2.13. Object-Oriented Ingredients

- Objects
- Class
- Methods
- Encapsulation
- Inheritance
- Polymorphism
- Reuse (to a given degree)

2.14. Object

Objects are the things you think about first in designing a program and they are also the units of code that are eventually derived from the process.

In between, each object is made into a generic class of object and even more generic classes are defined so that objects can share models and reuse the class definitions in their code.

Each object is an instance of a particular class or subclass with the class's own methods or procedures and data variables. An object is what actually runs in the computer.

2.15. Class

A class consists of all objects with like state which know exactly the same methods, i.e., a class knows the structure of its objects and contains the methods to manipulate the structure. An object is called an instance of a class.

Given a class, one normally creates objects. Objects are created dynamically with the prefix operator new which in turn calls a constructor method to initialize the instance variables. Uninitialized instance variables are zeroed.

Methods mostly access the instance variables of the receiver. If methods return their receiving objects, method calls (messages) can be cascaded.

The class is one of the defining ideas of object-oriented programming. Among the important ideas about classes are:

2.16. Encapsulation

Encapsulation is the inclusion within a program object of all the resources need for the object to function — basically, the methods and the data.

Other objects adhere to use the object without having to be concerned with how the object accomplishes it.

The idea is "don't tell me how you do it; just do it." An object can be thought of as a self-contained atom. The object interface consists of public methods and instance data.

2.17. Methods

A method is a programmed procedure that is defined as part of a class and included in any object of that class. A class (and thus an object) can have more than one method. A method in an object can only have access to the data known to that object, which ensures data integrity among the set of objects in an application. A method can be re-used in multiple objects.

- A class can have subclasses that can inherit all or some of the characteristics of the class. In relation to each subclass, the class becomes the superclass.
- Subclasses can also define their own methods and variables that are not part of their superclass. The structure of a class and its subclasses is called the class hierarchy.

Question: What is the difference between class and object

2.18. Inheritance

Inheritance is the concept that when a class of objects is defined, any subclass that is defined can inherit the definitions of one or more general classes.

This means for the programmer that an object in a subclass need not carry its own definition of data and methods that are generic to the class (or classes) of which it is a part.

This not only speeds up program development; it also ensures an inherent validity to the defined subclass object (what works and is consistent about the class will also work for the subclass)

2.19. Polymorphism

Polymorphism (from the Greek meaning "having multiple forms") is the characteristic of being able to assign a different meaning to a particular symbol or "operator" in different contexts.

2.20. Reuse

Do not reinvent the wheel, speed up the development, and reduces bugs. Existing classes have most likely no bugs.

3. What is Java?

Thanks to

Java is a new programming language developed at Sun under the direction of James Gosling. As far as possible it is based on concepts from C, Objective C and C++.

Java is interpreted and loads classes dynamically. There are CPU chips for Java; Sun showed a prototype Java computer early in 1996 and by now it is commercially available (if slow).

HotJava is a Web browser, that was implemented in Java using a modular architecture. It loads protocol modules and other applications (applets) dynamically from the local system or over a network and thus adapts to new tasks.

According to Hoff, Shaio and Starbuck, Java is ‘‘simple, object oriented, statically typed, compiled, architecture neutral, multi-threaded, garbage collected, robust, secure, extensible and well understood. Above all, however, Java is fun!’’

These terms are elaborated on below -- I do not quite accept all these claims, however ...

simple

Java is related to languages like C or C++, but to make the language small and easy to learn all inessentials were eliminated. This leads to more uniform programs and simpler maintenance.

Unfortunately, Java 1.1 introduced significant new, useful, but different concepts. There is a proliferation of libraries as various companies try to turn individuality into market dominance.

object-oriented

Modern programs are distributed and have graphical user interfaces. Both can be implemented more easily with object orientation. Unlike C++, Java is fully object oriented and thus furthers the right programming style for these problem areas.

statically typed

All data must be declared with types so that data and operations may be matched as far as possible during compilation. Methods are dynamically bound but overloaded signatures are decided during compilation. Like Objective C, Java permits type queries and object analysis, e.g., for the existence of methods, using the package java/lang/reflect at runtime.

compiled

Unlike TCL, Java avoids repeated source analysis. A program is compiled into byte codes, the machine language of the Java Virtual Machine (JVM). The JVM is interpreted or compiled just in time. Classes and methods are bound symbolically, i.e., classes can be recompiled individually.

architecture neutral

Byte codes are the same everywhere; only the JVM has to be implemented for a new platform. Java programs should run everywhere and can be distributed as binaries. Unlike C and C++, Java completely specifies the capacity and behavior of the primitive data types thus eliminating a serious portability problem.

Swing is implemented without any native code, relying only on the API defined in JDK 1.1.

multi-threaded

Graphical user interfaces provide the illusion of parallel execution. Threads offer an elegant implementation. Java has a thread system based on classes and the language contains simple synchronization mechanisms (monitors). Many class libraries are thread-safe.

garbage collected

Dynamic memory management as in C and C++, where the programmer attends to reusing resources, is efficient but error prone. Java only knows dynamic objects and vectors and completely frees the programmer from the problem of memory reuse. Garbage collection runs as a parallel thread and thus should not be a bottleneck in critical situations.

robust

Exceptions are an integral part of Java for error handling. The programmer is constantly forced to consider error possibilities in libraries and the compiler can check that exceptions are not hidden or overlooked.

secure

An interpreter can pretty much ensure that the interpreted program cannot crash it's platform. In connection with the Web, Java has additional security mechanisms that constrain foreign programs so that viruses are considered impossible — in spite of the fact that binary Java programs can run on arbitrary platforms.

Various groups have demonstrated, however, that security holes do exist. There is a move toward digitally signed programs and distributed trust.

extensible

Java methods can be implemented in other languages using the Java Native Interface (JNI). In principle, arbitrary libraries can be accessed as long as other security or portability aspects do not prevail.

well understood

While Java is a new language it's concepts are well known. The language definition is comprehensive and still short. Unlike C++, a programmer can certainly understand Java completely and use all of it. Java's relationship to C and it's dialects makes it easy to get into the language, although there are a number of subtle differences to be aware of.

3.1. Compilation

- `javac X.java` creates `X.class` and classes with end with `...$number.class`
- `java X.class` executes the main method of ***`.class`*** if present.

3.2. Java Execution Basics

- Classpath:
  - The JVM is using a class loader to load classes used by an application on an as-needed basis.
  - The CLASSPATH environment variable tells the class loader where to find the classes.
  - Classpath entries can be directories that contain class files for classes, or archive files (such as .zip or .jar files) that contain classes.
  - Classpath entries are colon-separated on Unix-type systems and semicolon-separated on MS Windows systems
  - In a bash/sh environment:

```
% CLASSPATH=/home/hpb/Jrms/classes.jar:$CLASSPATH
% export CLASSPATH
```

  - For *javac/java/...* on a UNIX platform:

```
% jdkTool -classpath path1:path2...
```

  - Which JDK version is used?

```
% java -version
Java HotSpot(TM) 64-Bit Server VM (build 9+178, mixed mode)
```

  - Which classes are used?

```
% java -verbose  Hello.java
java -verbose Hello.java
[0.006s][info][class,load] opened: /Library/Java/JavaVirtualMachines/jdk-9
[0.015s][info][class,load] java.lang.Object source: jrt:/java.base
[0.016s][info][class,load] java.io.Serializable source: jrt:/java.base
[0.016s][info][class,load] java.lang.Comparable source: jrt:/java.base
[0.016s][info][class,load] java.lang.CharSequence source: jrt:/java.base
[0.016s][info][class,load] java.lang.String source: jrt:/java.base
[0.016s][info][class,load] java.lang.reflect.AnnotatedElement source: jrt
 ...
```

- Coding Standard: See

3.3. Comment

See also

Java defines three kinds of comments:

/* text */
: A traditional comment: all the text from the ASCII characters /* to the ASCII characters */ is ignored (as in C and C++).

// text
: A single-line comment: all the text from the ASCII characters // to the end of the line is ignored (as in C++).

/** documentation */

A documentation comment: the text enclosed by the ASCII characters /** and */ can be processed by a separate tool to prepare automatically generated documentation of the following class, interface, constructor, or member (method or field) declaration.

3.4. Different JVMs ...

Java Standard Edition (J2SE)

Java 2 Micro Edition (J2ME), Connected Device Configuration (CDC)

Java 2 Micro Edition (J2ME), Connected Limited Device Configuration (CLDC)

Edition	Configuration	RAM	Typical Use
J2MW	Java Card	small	Smart Card
J2ME	CLDC	32k	Cellphone, pagers
J2ME	CDC	512k	PDA's
J2SE	CDC	Lots	Desktop applications

3.5. J2SE Platform at a Glance

Copied from

3.6. The first Program: Hello.java

```
 1      /**
 2       * Classical: Hello, World
 3       *
 4       *
 5       * @version   $Id: Hello.java,v 1.3 2017/06/06 23:05:46 hpb Exp hpb $
 6       *
 7       * @author    hpb
 8       *
 9       * Revisions:
10       *
11       *      Revision 1.41  2017/06/06 16:19:12  hpb
12       *      Initial revision
13       *
14       */
15      import java.util.Date;
16
17      class Hello {
18          public static void main (String args []) {
19              double aDouble = 123456.780;
20              String format = "%,15.2f\n";
21              System.out.println("Hello World");
22              System.out.printf("%-15.5s\n", "Hello World");
23              System.out.printf("%15.5s\n", "Hello");
24              System.out.printf("%,15.2f\n", aDouble);
25              System.out.printf(format, aDouble);
26          }
27      }
28
```

Source Code: Src/3/Hello.java

```
% javac Hello.java
% java Hello
Hello World
Hello
          Hello
     123,456.78
     123,456.78
```

Please take a look at the

You should use a or Do not make your source code publicly available.

3.7. The first Application

Hello demonstrates how to code a minimal application.

An application is a stand-alone Java program with a graphical user interface, usually based on the Abstract Window Toolkit

```
 1      import javax.swing.*;
 2      import java.awt.event.*;        // for WindowEvent
 3
 4      public class HelloWorld {
 5
 6          public static void main(String[] args) {
 7              JFrame frame = new JFrame("HelloWorld");
 8              final JLabel label = new JLabel("Hello World");
 9              frame.getContentPane().add(label);
10
11              frame.addWindowListener(new WindowAdapter() {
12                  public void windowClosing(WindowEvent e) {
13                      System.exit(0);
14                  }
15              });
16              frame.pack();
17              frame.setVisible(true);
18          }
19      }
```

Source Code: Src/3/HelloWorld.java

3.8. The first applet -- applets/hello

Hello is an applet and demonstrates how Java could be used in Web documents.

```
 1      /**
 2      * Classical Applet: Hello, World
 3      *
 4      * @version   $Id$
 5      *
 6      * @author    hpb
 7      *
 8      * Revisions:
 9      *       $Log$
10      */
11      import java.applet.*;           // find Applet class
12      import java.awt.*;              // find Graphics class
13      import javax.swing.*;           // find Swing class
14
15      /** A class with the first Java applet */
16      public class HelloApplet extends JApplet {
17
18              /** hook to initialize the applet */
19              public void init () {
20                      resize(150, 25);
21              }
22
23              /** redraw the Applet */
24              public void paint (Graphics g) {
25                      g.drawString("Hello, World", 50, 25);
26              }
27      }
```

Source Code: Src/3/HelloApplet.java

```
 1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
 2  <html><head>
 3  <title></title>
 4    <style type="text/css">
 5      <!--code { font-family: Courier New, Courier; font-size: 10pt; margin: 0px; }-
 6    </style>
 7    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
 8  </head><body>
 9
10
11  <!-- ================================================== -->
12  <!-- = Java Sourcecode to HTML automatically converted code = -->
13  <!-- =   Java2Html Converter 5.0 [2006-02-26] by Markus Gebhard  markus@jave.de
14  <!-- =     Further information: http://www.java2html.de     = -->
15  <div align="left" class="java">
16  <table border="0" cellpadding="3" cellspacing="0" bgcolor="#ffffff">
17     <tr>
18    <!-- start source code -->
19     <td nowrap="nowrap" valign="top" align="left">
```

```
20      <code>
21  <font color="#3f5fbf">/**</font><br />
22  <font color="#ffffff"> </font><font color="#3f5fbf">* Classical: He
23  <font color="#ffffff"> </font><font color="#3f5fbf">*</font><br />
24  <font color="#ffffff"> </font><font color="#3f5fbf">*</font><br />
25  <font color="#ffffff"> </font><font color="#3f5fbf">* </font><font color
26  <font color="#ffffff"> </font><font color="#3f5fbf">*</font><br />
27  <font color="#ffffff"> </font><font color="#3f5fbf">* </font><font color
28  <font color="#ffffff"> </font><font color="#3f5fbf">*</font><br />
29  <font color="#ffffff"> </font><font color="#3f5fbf">* Revisions:</font><
30  <font color="#ffffff"> </font><font color="#3f5fbf">*</font><br />
31  <font color="#ffffff"> </font><font color="#3f5fbf">*  Revision&nbs
32  <font color="#ffffff"> </font><font color="#3f5fbf">*  Initial 
33  <font color="#ffffff"> </font><font color="#3f5fbf">*</font><br />
34  <font color="#ffffff"> </font><font color="#3f5fbf">*/</font><br />
35  <font color="#ffffff"></font><br />
36  <font color="#7f0055"><b>class </b></font><font color="#000000">Hello </
37  <font color="#ffffff">    </font><font color="#7f0055"><b>publ
38  <font color="#ffffff">  </font><font color="#000000">System.out.println<
39  <font color="#ffffff">    </font><font color="#000000">}</font
40  <font color="#000000">}</font><br />
41  <font color="#ffffff"></font><font color="#ffffff">
42  </font></code>
43
44     </td>
45    <!-- end source code -->
46     </tr>
47  </table>
48  </div>
49  <!-- =       END of automatically generated HTML code       = -->
50  <!-- ================================================================ -->
51
52
53  </body></html>
```

Source Code: Src/3/Hello.html

```
% appletviewer Hello.html
```

gives us:

```
% netscape Hello.html
```

gives us an error message because of using swing. We will solve this problem later.

3.9. Documentation — javadoc

The various JDK packages are documented on HTML pages in a standardized format that contains very many references to other classes or replaced methods etc.

See here:

creates the documentation directly from the program sources. Special comments /** ... */ before class, variable and method declarations are transferred to the documentation. The comments may contain significant tags

See coding standard

4. Java Programming Basics

4.1. Identifier

An is an unlimited-length sequence of Java letters and Java digits, the first of which must be a Java letter. An identifier cannot have the same spelling as a keyword, Boolean literal, or the null literal.

1. Can be any length
2. First character must be a letter \ { 0, .. 9 }
3. Can not include '_' (JDK 10 modification)
4. Following characters can be letters or digits
5. Are case sensitive: TOM is NOT the same as Tom
6. A Java reserved word CANNOT be used as an identifier, ex. true
7. Must be declared to be of some type.

Identifier should/must be self explained. You might do the following

The words *i, k, lth, ...* have no real meaning for a human being.

lessThanTen	versus	*ltt*
thisIsSorted	versus	*t*
mph	versus	*speed*
maximum	versus	*m*

The basic syntax for declaring variables is:

```
typename identifier;
typename identifier = expression;
```

A variable has the following properties:

- memory location to store the value.
- type of data stored in the memory location.
- name used to refer to the memory location.

Note: You may find the definitive reference for the Java programming language

4.2. Declaration and Creation

```
 1      class DeclarationAndCreation
 2      {
 3         public static void main(String args[])
 4         {
 5            String aString;
 6            aString = new String("1234");
 7            aString.length();
 8         }
 9      }
```

Source Code: Src/4/DeclarationAndCreation.java

4.3. Declaration versus Creation

A declaration declares and connects a variable with a type.

```
int index;
String aString;
```

A creation creates an object.

The following programs have problems:

```
 1      /**
 2       * The program will not work.
 3       * What is the problem?
 4       *
 5       */
 6
 7      class WillNotCompile
 8      {
 9         public static void main(String args[])
10         {
11            String aString;
12            aString.length();
13         }
14      }
```

Source Code: Src/4/WillNotCompile.java

```
yps 4 62 javac WillNotCompile.java
WillNotCompile.java:18: Variable aString may not have been initialized.
      aString.length();
      ^
1 error
```

```
 1      /**
 2       * The program will die.
 3       * What is the problem?
 4       *
 5       * @version     $Id$
 6       *
 7       * @author      hp bischof
 8       *
 9       * Revisions:
10       *            $Log$
11       */
12
13      class WillDie
14      {
15         public static void main(String args[])
16         {
17            String aString = new String();
18            aString = null;
19            aString.length();
20         }
21      }
22
```

Source Code: Src/4/WillDie.java

```
% javac WillDie.java
% java WillDie
java/lang/NullPointerException
        at WillDie.main(WillDie.java:19)
```

4.4. String Object/Class - Literals

-
-
- Strings are constants
- All String literals are instances of the String class and exist once in a JVM.

```
 1      class StringLiteral {
 2
 3        public static void main( String args[] ) {
 4              String aString = "you";
 5              String bString = "yo" + "u";
 6
 7              if ( "you".equals(aString) )
 8                      System.out.println("1. equal");
 9              if ( bString.equals(aString) )
10                      System.out.println("2. equal");
11              if ( "yo" + "u" == aString )
12                      System.out.println("3. ==");
13              if ( bString == aString )
14                      System.out.println("4. ==");
15              if ( bString == new String("you") )
16                      System.out.println("5. ==");
17              else
18                      System.out.println("6. !=");
19
20        }
21      }
```

Source Code: Src/4/StringLiteral.java

Result

```
1. equal
2. equal
3. ==
4. ==
6. !=
```

The values of of variable of type String are inmutable. Why?

4.5. Playing with Strings

```
 1      /**
 2       * Deal with Strings objects.
 3       *
 4       * @version     $Id$
 5       *
 6       * @author      hp bischof
 7       *
 8       * Revisions:
 9       *            $Log$
10       */
```

```
11
12      class StringThing
13      {
14         public static void main(String args[])
15         {
16            String aString;
17
18            aString = new String("Last Stop Wonderland! ");
19            System.out.println( aString.length() );
20            System.out.println( aString.toUpperCase() );
21            System.out.println( aString.toUpperCase() + ".");
22            System.out.println( aString.length() + 1 );
23            System.out.println( 1 + aString.length()  );
24            System.out.println( 1 + aString + 1 );
25            System.out.println( aString + ( 1 + 1 )  );
26         }
27      }
28
```

Source Code: Src/4/StringThing.java

Result:

```
22
LAST STOP WONDERLAND!
LAST STOP WONDERLAND! .
23
23
1Last Stop Wonderland! 1
Last Stop Wonderland! 2
```

Other Example

```
 1      /**
 2        * "abc" versus new String("abc")`
 3        */
 4
 5      class StringL {
 6
 7        public static void method(String id, String literal, String aNewString)
 8              System.out.println(id + " in method");
 9              System.out.print("\tliteral= aNewString\n       ");
10              System.out.println( literal == aNewString);
11        }
12        public static void main( String args[] ) {
13              String aString = "abc";
14              System.out.print("abc == aString\n      ");
15              System.out.println("abc" == aString);
16
17              String newString = new String("abc");
18              System.out.print("abc == new String(abc)\n      ");
19              System.out.println("abc" == newString);
20
21              method("1", "abc", "abc");
22              method("2", "abc", new String("abc") );
```

```
23              method("3", "abc", "ab" + "c");
24              method("4", "abc", "" + "abc");
25        }
26      }
```

Source Code: Src/4/StringL.java

4.6. From a Previous Bridge Exam - Now availbale - and I leave the Solution Up to You

```
 1      public class X_s1 {
 2
 3         public static void method_a()        {
 4              String one  = "1";
 5              String ten  = "10";
 6              do      {
 7                      System.out.println("a_1: " + ( one == ten )  );
 8                      one = one + "0";
 9              } while ( one == ten );
10         }
11         public static void method_b()        {
12              int one  = 1;
13              int ten  = 10;
14
15              System.out.println("b_1: " + 3 * one + 0 * ten );
16         }
17         public static void method_c()        {
18              String right = new String(10 * 1 - 10 + "");
19              String middle  = "00".substring(0, 1);
20              String left  = "0";
21              int    aInt  = 0;
22
23              System.out.println("c_1: " +     ( "0" == left )   );
24              System.out.println("c_2: " +     ( "0" == middle   ) );
25              System.out.println("c_3: " +     ( left  +  aInt + aInt ) );
26              System.out.println("c_4: " +     ( left  +  aInt * aInt ) );
27              System.out.println("c_5: " +     ( ( "1" + ""  ) == "1" ) );
28              System.out.println("c_6: " +     ( ( aInt  + "1" ) == ( left + "1" ) )
29         }
30         public static void main(String argv[]) {
31              method_c();
32              method_a();
33              method_b();
34         }
35      }
```

Source Code: Src/4/X_s1.java

```
% java X_s1
c_1: true
c_2: false
c_3: 000
c_4: 00
c_5: true
c_6: false
```

```
a_1: false
b_1: 30
```

4.7. Use of the StringThing Class

```
 1      /**
 2        * Play with the String class
 3        *
 4        * @version    $Id$
 5        *
 6        * @author     Hpb
 7        *
 8        *     $Log$
 9        */
10
11      class String_1 {
12
13        public static void main( String args[] ) {
14              String aString = "David";
15              String bString = "David Bowie";
16
17              if ( "hello".equals("hello") )
18                      System.out.println("equal");
19              if ( "David" == aString )
20                      System.out.println("David == aString ");
21              System.out.println(aString.length());
22              System.out.println(aString.charAt(0));
23
24              System.out.println(aString.indexOf("vid"));
25
26              System.out.println(aString.substring(2,3));
27              System.out.println(aString.substring(
28                                      aString.indexOf("a"),
29                                      aString.indexOf("i")
30                                                ));
31
32              System.out.println(aString.concat(" Bowie").length());
33
34              String help = bString.substring(0, aString.length());
35              System.out.println("-->" + help + "<--" );
36              if ( "David" == help )
37                      System.out.println("David == help ");
38              if ( "David" == aString )
39                      System.out.println("David == bString ");
40        }
41      }
```

Source Code: Src/4/String_1.java

Result:

```
equal
ava String_1
equal
David == aString
5
D
2
v
av
11
-->David<--
David == bString
```

A question was asked:

```
 1      class Sq {
 2
 3         public static void main(String args[]) {
 4              String aString = "0";
 5              String bString = "0" + "1";
 6              System.out.println(aString + "1" == "01" );
 7              System.out.println(bString  == "01" );
 8              System.out.println("0" + "1" == "01");
 9
10        }
11      }
```

Source Code: Src/4/Sq.java

Why is it not not true in both cases?

4.8. Strings and Numbers

```
 1
 2      class StringAndInteger
 3      {
 4         public static void main(String args[])
 5         {
 6            System.out.println("Well, 3 + 4 = " + 7 );
 7            System.out.println("Well, 3 + 4 = " + 3 + 4 );
 8            System.out.println("Well, 3 + 4 = " + ( 3 + 4 ) );
 9         }
10      }
11
```

Source Code: Src/4/StringAndInteger.java

```
 1
 2      class StringAndInteger2
 3      {
 4         public static void main(String args[])
 5         {
 6            System.out.println(3 + 7);
 7            System.out.println(3 + 7 + "abc");
 8            System.out.println(3 + 7 + "abc" + 1);
 9            System.out.println(3 + 7 + "abc" + 1 + 2);
10            System.out.println("" + 3 + 7 + "abc" + 1 + 2);
11            System.out.println("" + ( 3 + 7 ) + "abc" + ( 1 + 2 ));
12         }
13      }
14
```

Source Code: Src/4/StringAndInteger2.java

4.9. More on Strings

```
 1      /**
 2        * Play with the String class
 3        *
 4        * @version    $Id$
 5        *
 6        * @author     Hpb
 7        *
 8        *     $Log$
 9        */
10
11      class StringUse {
12
13        public static void compare(String aString, String bString)    {
14              if ( aString.equals(bString) )
15                      System.out.println("\tequal");
16              else
17                      System.out.println("\t! equal");
18              if ( aString == bString)
19                      System.out.println("\t== ");
20              else
21                      System.out.println("\t! ==");
22
23        }
24        public static void main( String args[] ) {
25              String aString = "David";
26              String bString = "David";
27              compare(aString, bString);
28
29              System.out.println("Using New");
30              aString = new String("David");
31              bString = new String("David");
32              compare(aString, bString);
33
34              System.out.println("Concatenation 1");
35              aString = "Da" + "vid";
36              bString = "" + "David";
37              compare(aString, bString);
38
39              System.out.println("Concatenation 2");
40              aString = "Da" + "vid";
41              bString = "D" + "a" + "vid";
42              compare(aString, bString);
43
44        }
45      }
```

Source Code: Src/4/StringUse.java

```
% java StringUse
equal
    ==
Using New
    equal
    ! ==
Concatenation 1
    equal
    ==
Concatenation 2
    equal
    ==
```

4.10. Confusion about this

```
 1      /**
 2       * Use of this!
 3       */
 4
 5      class UseOfThis
 6      {
 7         int id;
 8         UseOfThis(int id) {
 9              this.id = id;
10         }
11         private void method_2()
12         {
13              System.out.println("method_2: " + this);
14         }
15
16         private void method_1()
17         {
18              System.out.println("method_1: " + this);
19              this.method_2();
20              method_2();
21         }
22         public String toString()     {
23              return "" + id;
24         }
25
26         public static void main(String args[])
27         {
28            UseOfThis aUseOfThis = new UseOfThis(1);
29            UseOfThis bUseOfThis = new UseOfThis(2);
30
31            System.out.println(aUseOfThis);
32            System.out.println(bUseOfThis);
33
34            aUseOfThis.method_1();
35            bUseOfThis.method_1();
36         }
37      }
38
```

Source Code: Src/4/UseOfThis.java

4.11. Primitive/Basic Types and Values

A primitive type is predefined by the Java language and named by a reserved keyword. Please see also

Remember:

A variable has the following properties:

- memory location to store the value.
- type of data stored in the memory location.
- name used to refer to the memory location.

Java knows the following types:

type	#bits	def. v.	minimum value	maximum value
byte	8 bits	0	-128	127
char	16 bits	0	0	65535
short	16 bits	0	-32768	32767
int	32 bits	0	-2147483648	2147483647
long	64 bits	0	-9223372036854775808	9223372036854775807
float	32 bits	0.0	-3.40282347E38	3.40282347E38
double	64 bits	0.0	-1.79E+308	1.79E308

Constants as in ANSI-C, constants consist of a decimal mantissa with a decimal point and an exponent with prefix e or E and optional sign. Many parts can be omitted, but one of a decimal point, an exponent, or a suffix must be present.

Constants are float only with the suffix f or F. With or without the suffix d or D they are double.

With the methods *Float.intBitsToFloat()* and *Double.longBitsToDouble()* one can change bitwise representations into floating point values.

Examples:

```
int     index;
int     milesPerHour, maximumSpeed;
float   pressure, sizeOfme;
double  starTrekSpeed;
int     picturesTakenSofar = 20;
double  probability = 0.789;
```

Conditions can only be expressed with boolean values.

boolean has the predefined constants

- true
- false.

The names are not reserved; however, they are only recognized as boolean constants.

4.12. Enum, EnumSet and EnumMap

Bitmap Solution for the problem to identify a few different cases:

```
    int one   = 1;   // 0001
    int three = 2;   // 0011
    int bit   = 2    // 0010
    if ( one == ( one & bit ) )
            //  0001 &&
            //  0010
            // ==   0000 != 0001
        ...
    if ( three == ( three & bit ) )
            //  0011 &&
            //  0001
            // ==   0000 == 0010
```

Solution for the problem to identify a few different cases

```
 1      import java.lang.Enum;
 2
 3      public class EnumExample        {
 4      enum TheColors {
 5                RED, GREEN, BLUE
 6        };
 7        public static void whatTheColors(TheColors theTheColors)      {
 8              if ( theTheColors == TheColors.RED )
 9                      System.out.println("Roses are red");
10              else    // can this happen?
11                      System.out.println("Unkonw color: " + theTheColors);
12        }
13        public static void main( String args[] ) {
14              whatTheColors(TheColors.RED);
15        }
16      }
```

Source Code: Src/4/EnumExample.java

From the documentation: "The space and time performance of this class should be good enough to allow its use as a high-quality, type safe alternative to traditional int-based "bit flags." Even bulk operations (such as containsAll and retainAll) should run very quickly if their argument is also an enum set." More later: Collection framework.

```
 1      import java.util.EnumSet;
 2      import java.util.Set;
 3
 4      public class EnumSetExample     {
 5      enum Color {
 6                RED, GREEN, BLUE
 7        };
 8        public static void main( String args[] ) {
 9              EnumSet<Color> redGreen = EnumSet.of(Color.RED, Color.GREEN);
10              EnumSet<Color> complementOf = EnumSet.complementOf(redGreen);
```

```
11              System.out.println("redGreen       = " + redGreen );
12              System.out.println("complementOf   = " + complementOf );
13        }
14      }
```

Source Code: Src/4/EnumSetExample.java

Result:

```
redGreen       = [RED, GREEN]
complementOf   = [BLUE]
```

4.13. Unicode

Skip

See also

It is necessary for a modern environment to handle, uniformly and comfortably, the textual representation of all the major written languages.

Unicode Standard defines a uniform 16-bit code based on the principle of unification:

two characters are the same if they look the same even though they are from different languages.

This principle, called Han unification, allows the large Japanese, Chinese, and Korean character sets to be packed comfortably into a 16-bit representation.

The UTF encoding of the Unicode Standard is backward compatible with ASCII.

Letting numbers be binary, a rune c is converted to a multibyte UTF sequence as follows: (See also

1. c in [00000000.0bbbbbbb] 0bbbbbbb
2. c in [00000bbb.bbbbbbbb] 110bbbbb, 10bbbbbb
3. c in [bbbbbbbb.bbbbbbbb] 1110bbbb, 10bbbbbb, 10bbbbbb

A byte in the ASCII range 0-127 represents itself in UTF.

Thus UTF is backward compatible with ASCII.

4.14. String to int

```
 1      /**
 2       * Converts a String to an int - this is one way out of many
 3       *
 4       */
 5
 6      class  StringToInt
 7      {
 8         public static void main(String args[])
 9         {
10            int i;
11            Integer aInt = new Integer("4");
12            i = aInt.intValue();
13            i = Integer.parseInt("4");
14
15         }
16      }
17
```

Source Code: Src/4/StringToInt.java

See also:

4.15. Arithmetic Expressions

- Exercises:

4.16. Arithmetic Operators

- the table below shows some of the arithmetic operators that Java provides, in the order of their precedence.
- parentheses can be used to change the order of evaluation
- an arithmetic expression returns (calculates) a value when executed.

binary Operator	Description
+	addition
-	subtraction
*	multiplication
/	integer division, if both operands are integer; real division otherwise
%	remainder
<<	bit shift left
>>	bit shift right
>>>	unsigned right shift
&	bitwise and
^	bitwise xor
|	bitwise or

Bit operation example:

```
 1      class BitPrint {
 2              private void printBytes (String what, int value) {
 3                      System.out.print(what + "\t=\t" + value + "\t=\t");
 4                      for ( int index = 31; index >= 0 ; index --)    {
 5                              if ( ( ( 1 << index ) & value ) == ( 1 << index ) )
 6                                      System.out.print("1");
 7                              else
 8                                      System.out.print("0");
 9
10                      }
11                      System.out.println();
12              }
13
14              public static void main (String args []) {
15                      BitPrint aBitPrint = new BitPrint();
16
17                      aBitPrint.printBytes("3       ", 3);
18                      aBitPrint.printBytes("-3      ", -3);
19                      aBitPrint.printBytes("5       ", 5);
20                      aBitPrint.printBytes("5 >> 1  ", (5 >> 1));
21                      aBitPrint.printBytes("5 >> 2  ", (5 >> 1));
22                      aBitPrint.printBytes("-5      ", -5);
23                      aBitPrint.printBytes("-5 >> 1 ", (-5 >> 1));
24                      aBitPrint.printBytes("-5 >>> 1", (-5 >>> 1));
25              }
26      }
```

Source Code: Src/4/BitPrint.java

Result:

```
% javac BitPrint.java && java BitPrint
3               =       3       =       00000000000000000000000000000011
-3              =       -3      =       11111111111111111111111111111101
5               =       5       =       00000000000000000000000000000101
5 >> 1          =       2       =       00000000000000000000000000000010
5 >> 2          =       2       =       00000000000000000000000000000010
-5              =       -5      =       11111111111111111111111111111011
-5 >> 1         =       -3      =       11111111111111111111111111111101
-5 >>> 1        =       2147483645      =       01111111111111111111111111111101
```

4.17. Unary/Binary Operator

unary Operator	Description
++	increment by 1
--	decrement by 1
!	not

Example:

```
int left;
int right;
int result;

     left  = 4;
     right = 3;
     result = 4 * 3;
     result = 4 + 3;
     result = 4 / 3;
     result = 4 % 3;

     left ++;
     result = left++;
     result = ++left;

     right ++;
     result = right++;
     result = ++right;
```

4.18. Mixed Mode Arithmetic and Casting

When an expression contains more than one arithmetic type all are converted to the heaviest.

byte → char → short → int → long → float → double

For example, 2 + 3.3 is interpreted as 2.0 + 3.3.

Java is strongly typed. However, the type of the results of evaluating an expression may be changed using casting. To cast, place the target type in parentheses before the operand or expression to be converted.

For example, if we really wanted the results of 2 + 3.3 to be integer, we could use

```
int index;

       index = 2 + (int) 3.3;
       index = (int) (2 + 3.3);
```

Example:

```
 1      /**
 2       * The program deals with operators.
 3       * Comment not included.
 4       *
 5       * @version   $Id$
 6       *
 7       * @author    hp bischof
 8       *
 9       * Revisions:
10       *      $Log$
11       */
12
13      class OpEx
14      {
15         public static void main(String args[])
16         {
17         char aChar           = 'b';
18         byte aByte           = 2;
19
20         int  intVar_1        = 1;
21         int  intVar_2        = 2;
22         int  intRes          = 3;
23         double       doubleVar_1     = 3.8;
24         double       doubleVar_2     = 4.8;
25         double       doubleRes       = doubleVar_1 - doubleVar_2;
26
27         System.out.println("1. " + aChar);           // man ascii decimal set
28         System.out.println("2. " + aByte);
29         System.out.println("3. " + aByte+aChar);
30         System.out.println("4. " + aByte+0);
31         System.out.println("5. " + aChar+0);
32
33         intRes = 5 / 3;        System.out.println("6. " + intRes);
34         intRes = 5 % 3;        System.out.println("7. " + intRes);
35      // intRes = 5 / doubleVar_2;    // Doesn't work, why?
36         intRes = (int)(5 / doubleVar_2); System.out.println("8. " + intRes);
37
38         doubleRes = 5   / doubleVar_2; System.out.println("9. " + doubleRes);
39         doubleRes = 5.0 / doubleVar_2; System.out.println("10. " + doubleRes);
40         }
41      }
```

Source Code: Src/4/OpEx.java

```
% javac OpEx.java
% java OpEx
1. b
2. 2
3. 2b
4. 20
5. b0
6. 1
```

7. 2
8. 1
9. 1.0416666666666667
10. 1.0416666666666667

4.19. Assignment Operators

There are 12 assignment operators; all are syntactically right-associative (they group right-to-left). See also

= *= /= %= += -=
<<= >>= >>>= &= ^= |=

<<=	bit shift left
>>=	bit shift right
>>>=	unsigned right shift
&=	and
^=	xor
|=	or

The syntax for an assignment is:

Assignment:
LeftHandSide AssignmentOperator AssignmentExpression

LeftHandSide	must be a variable
AssignmentOperator	must be one of = *= /= %= += -= <<= >>= >>>= &= ^= |=
AssignmentExpression	must be ConditionalExpression or Assignment

Note: A variable that is declared final cannot be assigned to.

4.20. Playing with the Basic Types

```
 1      /**
 2       * Ranges demonstrates integer value manipulation,
 3       * bit operations, and conversions.
 4       * Comment not included.
 5       * @version   $Id$
 6       *
 7       * @author    Axel T. Schreiner
 8       * @author    hp bischof
 9       *
10       * Revisions:
11       *      $Log$
12       */
13
14      class Ranges {
15              /** uses complement operations */
16              short x;
17              void intRange () {
18                      System.out.println("int\t" + (~0 >>> 1) +
19                              "\t" + (~ (~0 >>> 1)));
20              }
21              /** maximum and minimum long value */
22              static final long maxLong = ~0L >>> 1;
23              static final long minLong = ~ (~0L >>> 1);
24
25              void longRange () {
26                      System.out.println("long\t" + maxLong + "\t" + minLong);
27              }
28
29              /** uses casts and literals */
30              void shortRange () {
31                      System.out.println("short\t" + Short.MIN_VALUE + "\t" +
32                      Short.MAX_VALUE);
33                      System.out.println("short\t" + (short)077777 + "\t" +
34                      (short)0x8000);
35              }
36              /** shifts ones until no further changes occur */
37              void byteRange () {
38                      byte i, j = 1;
39
40                      do {
41                              i = j; j = (byte)(i << 1 | 1);
42                      } while (j > i);
43                      System.out.print("byte\t" + i);
44
45                      do {
46                              i = j; j = (byte)(i << 1);
47                      } while (j < i);
48                      System.out.println("\t" + i);
49              }
50
51
```

```
52              public static void main (String args []) {
53                      Ranges aRange = new Ranges();
54
55                      aRange.byteRange();
56                      aRange.shortRange();
57                      aRange.intRange();
58                      aRange.longRange();
59              }
60
61      }
```

Source Code: Src/4/Ranges.java

Result:

```
% java Ranges
byte    127     -128
short   -32768  32767
short   32767   -32768
int     2147483647      -2147483648
long    9223372036854775807     -9223372036854775808
```

intRange()
: produces a maximal bit pattern by complementing 0 and shifting without propagating the sign, and a minimal bit pattern by complementing once more.

\+
: concatenates strings; integer operands are converted to short strings. java/gtext has facilities for explicit formatting.

static
: defines *maxLong* and *minLong* as class variables,

final
: permits exactly one assignment. Initialization is done with a long literal and thus long arithmetic.

shortRange()
: uses int literals which must be explicitly cast to short. The following expression produces a minimal value in C but not in Java (why?):

```
(short) ~ ((unsigned short) ~0 >> 1)
```

byteRange()
: uses do-while loops. byte can be initialized with int, but for assignment an explicit cast is required.

4.21. Control Flow

4.22. Conditions

- Conditions can only be expressed with boolean values; unlike C, an implicit comparison to zero is not acceptable.
- Boolean has the predefined constants true and false. The names are not reserved; however, they are only recognized as boolean constants.

4.23. Relational Operators

- Simple boolean expressions consist of comparing things using relational operators. There two types of relational operators: equality and comparison.
-

Equality operators are defined for all objects and primitive types.

== equal
!= not equal

All comparisons produce true or false.

4.24. Logical Operators

These operators take boolean arguments and return boolean results.

- They are used to construct complex boolean expressions from simple ones consisting of boolean values and relational expressions.

&	bitwise and * - not Logical Operators
&&	conditional and (short circuits)
|	bitwise or * - not Logical Operators
||	conditional or (short circuits)
^	bitwise xor * - not Logical Operators
!	not (unary operator)/boolean complement

x && y	y will be evaluated only if x is true
x || y	y will be evaluated only if x if false

- Executed from left to right

```
 1      class LeftToRight {
 2
 3         public static int f(String whichTest, int x) {
 4              System.out.println("f(" + x + ")");
 5              return 1 / (3 - x );
 6
 7         }
 8         public static void test_1()  {
 9              boolean y;
10              for (int x = 0; x < 5; x ++ )           {
11                      y = ( x != 3 ) && ( f("1", x) > 0.5);
12              }
13         }
14         public static void test_2()  {
15              boolean y;
16              for (int x = 0; x < 5; x ++ )           {
17                      y = f("2", x) > 0.5 && ( x != 3 );
18              }
19         }
20         public static void main(String args[]) {
21              test_1();
22              test_2();
23         }
24
25      }
```

Source Code: Src/4/LeftToRight.java

Note:

```
 1
 2      class ShortCut
 3      {
 4         private boolean testIt(double n) {
 5              return n != 0.0;
 6         }
 7         private double oneDn(double n)      {
 8              return 1.0 / n;
 9         }
10
11         public static void main(String args[])
12         {  double n;
13            ShortCut aS = new ShortCut();
14
15            n = 0.5;
16            if ( aS.testIt(n) && ( aS.oneDn(n) > 1 ) )
17                    System.out.println("1:  1 / n " + 1 / n );
18
19            n = 0;
20            if ( aS.testIt(n) && ( aS.oneDn(n) > 1 ) )
21                    System.out.println("2:  1 / n " + 1 / n );
22
23            System.out.println("3: 1.0 / 0 = " + 1.0 / 0 );
24
25            if ( ( n == 0 ) || ( n == 1 ) )
26                    System.out.println("4: ( n == 0 ) || ( n == 1 ) )");
27
28             System.out.println("5. true || false && true: " + ( true || false && tr
29
30             if ( 4 == ( n = 4  ) )
31                     System.out.println("6. 4 == ( n = 4  )");
32         }
33      }
34
```

Source Code: Src/4/ShortCut.java

Result:

```
% javac ShortCut.java && java ShortCut
1:  1 / n 2.0
3: 1.0 / 0 = Infinity
4: ( n == 0 ) || ( n == 1 ) )
5. true || false && true: true
6. 4 == ( n = 4  )
```

The precedence for the arithmetic, relational, boolean operators, and assignment from highest to lowest is:

Operation	Symbol	Precedence	Association
		(from highest, 1, to lowest, 11)	
grouping	( )	1	left to right
unary	+ -	2	right to left
multiplication	*	3	left to right
division	/	3	left to right
remainder	%	3	left to right
addition	+	4	left to right
subtraction	-	4	left to right
less than	<	5	left to right
less than or equal	<=	5	left to right
greater than	>	5	left to right
greater than or equal	>=	5	left to right
equal	==	6	left to right
not equal	!=	6	left to right
bit and	&	7	left to right
xor	^	8	left to right
bit or	|	9	left to right
conditional and	&&	10	left to right
conditional or	||	11	left to right
assignment	=, +=, *= ...	12	N.A.

4.25. if Statement

See also

The if statement allows conditional execution of a statement or a conditional choice of two statements, executing one or the other but not both.

IfThenStatement:

```
if ( Expression )
        Statement
```

Example:

```
x = 3;
y = 4;
if ( x > y )
    z = x;
```

IfThenElseStatement:

```
if ( Expression )
        Statement
else
        Statement
```

Example:

```
x = 3;
y = 4;
if ( x > y )
    z = x;
else
    z = y;
```

How often and why is index incremented?

```
 1      class If
 2      {
 3         public static void main(String args[]) {
 4              int index = 2;
 5                      System.out.println("1. " + index );
 6                      if ( ++index == 2 )
 7                              System.out.println("2. " + index );
 8                      else if ( index++  == 3 )
 9                              System.out.println("3. " + index );
10                      else
11                              System.out.println("4. " + index );
12         }
13
14      }
```

Source Code: Src/4/If.java

```
% java If
1. 2
3. 4
```

4.26. Find the Maximum of two Numbers I

```
 1      /**
 2        *  Find the maximum of two numbers.
 3        *
 4        * @version    $Id$
 5        *
 6        * @author     Hpb
 7        *
 8        * Revisions:
 9        *     $Log$
10        */
11
12      class Maximum_2 {
13        public static double maximum(double _first, double _second ) {
14              double max;
15                                      // find maximum
16              if ( _first > _second )
17                      max = _first;
18              else
19                      max = _second;
20              return max;
21        }
22
23        private double minimum(double _first, double _second ) {
24              double minimum;
25                                      // find minimum
26              if ( _first < _second )
27                      minimum = _first;
28              else
29                      minimum = _second;
30              return minimum;
31        }
32
33        public static void main( String args[] ) {
34          Maximum_2 aMaximum_2 = new Maximum_2();
35          double firstN    = 42.0;
36          double secondN   = 666.0;
37
38          System.out.println("Maximum(" + firstN + ", " + secondN + ")  =  " +
39                             aMaximum_2.maximum( firstN, secondN) );
40
41          System.out.println("Minimum(" + firstN + ", " + secondN + ")  =  " +
42                             aMaximum_2.minimum( firstN, secondN) );
43        }
44      }
```

Source Code: Src/4/Maximum_2.java

As a Nassi Shneidermann diagram:

public static double maximum(double _first, double _second)

if if (_first > _second) then Yes : return _first else No
return _second }

public static double minimum(double _first, double _second)

if if (_first < _second) then Yes : return _first else No
return _second }

4.27. The Conditional Operator

?:

uses the boolean value of one expression to decide which of two other expressions should be evaluated.

The conditional operator is syntactically right-associative (it groups right-to-left), so that

```
a ? b : c ? d : e ? f : g
```

means the same as

```
a ? b : (c ? d : (e ? f : g)).
```

The conditional operator may be used to choose between second and third operands of numeric type, or second and third operands of type boolean, or second and third operands that are each of either reference type or the null type. All other cases result in a compile-time error.

See also:

```
 1
 2      class QuestionM
 3      {
 4         public static void main(String args[]) {
 5              int value       =        2 > 3 ? 2 : 3;
 6              String aString  = "      2 > 3 ? 2 : 3";
 7              System.out.println(aString + " = " + value);
 8
 9              value           =        ( 1 > 2 ? 3 :( 4 < 5 ? 6 : 7));
10              aString         = "      ( 1 > 2 ? 3 :( 4 < 5 ? 6 : 7))";
11              System.out.println(aString + " = " + value);
12
13              value           =        1 > 2 ? 3 : 4 > 5 ? 8 : 7;
14              aString         = "      1 > 2 ? 3 : 4 > 5 ? 8 : 7";
15              System.out.println(aString + " = " + value);
16         }
17      }
```

Source Code: Src/4/QuestionM.java

```
% java QuestionM
2 > 3 ? 2 : 3 = 3
( 1 > 2 ? 3 :( 4 < 5 ? 6 : 7)) = 6
1 > 2 ? 3 : 4 > 5 ? 8 : 7 = 7
```

Example:

```
 1      class Maximum_3 {
 2        private double maximum(double _first, double _second ) {
 3              return _first > _second ? _first : _second;
 4        }
 5
 6        public static double minimum(double _first, double _second ) {
 7              return _first < _second ? _first : _second;
 8        }
 9
10        public static void main( String args[] ) {
11          Maximum_3 aMax = new Maximum_3();
12          double firstN    = 42.0;
13          double secondN   = 7.0;
14
15
16          System.out.println("Maximum(" + firstN +
17              ", " + secondN + ") =  " +
18              aMax.maximum( firstN, secondN) );
19
20          System.out.println("Minimum(" + firstN +
21              ", " + secondN + ") =  "
22              + aMax.minimum( firstN, secondN) );
23        }
24      }
```

Source Code: Src/4/Maximum_3.java

4.28. while Statement

See also

The while statement executes an Expression and a Statement repeatedly until the value of the Expression is false.

WhileStatement:

```
while ( Expression )
       Statement
```

Example:

```
x = 1;
while ( x < 10 ){
    print x
    x += 2;
}
```

```
 1
 2
 3      class While
 4      {
 5         public static void main(String args[]) {
 6              int index = 1;
 7              while ( index > 0 ?  ( index == 7 ) : (index == 8 ) ) {
 8                      System.out.println("index = " + index );
 9              }
10              System.out.println("index = " + index );
11         }
12      }
```

Source Code: Src/4/While.java

```
% java While
index = 1
```

```
 1
 2
 3      class While_1
 4      {
 5         public static void main(String args[]) {
 6              int index = 1;
 7              while ( ++index++ < 4 ) {
 8                      System.out.println("index = " + index );
 9              }
10              System.out.println("index = " + index );
11         }
12      }
```

Source Code: Src/4/While_1.java

```
 1
 2
 3
 4      class While_2
 5      {
 6         public static void main(String args[]) {
 7              int index = 1;
 8              while ( ++index < 4 ) {
 9                      System.out.println("index = " + index );
10              }
11              System.out.println("index = " + index );
12         }
13      }
```

Source Code: Src/4/While_2.java

```
% java While_2
index = 2
index = 3
index = 4
```

The Expression must have type boolean, or a compile-time error occurs.

A while statement is executed by first evaluating the Expression. If evaluation of the Expression completes abruptly for some reason, the while statement completes abruptly for the same reason. Otherwise, execution continues by making a choice based on the resulting value:

If the value is true, then the contained Statement is executed. Then there is a choice:

If execution of the Statement completes normally, then the entire while statement is executed again, beginning by re-evaluating the Expression.

If the value of the Expression is false, no further action is taken and the while statement completes normally.

If the value of the Expression is false the first time it is evaluated, then the Statement is not executed.

4.29. Calculate Sqrt(2) without the MathClass

Algorithm: What is your idea?

```
 1      /**
 2        *  Calculate Sqrt(2) without the MathClass
 3        *
 4        * @version    $Id$
 5        * @author     Hpb
 6        * Revisions:
 7        *     $Log$
 8        */
 9
10      class Sqrt {
11
12        private double calculateSqrt_2() {
13              double n1 = 1.0;
14              double n2 = 2.0;
15              while ( (n2 * n2 - n1 * n1)  > 0.0001)  {
16                      double x = ( n2 + n1 ) * 0.5;
17                      if ( x * x > 2.0 )
18                              n2 = x;
19                      else
20                              n1 = x;
21              }
22              return n1;
23        }
24
25        public static void main( String args[] ) {
26
27          System.out.println("sqrt(2)  = " +
28                             new Sqrt().calculateSqrt_2() + " +- 0.0001 " );
29        }
30      }
```

Source Code: Src/4/Sqrt.java

4.30. Continue

- continue statement may occur only in a while, do, or for statement;
- continue statement with no label attempts to transfer control to the innermost enclosing while, do, or for statement;
- this statement, which is called the continue target, then immediately ends the current iteration and begins a new one.
- If no while, do, or for statement encloses the continue statement, a compile-time error occurs.

Example 1:

```
 1      class Continue_1 {
 2
 3        public static void main( String args[] ) {
 4
 5                int n = 0;
 6
 7      label1:
 8                 while ( n < 6 )  {
 9                      System.out.println("1. n == " + n);
10
11                      while ( n < 4 )  {
12                              n++;
13                              System.out.println("    2.  n == " + n);
14                              if ( n > 2 )
15                                      continue label1;
16                              System.out.println("    3.  n == " + n + "--------");
17                      }
18                      n++;
19              }
20        }
21      }
```

Source Code: Src/4/Continue_1.java

```
% java Continue_1
1. n == 0
    2.  n == 1
    3.  n == 1--------
    2.  n == 2
    3.  n == 2--------
    2.  n == 3
1. n == 3
    2.  n == 4
1. n == 4
1. n == 5
```

4.31. Break

- A break statement transfers control out of an enclosing iteration or switch statement.
- Labeled breaks are used to jump out of nested loops

- break statement with no label attempts to transfer control to the innermost enclosing switch, while, do, or for statement;
- If no switch, while, do, or for statement encloses the break statement, a compile-time error occurs.

Example 1:

```
 1      class Break_1 {
 2
 3        public static void main( String args[] ) {
 4
 5              int n = 0;
 6
 7         here:        {
 8              while ( true )  {
 9                      System.out.println("1. n == " + n);
10
11                      while ( n < 100 )  {    // while ( true ) --> which problem
12                              n++;
13                              System.out.println("    2. n == " + n);
14                              if ( n > 2 )
15                                      break here;
16                      }
17                      System.out.println("3.  n == " + n);
18                 }
19              }
20        }
21      }
```

Source Code: Src/4/Break_1.java

Example 2:

```
 1      class Break_2 {
 2
 3        public static void main( String args[] ) {
 4
 5              int n = 0;
 6
 7              System.out.println("start");
 8              while ( n < 100 )  {
 9                      if ( n > 4 )
10                              System.exit(1);
11
12                      while ( n < 100 )  {    // while ( true ) --> which problem
13                              n++;
14                              System.out.println("    inner while here n == " + n);
15                              if ( n > 2 )
16                                      break;
17                      }
18                      System.out.println("outer while here n == " + n);
19                 }
20              System.out.println("after here }");
```

```
21        }
22      }
```

Source Code: Src/4/Break_2.java

4.32. Return

A return statement returns control to the invoker of a method or constructor.

```
 1      /* How can we set the exit code?        */
 2
 3      class Return {
 4
 5        public static void main( String args[] ) {
 6              int x = 0;
 7              return x;
 8        }
 9      }
```

Source Code: Src/4/Return.java

```
 1      /* How can we set the exit code?        */
 2
 3      class Return_1 {
 4
 5        public static int method() {
 6              System.exit(2);
 7              return 0;
 8        }
 9        public static void main( String args[] ) {
10              method() ;
11              System.out.println("xxx");
12        }
13      }
```

Source Code: Src/4/Return_1.java

4.33. Return vs. Continue vs. Break

- What are the differences?

4.34. Abrupt Completion

Abrupt completion of the contained Statement is handled in the following manner:

An abrupt completion always has an associated reason, which is one of the following: (from

- A break with no label
- A break with a given label
- A continue with no label
- A continue with a given label
- A return with no value
- A return with a given value
- A throw with a given value, including exceptions thrown by the Java virtual machine

```
 1      class Break {
 2
 3        public static void main( String args[] ) {
 4
 5              int n = 0;
 6
 7         here: {
 8              while ( true )  {
 9                      System.out.println("a: outer while here n == " + n);
10
11                      if ( n > 4 )
12                              System.exit(1);
13
14                      while ( true )  {
15                              System.out.println("    inner while here n == " + n);
16                              if ( ++n == 0 )
17                                      System.out.println("n == 0");
18                              else if ( n++ == 1 )    {
19                                      System.out.println("    n == 1");
20                                      System.out.println("    break");
21                                      break;
22                              } else if ( n++ == 2 )
23                                      System.out.println("    n == 2");
24                              else
25                                      System.out.println("    n == 3");
26
27                              System.out.println("    executing break here");
28                              System.out.println("    n is " + n );
29
30                              break here;
31
32                      }
33                      System.out.println("b: outer while here n == " + n);
34                 }
35                 // unreachable statement ...System.out.println("here }");
36              }
37        }
38      }
```

Source Code: Src/4/Break.java

4.35. do Statement

See also

The do statement executes a Statement and an Expression repeatedly until the value of the Expression is false.

```
do Statement while ( Expression ) ;
```

4.36. for Statement

See also

The *for statement* executes some initialization code, then executes an Expression, a Statement, and some update code repeatedly until the value of the Expression is false.

ForStatement:

```
f.or ( ForInit; Expression; ForUpdate)
            Statement
```

Example:

```
 1      class For_1 {
 2
 3        public static void main( String args[] ) {
 4              int index = 0;
 5              for ( index = 0 ; index < 1; index ++  )        {
 6                      System.out.println("1. index = " + index );
 7              }
 8              System.out.println("2. index = " + index );
 9        }
10      }
```

Source Code: Src/4/For_1.java

```
 1      class For_2 {
 2
 3        public static void main( String args[] ) {
 4              int index = 0;
 5              for ( index = 0 ; index < 1; index ++  )        {
 6                      index = -1;
 7                      System.out.println("1. index = " + index );
 8                      break;
 9              }
10              System.out.println("2. index = " + index );
11        }
12      }
```

Source Code: Src/4/For_2.java

```
 1      class For_3 {
 2
 3        public static void main( String args[] ) {
 4              for ( int index = 0 ; index < 1; index ++  )    {
 5                      System.out.println("1. index = " + index );
 6                      break;
 7              }
 8              System.out.println("2. index = " + index );
 9        }
10      }
```

Source Code: Src/4/For_3.java

4.37. Find all Prime Numbers in [2 ... 100]

isPrime(n): do for index = 2 to n - 1 : if (index % n == 0) : return false } : return true *findAllPrimeN()*: do for index = 1 to 100 : if (isPrime(n)) : print index }

```
 1      /**
 2        *  Find all prime numbers in the range
 3        *  between 1 and 10
 4        *
 5        * @version    $Id$
 6        *
 7        * @author     Hpb
 8        *
 9        * Revisions:
10        *     $Log$
11        */
12
13      class Prime_1 {
14
15        private boolean isPrime(int n) {
16
17              for ( int index = 2; index < n; index ++ ) {
18                      if ( n % index  == 0 )
19                              return false;
20              }
21
22              return true;
23        }
24        public static void main( String args[] ) {
25          Prime_1 aPrime = new Prime_1();
26
27          for ( int index = 2; index <= 10; index ++ )
28              if ( aPrime.isPrime(index) )
29                      System.out.println(index + " " );
30        }
31      }
```

Source Code: Src/4/Prime_1.java

4.38. Switch Statement

See also

The switch statement transfers control to one of several statements depending on the value of an expression. The type of the switch expression can be

- char
- byte
- short
- int
- strings
- enum type See

Switch Statement:

```
switch ( Expression )  {
    case ConstantExpression_1 : action_1;
    case ConstantExpression_2 : action_2;
    ...
    default: action_d
```

Example:

```
 1      class Switch {
 2
 3      static void method(int k) {
 4              switch (k) {
 5              case 1:  System.out.println("with break: 1 ");
 6                       break;
 7              case 2:  System.out.println("with break: 2 ");
 8                       break;
 9              default: System.out.println("with break: default");
10              }
11      }
12
13      static void methodWithoutDefault(int k) {
14              switch (k) {
15              case 1:  System.out.println("   without break: 1 ");
16                       break;
17              case 2:  System.out.println("   without break: 2 ");
18                       break;
19              }
20      }
21
22      public static void main(String[] args) {
23              new Switch().method(3);
24              new Switch().methodWithOutDefault(2);
25              new Switch().methodWithOutDefault(3);
26
27        }
28      }
```

Source Code: Src/4/Switch.java

4.39. Partial Lowercase → Uppercase

```
 1      /**
 2        *  Test of the switch statement.
 3        *
 4        * @version    $Id$
 5        *
 6        * @author     hpb
 7        *
 8        * Revisions:
 9        *     $Log$
10        */
11
12
13      class Switch_1 {
14        private String itWasA(char c) {
15              switch( c )     {
16              case 'a':       return("A");    // break?
17              case 'b':       return("B");    // break?
```

```
18              case 'c':       return("C");    // break?
19              case 100:       return("D");    // break?
20              case 101:       return("E");    // break?
21              default:        return("no clue, but not an [a-e]");
22                                              // What happens if
23                                              // we delete this line?
24              }
25        }
26
27
28        public static void main( String args[] ) {
29            char theChar;
30
31            theChar = 'd';
32            System.out.println("You typed in an '" +
33                      new Switch_1().itWasA(theChar) + "'");
34
35            System.exit(0);   // beware of ...
36        }
37      }
```

Source Code: Src/4/Switch_1.java

Characters can be safely converted to a integers (Unicode), but should be avoided at all times.

4.40. Questions

- Which variable names are valid:

```
1       class X_1
2       {
3          public static void main(String args[])
4          {
5             int aInt;
6             int countUpTo5;
7             int 5IsA_niceNumber;
8             int ooo\";
9             int notToMany:areAllowd;
10         }
11      }
12
```

Source Code: Src/4/X_1.java

- What is the output of the following program:

```
1       class X_2
2       {
3          public static void main(String args[])
4          {
5             System.out.println("I like to play " + 6  + 2 );
6             System.out.println("I like to play " + 6  * 2 );
7             System.out.println("I like to play " + ( 6  + 2 ) );
```

```
 8         }
 9      }
10
```

Source Code: Src/4/X_2.java

- Will the following program compile?

```
 1      class X_3
 2      {
 3         public static void main(String args[])
 4         {
 5            i += 64;
 6            System.out.println("1. " + ( i << 2 ) );
 7         }
 8      }
 9
```

Source Code: Src/4/X_3.java

- What is the output of the following program:

```
 1      class X_4
 2      {
 3         public static void main(String args[])
 4         {
 5            int i = 0;
 6            i += 63;
 7            System.out.println("1. " + ( i++ >> 2 ) );
 8            System.out.println("2. " + ( 1 > 2 ? 3 : 6 ));
 9
10
11              /*
12               *      a ? b : c ? d : e ? f : g
13               *      means the same as
14               *      a ? b : (c ? d : (e ? f : g)).
15               */
16            System.out.println("3. " +
17               ( 1 > 2 ? 3 : 4 < 5 ? 6 : 9 < 10 ? 7 : 8));
18            System.out.println("4. " +
19               (1 > 2 ? 3 : (4 < 5 ? 6 : (9 < 10 ? 7 : 8))));
20         }
21      }
22
```

Source Code: Src/4/X_4.java

- What is the output of the following program:

```
 1      class X_5 {
 2
 3        public static void main( String args[] ) {
 4
 5              int n = 0;
 6
 7              while ( true )  {
 8                      System.out.println("xx");
 9                      if ( n++ == 0 ) {
10                              System.out.println("n == 0");
11                      } else if ( n++ == 1 )  {
12                              System.out.println("n == 1");
13                      } else if ( n-- == 2 )
14                              System.out.println("n == 2");
15
16              }
17        }
18      }
```

Source Code: Src/4/X_5.java

5. Scanner: Overview

- Introduced to satisfy faculty, students, and anyone who wants to write a quick-and-dirty Java program that uses console I/O.
- Works like StreamTokenizer
- Implements Iterator<String>
- See here:
- Perl-like pattern matching available
- See here:

5.1. Scanner: Constructors

- Scanner(File source)
- Scanner(InputStream source)
- Scanner(String source)
- System.in is an InputStream
- There are also constructors to work with:
  alternate character sets; input objects from the java.nio library.

