class `Foo`, named `myF`, in the method `main()`. Applied the method `getClass()` to this instance `myF` returned an object of type `Class`, here named `myClass`. From this object we select the methods of the class `Foo` as an array.

`getClass()`

```
Class myClass = myF.getClass();
Method[] myMethods =
  myClass.getDeclaredMethods();
```

In the second step we choose one method by the index with the type `Method`.

Foo.main()

Make an object of Foo
Get the class of this object
Get declared methods of this class
Print signature of all methods
Invoke the second method
Invoke the first method

Legende:
Java source code ↪ 2.2.1 p. 29

Figure 2.3: Foo.main() — Structure Diagram

```
Method m2 =  myMethods[2];
```

You see m2 is an object. This object m2 represents the method setSlot(). It knows the method
invoke(java.lang.Object, java.lang.Object[])
in java.lang.reflect.Method. The first parameter binds the object to which we apply the method m2. The second parameter binds the arguments for m2. Because there could be more than one argument it is an object array. To apply the method setSlot() with the argument "Hello, World!" we have to write: invoke()

```
Object[] arguments = new Object[]{"Hello, World!"};
m2.invoke(myF, arguments);
```

The method invoke() throws the both exceptions: IllegalAccessException and InvocationTargetException. That is why we have to code the try/catch-construction.

Class Foo

```
/**
```

```
 *  Reflection example "Invoke a method"
 *
 *@author     Bonin
 *@version    1.1
 */

package foo;

import java.lang.reflect.*;

public class Foo
{
   private String slot;


   public String getSlot()
   {
      return slot;
   }


   public void setSlot(String slot)
   {
      this.slot = slot;
   }


   Foo(String slot)
   {
      this.slot = slot;
   }


   public static void main(String[] args)
   {
      Foo myF = new Foo("Just started!");
      Class myClass = myF.getClass();

      System.out.println("Class: " + myClass.getName());

      Method[] myMethods = myClass.getDeclaredMethods();
```

```
        // Print out the signatur of the methods
        for (int i = 0; i < myMethods.length; i++)
        {
           Method m = myMethods[i];
           System.out.println("Method: " + m);
        }

        // Invocation of method setSlot()
        Object[] arguments
               = new Object[]{"Hello, World!"};
        try
        {
           Method m2 = myMethods[2];
           m2.invoke(myF, arguments);
        } catch (IllegalAccessException e)
        {
           System.err.println
                 ("Illegal access: " + e);
        } catch (InvocationTargetException e)
        {
           System.err.println
                 ("Invocation exception: " + e);
        }

        // Invocation of method getSlot()
        try
        {
           Method m1 = myMethods[1];
           System.out.println
                 (m1.invoke(myF, null));
        } catch (IllegalAccessException e)
        {
           System.err.println
                 ("Illegal access: " + e);
        } catch (InvocationTargetException e)
        {
           System.err.println
                 ("Invocation exception: " + e);
        }
    }
}
```

Protocol `Foo.log`

```
D:\bonin\aosd\code>java -fullversion
java full version "1.3.1-b24"

D:\bonin\aosd\code>javac foo/Foo.java

D:\bonin\aosd\code>java foo.Foo
Class: foo.Foo
Method: public static void foo.Foo.main(java.lang.String[])
Method: public java.lang.String foo.Foo.getSlot()
Method: public void foo.Foo.setSlot(java.lang.String)
Hello, World!

D:\bonin\aosd\code>
```

You might notice that the names of method parameters are not shown; parameter names are not stored in class files, so they are not available through the reflection package.

The next example demonstrates another use of the reflection package. The class `Bar` (↪ p. 32) loads the class identifed by a string of its name and builds an instance of this class. The relevant statement to load a class is:

`forName()`

```
Class myClass = Class.forName(args[0]);
```

If we load the class `Foo` (↪ p. 29) then we will have only one constructor with one argument of the type `String`. We get this constructor by the index value zero.

```
Constructor c0 = (Constructor) myConstructors[0];
Object[] arguments = new Object[]{"Hello, World!"};
```

`newInstance()`

The constructor `c0` is an object knowing the method `newInstance()`.

```
Object myO = c0.newInstance(arguments);
```

Class `Bar`

```
/**
 *  Reflection example "Load a class by name"
 *
 *@author     Bonin
```

```
 *@version    1.0
 */

package foo;

import java.lang.reflect.*;

public class Bar
{
   public static void main(String[] args)
          throws ClassNotFoundException
   {
      Class myClass = Class.forName(args[0]);

      Constructor[] myConstructors
             = myClass.getDeclaredConstructors();
      for (int i = 0; i < myConstructors.length; i++)
      {
         Constructor c
                = (Constructor) myConstructors[i];
         System.out.println
               ("Constructor " + i + ": " + c);
      }

      try
      {
         Constructor c0
                = (Constructor) myConstructors[0];
         Object[] arguments
                = new Object[]{"Hello, World!"};
         Object myO =
               c0.newInstance(arguments);
         Class myC =
               myO.getClass();
         Method[] myM =
               myC.getDeclaredMethods();

         System.out.println
               ("c0.getDeclaringClass() : " +
               c0.getDeclaringClass());
         System.out.println
               ("myO.getClass() : " +
```

```
               myO.getClass());

         // Invocation of method getSlot()
         System.out.println
               ("myM[1].invoke(myO, null) : " +
               myM[1].invoke(myO, null));

      } catch (InstantiationException e)
      {
         System.err.println
               ("Illegal access: " + e);
      } catch (IllegalAccessException e)
      {
         System.err.println
               ("Illegal access: " + e);
      } catch (IllegalArgumentException e)
      {
         System.err.println
               ("Invocation exception: " + e);
      } catch (InvocationTargetException e)
      {
         System.err.println
               ("Invocation exception: " + e);
      }
   }
}
```

Protocol `Bar.log`

```
C:\bonin\aosd\code>java -fullversion
java full version "1.4.0_01-b03"

C:\bonin\aosd\code>javac foo/Bar.java

C:\bonin\aosd\code>java foo.Bar foo.Foo
Constructor 0: foo.Foo(java.lang.String)
c0.getDeclaringClass() : class foo.Foo
myO.getClass() : class foo.Foo
myM[1].invoke(myO, null) : Hello, World!

C:\bonin\aosd\code>
```

2.2.2 Metaobject Protocol

Metaobject protocols (MOP[5]) are interfaces to a language that gives programmers the ability to incrementally modify the language's behavior and implementation, as well as the ability to write programs with that language. It introduces a metaphor from theater, distinguishing between *on-stage* objects seen by the audience and *backstage* machinery needed to make *on-stage* objects behave correctly. Finally, implementors are the *producers*: they get to see what happens both *on* and *off* stage, and they are the ones responsible for putting on the show. ([Kiczales+, 1991],p. 13)

2.3 Design by Contract

Conditions

In this style of programming, explicit preconditions test that callers of a method call it properly, and explicit postconditions test that methods properly do the work they are supposed to do. AspectJ makes it possible to implement pre- and post-condition testing in modular form.

The following simple example `RoadApplication` shows the implemention of such conditions in two versions. First, the contract is implement as a common method in the same class (↪ p.36), and second, it is implemented as an aspect with an additional method and an advice `around()` (↪ p. 39).

2.3.1 Common Solution: `checkRoadValue()`

The methods `getRoad()` and `setRoad()` are checking a value restriction by calling an additional method. This method `checkRoadValue()`, which returns `true` or `false`, depending of the fullfillment of the restriction, is called in the bodies of both methods. In the case of return value `false`, the declared default value is used instead of the original value. The code for this checking reduces the transparency. For example, if we think about the method `getRoad()`, we don't like to think about a value restriction. Fullfilling the value restriction is an

[5]There are a lot of MOP tools; e. g. Jonathan Bachrach at MIT has developed the *Java Syntactic Extender ↪ p. 282.*, a procedural macro system, combined with a runtime MOP.

other thought. That is why, we isolate this thought as a special aspect. Later, we implent it as an own piece of code (↪ Section 2.3.2 p. 39).

Argument list `filesI.lst`

```
RoadI.java
MyRoadI.java
RoadApplicationI.java
```

Class `RoadI`

```
/**
 *  "Three ways to go"
 *
 *@since      23-May-2003
 *@author     Hinrich Bonin
 *@version    1.0
 */
package de.fhnon.nemo.checking;

public abstract class RoadI
{
   int road;
   final int roadDefaultValue = 1;


   int getRoad()
   {
      if (checkRoadValue(road))
      {
         return road;
      } else
      {
         System.out.println("Use roadDefaultValue!");
         return roadDefaultValue;
      }
   }


   void setRoad(int road)
   {
      if (checkRoadValue(road))
      {
         this.road = road;
      } else
```

```
      {
         System.out.println("Set road to roadDefaultValue!");
         this.road = roadDefaultValue;
      }

   }


   void printWhereToGo()
   {
      switch (getRoad())
      {
         case 1:
            System.out.println("Go to one!");
            break;
         case 2:
            System.out.println("Go to two!");
            break;
         case 3:
            System.out.println("Go to three!");
            break;
      }
   }



   boolean checkRoadValue(int road)
   {
      if ((road >= 1) && (road <= 3))
      {
         return true;
      } else
      {
         return false;
      }
   }


}
```

Class `MyRoadI`

```
/**
 *  "Three ways to go"
 *
```

```
 *@since      24-May-2003
 *@author     Hinrich Bonin
 *@version    1.0
 */
package de.fhnon.nemo.checking;

public class MyRoadI extends RoadI
{
   MyRoadI(int road)
   {
      this.road = road;
   }
}
```

Class `RoadApplicationI`

```
/**
 *  "Three ways to go"
 *
 *@since      23-May-2003
 *@author     Hinrich Bonin
 *@version    1.0
 */
package de.fhnon.nemo.checking;

public class RoadApplicationI
{

   public static void main(String[] args)
   {
      MyRoadI road = new MyRoadI(0);
      road.printWhereToGo();

      road.setRoad(4);
      road.printWhereToGo();
   }
}
```

Protocol `RoadApplicationI.log`

```
C:\bonin\aosd\code>ajc -version
ajc version 1.0.6 (built 24.07.2002 18:21 PST) running on java 1.4.0_01

C:\bonin\aosd\code>ajc -argfile de/fhnon/nemo/checking/filesI.lst
```

```
C:\bonin\aosd\code>java de.fhnon.nemo.checking.RoadApplicationI
Use roadDefaultValue!
Go to one!
Set road to roadDefaultValue!
Go to one!

C:\bonin\aosd\code>
```

2.3.2 Aspect Solution: `Checker`

The aspect `Checker` implements the testing of our value restriction. In this example, only we controll the call of the method `getRaod()`. For that purpose we add the method `postCondition()` to the class `RoadII`. This method and all of the testing is coded in the aspect `Checker`. Even the default value is coded in this aspect. The transparency of the method `getRoad()` in the class `RoadII` is not affected.

Argument list `filesII.lst`

```
RoadII.java
MyRoadII.java
RoadApplicationII.java
Checker.java
```

Aspect `Checker`

```
/**
 *  "Three ways to go"
 *
 *@since      23-May-2003
 *@author     Hinrich Bonin
 *@version    1.0
 */
package de.fhnon.nemo.checking;

public aspect Checker
{
   final int RoadII.roadDefaultValue = 1;

   int RoadII.postCondition()
   {
```

```
      if ((road >= 1) && (road <= 3))
      {
         System.out.println("Value OK!");
         return road;
      } else
      {
         System.out.println("Use roadDefaultValue!");
         return roadDefaultValue;
      }

   }


/* This advice traps the execution of the join point;
 * it runs instead of the join point.
 */
   pointcut checking(RoadII r):
        target(r) && call(int getRoad());


   int around (RoadII r)  : checking(r)
   {
      return r.postCondition();
   }
}
```

Class `RoadII`

```
/**
 *  "Three ways to go"
 *
 *@since      24-May-2003
 *@author     Hinrich Bonin
 *@version    1.0
 */
package de.fhnon.nemo.checking;

public abstract class RoadII
{
   int road;


   int getRoad()
   {
      return road;
   }
```

```
   void setRoad(int road)
   {
      this.road = road;

   }


   void printWhereToGo()
   {
      switch (getRoad())
      {
         case 1:
            System.out.println("Go to one!");
            break;
         case 2:
            System.out.println("Go to two!");
            break;
         case 3:
            System.out.println("Go to three!");
            break;
      }
   }
}
```

Class `MyRoadII`

```
/**
 *  "Three ways to go"
 *
 *@since      24-May-2003
 *@author     Hinrich Bonin
 *@version    1.0
 */
package de.fhnon.nemo.checking;

public class MyRoadII extends RoadII
{

   MyRoadII(int road)
   {
      this.road = road;
   }
}
```

Class `RoadApplicationII`

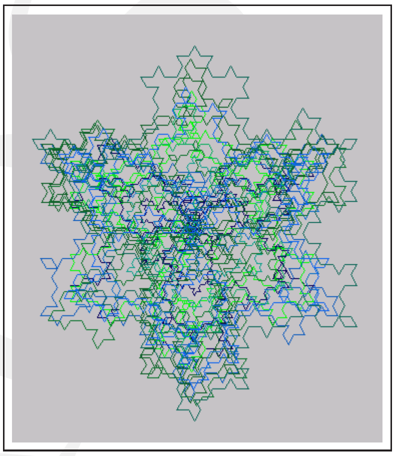
```
/**
 *  "Three ways to go"
 *
 *@since      23-May-2003
 *@author     Hinrich Bonin
 *@version    1.0
 */
package de.fhnon.nemo.checking;

public class RoadApplicationII
{

   public static void main(String[] args)
   {
      MyRoadII road = new MyRoadII(0);
      road.printWhereToGo();

      road.setRoad(4);
      road.printWhereToGo();
   }
}
```

Protocol `RoadApplicationII.log`

```
C:\bonin\aosd\code>ajc -version
ajc version 1.0.6 (built 24.07.2002 18:21 PST) running on java 1.4.0_01

C:\bonin\aosd\code>ajc -argfile de/fhnon/nemo/checking/filesII.lst

C:\bonin\aosd\code>java de.fhnon.nemo.checking.RoadApplicationII
Use roadDefaultValue!
Go to one!
Use roadDefaultValue!
Go to one!

C:\bonin\aosd\code>
```

Chapter 3

AspectJ's Basic Techniques



"AspectJ is a seamless aspect-oriented extension to Java, which means that programming in AspectJ is effectively programming in Java plus aspects."
(↪ [Hannemann / Kiczales, 2002])

This chapter presents two basic techniques of using AspectJ, one each from the two fundamental ways of capturing crosscutting concerns: with dynamic join points and advice, and with a static introduction.

Advice changes an application's behavior. Introduction changes both an application's behavior and its structure.

3.1 Join Points and `thisJoinPoint`

The idee of the example that follows is documented in the `AspectJ` contribution[1] The logfile `Example.log` (↪ p. 47) shows the different results of the class `Example` (↪ p. 44), running without and with the aspect `Inspection` (↪ p. 46). The file `files.lst` (↪ p. 47) contains names of the files to be compiled.

The class `Example` defines two methods `foo()` and `bar()` with different parameter lists and return types. Both are called, with suitable arguments, by `Example`'s method `go()` which was invoked from within `main()`.[2]

Class `Example`

```
/**
 *  Example to illustrate join points
 *  (Idee Xerox Corporation).
 *
 *@since      25-May-2003
 *@author     Hinrich Bonin
 *@version    1.0
 */
package de.fhnon.nemo.illustration;

public class Example
{

   static Example e;


   void foo(int alpha, Object theta)
   {
      System.out.println
            ("Example.foo(" +
            alpha + ", " +
            theta + ")");
   }
```


[1] `ajc version 1.0.1`, located in path `examples\tjp`.

[2] The explanation that follows is also created form the AspectJ contribution, located in path `doc\progguide`.

```
   String bar(Integer gamma)
   {
     System.out.println
            ("Example.bar(" +
            gamma + ")");
      return "Example.bar(" + gamma + ")";
   }

   void go()
   {
      e = new Example();
      e.foo(1, e);
      System.out.println
            (e.bar(new Integer(7)));
   }


   public static void main(String[] args)
   {
      new Example().go();
   }
}
```

The aspect `Inspection` uses around advice to intercept the execution of methods `foo()` and `bar()` in class `Example`, and prints out information garnered from `thisJoinPoint` to the console. The pointcut `goCut()` is defined as

cflow()

```
cflow(this(Example)) && execution(void go())
```

so that only executions made in the control flow of `Example.go` are intercepted. The control flow from the method `go()` includes the execution of `go` itself, so the definition of the around advice includes `!execution(* go())` to exclude it from the set of executions advised.[3] The name of the method and that method's defining class are available as parts of the signature, found using the method `getSignature()` of either `thisJoinPoint` or `thisJoinPointStaticPart`.

execution()
around()

The static portions of the parameter details, the name and types of the parameters, can be accessed through the `CodeSignature` associated with the join point. All

[3]If we delete `!execution(* go())` from the advice `around()` we get the additional output:

```
Intercepted message:  go
in class:  de.fhnon.nemo.illustration.Example
Arguments:
Running original method:
⋮
  result:  null
```

execution join points have code signatures, so the cast to `CodeSignature` cannot fail. The dynamic portions of the parameter details, the actual values of the parameters, are accessed directly from the execution join point object.

Aspect `Inspection`

```
/**
 *  Example to illustrate join points
 *  (Idee Xerox Corporation).
 *
 *@since      25-May-2003
 *@author     Hinrich Bonin
 *@version    1.0
 */
package de.fhnon.nemo.illustration;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.reflect.CodeSignature;

aspect Inspection
{
    pointcut goCut():
        cflow(this(Example) && execution(void go()));

    pointcut demoExecs():
        within(Example) && execution(* *(..));

    Object around():
        demoExecs() && !execution(* go()) &&
        goCut()
    {
        System.out.println(
          "Intercepted message: " +
          thisJoinPointStaticPart.
            getSignature().getName());

        System.out.println(
          "in class: " +
          thisJoinPointStaticPart.getSignature().
            getDeclaringType().getName());

        printParameters(thisJoinPoint);

        System.out.println("Running original method:" );

        Object result = proceed();
```

```
        System.out.println(" result: " + result );
        return result;
   }

    static private void printParameters(
      JoinPoint jp)
      {
        System.out.println("Arguments: " );
        Object[] args = jp.getArgs();
        String[] names =
            ((CodeSignature)
            jp.getSignature()).
                 getParameterNames();
        Class[] types =
            ((CodeSignature)
              jp.getSignature()).
                getParameterTypes();

        for (int i = 0; i < args.length; i++) {
             System.out.println(
               "   "  + i + ". " + names[i] +
       " : " +
               types[i].getName() +
               " = " +
               args[i]);
      }
   }
}
```

Argument list `files.lst`

```
Example.java
Inspection.java
```

Protocol `Example.log`

```
C:\bonin\aosd\code>java -version
java version "1.4.0_01"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.0_01-b03)
Java HotSpot(TM) Client VM
  (build 1.4.0_01-b03, mixed mode)

C:\bonin\aosd\code>javac de/fhnon/nemo/illustration/Example.java

C:\bonin\aosd\code>java de/fhnon/nemo/illustration/Example
Example.foo(1, de.fhnon.nemo.illustration.Example@f4a24a)
Example.bar(7)
```

```
Example.bar(7)

C:\bonin\aosd\code>ajc -version
ajc version 1.0.6
  (built 24.07.2002 18:21 PST) running on java 1.4.0_01

C:\bonin\aosd\code>ajc -argfile de/fhnon/nemo/illustration/files.lst

C:\bonin\aosd\code>java de/fhnon/nemo/illustration/Example
Intercepted message: foo
in class: de.fhnon.nemo.illustration.Example
Arguments:
  0. alpha : int = 1
  1. theta : java.lang.Object = de.fhnon.nemo.illustration.Example@87aeca
Running original method:
Example.foo(1, de.fhnon.nemo.illustration.Example@87aeca)
 result: null
Intercepted message: bar
in class: de.fhnon.nemo.illustration.Example
Arguments:
  0. gamma : java.lang.Integer = 7
Running original method:
Example.bar(7)
 result: Example.bar(7)
Example.bar(7)

C:\bonin\aosd\code>
```

3.2 Roles and Views Using Introduction

Introduction is AspectJ's form for modifying classes and their hierarchy. Introduction adds new members to classes and alters the inheritance relationship between classes. Unlike advice that operates primarily dynamically, introduction operates statically at compilation time.

The AspectJ example that follows is documented in the file `Point.log` (↪ p. 55). The files

- `Point.java` (↪ p. 49),
- `CloneablePoint.java` (↪ p. 51),
- `ComparablePoint.java` (↪ p. 52),
- `HashablePoint.java` (↪ p. 54), and
- `files.lst` (↪ p. 55)

belong to the `AspectJ` contribution (`ajc version 1.0.1`), located in path `examples\introduction`.[4]

The explanation that follows is also created form the AspectJ contribution, located in path `doc\progguide`.

Crosscutting is relative to a particular decomposition. In our case it is a simple object-oriented decomposition. Thus the example based on a normal object-oriented class `Point` (↪ p. 49). The class `Point` defines geometric points whose interface includes polar and rectangular coordinates, plus some simple operations to relocate points. It has attributes for both its polar and rectangular coordinates, plus flags to indicate which currently reflect the position of the point. Some operations cause the polar coordinates to be updated from the rectangular, and some have the opposite effect. This implementation, which is in intended to give the minimum number of conversions between coordinate systems, has the property that not all the attributes stored in a `Point` object are necessary to give a canonical representation such as might be used for storing, comparing, cloning or making hash codes from points. Thus the aspects, though simple, are not totally trivial.

Like advice, pieces of introduction are members of an aspect. They define new members that act as if they were defined on another class. Unlike advice, introduction affects not only the behavior of the application, but also the structural relationship between an application's classes. This is crucial: Affecting the class structure of an application at makes these modifications available to other components of the application. Introduction modifies a class by adding or changing.

Class `Point`

```
/*
 *  Copyright (c) Xerox Corporation 1998-2001.
 *  Modified Bonin 29-Jan-2002
*/

package introduction;

public class Point
{

   protected double x = 0;
   protected double y = 0;
   protected double theta = 0;
   protected double rho = 0;

   protected boolean polar = true;
   protected boolean rectangular = true;

   public double getX(){
      makeRectangular();
```

This software is made available AS IS, and Xerox Corporation makes no warranty about the software, its performance or its conformity to any specification.

```
        return x;
    }

    public double getY(){
        makeRectangular();
        return y;
    }

    public double getTheta(){
        makePolar();
        return theta;
    }

    public double getRho(){
        makePolar();
        return rho;
    }

    public void setRectangular
        (double x, double y){
        this.x = x;
        this.y = y;
        rectangular = true;
        polar = false;
    }

    public void setPolar
        (double theta, double rho){
        this.theta = theta;
        this.rho = rho;
        rectangular = false;
        polar = true;
    }

    public void rotate(double angle){
        setPolar(theta + angle, rho);
    }

    public void offset
        (double deltaX, double deltaY){
        setRectangular(x + deltaX, y + deltaY);
    }

    protected void makePolar(){
        if (!polar){
  theta = Math.atan2(y,x);
```

```
 rho = y / Math.sin(theta);
 polar = true;
      }
   }

   protected void makeRectangular(){
      if (!rectangular) {
         x = rho * Math.sin(theta);
         y = rho * Math.cos(theta);
         rectangular = true;
      }
   }

   public String toString(){
      return "(" + getX() + ", " + getY() + ")["
         + getTheta() + " : " + getRho() + "]";
   }

   public static void main(String[] args){
      Point p1 = new Point();
      System.out.println("p1 =" + p1);
      p1.setRectangular(5,2);
      System.out.println("p1 =" + p1);
      p1.setPolar( Math.PI / 4.0 , 1.0);
      System.out.println("p1 =" + p1);
      p1.setPolar( 0.3805 , 5.385);
      System.out.println("p1 =" + p1);
   }
}
```

The aspect `CloneablePoint` demonstrates the introduction of an interface (`Cloneable`) and a method (`clone()`) into the class `Point`. In Java, all objects inherit the method `clone()` from the class `Object`, but an object is not cloneable unless its class also implements the interface `Cloneable`. In addition, classes frequently have requirements over and above the simple ”bit-by-bit” copying that `Object.clone` does. In our case, we want to update a `Point`'s coordinate systems before we actually clone the `Point`. So we have to override `Object.clone` with a new method that does what we want.

The `CloneablePoint` aspect uses the `declare parents` form to introduce the interface `Cloneable` into the class `Point`. It then defines a method, `Point.clone`, which overrides the method `clone()` that was inherited from `Object`. `Point.clone` updates the `Point`'s coordinate systems before invoking its superclass' `clone()` method.

Note that since aspects define types just as classes define types, we can define a `main()` method that is invocable from the command line to use as a test method.

Aspect `CloneablePoint`

```
/*
 * Copyright (c) Xerox Corporation 1998-2001.
*/
package introduction;

public aspect CloneablePoint
{

   declare parents: Point implements Cloneable;

   public Object Point.clone()
      throws CloneNotSupportedException {
      // we choose to bring all fields
      //up to date before cloning.
      makeRectangular();
      makePolar();
      return super.clone();
   }

   public static void main(String[] args){
      Point p1 = new Point();
      Point p2 = null;

      p1.setPolar(Math.PI, 1.0);
      try {
         p2 = (Point)p1.clone();
      } catch (CloneNotSupportedException e) {}
      System.out.println("p1 =" + p1 );
      System.out.println("p2 =" + p2 );

      p1.rotate(Math.PI / -2);
      System.out.println("p1 =" + p1 );
      System.out.println("p2 =" + p2 );
   }
}
```

Making `Points` comparable the aspect `ComparablePoint` introduces another interface and method into the class `Point`. It defines the single method `compareTo()` which can be used to define a natural ordering relation among the objects of a class that implements it. The aspect `ComparablePoint` introduces implements `Comparable` into `Point` along with a `compareTo()` method that can be used to compare `Points`. A `Point p1` is said to be less than another `Point p2` if `p1` is closer to the origin.

Aspect `ComparablePoint`

```
/*
 * Copyright (c) Xerox Corporation 1998-2001.
```

```
*/

package introduction;

public aspect ComparablePoint
{

   declare parents: Point implements Comparable;

   public int Point.compareTo(Object o) {
      return (int) (this.getRho() - ((Point)o).getRho());
   }

   public static void main(String[] args){
      Point p1 = new Point();
      Point p2 = new Point();

      System.out.println
           ("p1 =?= p2 :" + p1.compareTo(p2));

      p1.setRectangular(2,5);
      p2.setRectangular(2,5);
      System.out.println
           ("p1 =?= p2 :" + p1.compareTo(p2));

      p2.setRectangular(3,6);
      System.out.println
           ("p1 =?= p2 :" + p1.compareTo(p2));

      p1.setPolar(Math.PI, 4);
      p2.setPolar(Math.PI, 4);
      System.out.println
           ("p1 =?= p2 :" + p1.compareTo(p2));

      p1.rotate(Math.PI / 4.0);
      System.out.println
           ("p1 =?= p2 :" + p1.compareTo(p2));

      p1.offset(1,1);
      System.out.println
           ("p1 =?= p2 :" + p1.compareTo(p2));
   }
}
```

The aspect `HashablePoint` overrides two previously defined methods to give to `Point` the hashing behavior we want. The method `Object.hashCode()` returns a unique integer, suitable for use as a hash table key. Different implementations are allowed to return different integers, but must return distinct integers for distinct objects, and the same integer for objects that test equals. But since the default implementation of `Object.equal()` returns `true` only when two objects are identical, we need to redefine both `equals` and `hashCode` to work correctly with objects of type `Point`. For example, we want two `Point` objects to test equals when they have the same `x` and `y` values, or the same `rho` and `theta` values, not just when they refer to the same object. We do this by overriding the methods `equals` and `hashCode` in the class `Point`.

The class `HashablePoint` introduces the methods `hashCode` and `equals` into the class `Point`. These methods use `Point`'s rectangular coordinates to generate a hash code and to test for equality. The `x` and `y` coordinates are obtained using the appropriate get methods, which ensure the rectangular coordinates are up-to-date before returning their values.

Aspect `HashablePoint`

```
/*
 * Copyright (c) Xerox Corporation 1998-2001.
*/

package introduction;

import java.util.Hashtable;

public aspect HashablePoint
{

   public int Point.hashCode() {
      return (int)
      (getX() + getY() % Integer.MAX_VALUE);
   }

   public boolean Point.equals(Object o) {
      if (o == this) { return true; }
      if (!(o instanceof Point)) { return false; }
      Point other = (Point)o;
      return (getX() == other.getX()) &&
             (getY() == other.getY());
   }

   public static void main(String[] args) {
      Hashtable h = new Hashtable();
      Point p1 = new Point();

      p1.setRectangular(10, 10);
      Point p2 = new Point();
```

```
      p2.setRectangular(10, 10);

      System.out.println("p1 = " + p1);
      System.out.println("p2 = " + p2);
      System.out.println
          ("p1.hashCode() = " + p1.hashCode());
      System.out.println
          ("p2.hashCode() = " + p2.hashCode());

      h.put(p1, "P1");
      System.out.println
          ("Got: " + h.get(p2));
   }
}
```

Argument list `files.lst`

```
Point.java
CloneablePoint.java
ComparablePoint.java
HashablePoint.java
```

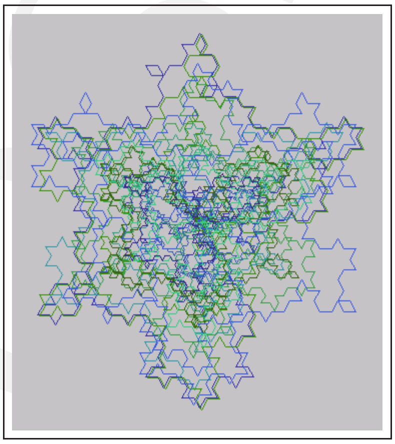
Protocol `Point.log`

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.1 (built 18.12.2001 11:11 PST) running on java 1.3.1

D:\bonin\aosd\code>ajc -argfile introduction/files.lst

D:\bonin\aosd\code>java introduction.Point
p1 =(0.0, 0.0)[0.0 : 0.0]
p1 =(5.0, 2.0)[0.3805063771123649 : 5.385164807134504]
p1 =(0.7071067811865475, 0.7071067811865476)[0.7853981633974483 : 1.0]
p1 =(1.9999069075401812, 4.999859734149856)[0.3805 : 5.385]

D:\bonin\aosd\code>java introduction.CloneablePoint
p1 =(1.2246467991473532E-16, -1.0)[3.141592653589793 : 1.0]
p2 =(1.2246467991473532E-16, -1.0)[3.141592653589793 : 1.0]
p1 =(1.0, 6.123233995736766E-17)[1.5707963267948966 : 1.0]
p2 =(1.2246467991473532E-16, -1.0)[3.141592653589793 : 1.0]

D:\bonin\aosd\code>java introduction.ComparablePoint
p1 =?= p2 :0
p1 =?= p2 :0
p1 =?= p2 :-1
```

```
p1 =?= p2 :0
p1 =?= p2 :0
p1 =?= p2 :2

D:\bonin\aosd\code>java introduction.HashablePoint
p1 = (10.0, 10.0)[0.7853981633974483 : 14.142135623730951]
p2 = (10.0, 10.0)[0.7853981633974483 : 14.142135623730951]
p1.hashCode() = 20
p2.hashCode() = 20
Got: P1

D:\bonin\aosd\code>
```

Chapter 4

Constructs of AspectJ



4.1 Join Points and Pointcuts

The AspectJ language uses a term called a *join point*. A join point is a well-defined "point" in the execution of the program, for instance a method call or an access to an attribute of a specific class.[1]. A *pointcut*, another AspectJ term, acts as a grouping for specific joint points. The name() part of the pointcut will be used as a reference. The aJoinPoint part of the pointcut is the signature of the join point where something should take place. The *designator*, another AspectJ term, indicates when a join point should be

[1] Note, most of the text and most of the examples in this chapter are taken from the AspectJ — exactly: `ajc version 1.0.1 (built 18.12.2001 11:11 PST) running on java 1.3.1` — contribution (↪ \aspectj1.0\doc\progguide)

associated with a pointcut. With the keyword `aspect` we create a single module (like a Jave class) to encapsulate the code of pointcuts and advices.

`pointcut` name(parameter$*$) : designator(aJoinPoint);

<u>with</u>:
$* \equiv$ no, one or many (parameter(s))

4.1.1 Designators

Here are examples of designators of:

- when a particular method body executes
  `execution(void Point.setX(int))`
- when a method is called
  `call(void Point.setX(int))`
- when an exception handler executes
  `handler(ArrayOutOfBoundsException)`
- when the object is currently executing (i. e. `this`) is of type `SomeType`
  `this(SomeType)`
- when the target object is of type `SomeType`
  `target(SomeType)`
- when the executing code belongs to class `MyClass`
  `within(MyClass)`

Designators compose through the operations:

or: "`||`",
and: "`&&`"
not: "`!`".

It is possible to use wildcards. For example `execution(* *(..))` means all the executions of methods with any return and parameter types. A notation `call(* set(..))` means method calls of set methods with any return and parameter types. In case of overloading there may be more than one; this designator picks out all of them.

4.1.2 Pointcut Parameters

Consider, for example, the following pointcut:

```
pointcut setter(): target(Point) &&
    (call(void setX(int)) || call(void setY(int)));
```

The right-hand side of the pointcut picks out the calls to `setX(int)` or `setY(int)` methods where the target is any object of type `Point`. On the left-hand side, the pointcut is given the name "setters" and no parameters. In the next version of the same pointcut:

```
pointcut setter(Point p): target(p) &&
    (call(void setX(int)) || call(void setY(int)));
```

This pointcut has a parameter `p` of type `Point`. When the events described on the right-hand side happen, a `Point` object is bound to `p`. It is the `Point` object that receives the calls.

The next example illustrates the mechanism for defining pointcut parameters:

```
pointcut testEquality(Point p):
    target(Point) &&
    args(p) &&
    call(boolean equals(Point));
```

When the events described on the right-hand side happen, a `Point` object, named by the parameter `p`, is available. But in this case, we find that the point in the parameters is not the `Point` object that receives the call; it's the argument of `equals(p)` on some other Point object. If we wanted access to both objects, then the pointcut definition should be:

```
pointcut testEquality(Point p1, Point p2):
    target(p1) &&
    args(p2) &&
    call(boolean equals(Point));
```

4.2 Advice

The action that should take place when a pointcut ist triggered has a specific term in AspectJ called *advice*.

An advice defines pieces of aspect implementation that execute at well-defined points in the execution of the program. Those points can be given either by named or by anonymous pointcuts.

adviceType(parameter$*$) : pointcut{
//code to execute before, after or around (instead)
}

with:

$*$	$\equiv$	no, one or many (parameter(s))
adviceType	$\equiv$	`before`
		`around`
		`after`

Here is an example of an advice on a named pointcut:

Named pointcut

```
pointcut setter(Point p1, int newval):
    target(p1) && args(newval)
        (call(void setX(int) ||
```

```
         call(void setY(int)));

  before(Point p1, int newval): setter(p1, newval)
      {
      System.out.println(
          "About to set something in " +
           p1 +
          " to the new value " + newval);
  }
```

before()

And here is exactly the same example, but using an anonymous pointcut:

```
before(Point p1, int newval):
    target(p1) && args(newval)
       (call(void setX(int)) ||
        call(void setY(int)))
    {
      System.out.println(
          "About to set something in " +
           p1 +
          " to the new value " + newval);
    }
```

Here are examples of the different advice:

```
before(Point p, int x): target(p) && args(x) &&
    call(void setX(int))
    {
      if (!p.assertX(x)) return;
    }
```

after()

This before advice runs just before the execution of the actions associated with the events in the (anonymous) pointcut.

```
after(Point p, int x): target(p) && args(x) &&
    call(void setX(int))
    {
        if (!p.assertX(x))
           throw new PostConditionViolation();
    }
```

returning()

This after advice runs just after each join point picked out by the (anonymous) pointcut, regardless of whether it returns normally or throws an exception.