5.2. Scanner: Reading Methods

- String next()
- String next(Pattern pattern)
- boolean nextBoolean()
- double nextDouble()
- int nextInt()
- int nextInt(int radix)
- String nextLine()

5.3. Scanner: Testing Methods

- boolean hasNext()
- boolean hasNext(Pattern ptrn)
- boolean hasNextBoolean()
- boolean hasNextDouble()
- boolean hasNextInt()
- boolean hasNextInt(int radix)
- boolean hasNextLine()

5.4. Scanner: Example 1

```
1      import java.util.Scanner;
2
3      public class Scanner1 {
4          public static void main( String[] args ) {
5              Scanner sc  = new Scanner( System.in);
6              System.out.printf("> ");
7              while ( sc.hasNext() )  {
```

```
 8                      String line = sc.nextLine();
 9                      System.out.printf("-%s-%n", line );
10                      System.out.printf("> ");
11              }
12              sc.close();
13          }
14      }
15
16
```

Source Code: Src/6_jdk15/Scanner1.java

```
% java Scanner1
> 1 2 3 4 hello
-1 2 3 4 hello-
> ups
-ups-
> # ^D here ....
```

5.5. Scanner: Example 2

```
 1
 2      import java.util.Scanner;
 3
 4      public class Scanner2 {
 5          public static void main( String[] args ) {
 6              Scanner sc  = new Scanner( System.in);
 7              System.out.printf("> ");
 8              while ( sc.hasNext() )  {
 9                      Integer aInteger = sc.nextInt();
10                      System.out.printf("-%d-%n", aInteger );
11                      System.out.printf("> ");
12              }
13              sc.close();
14          }
15      }
16
17
```

Source Code: Src/6_jdk15/Scanner2.java

```
% java Scanner2
> 1 2 3 4 1.0
-1-
> -2-
> -3-
> -4-
> Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:819)
        at java.util.Scanner.next(Scanner.java:1431)
        at java.util.Scanner.nextInt(Scanner.java:2040)
        at java.util.Scanner.nextInt(Scanner.java:2000)
        at Scanner2.main(Scanner2.java:9)
```

Reading from a file

5.6. Scanner: Example 3

```
 1
 2      import java.util.Scanner;
 3
 4      public class Scanner3 {
 5          public static void main( String[] args ) {
 6              Scanner sc  = new Scanner("1blobblob2blob3").useDelimiter("blob");
 7              System.out.printf("> ");
 8              while ( sc.hasNext() )  {
 9                      String line = sc.next();
10                      System.out.printf("-%s-%n", line );
11                      System.out.printf("> ");
12              }
13              sc.close();
14          }
15      }
16
17
```

Source Code: Src/6_jdk15/Scanner3.java

```
% java Scanner3
> -1-
> --
> -2-
> -3-
```

5.7. Scanner: Example 4

```
 1      /*
 2       * example is from: http://www.cs.rit.edu/~hpb/Jdk5/api/java/util/Scanner.html
 3       */
 4      import java.util.Scanner;
 5      import java.util.regex.MatchResult;
 6
 7      public class Scanner4 {
 8          public static void printIt(String input)    {
 9              Scanner sc = new Scanner(input);
10
11              System.out.println("sc.findInLine: " +
12                      sc.findInLine("(\\d+) fish (\\d+) fish (\\w+) fish (\\w+)"));
13              MatchResult result = sc.match();
14
15              for (int i=1; i<=result.groupCount(); i++) {
16                      System.out.println(i + ": " + result.group(i));
17              }
18          }
19
20          public static void main( String[] args ) {
```

```
21              String input = "1 fish 2 fish red fish blue fish";
22              printIt(input);
23
24          }
25      }
26
27
```

Source Code: Src/6_jdk15/Scanner4.java

```
% java Scanner4
1: 1
2: 2
3: red
4: blue
```

5.8. Scanner: Example 7

```
 1
 2      import java.util.Scanner;       // what is this good for?
 3      import java.io.File;            // what is this good for?
 4
 5      public class Scanner7 {
 6          public static void asIs() {
 7              Scanner sc  = new Scanner(System.in);
 8              System.out.print(": ");
 9              while ( sc.hasNext() )
10                      System.out.print("-" + sc.next() + "+");
11              sc.close();
12              System.out.println();
13          }
14          public static void whiteSpace(String description, String theDelimiter) {
15              Scanner sc  = null;
16              try {
17                      sc  = new Scanner(new File( "words.txt") );
18              } catch ( Exception e ) {}
19              sc.reset();
20              sc.useDelimiter(theDelimiter);  // A whitespace character: [ \t\n\x0B\
21              System.out.println(description);
22              System.out.println("\tdelemiter: " + theDelimiter);
23              while ( sc.hasNext() )
24                      System.out.println("\t-" + sc.next() + "+");
25              sc.close();
26              System.out.println();
27          }
28          public static void main( String[] args ) {
29              whiteSpace("java white space", "\\p{javaWhitespace}+");
30              whiteSpace("white space* and comma and white spice*", "\\s*,\\s*");
31              whiteSpace("white space+ and comma and white spice*", "\\s+,\\s*");
32              whiteSpace("comma or semicolom", "\\s*(,|;)\\s*");
33          }
34      }
35
```

36

Source Code: Src/6_jdk15/Scanner7.java

```
% cat words.txt
1, 2,3,    ,5;7
% cat words.txt | od -t a
0000000    1   ,  sp   2   ,   3   ,  sp  sp  sp  sp   ,   5   ;   7  nl
0000020
% java Scanner7 < words.txt
java white space
    delemiter: {javaWhitespace}+
    -1,+
    -2,3,+
    -,5;7+

white space* and comma and white spice*
    delemiter: ,
    -1+
    -2+
    -3+
    -+
    -5;7
+

white space+ and comma and white spice*
    delemiter:   -1, 2,3,+
    -5;7
+

comma or semicolom
    delemiter: (,|;)
    -1+
    -2+
    -3+
    -+
    -5+
    -7
+
```

6. Class Relationships

See also:

- Class declarations define new reference types and describe how they are implemented.
- Constructors are similar to methods, but cannot be invoked directly by a method call; they are used to initialize new class instances. Like methods, they may be overloaded.
- Static initializers are blocks of executable code hat may be used to help initialize a class when it is first loaded.
- The body of a class declares members, static initializers, and constructors.
- Static import feature allows to access classes of a package without package qualification (*Math.PI or PI*.
- The scope of the name of a member is the entire declaration of the class to which the member belongs. Field, method, and constructor declarations may include the access modifiers public, protected, or private. The members of a class include both declared and inherited members.
- Newly declared fields can hide fields declared in a superclass or super interface. Newly declared methods can hide, implement, or override methods declared in a superclass.
- visibility modifier: public/protected/private

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

- Return type: void/primitive type/reference to a object
- Class methods/Class variables are declared with static.
- Static declaration inside a method change the lifetime of a variable.

6.1. Example

```
 1      import static java.lang.Math.*;
 2      public class  X {
 3               static X aX = null;    // use before define
 4               static int cI = 0;
 5               int oI = 1;
 6
 7               public X()    {
 8               }
 9               static public void cm(int i){
10                      cI = i;
11                      System.out.println("cm: " + cI );
12               }
13               public void om(int i){
14                      cI = i;
15                      oI = i;
16                      System.out.println("om: " + this);
17                      System.out.println("    cI " + cI);
18                      System.out.println("    oI " + oI);
19               }
20
21               public static void main(String args[] )        {
22                    System.out.println(PI);
23                      X aX = new X();
24                      X aaX = new X();
25
26                      aX.cm(1);
27                      aaX.cm(2);
28
29                      aX.om(3);
30                      aaX.om(4);
31               }
32      }
```

Source Code: Src/5/X.java

```
% java X
cm: 1
cm: 2
om: X@1ad086a
    cI 3
    oI 3
om: X@10385c1
    cI 4
    oI 4
```

6.2. Class Details

6.3. Static in Classes/Methods Lifetime

- Class Variables: If a variable is declared static, there exists exactly one incarnation of the field,

- Static Methods: A method that is declared static is called a class method. A class method is always invoked without reference to a particular object.
- Non Static Methods: A method that is not declared static is called an instance method, and sometimes called a non-static method. An instance method is always invoked with respect to an object, which becomes the current object to which the keywords this and super refer during execution of the method body.
- Variables can be declared:
  - — static: class variable
  - — final: can be assigned once or zero time
  - — transient: not stored or saved via the standard serialization process
  - — volatile: A variable may be modified by multiple threads. This gives a hint to the compiler to fetch the value each time, rather store a locale copy. This also prohibits same optimization procedures.
- See also:

```
 1
 2     public class  Overview  {
 3       int                   instanceVariable;
 4       static    int         classVariable;
 5       final     int         finalVariable; // static?
 6       volatile  int         volatileVariable;
 7       transient int         transientVariable;
 8
 9
10       public Overview()     {
11             finalVariable = 42;
12       }
13       public Overview(int aLocalVariable)   {
14             finalVariable = 43;
15       }
16       void instanceMethod() {
17             finalVariable = 43;
18             instanceVariable = 22;
19             classVariable = 33;
20       }
21       static void classMethod()     {
22             classVariable = 3;
23       }
24
25       public static void main(String args[] )       {
26             Overview aOverview = new Overview();
27             Overview bOverview = new Overview();
28             Overview cOverview = new Overview(1);
29             cOverview = bOverview;
30             aOverview.instanceMethod();
31             instanceMethod();
32             bOverview.classMethod();
33             // values of aOverview.? bOverview.?
34             // aOverview.finalVariable??
35
36       }
```

```
37      }
```

Source Code: Src/5/Overview.java

- how many instances of the variables to exist?
- How many objects do exist?

6.4. Parameter Passing

- The formal parameters of a method, if any, are specified by a list of comma-separated parameter specifiers.
- Each parameter specifier consists of a type and an identifier (optionally followed by brackets) that specifies the name of the parameter.
- If a method has no parameters, only an empty pair of parentheses appears in the method's declaration.
- If two formal parameters are declared to have the same name (that is, their declarations mention the same identifier), then a compile-time error occurs.
- When the method is invoked, the values of the actual argument expressions initialize newly created parameter variables, each of the declared type, before execution of the body of the method.
- The scope of formal parameter names is the entire body of the method. These parameter names may not be redeclared as local variables or exception parameters within the method; that is, hiding the name of a parameter is not permitted.
- call by value

6.5. Example I

```
 1      public class  ExampleClass      {
 2        int aLocalVariable = 3;
 3
 4        public ExampleClass() {
 5              aLocalVariable = 2;
 6        }
 7        public ExampleClass(int aLocalVariable)       {
 8              this.aLocalVariable = aLocalVariable;
 9              aLocalVariable = 6;
10        }
11
12        public static void main(String args[] )       {
13              ExampleClass aExampleClass = new ExampleClass();
14              aExampleClass = new ExampleClass(3);
15              System.out.println("the value is: " + aExampleClass.aLocalVariable);
16              System.out.println(aLocalVariable);
17        }
18      }
```

Source Code: Src/5/ExampleClass.java

Questions:

- How does the the JVM find the main method, when you execute *java ExampleClass*?
- Describe the execution order of the constructors.
- Which variables will be modified when?

6.6. Example II

```
 1      import java.util.Vector;
 2
 3      public class  Args {
 4
 5              String aString;
 6              int[]  anArray = { 4, 2 };
 7
 8              public void testString(String arg )      {
 9                      arg = "b";
10              }
11              public void testArray(int[] anArray )      {
12                      anArray[1] = 3;
13              }
14              public void testString()        {
15                      aString = "a";
16                      System.out.println("1.  " + aString);
17                      testString(aString);
18                      System.out.println("2.  " + aString);
19              }
20              public void testArray() {
21                      System.out.println("3.  " + anArray[0] + ", " + anArray[1]);
22                      testArray(anArray);
```

```
23                      System.out.println("4.  " + anArray[0] + ", " + anArray[1]);
24              }
25              public static void main(String args[] )       {
26                      new Args().testString();
27                      new Args().testArray();
28              }
29      }
```

Source Code: Src/5/Args.java

```
% java Args
1.  a
2.  a
3.  4, 2
4.  4, 3
```

Questions:

- How does the the JVM find the the variable *aStaticLocalVariable*?
- Which variables will be modified when?

6.7. Example III

```
 1      public class  ExampleClassIII   {
 2
 3        String aString = null;
 4
 5        public void method(String a)  {
 6              a = new String("set in method");
 7              System.out.println("2. method:a:" + a );
 8        }
 9        public void test()    {
10              String aString = new String("set in test");
11
12              System.out.println("1. test:aString:" + aString );
13              method(aString);
14              System.out.println("3. test:aString:" + aString );
15        }
16        public static void main(String args[] )       {
17              new ExampleClassIII().test();
18        }
19      }
```

Source Code: Src/5/ExampleClassIII.java

6.8. Example IV

```
 1
 2      public class  XX        {
 3        int instanceV = 1;
 4
 5        static XX b;
 6        static XX a;
 7
 8        public XX()   {
 9        }
10        public void method(int i){
11              instanceV = i;
12        }
13
14        public String toString() {
15              return "instanceV = " + instanceV;
16        }
17
18        public void m2(int i){
19              a.method(i);
20              method(2 * i);
21              System.out.println("----------------------------------------");
22              System.out.println("print itself : " + this);
23              System.out.println("print a: " + a);
24              System.out.println("========================================");
25        }
26
```

```
27        public static void main(String args[] )       {
28              b = new XX();
29              a = new XX();
30
31              b.m2(3);
32              a.m2(24);
33        }
34      }
```

Source Code: Src/5/XX.java

- does it compile?
- Output?

```
----------------------------------------
print itself : instanceV = 12
print aaaaaa: instanceV = -9
========================================
----------------------------------------
print itself : instanceV = 12
print aaaaaa: instanceV = 12
========================================
```

```
 1
 2      public class  ClassXX   {
 3        static int instanceV = 1;
 4
 5        static ClassXX bbbbb;
 6        static ClassXX aaaaaa;
 7
 8        public ClassXX()      {
 9        }
10        public void method(int i){
11              instanceV = i;
12        }
13
14        public String toString() {
15              return "instanceV = " + instanceV;
16        }
17
18        public void m2(int i){
19              aaaaaa.method(-9);
20              method(12);
21              System.out.println("----------------------------------------");
22              System.out.println("print itself : " + this);
23              System.out.println("print aaaaaa: " + aaaaaa);
24              System.out.println("========================================");
25        }
26
27        public static void main(String args[] )       {
28              bbbbb = new ClassXX();
29              aaaaaa = new ClassXX();
30
31              bbbbb.m2(3);
```

```
32              aaaaaa.m2(24);
33        }
34      }
```

Source Code: Src/5/ClassXX.java

6.9. Example VI

```
 1
 2      public class  XXX       {
 3        int oI = 1;
 4
 5        XXX aXXX = new XXX();;
 6
 7        public XXX() {
 8        }
 9        public static void main(String args[] )       {
10              XXX aaXXX = new XXX();
11        }
12      }
```

Source Code: Src/5/XXX.java

- does it compile? ja. Draw the memory pic during execution
-
- Does it execute? Ja, aber es wird sterben, weil im C. ein wort fehlt Output?

6.10. A Point Class

- An example without any comment:

  I. Use of a *Point* Class:

```
 1      /**
 2       * This class implements a point test program.
 3       *
 4       * @version   $Id$
 5       *
 6       * @author    hp bischof
 7       *
 8       * Revisions:
 9       *      $Log$
10       */
11
12
13
14      public class TestPoint {
15        private static Point aPoint;
16
17
18      /**
19        * The main program.
```

```
20        *
21        * @param    args    command line arguments (ignored)
22        */
23        public static void main(String args[])
24        {
25              System.out.println("Point.soManyPoints = " + Point.soManyPoints() );
26              aPoint = new Point(2, 3);
27              System.out.println("x = " + aPoint.getX() );
28              System.out.println("y = " + aPoint.getY() );
29
30              aPoint = new Point();
31              aPoint.initPoint(4, 5);
32              System.out.println("x = " + aPoint.getX() );
33              System.out.println("y = " + aPoint.getY() );
34
35              aPoint.move(6, 7);
36              System.out.println("x = " + aPoint.getX() );
37              System.out.println("y = " + aPoint.getY() );
38
39              System.out.println("nPoints = " + aPoint.getNPoints() );
40              System.out.println("aPoint.soManyPoints = " + aPoint.soManyPoints() );
41        }
42      }
43
44
45
46
```

Source Code: Src/5/TestPoint.java

II. The *Point* Class:

```
 1      /**
 2       * This class implements a point in a two dimensional
 3       * area.
 4       * All method print when they are called.
 5       *
 6       * @version   $Id$
 7       *
 8       * @author    hp bischof
 9       *
10       * Revisions:
11       *      $Log$
12       */
13
14      public class Point {
15                                      // class variable
16        static int nPoints;           // so many points were created.
17
18        private int x;                // x coordinate of the point
19        private int y;                // y cooridnate of the point
20
21      /**
22       * Default Constructor.
23       * Increases the counter nPoints by 1.
24       *
25       * @return            Point a Point object
26       */
27        public Point(){
28              super();
29              System.out.println("    in Point() constructor");
30              nPoints ++;
31        }
32
33      /**
34       * Constructor.
35       * initialize x and y values of a point
36       *
37       * @param       x       x coordinate
38       * @param       y       y coordinate
39       *
40       * @return            Point a Point object
41       */
42        public Point(int x, int y){
43              super();
44              int i ++;
45              this.x = x;
46              this.y = y;
47              System.out.println("    in Point(int, int) constructor");
48        }
49
50      /**
51       * So many points have been created.
```

```
52       *
53       * @return int So many points have been created
54       */
55        public static int soManyPoints(){
56              return nPoints;
57        }
58
59      /**
60       * initialzes x and y of a point.
61       *
62       * @param       x       int x coordinate
63       * @param       y       int y coordinate
64       *
65       * @return            Point a Point object
66       */
67        public Point initPoint(int x, int y){
68              System.out.println("    in initPoint(int, int)");
69
70              this.x = x;
71              this.y = y;
72
73              return this;
74        }
75
76      /**
77       * move a point
78       *
79       * @param       x       int delta x value
80       * @param       y       int delta y value
81       *
82       * @return            Point a Point object
83       */
84        public Point move(int x, int y){
85              System.out.println("    in move(int, int)");
86
87              this.x += x;
88              this.y += y;
89
90              return this;
91        }
92
93      /**
94       * Returns the x coordinate of a point
95       *
96       * @return            int x value
97       */
98        public int getX(){
99              System.out.println("    in getX()");
00              return this.x;
01        }
02
03      /**
04       * Returns the y coordinate of a point
05       *
```

```
06       * @return            int x value
07       */
08        public int getY(){
09              System.out.println("    in getY()");
10              return this.y;
11        }
12
13      /**
14       * Returns how many points are created so far.
15       *
16       * @return            int nPoints
17       */
18        public int getNPoints(){
19              System.out.println("    in getNPoints()");
20              return this.nPoints;
21        }
22      }
```

Source Code: Src/5/Point.java

II. Execution of the test program:

```
Point.soManyPoints = 0
        in Point() constructor
        in Point(int, int) constructor
        in getX()
x = 2
        in getY()
y = 3
        in Point() constructor
        in initPoint(int, int)
        in getX()
x = 4
        in getY()
y = 5
        in move(int, int)
        in getX()
x = 10
        in getY()
y = 12
        in getNPoints()
nPoints = 2
aPoint.soManyPoints = 2
```

You may find the javadoc pages

6.11. Additional Examples

See

```
 1      public class  Scope_1   {
 2
 3        String aString = null;
 4
 5        public void method(String aString)    {
 6              this.aString = new String("set in method");
 7              System.out.println("2. method:aString:" + this.aString );
 8        }
 9        public void test()    {
10              String aString = new String("set in test");
11
12              System.out.println("1. test:aString:" + aString );
13              method(aString);                // is there a way that "set in method"
14              System.out.println("3. test:aString:" + aString );
15        }
16        public static void main(String args[] )       {
17              new Scope_1().test();
18        }
19      }
```

Source Code: Src/5/Scope_1.java

```
 1      // see http://docs.oracle.com/javase/specs/jls/se7/html/jls-6.html#jls-6.3 exa
 2      public class  Scope_2   {
 3
 4        String aString = null;
 5
 6        public void test()    {
 7              String aString = new String("set in test");
 8              //if ( true )
 9              {
10                      String aString = new String("set in test");
11              }
12        }
13        public static void main(String args[] )       {
14              new Scope_2().test();
15        }
16      }
```

Source Code: Src/5/Scope_2.java

```
 1      // see http://docs.oracle.com/javase/specs/jls/se7/html/jls-6.html#jls-6.3 exa
 2      public class  Scope_3   {
 3
 4        String aString = null;
 5
 6        public void test()    {
 7              int i;
```

```
 8
 9              class Test {
10                      int i;
11              }
12
13              for (int index = 0; index < 10; index ++ )      {
14                      System.out.println("index = " + index );
15              }
16        }
17        public static void main(String args[] )       {
18              new Scope_3().test();
19        }
20      }
```

Source Code: Src/5/Scope_3.java

```
 1      // see http://docs.oracle.com/javase/specs/jls/se7/html/jls-6.html#jls-6.3 exa
 2      public class  Scope_4   {
 3
 4        String aString = null;
 5
 6        public static void test_2()   {
 7              int k = 0;
 8              for (int index = 0; index < 10 ; index ++ ) {
 9                      int k = 3;
10              }
11        }
12        public static void test()     {
13              int i;
14              int k = 0;
15              switch (k) {
16                      case 1:         {
17                              int i=1;
18                              System.out.println("1: i == " + i);
19                              }
20                              break;
21                      default:
22                              System.out.println("something went wrong!");
23                              break;
24              }
25        }
26
27        public static void main(String args[] )       {
28              test();
29        }
30      }
```

Source Code: Src/5/Scope_4.java

6.12. Question arrived 22:44

- Newly declared fields can hide fields declared in a superclass or super interface. Newly

declared methods can hide, implement, or override methods declared in a superclass.
Would hiding just mean that new declared things would be accessed first, because we still can access the superclass members by means of (super.field) and/or (super.method(arg))?

- Static declaration inside a method change the lifetime of a variable.
  Can static variables be declared inside of a method?
  How does the the JVM find the variable aStaticLocalVariable?(Question from the second link)

- System.out.println("om: " + this); What kind of address is this? -> om: X@1ad086a
  Is it the value of the reference to this object? Is it some virtual address provided by the JVM?

- What are transient and volatile variables? When are they used?
  If I assign a value to the final variable at the time of declaration but I don't declare it to be static, will still there be copies of this variable with the objects or this variable will behave like a static final?

- This program is confusing for me. I am not able to comprehend the concept that it introduces.

- Some confusions regarding scopes.

```
void func(){
    int index=2;
    for (int index=0; index<5; index++) {//some code}
}
```

It doesn't compile saying that the variable "index" is already defined.

```
void func1(){
    for(int index=0; index<5; index++){//some code}

    System.out.println(index); // Does not compile saying that the variable "index" ca
}
```

Similar thing happens in switch case.
I also read somewhere that hiding of parameter inside the method body is illegal. What happens in the case of inner classes?

- *s.findInLine("(\d+) fish (\d+) fish (\w+) fish (\w+)");*
  Which of the following method executes it?
  StringfindInLine(Pattern pattern) OR *StringfindInLine(String pattern)*
  Is it a regular expression? If yes, why is the argument using the delimiter to be " fish ". It should have taken the entire thing as one.

- How does the auto-boxing actually proceeds in the following program from this link:-

```
public class SimpleBoxingI {
   public static void main( String[] args ) {
      Integer n1 = new Integer( 42 );
      int i1 = n1;
      int i2 = 31;
      Integer n2 = i1 + i2 + n1; // are all of them boxed?
      System.out.println(n1 + " " + n2 + " " + i1 + " " + i2 );
   } }
```

One more question regarding boxing:-

```
class  StringToInt
{
   public static void main(String args[])
   {
      int i;
      Integer aInt = new Integer("4");

      i=aInt;
//Will the difference between the line above and the line below be auto unboxing a
      i = aInt.intValue();
      i = Integer.parseInt("4");

   }
}
```

•

```
public class SimpleBoxingTypes {
    public static void main( String[] args ) {
        Float f1 = new Float( 42F );
        Integer i1 = new Integer( 42 );
        Double d1 = new Double( 42.0 );

        double f  = f1 + (double)i1 + (double)d1;
        // float ff  = f1 + (float)i1 + ((float)d1);

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

Boxing - primitive to Object type
Unboxing - Object to primitive type
What is the concept I am missing in this example? I see only type casting.
Secondly, why does the commented float line doesn't compile? Why does it start compiling when I use the primitive types instead of object types? Why do the boxing and unboxing things don't help? Does this demonstrate some type of a difference between the two types?

- I see a break statement at the end of the default case everywhere? Does anything significantly change if I don't provide it.
- Each parameter specifier consists of a type and an identifier (optionally followed by brackets) that specifies the name of the parameter.
  Where can we use brackets?

7. Inheritance

See also Java has simple inheritance, i.e., a class can only extend one superclass.

Class Members

The members of a class type are all of the following:

- Members inherited from its direct superclass, except in class Object, which has no direct superclass
- Members inherited from any direct super interfaces
- Members declared in the body of the class.
- Is a relationship

Members of a class that are declared private are not inherited by subclasses of that class. Only members of a class that are declared protected or public are inherited by subclasses declared in a package other than the one in which the class is declared.

Constructors and static initializers are not members and therefore are not inherited.

7.1. Syntax

```
class subClassName [ extends superClassName ]
{
 ...
}
```

Super class is object, if *extends superClassName* is missing.

See

7.2. Constructor Sequence

```
 1      public class S5 {
 2
 3        public int instanceV = 1;
 4
 5        public String toString()      {
 6              return "S5: " + instanceV;
 7        }
 8        public void both()    {
 9              instanceV = 200;
10        }
11        public static void main(String args[]) {
12              System.out.println(new S5());
13        }
14      }
15
```

Source Code: Src/6/B.java

```
 1      class BB extends  B{
 2
 3        public int x;
 4
 5        public BB()   {
 6              System.out.println(getClass() + "/ public BB()" );
 7              this.x = x;
 8        }
 9        public BB(int x)      {
10              System.out.println(getClass() + "/ public BB(int x)" );
11              this.x = x;
12        }
13        public String toString()      {
14              return getClass() + "/"  + x ;
15        }
16        public static void main(String args[])        {
17              System.out.println("1: " + new BB());
18              System.out.println("2: " + new BB(42));
19        }
20      }
```

Source Code: Src/6/BB.java

```
% java BB
class BB/ public B()
class BB/ public BB()
1: class BB/0
class BB/ public B()
class BB/ public BB(int x)
2: class BB/42
```

7.3. How to get access to super class methods/variables?

- Super class:

```
 1      public class S {
 2
 3        public int intS;              // what is the value of intS?
 4
 5        public S ()   {
 6              System.out.println("in S constructor");
 7        }
 8        public S method(int x)        {
 9              intS = x;
10              System.out.println("in S!ups");
11              return this;
12        }
13        public String toString()      {
14              return "S: " + intS;
15        }
16        public static void main(String args[])        {
17              System.out.println("new S()     " +  new S());
18        }
19      }
```

Source Code: Src/6/S.java

- Sub class:

```
 1      public class SubclassOfS extends S {
 2
 3        public int intS;
 4
 5        public SubclassOfS () {
 6              System.out.println("in SubclassOfS constructor");
 7        }
 8
 9        public S method(int x)        {
10              intS = x;
11              System.out.println("in SubclassOfS!method");
12              super.method(9);
13              System.out.println("4. super: " + super.toString() );
14              super.intS = 4;
15              System.out.println("5. super: " + super.toString() );
16              return (S)this;
17        }
18        public String toString()      {
19              return "SSubclassOfS: " + intS;
20        }
21
22
23        public static void main(String args[])        {
24              SubclassOfS aSubclassOfS = new SubclassOfS();
25              S  aS =  aSubclassOfS.method(42);
26              // System.out.println(aS);
```

```
27              // System.out.println(aSubclassOfS);
28              System.out.println("1. SubclassOfS!intS      = "
29                                 + aSubclassOfS.intS);
30              System.out.println("2. ((S)SubclassOfS)!intS = "
31                                 + ((S)aSubclassOfS).intS);
32      //      method(3);              // <--- what is the problem here ...
33
34        }
35      }
```

Source Code: Src/6/SubclassOfS.java

Execution:

```
% java SubclassOfS
in S constructor
in SubclassOfS constructor
in SubclassOfS!snoopy
in S!ups
SubclassOfS!intS      = 42
((S)SubclassOfS)!intS = 4
```

7.4. Private, Protected and Final I

Access control can be specified in a class, interface, method, or field declaration to control when access to a member is allowed. Access is a different concept from scope; access specifies the part of the Java program text within which the declared entity can be referred to by a qualified name, a field access expression, or a method invocation expression in which the method is not specified by a simple name. The default access is that a member can be accessed anywhere within the package that contains its declaration; other possibilities are public, protected, and private.

7.5. Determining Accessibility

Whether a package is accessible is determined by the host system . If a class or interface type is declared public, then it may be accessed by any Java code that can access the package in which it is declared. If a class or interface type is not declared public, then it may be accessed only from within the package in which it is declared.

A member type or a constructor of a class type is accessible only if the type is accessible and the member or constructor is declared to permit access:

- If the member or constructor is declared public, then access is permitted.
- Protected methods and variables are accessible to subclasses and provide a way of opening up the encapsulation.
- Private methods and variables are not accessible to subclasses.
- A final class can not be sub classed.

7.6. Packages

- A package is a set of related classes
- Classes in the same package can access each other's protected members.
- How to create:

```
package name;
```

Show Example in grapecluster ....

7.7. Polymorphism

A subclass can override an inherited method by providing a new, identical method declaration.

```
 1      public class OverWriteTop {
 2
 3        public static int var;
 4
 5        public void both(int x)       {
 6              var = x;
 7              System.out.println("    in OverWriteTop!both");
 8        }
 9        public void notBoth(int x)    {
10              var = x;
11              System.out.println("    in OverWriteTop!notBoth");
12        }
13      }
14
```

Source Code: Src/6/OverWriteTop.java

```
 1      public class OverWrite extends OverWriteTop {
 2
 3        public static int var;
 4
 5        public void both(int x)       {
 6              var = x;
 7              System.out.println("    in OverWrite!both");
 8        }
 9        public static void main(String args[])
10        {
11              OverWrite aOverWrite = new OverWrite();
12              aOverWrite.notBoth(42);
13              aOverWrite.both(84);
14              System.out.println("OverWrite.var    = " + OverWrite.var );
15              System.out.println("OverWriteTop.var = " + OverWriteTop.var );
16
17        }
18      }
```

Source Code: Src/6/OverWrite.java

Result:

```
% java OverWrite
    in OverWriteTop!notBoth
    in OverWrite!both
OverWrite.var    = 84
OverWriteTop.var = 42
```

7.8. Inner Classes

Since Java version 1.1 inner classes are possible. Inner classes cannot be declared as native, synchronized, transient or volatile.

Example:

```
 1      /**
 2       * This class implements a inner class.
 3       *
 4       * @version   $Id$
 5       *
 6       * @author    Axel T. Schreiner
 7       * @author    hp bischof
 8       *
 9       * Revisions:
10       *      $Log$
11       */
12      class InnerClass {
13          static class A {
14              static void hi () {
15                      System.err.println("4.hi");
16              }
17          }
18
19          class B {
20              void hi () {
21                      System.err.println("3.hi");
22              }
23          }
24
25          void hi () {
26                   class C {
27                      void hi () {
28                              System.err.println("2.hi");
29                      }
30                  }
31                  Object o = new C() {
32                      void hi () {
33                              System.err.println("1.hi");
34                      }
35                  };
36                  ((C)o).hi(); new C().hi(); new B().hi();
37          }
38          static public void main (String args []) {
39                   new InnerClass().hi();
40                   A.hi();
```

```
41          }
42      }
```

Source Code: Src/5/InnerClass.java

Result:

```
% java InnerClass
D.hi
C.hi
B.hi
A.hi
```

Hi.A is a nested top-level class and could be an interface. A has the same access to things as other classes in a package; more importantly, A can reach all static things in Hi.

Hi.B is a member class and cannot be an interface or contain static methods such as newInstance(). An instance of B can access all members of that instance of Hi which created it; therefore, an instance of B cannot be created in a class method of Hi. If necessary, the prefix Hi.this is used to access a member of Hi from B.

Hi.C is a local class and cannot be an interface. Methods of Hi.C can access all final local variables and parameters of it's surrounding method or block and additionally all members (or static members, depending on context) of Hi.

Hi.o is an object of an anonymous class and has the same access as an object of a local class. The class has no names and consequently no constructor; it can be subclassed from an interface.

All inner classes can have their own instance variables.

We will discuss inner classes more at the end of chapter 6.

7.9. Class Cast - 0

```
 1      public class S3 {
 2
 3        public int instanceV = 1;
 4
 5        public void set(int value)    {
 6              instanceV = value;
 7        }
 8        public String toString()      {
 9              return "S3: " + instanceV;
10        }
11
12        public static void main(String args[]) {
13              System.out.println(new S3());
14        }
15      }
16
```

Source Code: Src/6/S3.java

```
 1      public class S4 extends S3 {
 2
 3        public int instanceV = 4;
 4
 5        public void onlyInS4()        {
 6              System.out.println("S4: onlyInS4");
 7        }
```

```
 8          public void set(int value)    {
 9              instanceV = value;
10        }
11        public String toString()      {
12              return "S4: " + instanceV;
13        }
14
15        public static void main(String args[]) {
16              S4 aS4 = new S4();
17              S3 aS3 = (S3)aS4;
18
19              System.out.println("1.  aS4 =" + aS4 );
20              System.out.println("2.  aS4.instanceV = " + aS4.instanceV );
21
22              System.out.println("3.  aS3 = " + aS3 );
23              System.out.println("4.  aS3.instanceV = " + aS3.instanceV );
24
25              System.out.println("5.  S4.set(44);");
26              System.out.println("6.  S3.set(33);");
27              aS4.set(44);
28
29              System.out.println("7.  aS4 = " + aS4 );
30              System.out.println("8.  aS4.instanceV = " + aS4.instanceV );
31              System.out.println("9.  aS3 = " + aS3 );
32              System.out.println("10. aS3.instanceV = " + aS3.instanceV );
33
34              aS3.set(33);
35              System.out.println("11. aS4 = " + aS4 );
36              System.out.println("12. aS4.instanceV = " + aS4.instanceV );
37              System.out.println("13. aS3 = " + aS3 );
38              System.out.println("14. aS3.instanceV = " + aS3.instanceV );
39        }
40      }
41
```

Source Code: Src/6/S4.java

```
% java S4
aS4 =S4: 4
aS4.instanceV = 4
aS3 = S4: 4
aS3.instanceV = 1
S4.set(44);
S3.set(33);
aS4 = S4: 33
aS4.instanceV = 33
aS3 = S4: 33
aS3.instanceV = 1
```

7.10. Class Cast - I

```
 1      public class B1 {
 2
 3        public int instanceV = 1;
 4
 5        public String toString()      {
 6              return "B1: " + instanceV;
 7        }
 8        public void both()    {
 9              instanceV = 11;
10        }
11        public static void main(String args[]) {
12              System.out.println(new B1());
13        }
14      }
15
```

Source Code: Src/6/B1.java

```
 1      public class B2 extends B1 {
 2
 3        public int instanceV = 2;
 4        public boolean calledOddTimes = false;
 5
 6        public void both()    {
 7              instanceV = 22;
 8        }
 9        public void toCallSuper()     {
10              super.both();
11        }
12        public String toString()      {
13              return "B2: " + instanceV;
14        }
15
16        public static void main(String args[]) {
17              B2 aB2 = new B2();
18              B1 aB1 = (B1)aB2;
19
20              aB2.both();
21                      System.out.println("1. aB2 = " + aB2 );
22                      System.out.println("2. aB1 = " + aB1 + "\n" );
23              aB1.both();
24                      System.out.println("1. aB2 = " + aB2 );
25                      System.out.println("2. aB1 = " + aB1 + "\n" );
26
27              aB1.instanceV = 5;
28                      System.out.println("1. aB2.instanceV = " + aB2.instanceV );
29                      System.out.println("1. aB1.instanceV = " + aB1.instanceV + "\n
30
31              aB2.instanceV = 7;
32                      System.out.println("1. aB2.instanceV = " + aB2.instanceV );
33                      System.out.println("1. aB1.instanceV = " + aB1.instanceV + "\n
34
35              aB2.toCallSuper();
36                      System.out.println("X. aB2.instanceV = " + aB2.instanceV );
```

```
37                      System.out.println("Y. aB1.instanceV = " + aB1.instanceV + "\n
38
39              aB2 = (B2)aB1;          // will this compile?
40                      System.out.println("Z. aB2.instanceV = " + aB2.instanceV );
41
42
43              //  System.out.println("3. aB2.superA(): " + super.B2.instanceV);
44
45        }
46      }
47
```

Source Code: Src/6/B2.java

Output of:

```
1. aB2 = B2: 22
2. aB1 = B2: 22

1. aB2 = B2: 22
2. aB1 = B2: 22

1. aB2.instanceV = 22
1. aB1.instanceV = 5

1. aB2.instanceV = 7
1. aB1.instanceV = 5

X. aB2.instanceV = 7
Y. aB1.instanceV = 11
```

7.11. Class Cast - II

```
 1      public class S5 {
 2
 3        public int instanceV = 1;
 4        public int testIt = 1;
 5
 6        public String toString()      {
 7              return "S5: " + instanceV;
 8        }
 9        public void both()    {
10              instanceV = 11;
11        }
12        public static void main(String args[]) {
13              System.out.println(new S5());
14        }
15      }
16
```

Source Code: Src/6/S5.java

```
 1      public class S6 extends S5 {
 2
 3        public int instanceV = 2;
 4
 5        public void both()    {
 6              instanceV = 22;
 7        }
 8        public String toString()      {
 9              return "S6: " + instanceV;
10        }
11        public int superA()      {
12                return super.instanceV;
13        }
14
15        public static void main(String args[]) {
16              S6 aS6 = new S6();
17              S5 aS5 = (S5)aS6;
18
19              aS6.both();
20                      System.out.println("1. aS6 = " + aS6 );
21                      System.out.println("2. aS5 = " + aS5 + "\n" );
22              aS5.both();
23                      System.out.println("1. aS6 = " + aS6 );
24                      System.out.println("2. aS5 = " + aS5 + "\n" );
25
26              aS6.instanceV = 3;
27                      System.out.println("1. aS6 = " + aS6 );
28                      System.out.println("2. aS5 = " + aS5 + "\n" );
29
30              aS6.testIt = 4;
31              System.out.println("X. aS6.testIt " + aS6.testIt);
32              System.out.println("Z. aS5.testIt " + aS5.testIt + "\n");
33
34              aS5.testIt = 5;
35              System.out.println("XX. aS6.testIt " + aS6.testIt);
36              System.out.println("ZZ. aS5.testIt " + aS5.testIt);
37
38        }
39      }
40
```

Source Code: Src/6/S6.java

Output of:

```
1. aS6 = S6: 22
2. aS5 = S6: 22

1. aS6 = S6: 22
2. aS5 = S6: 22

1. aS6 = S6: 1
2. aS5 = S6: 1
```

7.12. Abstract Classes

An abstract class

- specifies a public method interface which can be inherited by direct or indirect subclasses.
- may declare methods, but not implement them.
- can not be instantiated.

Classes who extend an abstract class share the same, possibly extended, interface.

- Is a relationship

Use:

```
 1
 2
 3      public class TestAbstract {
 4
 5        public static void main(String args[])
 6        {
 7              Square aSquare = new Square(2);
 8              Circle aCircle = new Circle(2);
 9
10              System.out.println( "Circle");
11              System.out.println( "\t" +  aCircle.area() );
12              System.out.println( "\t" +  aCircle.perimeter() );
13
14              System.out.println( "Square");
15              System.out.println( "\t" +  aSquare.area() );
16              System.out.println( "\t" +  aSquare.perimeter() );
17
18        }
19      }
```

Source Code: Src/6b/TestAbstract.java

```
 1      /**
 2       * Abstract class
 3       * @version   $Id$
 4       *
 5       * @author    hp bischof
 6       *
 7       * Revisions:
 8       *      $Log$
 9       */
10
11      abstract class Area extends Object {
12
13        String type;
14
15        public String getType()       {
16              return type;
17        }
18
19        public abstract int area();
20        public abstract int perimeter();
21      }
```

Source Code: Src/6b/Area.java

```
 1      /**
 2       * This class implements a Circle class.
 3       *
 4       * @version   $Id$
 5       *
 6       * @author    hp bischof
 7       *
 8       * Revisions:
 9       *      $Log$
10       */
11
12      public class Circle extends Area {
13        private int radius;
14        public Circle(int _radius)    {
15              type = "Circle";
16              radius = _radius;
17        }
18
19        public int area()     {
20              return (int)(Math.PI * radius * radius);
21        }
22      //   /*
23        public int perimeter()        {
24              return (int)(Math.PI * radius * radius);
25        }
26      //   */
27      }
```

Source Code: Src/6b/Circle.java

You will get a compiler error, if a class doesn’t implement all methods.

```
% javac C*a
Circle.java:12: class Circle must be declared abstract.
It does not define int perimeter() from class Area.
public class Circle extends Area {
             ^
1 error
```

```
 1      /**
 2       * This class implements a Square class.
 3       *
 4       * @version   $Id$
 5       *
 6       * @author    hp bischof
 7       *
 8       * Revisions:
 9       *      $Log$
10       */
11
12      public class Square extends Area {
13
14        private int length;
15
16        public Square(int _length)    {
17              type = "Square";
18              length = _length;
19        }
20
21        public int area()     {
22              return length * length;
23        }
24
25        public int perimeter()        {
26              return 4 * length;
27        }
28
29      }
```

Source Code: Src/6b/Square.java

7.13. Use in an Array

- Area allTwoDThings[] = new Area [MAXIMUM];

```
 1
 2
 3      public class TestAbstract_2 {
 4
 5        static final int MAXIMUM = 4;
 6
 7        public static void main(String args[])
 8        {
 9              Area  allTwoDThings[] = new Area [MAXIMUM];
10              for ( int i = 0; i < MAXIMUM; i++ )     {
11                      if ( i % 2 == 0 )
12                              allTwoDThings[i] = new Square(2 * ( i + 24 ));
13                      else
14                              allTwoDThings[i] = new Circle(2 * ( i + 24 ));
15              }
16              int  sumOfAllAreas = 0;
```

```
17
18              for ( int i = 0; i < MAXIMUM; i++ )
19                      sumOfAllAreas += allTwoDThings[i].area();
20
21              System.out.println("sumOfAllAreas = " + sumOfAllAreas );
22        }
23      }
```

Source Code: Src/6b/TestAbstract_2.java

7.14. Class Cast and Abstract Classes

```
1       abstract class A {
2
3         public int x;
4
5         abstract public A a(int x);
6
7         public A aa(int x)    {
8               System.out.print("- in A!aa");
9               return this;
10        }
11
12      }
```

Source Code: Src/6/A.java

```
1       class AX extends A {
2
3         public int x;
4
5         public A a(int x)     {
6               System.out.print("= in AX!a");
7               return this;
8         }
9
10        public static void main(String args[])        {
11              AX aAX = new AX();
12              A  aA  = (A)aAX;
13
14              System.out.println("aAX.a(42)   " + aAX.a(42) );
15              System.out.println("aAX.a(43)   " + aAX.aa(43) );
16
17              System.out.println("aA.aa(44)   " + aA.aa(44) );
18              System.out.println("aA.a(45)    " + aA.a(45) ); // <--
19        }
20      }
```

Source Code: Src/6/AX.java

```
% java AX
= in AX!aaAX.a(42)   AX@e76cbf7
- in A!aaaAX.a(43)   AX@e76cbf7
- in A!aaaA.aa(44)   AX@e76cbf7
= in AX!aaA.a(45)    AX@e76cbf7
```

```
 1      class AXX extends A {
 2
 3        public int x;
 4
 5        public A a(int x)     {
 6              System.out.println("    in AX!a");
 7              return this;
 8        }
 9
10        public static void main(String args[])        {
11              AX  aAX   = new AX();
12              AXX aAXX  = new AXX();
13
14              System.out.println("aAX.a(42)   " + aAX.a(42) );
15              System.out.println("aAXX.a(43)  " + aAXX.a(43) );
16        }
17      }
```

Source Code: Src/6/AXX.java

7.15. Site Note: Documentation — javadoc

The various JDK packages are documented on HTML pages in a standardized format that contains many references to other classes or replaced methods etc.

creates the documentation directly from the program sources. Special comments /** ... */ before class, variable and method declarations are transferred to the documentation. The comments may contain significant elements:

@see class#method	creates a reference.
@param name text	describes a method parameter.
@return text	describes the result value of a method.
@exception class text	describes an exception.

The documentation is useful even without special comments because it describes the embedding in the class hierarchy and the redefinition of methods. References to existing documentation and graphics would have to be post processed, however.

Example:

```
% javadoc ----
javadoc: invalid flag: ----
usage: javadoc [options] [packagenames] [sourcefiles] [@files]
-overview <file>          Read overview documentation from HTML file
-public                   Show only public classes and members
-protected                Show protected/public classes and members (default)
-package                  Show package/protected/public classes and members
-private                  Show all classes and members
-help                     Display command line options
-doclet <class>           Generate output via alternate doclet
-docletpath <path>        Specify where to find doclet class files
-1.1                      Generate output using JDK 1.1 emulating doclet
-sourcepath <pathlist>    Specify where to find source files
-classpath <pathlist>     Specify where to find user class files
-bootclasspath <pathlist> Override location of class files loaded
                          by the bootstrap class loader
-extdirs <dirlist>        Override location of installed extensions
-verbose                  Output messages about what Javadoc is doing
-locale <name>            Locale to be used, e.g. en_US or en_US_WIN
-encoding <name>          Source file encoding name
-J<flag>                  Pass <flag> directly to the runtime system

Provided by Standard doclet:
-d <directory>            Destination directory for output files
-use                      Create class and package usage pages
-version                  Include @version paragraphs
-author                   Include @author paragraphs
-splitindex               Split index into one file per letter
-windowtitle <text>       Browser window title for the documenation
-doctitle <html-code>     Include title for the package index(first) page
-header <html-code>       Include header text for each page
-footer <html-code>       Include footer text for each page
-bottom <html-code>       Include bottom text for each page
-link <url>               Create links to javadoc output at <url>
-linkoffline <url> <url2> Link to docs at <url> using package list at <url2>
-group <name> <p1>:<p2>.. Group specified packages together in overview page
-nodeprecated             Do not include @deprecated information
-nodeprecatedlist         Do not generate deprecated list
-notree                   Do not generate class hierarchy
-noindex                  Do not generate index
-nohelp                   Do not generate help link
-nonavbar                 Do not generate navigation bar
-helpfile <file>          Include file that help link links to
-stylesheetfile <path>    File to change style of the generated documentation
-docencoding <name>       Output encoding name
1 error


% javadoc -d Html  TestTwoDThings.java Square.java Circle.java \
  Cube.java  TwoDThings.java
Loading source file TestTwoDThings.java...
Loading source file Square.java...
Loading source file Circle.java...
```

```
Loading source file Cube.java...
Loading source file TwoDThings.java...
Constructing Javadoc information...
Building tree for all the packages and classes...
Building index for all the packages and classes...
Generating Html/overview-tree.html...
Generating Html/index-all.html...
Generating Html/deprecated-list.html...
Building index for all classes...
Generating Html/allclasses-frame.html...
Generating Html/index.html...
Generating Html/packages.html...
Generating Html/Circle.html...
Generating Html/Cube.html...
Generating Html/Square.html...
Generating Html/TestTwoDThings.html...
Generating Html/TwoDThings.html...
Generating Html/serialized-form.html...
Generating Html/package-list...
Generating Html/help-doc.html...
Generating Html/stylesheet.css...
```

You may find the documentation here:

The makefile which I used to create the javadoc for chapter 5/Testpoint:

```
 1
 2      CLASSPATH = .#                          default explicit classpath
 3      C       = .class#                       class files
 4      J       = .java#                        Java source files
 5      JAR     = jar
 6      JAVA    = CLASSPATH=$(CLASSPATH) java
 7      JAVAC   = CLASSPATH=$(CLASSPATH) javac
 8      JAVADOC = CLASSPATH=$(CLASSPATH) javadoc
 9      JDOC    = javadoc
10
11
12      all::   $c
13
14      1:      $c
15              $(JAVA) Expression
16      2:      $c
17              @ $(JAVA) Expression -c
18      3:      $c
19              $(JAVA) Go
20
21      jdoc:
22              if [ ! -d $(JDOC) ]; then mkdir $(JDOC); fi
23              $(JAVADOC)                                          \
24                -d $(JDOC)                                                \
25                -use                                                      \
26                -splitIndex                                               \
27                -windowtitle 'Expression '                        \
28                -doctitle 'LP<sup><font size="-2">TM</font></sup> Expression' \
```

```
29                -header '<b>LP </b><br><font size="-1">v1.0</font>'   \
30                -bottom 'Copyright hpb.'      \
31                -version                                                \
32                -author Point.java TestPoint.java
```

Source Code: Src/5/makefile

7.16. Additional Examples

Given are the following class hierarchy:

•

```
 1
 2      abstract class A {
 3
 4        public abstract int isAbstract();
 5
 6        public A concrete()   {
 7              System.out.println("A!concrete()");
 8              return this;
 9        }
10
11      }
12
```

Source Code: Src/6_a/A.java

•

```
 1
 2      class B extends A {
 3
 4        public B()    {
 5              System.out.println("    B()");
 6        }
 7
 8        public int isAbstract()       {
 9              System.out.println("    B!isAbstract()");
10              return 1;
11        }
12
13        public A concrete()   {
14              System.out.println("B!concrete()");
15              return this;
16        }
17
18      }
19
```

Source Code: Src/6_a/B.java

•

```
 1
 2      class C extends A {
 3
 4        public C()    {
 5              System.out.println("    C()");
 6        }
 7
 8        public int isAbstract()       {
 9              System.out.println("    C!isAbstract()");
10              return 2;
11        }
12
13        public static void main(String args[] )       {
14              B aB = new B();
15              C aC = new C();
16
17              aB.isAbstract();
18              aC.isAbstract();
19
20              (aB.concrete()).isAbstract();
21              (aC.concrete()).isAbstract();
22
23        }
24
25      }
26
```

Source Code: Src/6_a/C.java

- Draw the class hierarchy.

```
% java C
    B()
    C()
        B!isAbstract()
        C!isAbstract()
B!concrete()
        B!isAbstract()
A!concrete()
        C!isAbstract()
```

- Why what is happening in the marked lines. Will this program compile? Will this program run?

```
 1
 2      class UseBandC {
 3
 4        public static void main(String args[] )       {
 5              int sum = 0;
 6              final int MAX;          // or final int MAX = 6;
 7              MAX = 3;
 8              Object[] aArray = new Object[MAX];      // ***
 9              // A[] aArray = new A[MAX];             // ***
10
```

```
11              for ( int i = 0; i < aArray.length; i ++ )
12                      if ( i % 2 == 0 )
13                              aArray[i] = new B();    // ***
14                      else
15                              aArray[i] = new C();    // ***
16
17              for ( int i = 0; i < aArray.length; i ++ )
18                      sum += aArray[i].isAbstract();  // ***
19                      // sum += ( (A)aArray[i]).isAbstract();  // ***
20
21        }
22
23      }
24
```

Source Code: Src/6_a/UseBandC.java

```
% java UseBandC
    B()
    C()
    B()
        B!isAbstract()
        C!isAbstract()
        B!isAbstract()
```

- Why what is happening in the marked lines. Will this program compile? Will this program run?

```
 1
 2      import java.util.Vector;
 3      class UseBandCandV {
 4
 5        public static void main(String args[] )       {
 6              int sum = 0;
 7              final int MAX = 4;
 8              Vector aVector = new Vector();
 9
10              for ( int i = 0; i < MAX; i++ )
11                      if ( i % 2 == 0 )
12                              aVector.add( new B());
13                      else
14                              aVector.add( new C());
15
16              for ( int i = 0; i < MAX; i ++ )
17              //      sum += aVector.elementAt(i).isAbstract();  ///////
18                      sum += ((A)aVector.elementAt(i)).isAbstract();  ///////
19
20        }
21
22      }
23
```

Source Code: Src/6_a/UseBandCandV.java

7.17. Interfaces

- An interface specifies which methods must be implement.
- An interface defines an public API.
- This means, we can make sure, that unrelated classes share the same part of the interface.
- An interface defines constants. They are public, static and final regardless of wether these modifiers have been specified.
- Methods implementations have been added to be able to extend the funtionality by modifying the interface and not the implementations.
- Interface methods can’t be native, static, synchronized, final, private, or protected
- Abstract and native methods can’t have a body.
- Fields in a field a static and final.

```
 1      public interface X {
 2
 3              static double MIMUM_INCREASE = 1.6 ;    // %  final
 4
 5      /*
 6       * Interface methods can’t be native,
 7       * static, synchronized, final, private, or protected
 8         Abstract and native methods can’t have a body.
 9       */
10              public void volume()
11              {
12                      System.out.println("xxxx");
13              }
14              public void setPrice(int x);
15      }
```

Source Code: Src/6c/X.java

```
 1      public interface InCommon {
 2              static double MIMUM_INCREASE = 1.6 ;
 3
 4              public void volume();
 5              public void setPrice(int x);
 6      }
```

Source Code: Src/6c/InCommon.java

```
 1      public class Phone implements InCommon {
 2
 3          int price;
 4
 5          public void volume() {
 6              System.out.println("Pott!volume: 0.5l");
 7          }
 8
 9          public void setPrice(int x) {
10              int letSee= (int)(price * MIMUM_INCREASE);
```

```
11              price = letSee > x ? x : letSee ;
12          }
13      }
14
```

Source Code: Src/6c/Phone.java

```
 1      public class VCR implements InCommon {
 2
 3          int price;
 4
 5          public void volume() {
 6              System.out.println("Mug!volume: 0.3l");
 7          }
 8
 9          public void setPrice(int x) {
10              int letSee= (int)(price * MIMUM_INCREASE);
11              price = letSee > x ? x : letSee ;
12          }
13      }
14
```

Source Code: Src/6c/VCR.java

- An interface can extend or inherit from more than one interface, and can provide implementations:

```
1       public interface C extends A,B {
2
3               int AB = 1;
4               // Attempt to reference field AB in a int.
5
6               public void c();
7       }
```

Source Code: Src/6c/C.java

```
1       public interface B {
2
3               static int B  = 2;
4               int AB = 2;
5
6               public void b();
7       }
```

Source Code: Src/6c/B.java

```
1       public interface A {
2
3               static int A  = 1;
4               int AB = 1;
5
6               public void a();
7
8               default void b(){};
9
10              default double c(){};
11      }
```

Source Code: Src/6c/A.java

```
1       public class Cuse implements C {
2
3           public void a() {
4               System.out.println("CUse!a");
5               // System.out.println("B  = " + A.AB);
6           }
7
8           public void b() {
9               System.out.println("CUse!b");
10          }
11          public void c() {
12              System.out.println("CUse!c");
13          }
```

```
14
15          public static void main(String argv[])      {
16              new Cuse().a();
17              System.out.println("A  = " + A);
18              System.out.println("B  = " + B);
19          }
20      }
21
```

Source Code: Src/6c/Cuse.java

Implementation of methods defined in an interface and class:

```
 1      public interface AA {
 2
 3              public void a();
 4
 5      /*
 6              default void b(){
 7                      System.out.println("A.b()");
 8              }
 9      AAandBBuse.java:1: error: class AAandBBuse inherits unrelated defaults for b()
10      public class AAandBBuse implements AA, BB {
11      */
12      }
```

Source Code: Src/6c/AA.java

```
 1      public interface BB {
 2
 3              public void a();
 4
 5              default void b(){
 6                      System.out.println("A.b()");
 7              }
 8      }
```

Source Code: Src/6c/BB.java

```
 1      public class AAandBBuse implements AA, BB {
 2
 3          public void a() {           // interface
 4              System.out.println("AAandBBuse!a");
 5          }
 6
 7          public void b(){
 8              System.out.println("AAandBBuse.b()");
 9              BB.super.b();
10          }
11
12          public static void main(String argv[])      {
13              new AAandBBuse().a();
```

```
14              new AAandBBuse().b();
15          }
16      }
17
```

Source Code: Src/6c/AAandBBuse.java

>Bp AAandBBuse!a AAandBBuse.b() A.b()

7.18. Aggregation

- Aggregation is a design term, which means that you create an new object by composing it out of the others.
- Aggregation relationships are specified by classes and reflected by their instance objects.
- For example: A Cylinder class can be defined as:

```
 1      /**
 2       * This class implements a Cylinder Class
 3       * NOT COMPLETE
 4       * @version   $Id$
 5       *
 6       * @author    hp bischof
 7       *
 8       * Revisions:
 9       *      $Log$
10       */
11
12      public class Cylinder {
13        private aCircle;
14        private aRect;
15
16        public Cylinder(int _radius, _height) {
17              aCircle = new Circle(radius);
18              aRect = new Rectangle(aCircle.perimeter(), height);
19        }
20        public int area()     {
21              return aCircle.area * 2 + aRect.area();
22        }
23        ....
24      }
```

Source Code: Src/6b/Cylinder.java

7.19. Short Examples for Clarification

Default Constructor Sequence

Which constructor is called when?

```
 1      public class X_1 {
 2
 3        public X_1()    {
 4              System.out.println("    in X_1!X_1()");
 5        }
 6
 7        public X_1(int x)    {
 8              System.out.println("    in X_1!X_1(int x)");
 9        }
10
11        public X_1(int x, int y)    {
12              System.out.println("    in X_1!X_1(int x, int y)");
13        }
14
15      }
```

Source Code: Src/6g/X_1.java

```
 1      class X_2 extends X_1 {
 2
 3        public X_2()    {
 4              // super();     // default
 5              System.out.println("    in X_2!X_2()");
 6        }
 7        public X_2(int x)    {
 8              // super();     // default
 9              super(x);
10              System.out.println("    in X_2!X_2(int x)");
11        }
12
13        public X_2(int x, int y)    {
14              // super();     // default
15              System.out.println("    in X_2!X_2(int x, int y)");
16        }
17
18        public static void main(String args[])
19        {
20              X_2 aX_2 = new X_2();
21              X_2 aaX_2 = new X_2(3);
22              X_2 aaaX_2 = new X_2(3, 3);
23        }
24      }
25
```

Source Code: Src/6g/X_2.java

Result:

```
    in X_1!X_1()
        in X_2!X_2()
    in X_1!X_1(int x)
        in X_2!X_2(int x)
    in X_1!X_1()
        in X_2!X_2(int x, int y)
```

Constructor must match

Superclass has no *default()* constructor!

```
 1      public class X_1 extends Object {
 2
 3        public X_1()    {
 4              System.out.println("    in X_1!X_1()");
 5        }
 6        public X_1(int x)    {
 7              System.out.println("    in X_1!X_1(int x)");
 8        }
 9
10      }
```

Source Code: Src/6h/X_1.java

```
 1      class X_2 extends X_1 {
 2
 3      /*
 4        public X_2()    {
 5              System.out.println("    in X_2!X_2()");
 6        }
 7      */
 8
 9        public static void main(String args[])
10        {
11              X_2 aX_2 = new X_2();
12        }
13      }
14
```

Source Code: Src/6h/X_2.java

Result:

```
% javac X*a
X_2.java:3: No constructor matching X_1() found in class X_1.
  public X_2()    {
         ^
1 error
```

- Overloading of constructors is identical in behavior to overloading of methods. The overloading is resolved at compile time by each class instance creation expression.
- If a class contains no constructor declarations, then a default constructor that takes no parameters is automatically provided:
- If the class being declared is the primordial class Object, then the default constructor has an empty body.
- Otherwise, the default constructor takes no parameters and simply invokes the superclass constructor with no arguments.
- A compile-time error occurs if a default constructor is provided by the compiler but the superclass does not have a constructor that takes no arguments.

Methods

Access of methods and *super* methods.

```
 1      public class X_1 {
 2
 3        public X_1()    {
 4              System.out.println("    in X_1!X_1()");
 5        }
 6
 7        public X_1(int x)    {
 8              System.out.println("    in X_1!X_1(int x)");
 9        }
10
11        public void a()    {
12              System.out.println("    in X_1!a()");
13        }
14
15      }
```

Source Code: Src/6f/X_1.java

```
 1      class X_2 extends X_1 {
 2
 3        public X_2()    {
 4              // super();     // default
 5              super.a();
 6              System.out.println("    in X_2!X_2()");
 7        }
 8        public X_2(int x)    {
 9              // super();     // default
10              super(x);
11              System.out.println("    in X_2!X_2(int x)");
12        }
13
14        public void a()    {
15              super.a();
16              System.out.println("    in X_2!a()");
17        }
18
19
20        public static void main(String args[])
21        {
22              X_2 aX_2 = new X_2();
23              aX_2.a();
24              X_2 anOtherX_2 = new X_2(3);
25        }
26      }
27
```

Source Code: Src/6f/X_2.java

Result:

```
% java X_2
    in X_1!X_1()
    in X_1!a()
        in X_2!X_2()
    in X_1!a()
    in X_2!a()
```

Instance Variables

Which one do I get?

```
1       public class X_1 extends Object {
2
3         int x1 = 1;
4         int x2 = 11;
5
6       }
```

Source Code: Src/6i/X_1.java

```
 1      class X_2 extends X_1 {
 2
 3              int x1 = 2;
 4
 5        public void a()    {
 6              System.out.println("    in X_1!a()");
 7              System.out.println("    in X_1!a()!x1 = " + x1 );
 8              System.out.println("    in X_1!a()(X_1)this.x1 = " + this.x1 );
 9              System.out.println("    in X_1!a()!super.x1 = "
10                                              + super.x1 );
11        }
12
13        public static void main(String args[])
14        {
15              X_2 aX_2 = new X_2();
16              System.out.println("    main!x1 = " + aX_2.x1 );
17              aX_2.a();
18        }
19      }
20
```

Source Code: Src/6i/X_2.java

Result:

```
% java X_2
    main!x1 = 2
    in X_1!a()
    in X_1!a()!x1 = 2
    in X_1!a()(X_1)this.x1 = 2
    in X_1!a()!super.x1 = 1
```

Private or Protected?

Can I?

```
1       public class X_1 extends Object {
2
3         static int x1 = 1;
4         private int x2 = 11;
5
6       }
```

Source Code: Src/6k/X_1.java

```
 1      class X_2 extends X_1 {
 2
 3              static int x1 = 2;
 4              int x2 = 22;
 5
 6        public void a()    {
 7              System.out.println("    in X_1!a()");
 8              System.out.println("    in X_1!a()!x1 = " + x1 );
 9              System.out.println("    in X_1!a()!super.x1 = "
10                                              + super.x1 );
11              System.out.println("    in X_1!a()!X_1.x1 = "
12                                              + X_1.x1 );
13
14              /*   System.out.println("    in X_1!a()!super.x2 = "
15               *                              + super.x2);
16               *  X_2.java:13: Variable x2 in class
17               *               X_1 not accessible from class X_2.
18               *                                          ^
19               *
20               */
21        }
22
23        public static void main(String args[])
24        {
25              X_2 aX_2 = new X_2();
26              System.out.println("    main!x1 = " + aX_2.x1 );
27              System.out.println("    main!x1 = " + aX_2.x2 );
28              aX_2.a();
29        }
30      }
```

Source Code: Src/6k/X_2.java

Result:

```
% java X_2
    main!x1 = 2
    main!x1 = 22
    in X_1!a()
    in X_1!a()!x1 = 2
    in X_1!a()!super.x1 = 1
    in X_1!a()!X_1.x1 = 1
```

7.20. A Binary Search Tree

Interface:

```
 1      /**
 2       * This class represents objects that can be contained in nodes in a
 3       * binary tree.  At a minimum these objects know how to compare
 4       * themselves to objects belonging to the same class.
 5       *
 6       * @version    $Id$
 7       *
 8       * @author     Hans-Peter Bischof
 9       * @author     Paul Tymann
10       *
11       * Revisions:
12       *     $Log$
13       */
14
15      public interface Node {
16
17          /**
18           * Returns true if this Node is less than the Node referred to by the
19           * argument.
20           *
21           * @param    Node    the node to compare this node with
22           *
23           * @return   true if the this Node is less than the Node argument.
24           */
25
26          abstract public boolean isLess(Node aNode);
27
28          /**
29           * Returns true if this Node is equal to the Node referred to by the
30           * argument.
31           *
32           * @param    Node    the node to compare this node with
33           *
34           * @return   true if the this Node is equal to the Node argument.
35           */
36
37          abstract public boolean isEqual(Node aNode);
38
39          /**
40           * Returns true if this Node is greater than the Node referred to by the
41           * argument.
42           *
43           * @param    Node    the node to compare this node with
44           *
45           * @return   true if the this Node is greater than the Node argument.
46           */
47
48          abstract public boolean isGreater(Node aNode);
49
50      } // Node
```

```
51
52
53
54
55
56
```

Source Code: Src/6t/Node.java

Implementation I:

```
 1      /**
 2       * This class allows nodes in a binary tree to hold strings.  Since
 3       * it is an extension of the Node class, these objects can compare
 4       * themselves to similar objects.
 5       *
 6       * Note that the methods in this class take Node objects as a
 7       * parameter.  The method then casts the Node reference to a
 8       * reference to a StringNode.  Since the caller may pass a reference
 9       * to a Node object that cannot be converted to s StringNode, each of
10       * methods in this class may throw a CastClassException.
11       *
12       * @version    $Id$
13       *
14       * @author     Hans-Peter Bischof
15       * @author     Paul Tymann
16       *
17       * Revisions:
18       *     $Log$
19       */
20
21      public class StringNode implements Node {
22
23          // The string that contains the data
24
25          private String info;
26
27          /**
28           * Create a new StringNode.
29           *
30           * @param    info    the string that is to be stored in this node
31           */
32
33          public StringNode( String info ) {
34              this.info = info;
35          }
36
37          /**
38           * Returns true if this StringNode is less than the StringNode
39           * referred to by the argument.
40           *
41           * @param    StringNode    the StringNode to compare this
42           *                         StringNode with.
43           *
44           * @return   true if the this StringNode is less than the
45           *           StringNode argument.
46           *
47           * @exception    CastClassException    if the argument cannot be
48           *                                     converted to a StringNode
49           */
50
51          public boolean isLess( Node aNode ) {
```

```
52
53              return ( info.compareTo( aNode.info ) < 0 );// wrong
54          }
55
56          /**
57           * Returns true if this StringNode is equal to the StringNode
58           * referred to by the argument.
59           *
60           * @param    StringNode    the StringNode to compare this
61           *                         StringNode with.
62           *
63           * @return   true if the this StringNode is equal to the
64           *           StringNode argument.
65           *
66           * @exception    CastClassException    if the argument cannot be
67           *                                     converted to a StringNode
68           */
69
70          public boolean isGreater( Node aNode ) {
71              StringNode aStringNode = ( StringNode )aNode;
72
73              return ( info.compareTo( aStringNode.info ) > 0 );
74          }
75
76          /**
77           * Returns true if this StringNode is greater than the StringNode
78           * referred to by the argument.
79           *
80           * @param    StringNode    the StringNode to compare this
81           *                         StringNode with.
82           *
83           * @return   true if the this StringNode is greater than the
84           *           StringNode argument.
85           *
86           * @exception    CastClassException    if the argument cannot be
87           *                                     converted to a StringNode
88           */
89
90          public boolean isEqual( Node aNode ) {
91              StringNode aStringNode = ( StringNode )aNode;
92
93              return ( info.compareTo( aStringNode.info ) == 0 );
94          }
95
96          /**
97           * Return a string representation of the data contained in this
98           * StringNode.
99           *
00           * @return    a string representation of this StringNode.
01           */
02
03          public String toString() {
04              return ( info );
05          }
```

```
06
07      } // StringNode
```

Source Code: Src/6t/StringNode.java

Implementation II:

```
 1      /**
 2       * This class allows nodes in a binary tree to hold strings.  Since
 3       * it is an extension of the Node class, these objects can compare
 4       * themselves to similar objects.
 5       *
 6       * Note that the methods in this class take Node objects as a
 7       * parameter.  The method then casts the Node reference to a
 8       * reference to a IntegerNode.  Since the caller may pass a reference
 9       * to a Node object that cannot be converted to s IntegerNode, each of
10       * methods in this class may throw a CastClassException.
11       *
12       * @version    $Id$
13       *
14       * @author     Hans-Peter Bischof
15       * @author     Paul Tymann
16       *
17       * Revisions:
18       *     $Log$
19       */
20
21      public class IntegerNode implements Node {
22
23          // The string that contains the data
24
25          private Integer info;
26
27          /**
28           * Create a new IntegerNode.
29           *
30           * @param    info    the string that is to be stored in this node
31           */
32
33          public IntegerNode( int info ) {
34              this.info = new Integer (info);
35          }
36
37          /**
38           * Returns true if this IntegerNode is less than the IntegerNode
39           * referred to by the argument.
40           *
41           * @param    IntegerNode    the IntegerNode to compare this
42           *                          IntegerNode with.
43           *
44           * @return   true if the this IntegerNode is less than the
45           *           IntegerNode argument.
46           *
47           * @exception    CastClassException    if the argument cannot be
48           *                                     converted to a IntegerNode
49           */
50
51          public boolean isLess( Node aNode ) {
```

```
52              IntegerNode aIntegerNode = ( IntegerNode )aNode; // correct
53
54              return ( info.compareTo( aIntegerNode.info ) < 0 );
55          }
56
57          /**
58           * Returns true if this IntegerNode is equal to the IntegerNode
59           * referred to by the argument.
60           *
61           * @param    IntegerNode    the IntegerNode to compare this
62           *                          IntegerNode with.
63           *
64           * @return   true if the this IntegerNode is equal to the
65           *           IntegerNode argument.
66           *
67           * @exception    CastClassException    if the argument cannot be
68           *                                     converted to a IntegerNode
69           */
70
71          public boolean isGreater( Node aNode ) {
72              IntegerNode aIntegerNode = ( IntegerNode )aNode;
73
74              return ( info.compareTo( aIntegerNode.info ) > 0 );
75          }
76
77          /**
78           * Returns true if this IntegerNode is greater than the IntegerNode
79           * referred to by the argument.
80           *
81           * @param    IntegerNode    the IntegerNode to compare this
82           *                          IntegerNode with.
83           *
84           * @return   true if the this IntegerNode is greater than the
85           *           IntegerNode argument.
86           *
87           * @exception    CastClassException    if the argument cannot be
88           *                                     converted to a IntegerNode
89           */
90
91          public boolean isEqual( Node aNode ) {
92              IntegerNode aIntegerNode = ( IntegerNode )aNode;
93
94              return ( info.compareTo( aIntegerNode.info ) == 0 );
95          }
96
97          /**
98           * Return a string representation of the data contained in this
99           * IntegerNode.
00           *
01           * @return    a string representation of this IntegerNode.
02           */
03
04          public String toString() {
05              return ( info.toString() );
```

```
06          }
07
08      } // IntegerNode
```

Source Code: Src/6t/IntegerNode.java

- Get the name of a Class:

```
 1      import java.lang.reflect.Method;
 2
 3      public class PrintClassName {
 4
 5        public static void printClassName(Object o)   {
 6              System.out.println("o.getClass().getName() = " +
 7                                 o.getClass().getName());
 8        }
 9
10        public static void printMethods(Object o)     {
11              Method[] m = o.getClass().getMethods();
12              for (int i = 0; i < m.length; i ++ )
13                      System.out.println("m[" + i + "]\t=\t" +
14                              m[i].getName());
15
16        }
17
18        public static void main(String args[])        {
19              String aString = "aaa";
20              Integer aInteger = new Integer("0");
21
22              printClassName((Object)aString);
23              printClassName((Object)aInteger);
24
25              printMethods((Object)aInteger);
26        }
27      }
```

Source Code: Src/6/PrintClassName.java

7.21. Nested Classes

- Nested classes can be: non-static or static nested.
- Non-static nested classes are refered to as inner classes.
- A nested class is a class that is a member of another class.

```
class Outer{
   . . .
  class AnestedClass {
          . . .
  }
}
```

- A nested class can be declared static or not.
- A static nested class is called a static nested class.

- A non static nested class is called an inner class.
- As with static methods and variables, a static nested class is associated with its enclosing class.
- And like class methods, a static nested class cannot refer directly to instance variables or methods defined in its enclosing class-it can use them only through an object reference.
- As with instance methods and variables, an inner class is associated with an instance of its enclosing class and has direct access to that object's instance variables and methods.
- Because an inner class is associated with an instance, it cannot define any static members itself.
- Example:

```
 1      public class NestedClassEx {
 2
 3        public int inNestedClass;
 4
 5        void inNestedClass()  {
 6              System.out.println("NestedClass!inNestedClass");
 7              (new AinnerClass()).aInnerClassM2();
 8        }
 9
10        static class AstaticClass     {
11              static void aStaticClassM1()    {
12                      System.out.println("AstaticClass!aStaticClassM1");
13              }
14              void aStaticClassM2()   {
15                      System.out.println("AstaticClass!aStaticClassM2");
16              }
17        }
18
19        class AinnerClass     {
20      /*
21              static void aInnerClassM1()     {
22                      System.out.println("AinnerClass!aInnerClassM1");
23              }
24         NestedClassEx.java:15: inner classes cannot have static declarations
25              static void aInnerClassM1()     {
26       */
27
28              void aInnerClassM2()    {
29                      System.out.println("AinnerClass!aInnerClassM2");
30              }
31        }
32
33        public static void main(String args[])        {
34
35              AstaticClass.aStaticClassM1();
36              (new AstaticClass()).aStaticClassM2();
37
38              (new NestedClassEx()).inNestedClass();
39              // (new AinnerClass()).aInnerClassM2();
40        }
41      }
```

Source Code: Src/6/NestedClassEx.java

7.22. Anonymous Classes

- Inner class can be declared without naming it.
- Anonymous classes can make code difficult to read. Their use should be limit to those classes that are very small (no more than a method or two) and whose use is well-understood (like the AWT event-handling adapter classes).
- Example:

```
public static void main(String[] args) {
    JFrame f = new JFrame("RectangleDemo");
    f.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
    RectangleDemo controller = new RectangleDemo();
    controller.buildUI(f.getContentPane());
    f.pack();
    f.setVisible(true);
}
```

7.23. Static Initializer Blocks

- Static initializer blocks are primarily used for initialization.
- The code in a static initializer block is executed when the class is initialized/
- A class can have more than one static initializer block.
- S.java:

```
 1      public class S {
 2
 3        static public int intS;
 4
 5        public S ()   {
 6              System.out.println("in S constructor");
 7        }
 8
 9        static {
10              System.out.println("S:Static 1");
11        }
12
13        static {
14              System.out.println("S: Static 2");
15        }
16
17        public static void main(String args[])        {
18              System.out.println("new S()     " +  new S());
19        }
20      }
```

Source Code: Src/6_AddOn/S.java

- SubclassOfS.java:

```
 1      public class SubclassOfS extends S {
 2
 3        public int intS;
 4
 5        static {
 6              System.out.println("SubclassOfS: Static 1");
 7        }
 8
 9        public SubclassOfS () {
10              System.out.println("in SubclassOfS constructor");
11        }
12
13        public static void main(String args[])        {
14              System.out.println("In SubClass of S");
15              SubclassOfS aSubclassOfS = new SubclassOfS();
16        }
17      }
```

Source Code: Src/6_AddOn/SubclassOfS.java

7.24. Questions

- Will it compile:

```
1      public class Yellow {
2        private String yellowPrivate = "yellowPrivate";
3
4       public static void main(String args[]) {
5              System.out.println(yellowPrivate);
6       }
7      }
```

Source Code: Src/6_q/Yellow.java

- Will it compile and if yes, what is the output: (eine neue variable wird erzeugt).

```
1      public class Coke {
2        private        String cokePrivate  = "cokePrivate";
3        private        String s;
4        private static String cokePrivateS = "cokePrivateS";
5
6        public void m()       {
7              cokePrivate = "java";
8        }
9
10       public void change(String cokePrivate)        {
11             cokePrivate = "hello";
12        }
13       public void print()   {
14             System.out.println("1. cokePrivate  = " + cokePrivate  );
15             System.out.println("2. cokePrivateS = " + cokePrivateS );
16             System.out.println("---------------------------------");
17       }
18       public static void main(String args[])        {
19             Coke aCoke = new Coke();
20             aCoke.m();
21             aCoke.print();
22             aCoke.change("t");
23             aCoke.print();
24      }
25     }
```

Source Code: Src/6_q/Coke.java

- Will it compile: (die antwort zu der frage kann nur verneint werden)

```
1      public class Red {
2        private String redPrivate = "redPrivate";
3      }
```

Source Code: Src/6_q/Red.java

```
 1      public class Blue {
 2        private String bluePrivate = "bluePrivate";
 3
 4        public boolean isLess(Red aRed)       {
 5              return bluePrivate == aRed.redPrivate;
 6        }
 7
 8       public static void main(String args[]) {
 9              Red  aRed = new Red();
10              Blue aBlue = new Blue();
11              aBlue.isLess(aRed);
12       }
13      }
```

Source Code: Src/6_q/Blue.java

- Will it compile and if yes, what is the output:

```
 1      public class H {
 2        private        String hPrivate  = "hPrivate";
 3        private static String hPrivateS = "hPrivateS";
 4
 5        public H(String hPrivate)     {
 6              this.hPrivate = hPrivate;
 7        }
 8
 9        public void knete()   {
10              this = this("RIT");
11        }
12
13        public void print(String tag) {
14              System.out.println(tag + "hPrivate  = " + hPrivate  );
15        }
16
17        public static void main(String args[])        {
18              H aH = new H();
19              aH.print("1.");
20              aH.knete();
21              aH.print("2.");
22       }
23      }
```

Source Code: Src/6_q/H.java

```
H.java:10: Invalid left hand side of assignment.
        this = this("RIT");
        ^
H.java:10: Only constructors can invoke constructors.
        this = this("RIT");
                    ^
H.java:18: No constructor matching H() found in class H.
        H aH = new H();
```

- Will it compile and if yes, what is the output:

```
 1      public class Bauer {
 2        private        String bauerPrivate  = "bauerPrivate";
 3        private static String bauerPrivateS = "bauerPrivateS";
 4
 5        public Bauer()        {
 6        }
 7
 8        public Bauer(String bauerPrivate)     {
 9              this.bauerPrivate = bauerPrivate;
10        }
11
12        public void knete()   {
13              this = new Bauer("RIT");
14        }
15
16        public static void main(String args[])        {
17              Bauer aBauer = new Bauer();
18       }
19      }
```

Source Code: Src/6_q/Bauer.java

- Will it compile and if yes, what is the output: (tja, das sollte nicht so schwierig sein)

```
 1      class Oh {
 2        public static void intMethod(int intArg) {
 3              intArg = 22;
 4        }
 5        public static void intArrayMethod(int[] intArg) {
 6              intArg[0] = 22;
 7        }
 8        public static void main(String[] args) {
 9              int intVariable = 2;
10              int intArray[]  = { 2, 2, 2 };
11              System.out.println("1. " + intVariable );
12              intMethod(intVariable);
13              System.out.println("2. " + intVariable );
14
15              System.out.println("3. " + intArray[0] );
16              intArrayMethod(intArray);
17              System.out.println("4. " + intArray[0] );
18           }
19      }
```

Source Code: Src/6_q/Oh.java

- Will it compile:

```
1      public interface I {
2              static int iStatic       = 1 ;
3              public int iPublic       = 6 ;
4              private int iPrivate     = 6 ;
5              protected int iProtected = 6 ;
6      }
```

Source Code: Src/6_q/I.java

- Will it compile:

```
1      public interface Ic {
2              static int iStatic       = 1 ;
3              public int iPublic       = 6 ;
4
5              public void furyo();
6      }
```

Source Code: Src/6_q/Ic.java

```
1      public class Ic1 implements Ic {
2
3       public void furyo ()   {
4              iPublic = 4;
5              iStatic = 2;
6       }
7
8      }
```

Source Code: Src/6_q/Ic1.java

- Will it compile: (eins ist genug)

```
1      public class Bier {
2        private int bier;
3
4        public Bier() {
5              bier ++;
6        }
7
8        public void print()   {
9              System.out.println("bier  = " + bier  );
10       }
11
12       public static void main(String args[])        {
13                     Bier aBier = new Bier();
14             for ( int i = 0; i < 1000; i ++ )
15                     aBier = new Bier();
16             aBier.print();
```

```
17       }
18      }
```

Source Code: Src/6_q/Bier.java

- Will it compile and what is the output: (ein Object koennte nicht zugwiesen sein)

```
 1      public class Wein {
 2        private int wein;
 3
 4        public Wein() {
 5              wein ++;
 6        }
 7
 8        public void print()   {
 9              System.out.println("wein  = " + wein  );
10        }
11
12        public static void main(String args[])        {
13              Wein aWein;
14              for ( int i = 0; i < 1000; i ++ )
15                      aWein = new Wein();
16              aWein.print();
17       }
18      }
```

Source Code: Src/6_q/Wein.java

- Will it compile and what is the output:

```
 1      public class ApfelSaft {
 2        private int gut;
 3
 4        public ApfelSaft()    {
 5              gut ++;
 6        }
 7
 8        public void print()   {
 9              System.out.println("gut  = " + gut  );
10        }
11
12        public static void main(String args[])        {
13              ApfelSaft aApfelSaft = null;
14              for ( int i = 0; i < 1000; i ++ )
15                      aApfelSaft = new ApfelSaft();
16              aApfelSaft.print();
17       }
18      }
```

Source Code: Src/6_q/ApfelSaft.java

- Will it compile and what is the output: (das dritte ist falsch)

```
 1
 2      public class SEqual {
 3
 4        public static void main(String args[])        {
 5              String s = "a";
 6              String b = "a";
 7
 8              if ( s == b )
 9                      System.out.println("1. s == b ");
10              if ( s.equals(b) )
11                      System.out.println("1. s.equals(b) ");
12
13              if ( "Furyo" == "Furyo" )
14                      System.out.println("2. \"Furyo\" == \"Furyo\" ");
15
16              s = new String("a");
17              b = new String("a");
18
19              if ( s == b )
20                      System.out.println("3. s == b ");
21              if ( s.equals(b) )
22                      System.out.println("4. s.equals(b) ");
23       }
24
25      }
```

Source Code: Src/6_q/SEqual.java

```
1. s == b
1. s.equals(b)
2. "Furyo" == "Furyo"
4. s.equals(b)
```

7.25. What is the following Example doing?

•

• Example 1:

```
 1
 2      public abstract class Node extends Number {
 3
 4        public byte byteValue () {
 5          return (byte)longValue();
 6        }
 7
 8        public short shortValue () {
 9          return (short)longValue();
10        }
11
12        public int intValue () {
13          return (int)longValue();
14        }
15
16        public float floatValue () {
17          return (float)doubleValue();
18        }
19
20        protected abstract static class Binary extends Node {
21
22          protected Number left;
23
24          protected Number right;
25
26          protected Binary () {}
27
28          protected Binary (Number left, Number right) {
29            this.left = left; this.right = right;
30          }
31        }
32
33        protected abstract static class Unary extends Node {
34
35          protected Number tree;
36
37          protected Unary (){}
38
39          protected Unary (Number tree) {
40            this.tree = tree;
41          }
42        }
43
44       public static class Add extends Binary {
45
46          public Add (Number left, Number right) {
47            super(left, right);
48          }
49
```

```
50          public long longValue () {
51            return left.longValue() + right.longValue();
52          }
53
54          public double doubleValue () {
55            return left.doubleValue() + right.doubleValue();
56          }
57        }
58
59        public static class Mul extends Binary {
60          public Mul (Number left, Number right) {
61            super(left, right);
62          }
63
64          public long longValue () {
65            return left.longValue() * right.longValue();
66          }
67
68          public double doubleValue () {
69            return left.doubleValue() * right.doubleValue();
70          }
71        }
72
73        public static class Minus extends Unary {
74          public Minus (Number tree) {
75            super(tree);
76          }
77
78          public long longValue () {
79            return - tree.longValue();
80          }
81
82          public double doubleValue () {
83            return - tree.doubleValue();
84          }
85        }
86
87        public static void main(String args[])        {
88              Node aNode = new Node.Add(new Double(1), new Double(2));
89              System.out.println("i) aNode = " + aNode.floatValue() );
90
91              aNode = new Node.Add(
92                        new Node.Mul( new Double(2), new Double(3)),
93                        new Node.Mul( new Double(4), new Double(5))
94                                  );
95              System.out.println("ii) aNode = " + aNode.floatValue() );
96              aNode = new Node.Add(
97                        new Double(1),
98                        new Node.Minus(new Double(2))
99                                  );
00              System.out.println("iii) aNode = " + aNode.floatValue() );
01        }
02      }
```

Source Code: Src/6_AddOn/Node.java

7.26. Additional Questions

7.27. Questions:

Basic Type vs. Object

```
 1      /*
 2       * Test.java
 3       *
 4       * Doubts regarding array of object creation
 5       */
 6      class A {
 7              int x = 1;
 8      }
 9      public class Test {
10              public static void main(String[] args) {
11                      int array_int[];
12                      array_int = new int[5];
13                      A cl_a = new A(); // reference to a object
14                      System.out.println("1 " + cl_a);
15                      A[] cl_array;// reference to the array
16                      cl_array = new A[5];
17                      // cl_array[1]= new A();
18                      System.out.println("2 " + cl_array);
19                      System.out.println("3 " + cl_array[1]);
20                      try {
21                              System.out.println("4 " + cl_array[1].x);
22                      } catch (Exception e) {
23                              System.out.println("5 " + e);
24
25                      }
26              }
27
28      }
```

Source Code: Src/8/Test.java

Given is the following hierarchy:

```
 1      public interface I1 {}
```

Source Code: Src/6_a/I1.java

```
 1      public interface I2 {}
```

Source Code: Src/6_a/I2.java

```
 1      public interface I3 extends I1, I2 {}
```

Source Code: Src/6_a/I3.java

```
1       public interface I4 extends I2 {}
```

Source Code: Src/6_a/I4.java

```
1       class C1 implements I3 {}
```

Source Code: Src/6_a/C1.java

```
1       class C2 implements I4 {}
2
```

Source Code: Src/6_a/C2.java

```
 1      class C3 extends C1  implements  I4 {
 2              public static void main(String args[])  {
 3                      C1 aC1 = new C1();
 4                      C2 aC2 = new C2();
 5                      C3 aC3 = new C3();
 6                      I1 aI1 = null; I2 aI2 = null; I3 aI3 = null;
 7                      aC1 = (C1)aC3;          // 1
 8                      aI1 = (I1)aC3;          // 2
 9                      aI1 = (I1)aC2;          // 3
10                      aI2 = (I2)aC3;          // 4
11                      aI1 = (aI1)aI2;         // 5
12                      aI2  = (I2)aI3;         // 6
13
14              }
15      }
```

Source Code: Src/6_a/C3.java

Draw a graphical representation of the hierarchy.

Please explain which assignment(s) are legal, and which assignment(s) are not valid.

1.
2.
3.
4.
5.

Which assignment(s) will cause a compile time error?

Which assignment(s) will cause a run time error?

8. Generics

See also: and

and stolen from their:

- Generics enable types (classes and interfaces) to be parameters when defining classes, interfaces and methods. Much like the more familiar formal parameters used in method declarations, type parameters provide a way for you to re-use the same code with different inputs. The difference is that the inputs to formal parameters are values, while the inputs to type parameters are types.

8.1. Generic Class Definition

```
class name<T1, T2, ..., Tn> { .... }
```

Type parameter names are, by convention, single, uppercase letters. This makes it easy to differentiate between a type variable and a class/interface.

Type Parameter Naming Conventions

E - Element
K - Key
N - Number
T - Type
V - Value

8.2. Old versus New

- Old:

```
List myIntList = new LinkedList();
myIntList.add(new Integer(0));
Integer x = (Integer) myIntList.iterator().next();
```

- New:

```
List<Integer> myIntList = new LinkedList<Integer>();
myIntList.add(new Integer(0));
Integer x = myIntList.iterator().next();
```

8.3. Defining Simple Generics I

•

```
1
2       public interface List<E> {
3               void add(E x);
4               Iterator<E> iterator();
5       }
6       public interface Iterator<E> {
7               E next();
8               boolean hasNext();
9       }
```

Source Code: Src/11_jdk15/List.java

8.4. Defining Simple Generics II

- Node:

```
1       public class Node<F> {
2           public Node( F value, Node<F> next ) {
3               this.value = value;
4               this.next = next;
5           }
6           public F value;
7           public Node<F> next;
8       }
```

Source Code: Src/11_jdk15/Node.java

- The Generic Stack class:

```
1       class RStack<E> {
2           public void push( E x ) {
3               head = new Node<E>(
4                           x, head );
5           }
6           public E pop() {
7               E result = head.value;
8               head = head.next;
```

```
 9              return result;
10          }
11          private Node<E> head = null;
12
13          public static void main(String args[] )     {
14              RStack<String> s;
15              s = new RStack<String>();
16              s.push("hello");
17              String w = s.pop();
18          }
19      }
```

Source Code: Src/11_jdk15/RStack.java

- How about legacy code:

```
 1      class Stack {
 2
 3          public static void main(String args[] )     {
 4              RStack s;
 5              s = new RStack();
 6              s.push("hello");
 7              String w = (String)s.pop();
 8          }
 9      }
```

Source Code: Src/11_jdk15/Stack.java

```
% javac  Stack.java
Note: Stack.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details
```

8.5. Remarks

- A compile-time only language extension.
- Parameterized types are NOT macro expanded. i.e., no real instantiation
- Compiler forgets type parameters after checking. Some usage limitations (arrays, instanceof) due to lack of guaranteed run-time type knowledge
- Type safety was primary concern. *Readability is usually improved* [Doug Lea]
- JDK 7 or later: type arguments can be empty, if the compuler can determine the type from the content

```
Box<Integer> integerBox = new Box<>();
```

8.6. An other Example

```
static void cleanUp( Collection <String>  c ) {
    for (
          Iterator <String> i = c.iterator();
          i.hasNext();
    )
        if (<String> i.next().length() == 4)
            i.remove();
```

```
}
```

8.7. Multiple Parameter Types

```
 1      import java.util.*;
 2
 3      public class MultipleTypes<E, V> {
 4              List<E>     data = new ArrayList<E>();
 5              Vector<V> volume = new Vector<V>();
 6
 7              public V getV() {
 8                      return volume.elementAt(0);
 9              }
10      }
```

Source Code: Src/11_W/MultipleTypes.java

8.8. Remarks II

- No matter to what the type parameters are bound, the code for the generic class is the same.
  - — Objects* are assumed at compile time.
  - — No basic types allowed (see autoboxing)

The concept of instantiating a generic class is misleading in Java's case Is this legal?

-

```
List<String> ls = new ArrayList<String>();   // 1
List<Object> lo = ls;                        // 2
```

- Is an ArrayList of String objects
- It is not legal, because:

```
lo.add(new Object());
String s = ls.get(0); // Versucht eine List von Objekten einer Liste mit Strings z
```

- Die Zweite Zeile wird einen C Fehler verursachen

8.9. Bounded Type Parameters

Bound Type Parameters allow to restrict the types that can be used as type arguments

```
 1      public class EvenNumbers<T extends Integer> {
 2
 3          T n;
 4
 5          public EvenNumbers(T n)  {
 6              this.n = ( n.intValue() % 2  == 0 ) ? n : null;
 7          }
 8
 9      }
```

Source Code: Src/11_W/EvenNumbers.java

8.10. Multiple Bounds

```
<T extends B & C & D>
```

8.11. A Tricky Thing

- Why is This Example Illegal?

```
 1      public class MyVector<E> {
 2              E[] data = null;
 3              public MyVector( int size ) {
 4                      data = new E[size];
 5              }
 6              public E get( int i ) {
 7                      return data[i];
 8              }
 9              public void set( int i, E val ) {
10                      data[i] = val;
11              }
12      }
```

Source Code: Src/11_W/MyVector.java

```
% javac MyVector.java && java ‘echo MyVector.java > .x; sed -e ’s/.java//’ .x‘
MyVector.java:4: generic array creation
                data = new E[size];
                       ^
1 error
```

```
 1      import java.lang.reflect.Array;
 2
 3      public class MyVector1<E> {
 4              E[] data = null;
 5              public MyVector1( int size ) {
 6                      // data = (E[])new Object[size];
 7                      data = (E[])getArray( new Object().getClass(), size );
 8
 9              }
10              public E get( int i ) {
11                      return data[i];
12              }
13              public void set( int i, E val ) {
14                      data[i] = val;
15              }
16              public <E> E[] getArray(Class<E> aClass, int size) {
17                      @SuppressWarnings("unchecked")
18                              E[] arr = (E[]) Array.newInstance(aClass, size);
19
20                  return arr;
21              }
22              public <E> E[] getArray(int size) {
```

```
23                      @SuppressWarnings("unchecked")
24                              E[] arr = (E[]) Array.newInstance(new Object().getClas
25                  return arr;
26              }
27              public static void main(String args[]) {
28                      MyVector1<String> aMyVector1 = new MyVector1<String>(11);
29                      aMyVector1.set(0, "a");
30                      System.out.println("aMyVector1.get(0): " + aMyVector1.get(0));
31
32              }
33      }
34
35      /*
36      MyVector1.java:7: warning: [unchecked] unchecked cast
37                      data = (E[])getArray( new Object().getClass(), size );
38                                                 ^
39        required: E[]
40        found:    CAP#1[]
41        where E is a type-variable:
42          E extends Object declared in class MyVector1
43        where CAP#1 is a fresh type-variable:
44          CAP#1 extends Object from capture of ? extends Object
45      1 warning
46
47      */
```

Source Code: Src/11_W/MyVector1.java

```
% javac -Xlint MyVector1.java
yVector1.java:6: warning: [unchecked] unchecked cast
found   : java.lang.Object[]
required: E[]
        data = (E[])new Object[size];
                    ^
MyVector1.java:7: warning: [unchecked] unchecked cast
found   : java.lang.Object[]
required: E[]
        data = (E[])getArray( new Object().getClass(), size );
                              ^
MyVector1.java:24: warning: [unchecked] unchecked call to set(int,E) as a member of th
        aMyVector1.set(0, "a");
                       ^
3 warnings
% java MyVector1
aMyVector1.get(0): a
```

8.12. A Tricky Thing II

- Corrected Version (gives unchecked cast warning)

```
 1      public class MyVector2<E> {
 2              Object[] data = null;
 3              public MyVector2( int size ) {
 4                      data = new Object[size];
 5              }
 6              public E get( int i ) {
 7                      return (E)data[i];
 8              }
 9              public void set( int i, E val ) {
10                      data[i] = val;
11              }
12              public static void main(String args[]) {
13                      MyVector2 aMyVector2 = new MyVector2(12);
14                      aMyVector2.set(0, "a");
15                      System.out.println("aMyVector2.get(0): " + aMyVector2.get(0));
16              }
17      }
18
```

Source Code: Src/11_W/MyVector2.java

```
 # % javac  MyVector2.java
% javac -Xlint MyVector2.java
MyVector2.java:7: warning: [unchecked] unchecked cast
        return (E)data[i];
                  ^
  required: E
  found:    Object
  where E is a type-variable:
    E extends Object declared in class MyVector2
MyVector2.java:13: warning: [rawtypes] found raw type: MyVector2
        MyVector2 aMyVector2 = new MyVector2(12);
        ^
  missing type arguments for generic class MyVector2<E>
  where E is a type-variable:
    E extends Object declared in class MyVector2
MyVector2.java:13: warning: [rawtypes] found raw type: MyVector2
        MyVector2 aMyVector2 = new MyVector2(12);
                                   ^
  missing type arguments for generic class MyVector2<E>
  where E is a type-variable:
    E extends Object declared in class MyVector2
MyVector2.java:14: warning: [unchecked] unchecked call to set(int,E) as a member of th
        aMyVector2.set(0, "a");
                   ^
  where E is a type-variable:
    E extends Object declared in class MyVector2
4 warnings
Note: Recompile with -Xlint:unchecked for details.
```

8.13. Restrictions on Generics

Stolen from:

- Cannot Instantiate Generic Types with Primitive Types
- Cannot Create Instances of Type Parameters
- Cannot Declare Static Fields Whose Types are Type Parameters
- Cannot Use Casts or instanceof With Parameterized Types
- Cannot Create Arrays of Parameterized Types
- Cannot Create, Catch, or Throw Objects of Parameterized Types
- Cannot Overload a Method Where the Formal Parameter Types of Each Overload Erase to the Same Raw Type

8.14. Wildcards

In generic code, the question mark (?), called the wildcard, represents an unknown type. The wildcard can be used in a variety of situations: as the type of a parameter, field, or local variable; sometimes as a return type (though it is better programming practice to be more specific). The wildcard is never used as a type argument for a generic method invocation, a generic class instance creation, or a supertype.

8.15. Upper Bounded Wildcards

Stolen from above:

Example: public static void process(List<? extends Number>)

- List<Integer>, List<Double>, and List<Number>;
- Integer|Double extends Number

Upper Bound Wildcard: "? extends E": Denotes a family of subtypes of type Type. This is the most useful wildcard
(bounded by its super class E) A method that works on lists of Vector and the subtype of Vector, such as Stack, *List<extends Vector>*
A method that works on lists of Integer and the super types of Integer, such as Integer, Number, and Object: *List<? super Integer>*

```
 1      import java.util.*;
 2
 3      public class UpperBound {
 4          public static void main(String[] args) {
 5              List<Double>  listOfDoubles   = new LinkedList<Double>();
 6              List<Integer> listOfIntegers  = new LinkedList<Integer>();
 7
 8              listOfDoubles.add( Double.valueOf(1.0) );
 9              listOfIntegers.add( Integer.valueOf(2) );
10
11              System.out.println("sum of integer's is: " + sum(listOfIntegers));
12              System.out.println("sum of double's is:  "  + sum(listOfDoubles));
13          }
14
15          // insgtead of double?
16          private static double sum(List<? extends Number> list)  {
17              double sum=0.0;
18              for (Number i: list)
19                  sum+=i.doubleValue();
20              return sum;
21          }
22      }
```

Source Code: Src/11_W/UpperBound.java

```
% java UpperBound
sum of integer's is: 2.0
sum of double's is:  1.0
```

8.16. Unbounded Wildcards

Unbound: "?": Denotes the set of all types or any

Example: public static void printList(List<?> list)

Can print a list of any type

Useful approach:

- a method that can be implemented using functionality provided in the Object class.
- code is using methods in the generic class that don't depend on the type parameter.
- for ex: List.size or List.clear.
- Class<?> is so often used because most of the methods in Class<T> do not depend on T.

```
 1      import java.util.*;
 2
 3      public class Unbound {
 4
 5              public static void printList(List<?> list) {
 6                      for (Object elem: list)
 7                              System.out.print(elem + " ");
 8                      System.out.println();
 9              }
10
11              public static void main(String args[]) {
12                      List<Integer> listOfIntegers = Arrays.asList(1, 2, 3);
13                      List<String>  listOfStrings = Arrays.asList("a", "b", "c");
14                      printList(listOfIntegers);
15                      printList(listOfStrings);
16              }
17      }
18
```

Source Code: Src/11_W/Unbound.java

```
% java Unbound
1 2 3
a b c
```

8.17. Lower Bounded Wildcards

A lower bounded wildcard is expressed using the wildcard character ('?'), following by the super keyword, followed by its lower bound:

Eample: public static void addNumbers(List<? super Integer> list)

The above method that works on lists of Integer and the supertypes of Integer, such as Integer, Number, and Object,

```
 1      import java.util.LinkedList;
 2      import java.util.List;
 3      import java.util.Date;
 4      import java.sql.Time;
 5
 6      /*
```

```
 7      import java.util.*;
 8      import java.sql.*;
 9      */
10
11
12      class LowerBound {
13          public static void main(String[] args) {
14              List<Date>  listOfDates           = new LinkedList<Date>();
15              List<Time> listOfTimes  = new LinkedList<Time>();
16
17              listOfDates.add( new Date() );
18              listOfTimes.add( new Time(12) );
19
20              printOnlyTimeClassorSuperClass(listOfTimes);
21              printOnlyTimeClassorSuperClass(listOfDates);
22          }
23
24          public static void printOnlyTimeClassorSuperClass(List<? super Time> list)
25              System.out.println(list);
26          }
27      }
```

Source Code: Src/11_W/LowerBound.java

```
% java LowerBound
[19:00:00]
[Wed Oct 03 13:42:33 EDT 2018]
```

8.18. Summary

- Upper bound: public static void process(List<? extends Number>)
  Number class and sub classes of number, Integer, Double
- Unbound: public static void printList(List<?> list) -
  List of all types
- Lower Bounded public static void addNumbers(List<? super Integer> list)
  Integer and super classes of Integer, Integer, Numbers, Object

8.19. Wildcard Example

```
 1      import java.util.*;
 2
 3      public class WildCard {
 4
 5          public static void printCollection_2(Collection<?> c) {
 6              for (Object e : c)
 7                      System.out.println("2: " + e);
 8          }
 9          public static void printCollection(Collection c) {
10              Iterator i = c.iterator();
11              while ( i.hasNext() ) {
12                      System.out.println("1: "  + i.next());
13              }
```

```
14          }
15          public static void main(String args[]) {
16              String anArray[] = {"echoes", "Shine", "Tiger" } ;
17              List l = Arrays.asList(anArray);
18              printCollection(l);
19              printCollection_2(l);
20          }
21      }
22
```

Source Code: Src/11_W/WildCard.java

8.20. Bound Wildcards

•

```
public abstract class Shape {
        public abstract void draw(Canvas c);
}
public class Circle extends Shape {
        private int x, y, radius;
        public void draw(Canvas c) { ... } }
public class Rectangle extends Shape {
        private int x, y, width, height;
        public void draw(Canvas c) { ... } }
```

- These classes can be drawn on a canvas:

```
public class Canvas { public void draw(Shape s) {
        s.draw(this);
   }
}
```

- Any drawing will typically contain a number of shapes.
- Assuming that they are represented as a list, it would be convenient to have a method in Canvas that draws them all:

```
public void drawAll(List<Shape> shapes) {
        for (Shape s: shapes) {
                s.draw(this);
}
```

- What we really want is:

```
public void drawAll(List<? extends Shape> shapes) { ... }
```

- *class C<T extends Figure> {..}*
  - — T must descend from Figure.
  - — Allows Figure’s features to be referenced.
  - — Keyword extends is even used for interfaces!
- Example:

```
public void drawAll(List<? extends Shape> shapes) { ... }
```

- List<? extends Shape> is an example of a bounded wildcard
- However, in this case, we know that this unknown type is in fact a subtype of Shape 1. Shape is the upper bound of the wildcard.
- This defines upper bounds

- The type Collection<? super String> is a super type of any Collection where T is a super type of String
- This collection can store Strings and Objects

8.21. Model View Controller

- The MVC paradigm is a way of breaking an application, into three parts:
  - — the model,
  - — the view, and the
  - — controller.
- Input → Processing → Output
- Controller → Model → View
- Picture:

- Model: encapsulate the logic
- View: I/O, view
- Controller: command center

How does the communication work?

- method calls
- events (move: model, operations, view, events)

Persistance

- Controller
  - — + easy to test
    + makes it easier to re-use the mode
    - obviously more complex
- Model - Active Record Pattern + easy
  - harder to test
  - reusing may be negatively impacted
  - obviously more complex

8.22. Observer - Observable Model

- Simply, the Observer pattern allows one object (the observer) to watch another (the subject).
- The Java programming language provides support for the Model/View/Controller architecture with two classes:
- — object that wishes to be notified when the state of another object changes
- — any object whose state may be of interest, and in whom another object may register an interest

8.23. Observer

- Interface
- `void update(Observable o, Object arg)`
  This method is called whenever the observed object is changed. An application calls an Observable object's notifyObservers method to have all the object's observers notified of the change.

8.24. Observable

From api spec:

- This class represents an observable object, or "data" in the model-view paradigm. It can be subclassed to represent an object that the application wants to have observed.

  An observable object can have one or more observers. An observer may be any object that implements interface Observer. After an observable instance changes, an application calling the Observable's notifyObservers method causes all of its observers to be notified of the change by a call to their update method.

  The order in which notifications will be delivered is unspecified. The default implementation provided in the Observable class will notify Observers in the order in which they registered interest, but subclasses may change this order, use no guaranteed order, deliver notifications on separate threads, or may guarantee that their subclass follows this order, as they choose.

  Note that this notification mechanism is has nothing to do with threads and is completely separate from the wait and notify mechanism of class Object.

Idea:

8.25. Example

```
 1
 2
 3
 4      import java.util.Observer;
 5      import java.util.Observable;
 6
 7      class TheObserver implements Observer {
 8
 9              public TheObserver()    {
10              }
11
12              public void update( Observable aObservable, Object o ) {
13                      System.out.println("    " + this + ":  TheObserver!update");
14              }
15      }
16
17      class TheObservable extends Observable {
18
19              String aString;
20
21              public TheObservable()  {
22                      System.out.println("    " + this + ": TheObserver!TheObservabl
23              }
24
25              public void exectueSetChanged() {
26                      setChanged();
27              }
28              public void setName(String aString) {
29                      this.aString = aString;
30                      notifyObservers();
31              }
32      }
33
```

```
34      public class Test {
35
36              public static void main(String args[]) {
37                      TheObservable s = new TheObservable();
38                      TheObserver  aTheObserver = new TheObserver();
39
40                      s.addObserver(new TheObserver());
41                      s.addObserver(new TheObserver());
42                      s.addObserver(new TheObserver());
43                      System.out.println("setName(you)");
44                      s.setName("you");
45                      System.out.println("s.exectueSetChanged()");
46                      System.out.println("setName(me)");
47                      s.exectueSetChanged();
48                      s.setName("me");
49              }
50      }
```

Source Code: Src/13/Test.java

```
    TheObservable@2ac1fdc4: TheObserver!TheObservable
setName(you)
s.exectueSetChanged()
setName(me)
    TheObserver@50cbc42f:  TheObserver!update
    TheObserver@75412c2f:  TheObserver!update
    TheObserver@282ba1e:  TheObserver!update
```

Other example

```
 1
 2      import java.util.Observer;
 3      import java.util.Observable;
 4
 5      public class TheObserver implements Observer {
 6
 7              int id = 0;
 8              static int counter = 0;
 9
10              public TheObserver()    {
11                      id = counter ++;
12              }
13
14              public void update( Observable aObservable, Object o ) {
15                      if ( o instanceof String )      {
16                              System.out.println("TheObserver:update: a String objec
17                              System.out.println("TheObserver:update:o " + o );
18                      } else {
19                              System.out.println("TheObserver:update: ! a String obj
20                              System.out.println("TheObserver:update:o " + o );
21                      }
22              }
23      }
24
```

Source Code: Src/13/TheObserver.java

```
 1
 2      import java.util.Observer;
 3      import java.util.Observable;
 4
 5      public class TheObservable extends Observable {
 6
 7              String name = null;
 8              Integer value = null;
 9
10              public TheObservable(String name)       {
11                      this.name = name;
12              }
13
14              public void setName(String name) {
15                      this.name = name;
16                      setChanged();
17                      notifyObservers(name);
18              }
19              public void go(Integer value) {
20                      this.value = value;
21                      setChanged();
22                      notifyObservers(value);
23              }
24      }
```

Source Code: Src/13/TheObservable.java

```
 1      public class Main {
 2
 3              public static void main(String args[]) {
 4                      TheObservable s = new TheObservable("Dr Hook and the Medicine
 5                      TheObserver  aTheObserver = new TheObserver();
 6
 7                      s.addObserver(aTheObserver);
 8                      s.setName("you");
 9                      s.go(1234);
10              }
11      }
```

Source Code: Src/13/Main.java

Output

```
% java Main
TheObserver:update: a String object came in
TheObserver:update:o you
TheObserver:update: ! a String object came in
TheObserver:update:o 1234
```

9. Exceptions and Assertions

9.1. Exceptions

See also: and

From:

When a Java program violates the semantic constraints of the Java language, a Java Virtual Machine signals this error to the program as an exception. An example of such a violation is an attempt to index outside the bounds of an array.

Java specifies that an exception will be thrown when semantic constraints are violated and will cause a non-local transfer of control from the point where the exception occurred to a point that can be specified by the programmer. An exception is said to be thrown from the point where it occurred and is said to be caught at the point to which control is transferred.

Java programs can also throw exceptions explicitly, using throw statement.

The Java language checks, at compile time, that a Java program contains handlers for checked exceptions, by analyzing which checked exceptions can result from execution of a method or constructor.

An Error indicates are serious, most likely not recoverable, situation.

9.2. Runtime Exceptions

- unchecked exceptions classes are the class RuntimeException and its subclasses
- class Error and its subclasses

9.3. Compile Time Exceptions

- All others

9.4. Runtime Exceptions are Not Checked

The runtime exception classes (RuntimeException and its subclasses) are exempted from compile-time checking because, in the judgment of the designers of Java, having to declare such exceptions would not aid significantly in establishing the correctness of Java programs. Many of the operations and constructs of the Java language can result in runtime exceptions. The information available to a Java compiler, and the level of analysis the compiler performs, are usually not sufficient to establish that such runtime exceptions cannot occur, even though this may be obvious to the Java programmer. Requiring such exception classes to be declared would simply be an irritation to Java programmers.

For example, certain code might implement a circular data structure that, by construction, can never involve null references; the programmer can then be certain that a NullPointerException cannot occur, but it would be difficult for a compiler to prove it. The theorem-proving technology that is needed to establish such global properties of data structures is beyond the scope of this Java Language Specification.

9.5. Runtime Exceptions--The Controversy

Copied from:

- Although Java requires that methods catch or specify checked exceptions, they do not have to catch or specify runtime exceptions, that is, exceptions that occur within the Java runtime system.
- Because catching or specifying an exception is extra work, programmers may be tempted to write code that throws only runtime exceptions and therefore doesn't have to catch or specify them.
- This is "exception abuse" and is not recommended.

9.6. Throwable and Error

Please see:

- An Error is a subclass of Throwable that indicates serious problems that a reasonable application should not try to catch.
- Most such errors are abnormal conditions.
- The ThreadDeath error, though a "normal" condition, is also a subclass of Error because most applications should not try to catch it.

```
 1      public class ErrorE {
 2
 3        private void thisMethodThrowsAnE(int index) throws Exception, Error {
 4
 5              if ( index == 0 )       {
 6                      System.out.println("thisMethodThrowsAnException() ---> " );
 7                      throw new Exception("in thisMethodThrowsAnException");
 8              } else {
 9                      System.out.println("thisMethodThrowsAnError() ---> " );
10                      throw new Error("in thisMethodThrowsAnException");
11              }
12
13        }
14
15        private void caller() {
16              for ( int index = 0; index < 2; index ++ )      {
17                      try {
18                              thisMethodThrowsAnE(index);
19                      } catch (Exception e)   {
20                              e.printStackTrace();
21                      } catch (Error e)       {
22                              e.printStackTrace();
23                      } finally       {
24                              System.out.println("Finally");
25                              System.out.println("Ok, a few things to clean up" );
26                      }
27              }
28        }
29
30        public static void main(String[] args) {
31              new ErrorE().caller();
32        }
```

```
33      }
```

Source Code: Src/7/ErrorE.java

```
java ErrorE
thisMethodThrowsAnException() --->
java.lang.Exception: in thisMethodThrowsAnException
        at ErrorE.thisMethodThrowsAnE(ErrorE.java:7)
        at ErrorE.caller(ErrorE.java:18)
        at ErrorE.main(ErrorE.java:31)
Finally
Ok, a few things to clean up
thisMethodThrowsAnError() --->
java.lang.Error: in thisMethodThrowsAnException
        at ErrorE.thisMethodThrowsAnE(ErrorE.java:10)
        at ErrorE.caller(ErrorE.java:18)
        at ErrorE.main(ErrorE.java:31)
Finally
Ok, a few things to clean up
```

9.7. Try

When an exception is thrown, control is transferred from the code that caused the exception to the nearest dynamically-enclosing catch clause of a try statement that handles the exception.

Syntax:

```
try
{
    statement sequence
}
```

The exception can be thrown and caught in the same try block if necessary.

```
try
{
    f();
}
catch (ExceptionType1 e1 [ | ExceptionType2 e2 ]) {
    throw e;
}
```

9.8. Catch

It can be followed by zero or more catch blocks:

```
catch ( parameter )
{
    statement sequence
}
```

Example:

```
try {
    anObject.f();
    anObject.g();
} catch (SomeExeption_1 e) {
    // do something to recover
}
catch (SomeExeption_2 e) {
    // do something to recover
}
```

9.9. Finally

The finally block will be always executed, regardless of what happens in the try block. (with the exception of system exit)

This provides a place where you can put statements which will be always executed.

```
finally ( )
{
    statement sequence
}
```

```
 1
 2      public class Finally_1 {
 3
 4        private void caller()  {
 5                        try {
 6                                throw new Exception("in try:    caller(): in thisMetho
 7                        } catch (Exception e)   {
 8                                System.out.println("    in catch caller(): caught exce
 9                                e.printStackTrace();
10                        } finally       {
11                                System.out.println("            Finally: caller()");
12                        }
13        }
14        private void returnS()         {
15                        try {
16                                throw new Exception("in try:    returnS(): in thisMeth
17                        } catch (Exception e)   {
18                                System.out.println("    in catch(): returnS: caught ex
19                                System.out.println("    in catch(): calling System.ret
20                                return;
21                        } finally       {
22                                System.out.println("            Finally: returnS()");
23                        }
24                        // Finally_1.java:24: error: unreachable statement
25                        // System.out.println("returnS(): you will not see this line")
26        }
27        private void exit()    {
28                        try {
29                                throw new Exception("in try:    exit(): in thisMethodT
30                        } catch (Exception e)   {
31                                System.out.println("    in catch exit(): caught except
32                                System.out.println("    in catch exit() calling System
33                                System.exit(0);
34                        } finally       {
35                                System.out.println("            Finally: exit()");
36                        }
37                        System.out.println("exit(): you will not see this line");
38        }
39
40        public static void main(String[] args) {
41              new Finally_1().caller();
42              new Finally_1().returnS();
43              new Finally_1().exit();
```

```
44        }
45      }
```

Source Code: Src/7/Finally_1.java

```
    in catch caller(): caught exception
java.lang.Exception: in try: caller(): in thisMethodThrowsAnException
    at Finally_1.caller(Finally_1.java:6)
    at Finally_1.main(Finally_1.java:42)
        Finally: caller()
    in catch(): returnS: caught exception
    in catch(): calling System.returnS(0)
        Finally: returnS()
    in catch exit(): caught exception
    in catch exit() calling System.exit(0)
```

9.10. Try-with-resources Statement

Problem:

```
 // http://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html
static String readFirstLineFromFileWithFinallyBlock(String path) throws IOException {
    BufferedReader br = new BufferedReader(new FileReader(path));
    try {
        return br.readLine();
    } finally {
        if (br != null) br.close();
    }
}
```

- Resources like streams must be closed under all circumstances.
- Try-finaly can be used to do so.
- Try-with-resources statement is the way to do it.
- For example - BufferedReader, in Java SE 7 and later, implements the interface java.lang.Auto-Closeable.

Example:

```
 1      // first line
 2      import java.io.*;
 3      public class TryWithResource {
 4
 5
 6      static void readAndPrint(String inF ) throws IOException  {
 7
 8              try (
 9                      BufferedReader in = new BufferedReader( new FileReader(inF) );
10                  ) {
11                              System.out.println(in.readLine() );
12              } catch (Exception e) {
13                      System.out.println("Could not open file");
14                      e.printStackTrace();
15              }
16        }
```

```
17
18        public static void main(String args[]) {
19              if ( args.length != 1 )  {
20                      System.out.println("Usage: java FileIO file");
21              } else  {
22                      System.out.println("Inputfile: " + args[0]);
23                      try {
24                              readAndPrint(args[0]);
25                      } catch (Exception e) {
26                              e.printStackTrace();
27                      }
28              }
29        }
30      }
```

Source Code: Src/7/TryWithResource.java

Example with finally:

```
 1      // first line
 2      import java.io.*;
 3      public class TryWithOutResourceAndFinally {
 4
 5
 6      static void readAndPrint(String inF ) throws IOException  {
 7              BufferedReader in = null;
 8
 9              try     {
10                      in = new BufferedReader( new FileReader(inF) );
11                      System.out.println(in.readLine() );
12              } catch (Exception e) {
13                      System.out.println("Could not open file");
14                      e.printStackTrace();
15              } finally {
16                      if ( in != null )
17                              in.close();
18              }
19        }
20
21        public static void main(String args[]) {
22              if ( args.length != 1 )  {
23                      System.out.println("Usage: java FileIO file");
24              } else  {
25                      System.out.println("Inputfile: " + args[0]);
26                      try {
27                              readAndPrint(args[0]);
28                      } catch (Exception e) {
29                              e.printStackTrace();
30                      }
31              }
32        }
33      }
```

Source Code: Src/7/TryWithOutResourceAndFinally.java

9.11. Throw

A throw statement allows an exception to be thrown.

Syntax:

```
throw typeThrowableException;
```

Example:

```
throw new Exception("Nope, this was not too good!");
```

9.12. Exceptions are Precise

Exceptions in Java are precise: when the transfer of control takes place, all effects of the statements executed and expressions evaluated before the point from which the exception is thrown must appear to have taken place.

No expressions, statements, or parts thereof that occur after the point from which the exception is thrown may appear to have been evaluated.

If optimized code has speculatively executed some of the expressions or statements which follow the point at which the exception occurs, such code must be prepared to hide this speculative execution from the user-visible state of the Java program.

9.13. Handling Asynchronous Exceptions

- Most exceptions occur synchronously as a result of an action by the thread in which they occur
- An asynchronous exception is an exception that can potentially occur at any point in the execution of a program.
- Asynchronous exceptions are rare. They occur only as a result of:
  - An invocation of the *stop()* (Thread or ThreadGroup)
  - An internal error in the Java virtual machine

9.14. Example 1

```
 1      /**
 2       * This class plays with exceptions
 3       *
 4       * @version   $Id$
 5       *
 6       * @author    hp bischof
 7       *
 8       * Revisions:
 9       *      $Log$
10       */
11
12      public class Excep_1 {
13
14        private int convert(String s) {
15              int result = 0;
16
17              try {
18                      result = Integer.parseInt(s);
19              } catch ( NumberFormatException e ) {
20                      System.out.println("Haeh? " + e );
21                      e.printStackTrace();
22              }
23              return result;
24        }
25
26        public static void main(String[] args) {
27              new Excep_1().convert("42");
28              new Excep_1().convert("opa");
29        }
30      }
```

Source Code: Src/7/Excep_1.java

Result:

```
% java Excep_1
Haeh? java/lang/NumberFormatException: opa
java/lang/NumberFormatException: opa
        at java/lang/Integer.parseInt(Integer.java)
        at java/lang/Integer.parseInt(Integer.java)
        at Excep_1.convert(Excep_1.java:18)
        at Excep_1.main(Excep_1.java:28)
```

9.15. Example 2

```
 1      /**
 2       * This class plays with exceptions
 3       *
 4       * @version   $Id$
 5       *
 6       * @author    hp bischof
 7       *
 8       * Revisions:
 9       *      $Log$
10       */
11
12      public class Excep_2 {
13
14        private void f(int n) throws NullPointerException,
15                                     InterruptedException {
16              System.out.println("f(" + n + ")" );
17              switch (n)      {
18                      case 1:  throw new NullPointerException("1");
19                      default: throw new InterruptedException("default");
20              }
21        }
22
23        public static void main(String[] args) {
24              for (int index = 1; index < 3; index ++ )       {
25                      try {
26                              new Excep_2().f(index);
27                      } catch (NullPointerException e)        {
28                              e.printStackTrace();
29                      }
30
31                      catch (Exception e)     {
32                              System.out.println(e.getMessage() );
33                      }
34
35              }
36        }
37      }
```

Source Code: Src/7/Excep_2.java

Result:

```
% java Excep_2
f(1)
java/lang/NullPointerException: 1
        at Excep_2.f(Excep_2.java:17)
        at Excep_2.main(Excep_2.java:25)
f(2)
default
```

Typical compiler errors:

```
Exception java/lang/Exception must be caught,
or it must be declared in the throws clause of this method.
                       new Excep_2().f(3);
                                     ^
1 error
```

9.16. Example 4

```
 1      public class Excep_3 {
 2
 3        private void thisMethodThrowsAnException() throws Exception {
 4              System.out.println("thisMethodThrowsAnException() ---> " );
 5              throw new Exception("in thisMethodThrowsAnException");
 6
 7              // javac Excep_3.java
 8              // Excep_3.java:6: unreachable statement
 9              // System.out.println("thisMethodThrowsAnException() <--- " );
10              //  ^
11              // 1 error
12              // System.out.println("thisMethodThrowsAnException() <--- " );
13        }
14
15        private void caller() {
16              try {
17                      new Excep_3().thisMethodThrowsAnException();
18                      return;
19              } catch (Exception e)   {
20                      e.printStackTrace();
21                      return;
22              } finally       {
23                      System.out.println("Finally");
24                      System.out.println("Ok, a few things to clean up" );
25              }
26        }
27
28        public static void main(String[] args) {
29              new Excep_3().caller();
30        }
31      }
```

Source Code: Src/7/Excep_3.java

Result:

```
% java Excep_3
thisMethodThrowsAnException() --->
java.lang.Exception: in thisMethodThrowsAnException
        at Excep_3.thisMethodThrowsAnException(Excep_3.java:5)
        at Excep_3.caller(Excep_3.java:18)
        at Excep_3.main(Excep_3.java:30)
Finally
Ok, a few things to clean up
```

9.17. Example 4

```
 1
 2      public class Finally_3 {
 3
 4        int rValue = 0;
 5        private int caller()   {
 6                        try {
 7                                throw new Exception("in thisMethodThrowsAnException");
 8                        } catch (Exception e)   {
 9                                rValue = 1;
10                                System.out.println("caller: before return! rValue =  "
11                                return rValue;
12                                // rValue = 11;
13
14                        } finally       {
15                                System.out.println("finaly: before return! rValue =  "
16                                rValue = 111;
17                                return rValue;
18                                // rValue = 1111;
19                        }
20        }
21        public static void main(String[] args) {
22              System.out.println("Calling new Finally_3().caller() " + new Finally_3
23        }
24      }
```

Source Code: Src/7/Finally_3.java

Result:

```
% java Finally_3
caller: before return! rValue =  1
finaly: before return! rValue =  1
Calling new Finally_3().caller() 111
```

9.18. Typical Compiler Error

```
Exception java/lang/Exception must be caught,
or it must be declared in the throws clause of this method.
                        new Excep_2().f(3);
                                        ^
1 error
```

9.19. Try Example