```
after(Point p) returning(int x): target(p) &&
    call(int getX())
    {
        System.out.println(
```

```
            "Returning int value " + x +
            " for p = " + p);
    }
```

This after returning advice runs just after each join point picked out by the (anonymous) pointcut, but only if it returns normally. The return value can be accessed, and is named `x` here. After the advice runs, the return value is returned.

`throwing()`

```
after() throwing(Exception e): target(Point) &&
    call(void setX(int))
    {
        System.out.println(e);
    }
```

This after throwing advice runs just after each join point picked out by the (anonymous) pointcut, but only when it throws an exception of type `Exception`. Here the exception value can be accessed with the name `e`. The advice re-raises the exception after it has been done.

`around()`

```
void around(Point p, int x): target(p)
    && args(x) &&
    call(void setX(int))
    {
        if (p.assertX(x)) proceed();
            p.releaseResources();
    }
```

This around advice traps the execution of the join point; it runs *instead* of the join point. The original action associated with the join point can be invoked through the special `proceed()` call.

4.3 Introduction

Introduction declarations add whole new elements in the given types, and so change the type hierarchy. Here are examples of introduction declarations:

Field

- `private boolean Server.disabled = false;`
  This statement introduces a field named `disabled` in `Server` and initializes it to `false`. Because it is declared `private`, only code defined in the aspect can access the field.

Method

- `public int Point.getX() { return x; }`
  This statement introduces a method named `getX()` in `Point`; the method returns an `int`, it has no arguments, and its body is return `x`. Because it is declared `public`, any code can call it.

Constructor

- `public Point.new(int x, int y) { this.x = x; this.y = y; }`
  This statement introduces a constructor in Point; the constructor has two arguments of type `int`, and its body is `this.x = x; this.y = y;`

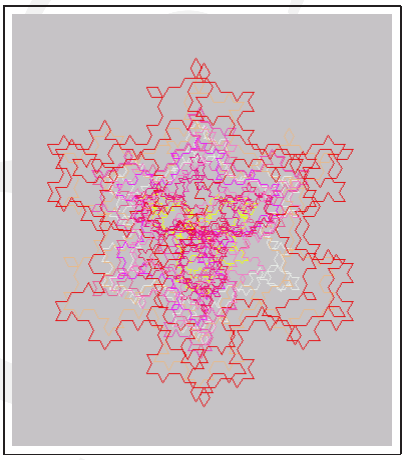
- `public int Point.x = 0;`
  This statement introduces a field named `x` of type `int` in `Point`; the field is initialized to `0`.

`declare parents:`

- `declare parents: Point implements Comparable;`
  This declares that the `Point` class now implements the `Comparable` interface. Of course, this will be an error unless `Point` defines the methods of `Comparable`.
- `declare parents: Point extends GeometricObject;`
  This declares that the `Point` class now extends the `GeometricObject` class.

Chapter 5

Guidelines



The central problem of aspect technologies is how to understand a number of software parts as separate artifacts and then integrate some of them into a coherent system ([Pace / Campo, 2001], p. 73).

For instance, when capturing a user-level feature as an aspect, it is established practice to express the feature in its own object structure. Then use an aspect to inject that feature into the base code.

5.1 Design Patterns

> "Each pattern describe a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use the solution a million times over, without ever doing it the same way twice." (↪ [Alexander+, 1977] quoted from [Gamma+, 1994], p. 2)

	Purpose		
	Creational	**Structural**†	**Behavioral**‡
Scope **Class**	Factory Method	Adapter (class)	Interpreter
Scope **Object**	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite ↪ p. 66 Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Legende:
Source ↪ [Gamma+, 1994], p. 10
† ≡ Structural patterns deal with the composition of classes or objects.
‡ ≡ Behavioral patterns characterize the ways in which classes or objects interact and distribute responsibility.

Table 5.1: 23 Design Patterns

A pattern has four essential elements:

- *Name*
  The pattern name provides to think and communicate at a higher level of abstraction (↪ Table 5.1, p. 64).
- *Problem*
  The problem describes whe to apply the pattern.
- *Solution*
  The solution describes the elements that make up the design, their relationships, responsibilities, and collaborations. It doesn't describe a particular concrete design or implementation.
- *Consequences*
  The consequences are the results and trade-offs of applying the pattern.

5.1.1 Inheritance: Class versus Interface

"Program to an interface,
not an implementation."

The Gang-of-Four design patterns ↪ [Gamma+, 1994], p. 18

The class defines the objects's internal state and the implementation of its operations. In contrast, an object's type only refers to its interface — the set of messages to which it can respond. An object can have many types, and objects of different classes can have the same type.

It's important to understand the difference between class inheritance and interface inheritance ($\equiv$ subtyping). Class inheritance defines an object's implementation in terms of another object's implementation. In short, it's a mechanism for code and representation sharing. In contrast, interface inheritance describes when an object can be used in place of another.

Class inheritance is basically just a mechanism for extending an application's functionality by reusing functionality in parent classes. It lets you define a new kind of object rapidly in terms of an old one. Inheritance's ability to define families of objects with *identical* interfaces (usually by inheriting from an abstract class) is also important. There are two benefits to manipulating objects solely in terms of the interfaces:

1. Clients remain unaware of the classes of objects they use, as long as the objects adhere to the interface that clients expect.
2. Clients remain unaware of the classes that implement these objects. Clients only know about the interface(s).

5.1.2 Inheritance versus Composition

"Favor object composition
over class inheritance."
The Gang-of-Four design patterns ↪ [Gamma+, 1994], p. 20

The common object-oriented techniques for reusing functionality are class inheritance ($\approx$ *white-box reuse*) and object composition ($\approx$ *black-box reuse*). The term "white-box" refers to visibility: With class inheritance, the internals of parent classes are often visible to subclasses. Object composition requires that the object being composed have well-defined interfaces. This style of reuse is called "black-box reuse", because no internal details of objects are visible. Objects appear only as "black boxes". *white-box* *black-box*

Class inheritance is defined statically at compile-time and makes it easier to modify the implementation being reused. When a subclass overrides some but not all operations, it can affect the operations it inherits as well, assuming they call the overridden operations. A disadvantage is, parent classes often define at least part of their subclasses' physical representation. Because inheritance exposes a subclass to details of its parent's implementation, it's often said that "inheritance breaks encapsulation" (Alan Snyder[1] quoted from [Gamma+, 1994], p. 19)

Object composition is defined dynamically at run-time throuh objects acquiring references to other objects. Any object can be replaced at run-time by another as long

[1] Alan Snyder; Encapsulation and inheritance in object-oriented languages; in: *Object-Oriented Programming Systems, Languages, and Applications Conference Proceedings*, p. 38–45, Portland, OR, (ACM Press), November 1986

as it has the same type. Ideally, you shouldn't have to create new components to achieve reuse. You should be able to get all functionality you need just by assembling existing components through object composition. But this is rarely the case, because the set of available components is never quite rich enough in practice. Reuse by inheritance makes it easier to make new components that can be composed with old ones. Inheritance and object composition thus work together.

Use the pattern *composition* when:

- you want to represent part-whole hierarchies of objects.
- you want clients to be able to ignore the difference between compositions of objects and individual objects. Clients will treat all objects in the composite structure *uniformly*.

A simple example coded in Java[2] creates a structure as follows (↪ `Client` p. 66)

- Composite c1 has four children: l1, l2, c2, and l3.
- Composite c2 has three children: l4, l5, and l6.
- Composite c3 has two children: l1, and l2.
- Composite c3 is added to composite c1 and then from c1.

Notice, in class `Client` there is no different between adding a leaf or a composite.

Argument list `files.lst`

```
Client.java
Component.java
Composite.java
Leaf.java
```

Class `Client`

It manipulates objects in the composition through the Component interface.

```
/**
 *  Pattern "Composite" Idea "Gang-of-Four design patterns"
 *  --- Java code structure Jan Hannemann / Gregor Kiczales
 *
 *@author     Bonin
 *@version    1.0
 *@see        Component
 *@see        Composite
 *@see        Leaf
 */
```

[2]The Composite pattern ↪ [Gamma+, 1994] p. 163–173. The Java code structure is similar to Jan Hannemann / Gregor Kiczales ↪ `http://www.cs.ubc.ca/labs/spl/projects/aodps.html` (visited 12-Jun-2003)

```java
package de.fhnon.nemo.composite;

public class Client
{
   private static int indent = 0;


   private static void indent()
   {
      for (int i = 0; i < indent; i++)
      {
         System.out.print(" ");
      }
   }


   private static void printStructure(Component component)
   {
      indent();
      System.out.println("Printing: " + component);
      indent += 4;
      for (int i = 0; i < component.getChildCount(); i++)
      {
         printStructure(component.getChild(i));
      }
      indent -= 4;
   }


   /*
    *  This client creates a structure as follows:
    *  Composite c1 has four children: l1, l2, c2, and l3. Composite c2
    *  has three children: l4, l5, and l6.
    *  Composite c3 has two children: l1, and l2 and is
    *  removed from composite c1.
    */
   public static void main(String[] args)
   {
      Composite c1 = new Composite(1);
      Composite c2 = new Composite(2);
      Composite c3 = new Composite(3);

      Leaf l1 = new Leaf(1);
      Leaf l2 = new Leaf(2);
      Leaf l3 = new Leaf(3);
      Leaf l4 = new Leaf(4);
```

```
      Leaf l5 = new Leaf(5);
      Leaf l6 = new Leaf(6);

      c1.add(l1);
      c1.add(l2);
      c1.add(c2);
      c1.add(l3);
      c1.add(c3);

      c2.add(l4);
      c2.add(l5);
      c2.add(l6);

      c3.add(l1);
      c3.add(l2);
      c1.remove(c3);

      printStructure(c1);
   }
}
```

Interface `Component` It declares the interface for all objects in the composition. Declaring the child management operations here gives transparency, because all components can be treated uniformly. However, it costs safety, because clients may try to do meaningless thinks like add and remove objects from leaves. Defining child management operations in the Composite class is more safe, because any attempt to add or remove object from leaves will be caught at compile-time. But then we lose transparency, because leaves and composites have different interfaces. That is why, here `Component` declares the child management operations.

```
/**
 *  Pattern "Composite" Idea "Gang-of-Four design patterns"
 *  --- Java code structure Jan Hannemann / Gregor Kiczales
 *
 *@author     Bonin
 *@version    1.0
 *@see        Client
 *@see        Composite
 *@see        Leaf
 */
package de.fhnon.nemo.composite;

public interface Component
{
   public void add(Component component);
```

```
   public void remove(Component component);


   public Component getChild(int index);


   public int getChildCount();
}
```

Class `Composite`

```
/**
 *  Pattern "Composite" Idea "Gang-of-Four design patterns"
 *   --- Java code structure Jan Hannemann / Gregor Kiczales
 *
 *@author     Bonin
 *@version    1.0
 *@see        Client
 *@see        Component
 *@see        Leaf
 */
package de.fhnon.nemo.composite;

import java.util.LinkedList;

public class Composite implements Component
{

   protected LinkedList children = new LinkedList();

   protected int id = 0;


   public Composite(int id)
   {
      this.id = id;
   }


   public String toString()
   {
      return "Composite with id = " + id;
   }
```

```
   public void add(Component component)
   {
      this.children.add(component);
   }


   public void remove(Component component)
   {
      this.children.remove(component);
   }


   public Component getChild(int index)
   {

      return (Component) this.children.get(index);
   }


   public int getChildCount()
   {
      return this.children.size();
   }
}
```

Class `Leaf` It represents leaf objects in the composition. A leaf has no children.

```
/**
 *  Pattern "Composite" Idea "Gang-of-Four design patterns"
 *  --- Java code structure Jan Hannemann / Gregor Kiczales
 *
 *@author     Bonin
 *@version    1.0
 *@see        Client
 *@see        Component
 *@see        Composite
 */
package de.fhnon.nemo.composite;

public class Leaf implements Component
{

   protected int id = 0;
```

```
   public Leaf(int id)
   {
      this.id = id;
   }


   public String toString()
   {
      return "Leaf with id = " + id;
   }


   public void add(Component component)
   {
      // Interface Component requirement
   }


   public void remove(Component component)
   {
      // Interface Component requirement
   }


   public Component getChild(int index)
   {
      return null;
   }


   public int getChildCount()
   {
      return 0;
   }
}
```

Protocol `Client.log`

```
C:\bonin\aosd\code>ajc -version
AspectJ Compiler 1.1.0

C:\bonin\aosd\code>ajc -argfile de/fhnon/nemo/composite/files.lst

C:\bonin\aosd\code>java -version
java version "1.4.0_01"
```

```
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.0_01-b03)
Java HotSpot(TM) Client VM
  (build 1.4.0_01-b03, mixed mode)

C:\bonin\aosd\code>java de.fhnon.nemo.composite.Client
Printing: Composite with id = 1
    Printing: Leaf with id = 1
    Printing: Leaf with id = 2
    Printing: Composite with id = 2
        Printing: Leaf with id = 4
        Printing: Leaf with id = 5
        Printing: Leaf with id = 6
    Printing: Leaf with id = 3

C:\bonin\aosd\code>
```

5.2 Profiling, Logging, Conditions

The following example `Hound` (↪ p. 72) shows some profiling, logging and pre-conditions. The defined aspects have an advantage over standard profiling or logging tools because they can be programmed to ask very specific and complex questions. We ask, ”How many times is the method `setName()` called for an instance, and when the other method `getName()` is called we add some values to the output.”

In the programming style *Design by Contract* (↪ Section 2.3, p. 35), we code explicit pre- und post-conditions. A pre-condition tests that the caller of a method calls it properly. A post-condition tests that a method properly does the work it is supposed to do.

This example contains the aspects `PutXML` (↪ p. 75), `Count` (↪ p. 76) and and `SexCheck` (↪ p. 77) programmed to the class `Hound` (↪ p. 72). This class has a recursive definition; the mother and the father are also hounds. The first aspect wrapped XML-tags around some slot values, when the method `getName()` is called. The second aspect counts the changes of the slot `name` for an instance. Therfore it introduces the slot `number` in all instances of `Hound`. The third aspect checks up the value of the slot `sex` and changes it to `female`, if illegal.

Class `Hound`

```
/*
 * Created on 18.11.2003
 */
package de.fhnon.as.hound;

/**
 * @author bonin
```

```
 *
 */
public class Hound {

private final String ident;
private String name;
private String sex;
private String performance;
private Hound mother;
private Hound father;

public String getIdent() {
return ident;
}

public String getName() {
return name;
}

public String getSex() {
return sex;
}

public String getPreformance() {
return performance;
}

public Hound getMother() {
return mother;
}

public Hound getFather() {
return father;
}

public void setName(String name) {
this.name = name;
}

public void setSex(String sex) {
this.sex = sex;
}
```

```
public void setPerformance(String performance) {
this.performance = performance;
}

public void setMother(Hound mother) {
this.mother = mother;
}

public void setFather(Hound father) {
this.father = father;
}

public Hound(String ident) {
this.ident = ident;
}

public static void main(String[] args) {
System.out.println(
"<?xml version=\"1.0\" encoding=\"utf-8\" ?>"
+ "\n"
+ "<!-- Hound application started -->"
+ "\n"
+ "<hound>");

Hound elsa = new Hound("99-032");
Hound wulf = new Hound("97-141");
Hound ukelei = new Hound("97-070");

ukelei.setName("Ukelei aus der Meute");
wulf.setSex("malus");

elsa.setMother(ukelei);
elsa.setFather(wulf);
elsa.getFather().setName("Wulf von ...");
elsa.getFather().setName("Wulf von Wallesau");

elsa.getFather().getName();

System.out.println(
"<!-- Hound application finished -->" + "\n" + "</hound>");
```

```
}
}
```

Aspect `PutXML`

```
/*
 * Created on 18.11.2003
 *
 */
package de.fhnon.as.hound;

/**
 * @author bonin
 *
 */
public aspect PutXML {

pointcut applyName(Hound h) : target(h) && call(String getName());

before(Hound h) : applyName(h) {
System.out.println(
"<!-- In Hound application ... start before calling getName() -->"
+ "\n"
+ "<!-- checking in aspect PutXML -->"
+ "\n"
+ "<ident>"
+ h.getIdent()
+ "</ident>"
+ "\n"
+ "<sex>"
+ h.getSex()
+ "</sex>"
+ "\n"
+ "<!-- In Hound application ... end before callig getName() -->");
}

after(Hound h) returning(String x) : applyName(h) {
System.out.println(
"<!-- In Hound application ... start after calling getName() -->"
+ "\n"
+ "<!-- checking in aspect PutXML -->"
+ "\n"
+ "<name>"
```

```
+ x
+ "</name>"
+ "\n"
+ "<!-- In Hound application ... end after callig getName() -->");
}

}
```

Aspect Count

```
/*
 * Created on 18.11.2003
 *
 */
package de.fhnon.as.hound;

/**
 * @author bonin
 *
 */
public aspect Count {

private int Hound.number = 0;

public int Hound.getNumber() {
return number;
}

public void Hound.setNumber(int number) {
this.number = number;
}

pointcut changeName(Hound h) : target(h) && call(void setName(String));

pointcut applyName(Hound h) : target(h) && call(String getName());

before(Hound h) : changeName(h) {
h.setNumber(h.getNumber() + 1);
System.out.println(
"<!-- In Hound application ... before calling setName() -->"
+ "\n"
+ "<!-- checking in aspect Count -->");
}
```

```
after(Hound h) : applyName(h) {
System.out.println(
"<!-- In Hound application ... after calling getName() -->"
+ "\n"
+ "<!-- checking in aspect Count -->"
+ "\n"
+ "<update>"
+ h.getNumber()
+ "</update>"
+ "\n"
+ "<!-- In Hound application ... after finishing getName() -->");
}

}
```

Aspect `SexCheck`

```
/*
 * Created on 18.11.2003
 *
  */
package de.fhnon.as.hound;

/**
 * @author bonin
 *
 */
public aspect SexCheck {

private boolean Hound.checkSex(String x) {
return (x == "male" || x == "female");

}

after(Hound h, String x) : target(h)
&& args(x)
&& call(void setSex(String)) {
if (!h.checkSex(x)) {
System.out.println(
"<!-- In Hound application ... start after calling setSex() -->"
+ "\n"
+ "<!-- appling checkSex() -->"
```

```
+ "\n"
+ "<error><sex>"
+ "Illegal sex; assume female!"
+ "</sex></error>");
h.setSex("female");
return;
}
}
}
```

Argument list `files.lst`

```
Hound.java
PutXML.java
Count.java
SexCheck.java
```

Protocol `Hound.log`

```
D:\bonin\aosd\code>java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM (build 1.4.2-b28, mixed mode)

D:\bonin\aosd\code>javac de/fhnon/as/hound/Hound.java

D:\bonin\aosd\code>java de.fhnon.as.hound.Hound
<?xml version="1.0" encoding="utf-8" ?>
<!-- Hound application started -->
<hound>
<!-- Hound application finished -->
</hound>

D:\bonin\aosd\code>cd de/fhnon/as/hound

D:\bonin\aosd\code\de\fhnon\as\hound>dir Hound.class
19.11.2003  10:59             1.804 Hound.class

D:\bonin\aosd\code\de\fhnon\as\hound>cd ..

D:\bonin\aosd\code\de\fhnon\as>cd ..

D:\bonin\aosd\code\de\fhnon>cd ..
```

```
D:\bonin\aosd\code\de>cd ..

D:\bonin\aosd\code>ajc -version
AspectJ Compiler 1.1.0

D:\bonin\aosd\code>ajc -argfile de/fhnon/as/hound/files.lst

D:\bonin\aosd\code>java de.fhnon.as.hound.Hound
<?xml version="1.0" encoding="utf-8" ?>
<!-- Hound application started -->
<hound>
<!-- In Hound application ... before calling setName() -->
<!-- checking in aspect Count -->
<!-- In Hound application ... start after calling setSex() -->
<!-- appling checkSex() -->
<error><sex>Illegal sex; assume female!</sex></error>
<!-- In Hound application ... before calling setName() -->
<!-- checking in aspect Count -->
<!-- In Hound application ... before calling setName() -->
<!-- checking in aspect Count -->
<!-- In Hound application ... start before calling getName() -->
<!-- checking in aspect PutXML -->
<ident>97-141</ident>
<sex>female</sex>
<!-- In Hound application ... end before callig getName() -->
<!-- In Hound application ... after calling getName() -->
<!-- checking in aspect Count -->
<update>2</update>
<!-- In Hound application ... after finishing getName() -->
<!-- In Hound application ... start after calling getName() -->
<!-- checking in aspect PutXML -->
<name>Wulf von Wallesau</name>
<!-- In Hound application ... end after callig getName() -->
<!-- Hound application finished -->
</hound>

D:\bonin\aosd\code>java de.fhnon.as.hound.Hound
   >D:\bonin\aosd\log\Hound.xml

D:\bonin\aosd\code\de\fhnon\as\hound>dir
         4.682 Count.class
```

```
          999 Count.java
           52 files.lst
        3.720 Hound.class
        1.682 Hound.java
        2.241 Hound.log
        2.790 PutXML.class
          940 PutXML.java
        2.787 SexCheck.class
          631 SexCheck.java

D:\bonin\aosd\code\de\fhnon\as\hound>
```

Output in the file `Hound.xml` The figure 5.1 p. 81 shows the output file `Hound.xml` in the browser *Microsoft Internet Explorer 6.0.2600*.

Attention: Old `AspecJ` version and old `Hound` version!!

pre-process

To see how the process of compiling works with `ajc` we look at the modified Java files in source code form. For that we use the option `-preprocess`, save the result of this preprocess in a working directory and take the file `files.lst` (↪ p. 78) containing the argument list.

Before the preprocess of compiling we have coded in our file `Hound.java` (↪ p. 72) the method getting the name as follows:

```
public String getName() { return name; }
```

After the preprocess of compiling we get for this method the following code (↪ p. 83):

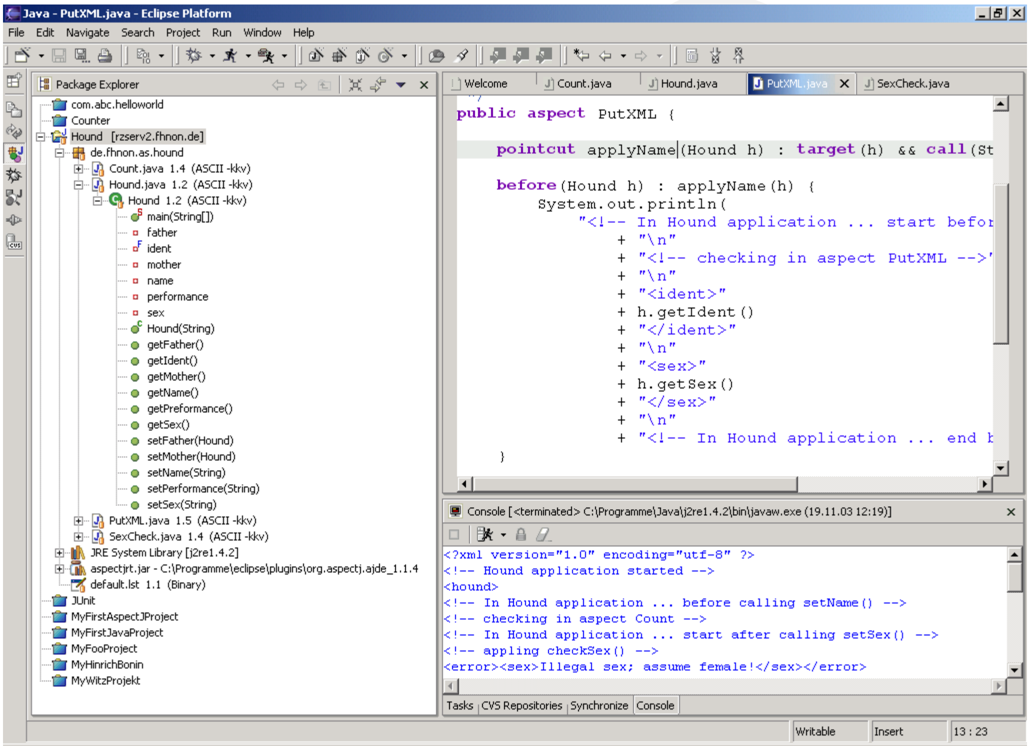
```
  public String getName() {
    try {
      PutXML.aspectInstance.before0$ajc(this);
      {
        String _return = this.getName$ajcPostCall();
        PutXML.aspectInstance.afterReturning0$ajc(this,
                                                  _return);
        return _return;
      }
    } finally {
      Count.aspectInstance.after0$ajc(this);
    }
  }
```

The aspect `PutXML` (↪ p. 75) ist transformed to a class `PutXML` (↪ p. 87). This class has the `static` slot `aspectInstance`. The value of this slot is an instance of the class itself. The instance method `before0$ajc()` is generated from the part `before()` of the aspect as followed:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<!-- Hound application started -->
- <hound>
    <!-- In Hound application ... before calling setName() -->
    <!-- checking in aspect Count -->
    <!-- In Hound application ... start after calling setSex() -->
    <!-- appling checkSex() -->
  - <error>
      <sex>Illegal sex; assume female!</sex>
    </error>
    <!-- In Hound application ... before calling setName() -->
    <!-- checking in aspect Count -->
    <!-- In Hound application ... before calling setName() -->
    <!-- checking in aspect Count -->
    <!-- In Hound application ... start before calling getName() -->
    <!-- checking in aspect PutXML -->
    <ident>97-141</ident>
    <sex>female</sex>
    <!-- In Hound application ... end before callig getName() -->
    <!-- In Hound application ... after calling getName() -->
    <!-- checking in aspect Count -->
    <update>2</update>
    <!-- In Hound application ... after finishing getName() -->
    <!-- In Hound application ... start after calling getName() -->
    <!-- checking in aspect PutXML -->
    <name>Wulf von Wallesau</name>
    <!-- In Hound application ... end after callig getName() -->
    <!-- Hound application finished -->
  </hound>
```

Legende:
This figure shows the output file `Hound.xml` in the browser *Microsoft Internet Explorer 6.0.2600*. The file `Hound.log` (↪ section 5.2 p. 78) documents the production.

Figure 5.1: Project `Hound` — output



Legende:

AspectJ in IBM's Integrated Development Environment *Eclipse* (↪ section A.3 p. 271)

Figure 5.2: Project `Hound` — IDE *Eclipse*

```
  public final void before0$ajc(Hound h)
  {
    System.out.println("<ident>" +
        h.getIdent() + "</ident>" +
        "<sex>" + h.getSex() + "</sex>");
  }
```

The preprocess includes new methods in the class `Hound`, one is the method `get-Name$ajcPostCall()` only returning `this.name`. The method `afterReturning0-$ajc()` in class `PutXML` is generated from the part `after()` of the aspect as followed:

```
  public final void afterReturning0$ajc(
      Hound h, String x)
  {
      System.out.println("<name>" + x + "</name>");
  }
```

The result of this `ajc` weaving is:

- Aspects are transformed in normal Java classes with methods normally binding instances of the ”original” classes to their parameters.
- Involved ”original” classes are enlarged with new methods.
- Methods of the involved ”original” classes are modified.

Protocol to generate regular Java code `Hound.log`

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.1 (built 18.12.2001 11:11 PST)
  running on java 1.3.1

D:\bonin\aosd\code>ajc -preprocess
  -workingdir D:\bonin\aosd\code\hound\preprocess
  -argfile hound/files.lst
D:\bonin\aosd\code>
```

Generated class `Hound` (↪ p. 72)

```
/*   Generated by AspectJ version 1.0.1 */
package hound;
public class Hound {
  private String ident;
  private String name;
  private String sex;
  private String performance;
  private Hound mother;
  private Hound father;
```

```
public String getIdent() {
  return this.ident;
}

public String getName() {
  try {
    PutXML.aspectInstance.before0$ajc(this);
    {
      String _return = this.getName$ajcPostCall();
      PutXML.aspectInstance.afterReturning0$ajc(this,
                                              _return);
      return _return;
    }
  } finally {
    Count.aspectInstance.after0$ajc(this);
  }
}

public String getSex() {
  return this.sex;
}

public String getPreformance() {
  return this.performance;
}

public Hound getMother() {
  return this.mother;
}

public Hound getFather() {
  return this.father;
}

public void setName(String name) {
  Count.aspectInstance.before0$ajc(this);
  this.setName$ajcPostCall(name);
}

public void setSex(String sex) {
  try {
    this.setSex$ajcPostCall(sex);
  } finally {
    SexCheck.aspectInstance.after0$ajc(this, sex);
  }
}
```

```
public void setPerformance(String performance) {
  this.performance = performance;
}

public void setMother(Hound mother) {
  this.mother = mother;
}

public void setFather(Hound father) {
  this.father = father;
}

public Hound(String ident) {
  super();
  {
    this.number_hound_Count = 0;
  }
  this.ident = ident;
}
public static void main(String[] args) {
  Hound elsa = new Hound("99-032");
  Hound wulf = new Hound("97-141");
  Hound ukelei = new Hound("97-070");
  Hound.setName$method_call(ukelei,
    "Ukelei aus der Meute");
  Hound.setSex$method_call(wulf, "malus");
  elsa.setMother(ukelei);
  elsa.setFather(wulf);
  Hound.setName$method_call0(elsa.getFather(),
    "Wulf von ...");
  Hound.setName$method_call1(elsa.getFather(),
    "Wulf von Wallesau");
  Hound.getName$method_call(elsa.getFather());
  System.out.println("<!-- Hound finished -->");
}

private int number_hound_Count;
public int getNumber() {
  return this.number_hound_Count;
}

public void setNumber(int number) {
  this.number_hound_Count = number;
}
```

```
  public boolean checkSex_hound_SexCheck(String x) {
    return (x == "male" || x == "female");
  }

  public void setName$ajcPostCall(String name) {
    this.name = name;
  }

  public void setSex$ajcPostCall(String sex) {
    this.sex = sex;
  }

  public String getName$ajcPostCall() {
    return this.name;
  }

  public final boolean
    checkSex_hound_SexCheck$ajc$backdoor(String x) {
    return this.checkSex_hound_SexCheck(x);
  }

  private static void
    setName$method_call(Hound target, final String name) {
    target.setName(name);
  }

  private static void
    setSex$method_call(Hound target, final String sex) {
    target.setSex(sex);
  }

  private static void
    setName$method_call0(Hound target, final String name) {
    target.setName(name);
  }

  private static void
    setName$method_call1(Hound target, final String name) {
    target.setName(name);
  }

  private static String getName$method_call(Hound target) {
    return target.getName();
  }

}
```

Generated class `PutXML` (↪ p. 75)

```
/*   Generated by AspectJ version 1.0.1 */
package hound;
class PutXML {
  public final void before0$ajc(Hound h) {
    System.out.println("<ident>" + h.getIdent() + "</ident>" +
      "<sex>" + h.getSex() + "</sex>");
  }

  public final void afterReturning0$ajc(Hound h, String x) {
    System.out.println("<name>" + x + "</name>");
  }

  PutXML() {
    super();
  }
  public static PutXML aspectInstance;
  public static PutXML aspectOf() {
    return PutXML.aspectInstance;
  }

  public static boolean hasAspect() {
    return PutXML.aspectInstance != null;
  }

  static {
    PutXML.aspectInstance = new PutXML();
  }

}
```

Generated class `Count` (↪ p. 76)

```
/*   Generated by AspectJ version 1.0.1 */
package hound;
class Count {
  /* IntroducedDec(dec: FieldDec(id: number)) */

  /* IntroducedDec(dec: MethodDec(id: getNumber)) */

  /* IntroducedDec(dec: MethodDec(id: setNumber)) */

  public final void before0$ajc(Hound h) {
    h.setNumber(h.getNumber() + 1);
  }
```

```
  public final void after0$ajc(Hound h) {
    System.out.println("<update>" + h.getNumber() +
      "</update>");
  }

  Count() {
    super();
  }
  public static Count aspectInstance;
  public static Count aspectOf() {
    return Count.aspectInstance;
  }

  public static boolean hasAspect() {
    return Count.aspectInstance != null;
  }

  static {
    Count.aspectInstance = new Count();
  }

}
```

Generated class `SexCheck` **(****↪ p. 77****)**

```
/*   Generated by AspectJ version 1.0.1 */
package hound;
class SexCheck {
  /* IntroducedDec(dec: MethodDec(id: checkSex)) */

  public final void after0$ajc(Hound h, String x) {
    if (!h.checkSex_hound_SexCheck$ajc$backdoor(x)) {
      System.out.println("Illegal sex; assume female!");
      this.setSex$method_call(h, "female");
      return;
    }
  }

  SexCheck() {
    super();
  }
  public static SexCheck aspectInstance;
  public static SexCheck aspectOf() {
    return SexCheck.aspectInstance;
  }
```

```
  public static boolean hasAspect() {
    return SexCheck.aspectInstance != null;
  }

  private void setSex$method_call(Hound target,
    final String sex) {
    target.setSex(sex);
  }

  static {
    SexCheck.aspectInstance = new SexCheck();
  }

}
```

5.3 Factory Pattern & Contract Enforcement

Creational pattern

A factory pattern is a *creational pattern*, that abstract the object instantiation process. It hides how objects are created and helps make the overall system indepentent of how its objects are created and composed. In Java the idiom for object creation is the `new` operator. Creational patterns allow us to write methods that create new objects without explicitly using the `new` operator. We are able to write a method that can instantiate different objects and that can be extended to instantiate other newly-developed objects, without modifying the method's code.

In the following example we assume that a fisherman uses a tool ”fishing gear” and a huntsman a tool ”rifle”. For the both tools special using intstructions exist. The class `Tool` (↪ p. 90) is an abstract base class that defines a common interface to both different types of tools in our context. The class `FishingGear` (↪ p. 90) is a specialization of `Tool` (↪ p. 90) and implements its abstract method `use()`. The same task has the class `Huntsman` (↪ p. 94). The abstract base class `Man` holds the common data for fishermen and huntsmen; here the slots `name`, `licence` and `isHunter`.

A fisherman has only access to a fishing gear and a huntsman only to a rifle. With the method `getTool()` we implement this access. In the body of this method we instantiate an object of the class `ToolCreator` (↪ p. 91), defining the factory method `createTool()`. To decide which type of tool should be created, the factory method has a parameter of type *int*. The valid values for this parameter are coded in static final slots of the class itself. That is why we can make the decision on the class variables `Tool.FISHINGGEAR` and `Tool.RIFLE`.

The same factory concept is implemented in the class `ManCreator` (↪ p. 94) for the instantiation of the both types `Fisherman` (↪ p. 93) and `Huntsman` (↪ p. 94).

All files of this example must be compiled. Therefore we define the package `factory` and the file `files.lst` (↪ p. 89) as an `argfile`.

Argument list `files.lst`

```
Tool.java
```

```
FishingGear.java
Rifle.java
ToolCreator.java
Man.java
Fisherman.java
Huntsman.java
ManCreator.java
Protection.java
Pattern.java
```

Class `Tool`

```
/**
 *  Example "Factory Pattern"
 *  Abstract base class Tool
 *  = common to type Rifle
 *  and FishingGear
 *
 *@author     Bonin
 *@version    1.0
 */

package factory;

public abstract class Tool
{
   public abstract String use();
}
```

Class `FishingGear`

```
/**
 *  Example "Factory Pattern" Class FishingGear
 *
 *@author     Bonin
 *@version    1.0
 */

package factory;

public class FishingGear extends Tool
{
```

```
   public String use()
   {
      return "Instructions for use the fishing gear.";
   }
}
```

Class `Rifle`

```
/**
 *  Example "Factory Pattern" Class Rifle
 *
 *@author     Bonin
 *@version    1.0
 */

package factory;

public class Rifle extends Tool
{
   public String use()
   {
      return "Instructions for use the rifle.";
   }
}
```

Class `ToolCreator` This factory method `createTool()` is able to create the two kinds of objects: `FishingGear` and `Rifle`. It takes a parameter that identifies the kind of object to create. The objects that the factory method creates, share the same interface. Once the identifier is read, the framework calls `createTool()` with the identifier `id` passed in. The `createTool()` method instantiates and returns the appropriate `Tool` reference.

```
/**
 *  Example "Factory Pattern" Class ToolCreator
 *
 *@author     Bonin
 *@version    1.0
 */

package factory;
```

```
public class ToolCreator
{
   public final static int FISHINGGEAR = 0;
   public final static int RIFLE = 1;


   public Tool createTool(int id)
   {
      switch (id)
      {
         case FISHINGGEAR:
            return new FishingGear();
         case RIFLE:
            return new Rifle();
      }
      return null;
   }
}
```

Class `Man`

```
/**
 *  Example "Factory Pattern" Abstract class Man
 *
 *@author     Bonin
 *@version    1.0
 */

package factory;

public abstract class Man
{
   protected String name;
   protected String licence;
   protected boolean isHunter;


   public Man(String name, String licence)
   {
      this.name = name;
      this.licence = licence;
   }
```

```
   public String getName()
   {
      return name;
   }


   public String getLicence()
   {
      return licence;
   }


   public boolean isHunter()
   {
      return isHunter;
   }


   public abstract Tool getTool();
}
```

Class `Fisherman`

```
/**
 *  Example "Factory Pattern" Class Fisherman
 *
 *@author     Bonin
 *@version    1.0
 */

package factory;

public class Fisherman extends Man
{
   public Fisherman(String name, String licence)
   {
      super(name, licence);
      isHunter = false;
   }
```

```
   public Tool getTool()
   {
      ToolCreator creator = new ToolCreator();
      return
            creator.createTool(ToolCreator.FISHINGGEAR);
   }
}
```

Class `Huntsman`

```
/**
 *  Example "Factory Pattern" Class Huntsman
 *
 *@author     Bonin
 *@version    1.0
 */

package factory;

public class Huntsman extends Man
{
   public Huntsman(String name, String licence)
   {
      super(name, licence);
      isHunter = true;
   }


   public Tool getTool()
   {
      ToolCreator creator = new ToolCreator();
      return
            creator.createTool(ToolCreator.RIFLE);
   }
}
```

Singleton Pattern

Class `ManCreator` This factory method `createMan()` is able to create the two kinds of objects: `Fisherman` and `Huntsman`. Its is implemented as a `Singleton Pattern`, with a `proteced` constructor and a `static` method `instance()` as the *singleton-wrapper* (↪ Section 5.4, p. 98).

```java
/**
 *  Example "Factory Pattern" Class ManCreator
 *
 *@author     Bonin
 *@version    1.0
 */

package factory;

public class ManCreator
{
   public final static int FISHERMAN = 0;
   public final static int HUNTSMAN = 1;


   // Singleton Pattern
   protected ManCreator() { }


   private static ManCreator _instance = null;


   public static ManCreator instance()
   {
      if (null == _instance)
      {
         _instance = new ManCreator();
      }
      return _instance;
   }


   // Dispatch the right Type creation
   public Man createMan
         (int id, String name, String licence)
   {
      switch (id)
      {
         case FISHERMAN:
            return
                  new Fisherman(name, licence);
         case HUNTSMAN:
```

```
            return
                  new Huntsman(name, licence);
      }
      return null;
   }
}
```

Aspect `Protection` This aspect uses the `withincode()` construct to denote the join points that occur within the body of the factory method on `ToolCreator`. The property-based crosscutting mechanisms can define more sophisticated *Contract Enforcement*. The following use of these mechanisms is to identify constructor call, here `Rifle()`, that, in the factory pattern program, should not exist.

`within()`

jumping aspect

This aspect supposes that AspectJ is able to memorize the aspect advices that have been already applied. The construct `cflow()` supports the so called *jumping aspect*, because the join points seems to be jumping around the code depending on the context in which a component is used (Brichau and al. ↪ [Pawlak+, 2001],p. 5).

```
/**
 * Example "Factory Pattern"
 * Aspect RifleProtection
 * @author Bonin
 * @version 1.0
 */

package factory;

aspect Protection
{
    pointcut ok() : within(ToolCreator) &&
        withincode(Tool createTool(..)) &&
        call(new (..));

    pointcut notOk() : !cflow(ok()) &&
        within(Rifle) &&
        execution(new (..));
    before() : notOk()
    {
        System.out.println
            ("Illegal call: " + thisJoinPoint);
    }
}
```

Class `Pattern`

```
/**
 *  Example "Factory Pattern" Class Pattern
 *
 *@author     Bonin
 *@version    1.0
 */

package factory;

public class Pattern
{
   public static void main(String[] args)
   {

      ManCreator mc = ManCreator.instance();
      Man f = mc.createMan
            (ManCreator.FISHERMAN, "Meyer", "invalid");
      Man h = mc.createMan
            (ManCreator.HUNTSMAN, "Bonin", "valid");

      System.out.println(f.getName() + ": " +
            f.getTool().use());
      System.out.println(h.getName() + ": " +
            h.getTool().use());

      // illegal use of constructor
      Rifle r = new Rifle();
   }
}
```

Protocol `Pattern.log`

```
d:\bonin\aosd\code>ajc -version
ajc version 1.0.1 (built 18.12.2001 11:11 PST)
    running on java 1.3.1

d:\bonin\aosd\code>ajc -argfile factory/files.lst

D:\bonin\aosd\code>java factory.Pattern
Meyer: Instructions for use the fishing gear.
```

```
Bonin: Instructions for use the rifle.
Illegal call: execution(factory.Rifle())

D:\bonin\aosd\code>
```

5.4 Singleton Pattern

Sometimes we need to create a single instance of a given class. A simple approach to this problem is to use `static` members and/or `static` methods for the `singleton` functionality.

5.4.1 `static`-Slot & `private`-Constructor

Class `SimpleApproach`

```
/**
 *  "Singleton Pattern" Example for a simple
 *   approach Class SimpleApproach
 *
 *@author     Bonin
 *@version    1.0
 */

package singleton;

public class SimpleApproach
{
   public final static SimpleApproach
         instance = new SimpleApproach();


   private SimpleApproach() { }


   public void someMethod()
   {
      System.out.println("Do something!");
   }
}
```

Class `UseSimpleApproach`

```
/**
 *  "Singleton Pattern" Example for use a singleton instance
 *  Class UseSimpleAppraoch
 *
 *@author     Bonin
 *@version    1.0
 */

package singleton;

public class UseSimpleApproach
{
   public static void main(String[] args)
   {
      SimpleApproach foo = SimpleApproach.instance;
      foo.someMethod();
   }
}
```

5.4.2 `static`-Method & `private`-Constructor

Class `Approach`

```
/**
 *  "Singleton Pattern" Example for an approach Class Approach
 *
 *@author     Bonin
 *@version    1.0
 */

package singleton;

public class Approach
{
   private static Approach _instance = null;


   protected Approach() { }


   public static Approach instance()
   {
```

```
      if (null == _instance)
      {
         _instance = new Approach();
      }
      return _instance;
   }


   public void someMethod()
   {
      System.out.println("Do something!");
   }
}
```

Class `UseApproach`

```
/**
 *  "Singleton Pattern" Example for use a singleton instance
 *  Class UseAppraoch
 *
 *@author     Bonin
 *@version    1.0
 */

package singleton;

public class UseApproach
{
   public static void main(String[] args)
   {
      Approach foo = Approach.instance();
      foo.someMethod();
   }
}
```

5.4.3 Registry based on `static Hashtable`

For the subclassing issue, we use a registry, which is a `static` hash table for mapping keys to singleton instances. Therefore we code a `static` initializer that creates an instance of the class and adds it to the registry when the byte-code for this class is first loaded.

Class `ApproachRegistry`

```
/**
 *  "Singleton Pattern" Example for an
 *  approach with a Hashtable Class
 *  ApproachRegistry
 *
 *@author     Bonin
 *@version    1.0
 */

package singleton;

import java.util.Hashtable;

public class ApproachRegistry
{
   protected static Hashtable _registry =
         new Hashtable();

   static
   {
      ApproachRegistry foo =
            new ApproachRegistry();
      _registry.put(foo.getClass().getName(), foo);
      System.out.println
            ("created: " + foo.getClass().getName());
   }


   protected ApproachRegistry() { }


   public static ApproachRegistry instance(String byname)
   {
      return (ApproachRegistry) (_registry.get(byname));
   }


   public void someMethod()
   {
      System.out.println("Do something!");
   }
```

```
}
```

Class `SubApproachRegistry`

```
/**
 *  "Singleton Pattern" Example for an approach with a
 *  Hashtable Class ApproachRegistry
 *
 *@author     Bonin
 *@version    1.0
 */

package singleton;

import java.util.Hashtable;

public class SubApproachRegistry extends ApproachRegistry
{
   static
   {
      SubApproachRegistry foo =
            new SubApproachRegistry();
      _registry.put(foo.getClass().getName(), foo);
      System.out.println
            ("created: " +
            foo.getClass().getName());
   }

   public String testSlot = "my value";


   public void subMethod()
   {
      System.out.println("Do subMethod things!");
   }
}
```

Class `UseApproachRegistry`

```
/**
 *  "Singleton Pattern" Example for use a singleton instance
```

```
 *  Class UseAppraochRegistry
 *
 *@author     Bonin
 *@version    1.0
 */

package singleton;

public class UseApproachRegistry
{
   public static void main(String[] args)
   {
      ApproachRegistry foo =
            ApproachRegistry.instance
            ("singleton.ApproachRegistry");

      SubApproachRegistry bar = (SubApproachRegistry)
            SubApproachRegistry.instance
            ("singleton.SubApproachRegistry");
      SubApproachRegistry baz = (SubApproachRegistry)
            SubApproachRegistry.instance
            ("singleton.SubApproachRegistry");

      foo.someMethod();
      bar.subMethod();
      bar.testSlot = "one value";
      System.out.println("baz.testSlot: " +
            baz.testSlot);
      System.out.println("bar.testSlot: " +
            bar.testSlot);
   }
}
```

Protocol `UseApproachRegistry.log`

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.1 (built 18.12.2001 11:11 PST)
   running on java 1.3.1

D:\bonin\aosd\code>ajc singleton/ApproachRegistry.java

D:\bonin\aosd\code>ajc singleton/SubApproachRegistry.java
```

```
D:\bonin\aosd\code>ajc singleton/UseApproachRegistry.java

D:\bonin\aosd\code>java singleton.UseApproachRegistry
created: singleton.ApproachRegistry
created: singleton.SubApproachRegistry
Do something!
Do subMethod things!
baz.testSlot: one value
bar.testSlot: one value

D:\bonin\aosd\code>
```

5.4.4 Abstract Factory, Factory Method and Functor

The following example is based on [Waldhoff, 1998]. This approach allows us to easily alter the specific `Singleton` instance published — either dynamically or at compile-time – without altering the code that uses the `Singleton`. The `SingletonWrapper` (↪ p. 105) can be created for pre-existing classes such as those provided with an API or framework.

Argument list `files.lst`

```
Singleton.java
SingletonFactoryFunctor.java
SingletonWrapper.java
UseSingletonWrapper.java
```

Class `Singleton`

```
/**
 *  "Singleton Pattern" Example for an
 *   Abstract Factory, Factory Method and
 *  Fuctor Class Singleton
 *
 *@author     Bonin
 *@version    1.0
 */

package singleton;

public class Singleton
{
   private String slot = "my value";
```

```
   public String getSlot()
   {
      return slot;
   }


   public void setSlot(String slot)
   {
      this.slot = slot;
   }
}
```

Interface `SingletonFactoryFunctor` We code an interface defining objects that create `Singleton` instances.

```
/**
 *  "Singleton Pattern" Example for an Abstract Factory,
 *  Factory Method and Fuctor Interface SingletonFactoryFunctor
 *  defining objects that can create Singleton
 *
 *@author     Rod Waldhoff
 *@version    1.0
 */

package singleton;

public interface SingletonFactoryFunctor
{
   public Singleton makeInstance();
}
```

Class `SingletonWrapper` This class is a static container for a single instance of the class `Singleton`.

```
/**
 *  "Singleton Pattern" Example for an Abstract Factory,
 *  Factory Method and Functor Class SingletonWrapper is a
 *  static container for a single instance of the Singleton
 *
```

```
 *@author     Rod Waldhoff (Bonin did little modifications)
 *@version    1.0
 */

package singleton;

public final class SingletonWrapper
{
   // A reference to a possible alternate factory
   private static SingletonFactoryFunctor
         _factory = null;

   // A reference to the current instance
   private static Singleton _instance = null;


   // This is the default factory method. It is
   // called to create a new Singleton when a
   // new instance is need and _factory is null.
   private static Singleton makeInstance()
   {
      return new Singleton();
   }


   // This is the accessor for the Singleton.
   public static synchronized Singleton instance()
   {
      if (null == _instance)
      {
         _instance = (null == _factory) ?
               makeInstance() :
               _factory.makeInstance();
      }
      return _instance;
   }


   // Sets the factory method used to create
   // new instances. You can set the factory
   // method to null to use the default method.
   public static synchronized void setFactory
         (SingletonFactoryFunctor factory)
   {
      _factory = factory;
   }
```

```
   // Sets the current Singleton instance.
   // You can set this to null to force a new
   // instance to be created the next
   // time instance() is called.
   public static synchronized void setInstance
         (Singleton instance)
   {
      _instance = instance;
   }
}
```

Class `UseSingletonWrapper`

```
/**
 *  "Singleton Pattern" Example for an Abstract Factory,
 *  Factory Method and Functor Class UseSingletonWrapper
 *
 *@author     Bonin
 *@version    1.0
 */

package singleton;

public class UseSingletonWrapper
{
   public static void main(String[] args)
   {
      Singleton foo = SingletonWrapper.instance();
      foo.setSlot("one value");

      Singleton bar = SingletonWrapper.instance();

      System.out.println("foo: " + foo.getSlot());
      System.out.println("bar: " + bar.getSlot());

      SingletonWrapper.setInstance(null);
      Singleton baz = SingletonWrapper.instance();
      System.out.println("baz: " + baz.getSlot());
   }
}
```

Protocol `UseSingletonWrapper.log`

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.1 (built 18.12.2001 11:11 PST)
    running on java 1.3.1

D:\bonin\aosd\code>ajc -argfile singleton/files.lst

D:\bonin\aosd\code>java singleton.UseSingletonWrapper
foo: one value
bar: one value
baz: my value

D:\bonin\aosd\code>
```

5.4.5 Aspect-oriented Approach

`filesAspect.lst`

```
Singleton.java
SingletonAspect.java
UseSingletonAspect.java
```

Class `Singleton` ↪ p. 104

Class `SingletonAspect`

```
/**
 * "Singleton Pattern"
 * Example for an
 * aspect-oriented approach
 *@author Bonin
 *@version 1.0
 */

package singleton;

aspect SingletonAspect
{
    public static boolean
        Singleton.unused = true;
    public static Singleton
        Singleton.firstinstance = null;
```

```
    public static Singleton Singleton.instance()
    {
        if (Singleton.unused) {
           Singleton.unused = false;
           Singleton.firstinstance = new Singleton();
           }
        return Singleton.firstinstance;
    }
}
```

Class `UseSingletonAspect`

```
/**
 *  "Singleton Pattern" Example for an aspect-oriented
 *  approach Class UseSingletonAspect
 *
 *@author     Bonin
 *@version    1.0
 */

package singleton;

public class UseSingletonAspect
{
   public static void main(String[] args)
   {
      Singleton foo = Singleton.instance();
      foo.setSlot("one value");

      Singleton bar = Singleton.instance();

      System.out.println("foo: " + foo.getSlot());
      System.out.println("bar: " + bar.getSlot());
   }
}
```

Protocol `UseSingletonAspect.log`

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.1 (built 18.12.2001 11:11 PST)
    running on java 1.3.1
```

```
D:\bonin\aosd\code>ajc -argfile singleton/filesAspect.lst

D:\bonin\aosd\code>java singleton.UseSingletonAspect
foo: one value
bar: one value

D:\bonin\aosd\code>
```

5.5 Pointcut: Multiple Advices

`dominates`

An aspect may declare that the advice in it dominates the advice in some other aspect. Such *aspect domination* is specified by the key word `dominates`.
`aspect` AspectName `dominates` AspectNameOther {. . .}

For example, we declare that the advice in an aspect `Baz` should be done before the advice in an aspect `Bar` as followed:
`aspect Baz dominates Bar` {...}

Without this declaration AspectJ would use the alphabetical order, so first `Bar` and then `Baz`. In the following example we have the simple class `Foo` (↪ p. 110) with an instance variable `slot` and its get- and set-method. Our three aspects `Baa` (↪ p. 111), `Bar` (↪ p. 112) and `Baz` (↪ p. 112) have the same pointcut:

```
target(f) &&  call(String getSlot());
```

To get the sequence `Baa`, `Baz`, and `Bar` we declare that `Baz` dominates `Bar`. The result is shown in the file `Foo.log` (↪ p. 113).

Argument list `files.lst`

```
Foo.java
Baz.java
Bar.java
Baa.java
```

Class `Foo`

```
/**
 *  Simple "Foo" application
 *
 *@author     Bonin
 *@version    1.0
 */
```

```
package multiple;

public class Foo
{
   private String slot;


   public String getSlot()
   {
      return slot;
   }


   public void setSlot(String slot)
   {
      this.slot = slot;
   }


   public static void main(String[] args)
   {
      Foo myObj = new Foo();
      myObj.setSlot("This is my instance of class Foo.");

      System.out.println(myObj.getSlot());
   }
}
```

Aspect `Baa`

```
/**
 * More than one advice apply at a joint point
 * "Baa" aspect
 * @author Bonin
 * @version 1.0
 */

package multiple;

aspect Baa {
```

```
    pointcut performSlot(Foo f):
        target(f) &&  call(String getSlot());

    before(Foo f) : performSlot(f)
    {
        System.out.println(
          "Doing aspect Baa performSlot()!");
        f.setSlot("Value of aspect Baa!");
    }
}
```

Aspect `Bar`

```
/**
 * More than one advice apply at a joint point
 * "Bar" aspect
 *@author Bonin
 *@version 1.0
 */

package multiple;

aspect Bar {

    pointcut applySlot(Foo f):
        target(f) &&  call(String getSlot());

    before(Foo f) : applySlot(f)
    {
        System.out.println(
          "Doing aspect Bar applySlot()!");
        f.setSlot("Value of aspect Bar!");
    }
}
```

Aspect `Baz` The aspect `Baz` declares that its advice `before()` dominates the advice `before()` in the aspect `Bar`. That is why first doing `Baz` and then `Bar`.

```
/**
 * More than one advice apply at a joint point
 * "Baz" aspect
 *@author Bonin
 *@version 1.0
```

```
 */

package multiple;

aspect Baz dominates Bar {

    pointcut runSlot(Foo f):
        target(f) &&  call(String getSlot());

    before(Foo f) : runSlot(f)
    {
        System.out.println(
          "Doing aspect Baz runSlot()!");
        f.setSlot("Value of aspect Baz!");
    }
}
```

Protocol `Foo.log`

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.3
  (built 08.02.2002 12:47 PST) running on java 1.3.1

D:\bonin\aosd\code>ajc -argfile multiple/files.lst

D:\bonin\aosd\code>java multiple.Foo
Doing aspect Baa performSlot()!
Doing aspect Baz runSlot()!
Doing aspect Bar applySlot()!
Value of aspect Bar!

D:\bonin\aosd\code>
```

5.6 Access to `private` Members

The Java access control rules are valid to aspects. For example, code written in an aspect can't refer to `private` members of the base program. In our class `Foo` (↪ p. 110) the instance variable `slot` is `private` and that is why we would get a compile error[3] if we access `slot` in an aspect. To allow this directly access, the aspect must be declared `privileged`.

`privi-leged`

[3] `Foo.slot has private access`

`privileged aspect` AspectName {...}

A simple aspect `Baz` (↪ p. 114) with the modifier `privileged` documents this access. The result is shown in the file `Foo.log` (↪ p. 114).

Argument list `files.lst`

```
Foo.java
Baz.java
```

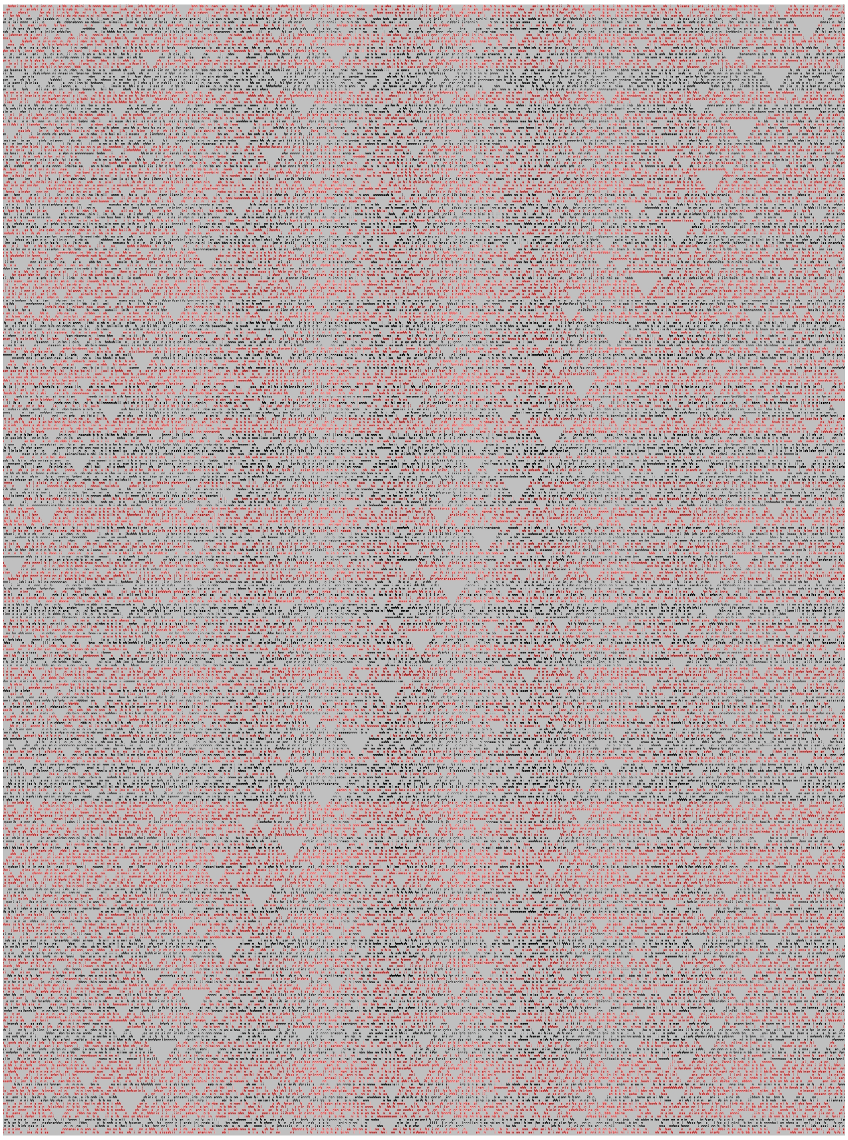
Aspect `Baz`

```
/**
 * Privileged aspect "Baz"
 *@author Bonin
 *@version 1.0
 */

package privilege;

privileged aspect Baz
{

    pointcut runSlot(Foo f):
        target(f) &&  call(String getSlot());

    before(Foo f) : runSlot(f)
    {
        System.out.println(
          "Doing aspect Baz runSlot()!");
        f.slot = "Value of aspect Baz!";
    }
}
```

Protocol `Foo.log`

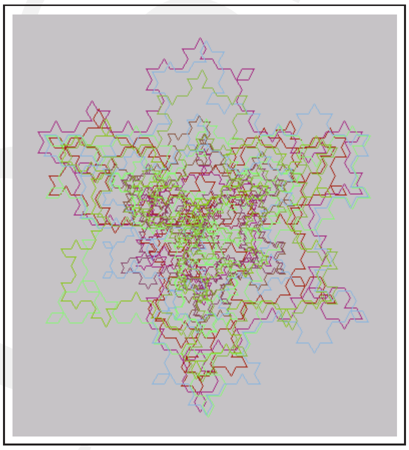
```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.3
  (built 08.02.2002 12:47 PST) running on java 1.3.1

D:\bonin\aosd\code>ajc -argfile privilege/files.lst

D:\bonin\aosd\code>java privilege.Foo
```

```
Doing aspect Baz runSlot()!
Value of aspect Baz!

D:\bonin\aosd\code>
```



Chapter 6

Composition Paradigm



The aspect-oriented programming paradigm evolves into a composition paradigm on a base of a component model, a flexible composition technique, and a composition language.

A composition language serves to write composition recipes, i. e., recipes for how to build a system from components off-the-shelf (↪ [Aßmann, 2003] p. 3).

6.1 Programming *in-the-small* and *in-the-large*

> "We argue that structuring a large collection of modules to form a "system" is an essentially distinct and different intellectual activity from that of constructing the individual modules. That is, we distinguish programming-in-the-large from programming-in-the-small."
> (↪ [DeRemer / Kron, 1976] quoted from [Aßmann, 2003] p. 72)

6.2 Inject/J

Inject/J

`Inject/J` has been developed as a scripting language for weaving (↪ [Genssler / Kuttruff, 2001]). `Inject/J` is a metaprogramming language, keeps a representation of the Java core, and describes weaving with a script. The script navigates over the program representation, matches join points, and weaves in advices. Instead of AspectJ, `Inject/J` provides a language to write new weavers and combines the aspect with the weaver script. While AspectJ expresses the core-aspect relation with specific language constructs in the aspect, `Inject/J` describes the weaving directly and represents the core-aspect relation implicitly. The scripting language is simple, and in many cases the weavers are easy to understand. (↪ [Aßmann, 2003] pp. 78)

6.3 Hyper/J

Hyper/J

While aspect-oriented programming distinguishes a primary dimension of concern (≡ the core) from secondary ones (≡ the aspects), *hyperspace programming* treats all dimensions equally. Hyperspace programming descibes how concerns can be merged. It uses hyperslices and hypermodules, a more general component concept than classes. For Java, it is exemplified in the tool Hyper/J[1] (↪ [Ossher / Tarr, 1999]).

6.4 Piccola

Piccola[2] (π-calculus-based composition language) is a small, pure language for building applications from software components. Piccola is small in the sense that its syntax is tiny, and it is pure in the sense that it provides only compositional features — computation is performed entirely by components of the host programming language.

JPiccola

There currently exist two experimental implementations of Piccola: `JPiccola` is used to compose Java components, and `SPiccola` can compose components written in `Squeak`[3].

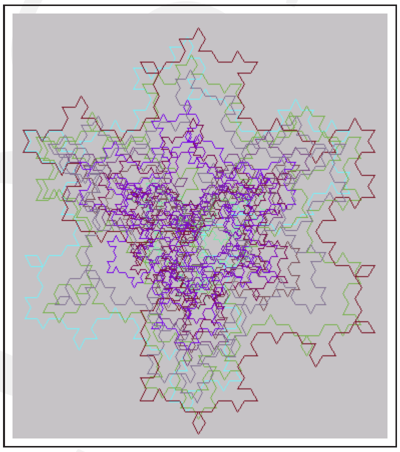
[1] ↪ `http://www.research.ibm.com/hyperspace/HyperJ/HyperJ.htm` (visited 7-Jul-2003)

[2] Software Composition (SCG), Institute of Computer Science and Applied Mathematics, University Bern, Switzerland, Web site: ↪ `http://www.iam.unibe.ch/ scg/Research/Piccola/` (visited 7-Jul-2003)

[3] Squeak is a "media authoring tool" — "Squeakland has been developed to offer a variety of fun experiences to people of all ages who use their computers to create. Squeakland is meant to be a playground for developing a community of people who want to work together to invent new media types." ↪ `http://www.squeakland.org/whatis/whatishome.html` (visited 7-Jul-2003)

Chapter 7

Exercises



7.1 Java Properties

You can determine your system locations, such as `sun.boot.class` or `user.home` directory, by running the following aspect `ListProperties`[1].

Argument list `files.lst`

```
ListProperties.java
```

[1] Pattern for this program ↪ [Flenner et al., 2003] p. 37–38.

Aspect `ListProperties`

```
/**
 *  "List Properties" application
 *@author Bonin
 * @version 1.0
 */

package property;

public aspect ListProperties
{
    public static void main(String[] args)
    {
        System.out.println(
          "user.home = " +
          System.getProperty("user.home") +
          "\n\n" +
          "java.ext.dirs = " +
          System.getProperty("java.ext.dirs") +
          "\n\n" +
          "java.class.path = " +
          System.getProperty("java.class.path") +
          "\n\n" +
          "sun.boot.class.path = " +
          System.getProperty("sun.boot.class.path"));
    }
}
```

Protocol `ListProperties.log`

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.5 (built 27.06.2002 16:59 PST)
  running on java 1.4.1-beta

D:\bonin\aosd\code>ajc -argfile property/files.lst

D:\bonin\aosd\code>java property.ListProperties
user.home = C:\Dokumente und Einstellungen\bonin.FBW

java.ext.dirs = c:\programme\java2\j2sdk1.4.1\jre\lib\ext

java.class.path = .;C:\Programme\POET61\lib\POET6ODMG3JC_SDK.jar;
  C:\Programme\POET61\lib\POET6ODMG3JC_IIOPClient.jar;
```

```
  C:\Programme\POET61\lib\POET6ODMG3JC_RMIClient.jar;
  C:\Programme\POET61\lib\POET6ODMG3JC_Runtime.jar;
  C:\Programme\POET61\lib\POET6ODMG3JC_JServer.jar;
  C:\programme\VisualCafe\Java\Lib;
  C:\programme\VisualCafe\Java\Lib\SYMCLASS.ZIP;
  C:\programme\VisualCafe\Java\Lib\CLASSES.ZIP;
  C:\programme\VisualCafe\Java\Lib\COLLECTIONS.ZIP;
  C:\programme\VisualCafe\Java\Lib\ICEBROWSERBEAN.JAR;
  C:\programme\VisualCafe\Java\Lib\JSDK.JAR;
  C:\programme\VisualCafe\Java\Lib\SYMTOOLS.JAR;
  C:\programme\VisualCafe\JFC\SWINGALL.JAR;
  C:\programme\VisualCafe\Bin\Components\SFC.JAR;
  C:\programme\VisualCafe\Bin\Components\SYMBEANS.JAR;
  C:\programme\VisualCafe\Java\Lib\DBAW.ZIP;
  C:\programme\VisualCafe\Bin\Components\DBAW_AWT.JAR;
  C:\programme\VisualCafe\Bin\Components\Databind.JAR;
  C:\programme\VisualCafe\Java\Lib\Olite35.jar;
  c:\programme\aspectj1.0\lib\aspectjrt.jar

sun.boot.class.path = c:\programme\java2\j2sdk1.4.1\jre\lib\rt.jar;
  c:\programme\java2\j2sdk1.4.1\jre\lib\i18n.jar;
  c:\programme\java2\j2sdk1.4.1\jre\lib\sunrsasign.jar;
  c:\programme\java2\j2sdk1.4.1\jre\lib\jsse.jar;
  c:\programme\java2\j2sdk1.4.1\jre\lib\jce.jar;
  c:\programme\java2\j2sdk1.4.1\jre\lib\charsets.jar;
  c:\programme\java2\j2sdk1.4.1\jre\classes

D:\bonin\aosd\code>
```

7.2 Shortest Java Application with Aspect

Argument list `files.lst`

```
Mini.java
MainMethod.java
```

Class `Mini`

```
/**
 *  "Shortest Java" application
 *
 *@author     Bonin
 *@version    1.0
 */
```

```
package mini;

public class Mini
{
}
```

Aspect `MainMethod`

```
/**
 * "Shortest Java" application
 *@author Bonin
 *@version 1.0
 */

package mini;

public aspect MainMethod
{
    public static void Mini.main(String[] args) {}
}
```

Protocol `Mini.log`

```
D:\bonin\aosd\code>java -fullversion
java full version "1.4.1-beta-b14"

D:\bonin\aosd\code>ajc -argfile mini/files.lst

D:\bonin\aosd\code>java mini.Mini

D:\bonin\aosd\code>cd mini

D:\bonin\aosd\code\mini>dir
        26 files.lst
       591 MainMethod.class
       179 MainMethod.java
       517 Mini.class
       117 Mini.java
       182 Mini.log

D:\bonin\aosd\code\mini>
```

7.3 Simple Object-Oriented Expressions

This example using simple expressions illustrating object-oriented programming features like method invocation, collaboration and cascaded method call. The class name _ is choosen getting a simple object-oriented notation for the integer arithmetic in Java.

Argument list files.lst

```
_.java
Use.java
```

Class _

```
/**
 *  "Simple expression illustrating OO" application
 *
 *@author     Bonin
 *@version    1.0
 */

package oo;

public class _
{
   int value = 0;


   public static _ make(int p)
   {
      _ internal = new _();
      internal.value = p;
      return internal;
   }


   public int getValue()
   {
      return value;
   }


   public void setValue(int value)
   {
```

```
   this.value = value;
}


public _ neg()
{
   this.setValue(this.getValue() * -1);
   return this;
}


public _ add(_ param)
{
   this.setValue(this.getValue() + param.getValue());
   return this;
}


public _ sub(_ param)
{
   this.setValue(this.getValue() - param.getValue());
   return this;
}


public _ mult(_ param)
{
   this.setValue(this.getValue() * param.getValue());
   return this;
}


public _ div(_ param)
{
   this.setValue(this.getValue() / param.getValue());
   return this;
}


public boolean identical(_ param)
{
   return this == param;
```

```
   }

   public boolean equal(_ param)
   {
      return this.getValue() == param.getValue();
   }

   public boolean less(_ param)
   {
      return this.getValue() < param.getValue();
   }

   public boolean greater(_ param)
   {
      return this.getValue() > param.getValue();
   }
}
```

Class `Use`

```
/**
 *  "Simple expression illustrating OO" application
 *
 *@author     Bonin
 *@version    1.0
 */

package oo;

public class Use
{
   public static void main(String[] args)
   {
      _ a = _.make(3);
      _ b = _.make(3);
      _ c = _.make(4);

      System.out.println
             (
```

```
            "a = " + a.getValue() + "\n" +
            "b = " + b.getValue() + "\n" +
            "c = " + c.getValue() + "\n" +
            "a == a? " + a.identical(a) + "\n" +
            "a == b? " + a.identical(b) + "\n" +
            "a < b? " + a.less(b) + "\n" +
            "a = b? " + a.equal(b) + "\n" +
            "a > b? " + a.greater(b) + "\n" +
            "-a = " + a.neg().getValue() + "\n" +
            "a + a = " + a.add(a).getValue() + "\n" +
            "a + b + c = " + a.add(b).add(c).getValue() + "\n" +
            "a - b * c = " + a.sub(b.mult(c)).getValue() + "\n" +
            "b = " + b.getValue() + "\n" +
            "b / a = " + b.div(a).getValue()
            );
   }
}
```

Protocol `Use.log`

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.5
    (built 27.06.2002 16:59 PST) running on java 1.4.1-beta

D:\bonin\aosd\code>ajc -argfile oo/files.lst

D:\bonin\aosd\code>java oo.Use
a = 3
b = 3
c = 4
a == a? true
a == b? false
a < b? false
a = b? true
a > b? false
-a = -3
a + a = -6
a + b + c = 1
a - b * c = -11
b = 12
b / a = -1
```

```
D:\bonin\aosd\code>
```

7.4 Simple Logging

The first exercise consists three classes and an aspect, representing a very simple logging:

- the class `Car` (↪ p. 127),
- the class `CarOwner` (↪ p. 128),
- the class `CarProg` (↪ p. 129), and
- the aspect `Logger` (↪ p. 130),

Argument list `files.lst`

```
Car.java
CarOwner.java
CarProg.java
Logger.java
```

Class `Car`

```
/**
 *  "Car" application
 *
 *@author     Bonin
 *@version    1.0
 */

package car;

public class Car
{
   private String typ;
   private int kw;


   public String getTyp()
   {
      return typ;
   }
```

```
   public int getKw()
   {
      return kw;
   }


   public void setKw(int kw)
   {
      this.kw = kw;
   }


   public Car(String typ)
   {
      this.typ = typ;
   }
}
```

Class `CarOwner`

```
/**
 *  "Car" application
 *
 *@author     Bonin
 *@version    1.0
 */

package car;

public class CarOwner
{
   private String owner;
   private Car car;


   public String getOwner()
   {
      return owner;
   }


   public Car getCar()
```

```
   {
      return car;
   }


   public void setCar(Car car)
   {
      this.car = car;
   }


   public CarOwner(String owner)
   {
      this.owner = owner;
   }
}
```

Class `CarProg`

```
/**
 *  "Car" application
 *
 *@author     Bonin
 *@version    1.0
 */

package car;

public class CarProg
{
   public static void main(String[] args)
   {
      Car landy = new Car("Land Rover Discovery I");
      landy.setKw(87);

      CarOwner i = new CarOwner("Hinrich Bonin");
      i.setCar(landy);

      System.out.println(
            i.getOwner() + " owns " +
            i.getCar().getTyp() + " with " +
            i.getCar().getKw() + "kw.");
```

```
   }
}
```

Aspect `Logger`

```
/**
 * "Car" application
 *@author Bonin
 *@version 1.0
 */

package car;

public aspect Logger
{
    pointcut logging():
        call(void setCar(Car)) ||
        call(String Car.getTyp()) ||
        call(void Car.setKw(int)) ||
        call(int Car.getKw());

    after(): logging()
        {
            System.out.print(
                thisJoinPoint.toLongString() +
                " Args:\n");

            Object[] o = thisJoinPoint.getArgs();
            for(int i = 0; i < o.length; i++) {
                System.out.print(
                    "-->" +
                    o[i].getClass() +
                    "(" + o[i] + ")");
            }
            System.out.println("\n");
        }
}
```

Protocol `CarProg.log`

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.3
```

```
    (built 08.02.2002 12:47 PST) running on java 1.3.1

D:\bonin\aosd\code>ajc -argfile car/files.lst

D:\bonin\aosd\code>java car.CarProg
call(public void car.Car.setKw(int)) Args:
-->class java.lang.Integer(87)

call(public void car.CarOwner.setCar(car.Car)) Args:
-->class car.Car(car.Car@7077e)

call(public java.lang.String car.Car.getTyp()) Args:


call(public int car.Car.getKw()) Args:


Hinrich Bonin owns Land Rover Discovery I with 87kw.

D:\bonin\aosd\code>
```

7.5 Sequence of Aspects

Pointcuts can be created by using join point wildcards. Our example (↪ p. 132) uses the following specification for the three methods `setSlot()`, `setNothing()` and `setFoo()`:

```
call(void set*(..))
```

The pointcut `callSet()` (↪ p. 132) contains the designator `target()` and the designator `args()`. The `target()` designator is used to provide access to the target object of the method calls. The purpose of the `args()` designator is to provide the associated advice code with access to the parameter orginally passed to the methods. By including a single parameter type, here `String`, in the method signature, only methods with a single parameter should be considered. The order of the object and the parameter in both the pintcut and the advice definitions must be the same between the two definitions.