```
 1      /**
 2       * This class plays with exceptions
 3       *
 4       * @version   $Id$
 5       *
 6       * @author    hp bischof
 7       *
 8       * Revisions:
 9       *      $Log$
10       */
11
12      public class Try {
13
14        private void f(int n) throws Exception {
15              System.out.println("f(" + n + ")" );
16              switch (n)      {
17                      case 1:  throw new NullPointerException("1");
18                      default: throw new Exception("default");
19              }
20        }
21
22        public static void main(String[] args) {
23              int countExceptions = 0;
24              for (int index = 0; index < 3; index ++ )       {
25                      try {
26                              new Try().f(index);
27                      } catch (Exception e)   {
28                              e.printStackTrace();
29                      }
30
31                      finally  {
32                              countExceptions ++;
33                      }
34              }
35              System.out.println("Caught " + countExceptions +
36                                      " exceptions.");
37        }
38      }
```

Source Code: Src/7/Try.java

Result:

```
% java Try
f(0)
java/lang/Exception: default
        at Try.f(Try.java:18)
        at Try.main(Try.java:26)
f(1)
java/lang/NullPointerException: 1
        at Try.f(Try.java:17)
        at Try.main(Try.java:26)
f(2)
```

```
java/lang/Exception: default
        at Try.f(Try.java:18)
        at Try.main(Try.java:26)
Caught 3 exceptions.
```

9.20. Example Throw and Re-throw

```
 1      // What is the exceution order?
 2
 3      public class Deep {
 4        static int exceptionCounter = 0;
 5        static final int  MAX = 2;
 6
 7        private void importantFunction(int n) throws NullPointerException,
 8                                     InterruptedException {
 9              System.out.println("importantFunction -->");
10              switch (n)      {
11                      case 1:  throw new NullPointerException("1");
12                      default: throw new InterruptedException("default");
13              }
14        }
15
16
17        private void smartFunction() throws Exception {
18              try {
19                      importantFunction(exceptionCounter);
20                      importantFunction(exceptionCounter);
21              } catch (NullPointerException e)        {
22                      e.printStackTrace();
23                      throw new Exception("Programming error, Please call 555 1234 3
24              } catch (InterruptedException e)        {
25                      e.printStackTrace();
26                      throw new Exception("User Error error, Please call your brain"
27              }
28              finally {
29                      if ( ++exceptionCounter >= MAX )        {
30                              System.err.println("Something is wrong");
31                              System.err.println("BYE");
32                              System.exit(1);////// never do this
33                      }
34              }
35        }
36
37        public static void main(String[] args) {
38              try {
39                      Deep aDeep = new Deep();
40                      System.out.println("----> ");
41                      aDeep.smartFunction();
42                      System.out.println("====> ");
43                      aDeep.smartFunction();
44              } catch (Exception e)   {
45                      System.out.println("Main ");
46                      e.printStackTrace();
47              }
48        }
49      }
```

Source Code: Src/7/Deep.java

Result:

```
% java Deep
---->
importantFunction -->
java.lang.InterruptedException: default
        at Deep.importantFunction(Deep.java:10)
        at Deep.smartFunction(Deep.java:17)
        at Deep.main(Deep.java:39)
Main
java.lang.Exception: User Error error, Please call your brain
        at Deep.smartFunction(Deep.java:24)
        at Deep.main(Deep.java:39)
```

9.21. Exceptions and Inheritance

```
 1      // What is the exceution order?
 2
 3      public class ExceptionsAndInheritance1 {
 4
 5        public void importantFunction() throws InterruptedException {
 6              System.out.println("ExceptionsAndInheritance1:importantFunction -->");
 7              throw new InterruptedException("ExceptionsAndInheritance1.java");
 8        }
 9
10        public static void main(String[] args) {
11              try {
12                      new ExceptionsAndInheritance1().importantFunction();
13              } catch (Exception e)   {
14                      System.out.println("Main ");
15                      e.printStackTrace();
16              }
17        }
18      }
```

Source Code: Src/7/ExceptionsAndInheritance1.java

```
 1      // What is the execution order?
 2
 3      public class ExceptionsAndInheritance2 extends
 4              ExceptionsAndInheritance1 {
 5
 6        public void importantFunction() {
 7              System.out.println("ExceptionsAndInheritance2:importantFunction -->");
 8        }
 9
10        public static void main(String[] args) {
11              ExceptionsAndInheritance2 e2 = new ExceptionsAndInheritance2();
12              ExceptionsAndInheritance1 e1 = (ExceptionsAndInheritance2)e2;
13                      e2.importantFunction();
14              try {
15                      e1.importantFunction();
16              } catch (Exception e)   {
17                      System.out.println("Main ");
18                      e.printStackTrace();
19              }
20        }
21      }
```

Source Code: Src/7/ExceptionsAndInheritance2.java

Will it compile? explain your answer.

9.22. A 'Real Example'

Use:

```
 1
 2
 3      public class TestAbstract {
 4
 5        public static void main(String args[])
 6        {
 7              Square aSquare;
 8              Circle aCircle;
 9
10              for (int index = 1; index >=  -1; index -= 2 ) {
11                      try {
12                              aSquare = new Square(index);
13                              aCircle = new Circle(index);
14
15                              System.out.println( "Circle");
16                              System.out.println( "\t" +  aCircle.area() );
17                              System.out.println( "\t" +  aCircle.perimeter() );
18
19                              System.out.println( "Square");
20                              System.out.println( "\t" +  aSquare.area() );
21                              System.out.println( "\t" +  aSquare.perimeter() );
22                      }
23                      catch ( Exception e )   {
24                              System.out.println(e.getMessage() );
25                      }
26              }
27
28        }
29      }
```

Source Code: Src/7/TestAbstract.java

```
 1      /**
 2       * Abstract class
 3       * @version   $Id$
 4       *
 5       * @author    hp bischof
 6       *
 7       * Revisions:
 8       *      $Log$
 9       */
10
11      abstract class Area {
12
13        public abstract int area() throws Exception;
14        public abstract int perimeter() throws Exception;
15
16      }
17
```

Source Code: Src/7/Area.java

```
 1      /**
 2       * This class implements a Circle class.
 3       *
 4       * @version   $Id$
 5       *
 6       * @author    hp bischof
 7       *
 8       * Revisions:
 9       *      $Log$
10       */
11
12      public class Circle extends Area {
13
14        private int radius;
15
16        public Circle(int _radius) throws Exception {
17              if ( radius < 0 )
18                      throw new Exception("Negativ radius (" +
19                                      radius + ") is not acceptable");
20              else
21                      radius = _radius;
22        }
23
24        public int area() throws Exception    {
25              if ( radius < 0 )
26                      throw new Exception("Circle is not initialized");
27              else
28                      return (int)(Math.PI * radius * radius);
29        }
30
31        public int perimeter() throws Exception {
32              if ( radius < 0 )
33                      throw new Exception("Circle is not initialized");
34              else
35                      return (int)(Math.PI * radius * radius);
36        }
37
38      }
```

Source Code: Src/7/Circle.java

```
 1      /**
 2       * This class implements a Square class.
 3       *
 4       * @version   $Id$
 5       *
 6       * @author    hp bischof
 7       *
 8       * Revisions:
 9       *      $Log$
10       */
11
12      public class Square extends Area {
13
14        private int length;
15
16        public Square(int _length) throws Exception {
17              if ( _length < 0 )
18                      throw new Exception("Negative length (" +
19                                      length + ") is not acceptable");
20              else
21                      length = _length;
22        }
23
24        public int area() throws Exception {
25              if ( length < 0 )
26                      throw new Exception("Square is not initialized");
27              else
28                      return length * length;
29        }
30
31        public int perimeter() throws Exception {
32              if ( length < 0 )
33                      throw new Exception("Square is not initialized");
34              else
35                      return 4 * length;
36        }
37
38      }
```

Source Code: Src/7/Square.java

9.23. Reading From Files

```
 1      import java.io.*;
 2      public class FileIO {
 3
 4        static void cp(String inF, String outF )      {
 5              BufferedReader in  = null;
 6              BufferedWriter out = null;;
 7
 8              try {
 9                      in = new BufferedReader(
10                              new FileReader(inF) );
11                      out = new BufferedWriter(
12                              new FileWriter(outF) );
13                      int oneInt;
14                      while ( ( oneInt = in.read() ) >= 0 )   {
15                              out.write(oneInt);
16                      }
17              }
18              catch ( FileNotFoundException e )       {
19                      e.printStackTrace();
20                      System.out.println("Can't find the file!");
21              }
22              catch ( IOException e ) {  // Throws: IOException !!!
23                      e.printStackTrace();
24                      System.out.println("Could not be opened for writing!");
25              }
26              catch ( Exception e )   {
27                      System.out.println("Can't find the file!");
28              }
29              finally {
30                      try {
31                              if ( in != null )
32                                      in.close();
33                              if ( out != null )
34                                      out.close();
35                      } catch ( IOException e ) {}
36              }
37        }
38
39        public static void main(String args[]) {
40              if ( args.length != 2 )
41                      System.out.println("Usage: java FileIO f1 f2");
42              else    {
43                      System.out.println(args[0] + "  " + args[1] );
44                      cp(args[0], args[1]);
45              }
46        }
47      }
```

Source Code: Src/7/FileIO.java

With try-with-resources

```
 1      import java.io.*;
 2      public class FileIO_withTR {
 3
 4        static void cp(String inF, String outF )      {
 5
 6              try (
 7                      BufferedReader in = new BufferedReader(
 8                              new FileReader(inF) );
 9                      BufferedWriter out = new BufferedWriter(
10                              new FileWriter(outF) );
11              )       {
12                      int oneInt;
13                      while ( ( oneInt = in.read() ) >= 0 )   {
14                              out.write(oneInt);
15                      }
16              }
17              catch ( FileNotFoundException e )       {
18                      e.printStackTrace();
19                      System.out.println("Can't find the file!");
20              }
21              catch ( IOException e ) {  // Throws: IOException !!!
22                      e.printStackTrace();
23                      System.out.println("Could not be opened for writing!");
24              }
25              catch ( Exception e )   {
26                      System.out.println("Can't find the file!");
27              }
28        }
29
30        public static void main(String args[]) {
31              if ( args.length != 2 )
32                      System.out.println("Usage: java FileIO_withTR f1 f2");
33              else    {
34                      System.out.println(args[0] + "  " + args[1] );
35                      cp(args[0], args[1]);
36              }
37        }
38      }
```

Source Code: Src/7/FileIO_withTR.java

See also

9.24. Incomplete Exception Index

- ArithmeticException
- ArrayIndexOutOfBoundsException
- ArrayStoreException
- ClassCastException
- ClassNotFoundException
- CloneNotSupportedException

- Exception
- IllegalAccessException
- IllegalArgumentException
- IllegalMonitorStateException
- IllegalStateException
- IllegalThreadStateException
- IndexOutOfBoundsException
- InstantiationException
- InterruptedException
- NegativeArraySizeException
- NoSuchFieldException
- NoSuchMethodException
- NullPointerException
- NumberFormatException
- RuntimeException
- SecurityException
- StringIndexOutOfBoundsException
- ...

9.25. Exception Class

See also

Exception()

Constructs an Exception with no specified detail message.

Exception(String)

Constructs an Exception with the specified detail message.

9.26. Throwable

Exception Class is a subclass of

fillInStackTrace()
: Fills in the execution stack trace.

getLocalizedMessage()
: Creates a localized description of this Throwable.

getMessage()
: Returns the detail message of this throwable object.

printStackTrace()
: Prints this Throwable and its backtrace to the standard error stream.

printStackTrace(PrintStream)
: Prints this Throwable and its backtrace to the specified print stream.

printStackTrace(PrintWriter)
: Prints this Throwable and its backtrace to the specified print writer.

toString()
: Returns a short description of this throwable object.

9.27. Create a new Exception class

Use standard inheritance techniques. The superclass should be or a sub class.

```
 1      /**
 2       * Thrown to indicate that a method has been passed
 3       * an illegal or inappropriate ssid.
 4       */
 5
 6      class  NumberException extends Exception
 7      {
 8              /**
 9               * Constructs an NumberException with no detail message
10               */
11              public NumberException()        {
12                      super();
13              }
14
15              /**
16               * Constructs an NumberException with Number.Exception detail message
17               *
18               ** @param    s    the detail message.
19               */
20              public NumberException(String s)        {
21                      super(s);
22              }
23      }
```

Source Code: Src/7/NumberException.java

9.28. Exception and Inheritance

Is the 'throws' part of the method signature?

Example:

```
Class A: private void f(int n) throws Exception {

Class B extends A: private void f(int n) throws Exception {
```

```
 1
 2      public class A {
 3
 4        private void f(int n) throws Exception {
 5              System.out.println("f(" + n + ")" );
 6              switch (n)      {
 7                      case 1:  throw new NullPointerException("1");
 8                               // break;      unreachable
 9                      default: throw new Exception("default");
10              }
11        }
12
13        public static void main(String[] args) {
14              try {
15                      new A().f(1);
16              } catch (Exception e)   {
17                      e.printStackTrace();
18              }
19        }
20      }
```

Source Code: Src/7/A.java

```
 1
 2      public class B extends A {
 3
 4        private void f(int n) {
 5              System.out.println("f(" + n + ")" );
 6        }
 7
 8        public static void main(String[] args) {
 9              new B().f(1);
10        }
11      }
```

Source Code: Src/7/B.java

9.29. Student Question

```
 1      public class Test       {
 2
 3      /*
 4      http://download.oracle.com/javase/tutorial/essential/exceptions/finally.html
 5      */
 6              public int tryCatchFinally() {
 7                   try {
 8                      try {
 9                          System.out.println("Inner TRY");
10                          int i = 1/0;
11                          System.out.println("Inner TRY after 1/0");
12                          return 1;
13                      } catch (Exception e) {
14                          System.out.println("Inner CATCH");
15                          int i = 1/0;
16                          System.out.println("Inner CATCH after 1/0");
17                          return 2;
18                      } finally {
19                          System.out.println("Inner FINALLY!");
20                          // int i = 1/0;
21                          // System.exit(1);
22                          return 3;   // what will happen if we comment this line ou
23                      }
24                   } catch (Exception e) {
25                          System.out.println("Outer Catch");
26                          return 4;
27                   } finally {
28                          System.out.println("Outer FINALLY");
29                          // System.exit(1);  // hat will happen if we comment this
30                          // return 6;        // what will happen if we comment this
31                   }
32              }
33
34
35              public static void main(String[] args ) {
36                      // return value is?
37                      System.out.println("new Test().tryCatchFinally(); = " +
38                                      new Test().tryCatchFinally() );
39              }
40      }
```

Source Code: Src/7/Test.java

- Will it compile?
- Will it execute?

9.30. Student Question II

```
 1      public class Test_2     {
 2      /*
 3      TRY
 4      CATCH
 5      nested catch
 6      CATCH after 1/0
```

```
 7      FINALLY
 8      new Test_2().tryCatchFinally(); = 3
 9      */
10      /*
11      http://download.oracle.com/javase/tutorial/essential/exceptions/finally.html
12      */
13              public int tryCatchFinally() {
14                      try {
15                          System.out.println("TRY");
16                          int i = 1/0;
17                          System.out.println("TRY after 1/0");
18                          return 1;
19                      } catch (Exception e) {
20                          System.out.println("CATCH");
21                          try {
22                                  int i = 1/0;
23
24                          } catch (Exception ee) {
25                                  System.out.println("nested catch");
26                          }
27                          System.out.println("CATCH after 1/0");
28                          return 2;
29                      } finally {
30                          System.out.println("FINALLY");
31                          return 3;   // same question
32                      }
33              }
34
35
36              public static void main(String[] args ) {
37                      // is the return value 2?
38                      System.out.println("new Test_2().tryCatchFinally(); = " +
39                                      new Test_2().tryCatchFinally() );
40              }
41      }
```

Source Code: Src/7/Test_2.java

- Will it compile?
- Will it execute?

9.31. Assertions

- assertions are used for pre/post conditions.
- An assertion is a boolean expression that a programmer specifically proclaims to be true during program runtime execution
- Declaration:

```
assert expression1;
assert expression1 : errorMessageExpr
```

9.32. Assertions: Example

```
 1      /*
 2       * Execution: java -ea Assertion_1
 3       */
 4      public class Assertion_1 {
 5              public void method( int value ) {
 6                      assert 0 <= value : -1 ;
 7                      System.out.println("asertM ---->");
 8                      System.out.println("\tvalue = " + value );
 9                      System.out.println("asertM <----");
10              }
11
12              public static void main( String[] args ) {
13                      Assertion_1 asertM = new Assertion_1();
14                      asertM.method( 1 );
15                      asertM.method( -1 );
16              }
17      }
```

Source Code: Src/7/Assertion_1.java

```
% javac Assertion_1.java
% java Assertion_1
asertM ---->
        value = 1
asertM <----
asertM ---->
        value = -1
asertM <----
% java -ea Assertion_1
asertM ---->
        value = 1
asertM <----
Exception in thread "main" java.lang.AssertionError: 0
    at Assertion_1.method(Assertion_1.java:6)
    at Assertion_1.main(Assertion_1.java:15)
make: *** [run] Error 1
```

9.33. Assertions: Processed

```
if if true
then yes
    : No further  action
else no
    if if expression2 exists
    then yes
        : Evaluate expression2 and use the  result in a single-parameter form of the A
    else np
 use the default AssertionError constructor
}
}
```

9.34. Assertions: Enable

- Java command line argument `-ea`

```
% java -verbose
    ...
    -ea[:<packagename>...|:<classname>]
    -enableassertions[:<packagename>...|:<classname>]
                  enable assertions
    -da[:<packagename>...|:<classname>]
    -disableassertions[:<packagename>...|:<classname>]
                  disable assertions
    -esa | -enablesystemassertions
                  enable system assertions
    -dsa | -disablesystemassertions
                  disable system assertions
    ...
```

9.35. Assertions: Throwable

See also:

-

```
 1      /*
 2       * Execution: java -ea:
 3       */
 4      public class Assertion_2 {
 5              public void method( int value ) {
 6                      assert 0 <= value: "Value must be postive =" + value + "=";
 7                      System.out.println("asertM ---->");
 8                      System.out.println("\tvalue = " + value );
 9                      System.out.println("asertM <----");
10              }
11
12              public static void printAssertion( AssertionError ae ) {
13                      StackTraceElement[] stackTraceElements = ae.getStackTrace();
14                      StackTraceElement stackTraceElement = stackTraceElements[ 0 ];
15
16                      System.err.println( "AssertionError" );
17                      System.err.println( "   class=   " + stackTraceElement.getClas
18                      System.err.println( "   method=  " + stackTraceElement.getMeth
19                      System.err.println( "   message= " + ae.getMessage() );
20              }
21
22              public static void main( String[] args ) {
23                      Assertion_2 asertM = new Assertion_2();
24                      try {
25                              asertM.method( 1 );
26                              asertM.method( -1 );
27                      } catch( AssertionError ae )    {
28                              printAssertion(ae);
29                      }
30              }
31      }
```

Source Code: Src/7/Assertion_2.java

•

```
% java -ea Assertion_2
asertM ---->
        value = 1
asertM <----
AssertionError
   class=   Assertion_2
   method=  method
   message= Value must be positive =-1=
```

```
 1      /*
 2       * Execution: java -ea:
 3       */
 4      public class Assertion_3 {
 5              public int method( int i ) {
 6
 7                      if (i % 3 == 0) {
 8                              System.out.println("i % 3 == 0");
 9                      } else if (i % 3 == 1) {
10                              System.out.println("i % 3 == 1");
11                      } else {
12                              System.out.println("else");
13                              assert false : i;
14                      }
15                      return 99;
16              }
17
18              public static void main( String[] args ) {
19                      Assertion_3 asertM = new Assertion_3();
20                      try {
21                              System.out.println( asertM.method( 3 ));
22                              System.out.println( asertM.method( 5 ));
23                      } catch( AssertionError ae )    {
24                              ae.printStackTrace();
25                      }
26              }
27      }
```

Source Code: Src/7/Assertion_3.java

% java -ea Assertion_3 i % 3 == 0 99 else java.lang.AssertionError: 5 at Assertion_3.method(Assertion_3.java:13) at Assertion_3.main(Assertion_3.java:22)

9.36. Assertions: Disabling

- Assertions can also be disabled down to the class level
- the command line argument `-disableassertions` (`-da` parallels the syntax of the assertion-enabling switch.

- a command line can contain as many enable- and disable-assertion switches as desired.

10. I/O: Files and Streams

See also:

File:

- Files can store persistent information.
- Objects can be stored in a file.
- Files offer random access.
- Files can be created, removed, overwritten and appended.
- Files must have a name — the name depends on the operating system. They don't depend on Java.

In Java, file I/O as well as keyboard/screen I/O is handled by streams.

10.1. Overview

- package has two inheritance chains that can be used for dealing with files. One chain starts with the abstract classes and and the second chain starts with and

- InputStream and OutputStream and their subclasses are used for input and output of byte values.

  InputStreamReader and OutputStreamWriter combine an encoding and a stream and thus are the preferred classes for text input and output. This means they are byte based.

- The Reader and Writer classes are defined for reading from character files. They provide facilities specific to character files, such as, reading a line of input.

  Reader and Writer classes are abstract — they can have abstract methods (declaration without a body) and no objects may be instantiated. Abstract classes define basic behavior for related subclasses including instance variables and some method implementations.

  Text files normally contain byte values that represent a selection of char values by way of an encoding like 8859_1 (ISO Latin-1), or a code page like Cp850 (PC Latin-1), or with the unicode transfer format UTF8.

10.2. Input Stream

For an input stream, the source of data might be a file, a String, an array of bytes, or bytes written to an output stream (typically by another thread). There are also "filter input streams" that take data from another input stream and transform or augment the data before delivering it as input. For example, a passes bytes through verbatim but counts line terminators as they are read.

The drawings have been created by

10.3. Output Stream

For an output stream, the sink of data might be a file, an array of bytes, or a buffer to be read as an input stream (typically by another thread). There are also "filter output streams" that transform or augment data before writing it to some other output stream.

An instance of class File represents a path name (a String) that might identify a particular file within a file system. Certain operations on the file system, such as renaming and deleting files, are done by this class rather than through streams.

10.4. Using Streams

No matter where the information is coming from or going to the algorithm for reading/writing is pretty much always the same:

Reading:

```
open a stream for reading
while more information
     read
     process

close stream
```

Writing:

```
open a stream for writing
while more information
     process
     write

close stream
```

10.5. File Descriptor

An instance of class represents an abstract indication of a particular file within a file system; such file descriptors are created internally by the Java I/O system.

Methods:

public native boolean valid()

Tests if this file descriptor object is valid. Returns: true if the file descriptor object represents a valid, open file or socket; false otherwise.

sync

Force all system buffers to synchronize with the underlying device. This method returns after all modified data and attributes of this FileDescriptor have been written to the relevant device(s). In particular, if this FileDescriptor refers to a physical storage medium, such as a file in a file system, sync will not return until all in-memory modified copies of buffers associated with this File Descriptor have been written to the physical medium.

From the source file (FileDescriptor.java):

```
/**
 * Force all system buffers to synchronize with the underlying
 * device.  This method returns after all modified data and
 * attributes of this FileDescriptor have been written to the
 * relevant device(s).  In particular, if this FileDescriptor
 * refers to a physical storage medium, such as a file in a file
 * system, sync will not return until all in-memory modified copies
 * of buffers associated with this FileDescriptor have been
 * written to the physical medium.
 *
 * sync is meant to be used by code that requires physical
 * storage (such as a file) to be in a known state  For
 * example, a class that provided a simple transaction facility
 * might use sync to ensure that all changes to a file caused
 * by a given transaction were recorded on a storage medium.
 *
 * sync only affects buffers downstream of this FileDescriptor.  If
 * any in-memory buffering is being done by the application (for
 * example, by a BufferedOutputStream object), those buffers must
 * be flushed into the FileDescriptor (for example, by invoking
 * OutputStream.flush) before that data will be affected by sync.
 *
 * @exception SyncFailedException
 *        Thrown when the buffers cannot be flushed,
 *        or because the system cannot guarantee that all the
 *        buffers have been synchronized with physical media.
 * @since     1.1
 */
public native void sync() throws SyncFailedException;
```

The Solaris man

sync() causes all information in memory that should be on disk to be written out. This includes modified super blocks, modified i-nodes, and delayed block I/O.

It should be used by programs that examine a file system, such as fsck.1m df.1m etc. It is mandatory before a re-boot.

The writing, although scheduled, is not necessarily completed before sync() returns. The fsync function completes the writing before it returns.

10.6. Data Processing Streams

Processing streams perform some sort of operation, such as buffering or character encoding, as they read and write. Like the data sink streams, java/io often contains pairs of streams: one that performs a particular operation during reading and another that performs the same operation (or reverses it) during writing. This table gives java/io's processing streams

Process	Character Stream	Byte Stream
Buffering	BufferedReader, BufferedWriter	BufferedInputStream, BufferedOutputStream
Filtering	FilterReader, FilterWriter	FilterInputStream, FilterOutputStream
Converting between Bytes and Characters	InputStreamReader, OutputStreamWriter	
Concatenation		SequenceInputStream
Object Serialization		ObjectInputStream, ObjectOutputStream
Data Conversion		DataInputStream, DataOutputStream
Counting	LineNumberReader	LineNumberInputStream
Peeking Ahead	PushbackReader	PushbackInputStream
Printing	PrintWriter	PrintStream

10.7. Data Sink Streams

Data sink streams read from or write to specialized data sinks such as strings, files, or pipes. Typically, for each reader or input stream intended to read from a specific kind of input source, java/io contains a parallel writer or output stream that can create it. The following table gives java/io's data sink streams.

Sink Type	Character Streams	Byte Streams
Memory	CharArrayReader,	ByteArrayInputStream,
	CharArrayWriter	ByteArrayOutputStream
	StringReader,	StringBufferInputStream
	StringWriter	StringBufferInputStream
Pipe	PipedReader,	PipedInputStream,
	PipedWriter	PipedOutputStream
File	FileReader,	FileInputStream,
	FileWriter	FileOutputStream

CharArrayReader and CharArrayWriter
ByteArrayInputStream and ByteArrayOutputStream

Use these streams to read from and write to memory. You create these streams on an existing array and then use the read and write methods to read from or write to the array.

FileReader and FileWriter
FileInputStream and FileOutputStream

Collectively called file streams, these streams are used to read from or write to a file on the native file system.

PipedReader and PipedWriter
PipedInputStream and PipedOutputStream

Implement the input and output components of a pipe. Pipes are used to channel the output from one program (or thread) into the input of another.

StringReader and StringWriter
StringBufferInputStream

Use StringReader to read characters from a String as it lives in memory. Use StringWriter to write to a String. StringWriter collects the characters written to it in a StringBuffer, which can then be converted to a String. StringBufferInputStream is similar to StringReader, except that it reads bytes from a StringBuffer.

java/io

The hierarchy of classes defined in package

10.8. A Copy Program

```
 1      import java.io.*;
 2
 3      public class InOut_1 {
 4        public static void main( String args[] ) {
 5          byte[]  buffer = new byte[1024];
 6          int     n;
 7
 8          if ( args.length < 2 )      {
 9              System.err.println(
10                   "Usage: java InOut_1 from to");
11              System.exit(1);
12          }
13
14          try (
15              DataInputStream in = new DataInputStream(
16                                       new FileInputStream(args[0]) );
17              DataOutputStream out = new DataOutputStream(
18                                       new FileOutputStream(args[1]) );
19              ) {
20
21                      while ( (n = in.read(buffer) ) != -1 ) {
22                              out.write(buffer, 0, n);
23                      }
24
25          }
26          catch ( FileNotFoundException ef)   {
27              System.out.println("File not found: " + args[1]);
28          }
29          catch ( IOException ef)     {
30              System.out.println("File not found: " + args[1]);
31          }
32          catch ( Exception e)        {
33              System.out.println("ExceptionType occurred: " +
34                      e.getMessage() );
35          }
36
37        }
38      }
```

Source Code: Src/9_was/InOut_1.java

Result:

```
% java InOut_1
% java InOut_1 InOut_2.class x
% diff InOut_1.class x
%
```

You can skip forward ...

```
 1      import java.io.*;
 2
 3      public class Skip {
 4        public static void main( String args[] ) {
 5          byte[]  buffer = new byte[1024];
 6          int     n;
 7
 8          if ( args.length < 2 )      {
 9              System.err.println(
10                   "Usage: java Skip from to");
11              System.exit(1);
12          }
13
14          try (
15              DataInputStream in = new DataInputStream(
16                                       new FileInputStream(args[0]) );
17              DataOutputStream out = new DataOutputStream(
18                                       new FileOutputStream(args[1]) );
19
20              ) {
21                      in.skipBytes(10);
22                      while ( (n = in.read(buffer) ) != -1 ) {
23                              out.write(buffer, 0, n);
24
25                  }
26          } catch ( FileNotFoundException ef) {
27              System.out.println("File not found: " + args[1]);
28          }
29          catch ( IOException ef)     {
30              System.out.println("File not found: " + args[1]);
31          }
32          catch ( Exception e)        {
33              System.out.println("ExceptionType occurred: " +
34                      e.getMessage() );
35          }
36
37        }
38      }
```

Source Code: Src/9_was/Skip.java

10.9. Size Matters: The second Copy Program

Does the size of the read and write blocks matter?

```
 1      import java.io.*;
 2
 3
 4      public class InOut_2 {
 5        static final int BUFSIZE = 1024;
 6
 7        public static void copy( String inF , String outF, int bufSize ) {
 8          DataInputStream in;
 9          DataOutputStream out;
10          byte[]  buffer = new byte[bufSize];
11          int     n;
12
13
14          try (
15              DataInputStream in = new DataInputStream(
16                                       new FileInputStream(inF) );
17              DataOutputStream out = new DataOutputStream(
18                                       new FileOutputStream(outF) );
19
20              ) {
21                      while ( (n = in.read(buffer) ) != -1 ) {
22                              out.write(buffer, 0, n);
23                      }
24
25          }
26          catch ( FileNotFoundException ef)   {
27              System.out.println(ef.getMessage() );
28          }
29          catch ( IOException ef)     {
30              System.out.println(ef.getMessage() );
31          }
32          catch ( Exception e)        {
33              System.out.println("ExceptionType occurred: " +
34                      e.getMessage() );
35          }
36        }
37
38
39        public static void main( String args[] ) {
40          int     bufSize = BUFSIZE;
41
42          if ( args.length < 2 )      {
43              System.err.println(
44                   "Usage: java InOut_1 from to [size]");
45              System.exit(1);
46          }
47
48          if ( args.length  == 3 )    {
49              try {
50                      bufSize = Integer.parseInt(args[2]);
```

```
51              }
52              catch ( NumberFormatException e )       {
53                      System.out.println("Can’t convert " + args[2]
54                              + " to an integer.");
55              }
56
57          }
58          System.out.println("BufferSize = " + bufSize);
59          copy(args[0], args[1], bufSize);
60        }
61      }
```

Source Code: Src/9_was/InOut_2.java

Result:

```
% for i in 1 2 512 1024 10240
> do
> /usr/bin/time java InOut_2 from to $i && diff from to
> done
BufferSize = 1

real       49.8
user       21.3
sys        24.4
BufferSize = 2

real       27.0
user       10.8
sys        12.3
BufferSize = 512

real        0.8
user        0.2
sys         0.2
BufferSize = 1024

real        0.8
user        0.1
sys         0.2
BufferSize = 10240

real        1.0
user        0.2
sys         0.1
```

10.10. Reading from Stdin

```
 1      import java.io.*;
 2
 3      public class stdin {
 4        public static void main( String args[] ) {
 5          LineNumberInputStream input;
 6
 7          if ( args.length > 1 )      {
 8              System.err.println(
 9                   "Usage: java stdin file-name");
10              System.exit(1);
11          }
12
13          try {
14              String line;
15      //
16              if ( args.length == 1 )
17                      input = new LineNumberInputStream(
18                                  new DataInputStream(
19                                    new FileInputStream(args[0]) ) );
20              else
21                      input = new LineNumberInputStream( System.in );
22
23
24              while ( ( input.read() )  != -1  ) {
25                      ;
26              }
27              System.out.println("# lines = " + input.getLineNumber() );
28              input.close();
29          }
30          catch ( FileNotFoundException e)    {
31              System.out.println(e.getMessage());
32          }
33          catch ( IOException e)      {
34              System.out.println(e.getMessage());
35          }
36          catch ( Exception e)        {
37              System.out.println("ExceptionType occurred: " +
38                      e.getMessage() );
39          }
40        }
41      }
```

Source Code: Src/14/stdin.java

10.11. Reading/Writing Compressed Files

```
 1
 2      /**
 3        * Opens the file with the data.
 4        * A rewind will happen, if this method gets called more than once.
 5        **/
 6          public void openFileForReading() {
 7              try {
 8                      if ( isBinaryInput )    {
 9                              bInputStream = new DataInputStream(
10                                                      new GZIPInputStream(
11                                                        new FileInputStream(inpu
12                                                      )
13                                                    );
14
15                      } else {
16                              inputStream = new BufferedReader(new FileReader(inputF
17                      }
18                      ...
19              } catch ( Exception e ) {
20                      e.printStackTrace();
21                      InOutErr.out.println("ParticleViewExtractor: -openFileForReadi
22              }
23          }
24
25
26         /*
27          * Convert Data to Binary format
28          */
29          public void doConvertToBinary(String fileName)      {
30              try {
31                      BufferedReader inputStream = new BufferedReader(new FileReader
32                      DataOutputStream outputStream = new DataOutputStream(
33                                                      new GZIPOutputStream(
34                                                        new FileOutputStream(fil
35                                                      )
36          ...                                          );
37          }
```

Source Code: Src/14/Compressed.java

10.12. A Grep Program

Extract from:

```
BufferedReader(Reader)
    Create a buffering character-input
    stream that uses a default-sized input buffer.
BufferedReader(Reader, int)
    Create a buffering character-input
    stream that uses an input buffer of the specified
    size.
```

```
 1      import java.io.*;
 2
 3      public class Grep {
 4        public static void main( String args[] ) {
 5
 6          if ( args.length < 2 )      {
 7              System.err.println(
 8                   "Usage: java Grep search-string file-name [outputfilename]");
 9              System.exit(1);
10          }
11      // Grep.java:20: error: auto-closeable resource output may not be assigned
12      //                  output = new PrintWriter( new FileWriter(args[2]) );
13      //                  ^
14      // Grep.java:22: error: auto-closeable resource output may not be assigned
15      //                  output = new PrintWriter(System.out);
16      //                  ^
17
18          try  {
19              BufferedReader input = new BufferedReader(
20                                  new FileReader(args[1])
21                                        );
22              PrintWriter output = null;
23
24              String line;
25              if ( args.length == 3 )         {
26                      output = new PrintWriter( new FileWriter(args[2]) );
27              } else
28                      output = new PrintWriter(System.out);
29
30                      while ( ( line = input.readLine() )  != null ) {
31                              if ( line.indexOf(args[0]) >= 0 )
32                                      output.println(line);
33              }
34          }
35          catch ( FileNotFoundException e)    {
36              System.out.println(e.getMessage());
37          }
38          catch ( IOException e)      {
39              System.out.println(e.getMessage());
40          }
41          catch ( Exception e)        {
42              System.out.println("ExceptionType occurred: " +
43                      e.getMessage() );
44          }
45
46        }
47      }
```

Source Code: Src/9_was/Grep.java

```
% java Grep Grep Grep.java
public class Grep {
             "Usage: java Grep search-string file-name [output-filename]");
% java Grep Grep
Usage: java Grep search-string file-name [output-filename]
```

If you are interested in line numbers, choose LineNumberReader

```
 1      import java.io.*;
 2
 3      public class Grep2 {
 4        public static void main( String args[] ) {
 5          LineNumberReader input;
 6          PrintWriter    output;
 7
 8          if ( args.length < 2 )      {
 9              System.err.println(
10                   "Usage: java Grep search-string file-name [outputfilename]");
11              System.exit(1);
12          }
13
14          try {
15              String line;
16              input = new LineNumberReader (
17                           new  BufferedReader(
18                                  new FileReader(args[1])
19                                                 )
20                                           );
21              if ( args.length == 3 )         {
22                  output = new PrintWriter( new FileWriter(args[2]) );
23              } else
24                  output = new PrintWriter(System.out);
25
26              while ( ( line = input.readLine() )  != null ) {
27                      if ( line.indexOf(args[0]) >= 0 )
28                              output.println(input.getLineNumber() +
29                                      ": " + line);
30              }
31              output.close();
32              input.close();
33          }
34          catch ( FileNotFoundException e)    {
35              System.out.println(e.getMessage());
36          }
37          catch ( IOException e)      {
38              System.out.println(e.getMessage());
39          }
40          catch ( Exception e)        {
41              System.out.println("ExceptionType occurred: " +
42                      e.getMessage() );
43          }
44
45        }
46      }
```

Source Code: Src/9_was/Grep2.java

```
% java Grep2 Grep Grep.java
3: public class Grep {
10:          "Usage: java Grep search-string file-name [output-filename]");
%
```

10.13. Regular Expressions

- See also here:

10.14. Regular Expressions in Java

See also here:

- Classes for matching character sequences against patterns specified by regular expressions.
- An instance of the Pattern class represents a regular expression that is specified in string form in a syntax similar to that used by Perl.
- Instances of the Matcher class are used to match character sequences against a given pattern.

10.15. Example 1

```
 1      /*
 2       * Checks for invalid characters
 3       * in email addresses
 4       */
 5
 6      import java.util.regex.*;
 7
 8      public class EmailValidation {
 9
10        /*
11         * Checks for email addresses starting with
12         * inappropriate symbols like dots or @ signs.
13         */
14         public static void checkForPorA(String aPossibleEmail )      {
15            Pattern p = Pattern.compile("^\\.|^\\@");
16            Matcher m = p.matcher(aPossibleEmail);
17            if (m.find())
18               System.err.println(aPossibleEmail + " - Email addresses don't start"
19                                  " with dots or @ signs.");
20            else
21               System.err.println(aPossibleEmail + " is valid.");
22         }
23
24        /*
25         * Checks for email addresses starting with
26         * www.
27         */
28         public static void checkForWWW(String aPossibleEmail )       {
29            Pattern p = Pattern.compile("^www\\.");
30            Matcher m = p.matcher(aPossibleEmail);
31            if (m.find())
32               System.err.println(aPossibleEmail + " - Email addresses don't start"
33                                  " with www.");
34            else
35               System.err.println(aPossibleEmail + " is valid.");
36         }
37
38
39        /*
```

```
40         * Checks for invalid characters in email addresses.
41         */
42         public static void checkForInvalidC(String aPossibleEmail )  {
43            Pattern p = Pattern.compile("[^A-Za-z0-9\\.\\@_\\-~#]+");
44            Matcher m = p.matcher(aPossibleEmail);
45            StringBuffer sb = new StringBuffer();
46            boolean result = m.find();
47            boolean deletedIllegalChars = false;
48
49            while(result) {
50               deletedIllegalChars = true;
51               m.appendReplacement(sb, "");
52               result = m.find();
53            }
54
55            if (deletedIllegalChars) {
56               System.out.println("It contained incorrect characters" +
57                                  " , such as spaces or commas.");
58            }
59         }
60
61
62         public static void main(String[] args) throws Exception {
63
64
65            checkForPorA("hpb@cs.rit.edu");
66            checkForPorA("@cs.rit.edu");
67
68            checkForWWW("www.cs.rit.edu");
69
70            checkForInvalidC("hpb@cs.rit.edu");
71            checkForInvalidC("p b@cs.rit.edu");
72
73         }
74      }
```

Source Code: Src/9_reg_ex/EmailValidation.java

10.16. Example 2

```
 1      /*
 2       * Checks for invalid characters
 3       * in email addresses
 4       */
 5
 6      import java.util.regex.*;
 7
 8      public class TheN {
 9
10        /*
11         * Palindroms
12         */
13         public static void checkForP(String aPossibleEmail ) {
```

```
14            Pattern p = Pattern.compile("^.$");
15            Matcher m = p.matcher(aPossibleEmail);
16            if (m.find())
17               System.err.println(aPossibleEmail + " one character");
18            else
19               System.err.println(aPossibleEmail + " more than one character");
20         }
21
22         public static void checkForP2(String aPossibleEmail )        {
23            Pattern p = Pattern.compile("^(.).\\1$");
24            Matcher m = p.matcher(aPossibleEmail);
25            if (m.find())
26               System.err.println(aPossibleEmail + " 2 char palindrom");
27            else
28               System.err.println(aPossibleEmail + " ! a 2 char palindrom");
29         }
30
31         public static void main(String[] args) throws Exception {
32
33
34            checkForP("a");
35            checkForP("aa");
36
37            checkForP2("a");
38            checkForP2("ata");
39
40            if (  Pattern.matches("^(.)\\1$", "aa" ))
41                      System.err.println("palindrom");
42
43         }
44      }
```

Source Code: Src/9_reg_ex/TheN.java

10.17. Serializing Objects

- Two of the byte streams, and are specialized streams that let you read and write objects. Reading and writing objects is a process known as object serialization. Object serialization has many uses, including remote method invocation (RMI). In addition to the object streams, java/io has other classes and interfaces that define the API to help classes perform serialization for its instances.
- Only objects that support the interface can be written to streams. The class of each serializable object is encoded including the class name and signature of the class, the values of the object's fields and arrays, and the closure of any other objects referenced from the initial objects
- Reconstructing an object from a stream requires that the object first be written to a stream.
- The default serialization mechanism for an object writes the class of the object, the class signature, and the values of all non-transient and non-static fields. References to other objects (except in transient or static fields) cause those objects to be written also. Multiple references to a single object are encoded using a reference sharing mechanism so that graphs of objects can be restored to the same shape as when the original was written.

Object writer:

```
 1      import java.io.*;
 2      import java.util.Date;
 3
 4      public class ObjectWriter_1 {
 5        public static void main( String args[] ) {
 6
 7          Date d = new Date();
 8
 9          try {
10              FileOutputStream ostream =
11                              new FileOutputStream("object_1.ser");
12              ObjectOutputStream p = new ObjectOutputStream(ostream);
13              p.writeInt(12345);
14                System.out.println("Integer = " + 1234);
15              p.writeObject("Today");
16                System.out.println("String = " + "Today");
17              p.writeObject(d);
18                System.out.println("Date = " + d);
19              p.flush();
20              p.close();
21          }
22          catch ( IOException e)      {
23              System.out.println(e.getMessage());
24          }
25
26        }
27      }
```

Source Code: Src/9_was/ObjectWriter_1.java

```
% java ObjectWriter_1
Integer = 1234
String = Today
Date = Mon Nov  1 08:42:38 EDT 2010
```

Object Reader:

```
 1      import java.io.*;
 2      import java.util.Date;
 3
 4      public class ObjectReader_1 {
 5        public static void main( String args[] ) {
 6
 7          try {
 8              FileInputStream istream =
 9                           new FileInputStream("object_1.ser");
10              ObjectInputStream p = new ObjectInputStream(istream);
11              int i = p.readInt();
12                System.out.println("Integer = " + i);
13              String today = (String)p.readObject();
14                System.out.println("String = " + today);
15              Date date = (Date)p.readObject();
16                System.out.println("Date = " + date);
17              p.close();
18              istream.close();
19          }
20          catch ( IOException e)      {
21              System.out.println(e.getMessage());
22          }
23          catch ( ClassNotFoundException e)   {
24              System.out.println(e.getMessage());
25          }
26
27        }
28      }
```

Source Code: Src/9_was/ObjectReader_1.java

```
% java ObjectReader_1
Integer = 1234
String = Today
Date = Mon Nov  1 08:42:38 EDT 2010
```

```
ls -l o*a
-rw-------   1 hpb      fac           60 Oct  4 10:49 object_1.ser
yps 9 85 od -c object_1.ser
0000000 254 355  \0 005   w 004  \0  \0   0   9   t  \0 005   T   o   d
0000020   a   y   s   r  \0 016   j   a   v   a   .   u   t   i   l   .
0000040   D   a   t   e   h   j 201 001   K   Y   t 031 003  \0  \0   x
0000060   p   w  \b  \0  \0  \0 326 365   <   o 232   x
0000074
```

When an object is serialized, any object reference it contains are also serialized. A example.

```
 1      import java.io.*;
 2      import java.util.*;
 3
 4      public class ObjectWriter_2 {
 5        public static void main( String args[] ) {
 6
 7          Hashtable aHashTable = new Hashtable();
 8          aHashTable.put("plus  Movie", "A little Voice");
 9          aHashTable.put("minus Movie", "Independence Day");
10
11          try {
12              FileOutputStream ostream =
13                           new FileOutputStream("object_2.ser");
14              ObjectOutputStream p = new ObjectOutputStream(ostream);
15              p.writeObject(aHashTable);
16                System.out.println("aHashTable = " + aHashTable.toString());
17              p.flush();
18              p.close();
19          }
20          catch ( IOException e)      {
21              System.out.println(e.getMessage());
22          }
23
24        }
25      }
```

Source Code: Src/9_was/ObjectWriter_2.java

```
% java ObjectWriter_2
aHashTable = {minus Movie=Independence Day, plus  Movie=A little Voice}
```

```
 1      import java.io.*;
 2      import java.util.*;
 3
 4      public class ObjectReader_2 {
 5        public static void main( String args[] ) {
 6
 7          Hashtable aHashTable;
 8
 9          try {
10              FileInputStream istream =
11                             new FileInputStream("object_2.ser");
12              ObjectInputStream p = new ObjectInputStream(istream);
13
14              aHashTable= (Hashtable)p.readObject();
15                System.out.println("aHashTable = " + aHashTable.toString());
16              p.close();
17          }
18          catch ( IOException e)      {
19              System.out.println(e.getMessage());
20          }
21          catch ( ClassNotFoundException e)   {
22              System.out.println(e.getMessage());
23          }
24        }
25      }
```

Source Code: Src/9_was/ObjectReader_2.java

```
% java ObjectReader_2
aHashTable = {minus Movie=Independence Day, plus  Movie=A little Voice}
```

10.18. Can an Object include itself?

```
 1      import java.io.*;
 2      import java.util.*;
 3
 4      public class Self {
 5        public static void main( String args[] ) {
 6
 7          Hashtable aHashTable = new Hashtable();
 8          aHashTable.put("plus  Movie", "A little Voice");
 9          aHashTable.put("The HashTable", aHashTable);
10
11          try {
12              FileOutputStream ostream =
13                           new FileOutputStream("self.ser");
14              ObjectOutputStream p = new ObjectOutputStream(ostream);
15              p.writeObject(aHashTable);
16              p.flush();
17              p.close();
18          }
19          catch ( IOException e)      {
20              System.out.println(e.getMessage());
21          }
22          catch ( Exception e)        {
23              System.out.println(e.getMessage());
24              e.printStackTrace();
25              System.exit(1);
26          }
27
28        }
29      }
```

Source Code: Src/9_was/Self.java

```
% java Self
% od -c self.ser
0000000 254 355  \0 005   s   r  \0 023   j   a   v   a   .   u   t   i
0000020   l   .   H   a   s   h   t   a   b   l   e 023 273 017   %   !
0000040   J 344 270 003  \0 002   F  \0  \n   l   o   a   d   F   a   c
 ...
0000160       M   o   v   i   e   t  \0 016   A       l   i   t   t   l
0000200   e       V   o   i   c   e   x
0000210
```

```
 1      import java.io.*;
 2      import java.util.*;
 3
 4      public class Self_Reader {
 5        public static void main( String args[] ) {
 6
 7          Hashtable aHashTable;
 8
 9          try {
10              FileInputStream istream =
11                             new FileInputStream("self.ser");
12              ObjectInputStream p = new ObjectInputStream(istream);
13
14              aHashTable= (Hashtable)p.readObject();
15              System.out.println("plus  Movie = " + aHashTable.get("plus  Movie"));
16              System.out.println("The HashTable" + aHashTable.get("The HashTable"));
17              System.out.println("aHashTable = " + aHashTable.toString());
18              p.close();
19          }
20          catch ( IOException e)      {
21              System.out.println(e.getMessage());
22          }
23          catch ( ClassNotFoundException e)   {
24              System.out.println(e.getMessage());
25          }
26        }
27      }
```

Source Code: Src/9_was/Self_Reader.java

```
% java Self_Reader 2>&1 | more // version jdk1.3.1
plus  Movie = A little Voice
java/lang/StackOverflowError
        at java/lang/StringBuffer.<init>(StringBuffer.java)
        at java/gutil.Hashtable.toString(Hashtable.java)
        at java/gutil.Hashtable.toString(Hashtable.java)
 ...
% java Self_Reader // version jdk1.4
plus  Movie = A little Voice
The HashTable{The HashTable=(this Map), plus  Movie=A little Voice}
aHashTable = {The HashTable=(this Map), plus  Movie=A little Voice}
```

10.19. Make it Serializable

The first shot is using a ’Serializable’ class:

```
 1      import java.io.*;
 2      import java.util.*;
 3
 4      public class ObjectWriter_3 extends Hashtable {
 5        int local = 42;
 6
 7        public static void main( String args[] ) {
 8
 9          ObjectWriter_3 aObjectWriter_3 = new ObjectWriter_3();
10
11          try {
12              FileOutputStream ostream =
13                           new FileOutputStream("object_3.ser");
14              ObjectOutputStream p = new ObjectOutputStream(ostream);
15              p.writeObject(aObjectWriter_3);
16                System.out.println("aObjectWriter_3 = " + aObjectWriter_3.toString()
17                System.out.println("aObjectWriter_3.local = " + aObjectWriter_3.loca
18              p.flush();
19              p.close();
20          }
21          catch ( IOException e)      {
22              System.out.println(e.getMessage());
23              e.printStackTrace();
24          }
25
26        }
27      }
```

Source Code: Src/9_was/ObjectWriter_3.java

```
% java ObjectWriter_3
aObjectWriter_3 = {}
aObjectWriter_3.local = 42
```

```
 1      import java.io.*;
 2      import java.util.*;
 3
 4      public class ObjectReader_3 {
 5        public static void main( String args[] ) {
 6
 7          ObjectWriter_3 aObjectWriter_3;
 8
 9          try {
10              FileInputStream istream =
11                           new FileInputStream("object_3.ser");
12              ObjectInputStream p = new ObjectInputStream(istream);
13
14              aObjectWriter_3= (ObjectWriter_3)p.readObject();
15                System.out.println("ObjectWriter_3.local = " + aObjectWriter_3.local
16              p.close();
17          }
18          catch ( IOException e)      {
19              System.out.println(e.getMessage());
20              e.printStackTrace();
21          }
22          catch ( ClassNotFoundException e)   {
23              System.out.println(e.getMessage());
24              e.printStackTrace();
25          }
26        }
27      }
```

Source Code: Src/9_was/ObjectReader_3.java

```
% java ObjectReader
aObjectWriter_3.local = 42
```

- An object is serializable only if its class implements the Serializable interface. Thus, if you want to serialize the instances of one of your classes, the class must implement the Serializable interface. The good news is that Serializable is an empty interface. That is, it doesn't contain any method declarations; it's purpose is simply to identify classes whose objects are serializable.
- You don't have to write any methods. The serialization of instances of this class are handled by the defaultWriteObject method of ObjectOutputStream. This method automatically writes out everything required to reconstruct an instance of the class, including the following:
  - — Class of the object
  - — Class signature
  - — Values of all non-transient and non-static members, including members that refer to other objects

```
 1      import java.io.*;
 2      import java.util.*;
 3
 4      public class ObjectWriter_4 implements Serializable {
 5        int local = 42;
 6        private void writeObject(ObjectOutputStream s) throws IOException {
 7                 s.defaultWriteObject();
 8               // customized serialization code
 9        }
10
11        private void readObject(ObjectInputStream s) throws IOException  {
12              try {
13                      s.defaultReadObject();
14              }
15              catch ( ClassNotFoundException e)       {
16                  System.out.println(e.getMessage());
17                  e.printStackTrace();
18              }
19
20               // customized deserialization code
21               //  ...
22               // followed by code to update the object, if necessary
23        }
24
25        public static void main( String args[] ) {
26
27          ObjectWriter_4 aObjectWriter_4 = new ObjectWriter_4();
28
29          try {
30              FileOutputStream ostream =
31                          new FileOutputStream("object_4.ser");
32              ObjectOutputStream p = new ObjectOutputStream(ostream);
33              p.writeObject(aObjectWriter_4);
34                System.out.println("aObjectWriter_4 = " + aObjectWriter_4.toString()
35                System.out.println("aObjectWriter_4.local = " + aObjectWriter_4.loca
36              p.flush();
37              p.close();
38          }
39          catch ( IOException e)      {
40              System.out.println(e.getMessage());
41              e.printStackTrace();
42          }
43
44        }
45      }
```

Source Code: Src/9_was/ObjectWriter_4.java

```
% java ObjectWriter_4
aObjectWriter_4 = ObjectWriter_4@1dc60810
aObjectWriter_4.local = 42
```

The reader program must not be modified.

10.20. Serialization Notes

Interitance Comments

- When a class implements the java.io.Serializable interface, all its sub-classes are serializable as well.
- When an object has a reference to another object, these objects must implement the Serializable interface separately

- Serialization is not secure
- Serialized data can be signed and sealed
- JVM associates a long number with each serializable class, a Serial Version UID
- If a serializable class doesnât declare a serialVersionUID, the JVM will generate one automatically at run-time
- It is highly recommended that each class declares its serialVersionUID as the generated one is compiler dependent
- static final long serialVersionUID = 42L;

10.21. StreamTokenizer

The StreamTokenizer class takes an input stream and parses it into "tokens", allowing the tokens to be read one at a time. The parsing process is controlled by a table and a number of flags that can be set to various states. The stream tokenizer can recognize identifiers, numbers, quoted strings, and various comment styles.

Each byte read from the input stream is regarded as a character in the range ’\u0000’ through ’\u00FF’. The character value is used to look up five possible attributes of the character: white space, alphabetic, numeric, string quote, and comment character. Each character can have zero or more of these attributes.

In addition, an instance has four flags. These flags indicate:

- Whether line terminators are to be returned as tokens or treated as white space that merely separates tokens.
- Whether C-style comments are to be recognized and skipped.
- Whether C++-style comments are to be recognized and skipped.
- Whether the characters of identifiers are converted to lowercase.

A typical application first constructs an instance of this class, sets up the syntax tables, and then repeatedly loops calling the nextToken method in each iteration of the loop until it returns the value TT_EOF.

The first program:

```
 1      import java.io.*;
 2      public class St_1 {
 3        public static void main( String args[] ) {
 4          StreamTokenizer input;
 5          if ( args.length > 1 )      {
 6              System.err.println("Usage: java St [file-name]");
 7              System.exit(1);
 8          }
 9          try {
10              String line;
11              if ( args.length == 1 )
12                  input = new StreamTokenizer( new FileReader(args[0]) );
13              else
14                  input = new StreamTokenizer(
15                              new InputStreamReader(System.in) );
16              while ( input.TT_EOF != input.nextToken() ) {
17                      System.out.println(input.lineno() + ": "
18                              + input.toString());
19              }
20          }
21          catch ( FileNotFoundException e)    {
22              System.out.println(e.getMessage());
23          }
24          catch ( IOException e)      {
25              System.out.println(e.getMessage());
26          }
27          catch ( Exception e)        {
28              System.out.println("Exception occurred: " + e.getMessage() );
29              e.printStackTrace();
30          }
31        }
32      }
```

Source Code: Src/9_was/St_1.java

Result:

```
% head -1 /etc/passwd
root:x:0:1:Super-User:/:/sbin/sh
% head -1 /etc/passwd  | java St_1
1: Token[root], line 1
1: Token[':'], line 1
1: Token[x], line 1
1: Token[':'], line 1
1: Token[n=0.0], line 1
1: Token[':'], line 1
1: Token[n=1.0], line 1
1: Token[':'], line 1
1: Token[Super-User], line 1
1: Token[':'], line 1

% java St_1 /etc/passwd | sed 7q
1: Token[root], line 1
1: Token[':'], line 1
1: Token[x], line 1
1: Token[':'], line 1
1: Token[n=0.0], line 1
1: Token[':'], line 1
1: Token[n=1.0], line 1

% java St_1
hello
1: Token[hello], line 1
a b:c d;e
2: Token[a], line 2
2: Token[b], line 2
2: Token[':'], line 2
2: Token[c], line 2
2: Token[d], line 2
2: Token[';'], line 2
2: Token[e], line 2

% java St_1
aa ///
1: Token[aa], line 1
sss // wwww
2: Token[sss], line 2
222 + #
3: Token[n=222.0], line 3
3: Token['+'], line 3
3: Token['#'], line 3
```

The second program is a beginning of a calculator:

```
 1      import java.io.*;
 2
 3      public class St_2 {
 4
 5        StreamTokenizer input;
 6
 7        public void adjustT() {
 8              input.resetSyntax();
 9              input.commentChar('#');          // comments from #
10                                               // to end-of-line
11              input.wordChars('0', '9');       // parse decimal
12                                               // numbers as words
13              input.wordChars('.', '.');
14              input.wordChars('+', '+');       // operators as words
15              input.wordChars('-', '-');       // operators as words
16              input.wordChars('*', '*');       // operators as words
17              input.wordChars('/', '/');       // operators as words
18              input.whitespaceChars(0, ' ');   // ignore white space
19              input.eolIsSignificant(true);    // need '\n'
20        }
21
22        public void processInput() throws IOException {
23              while ( input.TT_EOF != input.nextToken() ) {
24                      if ( input.ttype != input.TT_EOL )
25                              System.out.println(input.lineno() + ": " +
26                                                      input.sval);
27                              else
28                                      System.out.println("Saw EOL");
29              }
30       }
31
32
33        public static void main( String args[] ) {
34
35          St_2 aSt_2 = new St_2();
36
37          if ( args.length > 1 )      {
38              System.err.println(
39                   "Usage: java St [file-name]");
40              System.exit(1);
41          }
42
43          try {
44              String line;
45              if ( args.length == 1 )
46                  aSt_2.input = new StreamTokenizer(
47                                  new FileReader(args[0]) );
48              else
49                  aSt_2.input = new StreamTokenizer(
50                                   new InputStreamReader(System.in) );
51              aSt_2.adjustT();
```

```
52              aSt_2.processInput();
53
54          }
55          catch ( FileNotFoundException e)    {
56              System.out.println(e.getMessage());
57          }
58          catch ( IOException e)      {
59              System.out.println(e.getMessage());
60          }
61          catch ( Exception e)        {
62              System.out.println("ExceptionType occurred: " +
63                      e.getMessage() );
64              e.printStackTrace();
65          }
66        }
67      }
```

Source Code: Src/9_was/St_2.java

Result:

```
% java St_2
Saw EOL
2 + 3
1: 2
1: +
1: 3
Saw EOL
2 - 3
2: 2
2: -
2: 3
Saw EOL
```

A simple calculator:

The first idea for the process loop:

```
  public void processInput() throws IOException {
        while ( input.TT_EOF != new Expression(input) ) {
                ;
        }
 }
```

What do you think?

- Scanner:

```
 1
 2      import java.io.BufferedReader;
 3      import java.io.FilterReader;
 4      import java.io.IOException;
 5      import java.io.Reader;
 6      import java.io.StreamTokenizer;
 7
 8      /** lexical analyzer for arithmetic expressions.
 9              Comments extend from # to end of line.
10              Words are composed of digits and decimal point(s).
11              White space consists of control characters and space and is ignored;
12              however, end of line is returned.
13              Fixes the lookahead problem for TT_EOL.
14        */
15      public class Scanner extends StreamTokenizer {
16        /** kludge: pushes an anonymous Reader which inserts
17              a space after each newline.
18          */
19        public Scanner (Reader r) {
20          super (new FilterReader(new BufferedReader(r)) {
21            protected boolean addSpace;       // kludge to add space after \n
22            public int read () throws IOException {
23              int ch = addSpace ? ' ' : in.read();
24              addSpace = ch == '\n';
25              return ch;
26            }
27          });
28          resetSyntax();
29          commentChar('#');                   // comments from # to end-of-line
30          wordChars('0', '9');                // parse decimal numbers as words
31          wordChars('.', '.');
32          whitespaceChars(0, ' ');            // ignore control-* and space
33          eolIsSignificant(true);             // need '\n'
34        }
35      }
```

Source Code: Src/9_e/Scanner.java

- Expression.java

```
 1      /*
 2       * Thanks to ats
 3       */
 4      import java.io.InputStreamReader;
 5      import java.io.IOException;
 6      import java.io.ObjectOutputStream;
 7      import java.io.StreamTokenizer;
 8      import java.util.Vector;
 9
10      /** recognizes, stores, and evaluates arithmetic expressions.
11        */
12      public abstract class Expression {
13        final static int eol  = StreamTokenizer.TT_EOL; // switch use ...
14        final static int eof  = StreamTokenizer.TT_EOF; // must be const :(
15        final static int word  = StreamTokenizer.TT_WORD;
16
17        /** reads lines from standard input, parses, and evaluates them
18            or writes them as a Vector to standard output if -c is set.
19            @param args if -c is specified, a Vector is written.
20          */
21        public static void main (String args []) {
22          Scanner scanner = new Scanner(new InputStreamReader(System.in));
23          try {
24            do
25              try {
26                Number n = Expression.line(scanner);
27                System.out.println(n.floatValue());
28              } catch (java.lang.Exception e) {
29                System.err.println(scanner +": "+ e);
30                while (scanner.ttype != scanner.TT_EOL
31                       && scanner.nextToken() != scanner.TT_EOF)
32                  ;
33              } while (scanner.ttype == scanner.TT_EOL);
34          } catch (IOException ioe) { System.err.println(ioe); }
35        }
36        /** indicates parsing errors.
37          */
38        public static class Exception extends java.lang.Exception {
39          public Exception (String msg) {
40            super(msg);
41          }
42        }
43        /** recognizes line: sum '\n';
44            an empty line is silently ignored.
45            @param s source of first input symbol, may be at end of file.
46            @return tree for sum, null if only end of file is found.
47            @throws Exception for syntax error.
48            @throws IOException discovered on s.
49          */
50        public static Number line (Scanner s) throws Exception, IOException {
51          for (;;)
```