Argument list `files.lst`

```
MyAspectC.java
MyAspectA.java
MyAspectB.java
MyBaseProg.java
GlobalValue.java
```

Aspect `MyAspectA`

```
/**
 *  "Sequence of aspects"
 *
 *@since      10-Jun-2003
 *@author     Hinrich Bonin
 *@version    1.0
 */
package de.fhnon.nemo.priority;

public aspect MyAspectA
{
   pointcut callSet(MyBaseProg p, String s):
     target(p)&& args(s) && call(void set*(..));

   before(MyBaseProg p, String s): callSet(p, s)
   {
      System.out.println(
        "MyAspectA before(): args=" +
        s +  " slot=" + p.getSlot());
   }


   void around(MyBaseProg p, String s): callSet(p, s)
   {
      proceed(p,s);
      p.slot = value;
      System.out.println(
        "MyAspectA around(): args=" +
        s + " slot=" + p.getSlot());
   }


   after(MyBaseProg p, String s): callSet(p, s)
   {
      System.out.println(
        "MyAspectA after() : args=" +
        s + " slot=" + p.getSlot());
   }
}
```

Aspect `MyAspectB`

```
/**
 *  "Sequence of aspects"
 *
```

```
 *@since      10-Jun-2003
 *@author     Hinrich Bonin
 *@version    1.0
 */
package de.fhnon.nemo.priority;

public aspect MyAspectB
{

   pointcut callSet(MyBaseProg p, String s):
     target(p) && args(s) && call(void set*(..));

   before(MyBaseProg p, String s): callSet(p, s)
   {
      System.out.println(
        "MyAspectB before(): args=" +
        s + " slot=" + p.getSlot());
   }


   void around(MyBaseProg p, String s): callSet(p, s)
   {
      proceed(p,s);
      p.slot = value;
      System.out.println(
        "MyAspectB around(): args=" +
        s + " slot=" + p.getSlot());
   }


   after(MyBaseProg p, String s): callSet(p, s)
   {
      System.out.println(
        "MyAspectB after() : args=" +
        s + " slot=" + p.getSlot());
   }
}
```

Aspect `MyAspectC`

```
/**
 *  "Sequence of aspects"
 *
 *@since      10-Jun-2003
 *@author     Hinrich Bonin
 *@version    1.0
```

```
 */
package de.fhnon.nemo.priority;

public aspect MyAspectC
{


   pointcut callSet(MyBaseProg p, String s):
     target(p) && args(s) && call(void set*(..));

   before(MyBaseProg p, String s): callSet(p, s)
   {
      System.out.println(
        "MyAspectC before(): args=" +
        s + " slot=" + p.getSlot());
   }


   void around(MyBaseProg p, String s): callSet(p, s)
   {
      proceed(p,s);
      p.slot = value;
      System.out.println(
        "MyAspectC around(): args=" +
        s + " slot=" + p.getSlot());
   }


   after(MyBaseProg p, String s): callSet(p, s)
   {
      System.out.println(
        "MyAspectC after() : args=" +
        s + " slot=" + p.getSlot());
   }
}
```

Class `MyBaseProg`

```
/**
 *  "Sequence of aspects"
 *
 *@since      10-Jun-2003
 *@author     Hinrich Bonin
 *@version    1.0
 */
package de.fhnon.nemo.priority;
```

```
public class MyBaseProg
{
   String slot = "default";


   String getSlot()
   {
      return slot;
   }


   void setSlot(String slot)
   {
      this.setNothing();
      this.setFoo("-foo-");
      this.slot = slot;
      System.out.println(
            "MyBaseProg: done setSlot()");
   }


   void setFoo(Object o)
   {
      System.out.println(
            "MyBaseProg: done setFoo()");
   }


   void setNothing()
   {
      System.out.println(
            "MyBaseProg: done setNothing()");
   }


   public static void main(String[] args)
   {
      MyBaseProg myP = new MyBaseProg();
      System.out.println(
            "MyBaseProg: slot=" + myP.getSlot());
      myP.setSlot("start");
      System.out.println(
            "MyBaseProg: slot=" + myP.getSlot());
   }
}
```

Aspect `GlobalValue`

```
/**
 *  "Sequence of aspects"
 *
 *@since      10-Jun-2003
 *@author     Hinrich Bonin
 *@version    1.0
 */
package de.fhnon.nemo.priority;

public aspect GlobalValue
{
  static final String  MyAspectA.value = "A";
  static final String  MyAspectB.value = "B";
  static final String  MyAspectC.value = "C";


  pointcut application() :  call(* main(..));

  before() : application()
  {
   System.out.println("Start of the Java Application");
  }


  after() : application()
  {
   System.out.println("End of the Java Application");
  }
}
```

Protocol `MyBaseProg.log`

```
C:\bonin\aosd\code>ajc -version
ajc version 1.0.6
  (built 24.07.2002 18:21 PST) running on java 1.4.0_01

C:\bonin\aosd\code>ajc -argfile de/fhnon/nemo/priority/files.lst

C:\bonin\aosd\code>java de/fhnon/nemo/priority/MyBaseProg
Start of the Java Application
MyBaseProg: slot=default
MyAspectA before(): args=start slot=default
```

```
MyAspectB before(): args=start slot=default
MyAspectC before(): args=start slot=default
MyBaseProg: done setNothing()
MyAspectA before(): args=-foo- slot=default
MyAspectB before(): args=-foo- slot=default
MyAspectC before(): args=-foo- slot=default
MyBaseProg: done setFoo()
MyAspectC around(): args=-foo- slot=C
MyAspectC after() : args=-foo- slot=C
MyAspectB around(): args=-foo- slot=B
MyAspectB after() : args=-foo- slot=B
MyAspectA around(): args=-foo- slot=A
MyAspectA after() : args=-foo- slot=A
MyBaseProg: done setSlot()
MyAspectC around(): args=start slot=C
MyAspectC after() : args=start slot=C
MyAspectB around(): args=start slot=B
MyAspectB after() : args=start slot=B
MyAspectA around(): args=start slot=A
MyAspectA after() : args=start slot=A
MyBaseProg: slot=A
End of the Java Application

C:\bonin\aosd\code>
```

7.6 Roundoff Problem

“Roundoff errors are a fact of life when calculating with floating-point numbers.” (↪ [Horstmann, 2000] p. 62.) The following class `Roundoff` shows roundoff errors with the casting to tpye `int`. The `static` method `Math.round()` helps to avoid such errors.

Argument list `files.lst`

```
Roundoff.java
Value.java
```

Class `Roundoff`

```
/**
 *  "Roundoff Problem" application
 *
 *@author     Bonin
 *@version    1.0
 */
```

```
package conversion;

import java.text.NumberFormat;

public class Roundoff
{
   public static void main(String[] args)
   {
      double g = Double.parseDouble(S);

      double f = 100 * g;
      int n = (int) (100 * g);
      int m = (int) Math.round(100 * g);

      NumberFormat formatter =
            NumberFormat.getNumberInstance();
      formatter.setMinimumFractionDigits(D);

      System.out.println("1. S = " + S);
      System.out.println("2. g = " + g);
      System.out.println("3. f = " + f);
      System.out.println("4. f = " + formatter.format(f));
      System.out.println("5. n = " + n);
      System.out.println("6. m = " + m);
   }
}
```

Aspect `Value`

```
/**
 * "Roundoff Problem" application
 * @author Bonin
 * @version 1.0
 */
package conversion;

aspect Value {
    // Example value as a string
    final static String Roundoff.S = "4.12";

    // Number of digits allowed in
    // a fraction protion of a number
```

```
    final static int Roundoff.D  = 2;
}
```

Protocol `Roundoff.log`

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.3
    (built 08.02.2002 12:47 PST) running on java 1.3.1

D:\bonin\aosd\code>ajc -argfile conversion/files.lst

D:\bonin\aosd\code>java conversion.Roundoff
1. S = 4.35
2. g = 4.35
3. f = 434.99999999999994
4. f = 435,00
5. n = 434
6. m = 435

D:\bonin\aosd\code>
```

The protocol shows the fault `n = 434` (5the step). To convert any floating-point number into a binary, convert the whole part and the fractional part separately. For example, 4.35 is 100.01 0110 0110 0110 . . .. More explanation see for example ↪ [Horstmann, 2000] p. 64.

7.7 Move Rectangle

A rectangle is described by the *x*- and *y*-coordinates of its top left corner, its width, and its height. The method `translate` moves a rectangle by a certain distance in the *x*- and *y*-direction. For example,

```
foo.translate(5,10);
```

moves the rectangle be 5 units in the *x*-direction and 10 units in the *y*-direction.
Notice: This exercise includes two classes `Rectangle`, one from the standard package `java.awt` and one self defined class in the user package `rectangle`.

Argument list `files.lst`

```
Rectangle.java
Box.java
Factor.java
```

Class `Rectangle`

```
/**
 *  "Box" application
 *
 *@author     Bonin
 *@version    1.1
 */
package rectangle;

public class Rectangle extends java.awt.Rectangle
{
   public Rectangle(int x, int y, int width, int height)
   {
      super(x, y, width, height);
   }
}
```

Class `Box` Here is called the method `toString()` to display an object of the class `Box`. Normally a rectangle is used in a graphical context like an applet (for example ↪ Section 7.10,p. 148).

```
/**
 *  "Box" application
 *
 *@author     Bonin
 *@version    1.0
 */
package rectangle;
public class Box extends Rectangle
{

   public static void main(String[] args)
   {
      try
      {
         int x = Integer.parseInt(args[0]);
         int y = Integer.parseInt(args[1]);
         int width = Integer.parseInt(args[2]);
         int height = Integer.parseInt(args[3]);

         int delta = 5;
```

```
         Box boxA = new Box(x, y, width, height);

         Box boxB = (Box) boxA.clone();

         boxB.translate(delta, delta);

         Object boxC = (Object)
               boxA.intersection(boxB);

         System.out.println("boxA:" + boxA);
         System.out.println("boxB:" + boxB);
         System.out.println("boxC:" + boxC);
      } catch (ArrayIndexOutOfBoundsException e)
      {
         System.out.println(
               "You must specify four arguments:\n" + e);
         System.out.println(
               "Usage: java Box <x y width height>");
      } catch (NumberFormatException e)
      {
         System.out.println(
               "The argument you specify must be an integer:\n"
                + e);
      }
   }


   public Box(int x, int y, int width, int height)
   {
      super(x, y, width, height);
   }
}
```

Aspect `Factor` This aspect multiplies the *x*- and *y*-coordinates of a created rectangle with the value of a new slot `multiplier`. The rectangle gets a new top left corner, so it moves.

```
/**
 * "Box" application
 *@author Bonin
 *@version 1.0
```

```
 */
package rectangle;
aspect Factor
{
    private int Box.mulitiplier = 2;
    public int Box.getMultiplier() { return mulitiplier; }

    pointcut applyNewBox(Box foo) :
        target(foo) && within(Box) &&  execution( new (..));

    after(Box foo) : applyNewBox(foo)
        {
            foo.translate((int) foo.getX() * foo.getMultiplier(),
                (int)foo.getY() *  foo.getMultiplier());
        }
}
```

Protocol `Box.log`

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.3
    (built 08.02.2002 12:47 PST) running on java 1.3.1

D:\bonin\aosd\code>ajc -argfile rectangle/files.lst

D:\bonin\aosd\code>java rectangle.Box 1 2
You must specify four arguments:
java.lang.ArrayIndexOutOfBoundsException
Usage: java Box <x y width height>

D:\bonin\aosd\code>java rectangle.Box 1 2 10 20
boxA:rectangle.Box[x=3,y=6,width=10,height=20]
boxB:rectangle.Box[x=8,y=11,width=10,height=20]
boxC:java.awt.Rectangle[x=8,y=11,width=5,height=15]

D:\bonin\aosd\code>
```

7.8 Read Console Input

The exercise shows the `main()` method in the aspect `Bike` (↪ p.144). The class `ReadInput` turning `System.in` into a `BufferedReader` object. The class `BufferedReader` can read lines. These lines are strings. To convert a `String` object into a number (`int` type) we use the method `Integer.parseInt()`. The converting throws `NumberFormatException`. This exception stops the application with `System.exit(2)`. A job management could use the value 2 to control the next step.

Argument list `files.lst`

```
Bike.java
ReadInput.java
```

Class `ReadInput`

```
/**
 *  "Read console input" application
 *
 *@author     Bonin
 *@version    1.0
 */
package console;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

public class ReadInput
{
   public static int integer()
   {
      int value = 0;
      try
      {
         BufferedReader reader = new BufferedReader(
               new InputStreamReader(System.in));
         value = Integer.parseInt(reader.readLine());
      } catch (IOException e)
      {
         System.out.println("Read error:\n" + e);
         System.exit(1);
      } catch (NumberFormatException e)
      {
         System.out.println(
               "The input must be an integer:\n" + e);
         System.exit(2);
      }
```

```
        return value;
    }
}
```

Aspect `Bike`

```
/**
 * "Read console input" application
 *@author Bonin
 *@version 1.0
 */
package console;

public aspect Bike
{
    final static int FAN = 3;

    public static void main(String[] args)
    {
        int bikes = 0;

        System.out.println(
            "How many bikes do you have?");

        bikes = ReadInput.integer();

        if (bikes >= FAN)
            System.out.println(
                bikes + " bike(s) --- Happy bike fan!");
        else
            System.out.println(
                bikes + " bike(s) --- Not enough!");
    }
}
```

Auxiliary File `myBikeQuantity.txt`

```
3
```

Protocol `Bike.log`

```
D:\bonin\aosd\code>ajc -version
```

```
ajc version 1.0.3
    (built 08.02.2002 12:47 PST) running on java 1.3.1

D:\bonin\aosd\code>ajc -argfile console/files.lst

D:\bonin\aosd\code>java console.Bike
How many bikes do you have?
1.5
The input must be an integer:
java.lang.NumberFormatException: 1.5

D:\bonin\aosd\code>java console.Bike
How many bikes do you have?
2
2 bike(s) --- Not enough!

D:\bonin\aosd\code>java console.Bike
                     < console/myBikeQuantity.txt
How many bikes do you have?
3 bike(s) --- Happy bike fan!

D:\bonin\aosd\code>
```

7.9 Play of `this`

The use of the `this` keyword is a little confusing. Normally, `this` denotes a reference to the instance, but if `this` is following by parentheses, it denotes a call to another constructor of this class.

Argument list `files.lst`

```
Foo.java
Bar.java
```

Class `Foo`

```
/**
 *  "Play of <code>this</code>"
 *
 *@author     Bonin
 *@version    1.0
 */
package mythis;
```

```
public class Foo
{
   Foo(int i, double j)
   {
      this.setI(i);
      this.setJ(j);
   }


   Foo(int i)
   {
      this(i, J);
   }


   Foo(double j)
   {
      this(I, j);
   }


   Foo()
   {
      this(I, J);
   }


   public static void main(String[] args)
   {
      Foo f = new Foo(5E-1);
      System.out.println(
            f.m(4) + "\n" +
            f.m(8.8));
   }
}
```

Aspect `Bar`

```
/**
 * "Play of <code>this</code>"
 *@author Bonin
```

```
 *@version 1.0
 */
package mythis;

public aspect Bar
{

  static final int Foo.I = 1;
  static final double Foo.J = 7E2;

  private int Foo.i;
  private double Foo.j;

  public int Foo.getI() {
    return this.i;
  }

  public double Foo.getJ() {
    return this.j;
  }

  public void Foo.setI(int i) {
    this.i = i * I;
  }

  public void Foo.setJ(double j) {
    this.j = j * J;
  }
  public String Foo.m(int i) {
    return "int type: " + i;
  }

  public String Foo.m(double j) {
    return "double type: " + this.j;
  }
}
```

Protocol `Foo.log`

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.3
    (built 08.02.2002 12:47 PST) running on java 1.3.1
```

```
D:\bonin\aosd\code>ajc -argfile mythis/files.lst

D:\bonin\aosd\code>java mythis.Foo
int type: 4
double type: 350.0

D:\bonin\aosd\code>
```

7.10 Applet: Text & Font & Color

An applet is a program that runs inside a web browser. Therefore the applet is embedded in the loaded web page. For the embedding we use the XHTML-tag `<object>` (↪ p. 152). There are a lot of trouble with the embedding and the Java engine by many browsers. For example we got errors with *Opera 5.12* and *Microsoft Internet Explorer 6.0.2600*. That is why we show the results only with *Netscape 6: Mozilla/5.0 (Windows; U; Windows NT 5.0; de-DE; m18) Gecko/20010131 Netscape6/6.01*.

Argument list `files.lst`

```
Text.java
GlobalValue.java
```

Class `Text` An object of the class `FontRenderContext` knows how to transform letter shapes (which are described as curves) into pixels. The `TextLayout` object gets typographic measurements of the string TEXT (↪ figure 7.1, p. 151).

```
/**
 *  Applet example
 *
 *@author     Bonin
 *@version    1.0
 */
package graphic;

import java.applet.Applet;
import java.awt.Color;
import java.awt.Font;
import java.awt.font.FontRenderContext;
import java.awt.font.TextLayout;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Rectangle2D;
import java.awt.Graphics;
```

```
import java.awt.Graphics2D;
import java.util.Random;
import javax.swing.JOptionPane;

public class Text extends Applet
{
   private Color color1;
   private Color color2;
   private Font largeFont;
   private String input;


   public void init()
   {

      float r1;

      float g1;

      float b1;
      float r2;
      float g2;
      float b2;
      Random generator = new Random();

      /*
       *  Applet Parameters
       *  <param name="..." value="..." />
       */
      r1 = Float.parseFloat(getParameter("red"));
      g1 = Float.parseFloat(getParameter("green"));
      b1 = Float.parseFloat(getParameter("blue"));
      color1 = new Color(r1, g1, b1);

      /*
       *  Second color generated randomly.
       *  nextDouble() returns a random floating-point
       *  number between 0 (inclusive) and 1 (exclusive).
       */
      r2 = (float) generator.nextDouble();
      g2 = (float) generator.nextDouble();
      b2 = (float) generator.nextDouble();
```

```
      color2 = new Color(r2, g2, b2);

      /*
       *  Modal dialog (Swing toolkit)
       *  Waits until the user has enterd a string
       *  and clicks on the "OK" button.
       */
      input = JOptionPane.showInputDialog(
            "Your text?");
      if (input.length() == 0)
      {
         input = "No input!";
      }

      largeFont = new Font(
            "Courier", Font.ITALIC, LARGE_SIZE);
   }


   public void paint(Graphics g)
   {

      Graphics2D g2 = (Graphics2D) g;

      /*
       *  The font render context is an object that knows how
       *  to transform letter shapes (which are described as
       *  curves) into pixels.
       */
      FontRenderContext context = g2.getFontRenderContext();

      /*
       *  The TextLayout object gets typographic measurements
       *  of the string TEXT.
       */
      TextLayout layout =
            new TextLayout(input, largeFont, context);

      float xTextWidth = layout.getAdvance();
      float yTextHeight =
            layout.getAscent() + layout.getDescent();
      float xLeft = (getWidth() - xTextWidth) * 0.5F;
```

Legende:
Description of `getAscent()`, `getDescent()`, and `getAdvance()`

Figure 7.1: TextLayout: Ascent, Descent, Leading and Advance

```
        float yTop = (getHeight() - yTextHeight) * 0.5F;
        float yBase = yTop + layout.getAscent();

        Ellipse2D.Float egg =
              new Ellipse2D.Float(
              xLeft, yTop, xTextWidth, yTextHeight);
        Rectangle2D.Float box =
              new Rectangle2D.Float(
              xLeft, yTop, xTextWidth, yTextHeight);

        g2.setColor(color1);
        g2.fill(egg);
        g2.setColor(color2);
        g2.draw(box);

        g2.setFont(largeFont);
        g2.drawString(input, xLeft, yBase);
    }
}
```

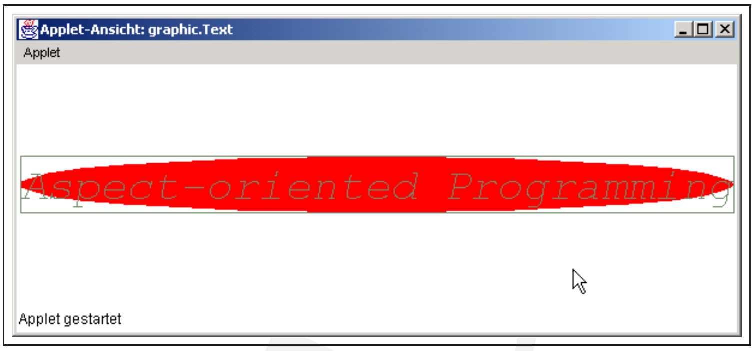
Aspect `GlobalValue`

```
/**
 * Applet example
 *@author Bonin
 *@version 1.0
 */
package graphic;

public aspect GlobalValue
{
  final int Text.LARGE_SIZE = 36;
}
```

File to run the Applet with `<object>`**-Element** `Text.html`

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Bonin Version 1.0  -->
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<meta http-equiv="Content-Type"
  content="text/html; charset=utf-8" />
<title>Little Applet</title>
</head>
<body>
<h1>Your Text:</h1>
<object
  codetype="application/java"
  classid="java:graphic.Text.class"
  code="graphic.Text"
  width="600" height="200"
  alt="Java: Just A Valid Application">
  <param name="red" value="1.0" />
  <param name="green" value="0.0" />
  <param name="blue" value="0.0" />
</object>
<p>Copyright bonin@fhnon.de</p>
</body>
</html>
```

Protocol `Text.log`



Legende:
Modal dialog: Aspect-oriented Programming

Figure 7.2: *Your-Text*-Applet with `appletviewer`

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.3
    (built 08.02.2002 12:47 PST) running on java 1.3.1

D:\bonin\aosd\code>ajc -argfile graphic/files.lst

D:\bonin\aosd\code>appletviewer Text.html

D:\bonin\aosd\code>
```

Result with *appletviewer* ↪ figure 7.2, p. 153. Result with Browser *Netscape 6*[2] ↪ figure 7.3, p. 154.

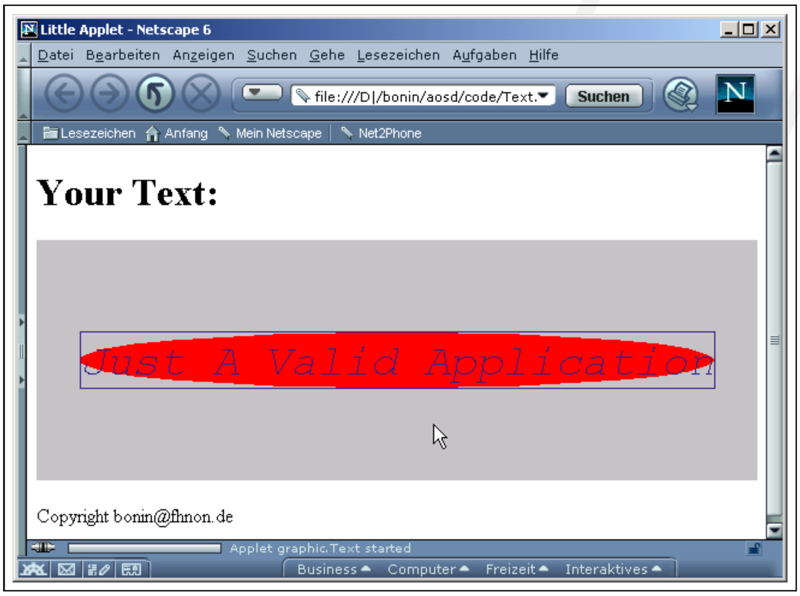
7.11 Applet: Listener & Inner Class

The method of the inner class are allowed to access the private instance variables and methods of the outer class. Usually, the event methods need to access the varaibles in an other class. So, it is good practice to define a listener as an inner class.

Argument list `files.lst`

```
Square.java
Draw.java
```

[2]Netscape 6: Mozilla/5.0 (Windows; U; Windows NT 5.0; de-DE; m18) Gecko/20010131 Netscape6/6.01



Legende:
Modal dialog: Just A Valid Application (≡ Java)

Figure 7.3: *Your-Text*-Applet with *Netscape 6*

Class `Square`

```
/**
 *  Inner Class example
 *
 *@author     Bonin
 *@version    1.0
 */
package innerclass;

import java.applet.Applet;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.geom.Rectangle2D;
import java.awt.Graphics;
import java.awt.Graphics2D;

public class Square extends Applet
{
   private final int LENGTH = 20;
   protected Rectangle2D.Double square;


   public Square()
   {
      square = new Rectangle2D.Double(
            0, 0, LENGTH, LENGTH);
      addMouseListener(new MouseClickListener());
   }


   // Inner class definition
   private class MouseClickListener
          extends MouseAdapter
   {
      public void mouseClicked(MouseEvent event)
      {
         int mouseX = event.getX();
         int mouseY = event.getY();
         square.setFrame(
               mouseX - LENGTH / 2, mouseY - LENGTH / 2,
               LENGTH, LENGTH);
         repaint();
```

```
      }
   }
}
```

Aspect `Draw`

```
/**
 * Inner Class example
 *@author Bonin
 *@version 1.0
 */
package innerclass;
import java.awt.Graphics;
import java.awt.Graphics2D;

public aspect Draw
{
    public void Square.paint(Graphics g) {
      Graphics2D g2 = (Graphics2D) g;
      g2.draw(square);
    }
}
```

File to run the Applet with `<object>`**-Element** `Square.html`

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Bonin Version 1.0  -->
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<meta http-equiv="Content-Type"
  content="text/html; charset=utf-8" />
<title>Move the Square</title>
</head>
<body>
<h1>Move the Square</h1>
<object
  codetype="application/java"
  classid="java:innerclass.Square.class"
  code="innerclass.Square"
  width="600" height="200"
```

```
  alt="Move the square">
</object>
<p>Copyright bonin@fhnon.de</p>
</body>
</html>
```

Protocol `Square.log`

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.3
    (built 08.02.2002 12:47 PST) running on java 1.3.1

D:\bonin\aosd\code>ajc -argfile innerclass/files.lst

D:\bonin\aosd\code>cd innerclass

D:\bonin\aosd\code\innerclass>dir
           574 Draw.class
           287 Draw.java
            24 files.lst
         1.091 Square$MouseClickListener.class
         1.128 Square.class
           858 Square.java

D:\bonin\aosd\code\innerclass>cd ..

D:\bonin\aosd\code>appletviewer Square.html

D:\bonin\aosd\code>
```

Result with Browser *Netscape 6*[3] ↪ figure 7.4, p. 158.

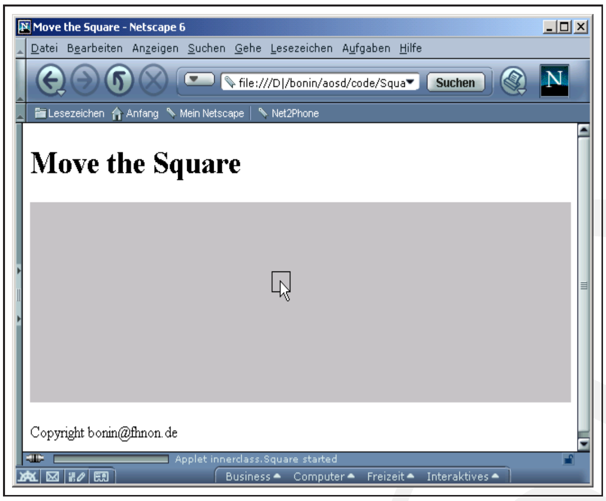
7.12 Pulldown Menus with `javax.swing.JMenu`

A menu is a collection of *menu items* and more menus. When the user selects a menu item, the menu item sends an action event. Therefore, we add a listener to each menu item:

myMenuItem`.addActionListener(`myListener`)`

We add action listeners only to menu items, not to menus or the menu bar. When the user clicks on a menu name and a submenu opens, no action event is sent (↪ protocol 7.12 p. 165).

[3]Netscape 6: Mozilla/5.0 (Windows; U; Windows NT 5.0; de-DE; m18) Gecko/20010131 Netscape6/6.01



Legende:
Click the mouse to move the square.

Figure 7.4: Mouse event Applet with *Netscape 6*

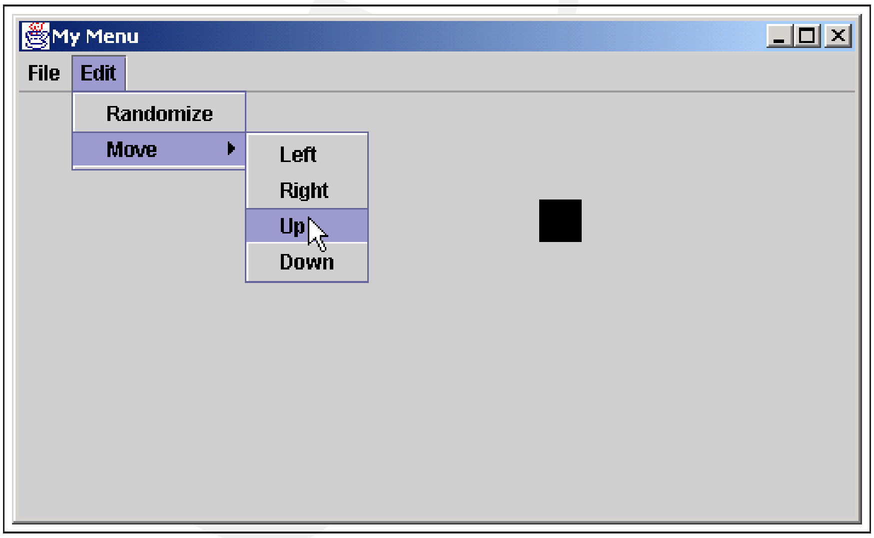
Notice: The idea of this example "Menu bar" is taken from ↪ [Horstmann, 2000], p. 499–503. The code is modified and two aspects are integrated.

Argument list `files.lst`

```
MyMenu.java
MyMenuFrame.java
SquarePanel.java
GlobalValue.java
ActionControl.java
```

Class `MyMenu`

```
/**
 *   "My Menu" application
 *
 *@author     Bonin
 *@version    1.0
 */
package swing;
```



Legende:
Idea ↪ [Horstmann, 2000], p. 499–503.

Figure 7.5: Pulldown Menus

```
public class MyMenu
{
   public static void main(String[] args)
   {
      MyMenuFrame frame = new MyMenuFrame();
      frame.setTitle("My Menu");
      frame.show();
   }
}
```

Class `MyMenuFrame`

```
/**
 *  "My Menu" application
 *
 *@author     Bonin
 *@version    1.0
 */
package swing;

import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;

public class MyMenuFrame extends JFrame
{
   private JMenuItem downMenuItem;
   private JMenuItem exitMenuItem;
   private JMenuItem leftMenuItem;
   private JMenuItem newMenuItem;
   private SquarePanel panel;
   private JMenuItem randomizeMenuItem;
   private JMenuItem rightMenuItem;
   private JMenuItem upMenuItem;
```

```
public MyMenuFrame()
{

   setSize(FRAME_WIDTH, FRAME_HEIGHT);

   addWindowListener(new WindowCloser());

   panel = new SquarePanel();
   Container contentPane = getContentPane();
   contentPane.add(panel, "Center");

   JMenuBar menuBar = new JMenuBar();
   setJMenuBar(menuBar);

   JMenu fileMenu = new JMenu("File");
   menuBar.add(fileMenu);

   MenuListener listener = new MenuListener();

   newMenuItem = new JMenuItem("New");
   fileMenu.add(newMenuItem);
   newMenuItem.addActionListener(listener);

   exitMenuItem = new JMenuItem("Exit");
   fileMenu.add(exitMenuItem);
   exitMenuItem.addActionListener(listener);

   JMenu editMenu = new JMenu("Edit");
   menuBar.add(editMenu);

   randomizeMenuItem = new JMenuItem("Randomize");
   editMenu.add(randomizeMenuItem);
   randomizeMenuItem.addActionListener(listener);

   JMenu moveMenu = new JMenu("Move");
   editMenu.add(moveMenu);

   leftMenuItem = new JMenuItem("Left");
   moveMenu.add(leftMenuItem);
   leftMenuItem.addActionListener(listener);

   rightMenuItem = new JMenuItem("Right");
```

```
      moveMenu.add(rightMenuItem);
      rightMenuItem.addActionListener(listener);

      upMenuItem = new JMenuItem("Up");
      moveMenu.add(upMenuItem);
      upMenuItem.addActionListener(listener);

      downMenuItem = new JMenuItem("Down");
      moveMenu.add(downMenuItem);
      downMenuItem.addActionListener(listener);
   }


   private class MenuListener implements ActionListener
   {
      public void actionPerformed(ActionEvent event)
      {
         Object source = event.getSource();

         if (source == exitMenuItem)
         {
            System.exit(0);
         } else if (source == newMenuItem)
         {
            panel.reset();
         } else if (source == upMenuItem)
         {
            panel.moveSquare(0, -1);
         } else if (source == downMenuItem)
         {
            panel.moveSquare(0, 1);
         } else if (source == leftMenuItem)
         {
            panel.moveSquare(-1, 0);
         } else if (source == rightMenuItem)
         {
            panel.moveSquare(1, 0);
         } else if (source == randomizeMenuItem)
         {
            panel.randomize();
         }
      }
```

```
   }


   private class WindowCloser extends WindowAdapter
   {
      public void windowClosing(WindowEvent event)
      {
         System.out.println("Good bye!");
         System.exit(0);
      }
   }
}
```

Class SquarePanel

```
/**
 *  "My Menu" application
 *
 *@author     Bonin
 *@version    1.0
 */
package swing;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.util.Random;
import javax.swing.JPanel;

public class SquarePanel extends JPanel
{
   private Rectangle square;


   public SquarePanel()
   {
      square = new Rectangle(
            0, 0, SQUARE_WIDTH, SQUARE_WIDTH);
   }


   public void paintComponent(Graphics g)
```

```
   {
      super.paintComponent(g);
      Graphics2D g2 = (Graphics2D) g;
      g2.fill(square);
   }


   public void reset()
   {
      square.setLocation(0, 0);
      repaint();
   }


   public void randomize()
   {
      Random generator = new Random();
      square.setLocation(
            generator.nextInt(getWidth()),
            generator.nextInt(getWidth()));
      repaint();
   }


   /**
    *  The square is moved by muliples of its full "width".
    *
    *@param  dx  Description of the Parameter
    *@param  dy  Description of the Parameter
    */
   public void moveSquare(int dx, int dy)
   {
      square.translate(
            dx * SQUARE_WIDTH, dy * SQUARE_WIDTH);
      repaint();
   }
}
```

Aspect `GlobalValue`

```
/**
 * "My Menu" application
```

```
 *@author Bonin
 *@version 1.0
 */
package swing;

public aspect GlobalValue
{
  int MyMenuFrame.FRAME_WIDTH = 500;
  int MyMenuFrame.FRAME_HEIGHT = 300;
  static int SquarePanel.SQUARE_WIDTH = 25;
}
```

Aspect `ActionControl`

```
/**
 * "My Menu" application
 *@author Bonin
 *@version 1.0
 */
package swing;

import java.awt.event.ActionEvent;
import javax.swing.JMenuItem;

public aspect ActionControl
{

    pointcut applyActionPerformed(ActionEvent e):
         args(e) &&
         call(void actionPerformed(ActionEvent));

    before(ActionEvent e) :
        applyActionPerformed(e)
        {
           JMenuItem mItem =
               (javax.swing.JMenuItem) e.getSource();
           System.out.println(
               "Action performed: " + mItem.getText());
        }
}
```

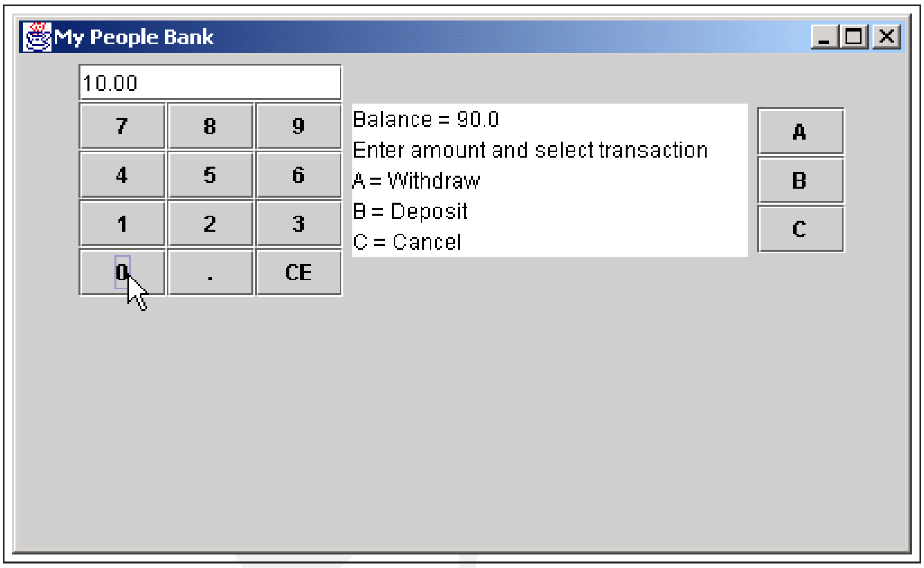
Protocol `MyMenu.log`

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.3
   (built 08.02.2002 12:47 PST) running on java 1.3.1

D:\bonin\aosd\code>ajc -argfile swing/files.lst

D:\bonin\aosd\code>java swing.MyMenu
Action performed: Randomize
Action performed: Up
Action performed: Right
Action performed: Up
Action performed: Exit

D:\bonin\aosd\code>java swing.MyMenu
Action performed: Randomize
Good bye!

D:\bonin\aosd\code>cd swing

D:\bonin\aosd\code\swing>dir
    1.300 ActionControl.class
      581 ActionControl.java
       87 files.lst
      602 GlobalValue.class
      243 GlobalValue.java
      627 MyMenu.class
      255 MyMenu.java
      675 MyMenu.log
    1.702 MyMenuFrame$MenuListener.class
      920 MyMenuFrame$WindowCloser.class
    2.500 MyMenuFrame.class
    3.246 MyMenuFrame.java
    1.626 SquarePanel.class
    1.056 SquarePanel.java
D:\bonin\aosd\code\swing>
```

Result ↪ figure 7.5, p. 159.

7.13 Example: My People Bank

The simulation of an *automatic teller maschine* (ATM) has a keypad to enter numbers, a display to show messages, and a set of buttons, labeled A, B, and C, whose function depens on the state of the maschine (↪ figure 7.6 ,p. 167).



Legende:
An Automatic Teller Maschine Simulation — Idea ↪ [Horstmann, 2000], p. 587–605

Figure 7.6: My People Bank

Notice: The idea of this example "An Automatic Teller Maschine Simulation" is taken from ↪ [Horstmann, 2000], p. 587–605. The code is modified and two aspects are integrated.

Argument list `files.lst`

```
MyATM.java
MyATMFrame.java
MyKeyPad.java
Bank.java
BankAccount.java
Customer.java
GlobalValue.java
TransactionControl.java
```

Class `MyATM`

```
/**
```

```
 *  "My People Bank" application ATM = Automatic Teller Machine
 *
 *@author     Bonin
 *@version    1.0
 */
package bank;

public class MyATM
{
   public static void main(String[] args)
   {
      MyATMFrame frame = new MyATMFrame();
      frame.setTitle("My People Bank");
      frame.show();
   }
}
```

Class `MyATMFrame`

```
/**
 *  "My People Bank" application ATM = Automatic Teller Machine
 *
 *@author     Bonin
 *@version    1.0
 */
package bank;

import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.io.IOException;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextArea;

public class MyATMFrame extends JFrame
```

```
{
   private final static int START_STATE = 1;
   private final static int PIN_STATE = 2;
   private final static int ACCOUNT_STATE = 3;
   private final static int TRANSACT_STATE = 4;
   private int state;

   private int customerNumber;
   private Customer currentCustomer;

   private Bank theBank;
   private BankAccount currentAccount;

   private JButton aButton;
   private JButton bButton;
   private JButton cButton;

   private MyKeyPad pad;
   private JTextArea display;


   public MyATMFrame()
   {
      setSize(FRAME_WIDTH, FRAME_HEIGHT);

      addWindowListener(new WindowCloser());

      theBank = new Bank();
      try
      {
         theBank.readCustomers(CUSTOMERS_DB);
      } catch (IOException e)
      {
         JOptionPane.showMessageDialog
               (null, "Error opening CUSTOMERS_DB.");
      }

      pad = new MyKeyPad();

      display = new JTextArea(4, 20);

      aButton = new JButton(" A ");
```

```
      aButton.addActionListener(new AButtonListener());
      bButton = new JButton(" B ");
      bButton.addActionListener(new BButtonListener());
      cButton = new JButton(" C ");
      cButton.addActionListener(new CButtonListener());

      JPanel buttonPanel = new JPanel();
      buttonPanel.setLayout(new GridLayout(3, 1));
      buttonPanel.add(aButton);
      buttonPanel.add(bButton);
      buttonPanel.add(cButton);

      Container contentPane = getContentPane();
      contentPane.setLayout(new FlowLayout());
      contentPane.add(pad);
      contentPane.add(display);
      contentPane.add(buttonPanel);

      setState(START_STATE);
   }


   public void setCustomerNumber()
   {
      customerNumber = (int) pad.getValue();
      setState(PIN_STATE);
   }


   public void selectCustomer()
   {
      int pin = (int) pad.getValue();
      currentCustomer =
            theBank.findCustomer(customerNumber, pin);
      if (currentCustomer == null)
      {
         setState(START_STATE);
      } else
      {
         setState(ACCOUNT_STATE);
      }
   }
```

```
public void selectAccount(int account)
{
   currentAccount =
         currentCustomer.getAccount(account);
   setState(TRANSACT_STATE);
}


public void withdraw()
{
   currentAccount.withdraw(pad.getValue());
   setState(ACCOUNT_STATE);
}


public void deposit()
{
   currentAccount.deposit(pad.getValue());
   setState(ACCOUNT_STATE);
}


public void setState(int newState)
{
   state = newState;
   pad.clear();
   if (state == START_STATE)
   {
      display.setText("Enter customer number\nA = OK");
   } else if (state == PIN_STATE)
   {
      display.setText("Enter PIN\nA = OK");
   } else if (state == ACCOUNT_STATE)
   {
      display.setText("Select Account\n" +
            "A = Checking\nB = Savings\nC = Exit");
   } else if (state == TRANSACT_STATE)
   {
      display.setText("Balance = " +
            currentAccount.getBalance() +
```

```
            "\nEnter amount and select transaction\n" +
            "A = Withdraw\nB = Deposit\nC = Cancel");
   }
}


private class AButtonListener
       implements ActionListener
{
   public void actionPerformed(ActionEvent event)
   {
      if (state == START_STATE)
      {
         setCustomerNumber();
      } else if (state == PIN_STATE)
      {
         selectCustomer();
      } else if (state == ACCOUNT_STATE)
      {
         selectAccount(Customer.CHECKING_ACCOUNT);
      } else if (state == TRANSACT_STATE)
      {
         withdraw();
      }
   }
}


private class BButtonListener
       implements ActionListener
{
   public void actionPerformed(ActionEvent event)
   {
      if (state == ACCOUNT_STATE)
      {
         selectAccount(Customer.SAVINGS_ACCOUNT);
      } else if (state == TRANSACT_STATE)
      {
         deposit();
      }
   }
}
```

```
   private class CButtonListener
          implements ActionListener
   {
      public void actionPerformed(ActionEvent event)
      {
         if (state == ACCOUNT_STATE)
         {
            setState(START_STATE);
         } else if (state == TRANSACT_STATE)
         {
            setState(ACCOUNT_STATE);
         }
      }
   }


   private class WindowCloser extends WindowAdapter
   {
      public void windowClosing(WindowEvent event)
      {
         System.out.println("Good bye!");
         System.exit(0);
      }
   }
}
```

Class `MyKeyPad`

```
/**
 *  "My People Bank" application
 *
 *@author     Bonin
 *@version    1.0
 */
package bank;

import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

```
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class MyKeyPad extends JPanel
{
   private JPanel buttonPanel;
   private JButton clearButton;
   private JTextField display;


   public MyKeyPad()
   {
      setLayout(new BorderLayout());
      display = new JTextField();
      add(display, "North");

      buttonPanel = new JPanel();
      buttonPanel.setLayout(new GridLayout(4, 3));

      ActionListener listener =
            new DigitButtonListener();
      addButton("7", listener);
      addButton("8", listener);
      addButton("9", listener);
      addButton("4", listener);
      addButton("5", listener);
      addButton("6", listener);
      addButton("1", listener);
      addButton("2", listener);
      addButton("3", listener);
      addButton("0", listener);
      addButton(".", listener);

      clearButton = new JButton("CE");
      buttonPanel.add(clearButton);
      clearButton.addActionListener(
            new ClearButtonListener());

      add(buttonPanel, "Center");
   }
```

```
public double getValue()
{
   return Double.parseDouble(display.getText());
}


public void clear()
{
   display.setText("");
}


public void addButton(
      String label, ActionListener listener)
{
   JButton button = new JButton(label);
   buttonPanel.add(button);
   button.addActionListener(listener);
}


private class DigitButtonListener
       implements ActionListener
{
   public void actionPerformed(ActionEvent event)
   {
      JButton source = (JButton) event.getSource();
      String label = source.getText();
      // don't add two decimal points
      if (label.equals(".") &&
            display.getText().indexOf(".") != -1)
      {
         return;
      }
      display.setText(display.getText() + label);
   }
}


private class ClearButtonListener
       implements ActionListener
```

```
   {
      public void actionPerformed(ActionEvent event)
      {
         clear();
      }
   }
}
```

Class Bank

```
/**
 *  "My People Bank" application
 *
 *@author     Bonin
 *@version    1.0
 */
package bank;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.StringTokenizer;
import java.util.Vector;

public class Bank
{
   private Vector customers;


   public Bank()
   {
      customers = new Vector();
   }


   public void readCustomers(String filename)
          throws IOException
   {
      BufferedReader in = new BufferedReader(
            new FileReader(filename));
      boolean done = false;
      while (!done)
```

```
      {
         String inputLine = in.readLine();
         if (inputLine == null)
         {
            done = true;
         } else
         {
            StringTokenizer tokenizer =
                  new StringTokenizer(inputLine);
            int number =
                  Integer.parseInt(tokenizer.nextToken());
            int pin =
                  Integer.parseInt(tokenizer.nextToken());
            Customer c = new Customer(number, pin);
            addCustomer(c);
         }
      }
      in.close();
   }


   public void addCustomer(Customer c)
   {
      customers.add(c);
   }


   public Customer findCustomer(int aNumber, int aPin)
   {
      for (int i = 0; i < customers.size(); i++)
      {
         Customer c = (Customer) customers.get(i);
         if (c.match(aNumber, aPin))
         {
            return c;
         }
      }
      return null;
   }
}
```

Class `BankAccount`

```
/**
 *  "My People Bank" application
 *
 *@author     Bonin
 *@version    1.0
 */
package bank;

public class BankAccount
{
   private double balance = +0.0;


   public BankAccount() { }


   public void deposit(double aValue)
   {
      balance = balance + aValue;
   }


   public void withdraw(double aValue)
   {
      balance = balance - aValue;
   }


   public double getBalance()
   {
      return balance;
   }
}
```

Class `Customer`

```
/**
 *  "My People Bank" application
 *
 *@author     Bonin
```

```
 *@version    1.0
 */
package bank;

public class Customer
{
   public final static int CHECKING_ACCOUNT = 0;
   public final static int SAVINGS_ACCOUNT = 1;

   private int customerNumber;
   private int pin;
   private BankAccount[] accounts;


   public Customer(int aNumber, int aPin)
   {
      customerNumber = aNumber;
      pin = aPin;
      accounts = new BankAccount[2];
      accounts[CHECKING_ACCOUNT] = new BankAccount();
      accounts[SAVINGS_ACCOUNT] = new BankAccount();
   }


   public boolean match(int aNumber, int aPin)
   {
      return customerNumber == aNumber
             && pin == aPin;
   }


   public BankAccount getAccount(int a)
   {
      if (0 <= a && a < accounts.length)
      {
         return accounts[a];
      } else
      {
         return null;
      }
   }
}
```

Aspect `GlobalValue`

```
/**
 * "My People Bank" application
 *@author Bonin
 *@version 1.0
 */
package bank;

public aspect GlobalValue
{
  int MyATMFrame.FRAME_WIDTH = 500;
  int MyATMFrame.FRAME_HEIGHT = 300;
  String MyATMFrame.CUSTOMERS_DB =
      "D:\\bonin\\aosd\\code\\bank\\customers.txt";
}
```

Aspect `TransactionControl`

```
/**
 * "My People Bank" application
 *@author Bonin
 *@version 1.0
 */
package bank;

public aspect TransactionControl
{
    pointcut applyWithdraw(double v):
         args(v) && call(void withdraw(double));
    before(double v) : applyWithdraw(v)
        {
        System.out.println("withdraw(" + v + ")");
        }

    pointcut applyDeposit(double v):
         args(v) && call(void deposit(double));
    before(double v) : applyDeposit(v)
        {
        System.out.println("deposit(" + v + ")");
        }
}
```

File `customers.txt`

```
4711 1234
01031945 2
77 9999
```

Protocol `MyATM.log`

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.3
    (built 08.02.2002 12:47 PST) running on java 1.3.1

D:\bonin\aosd\code>ajc -argfile bank/files.lst

D:\bonin\aosd\code>java bank.MyATM
java bank.MyATM
deposit(100.0)
deposit(50.0)
withdraw(60.0)
Good bye!

D:\bonin\aosd\code>cd bank

D:\bonin\aosd\code\bank>dir
     1.739 Bank.class
     1.241 Bank.java
       992 BankAccount.class
       417 BankAccount.java
       936 Customer.class
       826 Customer.java
        32 customers.txt
       131 files.lst
       598 GlobalValue.class
       292 GlobalValue.java
       623 MyATM.class
       325 MyATM.java
     1.022 MyATMFrame$AButtonListener.class
       906 MyATMFrame$BButtonListener.class
       882 MyATMFrame$CButtonListener.class
       906 MyATMFrame$WindowCloser.class
     4.405 MyATMFrame.class
     5.016 MyATMFrame.java
       767 MyKeyPad$ClearButtonListener.class
```

```
      1.450 MyKeyPad$DigitButtonListener.class
      2.210 MyKeyPad.class
      2.322 MyKeyPad.java
      1.252 TransactionControl.class
        542 TransactionControl.java

D:\bonin\aosd\code\bank>
```

Result ↪ figure 7.6, p. 167.

7.14 Example: Koch Curve

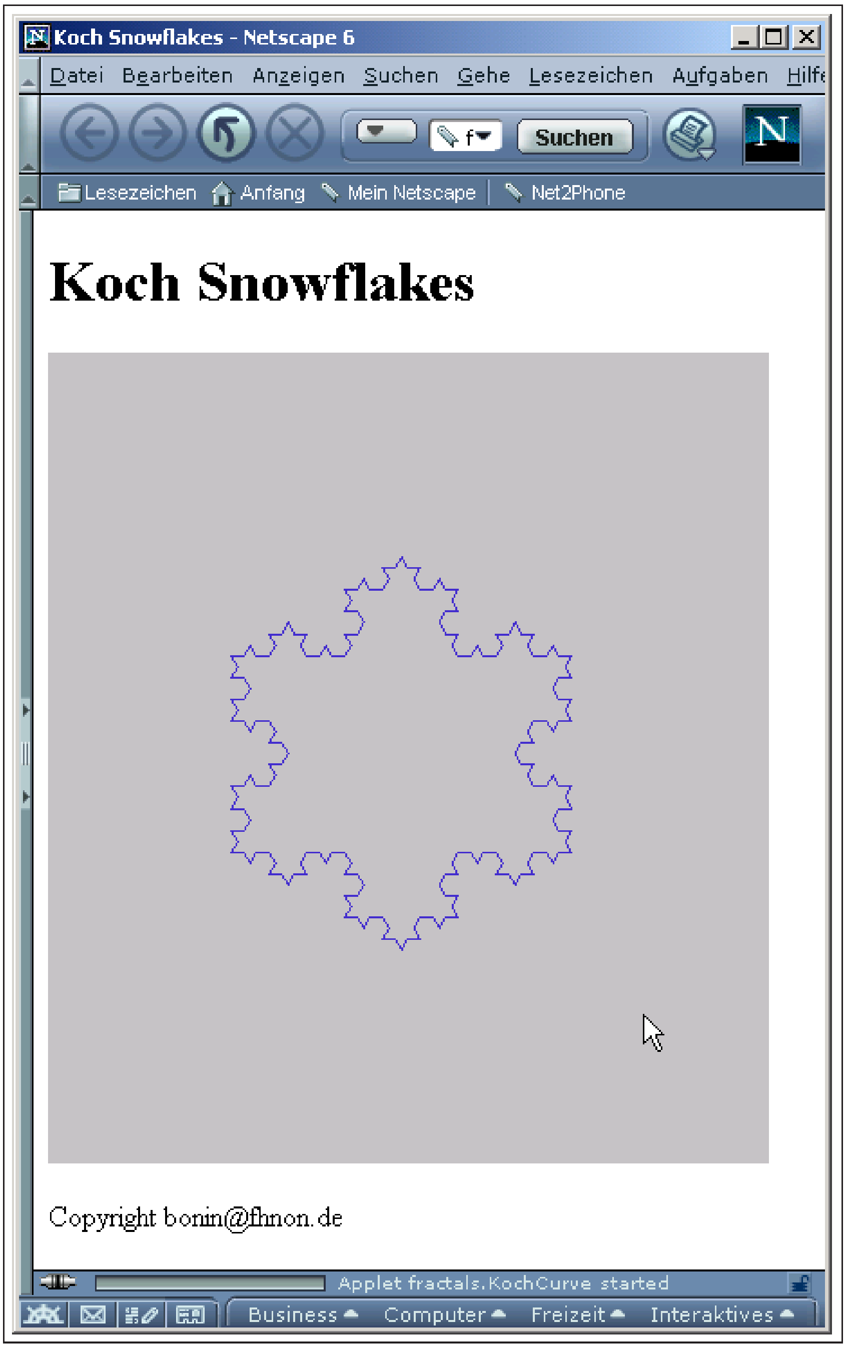
Helge von Koch was a Swedish mathematician who, in 1904, introduced what is now called the *Koch curve*. Fitting together three suitably rotated copies of the Koch curve produces a figure, which for obvious reason ist called the *snowflake curve* (↪ p.183) or the *Koch island* ↪ [Peitgen+, 1992a], p. 103.

Argument list `files.lst`

```
KochCurve.java
GlobalValue.java
ActionControl.java
```

Class `KochCurve`

```
/**
 *  "Koch Snowflakes"
 *
 *@author     Bonin
 *@version    1.0
 */
package fractals;

import java.applet.Applet;
import java.awt.Color;
import java.awt.geom.Rectangle2D;
import java.awt.geom.Line2D;
import java.awt.geom.Point2D;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.util.Random;

public class KochCurve extends Applet
{
```



Legende:
Netscape 6: Mozilla/5.0 (Windows; U; Windows NT 5.0; de-DE; m18) Gecko/20010131 Netscape6/6.01

Figure 7.7: Koch Snowflake

```
// Length of straight line at the beginning
private int l;
// Steps (of the recursion)
private int n;
// start point P(x,y)
private double x;
private double y;

// drawline (vector) with length rho
// and angle theta
public double rho;
private double theta;

Random generator = new Random();


public void init()
{
   n = Integer.parseInt(
         getParameter("depth"));
   l = Integer.parseInt(
         getParameter("length"));
}


public void paint(Graphics g)
{
   Graphics2D g2 = (Graphics2D) g;
   // SNOWFLAKES KochCurve()-Applications
   for (int i = 1; i <= SNOWFLAKES; i++)
   {
      setRandomColor(g2);
      setRandomRho();
      setStartPoint();
      drawKochCurve(n, g2, rho);
   }
}


public void drawKochCurve(
      int n, Graphics2D g2, double rho)
{
```

```
   /*
    *  P3
    *  \
    *  \
    *  \
    *  \
    *  \
    *  ----------
    *  P1          P2
    */
   // P1 --> P2
   drawFracCurve(n, g2, rho, 0.0);
   // P2 --> P3
   drawFracCurve(n, g2, rho, 2 * Math.PI / 3);
   // P3 --> P1
   drawFracCurve(n, g2, rho, 4 * Math.PI / 3.0);
}


public void drawFracCurve(
      int n, Graphics2D g2, double rho, double theta)
{
   if (n == 0)
   {
      drawFracL(g2, rho, theta);
   } else
   {
      int l = (int) Math.round(rho / 3);
      if (generator.nextInt(2) == 0)
      {
         /*
          *  ----\    /----
          *  \  /
          *  \/
          */
         drawFracCurve(
               n - 1, g2, l, theta);
         drawFracCurve(
               n - 1, g2, l, theta - Math.PI / 3);
         drawFracCurve(
               n - 1, g2, l, theta + Math.PI / 3);
         drawFracCurve(
```

```
                n - 1, g2, l, theta);
        } else
        {
           /*
            *  \
            *  \
            *  ----/    \----
            */
           drawFracCurve(
                 n - 1, g2, l, theta);
           drawFracCurve(
                 n - 1, g2, l, theta + Math.PI / 3);
           drawFracCurve(
                 n - 1, g2, l, theta - Math.PI / 3);
           drawFracCurve(
                 n - 1, g2, l, theta);
        }
     }
  }


  public void drawFracL(
        Graphics2D g2, double rho, double theta)
  {
     int l = (int) Math.round(rho / 3);
     if (generator.nextInt(2) == 0)
     {
        /*
         *  ----\    /----
         *  \  /
         *  \/
         */
        drawL(g2, l, theta);
        drawL(g2, l, theta - Math.PI / 3);
        drawL(g2, l, theta + Math.PI / 3);
        drawL(g2, l, theta);
     } else
     {
        /*
         *  \
         *  \
         *  ----/    \----
```

```
         */
        drawL(g2, l, theta);
        drawL(g2, l, theta + Math.PI / 3);
        drawL(g2, l, theta - Math.PI / 3);
        drawL(g2, l, theta);
     }
  }


  public void drawL(
        Graphics2D g2, double rho, double theta)
  {
     double xE = x + rho * Math.cos(theta);
     double yE = y + rho * Math.sin(theta);

     Point2D.Double p1 = new Point2D.Double(x, y);
     Point2D.Double p2 = new Point2D.Double(xE, yE);

     g2.draw(new Line2D.Double(p1, p2));

     x = xE;
     y = yE;
  }


  private void setRandomColor(Graphics2D g2)
  {
     float r;
     float g;
     float b;
     Random c = new Random();
     r = (float) c.nextDouble();
     g = (float) c.nextDouble();
     b = (float) c.nextDouble();
     g2.setColor(new Color(r, g, b));
  }


  private void setRandomRho()
  {
     rho = l * generator.nextDouble();
  }
```

```
   private void setStartPoint()
   {
      x = (getWidth() - rho) / 2;
      // delta 2/3: space for the first fractals
      double delta = 2.0 / 3.0;
      y = (getHeight() -
            delta * rho * Math.sin(Math.PI / 3.0)) / 2;
   }
}
```

Aspect `GlobalValue`

```
/**
 * "Koch Snowflakes"
 *@author Bonin
 *@version 1.0
 */
package fractals;

public aspect GlobalValue
{
  static int KochCurve.SNOWFLAKES = 20;
  static double ActionControl.SMALLEST_RHO = 100.00;
}
```

Aspect `ActionControl`

```
/**
 * "Koch Snowflakes"
 *@author Bonin
 *@version 1.0
 */
package fractals;

public aspect ActionControl {

    pointcut applySetStartPoint(KochCurve k):
        target(k) && call(void setStartPoint());
```

```
    before(KochCurve k) : applySetStartPoint(k) {
        if (k.rho <= SMALLEST_RHO ) {
                k.rho = k.rho + SMALLEST_RHO;
                System.out.println(
                   "corrected rho: " + k.rho);
        }
    }
}
```

File to run the Applet with `<object>`**-Element** `KochCurve.html`

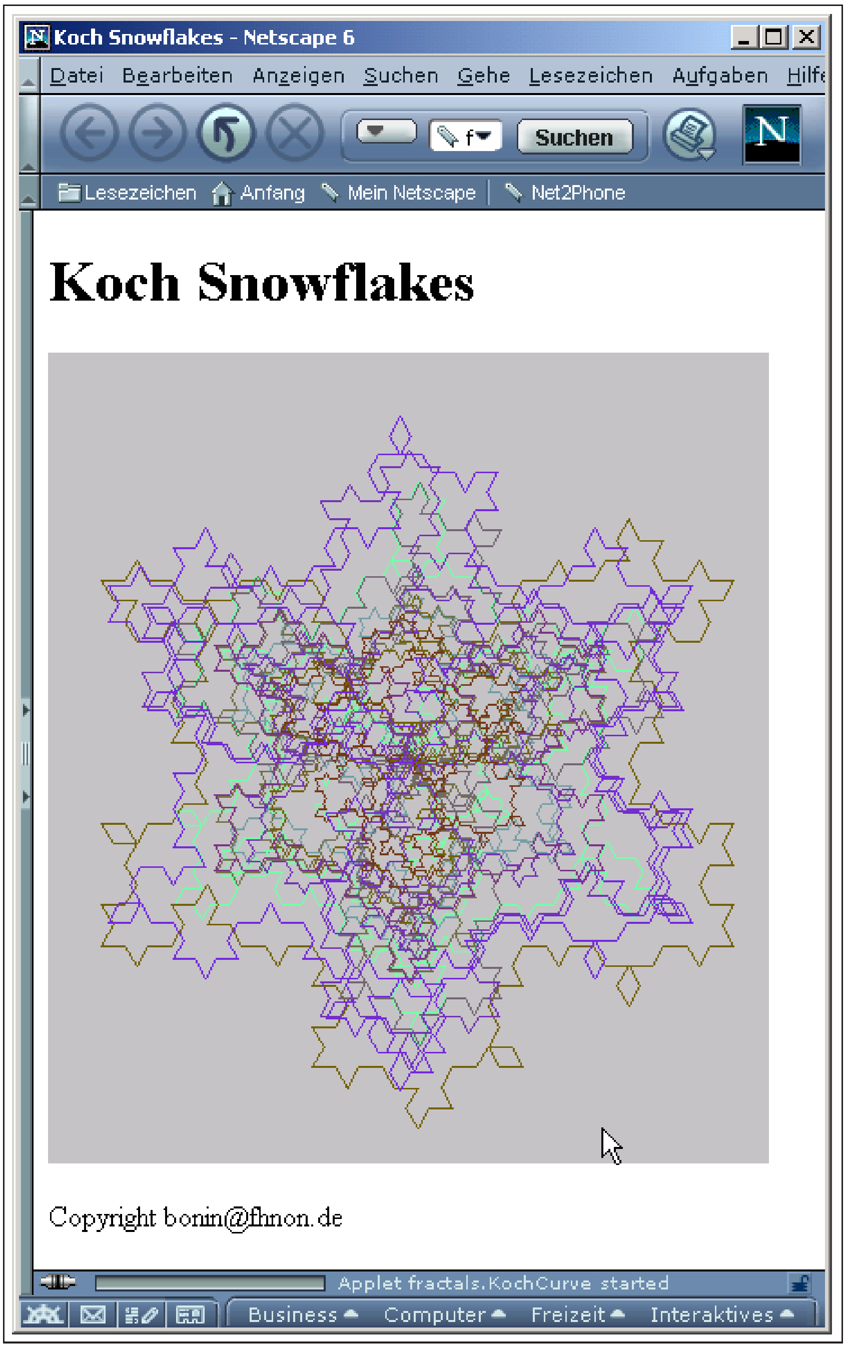
```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Bonin Version 1.0  -->
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<meta http-equiv="Content-Type"
  content="text/html; charset=utf-8" />
<title>Koch Snowflakes</title>
</head>
<body>
<h1>Koch Snowflakes</h1>
<object
  codetype="application/java"
  classid="java:fractals.KochCurve.class"
  code="fractals.KochCurve"
  width="400" height="450"
  alt="Fractals: Koch Curve">
  <param name="length" value="350" />
  <param name="depth" value="2" />
</object>
<p>Copyright bonin@fhnon.de</p>
</body>
</html>
```

Protocol `KochCurve.log`

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.3
    (built 08.02.2002 12:47 PST) running on java 1.3.1

D:\bonin\aosd\code>ajc -argfile fractals/files.lst
```

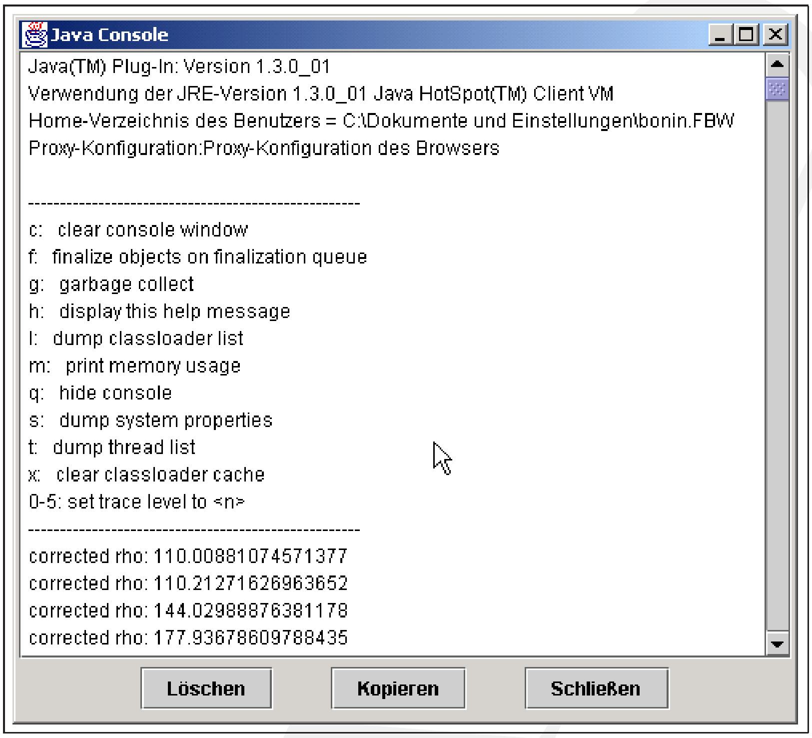
```
D:\bonin\aosd\code>
```

Result ↪ figure 7.8, p. 191 and *Java Console*↪ figure 7.9, p. 192.



Legende:
Netscape 6: Mozilla/5.0 (Windows; U; Windows NT 5.0; de-DE; m18) Gecko/20010131 Netscape6/6.01

Figure 7.8: Random Koch Curves



Legende:
Netscape 6: Mozilla/5.0 (Windows; U; Windows NT 5.0; de-DE; m18) Gecko/20010131 Netscape6/6.01

Figure 7.9: Java console of applet `fractals.KochCurve`

7.15 Java3D API

The Java 3D^{TM} API (*Application Programming Interface*) is an interface for writing programs to display and interact with three-dimensional graphics.

In the following simple example, the class `Figure3D` is defined to extend the class `Applet`. Java-3D-programs could be written as applications, but using `Applet` class gives an easy way to produce a windowed application. Therefore we create a `com.sun.j3d.utils.applet.MainFrame`[4] object that runs an applet as an application.

```
MainFrame(java.applet.Applet applet, int width, int
                         height)
```

Argument list `files.lst`

```
Figure3D.java
```

Class `Figure3D`

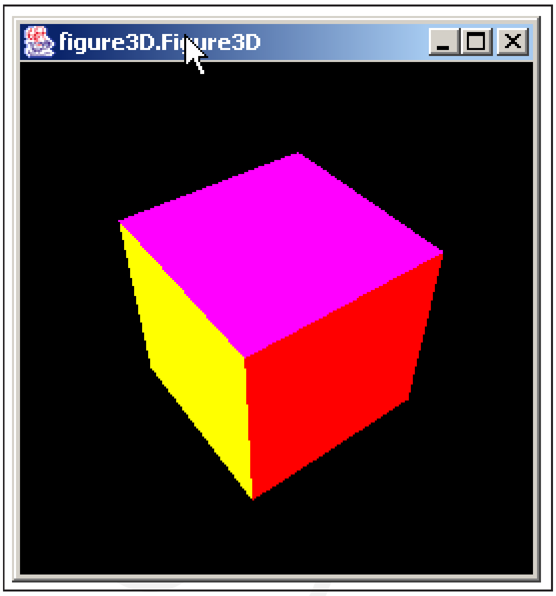
```
/**
 *  "Java 3D Example"
 *
 *@author     Bonin
 *@version    1.0
 */

package figure3D;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.Frame;
import java.awt.event.*;

import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.ColorCube;

import javax.media.j3d.*;
import javax.vecmath.*;

public class Figure3D extends Applet
```

[4] The `Mainframe` class is Copyright ©1996–1998 by Jef Poskanzer, ↪ http://www.acme.com/java/; see [Bouvier, 1999] p. 1-15

```
{
   public Figure3D()
   {
      setLayout(new BorderLayout());
      Canvas3D canvas3D = new Canvas3D(null);
      add("Center", canvas3D);
      BranchGroup scene = createSceneGraph();
      scene.compile();
      SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
      simpleU.getViewingPlatform().setNominalViewingTransform();
      simpleU.addBranchGraph(scene);
   }


   public BranchGroup createSceneGraph()
   {
      BranchGroup objRoot = new BranchGroup();
      Transform3D rotate = new Transform3D();
      Transform3D tempRotate = new Transform3D();
      rotate.rotX(Math.PI / 4.0d);
      tempRotate.rotY(Math.PI / 5.0d);
      rotate.mul(tempRotate);
      TransformGroup objRotate = new TransformGroup(rotate);

      objRotate.addChild(new ColorCube(0.4));
      objRoot.addChild(objRotate);
      return objRoot;
   }


   public static void main(String[] args)
   {
      Frame frame = new MainFrame(new Figure3D(), 256, 256);
   }

}
```

Protocol `Figure3D.log`

```
C:\bonin\aosd\eps>ajc -version
ajc version 1.0.6
  (built 24.07.2002 18:21 PST) running on java 1.4.0_01
```



Legende:
Source↪ `Figure3D` 25 p. 193

Figure 7.10: Java3D API — Simple figure

```
C:\bonin\aosd\code>ajc -argfile figure3D/files.lst

C:\bonin\aosd\code>java figure3D.Figure3D
WARNING: Canvas3D constructed with a null GraphicsConfiguration.
```

7.16 Execute a Daemon

Argument list `files.lst`

```
StartDaemon.java
Daemon.java
GlobalValue.java
```

Class `StartDaemon`

```
/**
 *  "Execute a Daemon"
```

```
 *
 *@author     Bonin
 *@version    1.0
 */

package daemon;

import java.lang.Runtime;
import java.util.Date;

class StartDaemon
{

   public static void main(String[] args)
   {

      Date tBegin = new Date();

      System.out.println(tBegin.getTime() + ": " +
            "StartDeamon start!");

      try
      {
         Runtime.getRuntime().exec(PATHJAVA +
               " daemon.Daemon ");
      } catch (Exception e)
      {
         System.err.println("Exception " + e);
      }

      Date tEnd = new Date();

      System.out.println(tEnd.getTime() + ": " +
            "StartDeamon end!");
   }
}
```

Class Daemon

```
/**
 *  "Execute a Daemon"
 *
```

```
 *@author     Bonin
 *@version    1.0
 */

package daemon;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Date;

class Daemon
{

   public static void main(String[] args)
   {

      Date tBegin = new Date();

      try
      {
         BufferedWriter w = new BufferedWriter(
               new FileWriter("Daemon.txt"));
         w.write(tBegin.getTime() + ": " +
               "Deamon is working!" +
               System.getProperty("line.separator"));

         long i = 0;
         while (i < 1000000000)
         {
            i++;
         }

         Date tEnd = new Date();

         w.write(tEnd.getTime() + ": " +
               "Deamon end!" +
               System.getProperty("line.separator"));
         w.close();
      } catch (Exception ignore)
      {
```

```
      }
   }
}
```

Aspect `GlobalValue`

```
/**
   "Execute a Deamon"
   @author Bonin
   @version 1.0
*/

package daemon;

public aspect GlobalValue
{
  static String StartDaemon.PATHJAVA =
      "C:\\Programme\\java2\\j2sdk1.4.1\\bin\\java.exe";
}
```

Protocol `StartDaemon.log`

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.5
    (built 27.06.2002 16:59 PST) running on java 1.4.1-beta

D:\bonin\aosd\code>ajc -argfile daemon/files.lst

D:\bonin\aosd\code>java daemon.StartDaemon
1028712543223: StartDeamon start!
1028712543243: StartDeamon end!

D:\bonin\aosd\code>
```

Protocol `Daemon.txt`

```
1028712543613: Deamon is working!
1028712561749: Deamon end!
```

7.17 Threads

Java threads provide an easy approach to providing multiple streams of execution within a process. The class `Monitor` (↪ p. 199) is a thread that runs, prints out the system time, waits a specified amount of time, then runs again.[5]

Argument list `files.lst`

```
Monitor.java
MonitorApp.java
GlobalValue.java
```

Class `Monitor` The modifier `volatile` instructs the compiler to generate loads and stores on each access to the attribute, rather than caching the value in a register.

```
/**
 *  "Thread" application
 *
 *@author     Bonin
 *@version    1.0
 */

package thread;

import java.lang.InterruptedException;

public class Monitor extends Thread
{

   private volatile Thread listener;
   private int frequency;


   public Monitor()
   {
      frequency = FREQUENCY;
   }


   public void run()
   {
      System.out.println("I am working!");
      try
      {
         listener = Thread.currentThread();
```

[5] Pattern for this program ↪ [Flenner et al., 2003] p. 44–45.

```
         while (listener != null)
         {
            System.out.println(
                  System.currentTimeMillis());
            Thread.sleep(frequency);
         }
      } catch (InterruptedException e)
      {
         System.err.println(
               "InterruptedException " + e);
      }
   }


   public void stopMonitor()
   {
      listener = null;
   }
}
```

Aspect `GlobalValue`

```
/**
 *  "Thread" application
 *@author Bonin
 * @version 1.0
 */

package thread;

public aspect GlobalValue
{
    static int Monitor.FREQUENCY = 60000;
}
```

Class `MonitorApp`

```
/**
 *  "Thread" application
 *
 *@author     Bonin
```

```
 *@version    1.0
 */

package thread;

public class MonitorApp
{
   public static void main(String[] args)
   {
      (new Monitor()).start();
   }
}
```

Protocol `MonitorApp.log`

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.5 (built 27.06.2002 16:59 PST)
  running on java 1.4.1-beta

D:\bonin\aosd\code>ajc -argfile thread/files.lst

D:\bonin\aosd\code>java thread.MonitorApp
I am working!
1034517371197
1034517431203
```

7.18 Self Replication

7.18.1 Calling the own `main()`

Argument list `files.lst`

```
Replica.java
GlobalValue.java
```

Class `Replica`

```
public Object invoke(Object obj, Object[] args)
        throws IllegalAccessException,
               IllegalArgumentException,
               InvocationTargetExceptionInvokes
```

The underlying method represented by this `Method` object, on the specified object `obj` with the specified parameters `args`. Individual parameters are automatically unwrapped to match primitive formal parameters, and both primitive and reference parameters are subject to method invocation conversions as necessary. If the underlying method is `static`, then the specified `obj` argument is ignored. It may be `null`.