```
52            switch (s.nextToken()) {
53            default:
54              Number result = sum(s);
55              if (s.ttype != eol) throw new Exception("expecting nl");
56              return result;
57            case eol: continue;       // ignore empty line
58            case eof: return null;
59            }
60          }
61          /** recognizes product: term [{ ('*'|'%'|'/') term }];
62              @param s source of first input symbol, advanced beyond product.
63              @return tree with evaluators.
64              @see Expression#sum
65            */
66          public static Number product (Scanner s) throws Exception, IOException {
67            Number result = term(s);
68            for (;;)
69              switch (s.ttype) {
70              case '*':
71                s.nextToken();
72                result = new Node.Mul(result, term(s));
73                continue;
74              case '/':
75                s.nextToken();
76                result = new Node.Div(result, term(s));
77                continue;
78              case '%':
79                s.nextToken();
80                result = new Node.Mod(result, term(s));
81                continue;
82              default:
83                return result;
84              }
85          }
86          /** recognizes sum: product [{ ('+'|'-') product }];
87              @param s source of first input symbol, advanced beyond sum.
88              @return tree with evaluators.
89              @see Expression#line
90            */
91          public static Number sum (Scanner s) throws Exception, IOException {
92            Number result = product(s);
93            for (;;)
94              switch (s.ttype) {
95              case '+':
96                s.nextToken();
97                result = new Node.Add(result, product(s));
98                continue;
99              case '-':
00                s.nextToken();
01                result = new Node.Sub(result, product(s));
02                continue;
03              default:
04                return result;
05              }
```

```
06        }
07        /** recognizes term: '('sum')' | Number;
08            @param s source of first input symbol, advanced beyond term.
09            @return tree with evaluators.
10            @see Expression#sum
11          */
12        public static Number term (Scanner s) throws Exception, IOException {
13          switch (s.ttype) {
14          case '(':
15            s.nextToken();
16            Number result = sum(s);
17            if (s.ttype != ')') throw new Exception("expecting )");
18            s.nextToken();
19            return result;
20          case word:
21            result = s.sval.indexOf(".") < 0 ? (Number)new Long(s.sval)
22                                             : (Number)new Double(s.sval);
23            s.nextToken(); return result;
24          }
25          throw new Exception("missing term");
26        }
27      }       // end of class Expression
```

Source Code: Src/9_e/Expression.java

```
 1      /*
 2       * Thanks to ats
 3       */
 4
 5      import java.io.Serializable;
 6
 7      /** base class to store and evaluate arithmetic expressions.
 8          Defines most value-functions so that subclasses need only deal
 9          with long and double arithmetic.
10        */
11      public abstract class Node extends Number implements Serializable {
12
13        /** maps byte arithmetic to long.
14            @return truncated long value.
15          */
16        public byte byteValue () {
17          return (byte)longValue();
18        }
19
20        /** maps short arithmetic to long.
21            @return truncated long value.
22          */
23        public short shortValue () {
24          return (short)longValue();
25        }
26
27        /** maps int arithmetic to long.
28            @return truncated long value.
29          */
```

```
30        public int intValue () {
31          return (int)longValue();
32        }
33
34        /** maps float arithmetic to double.
35            @return truncated double value.
36          */
37        public float floatValue () {
38          return (float)doubleValue();
39        }
40
41        /** represents a binary operator.
42            Must be subclassed to provide evaluation.
43          */
44        protected abstract static class Binary extends Node {
45          /** left operand subtree.
46              @serial left operand subtree.
47            */
48          protected Number left;
49
50          /** right operand subtree.
51              @serial right operand subtree.
52            */
53          protected Number right;
54
55          /** builds a node with two subtrees.
56              @param left left subtree.
57              @param right right subtree.
58            */
59          protected Binary (Number left, Number right) {
60            this.left = left; this.right = right;
61          }
62        }
63
64        /** represents a unary operator.
65            Must be subclassed to provide evaluation.
66          */
67        protected abstract static class Unary extends Node {
68          /** operand subtree.
69              @serial operand subtree.
70            */
71          protected Number tree;
72
73          /** builds a node with a subtree.
74              @param tree subtree.
75            */
76          protected Unary (Number tree) {
77            this.tree = tree;
78          }
79        }
80
81        /** implements addition.
82          */
83        public static class Add extends Binary {
```

```
84          /** builds a node with two subtrees.
85              @param left left subtree.
86              @param right right subtree.
87            */
88          public Add (Number left, Number right) {
89            super(left, right);
90          }
91
92          /** implements long addition.
93              @return sum of subtree values.
94            */
95          public long longValue () {
96            return left.longValue() + right.longValue();
97          }
98
99          /** implements double addition.
00              @return sum of subtree values.
01            */
02          public double doubleValue () {
03            return left.doubleValue() + right.doubleValue();
04          }
05        }
06
07        /** implements subtraction.
08          */
09        public static class Sub extends Binary {
10          /** builds a node with two subtrees.
11              @param left left subtree.
12              @param right right subtree.
13            */
14          public Sub (Number left, Number right) {
15            super(left, right);
16          }
17
18          /** implements long subtraction.
19              @return difference of subtree values.
20            */
21          public long longValue () {
22            return left.longValue() - right.longValue();
23          }
24
25          /** implements double subtraction.
26              @return difference of subtree values.
27            */
28          public double doubleValue () {
29            return left.doubleValue() - right.doubleValue();
30          }
31        }
32
33        /** implements multiplication.
34          */
35        public static class Mul extends Binary {
36          /** builds a node with two subtrees.
37              @param left left subtree.
```

```
38            @param right right subtree.
39          */
40        public Mul (Number left, Number right) {
41          super(left, right);
42        }
43
44        /** implements long multiplication.
45            @return product of subtree values.
46          */
47        public long longValue () {
48          return left.longValue() * right.longValue();
49        }
50
51        /** implements double multiplication.
52            @return product of subtree values.
53          */
54        public double doubleValue () {
55          return left.doubleValue() * right.doubleValue();
56        }
57      }
58
59      /** implements division.
60        */
61      public static class Div extends Binary {
62        /** builds a node with two subtrees.
63            @param left left subtree.
64            @param right right subtree.
65          */
66        public Div (Number left, Number right) {
67          super(left, right);
68        }
69
70        /** implements long division.
71            @return quotient of subtree values.
72          */
73        public long longValue () {
74          return left.longValue() / right.longValue();
75        }
76
77        /** implements double division.
78            @return quotient of subtree values.
79          */
80        public double doubleValue () {
81          return left.doubleValue() / right.doubleValue();
82        }
83      }
84
85      /** implements modulus.
86        */
87      public static class Mod extends Binary {
88        /** builds a node with two subtrees.
89            @param left left subtree.
90            @param right right subtree.
91          */
```

```
92          public Mod (Number left, Number right) {
93            super(left, right);
94          }
95
96          /** implements long modulus.
97              @return remainder after division of subtree values.
98            */
99          public long longValue () {
00            return left.longValue() % right.longValue();
01          }
02
03          /** implements double modulus.
04              @return remainder after division of subtree values.
05            */
06          public double doubleValue () {
07            return left.doubleValue() % right.doubleValue();
08          }
09        }
10
11        /** implements sign change.
12          */
13        public static class Minus extends Unary {
14          /** builds a node with a subtree.
15              @param tree subtree.
16            */
17          public Minus (Number tree) {
18            super(tree);
19          }
20
21          /** implements long sign change.
22              @return negative of subtree value.
23            */
24          public long longValue () {
25            return - tree.longValue();
26          }
27
28          /** implements double sign change.
29              @return negative of subtree values.
30            */
31          public double doubleValue () {
32            return - tree.doubleValue();
33          }
34        }
35      }
```

Source Code: Src/9_e/Node.java

Result:

```
% java Expression
2 * ( 1 - 2 )
-2.0
```

10.22. Not Discussed.

- PipedInputStream
  A piped input stream should be connected to a piped output stream; the piped input stream then provides whatever data bytes are written to the piped output stream. Typically, data is read from a PipedInputStream object by one thread and data is written to the corresponding PipedOutputStream by some other thread. Attempting to use both objects from a single thread is not recommended, as it may deadlock the thread. The piped input stream contains a buffer, decoupling read operations from write operations, within limits
- PushbackInputStream
  A PushbackInputStream adds functionality to another input stream, namely the ability to "push back" or "unread" one byte. This is useful in situations where it is convenient for a fragment of code to read an indefinite number of data bytes that are delimited by a particular byte value; after reading the terminating byte, the code fragment can "unread" it, so that the next read operation on the input stream will reread the byte that was pushed back. For example, bytes representing the characters constituting an identifier might be terminated by a byte representing an operator character; a method whose job is to read just an identifier can read until it sees the operator and then push the operator back to be re-read.
- DigestInputStream
  A transparent stream that updates the associated message digest using the bits going through the stream. To complete the message digest computation, call one of the digest methods on the associated message digest after your calls to one of this digest input stream’s read methods.
- JCE (Java Crypto Package)
- ZipIn/Out-putStream
  This class implements an input stream filter for reading files in the ZIP file format. Includes support for both compressed and uncompressed entries.
- and more .....

10.23. try-with-resources Statement

- try-with-resources statement is a try statement that declares one or more resources
- resource is an object that must be closed after the program is finished with it
- try-with-resources statement ensures that each resource is closed at the end of the statemento

Example: Not using try-with-resources

```
 1      // first line
 2      import java.io.*;
 3      public class TryWithOutResourceAndFinally {
 4
 5
 6      static void readAndPrint(String inF ) throws IOException  {
 7              BufferedReader in = null;
 8
 9              try     {
10                      in = new BufferedReader( new FileReader(inF) );
11                      System.out.println(in.readLine() );
```

```
12              } catch (Exception e) {
13                      System.out.println("Could not open file");
14                      e.printStackTrace();
15              } finally {
16                      if ( in != null )
17                              in.close();
18              }
19        }
20
21        public static void main(String args[]) {
22              if ( args.length != 1 )  {
23                      System.out.println("Usage: java FileIO file");
24              } else  {
25                      System.out.println("Inputfile: " + args[0]);
26                      try {
27                              readAndPrint(args[0]);
28                      } catch (Exception e) {
29                              e.printStackTrace();
30                      }
31              }
32        }
33      }
```

Source Code: Src/7/TryWithOutResourceAndFinally.java

Example: Not using try-with-resources

```
 1      // first line
 2      import java.io.*;
 3      public class TryWithResource {
 4
 5
 6      static void readAndPrint(String inF ) throws IOException  {
 7
 8              try (
 9                      BufferedReader in = new BufferedReader( new FileReader(inF) );
10                  ) {
11                              System.out.println(in.readLine() );
12              } catch (Exception e) {
13                      System.out.println("Could not open file");
14                      e.printStackTrace();
15              }
16        }
17
18        public static void main(String args[]) {
19              if ( args.length != 1 )  {
20                      System.out.println("Usage: java FileIO file");
21              } else  {
22                      System.out.println("Inputfile: " + args[0]);
23                      try {
24                              readAndPrint(args[0]);
25                      } catch (Exception e) {
26                              e.printStackTrace();
```

```
27                      }
28              }
29        }
30      }
```

Source Code: Src/7/TryWithResource.java

```
% java TryWithResource TryWithResource.java
Inputfile: TryWithResource.java
// first line
```

10.24. Examples from the JDK Distribution

The following are examples from the JDK 10 release I used them as are.

10.25. Examples from the JDK Distribution: CustomAutoCloseableSample.java

```
 1      /*
 2       * Copyright (c) 2014, Oracle and/or its affiliates. All rights reserved.
 3       *
 4       * Redistribution and use in source and binary forms, with or without
 5       * modification, are permitted provided that the following conditions
 6       * are met:
 7       *
 8       *   - Redistributions of source code must retain the above copyright
 9       *     notice, this list of conditions and the following disclaimer.
10       *
11       *   - Redistributions in binary form must reproduce the above copyright
12       *     notice, this list of conditions and the following disclaimer in the
13       *     documentation and/or other materials provided with the distribution.
14       *
15       *   - Neither the name of Oracle nor the names of its
16       *     contributors may be used to endorse or promote products derived
17       *     from this software without specific prior written permission.
18       *
19       * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
20       * IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
21       * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
22       * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
23       * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
24       * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
25       * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
26       * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
27       * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
28       * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
29       * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
30       */
31
32      /*
33       * This source code is provided to illustrate the usage of a given feature
34       * or technique and has been deliberately simplified. Additional steps
35       * required for a production-quality application, such as security checks,
36       * input validation, and proper error handling, might not be present in
37       * this sample code.
```

```
 */

import java.io.BufferedOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

/**
 * This sample demonstrates the ability to create custom resource that
 * implements the {@code AutoCloseable} interface. This resource can be used i
 * the try-with-resources construct.
 */
public class CustomAutoCloseableSample {

    /**
     * The main method for the CustomAutoCloseableSample program.
     *
     * @param args is not used.
     */
    public static void main(String[] args) {
        /*
         * TeeStream will be closed automatically after the try block.
         */
        try (TeeStream teeStream = new TeeStream(System.out, Paths.get("out.tx
             PrintStream out = new PrintStream(teeStream)) {
            out.print("Hello, world");
        } catch (Exception e) {
            e.printStackTrace();
            System.exit(1);
        }
    }

    /**
     * Passes the output through to the specified output stream while copying
     * The TeeStream functionality is similar to the Unix tee utility.
     * TeeStream implements AutoCloseable interface. See OutputStream for deta
     */
    public static class TeeStream extends OutputStream {

        private final OutputStream fileStream;
        private final OutputStream outputStream;

        /**
         * Creates a TeeStream.
         *
         * @param outputStream an output stream.
         * @param outputFile   an path to file.
         * @throws IOException If an I/O error occurs.
         */
        public TeeStream(OutputStream outputStream, Path outputFile) throws IO
            this.fileStream = new BufferedOutputStream(Files.newOutputStream(o
```

```
92                  this.outputStream = outputStream;
93              }
94
95              /**
96               * Writes the specified byte to the specified output stream
97               * and copies it to the file.
98               *
99               * @param b the byte to be written.
00               * @throws IOException If an I/O error occurs.
01               */
02              @Override
03              public void write(int b) throws IOException {
04                  fileStream.write(b);
05                  outputStream.write(b);
06              }
07
08              /**
09               * Flushes this output stream and forces any buffered output bytes
10               * to be written out.
11               * The <code>flush</code> method of <code>TeeStream</code> flushes
12               * the specified output stream and the file output stream.
13               *
14               * @throws IOException if an I/O error occurs.
15               */
16              @Override
17              public void flush() throws IOException {
18                  outputStream.flush();
19                  fileStream.flush();
20              }
21
22              /**
23               * Closes underlying streams and resources.
24               * The external output stream won't be closed.
25               * This method is the member of AutoCloseable interface and
26               * it will be invoked automatically after the try-with-resources block
27               *
28               * @throws IOException If an I/O error occurs.
29               */
30              @Override
31              public void close() throws IOException {
32                  try (OutputStream file = fileStream) {
33                      flush();
34                  }
35              }
36          }
37      }
```

Source Code: Src/7_JDK/CustomAutoCloseableSample.java

10.26. Examples from the JDK Distribution: Unzip.java

```
/*
 * Copyright (c) 2014, Oracle and/or its affiliates. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *   - Redistributions of source code must retain the above copyright
 *     notice, this list of conditions and the following disclaimer.
 *
 *   - Redistributions in binary form must reproduce the above copyright
 *     notice, this list of conditions and the following disclaimer in the
 *     documentation and/or other materials provided with the distribution.
 *
 *   - Neither the name of Oracle nor the names of its
 *     contributors may be used to endorse or promote products derived
 *     from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
 * IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/*
 * This source code is provided to illustrate the usage of a given feature
 * or technique and has been deliberately simplified. Additional steps
 * required for a production-quality application, such as security checks,
 * input validation, and proper error handling, might not be present in
 * this sample code.
 */

import java.io.IOException;
import java.io.UncheckedIOException;
import java.nio.file.*;

import static java.nio.file.StandardCopyOption.REPLACE_EXISTING;

/**
 * Extract (unzip) a file to the current directory.
 */
public class Unzip {

    /**
     * The main method for the Unzip program. Run the program with an empty
     * argument list to see possible arguments.
     *
```

```
55           * @param args the argument list for {@code Unzip}.
56           */
57          public static void main(String[] args) {
58              if (args.length != 1) {
59                  System.out.println("Usage: Unzip zipfile");
60              }
61              final Path destDir = Paths.get(".");
62              /*
63               * Create AutoCloseable FileSystem. It will be closed automatically
64               * after the try block.
65               */
66              try (FileSystem zipFileSystem = FileSystems.newFileSystem(Paths.get(ar
67
68                  Path top = zipFileSystem.getPath("/");
69                  Files.walk(top).skip(1).forEach(file -> {
70                      Path target = destDir.resolve(top.relativize(file).toString())
71                      System.out.println("Extracting " + target);
72                      try {
73                          Files.copy(file, target, REPLACE_EXISTING);
74                      } catch (IOException e) {
75                          throw new UncheckedIOException(e);
76                      }
77                  });
78              } catch (UncheckedIOException | IOException e) {
79                  e.printStackTrace();
80                  System.exit(1);
81              }
82          }
83      }
```

Source Code: Src/7_JDK/Unzip.java

10.27. Examples from the JDK Distribution: ZipCat.java

```
1       /*
2        * Copyright (c) 2014, Oracle and/or its affiliates. All rights reserved.
3        *
4        * Redistribution and use in source and binary forms, with or without
5        * modification, are permitted provided that the following conditions
6        * are met:
7        *
8        *   - Redistributions of source code must retain the above copyright
9        *     notice, this list of conditions and the following disclaimer.
10       *
11       *   - Redistributions in binary form must reproduce the above copyright
12       *     notice, this list of conditions and the following disclaimer in the
13       *     documentation and/or other materials provided with the distribution.
14       *
15       *   - Neither the name of Oracle nor the names of its
16       *     contributors may be used to endorse or promote products derived
17       *     from this software without specific prior written permission.
18       *
19       * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
```

```
20       * IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
21       * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
22       * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
23       * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
24       * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
25       * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
26       * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
27       * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
28       * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
29       * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
30       */
31
32      /*
33       * This source code is provided to illustrate the usage of a given feature
34       * or technique and has been deliberately simplified. Additional steps
35       * required for a production-quality application, such as security checks,
36       * input validation, and proper error handling, might not be present in
37       * this sample code.
38       */
39
40      import java.io.IOException;
41      import java.io.InputStream;
42      import java.nio.file.FileSystem;
43      import java.nio.file.FileSystems;
44      import java.nio.file.Files;
45      import java.nio.file.Paths;
46
47      /**
48       * Prints data of the specified file to standard output from a zip archive.
49       */
50      public class ZipCat {
51
52          /**
53           * The main method for the ZipCat program. Run the program with an empty
54           * argument list to see possible arguments.
55           *
56           * @param args the argument list for ZipCat
57           */
58          public static void main(String[] args) {
59              if (args.length != 2) {
60                  System.out.println("Usage: ZipCat zipfile fileToPrint");
61              }
62              /*
63               * Creates AutoCloseable FileSystem and BufferedReader.
64               * They will be closed automatically after the try block.
65               * If reader initialization fails, then zipFileSystem will be closed
66               * automatically.
67               */
68              try (FileSystem zipFileSystem
69                      = FileSystems.newFileSystem(Paths.get(args[0]),null);
70                      InputStream input
71                      = Files.newInputStream(zipFileSystem.getPath(args[1]))) {
72                          byte[] buffer = new byte[1024];
73                          int len;
```

```
74                          while ((len = input.read(buffer)) != -1) {
75                              System.out.write(buffer, 0, len);
76                          }
77
78              } catch (IOException e) {
79                  e.printStackTrace();
80                  System.exit(1);
81              }
82          }
83      }
```

Source Code: Src/7_JDK/ZipCat.java

11. Threads

See also: and

Homework Idea: Correct a selfie

11.1. Intro

Java programs, applications, and applets can consists of threads which conceptually are executed in parallel. This section demonstrates with simple examples how threads are created and manipulated, how exclusive access to common variables is managed, how conditional access is obtained, and how threads are connected with pipelines. Two classical examples concern communication with semaphores and conditional access to resources.

11.2. Principles and Features

- A thread is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently.
- Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority.
- Each thread may or may not also be marked as a daemon. The Java Virtual Machine exits when the only threads running are daemon threads.
- When code running in some thread creates a new Thread object, the new thread has its priority initially set equal to the priority of the creating thread, and is a daemon thread if and only if the creating thread is a daemon, unless specified otherwise.
- As is everything else, during construction the object is controlled by method; afterwards it's execution can be controlled through *start()*, *setPriority()*,
  ... investigated with *getPriority()* and *isAlive()*. JDK 1.1 is supposed to implement interruptions as well

The Thread java doc page:

11.3. Creation and Using

- Threads can be created using an instance of a class which is a subclass of Thread.
- Threads can be created using an instance of a class that implements the interface.

Why are there two different ways?

```
public interface Runnable    {
    public abstract void run();
}
```

Example: run

```
 1
 2      public class Thread_1 extends Thread    {
 3
 4              private int info;
 5              static int x = 0;
 6
 7              public Thread_1 (int info) {
 8                      this.info = info;
 9              }
10
11              public void run () {
12                      if ( info == 1 )        {
13                              x = 3;
14                              try { sleep(100); } catch (Exception e ) {}
15                      } else
16                              x = 1;
17              }
18
19              public static void main (String args []) {
20                      Thread_1 aT1  = new Thread_1(1);
21                      Thread_1 aT2  = new Thread_1(2);
22                      aT1.start();
23                      aT2.start();
24                      System.err.println(x);
25              }
26      }
```

Source Code: Src/11/Thread_1.java

Result:

```
% java Thread_1 a b c d e f g h i j k l m n o p q r s
bdfhjlnpracegikmoqs%
```

Example: runnable

```
 1
 2      public class Thread_1b implements Runnable {
 3
 4              private String info;
 5              int x = 0;
 6
```

```
 7              public Thread_1b (String info) {
 8                      this.info = info;
 9              }
10
11              public void run () {
12                      x=1;
13                      System.out.print(info);
14              }
15
16              public static void main (String args []) {
17                      if (args != null)       {
18                              for (int n = 0; n < args.length; ++ n)  {
19                                      new Thread( new Thread_1b("" + n ) ).start();
20                              }
21                      }
22              }
23      }
```

Source Code: Src/11/Thread_1b.java

Example: termination

```
 1
 2      public class Thread_2 extends Thread    {
 3
 4              private String info;
 5
 6              public Thread_2 (String info) {
 7                      this.info = info;
 8              }
 9
10              public void run () {
11                      long sleep = (int)(Math.random() * 10000);
12                      System.out.println(info + " sleeps for " + sleep );
13                      try {
14                              sleep(sleep);
15                      }
16                      catch (  InterruptedException e ) {
17                              e.getMessage();
18                      }
19              }
20
21              public static void main (String args []) {
22                      int count = 0;
23                      if (args != null)
24                      for (int n = 0; n < args.length; ++ n)  {
25                              Thread_2 aT1  = new Thread_2(args[n]);
26                              if ( n % 2 == 0 )
27                                      aT1.setPriority(Thread.MIN_PRIORITY);
28                              aT1.start();
29                      }
30                      while ( count != 1 )    {
31                              try {
32                                      count = activeCount();
33                                      System.out.println("activeCount() = " +
34                                              count );
35                                      sleep(500);
36                              }
37                              catch (  InterruptedException e ) {
38                                      e.getMessage();
39                              }
40                      }
41
42              }
43      }
```

Source Code: Src/11/Thread_2.java

```
java  Thread_2 a b c d
activeCount() = 5
b sleeps for 1063
d sleeps for 8295
a sleeps for 2197
c sleeps for 2619
activeCount() = 5
activeCount() = 5
activeCount() = 4
activeCount() = 4
activeCount() = 3
activeCount() = 2
 ..
activeCount() = 2
activeCount() = 1
```

Stolen from Java doc (sun)

- Threads belong to groups, represented as ThreadGroup objects and ordered hierarchically. Through a group, several threads can be controlled together; e.g., a group defines a maximum priority for it's members. As a thread can only influence threads in it's own group, the extent of a thread's influence on others can be limited.
- Threads and groups have names, which, however, are mostly useful for documentation and debugging. Theoretically, the classes can be used to identify all threads and all groups; this can be used to build a thread variant of *ps(1)*.
- Internally, the Java runtime system uses several threads that deal, for example, with unreachable objects (garbage collection). If a main program is started, this takes place in the thread main. Once a window is opened, more threads are added.

The execution of the Java Virtual Machine ends once most threads reach the end of their run() methods. ''Most'' means that there are user and daemon threads and thread groups; execution of the JVM ends when there are no more user threads. The distinction is made by calling setDaemon() ; the distinction is necessary, because threads that deliver events and manage painting in a window system are irrelevant for deciding if an application has completed it's job

11.4. Interruption

from:

```
 1      // thanks to
 2      //      mjs
 3      // and
 4      //      tb
 5      public class InterruptExample extends Thread {
 6
 7          private String name;
 8
 9          public InterruptExample(String name) {
10              this.name = name;
11          }
12
13          public static void sleepForAbit(long sleepTime )    {
14              try {
15                  sleep(sleepTime);
16              } catch (InterruptedException e) {
17                  System.err.println("Was interrupted in sleepForAbit");
18              }
19
20          }
21          public void run() {
22
23              System.err.println(name + " has started!");
24              double x = 1;
25              while ( x > 0 ) {               // forever loop
26                      x = x * 2 - x;          // x is constant
27                      if ( isInterrupted() )  {
28                              System.err.println(name + "is interrupted");
29                              break;
30                      }
31              }
32              System.err.println(name + " has exited!");
33          }
34
35          public static void main(String args[]){
36
37              InterruptExample aThread = new InterruptExample("aThread");
38              aThread.start();
39              sleepForAbit(100);      // should allow the thread to enger the while
40              aThread.interrupt();
41
42          }
43      }
```

Source Code: Src/11/InterruptExample.java

State: see java doc

```
% java InterruptExample
aThread has started!
aThreadis interrupted
aThread has exited

if (Thread.interrupted())  // Clears interrupted status!
      throw new InterruptedException();
```

```
 1      public class Thread_3 extends Thread    {
 2              private String info;
 3              Thread_3 aT1;
 4
 5              public Thread_3 (String info) {
 6                      this.info = info;
 7              }
 8              public static void sleepForAbit(long sleepTime )        {
 9                      try {
10                          sleep(sleepTime);
11                      } catch (InterruptedException e) {
12                          System.err.println("Was interrupted in sleepForAbit");
13                      }
14
15              }
16              public void run () {
17                      System.out.println(info + " is running");
18                      try {
19                              sleep(1000000);         // thread has to be here
20                      }
21                      catch (  InterruptedException e ) {
22                              System.err.println("Interrupted!");
23                              if ( isInterrupted() )
24                                      System.err.println("yup it's true.");
25                      }
26                      System.out.println(info + ": exit run");
27
28              }
29              public static void main (String args []) {
30                      Thread_3 aT1  = new Thread_3("first");
31
32                      aT1.start();
33                      sleepForAbit(100);
34                      System.err.println("interrupt 'first'");
35                      aT1.interrupt();
36              }
37      }
```

Source Code: Src/11/Thread_3.java

Result:

```
% java Thread_3
first is running
interrupt ’first’
Interrupted!
first: exit run
```

Extract from Javadoc:

```
The interrupted status of the
current thread is cleared when
this exception is thrown.
```

11.5. Join

```
 1      public class Join extends Thread        {
 2              private String info;
 3              Join aT1;
 4
 5              public Join (String info) {
 6                      this.info = info;
 7              }
 8
 9              public void run () {
10                      System.out.println(info + " is running");
11                      try {
12                              sleep(1000);
13                      }
14                      catch (  InterruptedException e ) {
15                              System.err.println("Interrupted!");
16                      }
17                      System.out.println(info + ": exit run");
18
19              }
20              public static void main (String args []) {
21                      Join aT1  = new Join("first");
22
23                      aT1.start();
24
25                      try {
26                              aT1.join();
27                              System.err.println("Got it");
28                      }
29                      catch (  InterruptedException e ) {
30                              e.printStackTrace();
31                      }
32                      System.err.println("main end");
33              }
34      }
```

Source Code: Src/11/Join.java

Result:

```
first is running
third is running
second is running
second: exit run
third: exit run
first: exit run
Got it
```

Extract from Javadoc:

```
The interrupted status of the
current thread is cleared when
this exception is thrown.
```

11.6. Join II

Does this code produces he correct output?

```
 1      public class Evaluator extends Thread {
 2              int i, j;
 3              final static int MAX = 2;
 4              Evaluator()     {
 5              }
 6              Evaluator(int i, int j) {
 7                      this.i = i;
 8                      this.j = j;
 9              }
10              static int a[][] = new int[MAX][MAX];
11              static int b[][] = new int[MAX][MAX];
12              static int c[][] = new int[MAX][MAX];
13              public void run(){
14                      for ( int index = 0; index < MAX; index ++ )    {
15                              try { sleep (100); } catch (Exception e ){} ;
16                              c[i][j] += a[i][index] * b[index][j];
17                      }
18              }
19              public String print(int a[][], String whichOne){
20                      String rValue = whichOne + ": \n";
21                      for(int i = 0; i < MAX; i++){
22                              for(int j =0; j < MAX; j++){
23                                      rValue += a[i][j] + "  ";
24                              }
25                              rValue = rValue + "\n";
26                      }
27
28                      return rValue;
29              }
30              public void init()      {
31                      for(int i = 0; i < MAX; i++){                   // i-->
32                              for(int j =0; j < MAX; j++){            // j
33                                      a[i][j] = b[i][j] = 2 + i + j ;         // |
34                              }                                       // v
35                      }
36
37              }
38              public String toString(){
39                      String rValue = print(a, "A") + print(b, "B" ) + print(c, "C")
40                      return rValue;
41              }
42              public void multiply(){
43                      Evaluator et[] = new Evaluator[ MAX * MAX];
44                      int counter = 0;
45                      for(int i = 0; i < MAX; i++){
46                              for(int j =0; j < MAX; j++){
47                                      new Evaluator(i, j).start();
48                              }
49                      }
50                      System.out.println(this);
```

```
51              }
52              public static void main(String[] args) {
53                      Evaluator eval = new Evaluator();
54                      eval.init();
55                      eval.multiply();
56              }
57      }
```

Source Code: Src/11/Evaluator.java

Does this code produces he correct output? Any problems?

```
 1      public class Evaluator_2 extends Thread {
 2              int i, j;
 3              final static int MAX = 2;
 4              Evaluator_2()   {
 5              }
 6              Evaluator_2(int i, int j)       {
 7                      this.i = i;
 8                      this.j = j;
 9              }
10              static int a[][] = new int[MAX][MAX];
11              static int b[][] = new int[MAX][MAX];
12              static int c[][] = new int[MAX][MAX];
13              public void run(){
14                      for ( int index = 0; index < MAX; index ++ )    {
15                              c[i][j] += a[i][index] * b[index][j];
16                      }
17              }
18              public String print(int a[][], String whichOne){
19                      String rValue = whichOne + ": \n";
20                      for(int i = 0; i < MAX; i++){
21                              for(int j =0; j < MAX; j++){
22                                      rValue += a[i][j] + "  ";
23                              }
24                              rValue = rValue + "\n";
25                      }
26
27                      return rValue;
28              }
29              public void init()      {
30                      for(int i = 0; i < MAX; i++){                   // i-->
31                              for(int j =0; j < MAX; j++){            // j
32                                      a[i][j] = b[i][j] = 2 + i + j ;         // |
33                              }                                               // v
34                      }
35
36              }
37              public String toString(){
38                      String rValue = print(a, "A") + print(b, "B" ) + print(c, "C")
39                      return rValue;
40              }
41              public void multiply(){
42                      int counter = 0;
```

```
43                      System.out.println(this);
44                      for(int i = 0; i < MAX; i++){
45                              for(int j =0; j < MAX; j++){
46                                      Evaluator_2 et = new Evaluator_2(i, j);
47                                      et.start();
48                                      try {
49                                              et.join();
50                                      } catch ( Exception e ) {
51                                      }
52                              }
53                      }
54                      System.out.println(this);
55              }
56              public static void main(String[] args) {
57                      Evaluator_2 eval = new Evaluator_2();
58                      eval.init();
59                      eval.multiply();
60              }
61      }
```

Source Code: Src/11/Evaluator_2.java

```
 1      public class Evaluator_3 extends Thread {
 2              int i, j;
 3              final static int MAX = 2;
 4              Evaluator_3()   {
 5              }
 6              Evaluator_3(int i, int j)       {
 7                      this.i = i;
 8                      this.j = j;
 9              }
10              static int a[][] = new int[MAX][MAX];
11              static int b[][] = new int[MAX][MAX];
12              static int c[][] = new int[MAX][MAX];
13              public void run(){
14                      for ( int index = 0; index < MAX; index ++ )    {
15                              c[i][j] += a[i][index] * b[index][j];
16                      }
17              }
18              public String print(int a[][], String whichOne){
19                      String rValue = whichOne + ": \n";
20                      for(int i = 0; i < MAX; i++){
21                              for(int j =0; j < MAX; j++){
22                                      rValue += a[i][j] + "  ";
23                              }
24                              rValue = rValue + "\n";
25                      }
26
27                      return rValue;
28              }
29              public void init()      {
30                      for(int i = 0; i < MAX; i++){                   // i-->
31                              for(int j =0; j < MAX; j++){            // j
32                                      a[i][j] = b[i][j] = 2 + i + j ;         // |
```

```
33                              }                                               // v
34                      }
35
36              }
37              public String toString(){
38                      String rValue = print(a, "A") + print(b, "B" ) + print(c, "C")
39                      return rValue;
40              }
41              public void multiply(){
42                      Evaluator_3 et[] = new Evaluator_3[ MAX * MAX];
43                      int counter = 0;
44                      System.out.println(this);
45                      for(int i = 0; i < MAX; i++){
46                              for(int j =0; j < MAX; j++){
47                                      et[counter] = new Evaluator_3(i, j);
48                                      et[counter].start();
49                              }
50                      }
51                      for(int i = 0; i < MAX * MAX; i++){
52                              try{
53                                      et[counter].join();
54                              }
55                              catch(InterruptedException e){
56                                      System.out.println("Interrupted!");
57                              }
58                      }
59                      System.out.println(this);
60              }
61              public static void main(String[] args) {
62                      Evaluator_3 eval = new Evaluator_3();
63                      eval.init();
64                      eval.multiply();
65              }
66      }
```

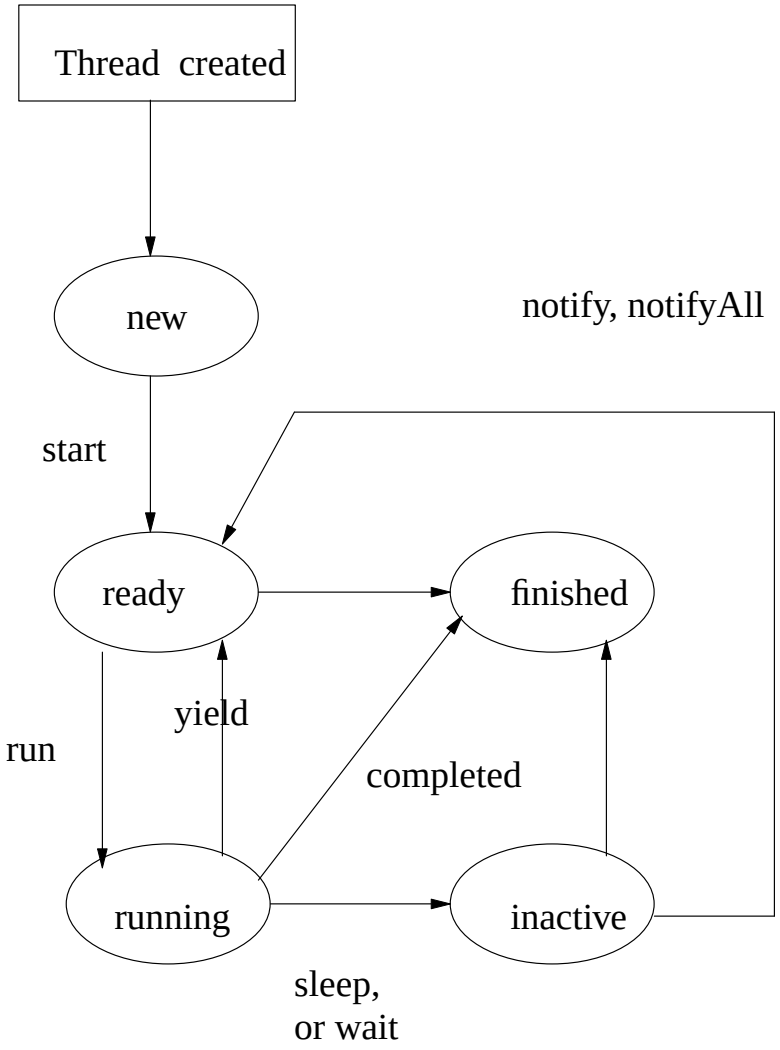
Source Code: Src/11/Evaluator_3.java

11.7. Using the Interface Runnable

```
 1      import java.util.*;
 2
 3      public class Thread_R implements Runnable       {
 4              private String name;
 5              private Vector aVector;
 6
 7              public Thread_R (String name) {
 8                      this.name = name;
 9              }
10
11              public void run () {
12                      System.out.println("Hi :) ... my name is: " + name );
13              }
14
15              public static void main (String args []) {
16                      String names[] = { "bono", "U2" };
17                      for ( int index = 0; index < names.length; index ++ )   {
18                              new Thread( new Thread_R( names[index] ) ).start();
19                      }
20              }
21      }
```

Source Code: Src/11/Thread_R.java

11.8. Thread States


Thread created
new
notify, notifyAll
start
ready
finished
yield
run
completed
running
inactive
sleep,
or wait

11.9. Threads and Cores

```
 1      import java.util.*;
 2
 3      public class CoresTest extends Thread    {
 4          static final int soManyThreads  = Runtime.getRuntime().availableProcessors
 5          static final int soOftenPerThread = 10000000;
 6          static final int multiplier       = 100000000;
 7          static long milliSeconds = 0;
 8          double result = 0;
 9          int id;
10
11          public CoresTest(int index) {
12              id = index;
13          }
14          public static void init()  {
15              milliSeconds = System.currentTimeMillis();
16          }
17          public static void end(String s)   {
18              System.err.println(s + ":       " + ( System.currentTimeMillis() - mil
19              System.err.println(" # of cores" +   ":       " +
20                                                   Runtime.getRuntime().availableProcess
21          }
22          public void singleThreadTest (int soOften) {
23              for (int index = 0; index < soOften; index ++ )         {
24                      for (int k = 0; k < multiplier; k ++ )          {
25                              result = Math.sqrt(index) + Math.sqrt(index) + Math.sq
26                      }
27              }
28          }
29          public void run () {
30              singleThreadTest(soOftenPerThread);
31          }
32          public static void main (String args []) {
33              CoresTest single = new CoresTest(0);
34              CoresTest[] many   = new CoresTest[soManyThreads];
35              CoresTest o = null;
36              init();
37                      single.singleThreadTest(soOftenPerThread * soManyThreads);
38              end("Single Thread Test");
39
40              init();
41                      for ( int index = 0; index < soManyThreads; index ++ )  {
42                              many[index] = new CoresTest(soOftenPerThread);
43                              many[index].start();
44                      }
45                      try {
46                              for ( int index = 0; index < soManyThreads; index ++ )
47                                      many[index].join();
48                              }
49                      } catch (Exception e ) {
50                              e.printStackTrace();}
51              end("Multiple Core Test");
```

```
52          }
53      }
```

Source Code: Src/11/CoresTest.java

```
Single Thread Test:       8632
 # of cores:       16
Multiple Core Test:       745
 # of cores:       16
```

11.10. Limited Number of Threads

- You can only run a limited number of threads at the same time

```
 1      import java.util.*;
 2
 3      public class ThreadMax extends Thread {
 4          int id;
 5          public ThreadMax(int id)    {
 6              this.id = id;
 7          }
 8          public void run () {
 9              try {
10                      System.out.println("ThreadMax Start =   " + id);
11                      Thread.sleep(4000);
12                      System.out.println("ThreadMax End =     " + id);
13              } catch (Exception e)   {
14                      e.printStackTrace();
15                      System.exit(0);
16              }
17          }
18          private void processCommand() {
19              System.out.println("processCommand");
20          }
21
22          public String toString(){
23                  return "ThreadMax: " + id;
24          }
25          public static void main (String args []) {
26              try {
27                      for (int index = 0; index < 100000; index ++ )
28                              new ThreadMax(index).start();
29              } catch (Error e)       {
30                      System.out.println("Error");
31                      e.printStackTrace();
32                      System.exit(0);
33              } catch (Exception e)   {
34                      System.out.println("Exception");
35                      e.printStackTrace();
36                      System.exit(0);
37              }
38          }
39      }
```

Source Code: Src/11/ThreadMax.java

```
% java ThreadMax
 ...
ThreadMax Start =    3033
[1.048s][warning][os,thread] Failed to start thread - pthread_create failed (EAGAIN) f
Error
java.lang.OutOfMemoryError: unable to create native thread: possibly out of memory or
    at java.base/java.lang.Thread.start0(Native Method)
    at java.base/java.lang.Thread.start(Thread.java:813)
    at ThreadMax.main(ThreadMax.java:28)
```

•

11.11. Thread Pools

- — Instead of staring a new thread for every task, the task is passed to the thread pool
- — A thread from the thread pool will take over this taks if a thread is free, or completed the previous task
- — The number of active threads at any given moment in time is bound
- — Question: Recourses of thread pool management vs. thread creation

• A thread pool has typically:

- — worker threads
- —
- —

An example:

```
 1      public class HpWorker_1 implements Runnable {
 2
 3          private int id;
 4          private int sleepTime;
 5
 6          public HpWorker_1(int id, int sleepTime){
 7              this.id         = id;
 8              this.sleepTime  = sleepTime;
 9          }
10          public void run() {
11              System.out.println("Start =     " + id);
12              try {
13                  Thread.sleep(sleepTime);
14                  } catch (InterruptedException e) {
15                          e.printStackTrace();
16              }
17              System.out.println("End =       " + id);
18          }
19          public String toString(){
20                  return "HpWorker_1: " + id;
21          }
22      }
```

Source Code: Src/11/HpWorker_1.java

```
 1      import java.util.concurrent.ExecutorService;
 2      import java.util.concurrent.Executors;
 3
 4      public class HpSimpleThreadPoolUse_1 {
 5          public static final int MAX = 2;
 6          public static final int SLEEP_TIME = 1000; // 100
 7          public static void main(String[] args) {
 8              ExecutorService executor = Executors.newFixedThreadPool(MAX);
 9              for (int i = 0; i < MAX * 3; i++) {
10                  if ( i % 2 == 0 )
11                          executor.execute(new HpWorker_1(i, 4 * SLEEP_TIME));
12                  else
13                          executor.execute(new HpWorker_1(i, SLEEP_TIME));
14                }
15              executor.shutdown();
16              // awaitTermination(long timeout, TimeUnit unit) throws InterruptedExc
17              // Blocks until all tasks have completed execution after a
18              // shutdown request, or the timeout occurs,
19              // or the current thread is interrupted, whichever happens first.
20
21              while ( !executor.isTerminated() ) {
22                      try { Thread.sleep(100); } catch ( Exception e ) { }
23              }
24              System.out.println("all threads have terminated");
25          }
26
27      }
```

Source Code: Src/11/HpSimpleThreadPoolUse_1.java

Example:

```
% java HpSimpleThreadPoolUse_1
Start = 0
Start = 1
End =   1
Start = 2
End =   0
Start = 3
End =   3
Start = 4
End =   2
Start = 5
End =   5
End =   4
all threads have terminated
```

A Very Simplified Thread Pool Implementation

```
 1      import java.util.concurrent.LinkedBlockingQueue;
 2
 3      public class HpThreadPool {
 4          private final int nThreads;
 5          private final Worker[] threads;
 6          private final LinkedBlockingQueue queue;
 7
 8          private boolean shutDownThePool = false;
 9
10          public HpThreadPool(int nThreads) {
11              this.nThreads = nThreads;
12              queue = new LinkedBlockingQueue();
13              threads = new Worker[nThreads];
14
15              for (int i = 0; i < nThreads; i++)
16                  ( threads[i] = new Worker()).start();
17          }
18
19          public void execute(Runnable task) {
20              if ( ! shutDownThePool )    {
21                  synchronized (queue) {
22                      queue.add(task);
23                      queue.notify();
24                  }
25              }
26          }
27          public void shutdown() {
28              synchronized (queue) {
29                  shutDownThePool = true;
30              }
31          }
32          public boolean isTerminated() {
33              boolean rValue = false;
34                  synchronized (queue) {
35                          rValue = shutDownThePool && queue.isEmpty();
36                          if ( rValue )    {
37                              for (int i = 0; i < nThreads; i++) {
38                                      synchronized ( threads[i] )    {
39                                      threads[i].thisIsAnActiveThread = false;
40                                      queue.notify();
41                                  }
42                              }
43                  }
44              }
45              return rValue;
46          }
47
48          private class Worker extends Thread {
49              public boolean thisIsAnActiveThread = true;
50                  public void run() {
51                      Runnable task;
52
53                      while (thisIsAnActiveThread) {
54                          synchronized (queue) {
```

```
55                              while (queue.isEmpty()) {
56                                  try {
57                                      queue.wait();
58                                      if ( !  thisIsAnActiveThread )
59                                          return;
60                                  } catch (InterruptedException e) {
61                                      System.out.println("An error occurred while qu
62                                  }
63                              }
64                              task = (Runnable)queue.poll();
65                          }
66
67                          try {
68                              task.run();
69                          } catch (RuntimeException e) {
70                              System.out.println("HpThreadPool: Something went wrong
71                              e.printStackTrace();
72                          }
73                      }
74                  }
75              }
76      }
```

Source Code: Src/11/HpThreadPool.java

```
 1      import java.util.concurrent.ExecutorService;
 2      import java.util.concurrent.Executors;
 3
 4      public class HpSimpleThreadPoolUse_2 {
 5          public static final int MAX = 1;
 6          public static final int SLEEP_TIME = 1200; // 100
 7          public static void main(String[] args) {
 8              HpThreadPool executor = new HpThreadPool(MAX);
 9              // is it possible to have less than max threads running?
10              for (int i = 0; i < 1 + MAX * 2; i++) {
11                  if ( i % 2 == 0 )
12                          executor.execute(new HpWorker_1(i, 4 * SLEEP_TIME));
13                  else
14                          executor.execute(new HpWorker_1(i, SLEEP_TIME));
15
16              }
17              executor.shutdown();
18
19              while ( !executor.isTerminated() ) {
20                     System.out.println("check if terminated ...");
21                     try { Thread.sleep(5000); } catch ( Exception e ) { }
22              }
23              System.out.println("all threads have terminated");
24
25          }
26      }
```

Source Code: Src/11/HpSimpleThreadPoolUse_2.java

```
% java HpSimpleThreadPoolUse_2
check if terminated ...
Start = 1
Start = 2
Start = 0
End =   1
Start = 3
End =   3
Start = 4
End =   0
End =   2
Start = 5
Start = 6
all threads have terminated # io latency
End =   5
End =   4
End =   6
```

11.12. Competing Threads

Threads can obtain exclusive access to an object if all competing threads use a synchronized statement or call a method with synchronized attribute. Class methods monitor the class description, other methods monitor their receiver, the statement monitors the indicated value. The attribute synchronized precedes the result type.

Adding or deleting a synchronized modifier of a method does not break compatibility with existing binaries.

- the execution of methods can be synchronized.

```
public synchronized method( ...) { ... }
public static synchronized method( ...) { ... }
```

- synchronized statements which allow access to an associated object.

```
synchronized(aObj) { ... }
```

```
public synchronized method( ...) { ... }
```

is aequivalent to

```
public method( ...) {
     synchronized ( this ) {
     }
}
```

Note: Interface methods can’t be native, static, synchronized, final, private, or protected

11.13. Example 0

```
 1      import java.util.*;
 2
 3      public class Thread_0 extends Thread    {
 4          private String info;
 5          static  Object o = new Object();
 6
 7          public Thread_0 (String info) {
 8              this.info    = info;
 9          }
10
11          public void run () {
12              synchronized (o) {
13                      System.err.println(info + ": is in protected()");
14                      System.err.println(info + ": exit run");
15              }
16          }
17
18          public static void main (String args []) {
19              Thread_0 aT4_0 = new Thread_0("first");
20              Thread_0 aT4_1 = new Thread_0("second");
21
22              aT4_0.start();
23              aT4_1.start();
24          }
25      }
```

Source Code: Src/11/Thread_0.java

```
 1      import java.util.*;
 2
 3      public class Thread_00 extends Thread    {
 4          private String info;
 5
 6          public Thread_00 (String info) {
 7              this.info    = info;
 8          }
 9
10          public synchronized void run () {
11              System.err.println(info + ": is in protected()");
12              System.err.println(info + ": exit run");
13          }
14
15          public static void main (String args []) {
16              Thread_00 aT4_00 = new Thread_00("first");
17              Thread_00 aT4_1 = new Thread_00("second");
18
19              aT4_00.start();
20              aT4_1.start();
21          }
22      }
```

Source Code: Src/11/Thread_00.java

11.14. Example I

```
 1      import java.util.*;
 2
 3      public class M extends Thread    {
 4          private String info;
 5          private Vector aVector;
 6
 7          public M (String info) {
 8              this.info    = info;
 9          }
10
11          private synchronized void inProtected () {
12              System.err.println(info + ": is in protected()");
13              try {
14                      sleep(1000);
15              }
16              catch (  InterruptedException e ) {
17                  System.err.println("Interrupted!");
18              }
19              System.err.println(info + ": exit run");
20          }
21
22          public void run () {
23              inProtected();
24          }
25
26          public static void main (String args []) {
27              Vector aVector = new Vector();
28              M aT4_0 = new M("first");
29
30              aT4_0.start();
31              aT4_0.inProtected();
32          }
33      }
```

Source Code: Src/11/M.java

```
first: is in protected()
first: exit run
first: is in protected()
first: exit run
```

11.15. Example II

This example has a problem.

zwei underschiedlich empfaenger fuer die methode

```
 1      import java.util.*;
 2
 3      public class Thread_4 extends Thread    {
 4          private String info;
 5          private Vector aVector;
 6
 7          public Thread_4 (String info, Vector aVector) {
 8              this.info    = info;
 9              this.aVector = aVector;
10          }
11
12          private synchronized void inProtected () {
13              System.err.println(info + ": is in protected()");
14              aVector.addElement(info);
15              try {
16                  if ( info.equals("second") )
17                      sleep(1000);
18                  else
19                      sleep(3000);
20              }
21              catch (  InterruptedException e ) {
22                  System.err.println("Interrupted!");
23              }
24              System.err.println(info + ": exit run");
25          }
26
27          public void run () {
28              inProtected();
29          }
30
31          public static void main (String args []) {
32              Vector aVector = new Vector();
33              Thread_4 aT4_0 = new Thread_4("first",  aVector);
34              Thread_4 aT4_1 = new Thread_4("second", aVector);
35
36              aT4_0.start();
37              aT4_1.start();
38          }
39      }
```

Source Code: Src/11/Thread_4.java

Result:

```
first: is in protected()
second: is in protected()
second: exit run
first: exit run
```

the only possible output?

11.16. Object Synchronization

```
 1      import java.util.*;
 2
 3      public class Thread_5 extends Thread    {
 4              private String info;
 5              static  Vector aVector;
 6
 7              public Thread_5 (String info, Vector aVector) {
 8                      this.info = info;
 9                      this.aVector = aVector;
10              }
11
12              public void inProtected () {
13                 synchronized ( aVector )     {
14                      System.err.println(info + ": is in protected()");
15                      try {
16                              if ( info.equals("second") )
17                                      sleep(1000);
18                              else
19                                      sleep(3000);
20                      }
21                      catch (  InterruptedException e ) {
22                              System.err.println("Interrupted!");
23                      }
24                      System.err.println(info + ": exit run");
25                 }
26              }
27
28              public void run () {
29                      inProtected();
30              }
31
32              public static void main (String args []) {
33                      Vector aVector = new Vector();
34                      Thread_5 aT5_0 = new Thread_5("first", aVector);
35                      Thread_5 aT5_1 = new Thread_5("second", aVector);
36
37                      aT5_0.start();
38                      aT5_1.start();
39              }
40      }
```

Source Code: Src/11/Thread_5.java

Der Vector kann nich mehr veraendert werden, nachdem der Konstuctor genedet hat.

```
% java Thread_5
first: is in protected()
first: exit run
second: is in protected()
second: exit run
```

the only possible output?

11.17. Object Synchronization II

```
 1      import java.util.*;
 2
 3      public class Thread_5b extends Thread    {
 4              private String info;
 5              private Vector aVector = new Vector();
 6
 7              public Thread_5b (String info) {
 8                      this.info = info;
 9              }
10
11              public void inProtected () {
12                 synchronized ( aVector )     {
13                      System.err.println(info + ": is in protected()");
14                      try {
15                              sleep(3000);
16                      }
17                      catch (  InterruptedException e ) {
18                              System.err.println("Interrupted!");
19                      }
20                      System.err.println(info + ": exit run");
21                 }
22              }
23
24              public void run () {
25                      inProtected();
26              }
27
28              public static void main (String args []) {
29                      Thread_5b aT5_0 = new Thread_5b("first");
30                      Thread_5b aT5_1 = new Thread_5b("second");
31
32                      aT5_0.start();
33                      aT5_1.start();
34              }
35      }
```

Source Code: Src/11/Thread_5b.java

11.18. Object Synchronization III

What problem do you see?
How to fix it?

Das zu sychonizierende Object is zweimal vorhangen

```
 1
 2      import java.util.*;
 3
 4      public class Thread_5c extends Thread    {
 5              private String info;
 6              static Vector aVector;
 7
 8              public Thread_5c (Vector aVector, String info) {
 9                      this.aVector = aVector;
10                      this.info    = info;
11              }
12
13              public void inProtected () {
14                 synchronized ( aVector )     {
15                      System.err.println(info + ": is in protected()");
16                      try {
17                               sleep(100);
18                      }
19                      catch (  InterruptedException e ) {
20                              System.err.println("Interrupted!");
21                      }
22                      System.err.println(info + ": exit run");
23                 }
24              }
25
26              public void run () {
27                      inProtected();
28              }
29
30              public static void main (String args []) {
31                      Thread_5c aT5_0 = new Thread_5c(new Vector(), "first");
32                      aT5_0.start();
33
34                      Thread_5c aT5_1 = new Thread_5c(new Vector(), "second");
35                      aT5_1.start();
36              }
37      }
```

Source Code: Src/11/Thread_5c.java

Explain all possible outputs.

```
 1
 2      import java.util.*;
 3
 4      public class Thread_5d extends Thread    {
 5              private String info;
 6              static Vector aVector;
```

```
 7
 8              public Thread_5d (Vector aVector, String info) {
 9                      this.aVector = aVector;
10                      this.info    = info;
11              }
12
13              public void inProtected () {
14                 synchronized ( aVector )     {
15                      System.err.println(info + ": is in protected()");
16                      try {
17                               sleep(100);
18                      }
19                      catch (  InterruptedException e ) {
20                              System.err.println("Interrupted!");
21                      }
22                      System.err.println(info + ": exit run");
23                 }
24              }
25
26              public void run () {
27                      inProtected();
28              }
29
30              public static void main (String args []) {
31                      Vector aVector = new Vector();
32                      Thread_5d aT5_0 = new Thread_5d(aVector, "first");
33                      aT5_0.start();
34
35                      try { sleep(1000); } catch (  InterruptedException e ) { Syste
36
37                      aVector = new Vector();
38                      Thread_5d aT5_1 = new Thread_5d(aVector, "second");
39                      aT5_1.start();
40              }
41      }
```

Source Code: Src/11/Thread_5d.java

Explain all possible outputs.

Der Vektor kann ein oder zwei mal vorhanden sein

11.19. Class Synchronization

```
 1      import java.util.*;
 2
 3      public class ClassT extends Thread    {
 4          private String info;
 5          private Vector aVector;
 6
 7          public ClassT (String info, Vector aVector) {
 8              this.info    = info;
 9              this.aVector = aVector;
10          }
11
```

```
12          static synchronized void staticInProtected1(String s) {
13              System.err.println(s + ": ---->");
14              try {
15                      sleep(1000);
16              }
17              catch (  InterruptedException e ) {
18                  System.err.println("Interrupted!");
19              }
20              staticInProtected2(s);
21              System.err.println(s + ": <----");
22          }
23
24          static synchronized void staticInProtected2(String s) {
25              System.err.println(s + ": ====>");
26              try {
27                      sleep(1000);
28              }
29              catch (  InterruptedException e ) {
30                  System.err.println("Interrupted!");
31              }
32              System.err.println(s + ": ====>");
33          }
34
35          public void run () {
36              staticInProtected1(info);
37          }
38
39          public static void main (String args []) {
40              Vector aVector = new Vector();
41              ClassT aClassT_0 = new ClassT("first",  aVector);
42              ClassT aClassT_1 = new ClassT("second", aVector);
43
44              ClassT.staticInProtected1("main");
45              aClassT_0.start();
46              aClassT_1.start();
47              aClassT_0.staticInProtected1("aClassT_0");
48              aClassT_1.staticInProtected1("aClassT_1");
49          }
50      }
```

Source Code: Src/11/ClassT.java

Result:

```
% java ClassT
main: ---->
main: ====>
main: ====>
main: <----
aClassT_0: ---->
aClassT_0: ====>
aClassT_0: ====>
aClassT_0: <----
aClassT_1: ---->
aClassT_1: ====>
aClassT_1: ====>
aClassT_1: <----
first: ---->
first: ====>
first: ====>
first: <----
second: ---->
second: ====>
second: ====>
second: <----
```

One more:

```
 1      import java.util.*;
 2
 3      public class Thread_6 extends Thread    {
 4          private String info;
 5          private Vector aVector;
 6
 7          public Thread_6 (String info, Vector aVector) {
 8              this.info    = info;
 9              this.aVector = aVector;
10          }
11
12          private void inProtected_1 () {
13              synchronized ( aVector )        {
14                      System.err.println("1: " + info + ": is in ");
15                      try {
16                              sleep(1000);
17                      }
18                      catch (  InterruptedException e ) {
19                          System.err.println("Interrupted!");
20                      }
21                      System.err.println("1: " + info + ": exit");
22              }
23          }
24
25          private void inProtected_2 () {
26              synchronized ( info )   {
27                      System.err.println("2: " + info + ": is IN ");
28                      try {
29                              sleep(5000);
30                      }
31                      catch (  InterruptedException e ) {
32                          System.err.println("Interrupted!");
33                      }
34                      System.err.println("2: " + info + ": EXIT");
35              }
36          }
37
38          private static void inProtected_3 () {
39              System.err.println("3: IN ");
40              try {
41                      sleep(9000);
42              }
43              catch (  InterruptedException e ) {
44                  System.err.println("Interrupted!");
45              }
46              System.err.println("3: EXIT");
47          }
48
49          public void run () {
50              inProtected_1();
51              inProtected_2();
```

```
52              Thread_6.inProtected_3();
53          }
54
55          public static void main (String args []) {
56              Vector aVector = new Vector();
57              Thread_6 aT6_0 = new Thread_6("first",  aVector);
58              Thread_6 aT6_1 = new Thread_6("second", aVector);
59
60              aT6_0.start();
61              aT6_1.start();
62          }
63      }
```

Source Code: Src/11/Thread_6.java

11.20. Wait and Notify

By using *wait* and *notify* a thread can give up its lock at an abritary point and the wait for another thread to give it back for continuation.

```
 1      import java.util.Vector;
 2
 3      public class WaitAndNotify_0 extends Thread     {
 4
 5              private static int counter = 0;
 6              private String  name = null;
 7              private Vector aVector;
 8
 9              public WaitAndNotify_0 (String name, Vector aVector) {
10                      this.aVector = aVector;
11                      this.name = name;
12
13              }
14
15              public void run () {
16                 synchronized ( aVector )     {
17                      if (   name.equals("two")  )    {
18                              System.out.println(getName() + " will wait ...");
19                              aVector.notify();
20                              System.out.println(getName() + " done.");
21                      } else {
22                              try {
23                                      aVector.wait();
24                              } catch ( IllegalMonitorStateException  e )     {
25                                      System.out.println( ": IllegalMonitorStateExce
26                              } catch ( InterruptedException  e )     {
27                                      System.out.println(": InterruptedException");
28                              }
29                              System.out.println(getName() + " is awake!");
30                      }
31                }
32              }
33
34
35              public static void main (String args []) {
36                      Vector theVector = new Vector();
37                      new WaitAndNotify_0("one", theVector).start();
38                      new WaitAndNotify_0("two", theVector).start();
39              }
40      }
```

Source Code: Src/11/WaitAndNotify_0.java

Reihenfolge is nicht garatiert.

11.21. One after the Other

```
 1      import java.util.Vector;
 2
 3      public class WaitAndNotify_First extends Thread {
 4
 5              private static int counter = 0;
 6              private String  name = null;
 7              private Vector aVector;
 8
 9              public WaitAndNotify_First (String name, Vector aVector) {
10                      this.aVector = aVector;
11                      this.name = name;
12
13              }
14
15              public void run () {
16                 synchronized ( aVector )     {
17                      if (   name.equals("two")  )    {
18                              System.out.println(getName() + " will wait ...");
19                              aVector.notify();
20                              System.out.println(getName() + " done.");
21                      } else {
22                              System.out.println(getName() + " will wait ...");
23                              try {
24                                      new WaitAndNotify_First("two", aVector).start(
25                                      aVector.wait();
26                              } catch ( IllegalMonitorStateException  e )     {
27                                      System.out.println( ": IllegalMonitorStateExce
28                              } catch ( InterruptedException  e )     {
29                                      System.out.println(": InterruptedException");
30                              }
31                              System.out.println(getName() + " is awake!");
32                      }
33                }
34              }
35
36
37              public static void main (String args []) {
38                      Vector theVector = new Vector();
39                      new WaitAndNotify_First("one", theVector).start();
40              }
41      }
```

Source Code: Src/11/WaitAndNotify_First.java

Erzeugen verzeogert

11.22. Wait and Notify II

What is wrong here: Ordnung - der letzte muss nicht der letzte sein.

```
 1      import java.util.Vector;
 2
 3      public class WaitAndNotify extends Thread       {
 4
 5              private String info;
```

```
 6              static  Vector aVector = new Vector();
 7
 8              public WaitAndNotify (String info, Vector aVector) {
 9                      this.info = info;
10                      this.aVector = aVector;
11              }
12
13              public void doTheJob() {
14                 synchronized ( aVector )     {
15                      if (  info.equals("last") )     {
16                              System.out.println(info + " is waking up ...");
17                              aVector.notifyAll();
18                              System.out.println(info + " done.");
19                      } else {
20                              System.out.println(info + " is waiting");
21                              try {
22                                      aVector.wait();
23                              } catch ( IllegalMonitorStateException  e )     {
24                                      System.out.println(info +
25                                        ": IllegalMonitorStateException");
26                              } catch ( InterruptedException  e )     {
27                                      System.out.println(info +
28                                        ": InterruptedException");
29                              }
30                              System.out.println(info + " is awake!");
31                      }
32                }
33              }
34
35
36              public void run () {
37                      doTheJob();
38              }
39
40              public static void main (String args []) {
41                      new WaitAndNotify("first", aVector).start();
42                      new WaitAndNotify("second", aVector).start();
43                      new WaitAndNotify("last", aVector).start();
44              }
45      }
```

Source Code: Src/11/WaitAndNotify.java

Result:

```
% java WaitAndNotify
first is waiting
second is waiting
last is waking up ...
last done.
first is awake!
second is awake!
```

11.23. Wait and Notify III

```
 1      import java.util.Vector;
 2
 3      public class WaitAndNotify_2 extends Thread     {
 4
 5              private String info;
 6              static Integer monitor = new Integer(3);
 7              static int count = 0;
 8              static int max = 0;
 9
10              public WaitAndNotify_2 (String info) {
11                      this.info = info;
12                      // max ++;
13              }
14
15              public void doTheJob() {
16                 synchronized ( monitor )     {
17                              System.out.println(info + " is waiting");
18                              count ++;
19                              // if ( count == max );
20                              if ( count == 3 )
21                                      monitor.notifyAll();
22                              else
23                                      try {
24                                              monitor.wait();
25                                              sleep(1000);
26                                      } catch ( Exception  e )        {
27                                              System.out.println(info +
28                                                ": IllegalMonitorStateException");
29                                      }
30                              System.out.println(info + " is awake!");
31                      }
32              }
33
34
35              public void run () {
36                      doTheJob();
37              }
38
39              public static void main (String args []) {
40                      new WaitAndNotify_2("first").start();
41                      new WaitAndNotify_2("second").start();
42                      new WaitAndNotify_2("last").start();
43              }
44      }
```

Source Code: Src/11/WaitAndNotify_2.java

Result:

```
% java WaitAndNotify_2
first is waiting
second is waiting
last is waiting
last is awake!
first is awake!
second is awake!
```

11.24. Be carefull with wait(long timeout)

```
 1      import java.util.Vector;
 2      import java.util.Date;
 3
 4
 5      public class WaitAndNotify_3 extends Thread     {
 6
 7              private String info;
 8              static  Vector aVector = new Vector();
 9
10              public WaitAndNotify_3 (String info, Vector aVector) {
11                      this.info = info;
12                      this.aVector = aVector;
13              }
14
15              public void doTheJob() {
16                 synchronized ( aVector )     {
17                      System.out.println(info + " is waiting. " + new Date() );
18                      try {
19                              aVector.wait(1000);
20                      } catch ( Exception  e )        {
21                              System.out.println(info + ": Exception");
22                              e.printStackTrace();
23                      }
24                      System.out.println(info + " is awake! " + new Date());
25                }
26              }
27
28
29              public void run () {
30                      doTheJob();
31              }
32
33              public static void main (String args []) {
34                      new WaitAndNotify_3("first", aVector).start();
35                      new WaitAndNotify_3("second", aVector).start();
36                      // new WaitAndNotify_3("last", aVector).start();
37              }
38      }
```

Source Code: Src/11/WaitAndNotify_3.java

```
% java WaitAndNotify_3
first is waiting. Mon Apr 16 15:02:10 EDT 2001
second is waiting. Mon Apr 16 15:02:11 EDT 2001
first is awake! Mon Apr 16 15:02:12 EDT 2001
second is awake! Mon Apr 16 15:02:12 EDT 2001
%
```

11.25. Lock Objects

See also:

TBD

Stole from java doc:

Interfaces and classes providing a framework for locking and waiting for conditions that is distinct from built-in synchronization and monitors. The framework permits much greater flexibility in the use of locks and conditions, at the expense of more awkward syntax. The Lock interface supports locking disciplines that differ in semantics (reentrant, fair, etc), and that can be used in non-block-structured contexts including hand-over-hand and lock reordering algorithms. The main implementation is ReentrantLock.

The ReadWriteLock interface similarly defines locks that may be shared among readers but are exclusive to writers. Only a single implementation, ReentrantReadWriteLock, is provided, since it covers most standard usage contexts. But programmers may create their own implementations to cover nonstandard requirements.

The Condition interface describes condition variables that may be associated with Locks. These are similar in usage to the implicit monitors accessed using Object.wait, but offer extended capabilities. In particular, multiple Condition objects may be associated with a single Lock. To avoid compatibility issues, the names of Condition methods are different from the corresponding Object versions. Executors

Copied from:

- An object that executes submitted Runnable tasks.
- This interface provides a way of decoupling task submission from the mechanics of how each task will be run, including details of thread use, scheduling, etc.
- An Executor is normally used instead of explicitly creating threads. For example, rather than invoking new Thread(new(RunnableTask())).start() for each of a set of tasks, you might use:

Stolen from java doc:

In all of the previous examples, there's a close connection between the task being done by a new thread, as defined by its Runnable object, and the thread itself, as defined by a Thread object. This works well for small applications, but in large-scale applications, it makes sense to separate thread management and creation from the rest of the application. Objects that encapsulate these functions are known as executors. The following subsections describe executors in detail.

11.26. Executors: Thread Pools

- Using thread pools avoids the overhead of creating threads
- This reduces also memory management overheads
- Help to make usage of multiple processors
- Designed for problems which have recursive solutuions

11.27. Examples

```
1      /*
2       * is this output       1 0 1 0 1 ...
3       *                      ...
4       * the only possible output?
5       *
6       * Falsch: es ist nicht garantiert, in welcher die
7       * Threads eintreten.
8       */
```

```
 9      public class T extends Thread   {
10              private String info;
11              Object o = new Object();
12              public T (String info) {
13                      this.info    = info;
14              }
15              public void run () {
16                      synchronized ( o ) {
17                              while ( true )  {
18                                      System.out.println(info);
19                                      try {
20                                              o.notify();
21                                              sleep(100);
22                                              o.wait();
23                                      } catch ( Exception e ) { }
24                              }
25                      }
26              }
27              public static void main (String args []) {
28                      ( new T("0") ).start();
29                      ( new T("1") ).start();
30              }
31      }
```

Source Code: Src/Question_Week_5/T.java

```
 1      /*
 2       * is this output       --->
 3       *                      <--
 4       *                      ...
 5       * the only possible output?
 6       */
 7      public class T_1 extends Thread    {
 8
 9          private synchronized void inProtected () {
10             System.err.println("--> ");
11             try {
12                    sleep(1000);
13             }
14             catch (  InterruptedException e ) {
15                      System.err.println("Interrupted!");
16             }
17             System.err.println("<-- ");
18          }
19
20          public void run () {
21              inProtected();
22          }
23          public static void main (String args []) {
24              new T_1().start();
25              new T_1().start();
26          }
27      }
28
```

Source Code: Src/Question_Week_5/T_1.java

```
1      /*
2       * is this output       --->
3       *                      <--
4       *                      ...
5       * the only possible output?
6       * nein unterschiedliche zwei objekte
7       */
8      public class T_2 extends Thread    {
9          static String info;
10
11         public T_2(String info )    {
12             this.info = info;
13         }
14         private void inProtected () {
15            synchronized ( info )       {
16                  System.err.println("--> " + info);
17                  try {
18                           sleep(1000);
19                  } catch (  Exception e ) {
20                      e.printStackTrace();
21                  }
22                  System.err.println("<-- " + info);
23            }
24         }
25
26         public void run () {
27             inProtected();
28         }
29         public static void main (String args []) {
30             String aString = "a";
31             T_2 one = new T_2(aString);
32             one.start();
33             T_2 two = new T_2(aString);
34             two.start();
35             aString = "b";
36             // new T_2("a").start();
37             // new T_2("b").start();
38
39         }
40     }
```

Source Code: Src/Question_Week_5/T_2.java

```
1      /*
2       * is this output       --->
3       *                      <--
4       *                      ...
5       * the only possible output?
6       * ja ein objekt
7       */
```

```
 8      public class T_2b extends Thread    {
 9          private String info;
10
11          public T_2b(String info )   {
12              this.info = info;
13          }
14          private synchronized void inProtected () {
15                  System.err.println("--> " + info);
16                  try {
17                          sleep(1000);
18                  } catch (  Exception e ) {
19                      e.printStackTrace();
20                  }
21                  System.err.println("<-- " + info);
22          }
23
24          public void run () {
25              inProtected();
26          }
27          public static void main (String args []) {
28              new T_2b("hello").start();
29              new T_2b("hello").start();
30
31          }
32      }
```

Source Code: Src/Question_Week_5/T_2b.java

```
 1      /*
 2       * is this output       --->
 3       *                      <--
 4       *                      ...
 5       * the only possible output?
 6       * ja ein objekt
 7       */
 8      public class T_2c extends Thread    {
 9          private String info;
10
11          public T_2c(String info )   {
12              new T_2c("hello").start();
13              this.info = info;
14          }
15          private void inProtected () {
16             synchronized ( info )       {
17                  System.err.println("--> " + info);
18                  try {
19                          sleep(1000);
20                  } catch (  Exception e ) {
21                      e.printStackTrace();
22                  }
23                  System.err.println("<-- " + info);
24             }
25          }
26
```

```
27          public void run () {
28              inProtected();
29          }
30          public static void main (String args []) {
31              new T_2c("hello").start();
32
33          }
34      }
```

Source Code: Src/Question_Week_5/T_2c.java

```
 1      /*
 2       * is this output       --->
 3       *                      <--
 4       *                      ...
 5       * the only possible output?
 6       * wievele objekte werden benutzt?
 7       */
 8      public class T_3 extends Thread    {
 9          private int info;
10
11          public T_3 (int info) {
12              this.info    = info;
13          }
14
15          public synchronized void run () {
16              System.err.println("--> " + info);
17               try {
18                        sleep(1000);
19               } catch (  Exception e ) {
20                      e.printStackTrace();
21               }
22               System.err.println("<-- " + info);
23          }
24
25          public static void main (String args []) {
26              for ( int i = 1; i < 100; i ++ )
27                      new T_3(i).start();
28          }
29      }
```

Source Code: Src/Question_Week_5/T_3.java

```
 1      /*
 2       * is this output       --->
 3       *                      <--
 4       *                      ...
 5       * the only possible output?
 6       * nur ein objekt wird benutzt
 7       */
 8      public class T_4 extends Thread    {
 9
10          static Object o = new Object();
```

```
11          String info;
12
13          public T_4(String info )    {
14              this.info = info;
15          }
16
17          public void run () {
18              synchronized ( o ) {
19                  System.err.println("--->" + info);
20                  try {
21                          sleep(1000);
22                  }
23                  catch (  InterruptedException e ) {
24                      System.err.println("Interrupted!");
25                  }
26                  System.err.println("<---" + info);
27              }
28          }
29
30          public static void main (String args []) {
31              new T_4("1").start();
32              new T_4("2").start();
33              new T_4("3").start();
34          }
35      }
```

Source Code: Src/Question_Week_5/T_4.java

```
 1      /*
 2       * is    3             3
 3       *       4             4
 4       *       3             4
 5       *       4             3
 6       *
 7       */
 8      public class T_5 extends Thread    {
 9          static String i = "2";
10          static String  theValue ;
11          T_5(String i)       {
12              this.i = i;
13          }
14          public void run () {
15              if ( this.i.equals("1") )
16                      theValue = "3";
17              else
18                      theValue = "4";
19          }
20
21          public static void main (String args []) {
22              T_5 aT_5_a = new T_5("1");
23              T_5 aT_5_b = new T_5("2");
24
25              aT_5_a.start();
26              aT_5_a.run();           // <<-- ! start
```

```
27
28              System.out.println("aT_5.i = " + aT_5_b.theValue );
29              System.out.println("aT_5.i = " + aT_5_a.theValue );
30          }
31      }
```

Source Code: Src/Question_Week_5/T_5.java

```
 1      /*
 2       * is   e dd
 3       *      e ee
 4       *      a b
 5       * possible?
 6       */
 7      public class T_7 extends Thread    {
 8          String i = "2";
 9          static String  aValue = "a";
10          static String  bValue = "b";
11          T_7(String i)       {
12              this.i = i;
13          }
14          public void run () {
15              synchronized ( i )      {
16                      if  ( i == "1" )        {
17                              aValue = "e";
18                              bValue = "ee";
19                      } else
20                              aValue = "d";
21                              bValue = "dd";
22                      }
23              }
24          }
25
26          public static void main (String args []) {
27              new T_7("1").start();
28              new T_7("2").start();
29              synchronized ( theValue )       {
30                      System.out.println("aValue = " + aValue );
31                      System.out.println("bValue = " + bValue );
32              }
33          }
34      }
```

Source Code: Src/Question_Week_5/T_7.java

```
 1      import java.util.Vector;
 2
 3      /* is 0 1 0 1 ... the only possible output? */
 4
 5      public class T_8 extends Thread {
 6
 7              private String info;
 8              private Vector aVector;
```

```
 9
10              public T_8(String info, Vector aVector) {
11                      this.info = info;
12                      this.aVector = aVector;
13              }
14
15              public void run() {
16                      inProtected();
17              }
18
19              public void inProtected() {
20                      int x = 0;
21                      // currently considering only 10 output pairs, but will work f
22                      // infinity while(true)
23                      while (x < 4) {
24                              synchronized (aVector) {
25
26                                      if (info.equals("zero")) {
27                                              System.out.println("0");
28                                              aVector.notify();
29                                              try {
30                                                      aVector.wait();
31                                              } catch (InterruptedException e) {
32                                                      System.out.println(": Interrup
33                                              }
34                                      } else {
35                                              System.out.println("1");
36                                              try {
37                                                      aVector.notify();
38                                                      aVector.wait();
39                                              } catch (InterruptedException e) {
40                                                      System.out.println(": Interrup
41                                              }
42
43                                      }
44                              }
45                              x++;
46                      }
47              }
48
49              public static void main(String args[]) {
50                      @SuppressWarnings("rawtypes")
51                      Vector aVector = new Vector();
52                      T_8 t0 = new T_8("zero", aVector);
53                      T_8 t1 = new T_8("one", aVector);
54
55                      t0.start();
56                      t1.start();
57              }
58      }
```

Source Code: Src/Question_Week_5/T_8.java

```
 1      /*
 2       * is this output       --->
 3       *                      <--
 4       *                      ...
 5       * the only possible output?
 6       * nein, zwei objekte werden benutzt
 7       */
 8      public class T_9 extends Thread    {
 9
10          static Object o = new Object();
11          String info;
12
13          public T_9(String info )    {
14              this.info = info;
15          }
16
17          public void run () {
18              synchronized ( o ) {
19                  System.err.println("--->" + info);
20                  try {
21                          sleep(1000);
22                  }
23                  catch (  InterruptedException e ) {
24                      System.err.println("Interrupted!");
25                  }
26                  System.err.println("<---" + info);
27              }
28          }
29
30          public static void main (String args []) {
31              new T_9("1", new Object() ).start();
32              new T_9("2", new Object() ).start();
33          }
34      }
```

Source Code: Src/Question_Week_5/T_9.java

```
 1      /*
 2       * is this output       1 0 1 0 1 ...
 3       *                      ...
 4       * the only possible output?
 5       *
 6       * Falsch: es ist nichtgarantiert, in welcher die
 7       * Threads eintreten.
 8       */
 9      public class X extends Thread   {
10              private String info;
11              static Object o = new Object();
12              public X (String info) {
13                      this.info    = info;
14              }
15              public void run () {
16                      while ( true )  {
17                              synchronized ( o ) {
```

```
18                                      System.out.println(info);
19                                      try {
20                                              o.notify();
21                                              sleep(100);
22                                              o.wait(1);
23                                      } catch ( Exception e ) { }
24                              }
25                      }
26              }
27              public static void main (String args []) {
28                      ( new X("0") ).start();
29                      ( new X("1") ).start();
30              }
31      }
```

Source Code: Src/Question_Week_5/X.java

```
 1      /*
 2       * Should print out 0 1 0 1 0 1 ...
 3       * Is this correct?
 4       *
 5       * nicht richtig,
 6       * weil der Konstruktor fuer das Objekt mit der Id 0
 7       * nicht zuende gehen muss bevor der 2. Konstruktor
 8       * zuende geht.
 9       *
10       */
11      public class XX extends Thread  {
12              private String info;
13              static Object o = new Object();
14
15              public XX (String info) {
16                      this.info    = info;
17                      synchronized ( o ) {
18                              if ( info.equals("0") )
19                                      ( new XX("1") ).start();
20                      }
21              }
22              public void run () {
23                      while ( true )  {
24                              synchronized ( o ) {
25                                      System.out.println(info);
26                                      try {
27                                              o.notify();
28                                              sleep(100);
29                                              o.wait();
30                                      } catch ( Exception e ) { }
31                              }
32                      }
33              }
34              public static void main (String args []) {
35                      new XX("0").start();
36              }
37      }
```

Source Code: Src/Question_Week_5/XX.java

```
1      /*
2       * Should print out 0 1 0 1 0 1 ...
3       *
4       *
5       */
6      public class XXX extends Thread {
7              private String info;
8              static Object o = new Object();
9              static boolean oneIsRunning = false; // is static important?
10                                                  // es wird nur ein
11                                                  // Objekt erzeugt
12             public XXX (String info) {
13                     this.info    = info;
14             }
15             public void run () {
16                     while ( true )  {
17                             synchronized ( o ) {
18                                     o.notify();
19                                     System.out.println(info);
20                                     try {
21                                             if ( ! oneIsRunning )   {
22                                                ( new XXX("1") ).start();
23                                                oneIsRunning = true;
24                                             }
25                                             sleep(300);
26                                             o.wait();
27                                     } catch ( Exception e ) { }
28                             }
29                     }
30             }
31             public static void main (String args []) {
32                     new XXX("0").start();
33             }
34     }
```

Source Code: Src/Question_Week_5/XXX.java

11.28. Deadlock — an Overview

- Problem:

  Resource 1 and and resource 2 must be used exclusively

  Process 1 holds resource 1 and is requestion resource 2

  Process 2 holds resource 2 and is requestion resource 1

11.29. Deadlock

- A set of processes is in a deadlock state when every process in the set is waiting for an event that can be caused by only another process in the set

11.30. Necessary Conditions

- A deadlock can occur if the following four conditions hold
  - mutual exclusion: least one resource must be held in a non-sharable mode
  - hold and wait: there is a process that is holding a resource and is waiting to acquire another that is currently being held by other processes
  - no preemption: resources can only be released voluntarily
  - circular wait: See intro example

11.31. Resource Graphs

11.32. Addressing Deadlock

- Prevention: Design the system so that deadlock is impossible
- Avoidance: Construct a model of system states, then choose a strategy that will not allow the system to go to a deadlock state
- Detection & Recovery: Check for deadlock (periodically or sporadically), then recover

11.33. Prevention

- Necessary conditions for deadlock
  - Mutual exclusion
  - Hold and wait
  - Circular waiting
  - No preemption
- Ensure that at least one of the necessary conditions is false at all times
- Mutual exclusion must hold at all times (you can fudge things to get around this)

11.34. Hold and Wait

- Need to be sure a process does not hold one resource while requesting another
- Approach 1: Force a process to request all resources it needs at one time (usually at startup). The process dies, if not all ressources are available.
- Approach 2: If a process needs to acquire a new resource, it must first release all resources it holds, then reacquire all it needs
- Problems:
  - resource utilization may be low
  - starvation is possible

11.35. Circular Wait

Have a situation in which there are K processes holding units of K resources

- There is a cycle in the graph of processes and and resources
- Choose a resource request strategy by which no cycle will be introduced
- Total order on all resources, then can only ask for resources in numerical order (a minor variation is to merely insist that no process request a resource lower than what it is already holding).
- For example, if a set of resource types includes tape drives, disks, and printers, then the weights might be assigned as follows:

  - W(tape drive) = 1
  - W(disk drive) = 5
  - W(printer) = 12
- If A needs tape, disk and printer and B needs printer and disk
  - A → tape,
  - B → disk,
  - B → printer,
  - A → disk,
  - A → printer
- F should be defined in order of normal usage
- Proof by contradiction
  - Assume a circular wait exists
  - Let the set of processes involved in the circular wait be {P(0) , P(1) , P(2) , , P(n) }, where P(i) is waiting for a resource R(i), which is held by process P((i+1)%(n+1)) (modulo arithmetic is used on the indexes, so that P(n) is waiting on R(n) which is held by P(0))
  - Then, since P(i+1) is holding R(i) while requesting R(i+1), then W(R i) < W(R i+1) for all i
  - But this means that
    W(R 0) < W(R 1) < W(R 0)
    - By transitivity, W(R 0) < W(R 0), which is impossible

11.36. Avoid Starvation and Deadlock

- Fairness is a problem, if several concurrent threads are competing for resources.
- A system is fair when each thread gets enough access to a limited resource to make a reasonable progress.
- A fair system prevents starvation and deadlock. Starvation occurs when one or more threads in your program is blocked from gaining access to a resource and thus cannot make progress.
- Deadlock is the ultimate form of starvation; it occurs when two or more threads are waiting on a condition that cannot be satisfied. Deadlock most often occurs when two (or more) threads are each waiting for the other(s) to do something.

11.37. DeadLocks

- Is there a dead lock in this program?
  und ist im synchronozierten block, bevor dem Aufruf inprtected_1 und fuer den ersten gilt das aequivalente, dann ist ein deadlock.

```
1      import java.util.*;
2
3      public class DeadLock extends Thread    {
4              private static String o1  = new String();
5              private static String o2  = new String();
6              private String info;
7
```

```
 8              public DeadLock (String info) {
 9                      this.info    = info;
10              }
11
12              private void inProtected_1 () {
13                      synchronized ( o2 )   {
14                              inProtected_2();
15                      }
16              }
17
18              private void inProtected_2 () {
19                      synchronized ( o1 )   {
20                              inProtected_1();
21                      }
22              }
23
24              public void run () {
25                      if ( info.equals("first") )     {
26                              synchronized ( o1 )   {
27                                      inProtected_1();
28                              }
29                      } else
30                              synchronized ( o2 )   {
31                                      inProtected_2();
32                              }
33              }
34
35              public static void main (String args []) {
36                      new DeadLock("second").start();
37                      new DeadLock("first").start();
38              }
39      }
```

Source Code: Src/11/DeadLock.java

- Is there a dead lock in this program?

  Ja, falls es jeder thread in den erste. s. block schaft. Sonst, Nein, aber ein StackOverflow wird eintreten.

- Is there a dead lock in this program?

11.38. Dining Philosophers

The dining philosophers are often used to illustrate various problems that can occur when many synchronized threads are competing for limited resources.

The story goes like this: Five philosophers are sitting at a round table. In front of each philosopher is a bowl of rice. Between each pair of philosophers is one chopstick. Before an individual philosopher can take a bite of rice he must have two chopsticks — one taken from the left, and one taken from the right. The philosophers must find some way to share chopsticks such that they all eat with an reasonable frequency.

```
 1
 2      import java.util.Random;
 3      import java.util.Vector;
 4
 5      /** A class implementing the Dining Philosphers */
 6      public class Philosopher extends Thread {
 7
 8        protected static Random random = new Random();        // randomize
 9        protected int me;                     // number for trace
10        protected Integer left, right;        // my chopsticks
11
12        public Philosopher (int me, Integer left, Integer right) {
13          this.me = me; this.left = left; this.right = right;
14        }
15      /** philosopher's body: think and eat 5 times */
16        public void run () {
17          for (int n = 1; n <= 5; ++ n) {
18            System.out.println(me+" thinks");
19            try {
20              Thread.sleep((long)(random.nextFloat()*1000));
21            } catch(Exception e) {
22              e.printStackTrace();
23            }
24            System.out.println(me+" is trying to eat");
25            synchronized ( left )  {
26                 synchronized ( right )  {
27                         System.out.println("\t" + me+" eats");
28                         try {
29                           Thread.sleep((long)(random.nextFloat()*1000));
30                         } catch(Exception e) {
31                             e.printStackTrace();
32                         }
33                }
34          }
35          System.out.println("\t" + me+" leaves");
36        }
37       }
38      /** sets up for 5 philosophers */
39        public static void main (String args []) {
40          Integer f[] = new Integer[5];
41          for (int n = 0; n < 5; ++ n)
42                      f[n] = new Integer(n);
43          Philosopher p[] = new Philosopher[5];
44          p[0] = new Philosopher(0, f[4], f[0]);      // backwards
45          for (int n = 1; n < 5; ++ n)
46                      p[n] = new Philosopher(n, f[n-1], f[n]);
47          for (int n = 0; n < 5; ++ n) p[n].start();
48        }
49      }
```

Source Code: Src/11/Philosopher.java

What is wrong with this solution?

Angenommen es sind nur zwei Philosopher am Tisch un jeder greift nach seiner linken

Gabel und is erfolgreich, dann hat er keinen Zugriff zu rechten Gabel.

11.39. Semaphore

1965, suggested Edsger. W. Dijkstra to using an integer variable to count the number of wake ups: Semaphores (semaphore is a greek word, it stands for signal). A synchronization variable that take a positive integer variable. A semaphore has two operations:

- P (dutch for "to test", proberen): an atomic operation that waits for the semaphore to become positive, then decrements it by 1.
- V (dutch for "to increment", verhogen): an atomic operation that increments the semaphore by 1.

The P(S) operation on Semaphore S is:

```
If S > 0 then
       S := S - 1
else
       (Wait on S)
```

The V(S) operation on Semaphore S is:

```
If (One or more processes are waiting on S) then
       (Let one of the processes proceed)
else
       S := S + 1
```

It is assumed that P() and V() are indivisible.

If a thread tries to make a semaphore value to become negative, the thread is blocked until another thread makes the semaphore value positive.

```
 1      public class Semaphore {
 2              protected int n;
 3
 4              public Semaphore (int n) {
 5                      this.n = n;
 6              }
 7
 8              public synchronized void P () {
 9                      if (n <= 0) {
10                              try {
11                                      wait();         // see in object
12                              } catch(Exception e) {
13                                      e.printStackTrace();
14                              }
15                      }
16                      -- n;
17              }
18
19              public synchronized void V () {
20                      if (++ n > 0)
21                              notify();               // see in object
22              }
23      }
```

Source Code: Src/11/Semaphore.java

A thread blocks by calling wait(). Notification happens through notify() which, however, only releases one waiting thread from being blocked.

11.40. Semaphore II

```
 1      public class S extends Thread {
 2              protected int n;
 3
 4              public S (int n) {
 5                      this.n = n;
 6              }
 7
 8              public synchronized void P () {
 9                      if (n <= 0) {
10                              try {
11                                      wait();         // see in object
12                              } catch(Exception e) {
13                                      e.printStackTrace();
14                              }
15                      }
16                      -- n;
17              }
18
19              public synchronized void V () {
20                      if (++ n > 0)
21                              notify();               // see in object
22              }
23
24              public  void run() {
25                      for ( int i = n; i >= 0; i -- ) {
26                              System.out.println("i = " + i);
27                              System.out.println("before P ");
28                              P();
29                              System.out.println("after P ");
30                      }
31              }
32              static public void main(String args[] ) {
33                      S aS1 = new S(3);
34                      aS1.start();
35                      try {
36                              System.out.println("\tsleeping ...");
37                              sleep(1000);
38                              System.out.println("\tbefore V ");
39                              aS1.V();
40                              System.out.println("\tafter V ");
41                      }
42                      catch (  InterruptedException e ) {
43                              System.err.println("\tInterrupted!");
44                      }
45              }
46      }
```

Source Code: Src/11/S.java

```
i = 3
before P
after P
i = 2
before P
after P
i = 1
before P
after P
i = 0
before P
        sleeping ...
        before V
after P
        after V
```

11.41. Producer-Consumer Problem

Two processes share a common fixed size buffer. One of them, the producer, puts information into the buffer and the consumer takes it out.

Trouble arises when the producer wants to put a new item in the buffer, but it is already full. Similarily, if the consumer wants to remove an item from the buffer when the buffer is empty.

We will use three semaphores to solve this problem.

- *full* for counting the number of slots that are full.
- *empty* for counting the number of slots that are empty.
- *mutex* to make sure the producer and consumer do not access the buffer at the same time.

- *full* is initially 0
- *empty* is initially N
- *mutex* is initially 1 $\rightarrow$ no process is in its critical region.

What do you think about:

```
 1      public class Consumer extends Thread {
 2        final int N = 100;
 3        Semaphore mutex = new Semaphore(1);    // access the buffer semaphor
 4        Semaphore empty = new Semaphore(N);    // number of emtpy slots
 5        Semaphore full  = new Semaphore(0);    // number of used slots
 6
 7        void insertItem(int i)        {       }
 8        void consumeItem(int i)       {       }
 9        void workWithIt(int i)        {       }
10
11        void producer()
12        {
13        int item;
14                while ( true )       {
15                    item = (int)Math.random();
16                    empty.P();                // decrement count of full slo
17                    mutex.P();                // enter critical region
18                    insertItem(item);
19                    mutex.V();                // leave critical region
20                    full.V();                 // increment count of full slo
21              }
22        }
23        void consumer()
24        {
25        int item = 3;
26                while ( true )       {
27                    full.P();                 // decrement count of full slo
28                    mutex.P();                // enter critical region
29                    consumeItem(item);        // take it
30                    mutex.V();                // enter critical region
31                    empty.V();                // increment count of full slo
32                    workWithIt(item);
33                }
34        }
35      }
```

Source Code: Src/11/Consumer.java

What do you think about:

```
 1      public class Consumer_2 extends Thread {
 2        final int N = 100;
 3        Semaphore mutex = new Semaphore(1);    // access the buffer semaphor
 4        Semaphore empty = new Semaphore(N);    // number of emtpy slots
 5        Semaphore full  = new Semaphore(0);    // number of used slots
 6
 7        void insertItem(int i)        {       }
 8        void consumeItem(int i)       {       }
 9        void workWithIt(int i)        {       }
10
11        void producer()
12        {
13        int item;
14              // Die beiden down-Operationen des Erzeugers sind ausgefuehrt.
15              // Falls der Puffer voll ist, wird der Erzeuger blockiert und
16              // Wenn der Verbraucher das naechste Mal auf den Puffer zugrei
17                while ( true )       {
18                    item = (int)Math.random();
19                    mutex.P();                // enter critical region
20                    empty.P();                // decrement count of full slo
21                    insertItem(item);
22                    mutex.V();                // leave critical region
23                    full.V();                 // increment count of full slo
24              }
25        }
26        void consumer()
27        {
28        int item = 3;
29                while ( true )       {
30                    full.P();                 // decrement count of full slo
31                    mutex.P();                // enter critical region
32                    consumeItem(item);        // take it
33                    mutex.V();                // enter critical region
34                    empty.V();                // increment count of full slo
35                    workWithIt(item);
36                }
37        }
38      }
```

Source Code: Src/11/Consumer_2.java

11.42. Questions

What is going on here?

•

```
 1      public class T_1 extends Thread    {
 2
 3          private static synchronized void inProtected () {
 4             System.err.println("--> ");
 5             try {
 6                    sleep(1000);
 7             }
 8             catch (  InterruptedException e ) {
 9                      System.err.println("Interrupted!");
10             }
11             System.err.println("<-- ");
12          }
13
14          public void run () {
15              inProtected();
16          }
17          public static void main (String args []) {
18              new T_1().start();
19              new T_1().start();
20              new T_1().start();
21          }
22      }
```

Source Code: Src/11q/T_1.java

```
-->
<--
-->
<--
-->
<--
```

```
 1      public class T_2 extends Thread    {
 2          private String info;
 3
 4          public T_2 (String info) {
 5              this.info    = new String(info);
 6          }
 7
 8          private void inProtected () {
 9             synchronized ( info )       {
10                   System.err.println("--> " + info);
11                   try {
12                            sleep(1000);
13                   } catch (  Exception e ) {
14                       e.printStackTrace();
15                   }
16                   System.err.println("<-- " + info);
17             }
18          }
19
20          public void run () {
21              inProtected();
22          }
23          public static void main (String args []) {
24              String a = "hello";
25              new T_2(a).start();
26              new T_2(a).start();
27
28          }
29      }
```

Source Code: Src/11q/T_2.java

```
--> hello
--> hello
<-- hello
<-- hello
```

•

```
 1      import java.util.Vector;
 2      public class T_3 extends Thread    {
 3          static Vector aVector = new Vector();
 4          private int info;
 5
 6          public T_3 (int info) {
 7              this.info    = info;
 8          }
 9
10          public synchronized void run () {
11              System.err.println("--> " + info);
12               try {
13                       sleep(1000);
14               } catch (  Exception e ) {
15                      e.printStackTrace();
16               }
17               System.err.println("<-- " + info);
18          }
19
20          public static void main (String args []) {
21              for ( int i = 1; i < 100; i ++ )
22                      new T_3(i).start();
23          }
24      }
```

Source Code: Src/11q/T_3.java

```
--> 1
--> 2
--> 3
--> 4
--> 5
 ...
<-- 94
<-- 95
<-- 96
<-- 97
<-- 98
<-- 99
```

•

```
 1      import java.util.*;
 2
 3      public class T_4_1 extends Thread    {
 4          static Object o = new Object();
 5          public T_4_1()      {
 6              o  = new Object();
 7          }
 8          public void run () {
 9              synchronized ( o ) {
```

```
10                  System.err.println("--->");
11                  try {
12                          sleep(1000);
13                  }
14                  catch (  InterruptedException e ) {
15                      System.err.println("Interrupted!");
16                  }
17                  System.err.println("<---" );
18              }
19          }
20
21          public static void main (String args []) {
22              new T_4_1().start();
23              new T_4_1().start();
24              new T_4_1().start();
25          }
26      }
```

Source Code: Src/11q/T_4_1.java

```
--->
<---
--->
<---
--->
<---
```

```
 1      import java.util.*;
 2
 3      public class T_4 extends Thread    {
 4          static Object o = new Object();
 5          public void run () {
 6              synchronized ( o ) {
 7                  System.err.println("--->");
 8                  try {
 9                          sleep(1000);
10                  }
11                  catch (  InterruptedException e ) {
12                      System.err.println("Interrupted!");
13                  }
14                  System.err.println("<---" );
15              }
16          }
17
18          public static void main (String args []) {
19              new T_4().start();
20              new T_4().start();
21              new T_4().start();
22          }
23      }
```

Source Code: Src/11q/T_4.java

```
--->
<---
--->
<---
--->
<---
```

```
 1      import java.util.*;
 2
 3      public class T_5 extends Thread    {
 4          static Object o = new Object();
 5          static int    counter = 0;
 6
 7          public void run () {
 8              if ( ++counter == 1 )
 9                      o = new Object();
10      //      read x
11
12              synchronized ( o ) {
13                  System.err.println("--->" );
14                  try {
15                          sleep(1000);
16                  }
17                  catch (  InterruptedException e ) {
18                      System.err.println("Interrupted!");
19                  }
20                  System.err.println("<---" );
21              }
22          }
23
24          public static void main (String args []) {
25              new T_5().start();
26              new T_5().start();
27              new T_5().start();
28          }
29      }
```

Source Code: Src/11q/T_5.java

```
--->
<---
--->
<---
--->
<---
```

•

```
 1      import java.util.*;
 2
 3      public class T_6 extends Thread    {
 4          static Object o = new Object();
 5          static int    counter = 0;
 6
 7          public void run () {
 8              if ( counter++ == 1 )
 9                      o = new Object();
10
11              synchronized ( o ) {
12                  System.err.println("--->" );
13                  try {
14                          sleep(1000);
15                  }
16                  catch (  InterruptedException e ) {
17                      System.err.println("Interrupted!");
18                  }
19                  System.err.println("<---" );
20              }
21          }
22
23          public static void main (String args []) {
24              new T_6().start();
25              new T_6().start();
26              new T_6().start();
27          }
28      }
```

Source Code: Src/11q/T_6.java

```
--->
--->
<---
<---
--->
<---
```

• Will the following program terminate?

```
import java.util.Vector;
public class T_1 extends Thread    {
    private String info;
    Vector aVector;
    Vector bVector;

    public T_1 (String info, Vector aVector) {
        this.info = info;
        this.aVector = aVector;
    }
     public void run() {
       synchronized ( aVector )    {
        if (  info.equals("last") )    {
```

```
                aVector.notifyAll();
            } else {
                System.out.println(info + " is waiting");
                try {
                    aVector.wait();
                } catch ( Exception  e )    {
                    System.out.println(info +
                      ": InterruptedException");
                }
                System.out.println(info + " is awake!");
            }
          }
        }
        public static void main (String args []) {
            Vector aVector = new Vector();
            Vector bVector = new Vector();

            new T_1("first",  aVector).start();
            new T_1("second", bVector).start();
            new T_1("last",   bVector).start();
        }
    }
```

11.43. Questions

11.44. Questions from Students

```
 1      import java.util.*;
 2      public class M extends Thread    {
 3          private String info;
 4          private Vector aVector;
 5
 6          public M (String info) {
 7              this.info    = info;
 8          }
 9          private synchronized void inProtected () {
10              System.err.println(info + ": is in protected()");
11              try {
12                      sleep(1000);
13              }
14              catch (  InterruptedException e ) {
15                  System.err.println("Interrupted!");
16              }
17              System.err.println(info + ": exit run");
18          }
19          public void run () {
20              inProtected();
21          }
22          public static void main (String args []) {
23              Vector aVector = new Vector();
24              M aT4_0 = new M("first");
25              M at5_0 = new M("second");
```

```
26
27              aT4_0.start();
28              at5_0.start();
29              aT4_0.inProtected();
30              at5_0.inProtected();
31          }
32      }
33
```

Source Code: Src/StudentT_Q/M.java

```
 1      /*
 2
 3      Q2. If the object is synchronized in the main method, what is the sign
 4
 5      Q3.In what scenario will a running thread go to a ready state?
 6
 7      Q4.In the slide numbered 12.8, it says Interface methods cannot be syc
 8      */
 9      public class T_7 extends Thread    {
10          static String  theValue ;
11          T_7(String theValue)        {
12              this.theValue = theValue;
13          }
14          public void run () {
15              synchronized ( theValue )       {
16                      if ( this.theValue.equals("1") )
17                              theValue = "3";
18                      else
19                              theValue = "4";
20                      }
21          }
22
23          public static void main (String args []) {
24              T_7 aT_7_1 = new T_7("1");
25              T_7 aT_7_2 = new T_7("1");
26              aT_7_1.run();
27              aT_7_2.run();
28              synchronized ( theValue )       {
29                      System.out.println("aT_7_1.theValue _7.i = " + aT_7_1.
30                      System.out.println("aT_7_2.theValue   = " + aT_7_2.the
31              }
32          }
33      }
```

Source Code: Src/StudentT_Q/T_7.java

12. Lambda Expressions

Material copied from

- Enable to treat functionality as a method argument, or code as data.
- A function that can be created without belonging to any class.
- A lambda expression can be passed around as if it was an object and executed on demand.
- Lambda expressions provide a clear and concise way to represent one method interface using an expression.
- Lambda expressions helps to iterate, filter and extract data from collection.

Good Use Cases:

— Selection

— Pre/Post-conditions

12.1. Java Lambda Expression Example

```
 1      interface LambdaExpression1Interface {
 2
 3              void abstractFun(int x);
 4      }
 5
 6      public class LambdaExpression1 {
 7
 8         public static void main(String args[]) {
 9              LambdaExpression1Interface lambdaObj = (int aInt)->System.out.
10
11              lambdaObj.abstractFun(42);
12          }
13
14      }
```

Source Code: Src/18_1/LambdaExpression1.java

```
java LambdaExpression1
42
```

```
 1      import java.util.ArrayList;
 2      class LambdaExpression2
 3      {
 4          public static void main(String args[])
 5          {
 6              ArrayList<Integer> arrL = new ArrayList<Integer>();
 7              arrL.add(1); arrL.add(2); arrL.add(3); arrL.add(4);
 8
 9              arrL.forEach(n -> { if (n % 2 == 0) System.out.println(n); });
10          }
11      }
```

Source Code: Src/18_1/LambdaExpression2.java

```
java LambdaExpression2
1
2
3
4
2
4
```

```
 1      import java.util.ArrayList;
 2      class LambdaExpression3
 3      {
 4          public static void main(String args[])
 5          {
 6              ArrayList<Integer> arrL = new ArrayList<Integer>();
 7              arrL.add(1); arrL.add(2); arrL.add(3); arrL.add(4);
 8
 9              arrL.forEach(n -> { if (n % 2 == 0) System.out.println(n); });
10          }
11      }
```

Source Code: Src/18_1/LambdaExpression3.java

```
java LambdaExpression3
2
4
```

12.2. Java Lambda Expression Syntax

```
(argument-list) -> {body}
```

- Argument-list: It can be empty or non-empty as well.
- Arrow-token: It is used to link arguments-list and body of expression.
- Body: It contains expressions and statements for lambda expression.
- No return type: The java 8 compiler is able to infer the return type by checking the code. you need not to mention it explicitly.

12.3. Matching Lambdas to Interfaces

- A single method interface is referred to as a functional interface.
- Matching a Java lambda expression against a functional interface is divided into these steps:
    - Does the interface have only one method?
    - Does the parameters of the lambda expression match the parameters of the single method?
    - Does the return type of the lambda expression match the return type of the single method?

If the answer is yes to these three questions, then the given lambda expression is matched successfully against the interface.

12.4. Examples

The following examples show different ways om solution for the same problem.

12.5. Example - Without Lambda

```
 1      public class Id {
 2
 3          private String name;
 4          private int number;
 5
 6          public Id(String name, int number) {
 7              this.name = name;
 8              this.number = number;
 9          }
10          public String toString() {
11              return "my name is: " + name + "/" + number;
12          }
13          public String getName() {
14              return name;
15          }
16          public int getNumber() {
17              return number;
18          }
19          public void printNumber() {
20              System.out.println("    " + number);
21          }
22          public void printName() {
23              System.out.println("    " + name);
24          }
25      }
```

Source Code: Src/18_1/Id.java

```
 1      import java.util.Arrays;
 2      import java.util.Comparator;
 3      import java.util.Collections;
 4      import java.util.ArrayList;
 5      import java.util.List;
 6      import java.util.stream.Collectors;
 7
 8      public class WithoutLambda {
 9
10          public static void printIdsHigherThan(List<Id> roster, int low, in
11              for (Id p : roster) {
12                  if (  ( low <= p.getNumber() )
13                     && ( p.getNumber() <= high ) )
14                      p.printNumber();
15              }
16          }
17          public static void main(String[] args) {
18              List<Id> persons = Arrays.asList(
19                      new Id("Franz",         1234),
```

```
20                      new Id("Wolfgang",      123),
21                      new Id("Thomas",        12) );
22
23              printIdsHigherThan(persons, 12, 123);
24          }
25
26      }
```

Source Code: Src/18_1/WithoutLambda.java

12.6. Example - Without Lambda - With Interface

```
1       interface Check {
2           boolean doTheCheck(Id thisId, int low, int high);
3       }
```

Source Code: Src/18_1/Check.java

```
1       import java.util.Arrays;
2       import java.util.Comparator;
3       import java.util.Collections;
4       import java.util.ArrayList;
5       import java.util.List;
6       import java.util.stream.Collectors;
7
8       public class WithoutLambdaInterface implements Check {
9
10          public boolean doTheCheck(Id thisId, int low, int high) {
11              return (  ( low <= thisId.getNumber() )
12                     && ( thisId.getNumber() <= high ) );
13          }
14          public void printIdsHigherThan(List<Id> roster, int low, int high)
15              for (Id p : roster) {
16                  if ( doTheCheck(p, low, high) )
17                      p.printNumber();
18              }
19          }
20          public void printIdsHigherThanWithTester(List<Id> roster,
21                      Check tester,
22                      int low, int high) {
23              for (Id p : roster) {
24                  if ( tester.doTheCheck(p, low, high) )
25                      p.printNumber();
26                   }
27          }
28          public void work()  {
29              List<Id> persons = Arrays.asList(
30                      new Id("Franz",         1234),
31                      new Id("Wolfgang",      123),
32                      new Id("Thomas",        12) );
33
34              System.out.println("printIdsHigherThan(persons, 12, 123);");
```

```
35              printIdsHigherThan(persons, 12, 123);
36
37              System.out.println("printIdsHigherThanWithTester(persons, new
38              printIdsHigherThanWithTester(persons, new WithoutLambdaInterfa
39          }
40          public static void main(String[] args) {
41              new WithoutLambdaInterface().work();
42          }
43      }
```

Source Code: Src/18_1/WithoutLambdaInterface.java

12.7. Example - Without Lambda - With Anonymous Class

```
 1      import java.util.Arrays;
 2      import java.util.Comparator;
 3      import java.util.Collections;
 4      import java.util.ArrayList;
 5      import java.util.List;
 6      import java.util.stream.Collectors;
 7
 8      public class WithoutLambdaAnonmymousClass {
 9
10          public void printIdsHigherThanWithTester(List<Id> roster,
11                      Check tester,
12                      int low, int high) {
13              for (Id p : roster) {
14                  if ( tester.doTheCheck(p, low, high) )
15                      p.printNumber();
16                   }
17          }
18          public void work()  {
19              List<Id> persons = Arrays.asList(
20                      new Id("Franz",         1234),
21                      new Id("Wolfgang",      123),
22                      new Id("Thomas",        12) );
23
24              printIdsHigherThanWithTester(persons, new Check() {
25                      public boolean doTheCheck(Id thisId, int low, int high
26                              return (  ( low <= thisId.getNumber() )
27                                      && ( thisId.getNumber() <= high ) );
28
29                              } }, 12, 123);
30          }
31          public static void main(String[] args) {
32              new WithoutLambdaAnonmymousClass().work();
33          }
34      }
```

Source Code: Src/18_1/WithoutLambdaAnonmymousClass.java

12.8. Example - With Lambda

```
 1      interface Addable{
 2          int add(int a,int b);
 3      }
 4
 5      public class WithLambda_0{
 6          public static void main(String[] args) {
 7
 8              Addable ad1=(a,b)->(a+b);
 9              System.out.println(ad1.add(10,20));
10
11              Addable ad2=(int a,int b)->(a+b);
12              System.out.println(ad2.add(100,200));
13          }
14      }
```

Source Code: Src/18_1/WithLambda_0.java

12.9. Example - With Lambda 1

```
 1      import java.util.Arrays;
 2      import java.util.Comparator;
 3      import java.util.Collections;
 4      import java.util.ArrayList;
 5      import java.util.List;
 6      import java.util.stream.Collectors;
 7
 8      public class WithLambda_1 {
 9
10          public void work()  {
11              List<Id> persons = Arrays.asList(
12                      new Id("Franz",         1234),
13                      new Id("Wolfgang",      123),
14                      new Id("Thomas",        12) );
15
16              persons.forEach(  (p)->System.out.println(p.getName())  );
17          }
18          public static void main(String[] args) {
19              new WithLambda_1().work();
20          }
21      }
```

Source Code: Src/18_1/WithLambda_1.java

12.10. Example - With Lambda 2

```
 1      import java.util.Arrays;
 2      import java.util.Comparator;
 3      import java.util.Collections;
 4      import java.util.ArrayList;
 5      import java.util.List;
 6      import java.util.stream.Collectors;
 7
 8      public class WithLambda_2 implements CheckNoLimit {
 9
10          public boolean doTheCheck(Id p)  {
11                      return p.getNumber() >= 12  && p.getNumber() <= 123;
12          }
13          public static void printPersons(List<Id> roster, CheckNoLimit test
14                  for (Id p : roster) {
15                      if (tester.doTheCheck(p))
16                          p.printName();
17                  }
18          }
19
20          public void work()  {
21              List<Id> persons = Arrays.asList(
22                      new Id("Franz",         1234),
23                      new Id("Wolfgang",      123),
24                      new Id("Thomas",        12) );
25
26              printPersons(persons, new WithLambda_2() );
27          }
28          public static void main(String[] args) {
29              new WithLambda_2().work();
30          }
31      }
```

Source Code: Src/18_1/WithLambda_2.java

12.11. Example - With Lambda 3

```
 1      import java.util.Arrays;
 2      import java.util.Comparator;
 3      import java.util.Collections;
 4      import java.util.ArrayList;
 5      import java.util.List;
 6      import java.util.stream.Collectors;
 7
 8      public class WithLambda_3 {
 9
10          public static void printPersons(List<Id> roster, CheckNoLimit test
11                  for (Id p : roster) {
12                      if (tester.doTheCheck(p))
13                          p.printName();
14                  }
15          }
16
17          public void work()  {
```

```
18              List<Id> persons = Arrays.asList(
19                      new Id("Franz",         1234),
20                      new Id("Wolfgang",      123),
21                      new Id("Thomas",        12) );
22
23              printPersons(persons, (Id p) -> p.getNumber() >= 12 && p.getNu
24          }
25          public static void main(String[] args) {
26              new WithLambda_3().work();
27          }
28      }
```

Source Code: Src/18_1/WithLambda_3.java

12.12. Example - With Lambda 4

```
 1      import java.util.Arrays;
 2      import java.util.Comparator;
 3      import java.util.Collections;
 4      import java.util.ArrayList;
 5      import java.util.List;
 6      import java.util.stream.Collectors;
 7
 8      public class WithLambda_4 implements Check {
 9
10          public boolean doTheCheck(Id p, int low, int high)  {
11                      return p.getNumber() >= low  && p.getNumber() <= high;
12          }
13          public static void printPersons(List<Id> roster, Check tester, int
14                  for (Id p : roster) {
15                      if (tester.doTheCheck(p, low, high))
16                          p.printName();
17                  }
18          }
19
20          public void work()  {
21              List<Id> persons = Arrays.asList(
22                      new Id("Franz",         1234),
23                      new Id("Wolfgang",      123),
24                      new Id("Thomas",        12) );
25
26              printPersons(persons, new WithLambda_4(), 12, 123 );
27          }
28          public static void main(String[] args) {
29              new WithLambda_4().work();
30          }
31      }
```

Source Code: Src/18_1/WithLambda_4.java

12.13. Example - With Lambda 5

```
 1      import java.util.Arrays;
 2      import java.util.Comparator;
 3      import java.util.Collections;
 4      import java.util.ArrayList;
 5      import java.util.List;
 6      import java.util.stream.Collectors;
 7
 8      public class WithLambda_5 implements Check {
 9
10          public boolean doTheCheck(Id p, int low, int high)  {
11                      return p.getNumber() >= low  && p.getNumber() <= high;
12          }
13          public static void printPersons(List<Id> roster, Check tester, int
14                  for (Id p : roster) {
15                      if (tester.doTheCheck(p, low, high))
16                          p.printName();
17                  }
18          }
19
20          public void work()  {
21              List<Id> persons = Arrays.asList(
22                      new Id("Franz",         1234),
23                      new Id("Wolfgang",      123),
24                      new Id("Thomas",        12) );
25
26              printPersons(persons, (Id p, int low, int high) -> p.getNumber
27          }
28          public static void main(String[] args) {
29              new WithLambda_5().work();
30          }
31      }
```

Source Code: Src/18_1/WithLambda_5.java

13. Collections

40 questions:

13.1. What is a Collection

Stolen from

- A is simply an object that groups multiple elements into a single unit.
- Collections are used to store, retrieve and manipulate data, and to transmit data from one method to another.
- The primary use of the Collection interface is to pass around collections of objects where maximum generality is desired.
- Collections typically represent data items that form a natural group.

Sorting:

```
 1      import java.util.*;
 2
 3      public class Sort {
 4          public static void main(String args[]) {
 5              List l = Arrays.asList(args);
 6              Collections.sort(l);
 7              System.out.println(l);
 8          }
 9      }
10
```

Source Code: Src/9/Sort.java

```
% java Sort X Mac OS
[Mac, OS, X]
```

- How does this work? In detail.... See here:

13.2. How could we Implement the Previous Example?

- Assume the list is an array
- We handle only String objects
- How about this:

```
 1      public class BubbleSort {
 2
 3        public static void printIt(String aCollection[] )     {
 4                for (int index=0; index<aCollection.length; index++)
 5                      System.out.println(index + "\t" + aCollection[index] )
 6        }
 7
 8        public static void sort(String aCollection[] )     {
 9          for (int index=0; index < aCollection.length - 1; index++)     {
10            for (int walker=0; walker < aCollection.length - 1; walker++)  {
11              if ( aCollection[walker].compareTo(aCollection[walker+1]) > 0
12                              String tmp = aCollection[walker];
13                              aCollection[walker] = aCollection[walker + 1];
14                              aCollection[walker+1] = tmp;
15              }
16            }
17          }
18        }
19
20        public static void main( String args[] ) {
21          String[] aCollection = new String[3];
22          aCollection[0] =  "c";
23          aCollection[1] =  "b";
24          aCollection[2] =  "a";
25
26          sort(aCollection);
27          printIt(aCollection);
28        }
29      }
```

Source Code: Src/9/BubbleSort.java

- What do we need:
  - — a way to access every object in the array
  - — we use the String objects compareTo method
  - — Will this work for other kind of objects?

13.3. Implementation of Sort

```
 1      import java.util.*;
 2
 3      public class HpCollections {
 4
 5        static Object anArray[] = null;
 6
 7
 8        public static void sort(List aList) {
 9              anArray = aList.toArray();
10
11              for (int index=0; index<anArray.length - 1; index++)     {
12                  for (int walker=0; walker<anArray.length - index - 1; walk
13                      String left = (String) anArray[walker];
14                      String right = (String) anArray[walker+1];
15                      if ( left.compareTo( right ) > 0 )        {
16                              Object tmp = anArray[walker];
17                              anArray[walker] = anArray[walker + 1];
18                              anArray[walker+1] = tmp;
19                      }
20                  }
21              }
22              aList = Arrays.asList(anArray);
23
24        }
25        public String toString()      {
26              String s = new String ();
27              for (Object o: anArray )
28                      s = s + "/" + o ;
29              return s;
30         }
31
32        public static void main(String args[]) {
33              args = new String[4];
34              args[0] =  "z"; args[1] =  "x";
35              args[2] =  "a"; args[3] =  "t";
36              List l = Arrays.asList(args);
37              // HpCollections_remove.sort(l);
38              // HpOKCollections.sort(l);
39              // Collections.sort(l);
40              HpCollections.sort(l);
41              System.out.println(l);
42          }
43      }
```

44

Source Code: Src/Collection_5/HpCollections.java

```
1       import java.util.*;
2
3       public class HpCollections_remove {
4
5         public static void sort(List aList) {
6               Object anArray[] = aList.toArray();
7
8               for (int index=0; index<anArray.length - 1; index++)     {
9                   for (int walker=0; walker<anArray.length - index - 1; walk
10                      Comparable left = (Comparable) anArray[walker];
11                      Comparable right = (Comparable) anArray[walker+1];
12                      if ( left.compareTo( right ) > 0 )        {
13                              Object tmp = anArray[walker];
14                              anArray[walker] = anArray[walker + 1];
15                              anArray[walker+1] = tmp;
16                      }
17                  }
18              }
19
20              for (Object o: anArray )
21                      System.out.println("anArray: " + o );
22              for (int index=0; index<anArray.length ; index++)       {
23                      aList.remove(index );
24                      aList.add(anArray[index] );
25              }
26        }
27
28      }
29
```

Source Code: Src/Collection_5/HpCollections_remove.java

```
1       import java.util.*;
2
3       public class HpOKCollections {
4
5         public static void sort(List list) {
6               Object a[] = list.toArray();
7               Arrays.sort(a);         /////// this is the trick
8               ListIterator i = list.listIterator();
9               for (int j=0; j<a.length; j++) {
10                      i.next();               // this is it
11                      i.set(a[j]);            // modification of the list
12              }
13        }
14
15      }
16
```

Source Code: Src/Collection_5/HpOKCollections.java

```
1       import java.util.*;
2
3       public class Sort {
4           public static void main(String args[]) {
5               args = new String[4];
6               args[0] =  "z"; args[1] =  "x";
7               args[2] =  "a"; args[3] =  "a";
8               List l = Arrays.asList(args);
9               //  Collections.sort(l);
10              HpOKCollections.sort(l);
11              System.out.println(l);
12          }
13      }
14
```

Source Code: Src/Collection_5/Sort.java

Why does it not work?

- take a look at this class

13.4. What Is a Collections Framework?

A collections framework is a unified architecture for representing and manipulating collections. All collections frameworks contain three things:

- Interfaces: abstract data types representing collections. Interfaces allow collections to be manipulated independently of the details of their representation.
- Implementations: concrete implementations of the collection interfaces. In essence, these are reusable data structures.
- Algorithms: methods that perform useful computations, like searching and sorting, on objects that implement collection interfaces. These algorithms are said to be polymorphic because the same method can be used on many different implementations of the appropriate collections interface. In essence, algorithms are reusable functionality.

13.5. Iterators

- The object returned by the method is very similar to an Enumeration, but differs in two respects:
  - — Iterator allows the caller to remove elements from the underlying collection during the iteration with well-defined semantics.
  - — Method names have been improved.

Use of an iterator:

```
1
2       import java.util.*;
3
4       public class UseIterator {
5           public static void main(String args[]) {
6               int index = 0;
7               List l = Arrays.asList(args);
```

```
 8              Iterator aIterator = l.iterator();
 9              while ( aIterator.hasNext() )   {
10                      System.out.println(++index + ": " +
11                              (String)aIterator.next() );
12              }
13          }
14      }
```

Source Code: Src/9/UseIterator.java

```
java UseIterator A Day at the Races
1: A
2: Day
3: at
4: the
5: Races
```

13.6. Benefits of a Collections Framework

- It reduces programming effort
- It increases program speed and quality
- It allows interoperability among unrelated APIs
- It reduces effort to design new APIs
- It fosters software reuse

13.7. Collection Interface

- The core interfaces is:

copied from

- The interface is the root of the collection hierarchy. A Collection represents a group of objects, known as its elements. Some Collection implementations
  - — allow duplicate elements and others do not.
  - — some are ordered and others unordered.

  Collection is used to pass collections around and manipulate them when maximum generality is desired.
- Interfaces:
  Collection
  - Enumeration
  - List
  - Map
  - Queue
  - RandomAccess
  - Set
  - SortedMap
  - SortedSet

13.8. Set Interface

A is a collection that cannot contain duplicate elements.

- Two Set objects are equal if they contain the same elements.
- See also and
- A is an ordered collection

13.9. Lists

- Lists can contain duplicate elements.

13.10. Maps

- A is an object that maps keys to values.
- Maps cannot contain duplicate keys: Each key maps to one value.

13.11. Maps vs Collections

- Collections: add, remove, lookup
- Maps: key value pair, access values stored by key

13.12. A Picture

13.13. See Here

13.14. General Purpose Implementations

	Implementation			
	Hash Table	Resizable Array	Balanced Tree	Linked List
Interface Set	HashSet		TreeSet	
Interface List		ArrayList		LinkedList
Interface Map	HashMap		TreeMap	

- The fact that the new implementations are unsynchronized represents a break with the past
- If you need a synchronized collection, the synchronization wrappers, allow any collection to be transformed into a synchronized collection. Thus, synchronization is optional for the new collection implementations where it was mandatory for the old.
- As a rule of thumb, you should be thinking about the interfaces rather than the implementations.

13.15. Implementations: Set

- HashSet and
- TreeSet.
- HashSet is much faster (constant time vs. log time for most operations), but offers no ordering guarantees.
- If it is needed to use the operations in the SortedSet, or in-order iteration is important to use TreeSet.

13.16. Implementations: List

- ArrayList and
- LinkedList.
- ArrayList offers constant time positional access

13.17. Implementations: Map

- HashMap and
- TreeMap
- The situation for Map is exactly analogous to Set.

13.18. Algorithms

- prints out its arguments in lexicographic order :

```
1      import java.util.*;
2
3      public class Sort {
4          public static void main(String args[]) {
5              List l = Arrays.asList(args);
6              Collections.sort(l);
7              System.out.println(l);
8          }
9      }
```

```
10
```

Source Code: Src/9/Sort.java

13.19. Examples: HashSet

```
 1
 2      import java.util.HashSet;
 3      import java.util.Set;
 4
 5      public class HashSetEx_1 {
 6
 7          private Set<Integer> universe;
 8
 9          private Set<Integer> fill(int soMany) {
10              Set<Integer> universe = new HashSet<Integer>();
11              for ( int index = 0; index < soMany; index ++ )
12                      universe.add(new Integer(9999999 * index));
13              return universe;
14          }
15          public static void main(String args[])      {
16              Set<Integer> universe = null;
17              HashSetEx_1 aHashSetEx_1 = new HashSetEx_1();
18              universe = aHashSetEx_1.fill(253);
19              System.out.println("1: " + universe );
20
21              // universe.remove( new Integer(1) );
22              // System.out.println("2: " + universe );
23
24              universe.remove( new Integer(10) );
25              // System.out.println("3: " + universe );
26          }
27      }
28             /*
29              for(Integer id : stars.keySet()) {
30                  ids.add(id);
31
32                  if(ids.size() >= keepStars)
33                      break;
34              }
35
36              return ids;
37              */
```

Source Code: Src/9/HashSetEx_1.java

13.20. Examples: HashMap I

```
 1
 2      import java.util.HashMap;
 3      import java.util.Map;
 4      import java.util.Set;
 5      import java.util.Iterator;
 6
 7      public class HashMapEx {
 8
 9          private Map<Integer, String> universe;
10
11          private Map<Integer, String> fill(int soMany) {
12              universe = new HashMap<Integer, String>();
13              for ( int index = 0; index < soMany; index ++ )
14                      universe.put(new Integer(index), "_" + index);
15              return universe;
16          }
17
18          private Map<Integer, String> delete(int what) {
19              try {
20                      for (Integer id : universe.keySet() ) {
21                              System.out.println("try to delete: " + id);
22                              if ( id.equals(new Integer(what) ) )
23                                      universe.remove(id);
24                              System.out.println("deleted: " + id);
25                      }
26              } catch ( Exception e ) {
27                      System.out.println("Exception ..... ");
28                      e.printStackTrace();
29              }
30              return universe;
31
32          }
33      /*
34      The iterators returned by all of this class's "collection view methods
ConcurrentModificationException. Thus, in the face of concurrent modification,
35
36      Note that the fail-fast behavior of an iterator cannot be guaranteed a
on on a best-effort basis. Therefore, it would be wrong to write a program tha
37
38      */
39          private Map<Integer, String> deleteUsingKeySetCorrect(int what) {
40              try {
41                      Iterator aIterator = universe.keySet().iterator();
42                      while ( aIterator.hasNext() )           {
43                              aIterator.next();
44                              aIterator.remove();
45                      }
46              } catch ( Exception e ) {
47                      System.out.println("Exception ");
48                      e.printStackTrace();
49              }
50              return universe;
51
52          }
```

```
53
54          public static void main(String args[])      {
55              Map<Integer, String> universe;
56              HashMapEx aHashMapEx = new HashMapEx();
57              universe = aHashMapEx.fill(3);
58
59              System.out.println("1: " + universe );
60              aHashMapEx.deleteUsingKeySetCorrect(1);
61
62
63              universe = aHashMapEx.fill(3);
64              aHashMapEx.delete(1);
65              System.out.println("2: " + universe );
66
67          }
68      }
```

Source Code: Src/9/HashMapEx.java

```
1: {0=_0, 1=_1, 2=_2}
0
1
Exception .....
java.util.ConcurrentModificationException
    at java.util.HashMap$HashIterator.nextEntry(HashMap.java:793)
    at java.util.HashMap$KeyIterator.next(HashMap.java:828)
    at HashMapEx.delete(HashMapEx.java:18)
    at HashMapEx.main(HashMapEx.java:37)
2: {0=_0, 2=_2}
```

From: Note that this implementation is not synchronized. If multiple threads access this map concurrently, and at least one of the threads modifies the map structurally, it must be synchronized externally. (A structural modification is any operation that adds or deletes one or more mappings; merely changing the value associated with a key that an instance already contains is not a structural modification.) This is typically accomplished by synchronizing on some object that naturally encapsulates the map. If no such object exists, the map should be "wrapped" using the Collections.synchronizedMap method. This is best done at creation time, to prevent accidental unsynchronized access to the map:

```
Map m = Collections.synchronizedMap(new HashMap(...));
```

The iterators returned by all of this class's "collection view methods" are fail-fast: if the map is structurally modified at any time after the iterator is created, in any way except through the iterator's own remove or add methods, the iterator will throw a ConcurrentModificationException. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

Note that the fail-fast behavior of an iterator cannot be guaranteed as it is, generally speaking, impossible to make any hard guarantees in the presence of unsynchronized concurrent modification. Fail-fast iterators throw ConcurrentModificationException on a best-effort basis. Therefore, it would be wrong to write a program that depended on this exception for its correctness: the fail-fast behavior of iterators should be used only to detect bugs.

13.21. List Iterator

- An iterator for lists allows the programmer to traverse the list
  - — in either direction
  - — modify the list during iteration
  - — obtain the iterator's current position in the list.
- A ListIterator has no current element
- its cursor position always lies between the element that would be returned by a call to previous() and the element that would be returned by a call to next()
- In a list of length n, there are n+1 valid index values, from 0 to n, inclusive.