```
/**
 *  "Self Replication" example
 *
 *@author     Bonin
 *@version    1.0
 */
package replication;

import java.lang.reflect.*;

public class Replica implements Runnable
{

   private String slot = "Replica!";


   public String getSlot()
   {
      return slot;
   }


   public void setSlot(String slot)
   {
      this.slot = slot;
   }


   public static void main(String[] args)
   {
      System.out.println(
            "\n Enter replication.Replica.main()");
      while (true)
      {
         new Thread(new Replica()).start();
      }
   }
```

```
public void run()
{
   if (COUNT <= 3)
   {
      COUNT++;
   } else
   {
      System.out.println("Good bye!");
      System.exit(0);
   }
   try
   {
      Class c =
            Class.forName("replication.Replica");
      ClassLoader l = c.getClassLoader();
      Replica i = (Replica) c.newInstance();
      i.setSlot("More ...");
      Method[] m = c.getDeclaredMethods();
      for (int k = 0; k < m.length; k++)
      {
         System.out.println("method: " + m[k]);
      }
      /*
       *  i.setSlot("Hello World " + COUNT);
       */
      m[3].invoke(i,
            new Object[]{"Hello World " + COUNT});
      /*
       *  i.getSlot();
       */
      String s = (String) m[2].invoke(i, null);
      System.out.println(
            "class loader: " + l + "\n" +
            " instance: " + i + "\n" +
            " slot: " + s);
      /*
       *  Replica.main();
       *  If the underlying method is static, as main() is,
       *  then the specified obj argument is
       *  ignored. Here it is null.
```

```
         */
        Object[] args = new Object[1];
        args[0] = new String[1];
        m[0].invoke(null, args);

      } catch (ClassNotFoundException e)
      {
        System.out.println(
              "Class not found exception: " + e);
      } catch (InstantiationException e)
      {
        System.out.println(
              "Instantiation exception: " + e);
      } catch (IllegalAccessException e)
      {
        System.out.println(
              "Illegal access exception: " + e);
      } catch (InvocationTargetException e)
      {
        System.out.println(
              "Invocation target exception: " + e);
      }
   }
}
```

Aspect `GlobalValue`

```
/**
 * "Self Replication" example
 *@author Bonin
 *@version 1.0
 */
package replication;

public aspect GlobalValue
{
  static int Replica.COUNT = 0;
}
```

Protocol `Replica.log`

```
D:\bonin\aosd\code>ajc -version
```

```
ajc version 1.0.3 (built 08.02.2002 12:47 PST) running on java 1.3.1

D:\bonin\aosd\code>ajc -argfile replication/files.lst

D:\bonin\aosd\code>java replication.Replica

 Enter replication.Replica.main()
method: public static void replication.Replica.main(java.lang.String[])
method: public void replication.Replica.run()
method: public java.lang.String replication.Replica.getSlot()
method: public void replication.Replica.setSlot(java.lang.String)
class loader: sun.misc.Launcher$AppClassLoader@71732b
 instance: replication.Replica@62eec8
 slot: Hello World 1

 Enter replication.Replica.main()
method: public static void replication.Replica.main(java.lang.String[])
method: public void replication.Replica.run()
method: public java.lang.String replication.Replica.getSlot()
method: public void replication.Replica.setSlot(java.lang.String)
class loader: sun.misc.Launcher$AppClassLoader@71732b
 instance: replication.Replica@2a9835
 slot: Hello World 2
method: public static void replication.Replica.main(java.lang.String[])
method: public static void replication.Replica.main(java.lang.String[])
Good bye!
Good bye!
Good bye!

 Enter replication.Replica.main()
method: public void replication.Replica.run()
method: public void replication.Replica.run()
method: public java.lang.String replication.Replica.getSlot()
method: public java.lang.String replication.Replica.getSlot()
method: public void replication.Replica.setSlot(java.lang.String)

D:\bonin\aosd\code>java replication.Replica

 Enter replication.Replica.main()
Good bye!
method: public static void replication.Replica.main(java.lang.String[])
method: public void replication.Replica.run()
method: public java.lang.String replication.Replica.getSlot()
method: public void replication.Replica.setSlot(java.lang.String)
class loader: sun.misc.Launcher$AppClassLoader@71732b
 instance: replication.Replica@2a9835
 slot: Hello World 4

 Enter replication.Replica.main()
method: public static void replication.Replica.main(java.lang.String[])
method: public void replication.Replica.run()
method: public java.lang.String replication.Replica.getSlot()
method: public void replication.Replica.setSlot(java.lang.String)
```

```
method: public static void replication.Replica.main(java.lang.String[])
method: public void replication.Replica.run()
method: public java.lang.String replication.Replica.getSlot()
method: public void replication.Replica.setSlot(java.lang.String)
class loader: sun.misc.Launcher$AppClassLoader@71732b
 instance: replication.Replica@53c015
 slot: Hello World 4

 Enter replication.Replica.main()
class loader: sun.misc.Launcher$AppClassLoader@71732b
 instance: replication.Replica@67ac19
 slot: Hello World 4

 Enter replication.Replica.main()
method: public static void replication.Replica.main(java.lang.String[])
method: public void replication.Replica.run()
method: public java.lang.String replication.Replica.getSlot()
method: public void replication.Replica.setSlot(java.lang.String)
class loader: sun.misc.Launcher$AppClassLoader@71732b
 instance: replication.Replica@53ba3d
 slot: Hello World 4

 Enter replication.Replica.main()
Good bye!
Good bye!
Good bye!
Good bye!
Good bye!
Good bye!

D:\bonin\aosd\code>
```

7.18.2 Reading the own source code

Argument list `files.lst`

```
MySelf.java
MethodMain.java
```

Class `MySelf`

```
/**
 *  "Self Replication" application
 *
 *@author     Bonin
 *@version    1.0
 */

package selfrep;
```

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Random;

public class MySelf extends Thread
{
   static int state = 1;
   static String line;


   public void run()
   {
      BufferedReader r;
      BufferedWriter w;

      Random generator = new Random();

      /*
       *  constructing
       *  "selfrep\MySelf.java" and
       *  "selfrep\TempMySelf.java"
       */
      String c = getClass().toString();
      String myPackage = c.substring(6, 13);
      String myClassName = c.substring(14, 20);
      String place =
            myPackage +
            System.getProperty("file.separator") +
            myClassName + ".java";
      String temp =
            myPackage +
            System.getProperty("file.separator") +
            "Temp" +
            myClassName + ".java";

      synchronized (getClass())
      {
```

```
try
{
   r = new BufferedReader(
         new FileReader(place));
   w = new BufferedWriter(
         new FileWriter(temp));

   while ((line = r.readLine()) != null)
   {

      /*
       *  Modifing the class variable state
       */
      if ((line.length() > 23) &&
            line.substring(15, 24).equals("state = 1"))
      {
         w.write(line.substring(
               0, line.length() - 1) +
               generator.nextInt(2) +
               ";" +
               System.getProperty(
               "line.separator"));
      } else
      {
         w.write(line +
               System.getProperty(
               "line.separator"));
      }
   }
   r.close();
   w.close();
   change(place, temp);
} catch (IOException e)
{
   System.err.println("IOException " + e);
}

System.out.println(
      myClassName + ".state = " + state);

/*
```

```
         *  control in DOS batch
         */
        System.exit(generator.nextInt(2));
     }
     ;
  }


  /*
   *  changing TempMySelf.java to MySelf.java
   */
  synchronized void change(String p, String t)
  {
     (new File(p)).delete();
     (new File(t)).renameTo(new File(p));
  }
}
```

Aspect `MethodMain`

```
/**
 * "Self Replication" application
 *@author Bonin
 *@version 1.0
 */

package selfrep;

public aspect MethodMain
{
    public static void MySelf.main(String[] args)
        {
            Thread t1 = new MySelf();
            Thread t2 = new MySelf();

            t1.start();
            t2.start();
        }
}
```

DOS-Batch `MySelf.bat`

```
REM Compile and run MySelf
REM Bonin
call ajc -version
REM
:begin
call ajc -argfile selfrep/files.lst
REM
java selfrep.MySelf
if errorlevel 1 goto begin
REM
REM End of object MySelf.bat
```

Protocol `MySelf.log`

```
D:\bonin\aosd\code>MySelf

D:\bonin\aosd\code>REM Compile and run MySelf

D:\bonin\aosd\code>REM Bonin

D:\bonin\aosd\code>call ajc -version
ajc version 1.0.5
    (built 27.06.2002 16:59 PST) running on java 1.4.1-beta
MySelf.state = 1
MySelf.state = 11
MySelf.state = 111

D:\bonin\aosd\code>MySelf

D:\bonin\aosd\code>REM Compile and run MySelf

D:\bonin\aosd\code>REM Bonin

D:\bonin\aosd\code>call ajc -version
ajc version 1.0.5
    (built 27.06.2002 16:59 PST) running on java 1.4.1-beta
MySelf.state = 1110

D:\bonin\aosd\code>MySelf

D:\bonin\aosd\code>REM Compile and run MySelf

D:\bonin\aosd\code>REM Bonin

D:\bonin\aosd\code>call ajc -version
ajc version 1.0.5
```

```
   (built 27.06.2002 16:59 PST) running on java 1.4.1-beta
MySelf.state = 11100
MySelf.state = 111001
MySelf.state = 1110011
MySelf.state = 11100111
MySelf.state = 111001111
MySelf.state = 1110011111
.\selfrep\MySelf.java:18:24: invalid int decimal literal
    static int state = 11100111110;
                       ^
1 errors
MySelf.state = 1110011111

D:\bonin\aosd\code>
```

7.18.3 Compile & Execute the own Code

Argument list `files.lst`

```
ExecMySelf.java
GlobalValue.java
```

Class `MySelf`

```
/**
 *  "Compile and execute this Java-File"
 *
 *@author     Bonin
 *@version    1.0
 */

package runtime;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.lang.InterruptedException;
import java.lang.Process;
import java.lang.Runtime;
import java.util.Random;

class ExecMySelf
{
   static Random generator = new Random();
```

```
public static void main(String[] args)
{

   System.out.println("I am working!");

   try
   {
      /*
       *  compile with ajc and execute java
       */
      if (generator.nextInt(2) == 0)
      {
         Runtime rt1 = Runtime.getRuntime();
         Process pr1 = rt1.exec(PATHAJC +
               " -argfile runtime\\files.lst");
         pr1.waitFor();

         Runtime rt2 = Runtime.getRuntime();
         Process pr2 = rt2.exec(PATHJAVA +
               " runtime.ExecMySelf");
         pr2.waitFor();

         BufferedReader pr2out =
               new BufferedReader(
               new InputStreamReader(
               pr2.getInputStream()));
         String line;
         while ((line = pr2out.readLine()) != null)
         {
            System.out.println(" OUT> " + line);
         }
      }
   } catch (IOException e)
   {
      System.err.println("IOException " + e);
   } catch (InterruptedException e)
   {
      System.err.println("InterruptedException " + e);
   }
}
```

```
}
```

Aspect GlobalValue

```
/**
 * "Compile and execute this Java-File"
 *@author Bonin
 *@version 1.0
 */

package runtime;

public aspect GlobalValue
{
  static String ExecMySelf.PATHAJC =
      "C:\\Programme\\aspectj1.0\\bin\\ajc.bat";

  static String ExecMySelf.PATHJAVA =
      "C:\\Programme\\java2\\j2sdk1.4.1\\bin\\java.exe";
}
```

Protocol ExecMySelf.log

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.5
    (built 27.06.2002 16:59 PST) running on java 1.4.1-beta

D:\bonin\aosd\code>ajc -argfile runtime/files.lst

D:\bonin\aosd\code>java runtime.ExecMySelf
I am working!
 OUT> I am working!
 OUT>  OUT> I am working!
 OUT>  OUT>  OUT> I am working!
 OUT>  OUT>  OUT>  OUT> I am working!

D:\bonin\aosd\code>
I am working!
 OUT> I am working!
 OUT>  OUT> I am working!
 OUT>  OUT>  OUT> I am working!
 OUT>  OUT>  OUT>  OUT> Error occurred during initialization of VM
 OUT>  OUT>  OUT>  OUT> Could not reserve enough space for object heap
```

```
D:\bonin\aosd\code>
```

7.19 Casting Problem by Collections

This exercise is the classical example to demonstrates the need of generic classes in Java (↪ [Bracha et al., 1998] p. 3–5). Here we use it to show the casting problem by collections. The collection interface (↪ p. 214) provides a method to add an element to a collection (`add()`), and a method to return an iterator for the collection (`iterator()`). In turn, the iterator interface (↪ p. 215) provides a method to determine if the iterator is done (`hasNext()`), and (if it is not) a method to return the next element and advance the iterator (`next`). The linked list class (↪ p. 215) implements the collections interface, and contains a nested class for list nodes and an anonymous class for the list iterator. Each element has type `Object`, so one may form linked lists with elements of any reference type, including `Byte`, `String`, or `LinkedList` itself (↪ p. 217).

Argument list `files.lst`

```
Collection.java
Iterator.java
NoSuchElementException.java
LinkedList.java
CastProg.java
```

Interface `Collection`

```
/**
 *  "Casting Problem" application
 *
 *@author     Bonin
 *@version    1.0
 */

package casting;

interface Collection
{
   public void add(Object x);


   public Iterator iterator();
}
```

Interface `Iterator`

```
/**
 *  "Casting Problem" application
 *
 *@author     Bonin
 *@version    1.0
 */

package casting;

interface Iterator
{
   public Object next();


   public boolean hasNext();
}
```

Class `NoSuchElementException`

```
/**
 *  "Casting Problem" application
 *
 *@author     Bonin
 *@version    1.0
 */

package casting;

class NoSuchElementException extends RuntimeException
{

}
```

Class `LinkedList`

```
/**
 *  "Casting Problem" application
 *
 *@author     Bonin
```

```
 *@version    1.0
 */

package casting;

class LinkedList implements Collection
{
   protected class Node
   {
      Object elt;
      Node next = null;


      Node(Object elt)
      {
         this.elt = elt;
      }
   }


   protected Node head = null, tail = null;


   public LinkedList() { }


   public void add(Object elt)
   {
      if (head == null)
      {
         head = new Node(elt);
         tail = head;
      } else
      {
         tail.next = new Node(elt);
         tail = tail.next;
      }
   }


   public Iterator iterator()
   {
```

```
      return
         new Iterator()
         {
            protected Node ptr = head;


            public boolean hasNext()
            {
               return ptr != null;
            }


            public Object next()
            {
               if (ptr != null)
               {
                  Object elt = ptr.elt;
                  ptr = ptr.next;
                  return elt;
               } else
               {
                  throw new NoSuchElementException();
               }
            }
         };
   }
}
```

Class `CastProg`

```
/**
 *  "Casting Problem" application
 *
 *@author     Bonin
 *@version    1.0
 */

package casting;
import java.lang.Byte;
```

```
class CastProg
{
   public static void main(String[] args)
   {

      /*
       *  byte list
       */
      LinkedList bl = new LinkedList();
      bl.add(new Byte("0"));
      bl.add(new Byte("1"));
      boolean more = (boolean) bl.iterator().hasNext();
      Byte b = (Byte) bl.iterator().next();
      System.out.println(
            "List has next elements = " + more + "\n" +
            "b = " + b);

      /*
       *  string list
       */
      LinkedList sl = new LinkedList();
      sl.add("first");
      sl.add("second");
      String s = (String) sl.iterator().next();
      System.out.println("s = " + s);

      /*
       *  string list list
       */
      LinkedList ll = new LinkedList();
      ll.add(sl);
      String l = (String) (
            (LinkedList) ll.iterator().next()
            ).iterator().next();
      System.out.println("l = " + l);

      /*
       *  string list treated as byte list
       */
      /*
       *  a good solution
       */
```

```
      Object x = sl.iterator().next();
      Byte y;
      if (x instanceof Byte)
      {
         y = (Byte) x;
      } else
      {
         System.err.println("Wrong casting!");
      }
      /*
       *  a quick solution gets run-time exception
       */
      Byte z = (Byte) sl.iterator().next();
   }
}
```

Protocol `CastProg.log`

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.3
   (built 08.02.2002 12:47 PST) running on java 1.3.1

D:\bonin\aosd\code>ajc -argfile casting/files.lst

D:\bonin\aosd\code>java casting.CastProg
java casting.CastProg
List has next elements = true
b = 0
s = first
l = first
Wrong casting!
java.lang.ClassCastException: java.lang.String
at casting.CastProg.main(CastProg.java:49)
Exception in thread "main"

D:\bonin\aosd\code>cd casting

D:\bonin\aosd\code\casting>dir
    1.848 CastProg.class
    1.449 CastProg.java
      951 CastProg.log
      321 Collection.class
      196 Collection.java
```

```
       93 files.lst
      293 Iterator.class
      186 Iterator.java
      704 LinkedList$Node.class
      998 LinkedList$_1.class
      981 LinkedList.class
    1.151 LinkedList.java
      404 NoSuchElementException.class
      165 NoSuchElementException.java

D:\bonin\aosd\code\casting>
```

7.20 Peer-to-Peer platform JXTA

JXTA[6] is a peer-to-peer framework developed by Sun Microsystems under the direction of Bill Joy and Mike Clary with the follwing requirements (↪ [Gradecki, 2002] p. 15):

1. *Peers should be able to discover one another.*
2. *Peers should self-organize into peer groups.*
3. *Peers should advertise and discover network resources.*
4. *Peers should communicate with one another.*
5. *Peers should monitor one another.*
6. *The platform should not require the use of any particular computer language or operating system.*
7. *The platform should not require the use of any particular network transport or topology.*
8. *The platform should not require the use of any particular authentication, security, or encryption model.*

7.20.1 Peergroup

The JXTA application `HelloWorld` discovers and joins the `NetPeerGroup`[7]. New peergroups are created within the `NetPeerGroup` and are composed of a subset of the `NetPeerGroup` peer members.

The directory in which the application `HelloWorld` was run gets the new directory `.jxta` which contains the peergroup cache directory `cm`, the platform configuration file `PlatformConfig`, and the username directory `pse`.

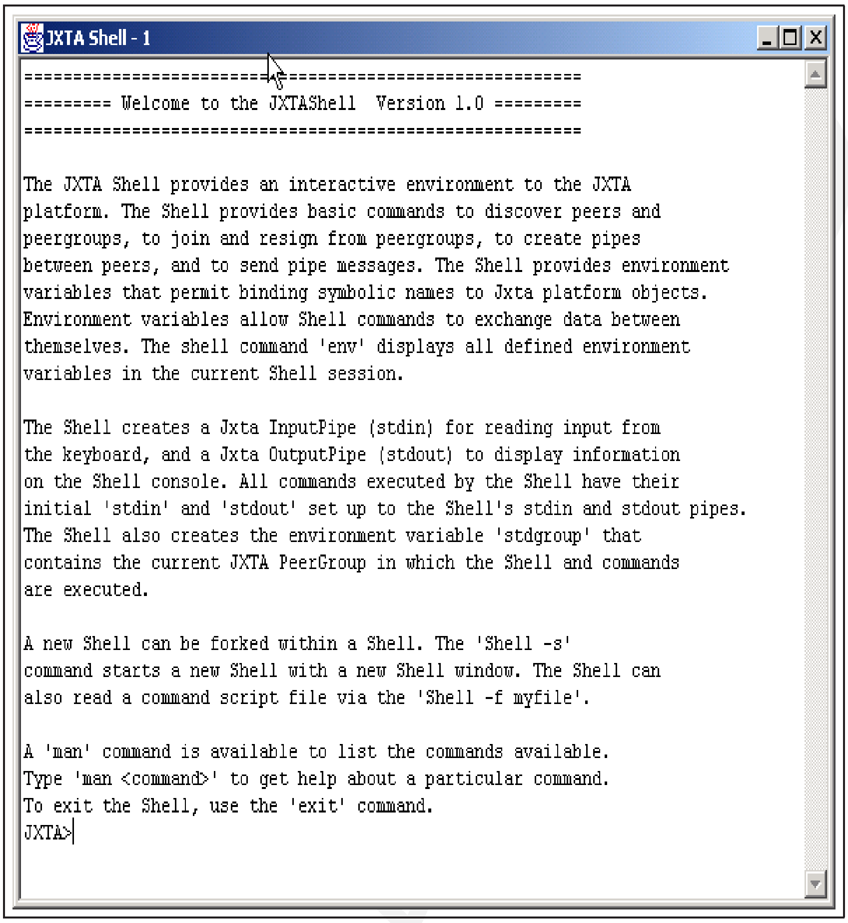
[6]JXTA ≡ <u>Ju</u><u>xta</u>pose (pronounced *juxta*)
[7]also called the *World Peergroup*.

Legende:
Web-Site ↪ `http://www.jxta.org/`

Figure 7.11: Project JXTA

Class `HelloWorld`
Notice: The idea of this example "Hello World" is taken from ↪ [Oaks+, 2002], p. 40. The code is modified.

```
/**
 *   "JXTA Hello World"
 *
 *@author     Bonin
 *@version    1.0
 */

package peergroup;

import net.jxta.exception.PeerGroupException;
import net.jxta.peergroup.PeerGroup;
import net.jxta.peergroup.PeerGroupFactory;

public class HelloWorld
{
   static PeerGroup myGroup = null;


   public static void main(String[] args)
```



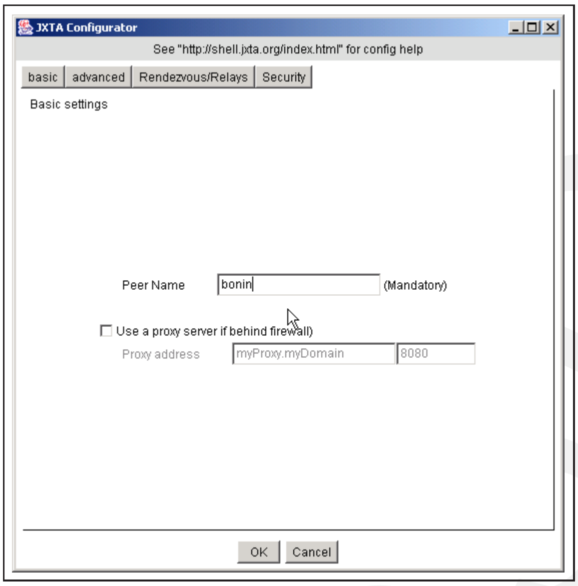
Legende:
Web-Site ↪ http://www.jxta.org/

Figure 7.12: JXTA-Shell

```
   {
      HelloWorld myApp = new HelloWorld();
      myApp.startJxta();
      System.exit(0);
   }

   private void startJxta()
   {
      try
      {
         /*
          *  Default JXTA PeerGroup
          */
         myGroup =
               PeerGroupFactory.newNetPeerGroup();
      } catch (PeerGroupException e)
      {
         System.err.println(
               "PeerGroupException " + e);
         System.exit(1);
      }
      System.out.println("JXTA --- Hello World!");
   }
}
```

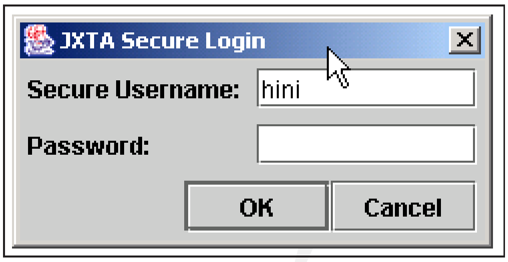
Protocol `HelloWorld.log`

```
D:\bonin\aosd\code>echo %CLASSPATH%
.;C:\Programme\POET61\lib\POET6ODMG3JC_SDK.jar; ....
c:\programme\jxta_demo\lib\jxta.jar;
c:\programme\jxta_demo\lib\jxtacms.jar;
c:\programme\jxta_demo\lib\jxtaptls.jar;
c:\programme\jxta_demo\lib\jxtasecurity.jar;
c:\programme\jxta_demo\lib\jxtashell.jar;
c:\programme\jxta_demo\lib\log4j.jar;
c:\programme\jxta_demo\lib\minimalBC.jar;
c:\programme\jxta_demo\lib\org.mortbay.jetty.jar;
c:\programme\jxta_demo\lib\beepcore.jar;
c:\programme\jxta_demo\lib\cmsshell.jar;
c:\programme\jxta_demo\lib\cryptix32.jar;
c:\programme\jxta_demo\lib\cryptix-asn1.jar;
```



Legende:
JXTA Application `HelloWorld.java` ↪ p. 221

Figure 7.13: JXTA Configurator



Legende:
JXTA Application `HelloWorld.java` ↪ p. 221

Figure 7.14: JXTA Secure Login

```
c:\programme\jxta_demo\lib\instantp2p.jar;
c:\programme\jxta_demo\lib\javax.servlet.jar

D:\bonin\aosd\code>java -fullversion
java full version "1.4.1-beta-b14"

D:\bonin\aosd\code>javac peergroup/HelloWorld.java

D:\bonin\aosd\code>java peergroup.HelloWorld

Security initialization in progress.
This will take 10 or more seconds ...

JXTA --- Hello World!

D:\bonin\aosd\code>java peergroup.HelloWorld
JXTA --- Hello World!

D:\bonin\aosd\code>cd .jxta\cm\jxta-NetGroup\Peers

D:\bonin\aosd\code\.jxta\cm\jxta-NetGroup\Peers>dir

 1.683 uuid-59616261646162614A78746150325033FD5463DB9CDE4CD387C2A33D0802C6A80
               1 Datei(en)          1.683 Bytes

D:\bonin\aosd\code\.jxta\cm\jxta-NetGroup\Peers>
```

Class `MyAdvertisement`

JXTA objects that represent resources usually contain their advertisement as a property. In this example we retrieve the advertisement used to create a peergroup via the method `getPeerGroupAdvertisement()`.
Notice: The idea of this example is taken from ↪ [Oaks+, 2002], p. 43. The code is modified.

```
/**
 *  "JXTA Advertisement"
 *
 *@author     Bonin
 *@version    1.0
 */

package peergroup;

import java.io.StringWriter;
import net.jxta.document.Advertisement;
```

```
import net.jxta.document.Document;
import net.jxta.document.Element;
import net.jxta.document.MimeMediaType;
import net.jxta.document.StructuredDocument;
import net.jxta.document.StructuredTextDocument;
import net.jxta.exception.PeerGroupException;
import net.jxta.peergroup.PeerGroup;
import net.jxta.peergroup.PeerGroupFactory;

public class MyAdvertisement
{
   static PeerGroup myGroup = null;
   static Advertisement myPgadv = null;


   public static void main(String[] args)
   {

      MyAdvertisement myApp = new MyAdvertisement();
      myApp.startJxta();
      System.exit(0);
   }


   private void startJxta()
   {
      try
      {
         /*
          *  Default JXTA PeerGroup
          */
         myGroup =
               PeerGroupFactory.newNetPeerGroup();

         myPgadv = myGroup.getPeerGroupAdvertisement();

         /*
          *  Print out the PeerGroup advertisement document
          */
         StructuredTextDocument myDoc =
               (StructuredTextDocument)
               myPgadv.getDocument(
```

```
                new MimeMediaType("text/xml"));
          StringWriter myOut = new StringWriter();
          myDoc.sendToWriter(myOut);
          System.out.println(myOut.toString());
          myOut.close();
       } catch (PeerGroupException e)
       {
          System.err.println(
                "PeerGroupException " + e);
          System.exit(1);
       } catch (Exception e)
       {
          System.err.println(
                "Exception " + e);
       }
    }
}
```

Protocol `MyAdvertisement.log`

```
D:\bonin\aosd\code>java -fullversion
java full version "1.4.1-beta-b14"

D:\bonin\aosd\code>javac peergroup/MyAdvertisement.java

D:\bonin\aosd\code>java peergroup.MyAdvertisement
<?xml version="1.0"?>

<!DOCTYPE jxta:PGA>

<jxta:PGA xmlns:jxta="http://jxta.org">
        <GID>
                urn:jxta:jxta-NetGroup
        </GID>
        <MSID>
                urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE000000010206
        </MSID>
        <Name>
                NetPeerGroup
        </Name>
        <Desc>
                NetPeerGroup by default
        </Desc>
</jxta:PGA>
```

IP Address	Address Range
Class A	0.0.0.0–127.255.255.255
Class B	128.0.0.0–191.255.255.255
Class C	192.0.0.0–223.255.255.255
Multicast: Class D	224.0.0.0–239.255.255.255
Reserved	240.0.0.0–247.255.255.255

Legende:
Reserved multicast addresses
↪ `http://www.iana.org/assignments/multicast-addresses`

Table 7.1: IPv4 Address Classifications

```
D:\bonin\aosd\code>
```

7.20.2 Multicast Messaging

Multicast messaging is something like radio, only those who have tuned their receivers to a particular channel receive the data. The sender sends the data without knowlegde of the number of receivers. *Unicast messaging* is the opposite, one sender and one receiver. Multicast addresses are in the Class D of IPv4 (*Internet Protocol Version 4*) ↪ Table 7.1 p. 228.

Argument list `files.lst`

```
MulticastSender.java
MulticastListener.java
GlobalValue.java
```

Aspect `GlobalValue`

```
/**
 * "Send a datagram using multicast"
 *@author Bonin
 *@version 1.0
 */

package multicast;

public aspect GlobalValue
{
```

```
  static int port = 6789;
  static String address =  "224.5.6.7";

  static int MulticastListener.PORT = port;
  static String MulticastListener.ADDRESS = address;
  static int MulticastSender.PORT = port;
  static String MulticastSender.ADDRESS = address;
}
```

Class `MulticastListener`

```
/**
 *  "Send a datagram using multicast"
 *
 *@author     Bonin
 *@version    1.0
 */

package multicast;

import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.MulticastSocket;
import java.net.UnknownHostException;
import java.io.IOException;

public class MulticastListener
{
   public static void main(String[] args)
   {
      byte[] data = new byte[1000];
      try
      {
         InetAddress ip =
               InetAddress.getByName(ADDRESS);
         MulticastSocket ms =
               new MulticastSocket(PORT);
         ms.joinGroup(ip);
         DatagramPacket packet =
               new DatagramPacket(
               data, data.length);
         ms.receive(packet);
         String message =
```

```
               new String(packet.getData(), 0,
               packet.getLength());
         System.out.println(message);
         ms.close();
      } catch (UnknownHostException e)
      {
         System.err.println(
               "UnknownHostException " + e);
      } catch (IOException e)
      {
         System.err.println("IOException " + e);
      }
   }
}
```

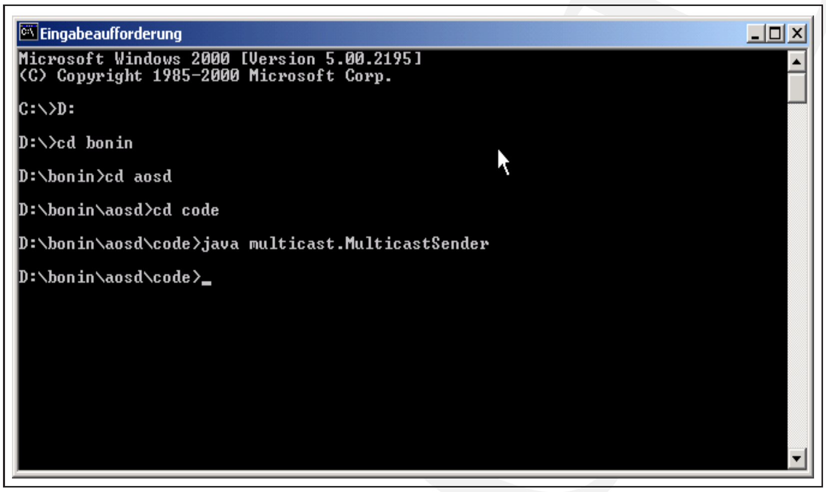
Class `MulticastSender`

```
/**
 *  "Send a datagram using multicast"
 *
 *@author     Bonin
 *@version    1.0
 */

package multicast;

import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.MulticastSocket;
import java.net.UnknownHostException;
import java.io.IOException;

public class MulticastSender
{
   public static void main(String[] args)
   {
      String data =
            "Hello World --- Multicast!";
      try
      {
         InetAddress ip =
               InetAddress.getByName(ADDRESS);
```

```
          DatagramPacket packet =
                new DatagramPacket(
                data.getBytes(), data.length(),
                ip, PORT);
          MulticastSocket ms =
                new MulticastSocket();
          ms.send(packet);
          ms.close();
       } catch (UnknownHostException e)
       {
          System.err.println(
                "UnknownHostException " + e);
       } catch (IOException e)
       {
          System.err.println("IOException " + e);
       }
    }
}
```

Protocol `Multicast.log` Protocol `java multicast.MulticastSender`
↪ Protocol 232

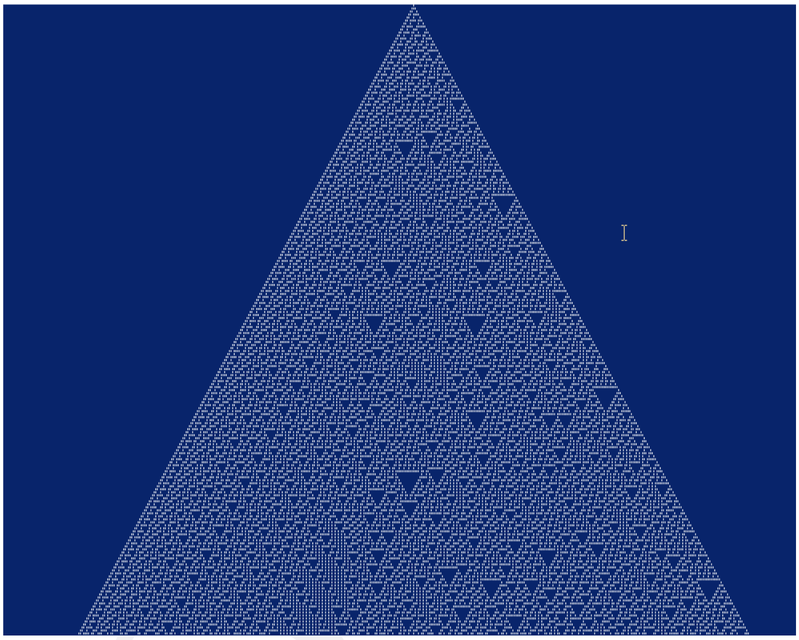
```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.5
  (built 27.06.2002 16:59 PST) running on java 1.4.1-beta

D:\bonin\aosd\code>ajc -argfile multicast/files.lst

D:\bonin\aosd\code>java multicast.MulticastListener
Hello World --- Multicast!

D:\bonin\aosd\code>
```



Legende:
Protocol `java multicast.MulticastListener` ↪ Protocol 231

Figure 7.15: `MulticastSender`-Example

7.21 Cellular Automaton



In computer science we usually asume that a complex output (behavior) is the result of a complex program. But a program with extremely simple construction can yield behavior of immense complexity. Cellular automata are such programs implementing simple rules. An important feature of them is that their behavior can readily be presented in a visual way. Stephen Wolfram shows in his book *A new Kind of Science* (↪ [Wolfram, 2002]) how their unexpected results force a whole new way of looking at the operation of our universe.

7.21.1 One-dimensional Automaton

A cellular automaton with simple rule that generates a pattern which seems in many respects random. The produced picture is an example of the fundamental phenomenon that even with simple underlying rules and simple initial conditions, it is possible to produce behavoir of great complexity (↪ [Wolfram, 2002] p. 27).

At the first step the cell in the center of the line is black (here represented with the string `@`) and all other cells are white. The on each successive step the rule is applied to make cells black. The figure 7.16 p. 234 shows the rule 30. For example a white cell will be black in the next step if the left and the right neighbor are black. ,

Argument list `files.lst`

```
Rule.java
```

`line`	1 1 1	1 1 0	1 0 1	1 0 0	0 1 1	0 1 0	0 0 1	0 0 0
`nextLine`	0	0	0	1	1	1	1	0
$\sum = 30$	$0 * 2^7$	$0 * 2^6$	$0 * 2^5$	$1 * 2^4$	$1 * 2^3$	$1 * 2^2$	$1 * 2^1$	$0 * 2^0$

<u>Legende:</u>
$\boxed{1}$ $\equiv$ black cell (■), in the output of `Automaton.java` String “@”
$\boxed{0}$ $\equiv$ white cell (□), in the output of `Automaton.java` String “ ”
The rules can be numbered from 0 to 255. Idea $\hookrightarrow$ [Wolfram, 2002] p. 53.