```
 1
 2      import java.util.Stack;
 3      import java.util.ListIterator;
 4      import java.util.Collection;
 5
 6      public class ListItereatorEx {
 7          // private Collection<String> palindrom;
 8          private Stack<String> palindrom;
 9
10          private Collection<String> fill(String words[]) {
11              palindrom  = new Stack<String>();
12              for (String id : words ) {
13                      palindrom.push(id);
14              }
15              return palindrom;
16          }
17
18          private Collection<String> leftRight()      {
19              ListIterator<String> aListIterator = palindrom.listIterator(2)
20              String s  = aListIterator.next();
21              System.out.println("s = " + s );
22              aListIterator.set("ZZ top");
23              return palindrom;
24          }
25
26          public static void main(String args[])      {
27              Collection<String> aStack;
28              String theOnes[] = { "a", "b", "c", "d" };
29              ListItereatorEx o = new ListItereatorEx();
30
31              aStack = o.fill(theOnes );
32              System.out.println("1: " + aStack );
33
34              aStack = o.leftRight();
35              System.out.println("2: " + aStack );
36
37          }
38      }
```

Source Code: Src/9/ListItereatorEx.java

Why is this important?

13.22. Collections.sort()

How does Collections.sort() work?

```
 1      import java.util.*;
 2
 3      public class Sort {
 4          public static void main(String args[]) {
 5              List l = Arrays.asList(args);
 6              Collections.sort(l);
 7              System.out.println(l);
 8          }
 9      }
10
```

Source Code: Src/9/Sort.java

From java doc

```
sort

public static void sort(List list)
Sorts the specified list into ascending order, according to the natural orderi
This sort is guaranteed to be stable: equal elements will not be reordered as

The specified list must be modifiable, but need not be resizable.

The sorting algorithm is a modified mergesort (in which the merge is omitted i

Parameters:
list - the list to be sorted.
Throws:
ClassCastException - if the list contains elements that are not mutually compa
UnsupportedOperationException - if the specified list's list-iterator does not
See Also:
Comparable
```

13.23. Object Ordering

- A List l may be sorted as follows:

  ```
  Collections.sort(l);
  ```

Class	Natural Ordering
Byte	signed numerical
Character	unsigned numerical
Long	signed numerical
Integer	signed numerical
Short	signed numerical
Double	signed numerical
Float	signed numerical
BigInteger	signed numerical

BigDecimal	signed numerical
File	system-dependent lexicographic on pathname.
String	lexicographic
Date	chronological
CollationKey	locale-specific lexicographic

- The Comparable interface consists of a single method:

```
public interface Comparable {
      public int compareTo(Object o);
}
```

Example:

```
 1      import java.util.*;
 2
 3      public class Name implements Comparable {
 4          protected String  firstName, lastName;
 5
 6          public Name(String firstName, String lastName) {
 7              if (firstName==null || lastName==null)
 8                  throw new NullPointerException();
 9              this.firstName = firstName;
10              this.lastName = lastName;
11          }
12
13          public String firstName()    {
14              return firstName;
15          }
16          public String lastName()     {
17              return lastName;
18          }
19
20          public boolean equals(Object o) {
21              if (!(o instanceof Name))
22                  return false;
23              Name n = (Name)o;
24              return n.firstName.equals(firstName) &&
25                     n.lastName.equals(lastName);
26          }
27
28          public String toString() {
29              return firstName + " " + lastName;
30          }
31
32          public int compareTo(Object o) {
33              Name n = (Name)o;
34              int lastCmp = lastName.compareTo(n.lastName);
35              return (lastCmp!=0 ? lastCmp :
36                      firstName.compareTo(n.firstName));
37          }
38
39
40          public static void main(String args[]) {
```

```
41              Name n[] = {
42                  new Name("Bond",    "James"),
43                  new Name("Jack",    "Blues"),
44                  new Name("Elwood",  "Blues"),
45                  new Name("You",     "Me")
46              };
47              List l = Arrays.asList(n);
48              Collections.sort(l);
49              System.out.println(l);
50          }
51      }
52
```

Source Code: Src/9/Name.java

See also

13.24. Filling a HashTable and using a reasonable hashfunction

- Object.hashCode contract: s1.equals(s2) implies that s1.hashCode()==s2.hash-Code() for any two sets s1 and s2,

```
public int hashCode() {
    return 31*super.firstName.hashCode() + super.lastName.hashCode();
}
```

```
 1      import java.util.*;
 2
 3      public class Hash_1 extends Name_1 {
 4          static final int MAX = 20000;
 5          static HashMap aHashMap = new HashMap();
 6
 7          public Hash_1(String firstName, String lastName) {
 8              super(firstName, lastName);
 9          }
10
11
12          public static void init()   {
13              long milliSeconds = System.currentTimeMillis();
14              for ( int index = 0; index <= MAX; index ++ ) {
15                      if ( index % 1000  == 0 )
16                              System.out.println(index + "/" + MAX );
17                      aHashMap.put( new Hash_1( "A" + index, "A" + index),
18                            new Hash_1( "A" + index, "A" + index)
19                           );
20              }
21              System.out.println("Time for filling: " +
22                      ( System.currentTimeMillis() - milliSeconds) );
23          }
24
25          public static void findIt(Hash_1 aHash_1)   {
26              long milliSeconds = System.currentTimeMillis();
27              if ( aHashMap.containsKey( aHash_1 ) )
28                      System.out.print("\taHashMap: containsKey takes: ");
```

```
29              System.out.println(System.currentTimeMillis() - milliSeconds);
30
31          }
32
33          public static void findMax()        {
34              Hash_1 aHash_1 = new Hash_1( "A" + MAX, "A" + MAX);
35              System.out.println("Find Max = " + aHash_1);
36              findIt(aHash_1);
37          }
38
39          public static void findMiddle()     {
40              Hash_1 aHash_1 = new Hash_1( "A" + ( MAX/2), "A" + ( MAX/2));
41              System.out.println("Find Middle = " + aHash_1);
42              findIt(aHash_1);
43          }
44
45          public static void findMin()        {
46              Hash_1 aHash_1 = new Hash_1( "A" + 0, "A" + 0);
47              System.out.println("Find Min = " + aHash_1);
48              findIt(aHash_1);
49          }
50
51
52          public static void main(String args[] )      {
53              long milliSeconds = System.currentTimeMillis();
54
55              init();
56              findMax();
57              findMiddle();
58              findMin();
59              System.exit(0);
60          }
61      }
62
```

Source Code: Src/9/Hash_1.java

```
Time for filling: 1638
Find Max = A20000 A20000
        aHashMap: containsKey takes: 0
Find Middle = A10000 A10000
        aHashMap: containsKey takes: 0
Find Min = A0 A0
        aHashMap: containsKey takes: 0
```

13.25. Filling a HashTable and not using a reasonable hashfunction

```
    public int hashCode() {
        return 1;
    }
```

```
 1      import java.util.*;
 2
 3      public class Hash_2 extends Name_2 {
 4          static final int MAX = 20000;
 5          static HashMap aHashMap = new HashMap();
 6
 7          public Hash_2(String firstName, String lastName) {
 8              super(firstName, lastName);
 9          }
10
11
12          public static void init()   {
13              long milliSeconds = System.currentTimeMillis();
14              for ( int index = 0; index <= MAX; index ++ ) {
15                      if ( index % 3000  == 0 )
16                              System.out.println(new Date() +
17                              ": "  + index + "/" + MAX );
18                      aHashMap.put( new Hash_2( "A" + index, "A" + index),
19                            new Hash_2( "A" + index, "A" + index)
20                           );
21              }
22              System.out.println("Time for filling: " +
23                      ( System.currentTimeMillis() - milliSeconds) );
24          }
25
26          public static void findIt(Hash_2 aHash_2)   {
27              long milliSeconds = System.currentTimeMillis();
28              if ( aHashMap.containsKey( aHash_2 ) )
29                      System.out.print("\taHashMap: containsKey takes: ");
30              System.out.println(System.currentTimeMillis() - milliSeconds);
31
32          }
33
34          public static void findMax()        {
35              Hash_2 aHash_2 = new Hash_2( "A" + MAX, "A" + MAX);
36              System.out.println("Find Max = " + aHash_2);
37              findIt(aHash_2);
38         }
39
40          public static void findMiddle()     {
41              Hash_2 aHash_2 = new Hash_2( "A" + ( MAX/2), "A" + ( MAX/2));
42              System.out.println("Find Middle = " + aHash_2);
43              findIt(aHash_2);
44         }
45
46          public static void findMin()        {
47              Hash_2 aHash_2 = new Hash_2( "A" + 0, "A" + 0);
48              System.out.println("Find Min = " + aHash_2);
49              findIt(aHash_2);
50         }
51
52
53         public static void main(String args[] )      {
54              long milliSeconds = System.currentTimeMillis();
```

```
55
56              init();
57              findMax();
58              findMiddle();
59              findMin();
60              System.exit(0);
61          }
62      }
63
```

Source Code: Src/9/Hash_2.java

```
% java Hash_2
Wed Sep 18 12:25:54 EDT 2002: 0/20000
Wed Sep 18 12:25:58 EDT 2002: 3000/20000
Wed Sep 18 12:26:08 EDT 2002: 6000/20000
Wed Sep 18 12:26:31 EDT 2002: 9000/20000
Wed Sep 18 12:27:11 EDT 2002: 12000/20000
Wed Sep 18 12:27:58 EDT 2002: 15000/20000
Wed Sep 18 12:29:01 EDT 2002: 18000/20000
Time for filling: 252173
Find Max = A20000 A20000
        aHashMap: containsKey takes: 1
Find Middle = A10000 A10000
        aHashMap: containsKey takes: 85
Find Min = A0 A0
        aHashMap: containsKey takes: 16
```

13.26. Routine Data Manipulation

The Collections class provides three algorithms for doing routine data manipulation on List objects.

- reverse: Reverses the order of the elements in a List.
- fill: Overwrites every element in a List with the specified value.
- copy: Takes two arguments, a destination List and a source List, and copies the elements of the source into the destination.
- See java doc ...

13.27. Sorting Maps

- A Map is not sorted
- Collections.sort():

```
public static void sort(List list)
```

- Convert Maps to Lists.

```
1       import java.util.*;
2
3       public class UseCollectionS     {
4           static ArrayList aArrayList = new ArrayList();
5           static HashMap aHashMap = new HashMap();
```

```
 6
 7         public static void main(String args[] )      {
 8
 9              for ( int index = 0; index < args.length; ++index)
10                      aHashMap.put(args[index], args[index] + " " + new Date
11
12              System.out.println("The HashMap: " + aHashMap);
13
14              List l = new ArrayList(aHashMap.values());
15              Collections.sort(l);
16              System.out.println("The List: " + l);
17
18         }
19      }
20
```

Source Code: Src/9/UseCollectionS.java

```
javac UseCollectionS.java && java UseCollectionS a b c d
The HashMap: {d=d Sun Oct 04 13:08:10 EDT 2010, b=b Sun Oct 04 13:08:10 EDT 20
The List: [a Sun Oct 04 13:08:09 EDT 2010, b Sun Oct 04 13:08:10 EDT 2010, c S
```

13.28. Shuffling

The shuffle algorithm does the opposite of what sort does:

- it destroys any trace of order that may have been present in a List.
- It's useful in implementing games of chance.
- It's useful for generating test cases.

13.29. Searching

The binarySearch algorithm searches for a specified element in a sorted List using the binary search algorithm.

```
 1      import java.util.*;
 2
 3      public class Name implements Comparable {
 4          protected String  firstName, lastName;
 5
 6          public Name(String firstName, String lastName) {
 7              if (firstName==null || lastName==null)
 8                  throw new NullPointerException();
 9              this.firstName = firstName;
10              this.lastName = lastName;
11          }
12
13          public String firstName()    {
14              return firstName;
15          }
16          public String lastName()     {
17              return lastName;
18          }
19
```

```
20          public boolean equals(Object o) {
21              if (!(o instanceof Name))
22                  return false;
23              Name n = (Name)o;
24              return n.firstName.equals(firstName) &&
25                     n.lastName.equals(lastName);
26          }
27
28          public String toString() {
29              return firstName + " " + lastName;
30          }
31
32          public int compareTo(Object o) {
33              Name n = (Name)o;
34              int lastCmp = lastName.compareTo(n.lastName);
35              return (lastCmp!=0 ? lastCmp :
36                      firstName.compareTo(n.firstName));
37          }
38
39
40          public static void main(String args[]) {
41              Name n[] = {
42                  new Name("Bond",    "James"),
43                  new Name("Jack",    "Blues"),
44                  new Name("Elwood",  "Blues"),
45                  new Name("You",     "Me")
46              };
47              List l = Arrays.asList(n);
48              Collections.sort(l);
49              System.out.println(l);
50          }
51      }
52
```

Source Code: Src/9/Name.java

```
 1      import java.util.*;
 2
 3      public class Name_1 extends Name {
 4          static final int MAX = 5;
 5
 6          public Name_1(String firstName, String lastName) {
 7              super(firstName, lastName);
 8          }
 9
10          public int hashCode() {
11              return 31*super.firstName.hashCode() + super.lastName.hashCode
12          }
13
14
15         public static void main(String args[] )      {
16              HashMap aHashMap = new HashMap();
17              long milliSeconds = System.currentTimeMillis();
18
```

```
19
20              milliSeconds = System.currentTimeMillis();
21              for ( int i = 1; i < MAX; i ++ )                {
22                      System.out.println("1: " + i );
23                      for ( int index = 0; index < 10000; index ++ )
24                              aHashMap.put( new Name_1( "A" + index, "A" + i
25                                    new Name_1( "B" + index, "A" + index)
26                                   );
27              }
28              System.out.println("Time Name_1: " + ( System.currentTimeMilli
29                                  milliSeconds )  );
30
31         }
32
33      }
34
```

Source Code: Src/9/Name_1.java

```
 1      import java.util.*;
 2
 3      public class SortTest extends Name_1 {
 4          static final int MAX = 20000;
 5          static HashMap aHashMap = new HashMap();
 6          static TreeMap aTreeMap = new TreeMap();
 7          static ArrayList aArrayList = new ArrayList();
 8
 9          public SortTest(String firstName, String lastName) {
10              super(firstName, lastName);
11          }
12
13
14          public static void init()   {
15              long milliSeconds = System.currentTimeMillis();
16              for ( int index = 0; index <= MAX; index ++ ) {
17                      if ( index % 1000  == 0 )
18                              System.out.println(index + "/" + MAX );
19                      aTreeMap.put( new SortTest( "A" + index, "A" + index),
20                            new SortTest( "A" + index, "A" + index)
21                           );
22                      aHashMap.put( new SortTest( "A" + index, "A" + index),
23                            new SortTest( "A" + index, "A" + index)
24                           );
25                      aArrayList.add( new SortTest( "A" + index, "A" + index
26              }
27              System.out.println("Time for filling: " +
28                      ( System.currentTimeMillis() - milliSeconds) );
29          }
30
31          public static void findIt(SortTest aSortTest)       {
32              long milliSeconds = System.currentTimeMillis();
33              if ( aArrayList.contains( aSortTest ) )
34                      System.out.print("\taArrayList: contains takes: ");
35              System.out.println( System.currentTimeMillis() - milliSeconds)
```

```
36
37              milliSeconds = System.currentTimeMillis();
38              if ( aArrayList.indexOf( aSortTest ) >= 0 )
39                      System.out.print("\taArrayList: indexOf takes: ");
40              System.out.println(System.currentTimeMillis() - milliSeconds);
41
42              milliSeconds = System.currentTimeMillis();
43              if ( aHashMap.containsKey( aSortTest ) )
44                      System.out.print("\taHashMap: containsKey takes: ");
45              System.out.println(System.currentTimeMillis() - milliSeconds);
46
47              milliSeconds = System.currentTimeMillis();
48              if ( aTreeMap.containsKey( aSortTest ) )
49                      System.out.print("\taTreeMap: containsKey takes: ");
50              System.out.println(System.currentTimeMillis() - milliSeconds);
51
52          }
53
54          public static void findMax()        {
55              SortTest aSortTest = new SortTest( "A" + MAX, "A" + MAX);
56              System.out.println("Find Max = " + aSortTest);
57              findIt(aSortTest);
58          }
59
60          public static void findMiddle()     {
61              SortTest aSortTest = new SortTest( "A" + ( MAX/2), "A" + ( MAX
62              System.out.println("Find Middle = " + aSortTest);
63              findIt(aSortTest);
64          }
65
66          public static void findMin()        {
67              SortTest aSortTest = new SortTest( "A" + 0, "A" + 0);
68              System.out.println("Find Min = " + aSortTest);
69              findIt(aSortTest);
70          }
71
72
73          public static void main(String args[] )      {
74              long milliSeconds = System.currentTimeMillis();
75
76              init();
77              findMax();
78              findMiddle();
79              findMin();
80              System.exit(0);
81          }
82
83      }
84
```

Source Code: Src/9/SortTest.java

```
0/20000
1000/20000
2000/20000
3000/20000
4000/20000
5000/20000
6000/20000
7000/20000
8000/20000
9000/20000
10000/20000
11000/20000
12000/20000
13000/20000
14000/20000
15000/20000
16000/20000
17000/20000
18000/20000
19000/20000
20000/20000
Time for filling: 128
Find Max = A20000 A20000
    aArrayList: contains takes: 5
    aArrayList: indexOf takes: 1
    aHashMap: containsKey takes: 0
    aTreeMap: containsKey takes: 0
Find Middle = A10000 A10000
    aArrayList: contains takes: 1
    aArrayList: indexOf takes: 0
    aHashMap: containsKey takes: 0
    aTreeMap: containsKey takes: 0
Find Min = A0 A0
    aArrayList: contains takes: 0
    aArrayList: indexOf takes: 0
    aHashMap: containsKey takes: 0
    aTreeMap: containsKey takes: 0
```

13.30. Finding Extreme Values

The min and max algorithms return, respectively, the minimum and maximum element contained in a specified Collection.

13.31. Comparable

How about comparing based on different criterias?

- See also
- The Comparable interface consists of a single method:

```
public interface Comparable {
      public int compareTo(Object o);
}
```

-

a < b	→ a.compareTo(b)	< 0
a == b	→ a.compareTo(b)	== 0
a > b	→ a.compareTo(b)	> 0

- Lists and arrays of objects that implement this interface can be sorted automatically by

  Collections.sort()

- A class's natural ordering is said to be consistent with equals if and only if

  (e1.compareTo((Object)e2)==0)

  has the same boolean value as

  e1.equals((Object)e2)

  for every e1 and e2 of class C.
- It is strongly recommended, but not strictly required that

  (x.compareTo(y)==0) == (x.equals(y))
- o = (name, age)
  - equals: name + age
  - compareTo: name

  -

  must ensure that

  - sgn(x.compareTo(y)) == -sgn(y.compareTo(x)) for all x and y.
  - (x.compareTo(y)>0 && y.compareTo(z)>0) implies x.compareTo(z)>0.
  - x.compareTo(y)==0 implies that sgn(x.compareTo(z)) == sgn(y.compareTo(z)), for all z.

13.32. Example I:

- Let's take a look at this example:

```
 1      /*
 2       * "Note: this class has a natural ordering
 3       *        that is inconsistent with equals.
 4       */
 5
 6      import java.util.*;
 7
 8      public class ComparableEx implements Comparable {
 9          protected String  firstName;
10          protected String  lastName;
11
12          public ComparableEx(String firstName, String lastName) {
13              this.firstName = firstName;
14              this.lastName  = lastName;
15          }
16
17          public boolean equals(Object o) {
18              if (!(o instanceof ComparableEx))
19                  return false;
20              ComparableEx n = (ComparableEx)o;
21              return firstName.equals(n.firstName)    &&
22                     lastName.equals(n.lastName);
23          }
24
25          public int compareTo(Object o) {
26              ComparableEx n = (ComparableEx)o;       // cast execption
27              return lastName.compareTo(lastName);
28          }
29          public String toString()    {
30                      return firstName + "/" + lastName;
31          }
32
33          public static void main(String args[]) {
34              ComparableEx n[] = {
35                  new ComparableEx("James",   "Bond"),
36                  new ComparableEx("James",   "Bond"),
37                  new ComparableEx("Jack",    "Blues"),
38                  new ComparableEx("Elwood",  "Blues")
39              };
40              List l = Arrays.asList(n);
41              Collections.sort(l);
42              System.out.println(l);
43          }
44      }
45
```

Source Code: Src/9/ComparableEx.java

Es wird ein Teil der Information des kompletten Objects verwendet.

13.33. Warning

What is the problem here?

Nur ein Nachname wird eingefuegt... doppelte Nachnamen werde ignoriert...

```
 1
 2      import java.util.*;
 3
 4      public class ComparatorExTree {
 5
 6          protected String  firstName;
 7          protected String  lastName;
 8
 9          static final Comparator nameC = new Comparator() {
10                  public int compare(Object o1, Object o2) {
11                      ComparatorExTree n1 = (ComparatorExTree)o1;     /
12                      ComparatorExTree n2 = (ComparatorExTree)o2;     /
13                      return n1.lastName.compareTo(n2.lastName);
14                  }
15          };
16
17          public ComparatorExTree(String firstName, String lastName) {
18              this.firstName = firstName;
19              this.lastName  = lastName;
20          }
21
22          public boolean equals(Object o) {
23              if (!(o instanceof ComparatorExTree))
24                  return false;
25              ComparatorExTree n = (ComparatorExTree)o;
26              return firstName.equals(n.firstName)    &&
27                     lastName.equals(n.lastName);
28          }
29
30         public String toString() {
31              return firstName + "; " + lastName;
32         }
33
34         public int compareTo(Object o) {
35              ComparableEx n = (ComparableEx)o;       // cast execption
36              if ( firstName.compareTo(firstName) == 0 )
37                      return lastName.compareTo(lastName);
38              else
39                      return  0;
40          }
41
42          public static void main(String args[]) {
43              ComparatorExTree n[] = {
44                  new ComparatorExTree("You",         "Name"),
45                  new ComparatorExTree("Roger",       "Bond"),
46                  new ComparatorExTree("James",       "Bond"),
47                  new ComparatorExTree("Jack",        "Blues"),
48                  new ComparatorExTree("Elwood",      "Blues")
49              };
50              TreeSet l = new TreeSet(nameC);
51
52              for ( int i = 0; i < n.length; i ++ )   {
53                      System.out.println(i + " " + n[i]);
```

```
54                      l.add(n[i]);
55              }
56              System.out.println("the TreeSet: " + l);
57          }
58      }
59
```

Source Code: Src/9/ComparatorExTree.java

```
java ComparatorExTree
0 You; Name
1 Roger; Bond
2 James; Bond
3 Jack; Blues
4 Elwood; Blues
the TreeSet: [Jack; Blues, Roger; Bond, You; Name]
```

13.34. Comparator

•

(From java doc)

The relation that defines the imposed ordering that a given comparator c imposes on a given set of objects S is:

For all x, y ∈ S:
{(x, y) such that c.compare(x, y) <= 0}.

The quotient for this total order is:
{(x, y) such that c.compare(x, y) == 0}.

13.35. Example

```
 1
 2      import java.util.*;
 3
 4      class HpComparator implements Comparator {
 5              public int compare(Object o1, Object o2)        {
 6                      String s1 = (String)o1;
 7                      String s2 = (String)o2;
 8                      return s1.compareTo(s2) ;
 9              }
10              public boolean equals(Object o) {
11                      return true;
12              }
13      }
14      public class HpC_C_ex   {
15        public static void sort(List aList) {
16              Object anArray[] = aList.toArray();
17
18              for (int index=0; index<anArray.length - 1; index++)     
19                  for (int walker=0; walker<anArray.length - index - 1;
20                      Comparable left = (Comparable) anArray[walker];
21                      Comparable right = (Comparable) anArray[walker+1]
22                      if ( left.compareTo( right ) > 0 )        {
```

```
23                              Object tmp = anArray[walker];
24                              anArray[walker] = anArray[walker + 1];
25                              anArray[walker+1] = tmp;
26                      }
27                  }
28              }
29              ListIterator anIterator  = aList.listIterator();
30              for (int j=0; j<anArray.length; j++) {
31                      anIterator.next();
32                      anIterator.set(anArray[j]);
33              }
34
35        }
36        public static void sort(List aList, Comparator aComparator) {
37              Object anArray[] = aList.toArray();
38
39              for (int index=0; index<anArray.length - 1; index++)     
40                  for (int walker=0; walker<anArray.length - index - 1;
41                      Object left = anArray[walker];
42                      Object right = anArray[walker+1];
43                      if ( aComparator.compare(left, right ) > 0 )
44                              Object tmp = anArray[walker];
45                              anArray[walker] = anArray[walker + 1];
46                              anArray[walker+1] = tmp;
47                      }
48                  }
49              }
50              ListIterator anIterator  = aList.listIterator();
51              for (int j=0; j<anArray.length; j++) {
52                      anIterator.next();
53                      anIterator.set(anArray[j]);
54              }
55
56        }
57
58        public static void main(String args[]) {
59              args = new String[4];
60
61              args[0] = "x"; args[1] = "b"; args[2] = "a"; args[3] = "d"
62              List l = Arrays.asList(args);
63              HpC_C_ex.sort(l);
64              System.out.println("1. " + l);
65
66              args[0] = "x"; args[1] = "b"; args[2] = "a"; args[3] = "d"
67              l = Arrays.asList(args);
68              HpC_C_ex.sort(l, new HpComparator()  );
69              System.out.println("2. " + l);
70
71        }
72      }
73
```

Source Code: Src/9/HpC_C_ex.java

13.36. Collections.sort()

```
 1      import java.util.*;
 2
 3      public class HpComparator {
 4
 5        public static void sort(List list ) {
 6              Object anArray[] = list.toArray();
 7                      // Arrays.sort(a, c);           /////// this is th
 8                      // http://www.cs.rit.edu/~hpb/Jdk5/api/java/util/L
 9
10              for (int index=0; index<anArray.length - 1; index++)     {
11                  for (int walker=0; walker<anArray.length - index - 1;
12                      Object left =  anArray[walker];
13                      Object right =  anArray[walker+1];
14                      if ( left.compareTo(right ) > 0 )        {
15                              Object tmp = anArray[walker];
16                              anArray[walker] = anArray[walker + 1];
17                              anArray[walker+1] = tmp;
18                      }
19                  }
20              }
21
22
23              ListIterator i = list.listIterator();
24              for (int j=0; j<anArray.length; j++) {
25                      i.next();
26                      i.set(anArray[j]);
27              }
28        }
29
30      }
31
```

Source Code: Src/9/HpComparator_1.java

```
 1      import java.util.*;
 2
 3      public class HpComparator {
 4
 5        public static void sort(List list, Comparator c) {
 6              Object anArray[] = list.toArray();
 7                      // Arrays.sort(a, c);           /////// this is th
 8                      // http://www.cs.rit.edu/~hpb/Jdk5/api/java/util/L
 9
10              for (int index=0; index<anArray.length - 1; index++)     {
11                  for (int walker=0; walker<anArray.length - index - 1;
12                      Object left =  anArray[walker];
13                      Object right =  anArray[walker+1];
14                      if ( c.compare(left,  right ) > 0 )        {
15                              Object tmp = anArray[walker];
16                              anArray[walker] = anArray[walker + 1];
17                              anArray[walker+1] = tmp;
```

```
18                      }
19                  }
20              }
21
22
23              ListIterator i = list.listIterator();
24              for (int j=0; j<anArray.length; j++) {
25                      i.next();
26                      i.set(anArray[j]);
27              }
28        }
29
30      }
31
```

Source Code: Src/9/HpComparator_3.java

Function pointer in C/C++

13.37. Comparator in separate Classes

The use:

```
 1      import java.util.*;
 2
 3      public class ComparatorExTreeClass {
 4          static HpbComparator theNth = new HpbComparator();
 5
 6          protected static int  soManyS;
 7          protected String  name;
 8          protected int     waitingListN;
 9
10          public ComparatorExTreeClass(String name) {
11              if (name==null)
12                  throw new NullPointerException();
13              this.name = name;
14              this.waitingListN = soManyS ++;
15          }
16
17          public ComparatorExTreeClass(String name, int waitingListN) {
18              this(name);
19              this.waitingListN = waitingListN;
20          }
21
22
23          public String getName()    {
24              return name;
25          }
26
27          public String toString() {
28              return name + " - " + waitingListN;
29          }
30
31          public static void main(String args[]) {
32              WaitingList n[] = {
```

```
33                  new WaitingList("Bond"),
34                  new WaitingList("Jack"),
35                  new WaitingList("Elwood"),
36                  new WaitingList("You", -1),
37                  new WaitingList("Me", -1)
38              };
39              TreeSet l = new TreeSet(theNth);
40
41              for ( int i = 0; i < n.length; i ++ )   {
42                      System.out.println(i + " " + n[i]);
43                      l.add(n[i]);
44              }
45              System.out.println("the TreeSet: " + l);
46          }
47      }
48
```

Source Code: Src/9/ComparatorExTreeClass.java

The comparator:

```
 1
 2      import java.util.*;
 3
 4      public class HpbComparator implements Comparator {
 5
 6          public int compare(Object o1, Object o2) {
 7
 8              if ( ( o1 instanceof WaitingList )  &&
 9                   ( o2 instanceof WaitingList )  )   {
10                     WaitingList n1 = (WaitingList)o1;
11                     WaitingList n2 = (WaitingList)o2;
12                     int nameCompareV = n1.name.compareTo(n2.name);
13                     return ( nameCompareV == 0 ?
14                             n1.waitingListN - n2.waitingListN :
15                             nameCompareV);;
16              } else
17                  return -1;
18          }
19      }
20
```

Source Code: Src/9/HpbComparator.java

Questions:

- Which classes do you need?
- Comparator, do you need them?
- which kind of collections do you need?

13.38. Exercise II

- What do you have to be aware of, if you write a compareTo method?
- What do you have to be aware of, if you write a compare method?

Under Construction

14. Networking

14.1. The Subject

Computer networks are the biggest *Big New Thing* for decades. They are now of central importance to all information technology. With the recent explosive growth of the internet, they are rapidly "she" becoming of crucial importance to all of modern society.

14.2. A Network Architecture Example: WWW

The World Wide Web is the *Big New Thing* in computer networking.

History:

> In 1989, Tim Berners Lee proposed a global hypertext project, to be known as the World Wide Web. Based on the earlier "Enquire" work, it was designed to allow people to work together by combining their knowledge in a web of hypertext documents. Tim Berners Lee wrote the first World Wide Web server and the first client, a wysiwyg hypertext browser/editor which ran in the NeXTStep environment. This work was started in October 1990, and the program "WorldWideWeb" was first made available within CERN in December, and on the Internet at large in the summer of 1991.
>
> Through 1991 and 1993, Tim Berners Lee continued working on the design of the Web, coordinating feedback from users across the Internet. His initial specifications of URIs, HTTP and HTML were refined and discussed in larger circles as the Web technology spread.

See also:

A browser, or viewer program is used to fetch and display "pages" of information from a server. A page is simply an ASCII text file, written using a simple markup language called Hypertext Meta Language (HTML). You may find an introduction

Uniform Resource Locators - URLs

The URL is the basis of the WWW. Think of a URL as an address that can lead you to any file on any machine anywhere in the world. Unlike the common postal address, however, these are written backwards. (Actually backwards makes more sense. My postal address was:

HP Bischof
3002 ST RT 48
Oswego, 13126 NY,
USA.

vs.

HP Bischof
Am Kaninchen Weg 12
8456 Laupheim

But if you want to deliver a letter to me, shouldn't you first go to the USA, then NY, then Oswego, then 3002 ST RT 48, then to HP Bischof? The URL is written in that more logical order.)

A URL defines the location of a WWW page in the following way:

service:host:port/file and resource details

For example:

http://www.cs.rit.edu:80/~hpb/CS3/all-2.2.html#section4
http://www.av.digital.com/cgi-bin/query?pg=q&what=web

URLs on the Web don't have to use the HTTP protocol. Some other URLs you might encounter are:

ftp file transfer protocol

news
for Usenet news groups

telnet
for telnet

mailto
to send email to a specific address

Connection Establishment

To fetch a WWW page, the browser application process running on your local computer first establishes a connection to the remote host.

What this means is that the browser process uses the facilities of the network connecting the two computers to send a "connection request" message to a server process running on the computer whose name was given in the URL.

If the remote server process is prepared to accept the connection, it responds with a "connection accepted" message.

Note that we are, for the moment, ignoring the process of "looking up" the remote host - discovering the network address associated with its domain name.

The HTTP Protocol

Once the two application processes have an established connection between them, they can communicate reliably.

The browser then sends a request, in ordinary plain text, to the server, thus:

```
GET /home.html
```

The string *GET something* is one of many commands defined in the Hypertext Transfer Protocol, HTTP. The server responds by returning the contents of a file.

Finally, the browser process interprets the HTML markup in the returned file, and displays it to the user.

14.3. What is the Internet

The Internet (short for inter networking, the practice of linking technologically different and independently operated networks), is a network of networks which allows users to communicate using electronic mail, to retrieve data stored in databases, and to access distant computers. The "core" of Internet includes the National Science Foundation's NSFNET, the Department of Energy's Energy Science Network (ESnet), the NASA Science Internet (NSI) as well as Defense's ARPANET and Terrestrial Wideband Network (TWBnet). Internet also includes a larger, and continually expanding, collection of interconnected regional, campus, and other networks throughout the U.S.
and overseas, as well as several networks that provide service on a for-profit basis.

These linked networks are independently operated; there is no central control of Internet. Internet began as an Advanced Research
Projects Agency research project to investigate computer networking technology. The networks that comprise the National Research and
Education Network (NREN), a component of the High Performance Computing and Communications Program, are a part of the current
Internet.

Copied from:

14.4. Protocol

(From: Douglas Comer defines a protocol as "a formal description of message formats and the rules two or more machines must follow to exchange those messages."

Protocols usually exist in two forms. First, they exist in a textual form for humans to understand. Second, they exist as programming code for computers to understand. Both forms should ultimately specify the precise interpretation of every bit of every message exchanged across a network.

Protocols exist at every point where logical program flow crosses between hosts. In other words, we need protocols every time we want to do something on another computer. Every time we want to print something on a network printer we need protocols. Every time we want to download a file we need protocols. Every time we want to save our work on disk, we don't need protocols - unless the disk is on a network file server.

Usually multiple protocols will be in use simultaneously. For one thing, computers usually do several things at once, and often for several people at once. Therefore, most protocols support multitasking. Also, one operation can involve several protocols. For example, consider the NFS (Network File System) protocol. A write to a file is done with an NFS operation, that uses another protocol (RPC) to perform a function call on a remote host, that uses another protocol (UDP) to deliver a datagram to a port on a remote host, that uses another protocol to deliver a datagram on an Ethernet, and so on. Along the way we made need to lookup host names (using the DNS protocol), convert data to a network standard form (using the XDR protocol), find a routing path to the host (using one or many of numerous protocols) - I think you get the idea.

14.5. Protocol Layers

Protocol layering is a common technique to simplify networking designs by dividing them into functional layers, and assigning protocols to perform each layer's task.

For example, it is common to separate the functions of data delivery and connection management into separate layers, and therefore separate protocols. Thus, one protocol is designed to perform data delivery, and another protocol, layered above the first, performs connection management. The data delivery protocol is fairly simple and knows nothing of connection management. The connection management protocol is also fairly simple, since it doesn't need to concern itself with data delivery.

Protocol layering produces simple protocols, each with a few well-defined tasks. These protocols can then be assembled into a useful whole. Individual protocols can also be removed or replaced.

The most important layered protocol designs are the Internet's original DoD model, and the OSI Seven Layer Model. The modern Internet represents a fusion of both models.

•

•

14.6. The OSI Seven-Layer Model

(From: An Internet Encyclopedia) In the 1980s, the European-dominated International Standards Organization (ISO), began to develop its Open Systems Interconnection (OSI) networking suite. OSI has two major components: an abstract model of networking (the Basic Reference Model, or — seven-layer model —), and a set of concrete protocols. The standard documents that describe OSI are for sale and not currently available online.

Parts of OSI have influenced Internet protocol development, but none more than the abstract model itself, documented in OSI 7498 and its various addenda. In this model, a networking system is divided into layers. Within each layer, one or more entities implement its functionality. Each entity interacts directly only with the layer immediately beneath it, and provides facilities for use by the layer above it. Protocols enable an entity in one host to interact with a corresponding entity at the same layer in a remote host.

```
[

boxht=0.5i boxwid=2i Application: box                    "Application"

Presentation: box with .n at last box .s          "Presentation"

Session: box with .n at last box .s          "Session"

Transport: box with .n at last box .s          "Transport"

Network: box with .n at last box .s          "Network"

DataLink: box with .n at last box .s          "Data Link"

Physiscal: box with .n at last box .s          "Physical"

]
```

The seven layers of the OSI Basic Reference Model are (from bottom to top):

- The Physical Layer describes the physical properties of the various communications media, as well as the electrical properties and interpretation of the exchanged signals. This layer defines the size of Ethernet coaxial cable, the type of BNC connector used, and the termination method.
- The Data Link Layer describes the logical organization of data bits transmitted on a particular medium. Ex: this layer defines the framing, addressing and checksumming of Ethernet packets.
- The Network Layer describes how a series of exchanges over various data links can deliver data between any two nodes in a network. Ex: this layer defines the addressing and routing structure of the Internet.
- The Transport Layer describes the quality and nature of the data delivery. Ex: this layer defines if and how retransmissions will be used to ensure data delivery.
- The Session Layer describes the organization of data sequences larger than the packets handled by lower layers. Ex: this layer describes how request and reply packets are paired in a remote procedure call.

- The Presentation Layer describes the syntax of data being transferred. Ex: this layer describes how floating point numbers can be exchanged between hosts with different math formats.
- The Application Layer describes how real work actually gets done. Ex: this layer would implement file system operations.

The original Internet protocol specifications defined a four-level model, and protocols designed around it (like TCP) have difficulty fitting neatly into the seven-layer model. Most newer designs use the seven-layer model.

14.7. TCP/IP

TCP/IP is the essential two-layer program that each Internet point-of-presence (POP) or SLIP/PPP user must use.

The *Transmission Control Protocol* (a protocol is a formal set of rules for communicating) manages the packaging of data into the packets that get routed on different paths over the Internet and reassembled at their destination.

The *Internet Protocol* handles the address part of each data packet so that it is routed to the right destination.

TCP/IP can be used on many data-link layers (can support many network hardware implementations).

These two protocols are the most important, TCP/IP is really a suite of protocols. (Some of these are viewed as alternative protocols and others as application protocols.) The ones you are most likely to use (directly or indirectly) are: HTTP, FTP, Telnet, Gopher, PPP, and SMTP.

Related protocols, some of them may be included in a TCP/IP package:

- User Datagram Protocol (UDP)
- Telnet
- File Transfer Protocol (FTP)
- Trivial File Transfer Protocol (TFTP)
- Simple Mail Transfer Protocol (SMTP)
- Gopher protocol
- Hypertext Transport Protocol (HTTP)

14.8. TCP/IP Layer

TCP/IP is normally considered to be a 4 layer system:

.so Pic/16/tcp_ip.pic

Link layer:

the network interface layer. Includes normally the device driver in an OS.

Network layer:

The network layer handles the movement of the packets around a network. Routing of packets takes place here. IP (Internet Protocol), ICMP (Internet Control Message Protocol), and IGMP (Internet Group Management Protocol) provide the network layer in the TCP/IP suite.

Transport Layer:

Provides a flow of data between two hosts for the application layer. There are two different protocols: TCP (Transmission Control Protocol) and UDP (User Datagram Protocol).

The function of the TCP protocol, is to provide a:

reliable	all data is delivered correctly
connection-oriented	the protocol provides procedures for establishing interprocess connections.
byte stream	ie, no visible packetisation so far as the application processes are concerned
end-to-end	

... interprocess communication service.

The User Datagram Protocol provides a connectionless alternative transport service to TCP for applications where reliable stream service is not needed. UDP datagrams can be dropped, duplicated or delivered out of order, same as for IP.

Application Layer:

The application layer handles the details of the particalur application.

- telnet for remote login
- ftp the file transfer protocol
- SMTP simple mail transfer protocol
- SNMP simple network management protocol

An example:

.so Pic/16/use_tcp_ip.pic

14.9. Internet Addresses

Every Internet-connected system has a unique Internet host address.

This is a 32 bit, or 4 byte, binary number.

Internet addresses are written as a dotted sequence of the form:

a.b.c.d

where *a, b, c* and *d* etc, are the decimal values (ranging from 0 to 255) of the 4 bytes which make up the internet address, for example:

129.21.36.56

129.21.36.56 is the IP address of ilon,or to use its full name

spiegel.cs.rit.edu

We will later see how the name of a computer is mapped to its IP-address.

14.10. IP Address Classes

```
[

boxht=0.5i boxwid=0.19i y_dist=0.8i line_ht=0.5 bit_border=0.1

define bits Y
  for i = 1 to 32 do X          box with .w at last box.e          if  ( i % 8 == 0 ) then Z
                line from last box.ne + ( 0, bit_border) to                                  last
box.se - ( 0, bit_border)          Z
  X Y

define tags Y        # Class_box first second text
  line from $1.ne + ( $2 * boxwid, 0 ) to                $1.ne + ( $2 * boxwid, line_ht
)
  line from $1.ne + ( $3 * boxwid, 0 ) to                $1.ne + ( $3 * boxwid, line_ht
)
  box invis with .c at  $1.ne + ( ( ( $3-$2 )  / 2 + $2 ) * boxwid, boxht / 2 ) $4 Y

A: box invis wid 1i "Class A"          bits(A) A_1: box with .w at A.e + ( 0 *
boxwid, 0 ) "0"          tags(A, 1,   8,    "7 bits — netid")          tags(A, 8,   32,   "24
bits — hostid")

B: box invis wid 1i "Class B" with .n at A.s - ( 0, y_dist )          bits(B) B_1: box
with .w at B.e + ( 0 * boxwid, 0 ) "1" B_1: box with .w at B.e + ( 1 * boxwid, 0 )
"0"          tags(B, 2,  16,    "14 bits — netid")          tags(B,16,  32,    "16 bits —
hostid")

C: box invis wid 1i "Class C" with .n at B.s - ( 0, y_dist )          bits(C) C_1: box
with .w at C.e + ( 0 * boxwid, 0 ) "1" C_1: box with .w at C.e + ( 1 * boxwid, 0 )
"1" C_1: box with .w at C.e + ( 2 * boxwid, 0 ) "0"          tags(C, 3,  24,    "21 bits
— netid")          tags(C,24,  32,    "8 bits — hostid")

D: box invis wid 1i "Class D" with .n at C.s - ( 0, y_dist )          bits(D) D_1: box
with .w at D.e + ( 0 * boxwid, 0 ) "1" D_1: box with .w at D.e + ( 1 * boxwid, 0 )
"1" D_1: box with .w at D.e + ( 2 * boxwid, 0 ) "1" D_1: box with .w at D.e + ( 3
* boxwid, 0 ) "0"          tags(D, 4,  32,    "28 bits — multicast group id")

E: box invis wid 1i "Class E" with .n at D.s - ( 0, y_dist )          bits(E) E_1: box
with .w at E.e + ( 0 * boxwid, 0 ) "1" E_1: box with .w at E.e + ( 1 * boxwid, 0 )
"1" E_1: box with .w at E.e + ( 2 * boxwid, 0 ) "1" E_1: box with .w at E.e + ( 3 *
boxwid, 0 ) "1" E_1: box with .w at E.e + ( 4 * boxwid, 0 ) "0"          tags(E, 5,
32,    "27 bits — reserved for future use")

]
```

14.11. Ethernet Address

An IP-address only makes sense to the TCP/IP suite. A data link such as an Ethernet or a token ring has its own addressing scheme (often 48 bits).

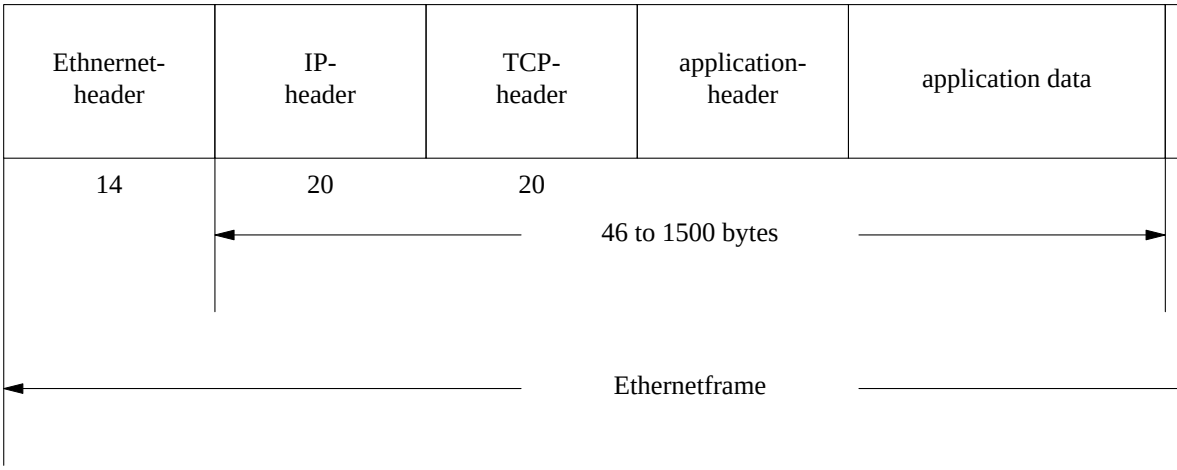
The-byte address is often used, which is divided into a 3-byte vendor ID and a 3-byte vendor-defined field. Ethernet manufacturers are assigned a unique vendor ID, and are then responsible for insuring that all of their devices have unique addresses in the last 3 bytes.

A network, such as an Ethernet can be used by different network layers at the same time. See also

14.12. Encapsulation

When an application sends data using TCP is sent down the protocol stack.

A physical property of an Ethernet frame is, that the size of its data must be between 46 and 1500 bytes.



14.13. TCP Ports

TCP is using protocol port numbers to identify the ultimate destination.

How does one determine the port to communicate with?

- Well known ports
- Randomly assigned ports.

See

14.14. Socket

Two popular Application Programming Interface using TCP/IP protocols are called sockets and TLI (transport layer interface).

A socket is one end-point of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program. The java.net package provides two classes--Socket and ServerSocket--that implement the client side of the connection and the server side of the connection, respectively.

A socket is a network communications endpoint.

A socket is an object from which messages are sent and received.

Socket operations resemble file operations in many respects:

- Data transfer operations on sockets work just like read and write operations on files.
- A socket is closed, just like a file, when communications is finished.

See also:

14.15. java.net

Through the classes in java.net, Java programs can use TCP or UDP to communicate over the Internet. The URL, URLConnection, Socket, and ServerSocket classes all use TCP to communicate over the network. The DatagramPacket, DatagramSocket, and MulticastSocket classes are for use with UDP.

14.16. Getting Information

Host info

```
 1      import java.net.*;
 2      import java.io.*;
 3      import java.util.*;
 4      public class HostInfo {
 5          public static void main(String argv[]) {
 6              InetAddress ipAddr;
 7              try {
 8                      ipAddr = InetAddress.getLocalHost();
 9                      System.out.println ("This is "+ipAddr);
10              } catch (UnknownHostException e) {
11                      System.out.println ("Unknown host");
12              }
13          }
14      }
```

Source Code: Src/16/HostInfo.java

```
% java HostInfo
This is spiegel.cs.rit.edu/129.21.36.56
```

14.17. Makefile for the Execution of the following Examples

```
 1      dayTimeRun:
 2              java DayTimeServer      -port 3456 &
 3              sleep 1
 4              java DayTime            -port 3456
 5              kill `ps -t \`tty\` | grep DayTimeServer | sed 's/ .*//'`
 6              echo $(pid)
 7
 8      echoRun:
 9              java EchoServer -port 4567 &
10              sleep 1
11              java HpEchoSocketTest -port 4567
12              kill `ps -t \`tty\` | grep EchoServer | sed 's/ .*//'`
13              echo $(pid)
```

```
14
15      mtsRun:
16              java MTS -port 63782 &
17              java MTSclient -port 63782; java MTSclient -port 63782
18              kill `ps -t \`tty\` | grep MTS | sed 's/ .*//'`
19              echo $(pid)
```

Source Code: Src/16/makefile

14.18. Daytime Client

see */etc/services*.