Figure 7.16: Cellular Automaton — Production Rule 30

```
Automaton.java
MyImgStore.java
ImgJpegStore.java
GlobalValue.java
```

Class `Rule`

```
/**
 *  "Cellular Automata"
 *
 *@since      10-Nov-2002
 *@author     Hinrich Bonin
 *@version    1.1
 */

package cellular;

public class Rule
{

   private int[][] condition = new int[8][3];
   private int[] action = new int[8];


   Rule(int[] action)
   {
      condition[0][0] = 1;
```

```
      condition[0][1] = 1;
      condition[0][2] = 1;
      this.action[0] = action[0];

      condition[1][0] = 1;
      condition[1][1] = 1;
      condition[1][2] = 0;
      this.action[1] = action[1];

      condition[2][0] = 1;
      condition[2][1] = 0;
      condition[2][2] = 1;
      this.action[2] = action[2];

      condition[3][0] = 1;
      condition[3][1] = 0;
      condition[3][2] = 0;
      this.action[3] = action[3];

      condition[4][0] = 0;
      condition[4][1] = 1;
      condition[4][2] = 1;
      this.action[4] = action[4];

      condition[5][0] = 0;
      condition[5][1] = 1;
      condition[5][2] = 0;
      this.action[5] = action[5];

      condition[6][0] = 0;
      condition[6][1] = 0;
      condition[6][2] = 1;
      this.action[6] = action[6];

      condition[7][0] = 0;
      condition[7][1] = 0;
      condition[7][2] = 0;
      this.action[7] = action[7];
   }


   public int produce(
```

```
         int leftValue,
         int middleValue,
         int rightValue)
   {
      int value = 0;
      boolean hit = false;

      for (int i = 0; i < 8; i++)
      {
         if ((condition[i][0] == leftValue) &
               (condition[i][1] == middleValue) &
               (condition[i][2] == rightValue))
         {
            value = action[i];
            hit = true;
            break;
         }
         ;
      }
      if (!hit)
      {
         System.err.println(
               "No production rule!");
         System.exit(1);
      }
      return value;
   }
}
```

Class `Automaton` Rule 30 is in the method `main()` implementet ↪ figure 7.16 234.

```
/**
 *  "Cellular Automaton"
 *
 *@since      10-Nov-2002
 *@author     Hinrich Bonin
 *@version    1.1
 */

package cellular;
```

```
import java.awt.*;
import java.util.Random;

public class Automaton
{

   private int[] line;
   private int[] nextLine;
   private int numberCells;
   private int middleCell;
   Random generator1 = new Random();
   Random generator2 = new Random();


   Automaton(int numberCells)
   {
      if (numberCells % 2 == 0)
      {
         this.numberCells = numberCells + 1;
      } else
      {
         this.numberCells = numberCells;
      }
      line = new int[this.numberCells];
      nextLine = new int[this.numberCells];
      for (int i = 0; i < this.numberCells; i++)
      {
         line[i] = 0;
         nextLine[i] = 0;
      }
      middleCell = this.numberCells / 2;
      this.line[middleCell] = 1;
   }


   private void setNextLineCell(Rule r, int step)
   {
      int i = middleCell;
      for (int k = 1; k <= step; k++)
      {
         nextLine[i] = r.produce(
               line[i - 1], line[i], line[i + 1]);
```

```
      nextLine[i - k] =
            r.produce(
            line[i - 1 - k], line[i - k],
            line[i + 1 - k]);
      nextLine[i + k] =
            r.produce(
            line[i - 1 + k], line[i + k],
            line[i + 1 + k]);
   }
   ;
}


private void changeNextLinetoLine()
{
   for (int i = 0; i < numberCells; i++)
   {
      line[i] = nextLine[i];
   }
}


private void printLine(
      Graphics2D g, String mySymbol,
      int startWidth, int startHeight)
{
   String s = "";

   for (int i = 0; i < numberCells; i++)
   {
      if (line[i] == 1)
      {
         /*
          *  Little joke!
          *  Instead of mySymbol
          *  the characters of bonin
          */
         int v = generator1.nextInt(5);
         if (v == 0)
         {
            s = s + "b";
         }
```

```
            if (v == 1)
            {
               s = s + "o";
            }
            if (v == 2)
            {
               s = s + "n";
            }
            if (v == 3)
            {
               s = s + "i";
            }
            if (v == 4)
            {
               s = s + "n";
            }
         } else
         {
            s = s + " ";
         }
      }
      int c = generator2.nextInt(2);
      if (c == 0)
      {
         g.setColor(Color.black);
      }
      if (c == 1)
      {
         g.setColor(Color.red);
      }
      g.drawString(s, startWidth, startHeight);
   }


   public void produce(
         Rule r,
         Graphics2D g,
         String mySymbol,
         int heightStep)
   {
      for (int i = 1; i < numberCells / 2; i++)
      {
```

```
         printLine(g, mySymbol, 0, i * heightStep);
         setNextLineCell(r, i);
         changeNextLinetoLine();
      }
   }

}
```

Aspect `GlobalValue`

```
/**
 *  "Cellular Automaton"
 *
 *@since      10-Nov-2002
 *@author     Hinrich Bonin
 *@version    1.1
 */

package cellular;

public aspect GlobalValue
{
   static int MyImgStore.FIGURE_WIDTH  = 5800;
   static int MyImgStore.FIGURE_HEIGHT = 3500;
   static int MyImgStore.NUMBER_CELLS  =  200;
   static int MyImgStore.FONT_SIZE     =   48;
}
```

Class `MyImgStore`

```
/**
 *  "Cellular Automaton"
 *
 *@since      10-Nov-2002
 *@author     Hinrich Bonin
 *@version    1.1
 */

package cellular;

import java.awt.*;

public class MyImgStore extends ImgJpegStore
```

```
{
   public static void main(String[] args)
   {
      try
      {
         MyImgStore mis = new MyImgStore();
         mis.store(
               FIGURE_WIDTH,
               FIGURE_HEIGHT,
               "./cellular/myPicture.jpg");
      } catch (Exception ex)
      {
         System.out.println(ex.getMessage());
         System.exit(1);
      }
      System.out.println("Image stored.");
      System.exit(0);
   }


   public void myPaintFunction(
         Graphics2D g,
         int width,
         int height,
         String imgFilename)
   {
      String mySymbol = "@";

      /*
       *  Background
       */
      g.setColor(Color.lightGray);
      g.fillRect(0, 0, width, height);

      g.setFont(new Font(
            "Courier", Font.PLAIN, FONT_SIZE));
      FontMetrics fm =
            g.getFontMetrics(g.getFont());
      /*
       *  getting two lines closer together
       */
```

```
      int heightStep =
           (int) ((float) fm.getHeight() / 1.8);
      /*
       *  All used symbols have the same width.
       */
      System.out.println(
            "heightStep: " +
            heightStep + "\n" +
            "blank width: " +
            fm.stringWidth(" ") + "\n" +
            "b width: " +
            fm.stringWidth("b") + "\n" +
            "o width: " +
            fm.stringWidth("o") + "\n" +
            "n width: " +
            fm.stringWidth("n") + "\n" +
            "i width: " +
            fm.stringWidth("i"));

      int[] action30 = {0, 0, 0, 1, 1, 1, 1, 0};
      Automaton a1 = new Automaton(NUMBER_CELLS);
      a1.produce(
            new Rule(action30),
            g, mySymbol, heightStep);
   }
}
```

Class `ImgJpegStore`

```
/**
 *  Erzeugung einer JPEG-Graphik;
 *
 *@since      16-Jan-2003
 *@version    1.0
 *@author     Hinrich Bonin
 */

package cellular;

import java.io.*;
import java.awt.*;
import java.awt.image.*;
```

```
import com.sun.image.codec.jpeg.*;

public abstract class ImgJpegStore
{
   public abstract void myPaintFunction(
         Graphics2D g,
         int width,
         int height,
         String imgFilename);


   public void store(
         int width,
         int height,
         String imgFilename)
          throws Exception
   {
      BufferedImage img =
            new BufferedImage(width, height,
            BufferedImage.TYPE_INT_RGB);
      myPaintFunction(
            img.createGraphics(),
            width, height,
            imgFilename);
      try
      {
         FileOutputStream out =
               new FileOutputStream(
               new File(imgFilename));
         JPEGImageEncoder enc =
               JPEGCodec.createJPEGEncoder(out);
         JPEGEncodeParam prm =
               enc.getDefaultJPEGEncodeParam(img);
         prm.setQuality(1.0f, false);
         enc.setJPEGEncodeParam(prm);
         enc.encode(img);
      } catch (Exception e)
      {
         throw new Exception(
               "\nError: Image storing to '" +
               imgFilename + "' failed: " +
               e.getMessage());
```

```
        }
    }
}
```

Protocol `Automaton.log`

```
C:\bonin\aosd\code>ajc -version
ajc version 1.0.6
  (built 24.07.2002 18:21 PST) running on java 1.4.0_01

C:\bonin\aosd\code>ajc -argfile cellular/files.lst

C:\bonin\aosd\code>java cellular.MyImgStore
java.lang.OutOfMemoryError
Exception in thread "main"

C:\bonin\aosd\code>java -Xms2046m -Xmx2046m cellular.MyImgStore
Error occurred during initialization of VM
Could not reserve enough space for object heap

C:\bonin\aosd\code>
C:\bonin\aosd\code>java -Xms512m -Xmx512m cellular.MyImgStore
heightStep: 35
blank width: 29
b width: 29
o width: 29
n width: 29
i width: 29
Image stored.

C:\bonin\aosd\code>
```

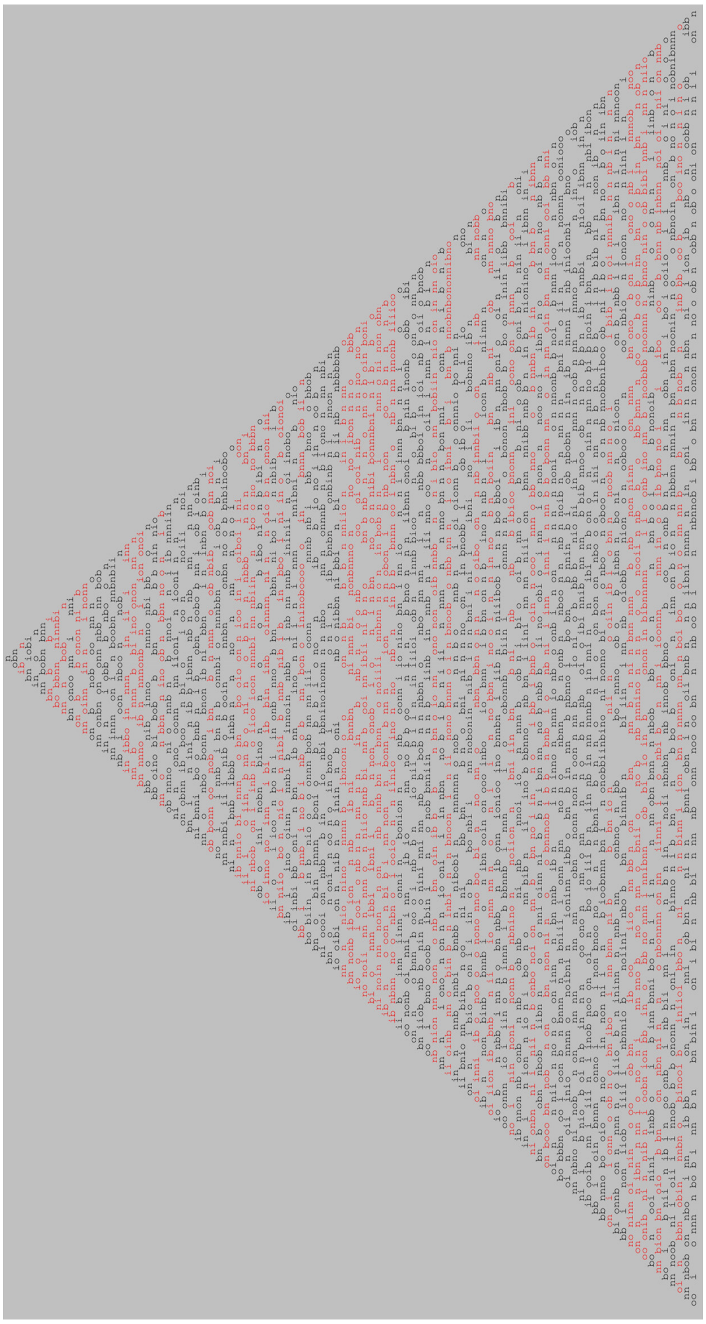


Figure 7.17: Cellular automaton (rule 30) with 200 steps of evolution

Legende:
Java application `MyImgStore.java` ↪ p. 240.

7.21.2 Two-dimensional Automaton

Here we implement a two-dimensional cellular automaton whose rule specifies that a particular cell should become black (here string @) if exactly one or all four of its neighbors were black on the previous step, but should otherwise stay the same color. (↪ [Wolfram, 2002] p. 171).

Argument list `files.lst`

```
Grid.java
Rule.java
```

Class `Rule`

```
/**
 *  "Two-dimensional Cellular Automaton"
 *
 *@author     Bonin
 *@version    1.0
 */

package grid;

public class Rule
{

   public static void rule4OrOnly1
         (String[][] squareGrid, int i, int j)
   {
      if (
      /*
       *  only if not @ then change if ..
       */
            !(squareGrid[i][j].equals("@")) &&
      /*
       *  all four neighbors were black
       */
            (((squareGrid[i - 1][j].equals("@")) &&
            (squareGrid[i + 1][j].equals("@")) &&
            (squareGrid[i][j - 1].equals("@")) &&
            (squareGrid[i][j + 1].equals("@"))) ||
      /*
       *  exactly one of its neighbors is black
       */
```

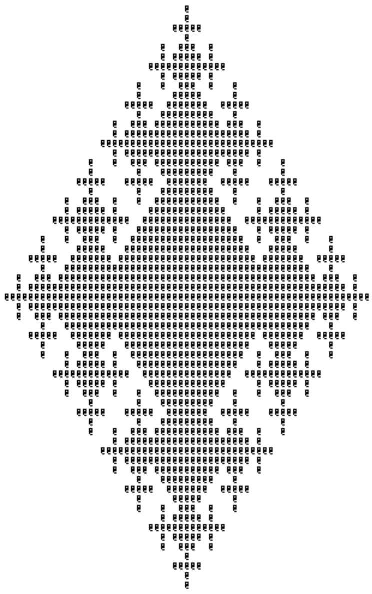
```
            (
            ((squareGrid[i - 1][j].equals("@")) &&
            !(squareGrid[i + 1][j].equals("@")) &&
            !(squareGrid[i][j - 1].equals("@")) &&
            !(squareGrid[i][j + 1].equals("@"))) ||
            (!(squareGrid[i - 1][j].equals("@")) &&
            (squareGrid[i + 1][j].equals("@")) &&
            !(squareGrid[i][j - 1].equals("@")) &&
            !(squareGrid[i][j + 1].equals("@"))) ||
            (!(squareGrid[i - 1][j].equals("@")) &&
            !(squareGrid[i + 1][j].equals("@")) &&
            (squareGrid[i][j - 1].equals("@")) &&
            !(squareGrid[i][j + 1].equals("@"))) ||
            (!(squareGrid[i - 1][j].equals("@")) &&
            !(squareGrid[i + 1][j].equals("@")) &&
            !(squareGrid[i][j - 1].equals("@")) &&
            (squareGrid[i][j + 1].equals("@")))
            )
            )
            )
      {
         squareGrid[i][j] = "n";
      }
   }
}
```

Class `Grid`

```
/**
 *  "Two-dimensional Cellular Automaton"
 *
 *@author     Bonin
 *@version    1.0
 */

package grid;

public class Grid
{

   private String[][] squareGrid;
   private int numberCells;
```

```
   private int middleCell;


   Grid(int numberCells)
   {
      if (numberCells % 2 == 0)
      {
         this.numberCells = numberCells + 1;
      } else
      {
         this.numberCells = numberCells;
      }
      squareGrid =
            new String[this.numberCells][this.numberCells];

      for (int i = 0; i < this.numberCells; i++)
      {
         for (int j = 0; j < this.numberCells; j++)
         {
            squareGrid[i][j] = " ";
         }
         ;
      }
      ;
      middleCell = this.numberCells / 2;
      this.squareGrid[middleCell][middleCell] = "@";
   }


   private void printGrid()
   {
      for (int i = 0; i < this.numberCells; i++)
      {
         String line = "";
         for (int j = 0; j < this.numberCells; j++)
         {
            line = line + squareGrid[i][j];
         }
         ;
         System.out.println(line);
      }
      ;
```

```
   }


   private void produce(int n)
   {
      for (int k = 0; k < n; k++)
      {
         for (int i = 1; i < this.numberCells - 1; i++)
         {
            for (int j = 1; j < this.numberCells - 1; j++)
            {
               Rule.rule4OrOnly1(squareGrid, i, j);
            }
            ;
         }
         ;
         for (int i = 0; i < this.numberCells; i++)
         {
            for (int j = 0; j < this.numberCells; j++)
            {
               if (squareGrid[i][j].equals("n"))
               {
                  squareGrid[i][j] = "@";
               }
               ;
            }
            ;
         }
         ;
      }
      ;
   }


   public static void main(String[] args)
   {
      Grid g = new Grid(80);
      g.produce(30);
      g.printGrid();
   }
}
```



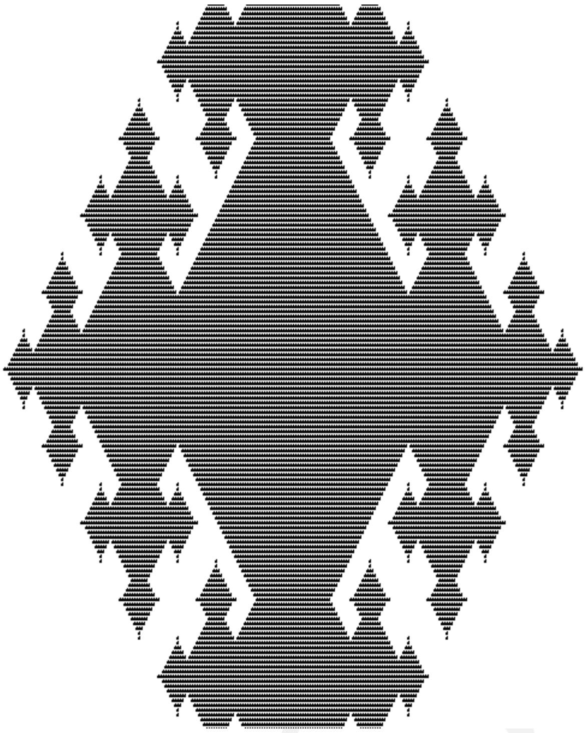
Legende:
Java application `Grid.java` ↪ p. 247.

Figure 7.18: Two-dimensional cellular automaton with 30 steps of evolution

Protocol `Grid.log`

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.5
  (built 27.06.2002 16:59 PST) running on java 1.4.1-beta

D:\bonin\aosd\code>ajc -argfile grid/files.lst

D:\bonin\aosd\code>java grid.Grid > grid/Grid.txt

D:\bonin\aosd\code>
```

A square grid with length 80 and 30 steps of evolution shows $\hookrightarrow$ figure 7.18 250.



7.22 Genetic Algorithm

Ch. Darwin

The Genetic Algorithm (GA) transforms one population of individuals into a new population of individuals using the principle of reproduction and survival of the fittest described by *Charles Darwin*[8]. The conventional GA operating on fixed-length character strings (L) can be described as follows (↪ [Koza, 1992] p. 28):

1. Randomly create an initial population of individual fixed-length character strings (X_i for example 011, 001, 110, 010 ↪ Table 7.2).
2. Iteratively perform the following substeps on the population of strings X_i until the termination criterion has been satisfied:
   (a) Evaluate the fitness $f(X_i)$ of each individual X_i in the population (in table 7.2 the integer value of the bits).
   (b) Create a new population of strings (generation 1) by applying at least the first two of the following three operations. The operations are applied to individual string(s) in the population (generation 0) chosen with a probability based on fitness.

   Reproduction: Copy existing individual strings to the new population (generation 0 → generation 1).

   Crossover: ≡ Sexual recombination: Create two new strings (≡ offsprings) for generation 1 by genetically recombining randomly chosen substrings from two existing strings (↪ table 7.3 p. 254).

   Mutation: Create a new string from an existing string by randomly mutating the character at one position in the string.
3. The best individual string that appeared in any generation (i. e. the best-so-far individual) is designated as the result of the genetic algorithm for the run. This result may represent a (approximate) solution to the problem.

Reproduction

In table 7.2 p. 254 the sum of the fitness values for all four individuals in the population is 12. The best-of-generation individual in the current pupulation (i. e., 110) has fitness 6. Therefore, the fraction of the population attributed to individual 110 is $\frac{1}{2}$. In fitness-proportionate selection, individual 110 is given a probability of $\frac{1}{2}$ of being selected for each of the four positions in the new population. In table 7.2 the string 110 occupy two of the four positions in the new population. The string 001 has only a probability $\frac{1}{12}$ of being selected and therefore it is absent in the new population. This resulting population is called the *mating pool*.

Crossover

The crossover (sexual recombination) operation begins by randomly selecting a number between 1 and $L - 1$ with $L \equiv$ length of the string. Each parent is then split at the crossover point into a crossover fragment an a remainder (↪ table 7.3 p. 254).

Mutation

The muatation operation is used very sparingly in the conventional genetic algorithm. It is an asexual operation because it operates on only one individual. It select randomly an individual and randomly a mutation point in the string (between 1 and L). The single character at the selected mutation point is changend. The mutation operation had the effect of increasing the genetic diversity of the population. The effect of

[8]Charles Darwin; *On the Origin of Species by Means of Natural Selection* (1859)

	Generation 0			Mating pool created after reproduction		After crossover Generation 1		
i	String X_i	Fitness $f(X_i)$	$\frac{f(X_i)}{\sum f(X_i)}$	Mating pool	Pool $f(X_i)$	Crossover point	X_i	$f(X_i)$
1	011	3	.25	011	3	2	111	7
2	001	1	.08	110	6	2	010	2
3	110	6	.50	110	6	—	110	6
4	010	2	.17	010	2	—	010	2
	Total	12			17			17
	Worst	1			2			2
	Average	3.00			4.25			4.25
	Best	6			6			7

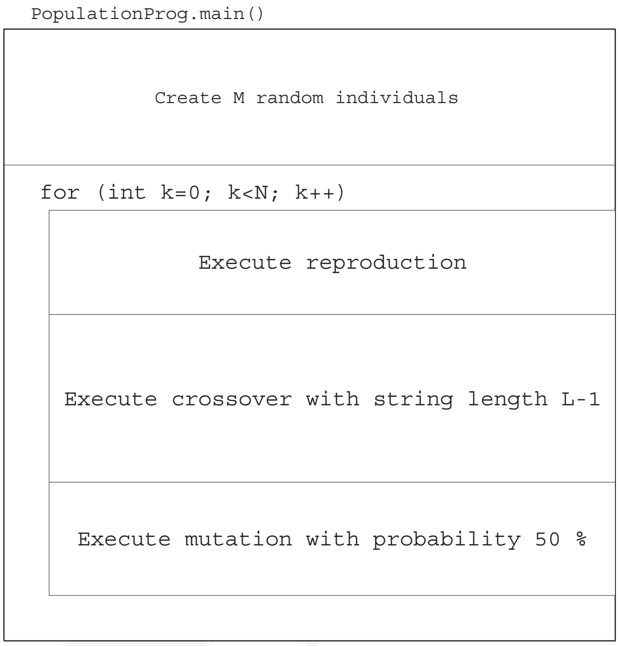
Legend: One possible outcome of applying the reproduction and crossover operations to generation 0 to create generation 1. Source↪ [Koza, 1992] p. 24.

Table 7.2: GA: Reproduction and crossover operations

Parent 1 011	Parent 2 110
Crossover fragment 1 01–	Crossover fragment 2 11–
Remainder 1 – – 1	Remainder 2 – – 0
Offspring 1 111	Offspring 2 010

Legend: Two parents selected proportionate to fitness ↪ table 7.2 p. 254. There are $L - 1 = 2$ interstitial locations lying between the positions of a string of length $L = 3$. Here the interstitial location 2 is selected.

Table 7.3: GA: Offsprings produced by crossover



Legende:
Java source code ↪ 29 p. 263
L ≡ string length
M ≡ numbers of individuals
N ≡ numbers of generations (iterations)

Figure 7.19: Conventional Genetic Algorithm — Structure Diagram

the reproduction operation may eliminate genetic diversity to the extent that the value 0 disappers from a position for the entire population. However, the global optimum may have a 0 in that position of the string.

The control structure of this conventional GA operating on a string of length L shows figure 7.19 p. 255.

Argument list `files.lst`

```
Individual.java
Population.java
PopulationProg.java
GlobalValue.java
```

Class `Individual`

```
/**
 *  "Genetic Programming"
 *
 *@author     Bonin
 *@version    1.0
 */

package genetic;

import java.util.Random;

public class Individual
{

   private String x = "";
   private int fitness = 0;

   private static Random Generator = new Random();


   Individual()
   {
      for (int i = 0; i < L; i++)
      {
         int bitValue =
               Individual.Generator.nextInt(2);
         x = bitValue + x;
      }
      computeFitness();
   }


   public String getX()
   {
      return x;
   }


   public void setX(String x)
   {
      this.x = x;
```

```
      computeFitness();
   }


   public int getFitness()
   {
      return fitness;
   }


   public void setFitness(int fitness)
   {
      this.fitness = fitness;
   }


   private void computeFitness()
          throws NumberFormatException
   {
      /*
       *  fitness = value of the
       *  "bits" of the string
       */
      String s = getX();
      int fit = 0;
      for (int i = 0; i < L; i++)
      {
         Integer j =
               new Integer(s.substring(i, i + 1));
         fit = fit + (int)
               (j.doubleValue() * Math.pow(2, L - 1 - i));
      }
      setFitness(fit);
   }
}
```

Class Population

```
/**
 *  "Genetic Programming"
 *
 *@author     Bonin
```

```
 *@version    1.0
 */

package genetic;

import java.util.Random;

public class Population
{
   private Individual[] individual;
   private int m = 0;
   private int totalFitness = 0;
   private int worstFitness = 0;
   private double averageFitness = 0.0;
   private int bestFitness = 0;

   private static Random Generator = new Random();


   public int getM()
   {
      return m;
   }


   public int getTotalFitness()
   {
      return totalFitness;
   }


   public void setTotalFitness(int totalFitness)
   {
      this.totalFitness = totalFitness;
   }


   public int getWorstFitness()
   {
      return worstFitness;
   }
```

```
public void setWorstFitness(int worstFitness)
{
   this.worstFitness = worstFitness;
}


public double getAverageFitness()
{
   return averageFitness;
}


public void setAverageFitness(double averageFitness)
{
   this.averageFitness = averageFitness;
}


public int getBestFitness()
{
   return bestFitness;
}


public void setBestFitness(int bestFitness)
{
   this.bestFitness = bestFitness;
}


public Population(int m)
{
   individual = new Individual[m];
   this.m = m;
   for (int i = 0; i < m; i++)
   {
      individual[i] = new Individual();
   }
   this.computeFitness();
}
```

```
private void computeFitness()
{
   int f = 0;
   int fit = 0;
   for (int i = 0; i < getM(); i++)
   {
      f = individual[i].getFitness();
      if (i == 0)
      {
         setWorstFitness(f);
      }
      if (f < getWorstFitness())
      {
         setWorstFitness(f);
      }
      if (f > getBestFitness())
      {
         setBestFitness(f);
      }
      fit = fit + f;
   }
   setTotalFitness(fit);
   setAverageFitness(((double) getTotalFitness()) /
         ((double) getM()));
}


public void showIndividuals()
{
   for (int i = 0; i < getM(); i++)
   {
      System.out.println("X" + (i + 1) + ": " +
            individual[i].getX() +
            " Fitness: " + individual[i].getFitness());
   }
}


public void showFitness()
{
   System.out.println(
```

```
                "Total: " + getTotalFitness() +
                " Worst: " + getWorstFitness() +
                " Best: " + getBestFitness() +
                " Average: " + getAverageFitness());
   }


   public void reproduction()
   {
      /*
       *  replace last worst individual
       *  by last best individual
       */
      int wP = 0;
      int bP = 0;
      for (int i = 0; i < getM(); i++)
      {
         if (getWorstFitness() ==
               individual[i].getFitness())
         {
            wP = i;
         }
         if (getBestFitness() ==
               individual[i].getFitness())
         {
            bP = i;
         }
      }
      individual[wP].setX(new String(individual[bP].getX()));
      computeFitness();
   }


   public void crossover()
   {
      /*
       *  sexual recombination of
       *  first and second individual
       */
      Individual parent1 = individual[0];
      Individual parent2 = individual[1];
      String fragment1 =
```

```
          parent1.getX().substring(0, Individual.L - 1);
    String remainder1 =
          parent1.getX().substring(Individual.L - 1, Individual.L);
    String fragment2 =
          parent2.getX().substring(0, Individual.L - 1);
    String remainder2 =
          parent2.getX().substring(Individual.L - 1, Individual.L);
    String offspring1X = fragment2 + remainder1;
    String offspring2X = fragment1 + remainder2;
    individual[0].setX(new String(offspring1X));
    individual[1].setX(new String(offspring2X));
    computeFitness();
 }


 public void mutation()
 {
    /*
     *  change of one random position
     *  of one random individual
     */
    String s = "";
    int indiv = Population.Generator.nextInt(getM());
    int position = Population.Generator.nextInt(Individual.L);

    for (int i = 0; i < getM(); i++)
    {
       if (i == indiv)
       {
          s = individual[i].getX();
          String sub = s.substring(position, position + 1);
          if (sub.equals("1"))
          {
             sub = "0";
          } else
          {
             sub = "1";
          }
          String sMutation =
                s.substring(0, position) + sub +
                s.substring(position + 1, Individual.L);
          individual[i].setX(new String(sMutation));
```

```
            break;
         }
      }
      computeFitness();
   }
}
```

Aspect `GlobalValue`

```
/**
*   "Genetic Programming"
*@author Bonin
*@version 1.0
*/

package genetic;

public aspect GlobalValue
{
    /* For crossover string length
       L must be greater 1 */
    final static int Individual.L = 3;

    /* Numbers of individuals */
    final static int PopulationProg.M = 4;

    /* Numbers of iterations */
    final static int PopulationProg.N = 2;
}
```

Class `PopulationProg`

```
/**
 *  "Genetic Programming"
 *
 *@author     Bonin
 *@version    1.0
 */

package genetic;

import java.util.Random;

public class PopulationProg
```

```
{

   private static Random Generator = new Random();


   public static void main(String[] args)
   {

      /*
       *  Initial random population
       */
      Population generation = new Population(M);
      System.out.println("Generation 0");
      generation.showIndividuals();
      generation.showFitness();

      /*
       *  Compute N new generations based on
       *  reproduction, crossover and mutation
       */
      for (int k = 0; k < N; k++)
      {

         /*
          *  reproduction probability 100 %
          */
         generation.reproduction();
         System.out.println("Reproduction: " + k);
         generation.showIndividuals();
         generation.showFitness();

         /*
          *  crossover probability 100 %
          */
         generation.crossover();
         System.out.println("Crossover: " + k);
         generation.showIndividuals();
         generation.showFitness();

         /*
          *  mutation probability 50 %
          */
```

```
            if (PopulationProg.Generator.nextInt(2) > 0)
            {
               generation.mutation();
               System.out.println("Mutation: " + k);
               generation.showIndividuals();
               generation.showFitness();
            }
         }
      }
   }
```

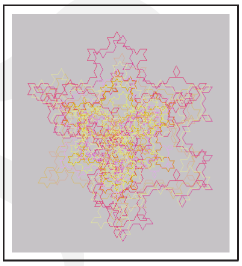
Protocol `PopulationProg.log`

```
D:\bonin\aosd\code>ajc -version
ajc version 1.0.5
  (built 27.06.2002 16:59 PST) running on java 1.4.1-beta

D:\bonin\aosd\code>ajc -argfile genetic/files.lst

D:\bonin\aosd\code>java genetic.PopulationProg
Generation 0
X1: 011 Fitness: 3
X2: 111 Fitness: 7
X3: 111 Fitness: 7
X4: 001 Fitness: 1
Total: 18 Worst: 1 Best: 7 Average: 4.5
Reproduction: 0
X1: 011 Fitness: 3
X2: 111 Fitness: 7
X3: 111 Fitness: 7
X4: 111 Fitness: 7
Total: 24 Worst: 3 Best: 7 Average: 6.0
Crossover: 0
X1: 111 Fitness: 7
X2: 011 Fitness: 3
X3: 111 Fitness: 7
X4: 111 Fitness: 7
Total: 24 Worst: 3 Best: 7 Average: 6.0
Mutation: 0
X1: 111 Fitness: 7
X2: 011 Fitness: 3
X3: 110 Fitness: 6
X4: 111 Fitness: 7
Total: 23 Worst: 3 Best: 7 Average: 5.75
Reproduction: 1
```

```
X1: 111 Fitness: 7
X2: 111 Fitness: 7
X3: 110 Fitness: 6
X4: 111 Fitness: 7
Total: 27 Worst: 6 Best: 7 Average: 6.75
Crossover: 1
X1: 111 Fitness: 7
X2: 111 Fitness: 7
X3: 110 Fitness: 6
X4: 111 Fitness: 7
Total: 27 Worst: 6 Best: 7 Average: 6.75

D:\bonin\aosd\code>
```

Appendix A

Software Engineering Tools



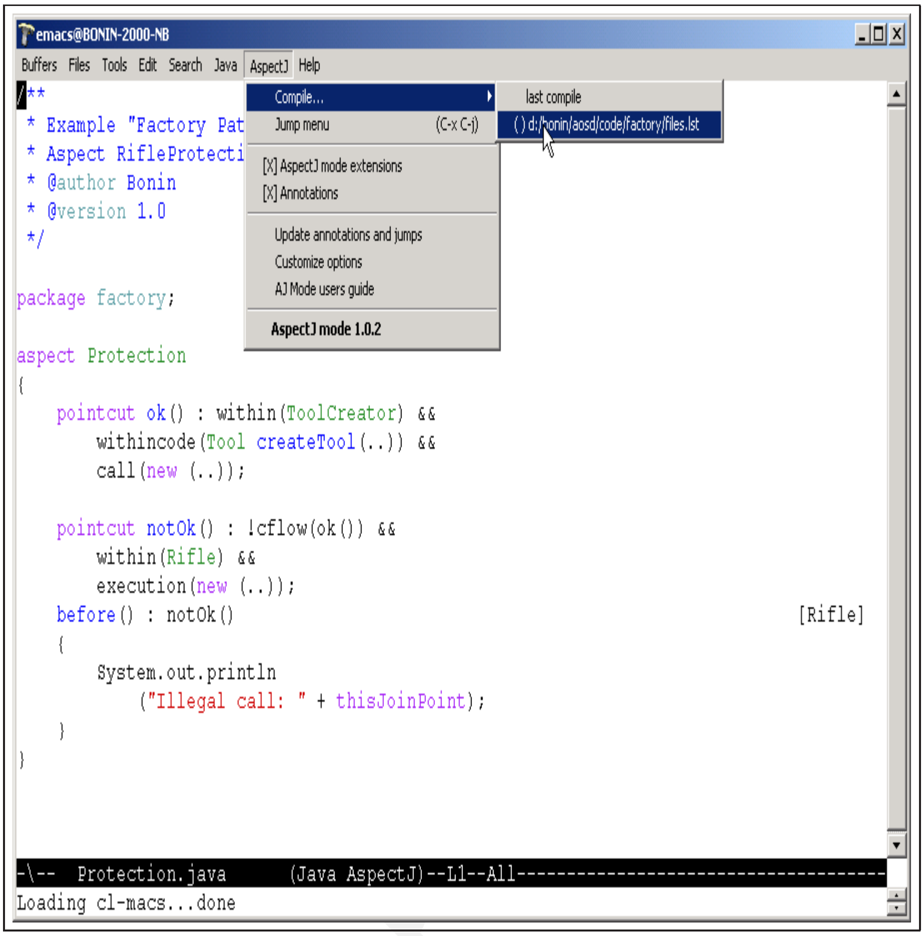
A.1 Emacs AspectJ-mode

Emacs

Emacs[1] *AspectJ minor mode* provides (↪ figure A.1, p. 268):

- Highlighting of AspectJ keywords and declaration names.
- Source code annotation of introduction and advice declarations, as well as the code they affect.
- AspectJ-style compilation, using files.`lst` to generate a compilation submenu.
- Viewing and navigation of aspect structures, permitting navigation between aspect code and the code that it affects, via a jump' menu (and in the speedbar and Classes menu for JDE users).

[1] AspectJ mode requires the installation of GNU Emacs 20.3.1 (↪ `http://www.gnu.org/software/emacs/`), XEmacs 21.1.14 (Unix & Linux)(↪ `http://www.xemacs.org/`), XEmacs 21.4 (Windows) (↪ `http://www.xemacs.org/`), or higher.



Legende:
Software engineering tool *GNU Emacs* (Version 20.7.1) with AspectJ

Figure A.1: Emacs AspectJ mode 1.0.2

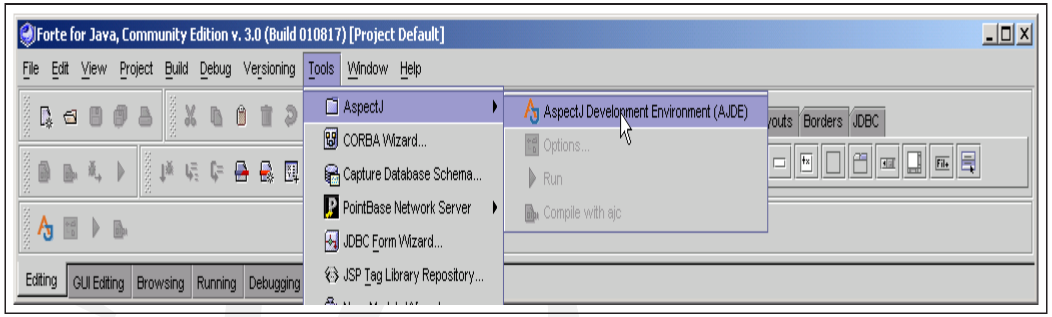
A.2 AJDE support for *Forte*

Forte

The AspectJ Development Environment (AJDE) support for Forte module extension to *Sun's Forte for Java*[2] will allow us to (↪ figure A.2, p. 270):

- compile AspectJ and Java files within the IDE,
- browse the structure of our AspectJ program, and
- set up a compile configuration that determines which files will be passed to the compiler.

[2] Sun's Forte for Java ↪ `http://www.sun.com/forte/ffj`. For release-specific documentation refer to the changes file (↪ `http://aspectj.org/doc/dist/changes.html`.)



Legende:
AspectJ Development Environment (AJDE) support for Forte module extension to *Sun's Forte for Java*

Figure A.2: AJDE support for *Forte*

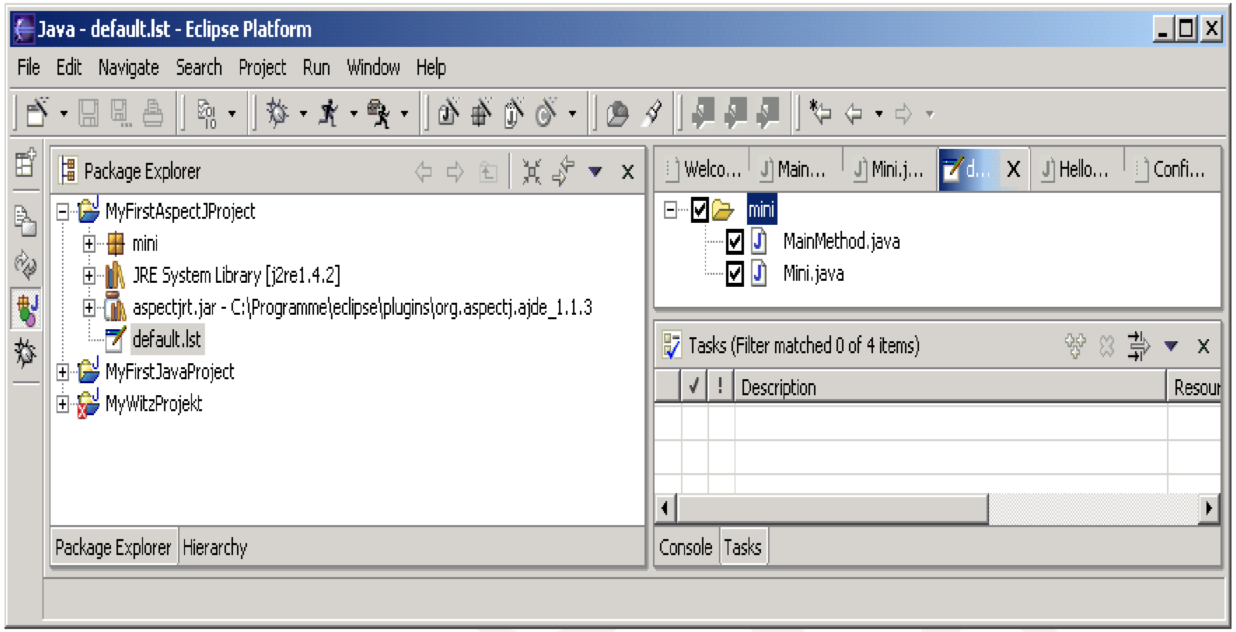
A.3 AJDE support for *Eclipse*

The IDE *Eclipse* (Version: 2.1.1), (c) Copyright IBM Corp. 2003.

↪ `http://www.eclipse.org/platform` (online 12-Oct-2003)

includes the plug-ins *AspectJ Development Tools* 1.1.3 (`org.eclipse.aspectj`) and *AspectJ Development Tools (AJDT) – (UI)* 0.6.3 `org.eclipse.ajdt.ui`. This product includes also software developed by the *Apache Software Foundation*

↪ `http://www.apache.org/` (online 12-Oct-2003).



Legende:
IDE *Eclipse* Version: 2.1.1, (c) Copyright IBM Corp. 2003. ↪`http://www.eclipse.org/platform`(online 12-Oct-2003).

Figure A.3: AJDE support for *Eclipse*

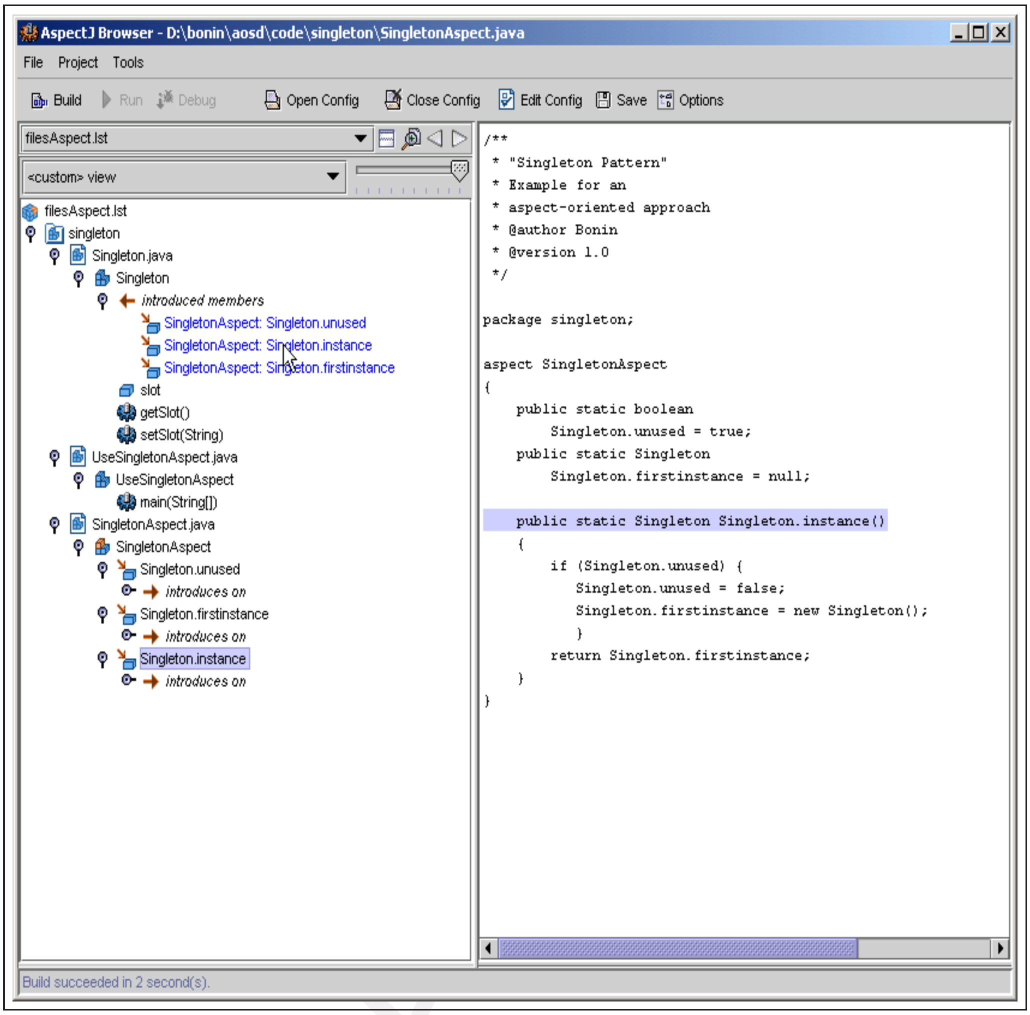
Attention: When you use the update manager for the AspectJ plugin you should close all projects before. After the upgrading restart Eclipse. Then you have to remove the `AspectJ nature` (from the context menu) of any existing AspectJ projects. Next you re-convert the projects to an AspectJ project (again, from the context menu). So the projects are running the new AJDT plugin.

A.4 AspectJ Browser

The AspectJ Browser is a development tool that will allow us (↪ figure A.4, p. 274): `ajbrowser`

- to compile using `ajc`,
- navigate our program's static structure,
- edit source files, and
- graphically edit build configuration files.

To use the browser launch it by typing `ajbrowser` and pass one or more build configuration files`.lst` as command line parameters to the browser in order to build them and navigate the corresponding structure. To compile click the "Build button". Select nodes in the program structure by clicking them. If one node is related to one or more nodes by an association the name of the association will appear below that node and will be displayed in italics. Links to other structure nodes appear in blue below the association. If there is no corresponding source for the link, it will appear in light-blue.



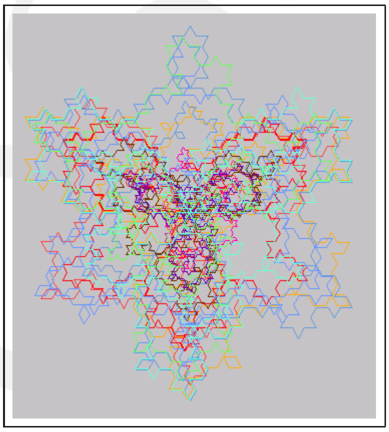
<u>Legende:</u>
The *AspectJ Browser* is a tool for compiling programs with `ajc` and navigating the crosscutting structure (early-access).

```
D:\bonin\aosd>ajbrowser ./code/singleton/filesAspect.lst
```

Figure A.4: AspectJ Browser

Appendix B

AspectJ Quick Reference



The following text belongs to the AspectJ contribution (`ajc version 1.0.1`), located in path `doc\progguide`.

Pointcut Designators

Methods and Constructors

- `call(`Signature`)` ≡ Method or constructor call join points when the signature matches Signature.
- `execution(`Signature`)` ≡ Method or constructor execution join points when the signature matches Signature.
- `initialization(`Signature`)` ≡ Object initialization join point when the first constructor called in the type matches Signature.

Exception Handlers

* `handler(`TypePattern`)` ≡ Exception handler execution join points when try handlers for the throwable types in TypePattern are executed. The exception object can be accessed with an `args` pointcut.

Fields

* `get(`Signature`)` ≡ Field reference join points when the field matches Signature.
* `set(`Signature`)` ≡ Field assignment join points when the field matches Signature. The new value can be accessed with an `args` pointcut.

Static Initializers

* `staticinitialization(`TypePattern`)` ≡ Static initializer execution join points for the types in TypePattern.

Objects

* `this(`TypePattern`)` ≡ Join points when the currently executing object is an instance of a type in TypePattern.
* `target(`TypePattern`)` ≡ Join points when the target object is an instance of a type in TypePattern.
* `args(`TypePattern, ...`)` ≡ Join points when the argument objects are instances of the TypePatterns.

Lexical Extents

* `within(`TypePattern`)` ≡ Join points when the code executing is defined in the types in TypePattern.
* `withincode(`Signature`)` ≡ Join points when the code executing is defined in the method or constructor with signature Signature.

Control Flow

* `cflow(`Pointcut`)` ≡ Join points in the control flow of the join points specified by Pointcut.
* `cflowbelow(`Pointcut`)` ≡ Join points in the control flow below the join points specified by Pointcut.

Conditional

* `if(`Expression`)` ≡ Join points when the boolean Expression evaluates to `true`.

Combination

* `!` Pointcut ≡ Join points that are not picked out by Pointcut.

* Pointcut$_0$ && Pointcut$_1$ ≡ Join points that are picked out by both Pointcut$_0$ and Pointcut$_1$.
* Pointcut$_0$ `||` Pointcut$_1$ ≡ Join points that are picked out by either Pointcut$_0$ or Pointcut$_1$.
* `(`Pointcut`)` ≡ Join points that are picked out by the parenthesized Pointcut

Type Patterns

Type Name Patterns

* `*` alone ≡ all types
* `*` in an identifier ≡ any sequence of characters, not including ”.”
* `..` in an identifier ≡ any sequence of characters starting and ending with ”.”
* The `+` wildcard can be appended to a type name pattern to indicate all subtypes.
* Any number of `[]`s can be put on a type name or subtype pattern to indicate array types.

Type Patterns

* TypeNamePattern ≡ all types in TypeNamePattern
* SubtypePattern ≡ all types in SubtypePattern, a pattern with a `+`.
* ArrayTypePattern ≡ all types in ArrayTypePattern, a pattern with one or more `[]`s.
* `!`TypePattern ≡ all types not in TypePattern
* TypePattern$_0$ `&&` TypePattern$_1$ ≡ all types in both TypePattern$_0$ and TypePattern$_1$.
* TypePattern$_0$ `||` TypePattern$_1$ ≡ all types in either TypePattern$_0$ or TypePattern$_1$.
* `(`TypePattern`)` ≡ all types in TypePattern

Advice

- `before(`Formals`) :` ≡ Run before the join point.
- `after(`Formals`) returning [ (` Formal `) ] :` ≡ Run after the join point if it returns normally. The optional formal gives access to the returned value.

– `after(`Formals`) throwing [ (`Formal `) ] :` ≡ Run after the join point if it throws an exception. The optional formal gives access to the `Throwable` exception value.
– `after(`Formals`) :` ≡ Run after the join point both when it returns normally and when it throws an exception.
– Type `around(`Formals`) [ throws` TypeList `] :` ≡ Run instead of the join point. The join point can be executed by calling `proceed`.

Static Crosscutting

Introduction

* Modifiers Type TypePattern.Id(Formals) { Body }; ≡ Defines a method on the types in TypePattern
* `abstract` Modifiers Type TypePattern.Id(Formals); ≡ Defines an abstract method on the types in TypePattern.
* Modifiers TypePattern`.new`(Formals){ Body }; ≡ Defines a a constructor on the types in TypePattern.
* Modifiers Type TypePattern.Id `[ =` Expression `];` ≡ Defines a field on the types in TypePattern.

Other declarations

* `declare parents:` TypePattern `extends` TypeList ≡ Declares that the types in TypePattern extend the types of TypeList.
* `declare parents:` TypePattern `implements` TypeList ≡ Declares that the types in TypePattern implement the types of TypeList.
* `declare warning:` Pointcut: String; ≡ Declares that if any of the join points in Pointcut possibly exist in the program, the compiler should emit a warning of String.
* `declare error:` Pointcut: String; ≡ Declares that if any of the join points in Pointcut possibly exist in the program, the compiler should emit an error of String.
* `declare soft:` TypePattern: Pointcut; ≡ Declares that any exception of a type in TypePattern that gets thrown at any join point picked out by Pointcut will be wrapped in `org.aspectj.lang.SoftException`.

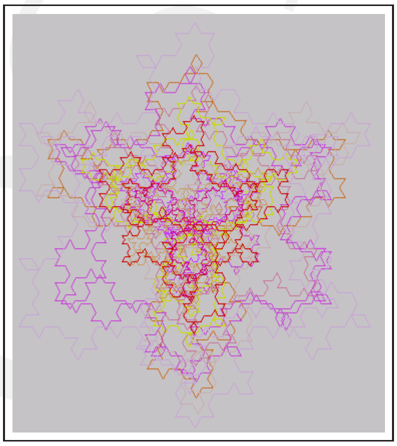
Aspect Associations

– `[ issingleton ]` ≡ One instance of the aspect is made. This is the default.
`aspectOf()` ≡ at all join points.

- `perthis(`Pointcut`)` ≡ An instance is associated with each object that is the currently executing object at any join point in Pointcut.
  `aspectOf(Object)` ≡ at all join points.
- `pertarget(`Pointcut`)` ≡ An instance is associated with each object that is the target object at any join point in Pointcut.
  `aspectOf(Object)` ≡ at all join points.
- `percflow(`Pointcut`)` ≡ The aspect is defined for each entrance to the control flow of the join points defined by Pointcut.
  `aspectOf()` ≡ at join points in `cflow(`Pointcut`)`.
- `percflowbelow(`Pointcut`)` ≡ The aspect is defined for each entrance to the control flow below the join points defined by Pointcut.
  `aspectOf()` ≡ at join points in `cflowbelow(`Pointcut`)`.

Appendix C

Resources



C.1 Web Sites

C.1.1 Java

James Gosling / Bill Joy /Guy Steele /Gilad Bracha; *The Java Language Specification, Second Edition*
↪ `http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html`
(visited June 2002)

C.1.2 AspectJ

`http://aspectj.org`

C.1.3 Aspect-Oriented Software Development

`http://www.aosd.net` (visited January 2002)

C.1.4 DJ Library (Demeter/Java Project)

`http://www.ccs.neu.edu/research/demeter/DJ` (visited January 2002)

C.1.5 HyperJ: Multi-Demensional Separation of Concerns

`http://www.research.ibm.com/hyperspace/HyperJ/HyperJ.htm`(visited February 2002)

C.1.6 Java Syntactic Extender

`http://www.ai.mit.edu/~jrb/jse` (visited January 2002)

C.1.7 BCEL API (Byte Code Engineering Libary)

`http://jakarta.apache.org/bcel/manual.html` (visited June 2002)

C.1.8 GJ (A Generic Java Language Extension)

Philip Wadler / Martin Odersky / Gilad Bracha / Dave Stoutamire; Sun releases prototype for adding generics to Java, based on GJ (May 2001).
`http://www.research.avayalabs.com/user/wadler/pizza/gj/` (visited June 2002)

C.1.9 Updates to this Book

`http://as.fhnon.de/aosd/updates.html` (visited January 2002)

C.2 Glossary

abstract class A class whose primary purpose is to define an interface. It defers some or all of its implementation to subclasses. It cannot be instantiated.

AJDE AspectJ Development Environment

AJDT AspectJ Development Tools project is a set of plugins for *Eclipse* that provide support for aspect-oriented software development using AspectJ within the Eclipse IDE.

AOP The term aspect-oriented programming is attributed to Kiczales et. a.

API Application programming interface

BCEL Byte Code Engineering Libary

class It specifies the object's internal data and representation and defines the operations the object can perform. It defines an object's interface (↪ p. 283) and implementation.

CLOS Common Lisp Object System

concern Properties or areas of interest of a system.

delegation An implementation mechanism in which an object (↪ p. 284) forwards or delegates a request to another object. The delegate carries out the request on behalf of the original object.

encapsulation The result of hiding a representation and implementation in an object (↪ p. 284). The representation is not visible and cannot be accessed directly from outside the object. Operations are the only way to access and modify an object's representation.

framework A set of cooperating classes (↪ p. 283) that makes up a reusable design for a specific kind of software. A framework provides architectural guidance by partitioning the design into abstract classes (↪ p. 282) and defining their responsibilities and collaborations. A developer customizes the framework to a particular application by subclassing and composing instances of framework classes.

GA Genetic Algorithm

GJ A Generic Java Language Extension

IANA Internet Assigned Numbers Authority

inheritance A relationship that defines one entity in terms of another. Class (↪ p. 283) inheritance combines interface (↪ p. 283) inheritance and implementation inheritance. Interface inheritance defines a new interface in terms of one or more existing interfaces. Implementation inheritance defines a new implementation in terms of one or existing implementations.

instance Objects (↪ p. 284) are created by instantiating a class. The object is said to be an instance of the class.

instantiating The process of instantiating a class (↪ p. 283) allocates storage for the object's internal data (made up of instance variables) and associates the operations with these data. Many similar instances of an object can be created by instantiating a class.

interface The set of all signatures (↪ p. 284) defined by an object's operations is called the interface to the object. An object's interface says nothing about its implementation.

J2EE Java 2 Enterprise Edition

J2ME Java 2 Micro Edition

J2SE Java 2 Standard Edition

JMS Java Messaging Service

JXTA Juxtapose (pronounced *juxta* — Peer-to-Peer-Framework)

MDA OMG's Model Driven Architecture

metaclass It is the class (↪ p. 283) of a class object.

message An object (↪ p. 284). performs an operation when it receives a corresponding message from another object. A common synonym for message is request.

MOF OMG's Meta-Object Facility

MOP Metaobject protocol; a programmer could override the default behavior of the dispatch method in order to affect what happens when a virtual function is called.

object A run-time entity that packages both data and the procedures that operate on that data.

OMG Object Management Group

OOP Currently, the dominant programming paradigm is object-oriented programming.

P2P Peer-to-Peer

polymorphism The ability to substitute objects of matching interface (↪ p. 283) for one another at run-time.

POP Post object-oriented programming

protocol Extends the concept of an interface (↪ p. 283) to include the allowable sequences of requests (↪ p. 284).

receiver The target object (↪ p. 284) of a request (↪ p. 284).

request A common synonym for request is message (↪ p. 284).

signature Every operation declared by an object specifies the operation's name, the objects it takes as parameters, and the operation's return value. This is known as the operation's signature.

SOC Separation of concerns

toolkit A collection of classes (↪ p. 283) that provides useful functionality but does not define the design of an application.

type It is a name used to denote a particular interface (↪ p. 283). Part of an object's interface may be characterized by one type, and other parts by other types.

UML Unified Modeling Language

XMI XML Metadata Interchange format

XML Extensible markup language

C.3 Bibliography

Bibliography

[Alexander+, 1977] Christopher Alexander / Sara Ishikawa / Murray Silverstein / Max Jacobson / Ingrid Fiksdahl-King / Shlomo Angel; A Pattern Language, New York (Oxford University Press), 1977.

[Arnold / Gosling, 1996] Ken Arnold / James Gosling; The Java Programming Language (Addison-Wesley) 1996.

[Aßmann, 2003] Uwe Aßmann; Invasive Software Composition (Springer) 2003, ISBN 3-540-44385-1.

[Bonin, 1991] Hinrich E. G. Bonin; Software-Konstruktion mit LISP, Berlin New York (Walter de Gruyter), 1991. (german)

[Bonin, 1992b] Hinrich E. G. Bonin; Teamwork between Non-Equals — Check-in & Check-out model for Producing Documents in a Hierarchy, in: SIGOIS Bulletin, Volume 13, Number 3, December 1992, (ACM Press), pp. 18–27.

[Bonin, 1992c] Hinrich E. G. Bonin; Object-Orientedness – a New Boxologie, FINAL Vol. 3, 1992 (ISSN 0939-8821).

[Booch, 1994] G. Booch; Object-oriented analysis and design with applications, 2nd ed., Red1wood City (Benjamin/Cummings), 1994.

[Bobrow / Moon 1988] Daniel G. Bobrow / David Moon u. a.; Common Lisp Object Systems Specification, ANSI X3J13 Document 88-002R, American National Standards Insitute, Washington, DC, June 1988 (published in: SIG-PLAN Notices, Band 23, Special Issus, September 1988).

[Bracha et al., 1998] Gilad Bracha / Martin Odersky / David Stoutamire / Philip Wadler; GJ: Extending the Java^{TM} programming language with type parameters, March 1998, revised August 1998, `http://www.research.avayalabs.com/user/wadler/pizza/gj/` (visited June 2002)

[Bouvier, 1999] Dennis J. Bouvier; Getting Started with the Java3D^{TM} API, tutorial v.1.5.1 (Java 3D API v1.2), Sun Microsystems, 1999–2002, `http://java.sun.com/products/java-media/3D/collateral/`.

[Clocksin / Mellish, 1987] W. F. Clocksin / C. S. Mellish; Programming in Prolog, Berlin New York u.a. (Springer-Verlag) Third Edition, 1987.

[Cooper, 2003] James W. Cooper; Aspects, Concerns, and Java — AspectJ may take some getting used to, but aspect-oriented programming can be a nice complement to object-oriented programming, JavaPro, `http://www.fawcette.com/javapro/2003_03/magazine/columns/javatecture/default_pf.asp` (visited 23-May-2003)

[Elrad+, 2001] Tzilla Elrad, Moderator / Mehmet Aksit / Gregor Kiczales / Karl Lieberherr / Harold Ossher; Discussing Aspects of AOP, in: Communications of the ACM (Association for Computing Machinery) October 2001/Vol. 44, No. 10, pp. 33–38.

[DeRemer / Kron, 1976] F. DeRemer / H. Kron; Programming in the Large vs. Programming in the Small, in: IEEE Transactions on Software Engineering, 2(2), 1976, pp. 80–86.

[Elrad et al., 2001a] Tzilla Elrad / Robert E. Filman / Atef Bader, guest editors; Aspect-oriented Programming, in: Communications of the ACM (Association for Computing Machinery) October 2001/Vol. 44, No. 10, pp. 29–32.

[FINAL] Fachhochschule Nordostniedersachsen, Informatik, Arbeitsberichte, Lüneburg (FINAL) editor: Hinrich E. G. Bonin, ISSN 0939-8821, since 1997 with CD-ROM, publisher FH NON, Volgershall 1, D-21339 Lüneburg, Germany.

[Flenner et al., 2003] Robert Flenner / Michael Abbott / Toufic Boubez / Frank Cohen / Navaneeth Krishnan / Alan Moffet / Rajam Ramamurti / Bilal Siddiqui / Frank Sommers; Java^{TM} P2P Unleashed — with JXTA, Web Services, XML, Jini^{TM}, JavaSpaces^{TM}, and J2EE, Indianapolis (Sams Publishing) 2003, ISBN 0-672-32399-0.

[Gabriel et al., 1991] Richard P. Gabriel / John L. White / Daniel G. Bobrow; CLOS: Integrating Object-Oriented and Functional Programming, in: Communications of the ACM, Vol. 34, No 9, September 1991, pp. 29–38.

[Gamma+, 1994] Erich Gamma / Richard Helm / Ralph Johnson / John Vlissides; Design Patterns — Elements of Reusable Object-Oriented Software, foreword by Grady Booch, Professional Computing Series (Addison-Wesley), ISBN 0-201-63361-2.

[Genssler / Kuttruff, 2001] T. Genssler / V. Kuttruff; Werkzeugunterstützte Softwareadaption mit Inject/J, in: Reengineering-Workshop 2001, Gesellschaft für Informatik e. V. Bonn (GI), Germany, `http://injectj.sourceforge.net/` (visted 7-Jul-2003)

[Goldberg, 1983] Adele Goldberg; Smalltalk-80: The Interactive Programming Environment, Reading 1983 (Addison-Wesley)

[Goldberg / Robson, 1983] Adele Goldberg / Dave Robson; Smalltalk-80: the lanugage, Reading, Massachusetts u. a. (Addision-Wesley) 1983.

[Goldfarb / Prescod, 2002] Charles F. Goldfarb / Paul Prescod; Charles F. Goldfarb's XML Handbook, fourth Edition 2002, (Prentice Hall PTR), ISBN 0-13-065198-2.

[Hannemann / Kiczales, 2002] Jan Hannemann / Gregor Kiczales; Design Pattern Implementation in Java and AspectJ, OOPSLA 2002, November 4–8, Seattle Washington, USA, source: `http://www.cs.ubc.ca/labs/spl/projects/aodps.html` (visted 11-Jun-2003)

[Horstmann, 2000] Cay S. Horstmann; Computing Concepts with Java 2 Essentials, Second Edition, New York u. a. (John Wiley & Sons, Inc.), ISBN 0-471-34609-8.

[JavaSpec] James Gosling / Bill Joy / Guy Steele; The Java Language Specification, Sun Microsystems, 2000; source: `http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html` (visited 4-Feb-2002)

[Gradecki+, 2003] Joseph D. Gradecki / Nicholas Lesiecki; Mastering AspectJ^{TM} — Aspect-Oriented Programming in Java, Indianapolis (Wiley Publishing, Inc.) 2003, ISBN 0-471-43104-4.

[Gradecki, 2002] Joseph D. Gradecki; Mastering JXTA — Building Java Peer-to-Peer Applications, Indianapolis (Wiley Publishing, Inc.) 2002, ISBN 0-471-25084-8.

[Jacobsen+, 1992] Ivar Jacobsen / M. Christerson / P. Jonsson / G. Övergaard; Object-Oriented Software Engineering, A Use Case Driver Approach, Workingham (Addision-Wesley) 1992.

[Kersten / Murphy, 1999] M. Kersten / G. Murphy; Atlas: Acase study in building a web-based learning environment using AOP, in: workshop Aspect-Oriented Programming at ECOOP'99 (June 1999), `http://trese.cs.utwente.nl/aop-ecoop99/` (visited: 4-Feb-2002)

[Kiczales+, 1991] Gregor Kiczales / Jim des Rivieres / Daniel G. Bobrow; The Art of the Metaobject Protocol, Cambridge, Massachusetts, London (The MIT Press) 1991.

[Kiczales+, 1997] Gregor Kiczales / John Lamping / Anurag Mendhekar / Chris Medea / Cristina Lopes / John-Marc Loingtier / John Irwin; Aspect-Oriented programming, in: Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP), Jyväskylä, Finland, 9–13 June 1997, pp. 220ff., published by Springer-Verlag as Lecture Notes in Computer Science no. 124 (Mehmet Akşit and Satoshi Matsuoka, editors).

[Kiczales+, 2001] Gregor Kiczales / Erik Hilsdale / Jim Hugunin / Mik Kersten / Jeffrey Palm / William G. Griswold; Getting Started with AspectJ, in: Communications of the ACM (Association for Computing Machinery) October 2001/Vol. 44, No. 10, pp. 59–65.

[KimLoch89] Won Kim / Frederick H. Lochovsky (Eds.); Object-Oriented Concepts, Databases, and Applications, Reading, Massachusetts (Addison-Wesley) 1989.

[Koza, 1992] John R. Koza; Genetic Programming — On the Programming of Computers by Means of Natural Selection, Cambridge Massachusetts (The MIT Press), 1992, sixth printing 1998, ISBN 0-262-11170-5.

[Lieberherr+, 2001] Karl Lieberherr / Doug Orleans / Johan Ovlinger; Aspect-Oriented programming with Adaptive Methods, in: Communications of the ACM (Association for Computing Machinery) October 2001/Vol. 44, No. 10, pp. 39–41.

[Liebermann, 1981] H. Lieberman; Thinking About Lots of Things at Once Without Getting Confused - Parallelism in ACT-1, Cambridge, MIT AIMemo 626, May 1981.

[Oaks+, 2002] Scott Oaks /Bernard Traversat / Li Gong; JXTA in a Nutshell, Beijing (O'Reilly & Associates, nc.) 2002, ISBN 0-596-00236-X.

[Ossher / Tarr, 1999] H. L. Ossher / P. Tarr; Multi-dimensional separation of concerns in hyperspace, in: Technical Report RC 21452(96717), IBM T. J. Watson Research Center, 1999.

[Pace / Campo, 2001] J. André Díaz Pace / Marcelo R. Campo; Analyzing the Role of Aspects in Software Design, in: Communications of the ACM (Association for Computing Machinery) October 2001/Vol. 44, No. 10, pp. 67–73.

[Pawlak+, 2001] Renaud Pawlak / Lionel Seinturier / Laurence Duchien / Gérard Florin; JAC: A Flexible Solution for Aspect-Oriented Programming in Java, in [Yonezawa / Matsuoka, 2001], pp. 1–24. ;

[Peitgen+, 1992a] Heinz-Otto Peitgen / Hartmut Jürgens / Dietmar Saupe; Fractals for the Classroom — Part One: Introduction to Fractals and Chaos, National Council of Teachers of Mathematics 440, New York, (Springer-Verlag) 1992, ISBN 0-387-97041-X.

[Peitgen+, 1992b] Heinz-Otto Peitgen / Hartmut Jürgens / Dietmar Saupe; Fractals for the Classroom — Part Two: Complex Systems and Mandelbrot Set, National Council of Teachers of Mathematics 507, New York, (Springer-Verlag) 1992, ISBN 0-387-97722-8.

[Rumbaugh+, 1991] J. Rumbaugh / M. Blaha / W. Premerlani / F. Eddy / W. Lorenson; Objekt-oriented Modelling and Design, Englewood Cliffs (Prentice-Hall), 1991

[Shavor+03] Sherry Shavor / Jim D'Anjou / Scott Fairbrother / Dan Kehn / John Kellerman / Pat McCarty; The Java^{TM} Developer's Guide to Eclipse, Boston u. a. (Addison-Wesley), ISBN 0-321-15964-0, {Hinweis: "This Book does an excellent job of helping you learn Eclipse."}

[Stroustrup, 1986] Bjarne Stroustrup; The $C++$ Programming Language, Reading Massachusetts (Addison-Wesley) 1986 (corrected reprinting, 1987).

[Stroustrup, 1989] Bjarne Stroustrup; The Evolution of $C++$: 1985 to 1989, in: Computing Systems, 2(3) Summer 1989, pp. 191 – 250.

[Sullivan, 2001] Gregory T. Sullivan; Aspect-oriented programming using reflection and metaobject protocols — Providing programmers with the capability to modify the default behavior of a programming language —, in: Communications of the ACM (Association for Computing Machinery) October 2001/Vol. 44, No. 10, pp. 95–97.

[UML1.4] Object Management Group; Unified Modeling Language, Version 1.4, September 2001, `http://www.omg.org/technology/documents/formal/uml.htm` (visited: 4-Feb-2002)

[Ungar / Smith, 1991] David Ungar / Randall B. Smith; SELF: The Power of Simplicity, in: LISP and Symbolic Computation (Kluwer Academic Publishers), Volume 4, Number 3, July 1991, pp. 187 – 205.

[Waldhoff, 1998] Rod Waldhoff; Implementing the *Singleton* Pattern in Java; in: `http://members.tripod.com/rwald/java/articles/Singleton_in_Java.htm` (visited 8-Feb-2002)

[Wolfram, 2002] Stephen Wolfram; A New Kind of Science, 2002, ISBN 1-57955-008-8.

[Yonezawa / Matsuoka, 2001] Akinori Yonezawa / Satoshi Matsuoka (Eds.); Metalevel Architectures and Separation of Crosscutting Concerns, Third International Conference, *REFLECTION 2001* Koyoto, Japan, September 2001, Proceedings, Springer 2001, Lecture notes in computer science; Vol. 2192), ISBN 3-540-42618-3.

C.4 About this Document

The following software is used to produce this document:

Editor: GNU Emacs 21.2.1; JEdit 4.1 pre 5

Layout: TeX, Version 3.14159 (Web2c 7.3.7x), LaTeX2e <2000/06/01>; Document Class: book 2001/04/21 v1.4e Standard LaTeX document class

Hardcopy: Corel CAPTURE 10; Corel PHOTO-PAINT 10 (version 10.427)

Figure: Microsoft Visio 2000 SR1 (6.0.2072)

Index: makeindex, version 2.13 [07-Mar-1997] (using kpathsea)

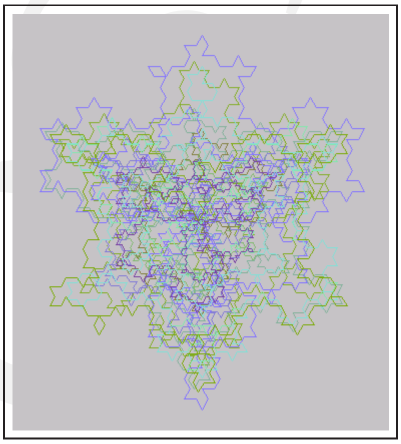
DVI→PS: LaTeX-File (Device Independent) to Postscript: dvips(k) 5.90a Copyright 2002 Radical Eye Software (www.radicaleye.com)

PS→PDF: Postscript file to PDF-File: Adobe Acrobat Distiller 5.0

Security: Adobe Acrobat 5.0 (version 5.01)

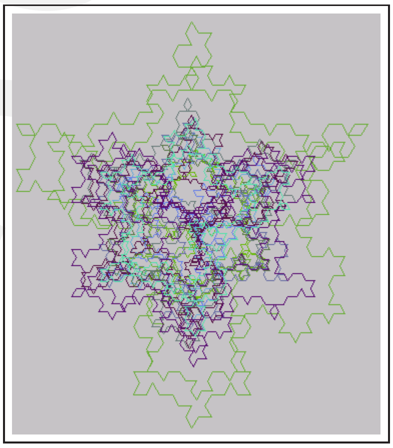
Appendix D

Index



Index

Index



⋆ ⋆ ⋆