```
 1      import java.net.*;
 2      import java.io.*;
 3      import java.util.*;
 4      public class DayTime {
 5
 6              String hostName = "spiegel.cs.rit.edu";
 7          int    port = 13;
 8
 9          private void printMessage() {
10              System.out.println("-h          ---->   help");
11              System.out.println("[-host              hostName]");
12              System.out.println("[-port              port]");
13          }
14
15          /**
16            * Parse the commandlind arguments and sets variables.
17            */
18          public void parseArgs(String args[]) {
19
20              for (int i = 0; i < args.length; i ++) {
21                      if (args[i].equals("-h"))
22                              printMessage();
23                      else if (args[i].equals("-host"))
24                              hostName = args[++i];
25                      else if (args[i].equals("-port"))
26                              port = new Integer(args[++i]).intValue();
27              }
28          }
29
30          public void doTheJob()       {
31              try {
32                      System.out.println("host: " +  hostName );
33                      System.out.println("port: " +  port );
34                      Socket aReadSocket = new Socket(hostName, port);
35
36                      System.out.println("aReadSocket = " + aReadSocket
37                      BufferedReader readFrom = new BufferedReader (
38                              new InputStreamReader (aReadSocket.getInpu
39                      String rTime = readFrom.readLine ();
40                      System.out.println (rTime);
41                      aReadSocket.close();
```

```
42              } catch (Exception e) {
43                      System.out.println (e);
44              }
45         }
46
47         public static void main(String argv[]) {
48              DayTime aDayTime = new DayTime();
49              aDayTime.parseArgs(argv);
50              aDayTime.doTheJob();
51
52         }
53      }
```

Source Code: Src/16/DayTime.java

```
% java DayTime
host: spiegel.cs.rit.edu
port: 13
java.net.ConnectException: Operation timed out
```

14.19. Daytime Server

```
 1      import java.net.*;
 2      import java.io.*;
 3      import java.util.*;
 4      // java DayTimeServer -port 12345
 5      public class DayTimeServer extends Thread {
 6
 7         ServerSocket         aServerSocket;
 8         int                  port     = 4242;
 9
10         public DayTimeServer()       {
11         }
12
13         public DayTimeServer(int port)       {
14              try {
15                  aServerSocket = new ServerSocket(port);
16                  System.out.println ("Listening on port: " + aServerSoc
17              } catch(Exception e) {
18                  System.out.println(e);
19              }
20         }
21
22          private void printMessage() {
23              System.out.println("-h          ---->   help");
24              System.out.println(" -port              port");
25              System.out.println(" {-port             port}");
26              System.out.println("or ");
27              System.out.println(" no argument");
28         }
29
30         /**
31           * Parse the commandlind arguments and sets variables.
```

```
32           */
33         private  void parseArgs(String args[]) {
34
35              for (int i = 0; i < args.length; i ++) {
36                      if (args[i].equals("-h"))
37                              printMessage();
38                      else if (args[i].equals("-port")) {
39                              port = new Integer(args[++i]).intValue();
40                              new DayTimeServer(port).start();
41                      }
42              }
43         }
44
45         public void run()    {
46              try {
47                  for(;;) {
48                      Socket connectionToClientSocket = aServerSocket.ac
49                      System.out.println(connectionToClientSocket.toStri
50                      PrintWriter out = new PrintWriter
51                          (connectionToClientSocket.getOutputStream (),
52                      out.println("It is now: " + new Date());
53                      connectionToClientSocket.close();
54                  }
55              } catch(Exception e) {
56                  System.out.println(e);
57                  e.printStackTrace();
58              }
59         }
60
61          public static void main(String argv[]) {
62              if ( argv.length == 0 )
63                      new DayTimeServer(0).start();
64              else
65                      new DayTimeServer().parseArgs(argv);
66          }
67      }
```

Source Code: Src/16/DayTimeServer.java

```
% java DayTimeServer &
Listening on port: 63248
Socket[addr=/129.21.36.56,port=63249,localport=63248]
% java DayTime  -port 63248
host: spiegel.cs.rit.edu
port: 63248
aReadSocket = Socket[addr=spiegel.cs.rit.edu/129.21.36.56,port=63248,loca
It is now: Wed Nov 14 12:57:39 EST 2018
```

14.20. Reading from and Writing to a Socket

```
 1      import java.io.*;
 2      import java.net.*;
 3
 4      class HpEchoSocketTest {
 5
 6             Socket aSocket;
 7             String hostName;
 8             int port;
 9             PrintWriter outPutStream = null;
10             BufferedReader inPutStream = null;
11
12             public HpEchoSocketTest()        {
13             }
14             public HpEchoSocketTest(String name, int port) {
15                    hostName = name;
16                    this.port = port;
17             }
18
19             public void parseArgs(String args[]) {
20
21                  for (int i = 0; i < args.length; i ++) {
22                         if (args[i].equals("-host"))
23                                hostName = args[++i];
24                         else if (args[i].equals("-port"))
25                                port = new Integer(args[++i]).intValue
26                  }
27             }
28
29             public void createIOconections() throws Exception {
30                    try {
31                           aSocket = new Socket(hostName, port);
32                           outPutStream = new PrintWriter( aSocket.getOu
33                           inPutStream = new BufferedReader( new InputSt
34                    } catch (Exception e )       {
35                           System.out.println(e.toString());
36                           System.exit(1);
37                    }
38             }
39             public void closeIOconections() throws Exception {
40                    try {
41                           inPutStream.close();
42                           outPutStream.close();
43                    } catch (Exception e )       {
44                           System.out.println(e.toString());
45                           System.exit(1);
46                    }
47             }
48             public void readAndPrint() throws Exception {
49                    InputStream ins;
50                    OutputStream os;
51
52                    BufferedReader stdIn = new BufferedReader(
53                                        new InputStreamReader(System.in
54                    String userInput;
```

```
55
56                   while ((userInput = stdIn.readLine()) != null) {
57                            outPutStream.println(userInput);
58                            System.out.println("echo: " + inPutStream
59                   }
60                   stdIn.close();
61            }
62
63              public void doTheJob()       {
64                 try {
65                        System.out.println("host: " +  hostName );
66                        System.out.println("port: " +  port );
67                        HpEchoSocketTest aHpEchoSocketTest = new HpEcl
68                        createIOconections();
69                        readAndPrint();
70                        closeIOconections();
71                 } catch (Exception e) {
72                        System.out.println (e);
73                 }
74            }
75
76            public static void main(String[] args) {
77                   HpEchoSocketTest st;
78                   String host = "spiegel.cs.rit.edu";
79                   int    port = 12345;
80                   HpEchoSocketTest aHpEchoSocketTest = new HpEchoSocke
81                   aHpEchoSocketTest.parseArgs(args);
82                   aHpEchoSocketTest.doTheJob();
83            }
84
85      }
```

Source Code: Src/16/HpEchoSocketTest.java

```
 1      import java.net.*;
 2      import java.io.*;
 3      import java.util.*;
 4      // java EchoServer -port 12345
 5      public class EchoServer extends Thread {
 6
 7         ServerSocket         aServerSocket;
 8         int                  port     = 4242;
 9
10         public EchoServer()  {
11         }
12
13         public EchoServer(int port)  {
14              try {
15                  aServerSocket = new ServerSocket(port);
16                  System.out.println ("Listening on port: " + aServerSoc
17              } catch(Exception e) {
18                  System.out.println(e);
19              }
20         }
```

```
21
22          private void printMessage() {
23              System.out.println("-h          ---->   help");
24              System.out.println(" -port              port");
25              System.out.println(" {-port             port}");
26              System.out.println("or ");
27              System.out.println(" no argument");
28          }
29
30          /**
31            * Parse the commandlind arguments and sets variables.
32            */
33          private  void parseArgs(String args[]) {
34
35              for (int i = 0; i < args.length; i ++) {
36                      if (args[i].equals("-h"))
37                              printMessage();
38                      else if (args[i].equals("-port")) {
39                              port = new Integer(args[++i]).intValue();
40                              new EchoServer(port).start();
41                      }
42              }
43          }
44
45          public void run()    {
46              try {
47                  for(;;) {
48                      Socket clientSocket = aServerSocket.accept();
49                      System.out.println(clientSocket.toString());
50                      PrintWriter out = new PrintWriter (clientSocket.ge
51                      BufferedReader in = new BufferedReader( new InputS
52                      while ( true )  {
53                              String input = in.readLine();
54                              if ( input == null ) System.exit(0);
55                              System.out.println("sending back: " + inpu
56                              out.println("back: " + input );
57                      }
58                      // clientSocket.close();
59                  }
60              } catch(Exception e) {
61                  System.out.println(e);
62                  e.printStackTrace();
63              }
64          }
65
66           public static void main(String argv[]) {
67              if ( argv.length == 0 )
68                      new EchoServer(12345).start();
69              else
70                      new EchoServer().parseArgs(argv);
71           }
72      }
```

Source Code: Src/16/EchoServer.java

```
% java  EchoServer
Listening on port: 12345
% java HpEchoSocketTest -port 12345
host: null
port: 12345
jkjkj
echo: back: jkjkj
lkklk
echo: back: lkklk
```

These are the typical steps:

1. Open a socket.
2. Open an input stream and output stream to the socket.
3. Read from and write to the stream according to the server's protocol.
4. Close the streams.
5. Close the socket.

14.21. Multi Client Server and Client

- telnet allows many clients to connect

14.22. MTS Server:

```
 1      // java MTS
 2
 3      import java.net.*;
 4      import java.io.*;
 5      import java.util.*;
 6
 7      public class MTS extends Thread {
 8
 9         ServerSocket         listen;
10         static String        hostName = "spiegel.cs.rit.edu";
11         int                  port     = 4242;
12         int                  id       = 0;
13
14         public MTS() {
15              listen = null;
16         }
17
18         public MTS(int port) {
19              try {
20                  listen = new ServerSocket(port);
21                  System.out.println ("Listening on port: " + getLocalPo
22              } catch(Exception e) {
23                  System.out.println(e);
24              }
25         }
26         public MTS(int port, int id) {
27              this(port);
```

```
28              this.id = id;
29          }
30
31          public int getLocalPort ()   {
32              return listen.getLocalPort();
33          }
34
35           private void printMessage() {
36              System.out.println("-h          ---->   help");
37              System.out.println("[-host              hostName");
38              System.out.println(" -port              port");
39              System.out.println(" {-port             port}");
40              System.out.println("or ");
41              System.out.println(" no argument");
42          }
43
44          /**
45            * Parse the commandlind arguments and sets variables.
46            */
47          public void parseArgs(String args[]) {
48
49              for (int i = 0; i < args.length; i ++) {
50                      if (args[i].equals("-h"))
51                              printMessage();
52                      else if (args[i].equals("-host"))
53                              hostName = args[++i];
54                      else if (args[i].equals("-port")) {
55                              port = new Integer(args[++i]).intValue();
56                              new MTS(port).listenToPort();
57                      }
58              }
59          }
60
61          public void run()    {
62              try {
63                      System.out.println("Wating for client to connect "
64                      Socket clientConnection = listen.accept();
65                      System.out.println(clientConnection.toString());
66                      PrintWriter out = new PrintWriter (clientConnectio
67                      out.println("It is now: " + new Date());
68                      System.out.println(id + " .... falling asleep");
69                      sleep(1000);
70                      System.out.println("\t" + id + " .... wake up");
71                      listen.close();
72              } catch(Exception e) {
73                  System.out.println(e);
74                  e.printStackTrace();
75              }
76          }
77
78          public void listenToPort()   {
79              try {
80                  int id = 0;
81                  for(;;) {
```

```
82                      System.out.println("Wating for client to connect "
83                      Socket clientConnection = listen.accept();
84                      System.out.println("Somebody connected ... ");
85                      MTS aServer = new MTS(0, id++);
86                      aServer.start();
87                      System.out.println("offer ... " + aServer.getLoca
88                      PrintWriter out = new PrintWriter (clientConnectio
89                      out.println(aServer.getLocalPort());
90                      clientConnection.close();
91                  }
92              } catch(Exception e) {
93                  System.out.println(e);
94                  e.printStackTrace();
95              }
96          }
97
98           public static void main(String argv[]) {
99              if ( argv.length == 0 )
00                      new MTS(0).listenToPort();
01              else
02                      new MTS().parseArgs(argv);
03           }
04      }
```

Source Code: Src/16/MTS.java

14.23. MTS Client:

```
 1      // java MTSclient -host spiegel -port 50405
 2      import java.net.*;
 3      import java.io.*;
 4      import java.util.*;
 5      public class MTSclient {
 6
 7          String hostName = "spiegel.cs.rit.edu";
 8          int    port = 63782;
 9
10          private void printMessage() {
11              System.out.println("-h          ---->   help");
12              System.out.println("[-host              hostName]");
13              System.out.println("[-port              port]");
14          }
15
16          /**
17            * Parse the commandlind arguments and sets variables.
18            */
19          public void parseArgs(String args[]) {
20
21              for (int i = 0; i < args.length; i ++) {
22                      if (args[i].equals("-h"))
23                              printMessage();
24                      else if (args[i].equals("-host"))
25                              hostName = args[++i];
```

```
26                      else if (args[i].equals("-port"))
27                              port = new Integer(args[++i]).intValue();
28              }
29          }
30
31          public void doTheJob()       {
32              try {
33                      Socket socket = new Socket(hostName, port);
34
35                      BufferedReader dataStreamIn = new BufferedReader
36                              new InputStreamReader (socket.getInputStre
37                      String newPort = dataStreamIn.readLine ();
38                      System.out.println ("Use from now in port: " + new
39                      socket.close();
40                      dataStreamIn.close();
41                      socket = new Socket(hostName, new Integer(newPort
42                      dataStreamIn = new BufferedReader ( new InputStrea
43                      System.out.println("got: " + dataStreamIn.readLine
44              } catch (Exception e) {
45                      System.out.println (e);
46              }
47          }
48
49          public static void main(String argv[]) {
50              MTSclient aMTSclient = new MTSclient();
51              aMTSclient.parseArgs(argv);
52              aMTSclient.doTheJob();
53
54          }
55      }
```

Source Code: Src/16/MTSclient.java

- Result:

```
# Window 1:
Listening on port: 53660
Wating for client to connect ServerSocket[addr=0.0.0.0/0.0.0.0,localp
Somebody connected ...
Listening on port: 53665
offer ... 53665
Wating for client to connect ServerSocket[addr=0.0.0.0/0.0.0.0,localp
Wating for client to connect ServerSocket[addr=0.0.0.0/0.0.0.0,localp
Socket[addr=/129.21.36.56,port=53666,localport=53665]
0 .... falling asleep
    0 .... wake up
Somebody connected ...
Listening on port: 53674
offer ... 53674
Wating for client to connect ServerSocket[addr=0.0.0.0/0.0.0.0,localp
Wating for client to connect ServerSocket[addr=0.0.0.0/0.0.0.0,localp
Socket[addr=/129.21.36.56,port=53675,localport=53674]
1 .... falling asleep
    1 .... wake up
```

```
# Window 2:
% java MTSclient -host spiegel.cs.rit.edu -port 53660
Use from now in port: 53665
got: It is now: Mon Oct 31 16:02:53 EDT 2016Ep
```

14.24. Connection to an URL

Class URL represents a Uniform Resource Locator, a pointer to a "resource" on the World Wide Web. A resource can be something as simple as a file or a directory, or it can be a reference to a more complicated object, such as a query to a database or to a search engine. More information on the types of URLs and their formats can be found at:

```
 1      import java.io.*;
 2      import java.net.URL;
 3      import java.net.MalformedURLException;
 4
 5      public class Url_Read {
 6
 7        public static void readFromUrl(String theUrl) {
 8
 9        URL aUrl = null;
10        BufferedReader in = null;
11        String line;
12
13        try {
14                aUrl = new URL(theUrl);
15                System.out.println("getPort() " + aUrl.getPort());
16                System.out.println("getHost() " + aUrl.getHost());
17                System.out.println("getProtocol() " + aUrl.getProtocol(
18                System.out.println("getFile() " + aUrl.getFile());
19                System.out.println("getRef() " + aUrl.getRef());
20
21                in = new BufferedReader(
22                         new InputStreamReader( aUrl.openStream() ) );
23
24                while ( ( line = in.readLine() ) != null ) {
25                      System.out.println(line);
26                }
27
28                in.close();
29
30        } catch (MalformedURLException e) {
31                System.err.println("Something is wrong with this " +
32                      theUrl +  ".");
33                System.exit(1);
34        } catch (IOException e) {
35                System.err.println("Couldn't get I/O for the connection
36                      + theUrl );
37                System.exit(1);
38        }
39
40        }
41
```

```
42        public static void main( String args[] ) {
43
44          if ( args.length != 1 )     {
45              System.err.println(
46                   "Usage: java Url_Read url");
47              System.exit(1);
48          }
49
50          try {
51              readFromUrl(args[0]);
52
53          }
54          catch ( NumberFormatException e)    {
55              System.out.println(args[0] + " is not a number ;-(");
56          }
57
58        }
59      }
```

Source Code: Src/16/Url_Read.java

```
% java Url_Read http://www.cs.rit.edu/~hpb | sed 15q
getPort() -1
getHost() www.cs.rit.edu
getProtocol() http
getFile() /~hpb
getRef() null
<HTML>

<HEAD>
<title>Hans-Peter Bischof's Home Page</title>
</HEAD>


<FRAMESET cols="230,*">
  <frame name="toc"    TARGET="_main" src="toc.html"     scrolling="auto">
  <frame name="intro"                 src="intro.html"   scrolling="auto">
```

14.25. Datagram Socket

- A datagram socket is the sending or receiving point for a packet delivery service.
- Each packet sent or received on a datagram socket is individually addressed and routed.
- Multiple packets sent from one machine to another may be routed differently, and may arrive in any order.
- UDP broadcasts sends and receives are always enabled on a DatagramSocket.

An example:

14.26. Datagram Server:

```
 1      import java.net.*;
 2      import java.io.*;
 3      import java.util.*;
 4
 5      public class DayTimeUDPServer extends Thread {
 6
 7         DatagramSocket       socket;
 8         static String        hostName = "spiegel";
 9         int                  port     = 1313;
10
11
12         public DayTimeUDPServer()    {
13         }
14
15         public DayTimeUDPServer(int port)    {
16              try {
17                  socket = new DatagramSocket(port);
18                  System.out.println ("Listening on port: "
19                      + socket.getLocalPort());
20              } catch(Exception e) {
21                  System.out.println(e);
22              }
23         }
24
25          private void printMessage() {
26              System.out.println("-h          ---->   help");
27              System.out.println("[-host              hostName");
28              System.out.println(" -port              port");
29              System.out.println(" {-port             port}");
30              System.out.println("or ");
31              System.out.println(" no argument");
32         }
33
34         /**
35           * Parse the commandlind arguments and sets variables.
36           */
37         public void parseArgs(String args[]) {
38
```

```
39              for (int i = 0; i < args.length; i ++) {
40                      if (args[i].equals("-h"))
41                              printMessage();
42                      else if (args[i].equals("-host"))
43                              hostName = args[++i];
44                      else if (args[i].equals("-port")) {
45                              port = new Integer(args[++i]).intValue();
46                              new DayTimeUDPServer(port).start();
47                      }
48              }
49          }
50
51          public void run()    {
52              byte[] buf = new byte[256];
53              try {
54                  for(;;) {
55                      String sendThis = "es schlaegt: " + new Date();
56                      DatagramPacket packet = new DatagramPacket(buf, bu
57                      socket.receive(packet);
58                      InetAddress address = packet.getAddress();
59                      int port = packet.getPort();
60                      buf = sendThis.getBytes();
61                      packet = new DatagramPacket(buf, buf.length, addre
62                      System.out.println("Sending to port: " + port );
63                      System.out.println("Sending    data: " + new Strin
64                      socket.send(packet);
65                  }
66              } catch(Exception e) {
67                  System.out.println(e);
68                  e.printStackTrace();
69              }
70          }
71
72           public static void main(String argv[]) {
73              if ( argv.length == 0 )
74                      new DayTimeUDPServer(0).start();
75              else
76                      new DayTimeUDPServer().parseArgs(argv);
77           }
78      }
```

Source Code: Src/16/DayTimeUDPServer.java

Datagram Client:

```
1       import java.net.*;
2       import java.io.*;
3       import java.util.*;
4       public class DayTimeUDP {
5
6           String hostName = "spiegel";
7           int    port = 1313;
8
9           private void printMessage() {
```

```
10              System.out.println("-h          ---->   help");
11              System.out.println("[-host              hostName]");
12              System.out.println("[-port              port]");
13          }
14
15          /**
16            * Parse the commandlind arguments and sets variables.
17            */
18          public void parseArgs(String args[]) {
19
20              for (int i = 0; i < args.length; i ++) {
21                      if (args[i].equals("-h"))
22                              printMessage();
23                      else if (args[i].equals("-host"))
24                              hostName = args[++i];
25                      else if (args[i].equals("-port"))
26                              port = new Integer(args[++i]).intValue();
27              }
28          }
29
30          public void doTheJob()       {
31              try {
32                      byte buf[] = new byte[64];
33                      InetAddress aInetAddress = InetAddress.getByName(h
34                      DatagramPacket dp = new DatagramPacket(buf, buf.le
35                      DatagramSocket socket = new DatagramSocket();
36                      DatagramPacket packet = new DatagramPacket(buf,
37                              buf.length, aInetAddress, port);
38                      socket.send(packet);
39
40                      System.out.println("host: " +  hostName );
41                      System.out.println("port: " +  port );
42                      System.out.println("after creation");
43                      socket.receive(dp);
44                      System.out.println("received: -" +
45                              new String(dp.getData() ) + "-"  );
46                      socket.close();
47              } catch (Exception e) {
48                      System.out.println (e);
49                      e.printStackTrace();
50              }
51          }
52
53          public static void main(String argv[]) {
54              DayTimeUDP aDayTimeUDP = new DayTimeUDP();
55              aDayTimeUDP.parseArgs(argv);
56              aDayTimeUDP.doTheJob();
57
58          }
59      }
```

Source Code: Src/16/DayTimeUDP.java

- Execution:

```
# Window 1:
% java DayTimeUDPServer -port 53818
Listening on port: 53818
Sending    data: es schlaegt: Wed Nov 02 10:55:56 EDT 2016
Sending to port: 53840
Fri Oct 28 11:32:14 EDT 2016

# Window 2:
% java DayTimeUDP -port 53818
host: yps
port: 53818
after creation
received: -es schlaegt: Wed Nov 02 10:55:56 EDT 2016-
```

14.27. Remote Method Invocation

See also

Part of the text and programs are from there. Copyright belongs to and

See also

The Java Remote Method Invocation (RMI) system allows an object running in one Java Virtual Machine (VM) to invoke methods on an object running in another Java VM. RMI provides for remote communication between programs written in the Java programming language.

Distributed object applications need to:

Locate remote objects

> Applications can use one of two mechanisms to obtain references to remote objects. An application can register its remote objects with RMI's simple naming facility, the rmiregistry, or the application can pass and return remote object references as part of its normal operation.

Communicate with remote objects

> Details of communication between remote objects are handled by RMI; to the programmer, remote communication looks like a standard Java method invocation.

Load class bytecodes for objects that are passed as parameters or return values

Because RMI allows a caller to pass pure Java objects to remote objects, RMI provides the necessary mechanisms for loading an object's code as well as transmitting its data.

The illustration below depicts an RMI distributed application that uses the registry to obtain references to a remote object. The server calls the registry to associate a name with a remote object. The client looks up the remote object by its name in the server's registry and then invokes a method on it.

14.28. Remote Method Invocation: Idea

The problem is: you need a reference of an object, before you can send a method to it.

14.29. Remote Method Invocation: Idea II

14.30. SenderProxy/Receiver Proxy.png

14.31. Stubs

```
 1      // Stub class generated by rmic, do not edit.
 2      // Contents subject to change without notice.
 3
 4      public final class HelloImplementation_Stub
 5          extends java.rmi.server.RemoteStub
 6          implements HelloInterface, java.rmi.Remote
 7      {
 8          private static final long serialVersionUID = 2;
 9
10          private static java.lang.reflect.Method $method_sayHello_0;
11
12          static {
13              try {
14                  $method_sayHello_0 = HelloInterface.class.getMethod("s
15              } catch (java.lang.NoSuchMethodException e) {
16                  throw new java.lang.NoSuchMethodError(
17                      "stub class initialization failed");
18              }
19          }
20
21          // constructors
22          public HelloImplementation_Stub(java.rmi.server.RemoteRef ref)
23              super(ref);
24          }
25
26          // methods from remote interfaces
27
28          // implementation of sayHello()
29          public java.lang.String sayHello()
30              throws java.rmi.RemoteException
31          {
32              try {
33                  Object $result = ref.invoke(this, $method_sayHello_0,
34                  return ((java.lang.String) $result);
35              } catch (java.lang.RuntimeException e) {
36                  throw e;
37              } catch (java.rmi.RemoteException e) {
38                  throw e;
39              } catch (java.lang.Exception e) {
40                  throw new java.rmi.UnexpectedException("undeclared che
41              }
42          }
```

```
43      }
```

Source Code: Src/16_D/HelloImplementation_Stub.java

Note: *rmic -keep* can generate the stubs.

14.32. Remote Method Innovation Registry

Where does the sender proxy come from?

14.33. Passing Non-remote Objects

A non-remote object, that is passed as a parameter of a remote method invocation or returned as a result of a remote method invocation, is passed by copy; that is, the object is serialized using the Java Object Serialization mechanism.

So, when a non-remote object is passed as an argument or return value in a remote method invocation, the content of the non-remote object is copied before invoking the call on the remote object.

When a non-remote object is returned from a remote method invocation, a new object is created in the calling virtual machine

14.34. Advantages of Dynamic Code Loading

- download the bytecodes (or simply code) of an object's class if the class is not defined in the receiver's virtual machine.
- types and the behavior of an object, previously available only in a single virtual machine, can be transmitted to another, possibly remote, virtual machine.
- RMI passes objects by their true type, so the behavior of those objects is not changed when they are sent to another virtual machine.
- allows new types to be introduced into a remote virtual machine, thus extending the behavior of an application dynamically.

14.35. Remote Interfaces, Objects, and Methods

- distributed application built using Java RMI is made up of interfaces and classes.
- In a distributed application some of the implementations are assumed to reside in different virtual machines.
- Objects that have methods that can be called across virtual machines are remote objects.
- An object becomes remote by implementing a remote interface, which has the following characteristics.
  - — A remote interface extends the interface java/rmi.Remote.
  - — Each method of the interface declares java/rmi.RemoteException in its throws clause, in addition to any application-specific exceptions.
- RMI passes a remote stub for a remote object.
- The stub acts as the local representative, or proxy, for the remote object and basically is, to the caller, the remote reference.
- The caller invokes a method on the local stub, which is responsible for carrying out the method call on the remote object.
- A stub for a remote object implements the same set of remote interfaces that the remote object implements.
- This allows a stub to be cast to any of the interfaces that the remote object implements.
- This also means that only those methods defined in a remote interface are available to be called in the receiving virtual machine.

14.36. Creating Distributed Applications Using RMI

1. Design and implement the components of your distributed application.
2. Compile sources and generate stubs.
3. Make classes network accessible.
4. Start the application.

Compile Sources and Generate Stubs

Compile as usual ... see makefile

Make Classes Network Accessible

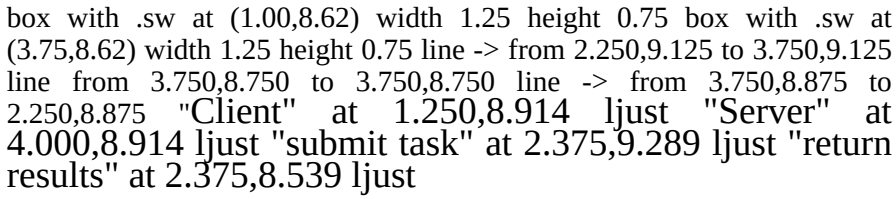
In this step you make everything--the class files associated with the remote interfaces, stubs, and other classes that need to be downloaded to clients.

Start the Application

Starting the application includes running the RMI remote object registry, the server, and the client.

14.37. Intro Example

We have to design a protocol that allows jobs to be submitted to the server and results of the job to be returned to the client. This protocol is expressed in interfaces supported by the server and by the objects that are submitted to the sever, as shown in the following figure.



14.38. Hello World

The Client:

```
 1      import java.rmi.*;
 2
 3      public class HelloC {
 4              public static void main(String args[] ) {
 5              String message = "";
 6              try {
 7                      HelloInterface obj = (HelloInterface)Naming.
 8                                      lookup("//spiegel.cs.rit.edu/IamAl
 9
10                      message = obj.sayHello();
11
12                      System.out.println(message);
13
14              } catch (Exception e) {
15                      System.out.println("HelloC exception: " +
16                      e.getMessage());
17                      e.printStackTrace();
18              }
19        }
20      }
```

Source Code: Src/16_D/HelloC.java

The interface:

```
1       public interface HelloInterface extends java.rmi.Remote {
2               String sayHello() throws java.rmi.RemoteException;
3       }
```

Source Code: Src/16_D/HelloInterface.java

The implementation of the interface:

```
 1      import java.rmi.*;
 2      import java.rmi.server.UnicastRemoteObject;
 3
 4      public class HelloImplementation
 5              extends UnicastRemoteObject
 6              implements HelloInterface {
 7
 8              public HelloImplementation() throws RemoteException {
 9              }
10
11              public String sayHello() throws RemoteException {
12                      return  "Spiegel: Hello World my Friend!";
13              }
14      }
```

Source Code: Src/16_D/HelloImplementation.java

The implementation of the server:

```
 1      import java.rmi.*;
 2
 3      public class HelloServer {
 4
 5              public static void main(String args[])
 6              {
 7                      // System.setSecurityManager(new RMISecurityManage
 8
 9                      try {
10                          HelloInterface obj = new HelloImplementation(
11                          Naming.rebind("//spiegel.cs.rit.edu/IamAhelloI
12                          // Naming.rebind("//129.21.36.56/HelloServer",
13                          System.out.println("HelloServer bound in regis
14                      } catch (Exception e) {
15                          System.out.println("HelloImpl err: " + e.getMe
16                          e.printStackTrace();
17                          System.exit(0);
18                      }
19              }
20      }
```

Source Code: Src/16_D/HelloServer.java

Compilation and Execution

```
all:
    javac HelloInterface.java
    javac HelloImplementation.java
    # rmic HelloImplementation
    javac HelloC.java HelloServer.java
    rmiregistry &
    sleep 1
    java HelloServer &
    sleep 4
    java HelloC
```

The stub:

Source Code: Src/16_D/Keep_HelloImplementation_Stub.java

```
rmic -keep CbVImplementation
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and stati
stubs. See the documentation for java.rmi.server.UnicastRemoteObj
```

14.39. Argument Passing Example

```
1      public interface CbVInterface extends java.rmi.Remote {
2              public String callByValue(int anArray[]) throws java.rmi.R
3      }
```

Source Code: Src/16_CbV/CbVInterface.java

```
1      import java.rmi.*;
2      import java.rmi.server.UnicastRemoteObject;
3
4      public class CbVImplementation
5              extends UnicastRemoteObject
6              implements CbVInterface {
7
8              int id = 0;
9              public CbVImplementation() throws RemoteException {
10             }
11
12             public String callByValue(int array[]) throws RemoteExcept
13                     array[0] = -42;
14                     return  id + ": CbV World my Friend!";
15             }
16     }
```

Source Code: Src/16_CbV/CbVImplementation.java

```
1      import java.rmi.*;
2
3
4      public class CbVC {
5              public static void makeCall(String id)  {
6                      String message = "";
7                      try {
8                              CbVInterface obj = (CbVInterface)Naming.lo
9                              int anArrayLocal[] = new int[2];
10
11                             anArrayLocal[0] = 42;
12                             System.out.println("    before Call: anArr
13                             message = obj.callByValue(anArrayLocal);
14                             System.out.println("    after  Call: anArr
15
16                             System.out.println(id + " - " + message);
17                     } catch (Exception e) {
18                             System.out.println("CbVC exception: " +
19                             e.getMessage());
20                             e.printStackTrace();
21                     }
22             }
23             public static void main(String args[] ) {
24                     makeCall("first");
25             }
```

```
26      }
```

Source Code: Src/16_CbV/CbVC.java

```
 1      import java.rmi.*;
 2
 3      public class CbVServer {
 4
 5              public static void main(String args[])
 6              {
 7                      // System.setSecurityManager(new RMISecurityManage
 8
 9                      try {
10                          CbVInterface obj = new CbVImplementation();
11                          Naming.rebind("//spiegel/IamACbVImplementatio
12                          // Naming.rebind("//129.21.36.56/CbVServer", c
13                          System.out.println("CbVServer bound in regist
14                      } catch (Exception e) {
15                          System.out.println("CbVImpl err: " + e.getMes
16                          e.printStackTrace();
17                      }
18              }
19      }
```

Source Code: Src/16_CbV/CbVServer.java

```
% java CbVC
    before Call: anArrayLocal[0] = 42
    after  Call: anArrayLocal[0] = 42
first - 0: CbV World my Friend!
```

14.40. RMISecurityManager

The class defines a default security policy for RMI applications (not applets). For code loaded from a class loader, the security manager disables all functions except class definition and access. This class may be subclassed to implement a different policy. To set a RMISecurityManager, add the following to an application's main() method:

```
System.setSecurityManager(new RMISecurityManager());
```

If no security manager has been set, RMI will only load classes from local system files as defined by CLASSPATH.

Naming

is the bootstrap mechanism for obtaining references to remote objects based on Uniform Resource Locator (URL) syntax. The URL for a remote object is specified using the usual host, port and name:

```
rmi://host:port/name
```

host = host
: name of registry (defaults to the current host)

port = port
: number of registry (defaults to the registry port number)

name = name
: for remote object

The makefile:

```
 1
 2
 3      all:
 4              javac HelloInterface.java
 5              javac HelloImplementation.java
 6              # rmic HelloImplementation
 7              javac HelloC.java HelloServer.java
 8              rmiregistry &
 9              sleep 4
10              java HelloServer &
11              sleep 4
12              java HelloC
13
14
15      clean:
16              rm -f *class
17
```

Source Code: Src/16_D/makefile

```
% make -f Makefile
javac HelloInterface.java
javac HelloImplementation.java
# rmic HelloImplementation
javac HelloC.java HelloServer.java
rmiregistry &
sleep 1
java HelloServer &
sleep 4
HelloServer bound in registry
java HelloC
Hello World my friend.
```

Note: Make sure that *rmiregistry* is dead before you log out!

Note: Make sure that every *java server* is dead before you log out!

```
 1      #!/bin/sh
 2
 3      killThisProcess ()      {       # $1
 4              echo $1
 5              ps -axl                 |       \
 6              grep "$1"               |       \
 7              grep -v grep            |       \
 8              grep -v kill            |       \
 9              grep -v vi              |       \
10              awk ' { print $2 }'     |       \
11              while read x
12              do
13                      echo "kill -9 $x"
14                      kill -9 $x 2> /dev/null
15              done
16
17      }
18      killThisProcess "rmiregistry"
19      killThisProcess "java"
```

Source Code: Src/16_D/killIt

```
% ps -edf | grep java
```

The improved makefile:

```
 1
 2
 3      all:
 4              javac HelloInterface.java
 5              javac HelloImplementation.java
 6              # rmic HelloImplementation
 7              javac HelloC.java HelloServer.java
 8              rmiregistry &
 9              sleep 4
10              java HelloServer &
11              sleep 4
12              java HelloC
13
14
15      clean:
16              rm -f *class
17
```

Source Code: Src/16_D/makefile

```
% make
rmic HelloImpl
javac HelloC.java
rmiregistry &
sleep 1
java HelloImpl &
sleep 4
        HelloImpl: HelloImpl(String s)
HelloServer bound in registry
java HelloC
Hello World my friend.
killIt java
kill -9 26491
kill -9 26489
kill -9 26438
kill -9 26332
kill -9 26501
Killed
killIt
Killed
```

The registry by default runs on port 1099. To start the registry on a different port, specify the port number in the command. For example, to start the registry on port 2001:

```
% rmiregistry 2001
```

For example, if the registry is running on port 2001 in the Hello World example, here is the call required to bind HelloServer to the remote object reference:

```
Naming.lookup("//yps:2001/HelloServer", obj);
```

14.41. Hello World II

The Client:

```
 1      import java.rmi.*;
 2
 3      public class HelloC {
 4              public static void main(String args[] ) {
 5              String message = "";
 6              try {
 7                      Hello obj =
 8                              (Hello)Naming.lookup("//spiegel.cs.rit.ed
 9
10                      message = obj.sayHello();
11
12                      System.out.println(message);
13
14              } catch (Exception e) {
15                      System.out.println("Something went wrong: " +
16                              e.getMessage());
17                      e.printStackTrace();
18              }
19        }
20      }
```

Source Code: Src/16_P/HelloC.java

The Server:

```
 1      import java.rmi.*;
 2      import java.rmi.server.UnicastRemoteObject;
 3
 4      public class HelloImpl
 5              extends UnicastRemoteObject
 6              implements Hello
 7      {
 8              private String name;
 9
10              public HelloImpl(String s) throws RemoteException {
11                      name = s;
12              }
13
14              public String sayHello() throws RemoteException {
15                      return  "Stanley Kubrick was there!";
16              }
17
18              public static void main(String args[])
19              {
20                      // Create and install a security manager
21                      // System.setSecurityManager(new RMISecurityManage
22
23                      try {
24                              HelloImpl obj = new HelloImpl("HelloServe
25                              Naming.rebind("//spiegel.cs.rit.edu:2001/
26                              System.out.println("HelloServer bound in
27                      } catch (Exception e) {
28                              System.out.println("HelloImpl err: " + e.
29                              e.printStackTrace();
30                      }
31              }
32      }
```

Source Code: Src/16_P/HelloImpl.java

The interface:

```
1       public interface Hello extends java.rmi.Remote {
2               String sayHello() throws java.rmi.RemoteException;
3       }
```

Source Code: Src/16_P/Hello.java

The makefile:

```
 1
 2
 3      all: Hello.class HelloC.class HelloImpl.class   \
 4           HelloImpl_Skel.class HelloImpl_Stub.class
 5
 6              rmiregistry 2001 &
 7              sleep 1
 8              java HelloImpl &
 9              sleep 4
10              java HelloC
11              killIt java
12              killIt
13
14
15      HelloImpl_Skel.class HelloImpl_Stub.class: HelloImpl.java
16              rmic HelloImpl
17
18      Hello.class:    Hello.java
19              javac Hello.java
20
21      HelloC.class:   HelloC.java
22              javac HelloC.java
23
24      HelloImpl.class:        HelloImpl.java
25              javac HelloImpl.java
26
27      clean:
28              rm -f *class
```

Source Code: Src/16_P/makefile

14.42. Multiple Servers

Client:

```
 1      import java.rmi.*;
 2
 3      public class Client {
 4              public static void main(String args[] ) {
 5              String message = "";
 6              try {
 7                      MyServer obj =
 8                              (MyServer)Naming.lookup("//localhost:200
 9                      message = obj.sayHello();
10                      System.out.println(message);
11
12
13                      obj = (MyServer)Naming.lookup("//localhost:2001/Se
14                      message = obj.sayHello();
15                      System.out.println(message);
16
17              } catch (Exception e) {
18                      System.out.println("Something went wrong: " +
19                      e.getMessage());
20                      e.printStackTrace();
21              }
22        }
23      }
```

Source Code: Src/16_M/Client.java

Server 1:

```
 1      import java.rmi.*;
 2      import java.rmi.server.UnicastRemoteObject;
 3
 4      public class Server1Impl
 5              extends UnicastRemoteObject
 6              implements MyServer
 7      {
 8              private String name;
 9
10              public Server1Impl(String s) throws RemoteException {
11                      name = s;
12              }
13
14              public String sayHello() throws RemoteException {
15                      return  "Server1()";
16              }
17
18              public static void main(String args[])
19              {
20                      // Create and install a security manager
21                      // System.setSecurityManager(new RMISecurityManage
22
23                      try {
24                              Server1Impl obj = new Server1Impl("Server1
25                              Naming.rebind("//localhost:2001/Server1",
26                              System.out.println("Server1 bound in regis
27                      } catch (Exception e) {
28                              System.out.println("Server1Impl err: "
29                                 + e.getMessage());
30                              e.printStackTrace();
31                      }
32              }
33      }
```

Source Code: Src/16_M/Server1Impl.java

Server 2:

```
 1      import java.rmi.*;
 2      import java.rmi.server.UnicastRemoteObject;
 3
 4      public class Server2Impl
 5              extends UnicastRemoteObject
 6              implements MyServer
 7      {
 8              private String name;
 9
10              public Server2Impl(String s) throws RemoteException {
11                      name = s;
12              }
13
14              public String sayHello() throws RemoteException {
15                      return  "Server2()";
16              }
17
18              public static void main(String args[])
19              {
20                      // Create and install a security manager
21                      // System.setSecurityManager(new RMISecurityManage
22
23                      try {
24                              Server2Impl obj = new Server2Impl("Server2
25                              Naming.rebind("//localhost:2001/Server2",
26                              System.out.println("Server2Server bound in
27                      } catch (Exception e) {
28                              System.out.println("Server2Impl err: "
29                                 + e.getMessage());
30                              e.printStackTrace();
31                      }
32              }
33      }
```

Source Code: Src/16_M/Server2Impl.java

```
% make
rmic Server1Impl
rmic Server2Impl
javac Client.java
rmiregistry 2001 &
sleep 1
java Server1Impl &
java Server2Impl &
sleep 4
Server2Server bound in registry
Server1 bound in registry
java Client
Server1()
Server2()
```

14.43. Running Multiple Server on different Machines

- Before a client can connect to a server, a server process must run on this machine.
- Server can be started — on a UNIX box — during
  - — boot time
  - — via *inetd*
  - — "by hand"

- In order to start a a server by hand, you have to log on this machine.
- ssh
- rsh ????

14.44. Startup Multiple Server on different Machines

Client:

```
 1      import java.rmi.*;
 2      import java.math.*;
 3
 4      public class Client {
 5
 6          public static void doIt(String catServer, String mouseServer,
 7
 8              MyServer aCatServer;
 9              MyServer aMouseServer;
10              Point    aPoint = new Point(4, 2 );
11
12              System.out.println("In Client: cat   is on: " + catServer
13              System.out.println("In Client: mouse is on: " + mouseServe
14              System.out.println("In Client: port     is: " + port );
15              try {
16                      aCatServer = (MyServer)Naming.lookup("rmi://" +
17                              catServer + ":" + port + "/CatServer");
18
19                      aMouseServer = (MyServer)Naming.lookup("rmi://" +
20                              mouseServer + ":" + port + "/MouseServer"
21
22
23      // -------------- Cat --------------------
24              System.out.println("In Client: aCatServer.movePoint(aPoint
25                   (aPoint = aCatServer.movePoint(aPoint)).toString() )
26              System.out.println("In Client: aCatServer.movePoint(aPoint
27                              aCatServer.movePoint(aPoint).toString() )
28              System.out.println("In Client: aCatServer.movePoint(aPoint
29                              aCatServer.movePoint(aPoint).toString() )
30
31      // -------------- Mouse --------------------
32
```

```
33              System.out.println("In Client: aMouseServer.movePoint(aPo
34                   (aPoint = aMouseServer.movePoint(aPoint)).toString()
35              System.out.println("In Client: aMouseServer.movePoint(aPo
36                              aMouseServer.movePoint(aPoint).toString()
37              System.out.println("In Client: aMouseServer.movePoint(aPo
38                              aMouseServer.movePoint(aPoint).toString()
39
40
41
42              } catch (Exception e) {
43                      System.out.println("Something went wrong: " +
44                      e.getMessage());
45                      e.printStackTrace();
46              }
47
48          }
49
50           public static void main(String args[] ) {
51           int    port   = 1099;
52           String catServer   = "yps";
53           String mouseServer = "yps";
54
55           if ( args.length  >= 1 )
56               catServer = args[0];
57           if ( args.length  >= 2 )
58               mouseServer = args[1];
59           if ( args.length  == 3 )
60               try {
61                       port = Integer.parseInt(args[2]);
62               }
63               catch ( NumberFormatException e )       {
64                       System.out.println("Hm , port = " +
65                               args[2] + " is not valid.");
66                       System.exit(1);
67               }
68
69           if ( args.length  > 3 )     {
70               System.out.println("Usage: " +
71                               "java Client [CatServer [MouseServer [port
72               System.exit(1);
73           }
74
75           doIt(catServer, mouseServer, port);
76         }
77      }
```

Source Code: Src//16_MS/Client.java

Interface:

```
1      public interface MyServer extends java.rmi.Remote {
2              Point movePoint(Point aPoint) throws java.rmi.RemoteExcep
3              int getX()                    throws java.rmi.RemoteExcep
4              int getY()                    throws java.rmi.RemoteExcep
5      }
```

Source Code: Src//16_MS/MyServer.java

Cat Server

```
 1      import java.rmi.*;
 2      import java.rmi.server.UnicastRemoteObject;
 3
 4      public class CatServer extends UnicastRemoteObject implements MySe
 5      {
 6              final private int DELTA = 10;
 7              private int x;
 8              private int y;
 9              private Point aPoint;
10
11              public CatServer() throws RemoteException {
12                      ;
13              }
14
15
16              public Point movePoint(Point aPoint) throws RemoteExceptio
17                      System.out.println("\tIN CatServer: movePoint(): "
18                              + aPoint.toString() );
19                      return aPoint.move(DELTA, DELTA);
20              }
21              public int getX() throws RemoteException {
22                      System.out.println("\tCIN atServer: getX(): " + x
23                      return  x;
24              }
25
26              public int getY() throws RemoteException {
27                      System.out.println("\tCIN atServer: getY(): " + y
28                      return  x;
29              }
30
31      public static void main(String args[])
32        {
33                int port = 1099;
34
35                // System.setSecurityManager(new RMISecurityManager());
36
37                if ( args.length  == 1 )
38                        try {
39                                port = Integer.parseInt(args[0]);
40                         }
41                         catch ( NumberFormatException e )    {
42                                System.out.println("Hm , port = " +
43                                args[0] + " is not valid.");
44                                System.exit(1);
45                          }
46
47                try {
48                        CatServer obj = new CatServer();
49                        System.out.println("\tIN CatServer: " +
50                                        "rmi://:" + port + "/CatServer");
51                        Naming.rebind("rmi://:" + port + "/CatServer", o
```

```
52                      System.out.println("\tIN CatServer bound in regi
53              } catch (RemoteException  e) {
54                      System.out.println("CatServer RemoteException "
55                      e.printStackTrace();
56              } catch (Exception e) {
57                      System.out.println("CatServer err: "
58                      + e.getMessage());
59                      e.printStackTrace();
60              }
61        }
62      }
```

Source Code: Src//16_MS/CatServer.java

Point Class:

```
 1      /**
 2       * This class implements a point in a two dimensional
 3       * area.
 4       * All methods print the method name, when they are called.
 5       * state information includes:
 6       *
 7       * @version   $Id$
 8       *
 9       * RIT's home page: <a href="http://www.cs.rit.edu/~hpb">RIT</a>
10       *
11       * Revisions:
12       *      $Log$
13       */
14      import java.io.*;
15
16
17      public class Point implements Serializable {
18
19        private int x;                // x coordinate of the point
20        private int y;                // y cooridnate of the point
21
22      /**
23       * Constructor.
24       * initialize x and y values of a point
25       *
26       * @param       x       x coordinate
27       * @param       y       y coordinate
28       *
29       * @return      a Point object
30       */
31        public Point(int _x, int _y){
32              this.x = _x;
33              this.y = _y;
34        }
35
36        private void writeObject(ObjectOutputStream s) throws IOExceptio
37                 s.defaultWriteObject();
38        }
39
40        private void readObject(ObjectInputStream s) throws IOException
41              try {
42                      s.defaultReadObject();
43              }
44              catch ( ClassNotFoundException e)       {
45                  System.out.println(e.getMessage());
46                  e.printStackTrace();
47              }
48        }
49      /**
50       * initialzes x and y of a point.
51       *
```

```
52       * @param       x       int x coordinate
53       * @param       y       int y coordinate
54       *
55       * @return      a Point object
56       */
57        public Point initPoint(int _x, int _y){
58
59              this.x = _x;
60              this.y = _y;
61
62              return this;
63        }
64
65      /**
66       * moves a point
67       *
68       * @param       _x      int delta x value
69       * @param       _y      int delta y value
70       *
71       * @return      a Point object
72       */
73        public Point move(int _x, int _y){
74
75              this.x += _x;
76              this.y += _y;
77              return this;
78        }
79
80      /**
81       * Returns the x coordinate of a point
82       *
83       * @return x value
84       */
85        public int getX(){
86              return this.x;
87        }
88
89      /**
90       * Returns the y coordinate of a point
91       *
92       * @return y value
93       */
94        public int getY(){
95              return this.y;
96        }
97
98      /**
99       * Returns a String reperesentation of the point
00       *
01       * @return String reprasentation of the point
02       */
03        public String toString(){
04              return "Point at (" + x + "/" + y + ")";
05        }
```

```
06      }
```

Source Code: Src/16_MS/Point.java

Makefile:

```
 1
 2
 3      all: Point.class                                          \
 4           CatServer_Skel.class CatServer_Stub.class  \
 5           MouseServer_Skel.class MouseServer_Stub.class      \
 6           MyServer.class Client.class
 7
 8              fireItUp
 9
10
11      CatServer_Skel.class CatServer_Stub.class: CatServer.java
12              rmic CatServer
13      MouseServer_Skel.class MouseServer_Stub.class: MouseServer.java
14              rmic MouseServer
15
16      MyServer.class: MyServer.java
17              javac MyServer.java
18
19      Client.class:   Client.java
20              javac Client.java
21
22      CatServer.class:        CatServer.java
23              javac CatServer.java
24
25      MouseServer.class:      MouseServer.java
26              javac MouseServer.java
27
28      Point.class:    Point.java
29              javac Point.java
30
31      clean:
32              rm -f *class
```

Source Code: Src/16_MS/makefile

Result:

```
        IN CatServer: //yps:2001/CatServer
        IN MouseServer: /yps:2001/MouseServer
        IN CatServer bound in registry
In Client: cat   is on: yps
In Client: mouse is on: yps
In Client: port     is: 2001
        IN MouseServer bound in registry
        IN CatServer: movePoint(): Point at (4/2)
In Client: aCatServer.movePoint(aPoint): Point at (14/12)
        IN CatServer: movePoint(): Point at (14/12)
In Client: aCatServer.movePoint(aPoint): Point at (24/22)
        IN CatServer: movePoint(): Point at (14/12)
In Client: aCatServer.movePoint(aPoint): Point at (24/22)
```

Start of a fireItUp Script:

```
#!/bin/sh

KILL_IT="killIt; killIt java"
ME="`who am i | sed 's/ .*//'`"
HOSTNAME="`hostname`"
USEDHOSTS="yps yps yps" # <-- modify here ...
WD=`pwd`


remote_cmd()            # bg host cmd
{
        echo "$HOSTNAME $ME" > $HOME/.rhosts
        if [ $1 = "bg" ]
        then
                rsh $2 "rm -f $HOME/.rhosts; cd $WD && $3" &
        else
                rsh $2 "rm -f $HOME/.rhosts; cd $WD && $3"
        fi
}

kill_all()
{
        for i in $USEDHOSTS
        do
                remote_cmd fg $i "$KILL_IT" 2>&1 > /dev/null
        done
}









kill_all
sleep 2

echo 1
rmiregistry &
echo "Waiting for rmiregistry .... chrr ... "; sleep 1
java CatServer &

echo 2
remote_cmd bg yps "rmiregistry &"
```

```
52
53      echo 3
54      echo "Waiting for rmiregistry .... chrr ... "; sleep 2
55      remote_cmd bg yps "java MouseServer &"
56
57      echo 4
58      echo "Waiting for the servers  .... chrr ... "; sleep 2
59      remote_cmd fg yps "java Client $HOSTNAME stones"
60
61      kill_all
62
63      exit 0
64
```

Source Code: Src/16_MS/fireItUp

14.45. Calculating PI

```
% make
javac Client.java
rmiregistry &
sleep 1
java PiServer &
sleep 4
        PiServer: PiServer()
PiServer bound in registry
java Client
3.1415926536
% java Client 100
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899
86280348253421170680
%
```

Server Interface:

```
1      public interface MyServer extends java.rmi.Remote {
2              String sayHello() throws java.rmi.RemoteException;
3      }
```

Source Code: Src/16_M/MyServer.java

Class java/gmath.BigDecimal

Immutable, arbitrary-precision signed decimal numbers. A BigDecimal consists of an arbitrary precision integer value and a non-negative integer scale, which represents the number of decimal digits to the right of the decimal point. (The number represented by the BigDecimal is intVal/7**scale.) BigDecimals provide operations for basic arithmetic, scale manipulation, comparison, format conversion and hashing.

The compute engine, a remote object in the server, takes tasks from clients, runs them, and returns any results. The tasks are run on the machine where the server is running. This sort of distributed application could allow a number of client machines to make use of a particularly powerful machine or one that has specialized hardware.

Client:

```
 1      import java.rmi.*;
 2      import java.math.*;
 3
 4      public class Client {
 5
 6          public static void doIt(String host, String port, int digits)
 7              String message = "";
 8              try {
 9                      MyServer obj = (MyServer)Naming.lookup("//" +
10                              host + ":" + port + "/PiServer");
11                      System.out.println(obj.computePi(digits));
12
13              } catch (Exception e) {
14                      System.out.println("Something went wrong: " +
15                      e.getMessage());
16                      e.printStackTrace();
17              }
18
19         }
20
21          public static void main(String args[] ) {
22          int    digits = 10;
23          String host   = "yps";
24          String port   = "";
25
26
27          if ( args.length  >= 1 )    {
28              try {
29                      digits = Integer.parseInt(args[0]);
30              }
31              catch ( NumberFormatException e )       {
32                      System.out.println("Hm , digits = " + args[0]);
33                      System.exit(1);
34              }
35
36          }
37          if ( args.length >= 2 )     {
38              host = args[1];
39          }
40
41          if ( args.length  == 3 )    {
42              try {
43                      port = args[2];
44                      Integer.parseInt(port);
45              }
46              catch ( NumberFormatException e )       {
```

```
47                      System.out.println("Port = " + port + " is not va
48                      System.exit(1);
49              }
50
51          }
52          if ( args.length  > 3 )     {
53              System.out.println("Usage: java Client [digits [host [por
54              System.exit(1);
55          }
56          doIt(host, port, digits);
57        }
58      }
```

Source Code: Src/16_C/Client.java

Interface:

```
1       import java.math.*;
2
3       public interface MyServer extends java.rmi.Remote {
4               BigDecimal computePi(int digits)
5               throws java.rmi.RemoteException;
6       }
```

Source Code: Src/16_C/MyServer.java

Server:

```
 1      import java.rmi.*;
 2      import java.math.*;
 3      import java.rmi.server.UnicastRemoteObject;
 4
 5      public class PiServer
 6              extends UnicastRemoteObject
 7              implements MyServer
 8      {
 9          /** constants used in pi computation */
10          private static final BigDecimal ZERO =
11              BigDecimal.valueOf(0);
12          private static final BigDecimal  ONE =
13              BigDecimal.valueOf(1);
14          private static final BigDecimal FOUR =
15              BigDecimal.valueOf(4);
16
17          /** rounding mode to use during pi computation */
18          private static final int roundingMode =
19              BigDecimal.ROUND_HALF_EVEN;
20
21          /** digits of precision after the decimal point */
22          private int digits;
23
24
25          /**
26           * Construct a task to calculate pi to the specified
27           * precision.
28           */
29          public PiServer() throws RemoteException {
30              System.out.println("\tPiServer: PiServer()");
31          }
32          /**
33           * Compute the value of pi to the specified number of
34           * digits after the decimal point.  The value is
35           * computed using Machin's formula:
36           *
37           *          pi/4 = 4*arctan(1/5) - arctan(1/239)
38           *
39           * and a power series expansion of arctan(x) to
40           * sufficient precision.
41           */
42          public BigDecimal computePi(int digits) throws RemoteExceptio
43              int scale = digits + 5;
44              BigDecimal arctan1_5 = arctan(5, scale);
45              BigDecimal arctan1_239 = arctan(239, scale);
46              BigDecimal pi = arctan1_5.multiply(FOUR).subtract(
47                                          arctan1_239).multiply(FOUR);
48              return pi.setScale(digits,
49                                 BigDecimal.ROUND_HALF_UP);
50          }
51          /**
```

```
52          * Compute the value, in radians, of the arctangent of
53          * the inverse of the supplied integer to the speficied
54          * number of digits after the decimal point.  The value
55          * is computed using the power series expansion for the
56          * arc tangent:
57          *
58          * arctan(x) = x - (x^3)/3 + (x^5)/5 - (x^7)/7 +
59          *     (x^9)/9 ...
60          */
61         public static BigDecimal arctan(int inverseX,
62                                         int scale)
63         {
64             BigDecimal result, numer, term;
65             BigDecimal invX = BigDecimal.valueOf(inverseX);
66             BigDecimal invX2 =
67                 BigDecimal.valueOf(inverseX * inverseX);
68
69             numer = ONE.divide(invX, scale, roundingMode);
70
71             result = numer;
72             int i = 1;
73             do {
74                 numer =
75                     numer.divide(invX2, scale, roundingMode);
76                 int denom = 2 * i + 1;
77                 term =
78                     numer.divide(BigDecimal.valueOf(denom),
79                                  scale, roundingMode);
80                 if ((i % 2) != 0) {
81                     result = result.subtract(term);
82                 } else {
83                     result = result.add(term);
84                 }
85                 i++;
86             } while (term.compareTo(ZERO) != 0);
87             return result;
88         }
89
90
91             public static void main(String args[])
92             {
93                     // Create and install a security manager
94                     // System.setSecurityManager(new RMISecurityManage
95
96                     try {
97                             PiServer obj = new PiServer();
98                             Naming.rebind("//yps:2042/PiServer", obj)
99                             System.out.println("PiServer bound in regi
00                     } catch (Exception e) {
01                             System.out.println("PiServer err: " + e.ge
02                             e.printStackTrace();
03                     }
04             }
05     }
```

Source Code: Src/16_C/PiServer.java

14.46. Receiving and Sending Objects

Interface:

```
1      import java.util.*;
2
3      public interface HashTableInterface extends java.rmi.Remote {
4          Hashtable playWithAHashTable(String t)
5                       throws java.rmi.RemoteException;
6      }
```

Source Code: Src/16_Hash/HashTableInterface.java

Client:

```
 1      import java.rmi.*;
 2      import java.util.*;
 3
 4      public class HashTableC {
 5              public static void main(String args[] ) {
 6              String plusMovie = "Smoke Signals";
 7              Hashtable aHashTable = new Hashtable();
 8              aHashTable.put("plusplus  Movie", "Comedian Harmonists");
 9
10              System.out.println("Client: aHashTable local = " +
11                              aHashTable.toString());
12              try {
13
14                      HashTableInterface obj =
15                               (HashTableInterface)Naming.lookup("//yps/
16
17                      aHashTable = obj.playWithAHashTable(plusMovie);
18
19              } catch (Exception e) {
20                      System.out.println("HelloApplet exception: " +
21                      e.getMessage());
22                      e.printStackTrace();
23              }
24              System.out.println("Client: aHashTable remote = " +
25                              aHashTable.toString());
26        }
27      }
```

Source Code: Src/16_Hash/HashTableC.java

Server:

```
 1      import java.util.*;
 2      import java.rmi.*;
 3      import java.rmi.server.UnicastRemoteObject;
 4
 5      public class HashTableServer
 6              extends UnicastRemoteObject
 7              implements HashTableInterface {
 8              private String name;
 9
10              public HashTableServer(String s) throws RemoteException {
11                      System.out.println(
12                         "\tHashTableServer: HashTableServer(String s)"
13                      name = s;
14              }
15
16              public Hashtable playWithAHashTable(String t)
17                      throws java.rmi.RemoteException {
18                      Hashtable aHashTable = new Hashtable();
19                      aHashTable.put("plusplus  Movie", t);
20                      System.out.println("\tserver: aHashTable = " +
21                              aHashTable.toString());
22                      t = "done";
23                      return aHashTable;
24              }
25              public static void main(String args[])
26              {
27                      // System.setSecurityManager(new RMISecurityManage
28
29                      try {
30                          HashTableServer obj = new HashTableServer("He
31                          Naming.rebind("//yps/HelloServer", obj);
32                          System.out.println("HelloServer bound in regis
33                      } catch (Exception e) {
34                          System.out.println("HashTableServer err: " + e
35                          e.printStackTrace();
36                      }
37              }
38      }
```

Source Code: Src/16_Hash/HashTableServer.java

Result:

```
% make
rmic HashTableServer
javac HashTableC.java
rmiregistry &
sleep 1
java HashTableServer &
sleep 4
        HashTableServer: HashTableServer(String s)
HelloServer bound in registry
java HashTableC
Client: aHashTable = {plusplus  Movie=Comidian Harmonists}
        server: aHashTable = {plusplus  Movie=Smoke Signals}
Client: aHashTable = {plusplus  Movie=Smoke Signals}
Client: plusMovie = Smoke Signals
killIt java
kill -9 27386
Killed
killIt
kill -9 27366
kill -9 27374
# make clean
```

14.47. RMI and Multi Threaded Systems

How does a remote method gets executed?

The Interface:

```
1       public interface MultiTInterface extends java.rmi.Remote {
2               String comeBackASAP() throws java.rmi.RemoteException;
3               String sleepForAwhile() throws java.rmi.RemoteException;
4       }
```

Source Code: Src/16_T/MultiTInterface.java

The Client:

```
1       import java.rmi.*;
2
3       public class MultiTC {
4               public static void main(String args[] ) {
5               String message = "";
6               System.out.println("Going ... ");
7               try {
8                       MultiTInterface obj =
9                               (MultiTInterface)Naming.lookup("//spiege
10                      // 1
11                      System.out.println("Call sleepForAwhile ...");
12                      message = obj.sleepForAwhile();
13                      System.out.println(message);
14                      System.out.println("Call comeBackASAP ...");
15                      message = obj.comeBackASAP();
```

```
16                      System.out.println(message);
17                      // 2
18                      System.out.println("Call comeBackASAP ...");
19                      message = obj.comeBackASAP();
20                      System.out.println(message);
21                      System.out.println("Call sleepForAwhile ...");
22                      message = obj.sleepForAwhile();
23                      System.out.println(message);
24
25              } catch (Exception e) {
26                      System.out.println("Something went wrong: " +
27                              e.getMessage());
28                      e.printStackTrace();
29              }
30        }
31      }
```

Source Code: Src/16_T/MultiTC.java

The Server:

```
 1      import java.rmi.*;
 2      import java.rmi.server.UnicastRemoteObject;
 3
 4      public class MultiTImpl
 5              extends UnicastRemoteObject
 6              implements MultiTInterface
 7      {
 8              private String name;
 9
10              public MultiTImpl(String s) throws RemoteException {
11                      name = s;
12              }
13
14              public String sleepForAwhile() throws RemoteException {
15                      try {
16                              Thread.sleep(2000);
17                      } catch ( Exception e )         {
18                              e.printStackTrace();
19                      }
20                      return  "sleepForAwhile";
21              }
22
23              public String comeBackASAP() throws RemoteException {
24                      return  "comeBackASAP";
25              }
26
27              public static void main(String args[])
28              {
29                      try {
30                              MultiTImpl obj = new MultiTImpl("MultiTSer
31                              Naming.rebind("//spiegel:2001/MultiTServer
32                              System.out.println("MultiTServer bound in
33                      } catch (Exception e) {
```

```
34                              System.out.println("MultiTImpl err: " + e
35                              e.printStackTrace();
36                      }
37              }
38      }
```

Source Code: Src/16_T/MultiTImpl.java

The makefile:

```
 1
 2
 3      all: MultiTInterface.class MultiTC.class MultiTImpl.class       \
 4           MultiTImpl_Skel.class MultiTImpl_Stub.class
 5              rmiregistry 2001 &
 6              sleep 1
 7              java MultiTImpl &
 8              sleep 4
 9              java MultiTC 1 &
10              java MultiTC  &
11              killIt java
12              killIt
13
14
15      MultiTImpl_Skel.class MultiTImpl_Stub.class: MultiTImpl.java
16              rmic MultiTImpl
17
18      MultiTInterface.class:  MultiTInterface.java
19              javac MultiTInterface.java
20
21      MultiTC.class:  MultiTC.java
22              javac MultiTC.java
23
24      MultiTImpl.class:       MultiTImpl.java
25              javac MultiTImpl.java
26
27      clean:
28              rm -f *class
```

Source Code: Src/16_T/makefile

14.48. Dynamic Class Loading

RMI allows parameters, return values and exceptions passed in RMI calls to be any object that is serializable. RMI uses the object serialization mechanism to transmit data from one virtual machine to another and also annotates the call stream with the appropriate location information so that the class definition files can be loaded at the receiver.

When parameters and return values for a remote method invocation are unmarshaled to become live objects in the receiving VM, class definitions are required for all of the types of objects in the stream. The unmarshmaling process first attempts to resolve classes by name in its local class loading context (the context class loader of the current thread). RMI also provides a facility for dynamically loading the class definitions for the actual types of objects passed as parameters

and return values for remote method invocations from network locations specified by the transmitting endpoint. This includes the dynamic downloading of remote stub classes corresponding to particular remote object implementation classes (and used to contain remote references) as well as any other type that is passed by value in RMI calls, such as the subclass of a declared parameter type, that is not already available in the class loading context of the unmarshmaling side.

To support dynamic class loading, the RMI runtime uses special subclasses of java/io.ObjectOutputStream and java/io.ObjectInputStream for the marshal streams that it uses for marshaling and unmarshmaling RMI parameters and return values.

14.49. Java Object Serialization Security Issues

See here:

Problem:

- Classes/Objects are to remote process (JVM)
- Unmarshalled
- Developer:
  - — Trust communication channel
  - — Assume binary objects can not be changed
  - — Assume Serialization is safe
- Idea: how to allow "xxx" to login, instead of "hpb"

```
 1      import java.io.*;
 2      import java.util.Date;
 3
 4      public class ObjectWriter_5 {
 5        public static void main( String args[] ) {
 6          try {
 7              FileOutputStream ostream = new FileOutputStream("object_5
 8              ObjectOutputStream p = new ObjectOutputStream(ostream);
 9              p.writeObject("User: " + "hpb");
10              p.close();
11          }
12          catch ( IOException e)      {
13              System.out.println(e.getMessage());
14          }
15        }
16      }
```

Source Code: Src/9_was/ObjectWriter_5.java

```
% od -c object_5.data
0000000  254 355    005   t        U   s   e   r   :       h   p   b
0000020
% od -c x.data
0000000  254 355    005   t        U   s   e   r   :
0000015
% echo -n  xxx  >> x.data
% od -c x.data
```

```
0000000  254 355    005   t       U   s   e   r   :       x   x   x
0000020
```

- the same can be done with classes, and objects

Abritary Code Execution

- Code Reuse attack (return-oriented programming)
  - control of the call stack
  - Executed carefully chosen machine instructions (gadgets)
  -
  -
  -

```
if (1 + 2 + payload + 16 > s->s3->rrec.length) return 0; /* silen
```

-
- defaultReadObject -

Restrict Deserialization

- Default ObjectInputStream will deserialize any serializable class
- Class Black/White Listening

15. Reflection API

Most of the stuff ’stolen’ from

The represents, or reflects, the classes, interfaces, and objects in the current Java Virtual Machine.

Typical use:

- debuggers,
- class browsers,
- GUI builders.

With the reflection API you can:

- Determine the class of an object.
- Get information about a class’s modifiers, fields, methods, constructors, and superclasses.
- Find out what constants and method declarations belong to an interface.
- Create an instance of a class whose name is not known until runtime.
- Get and set the value of an object’s field, even if the field name is unknown to your program until runtime.
- Invoke a method on an object, even if the method is not known until runtime.

- Create a new array, whose size and component type are not known until run-time, and then modify the array’s component

15.1. Intro example

```
 1      import java.lang.reflect.*;
 2
 3      class Intro {
 4
 5              static void printName(Object o) {
 6                      Class c = o.getClass();
 7                      String s = c.getName();
 8                      System.out.println(s);
 9              }
10
11              public static void main(String[] args) {
12                      String aS = new String();
13                      printName(aS);
14              }
15      }
```

Source Code: Src/17/Intro.java

Result:

```
% javac Intro.java
% java Intro
java.lang.String
```

15.2. Discovering Class Modifiers

See

```
 1      import java.lang.reflect.*;
 2
 3      public final class MyModifier {
 4
 5              void printName(Object o) {
 6                      Class c = o.getClass();
 7                      String s = c.getName();
 8                      System.out.println(s);
 9              }
10
11              public void printModifiers(Object o) {
12              Class c = o.getClass();
13              int m = c.getModifiers();
14
15              if (Modifier.isPrivate(m))
16                      System.out.println("private");
17              if (Modifier.isPublic(m))
18                      System.out.println("public");
19              if (Modifier.isAbstract(m))
20                      System.out.println("abstract");
21              if (Modifier.isFinal(m))
22                      System.out.println("final");
23              }
24
25              public static void main(String[] args) {
26                      MyModifier aM = new MyModifier();
27                      aM.printName(aM);
28                      aM.printModifiers(aM);
29              }
30
31      }
```

Source Code: Src/17/MyModifier.java

Result:

```
% java MyModifier
MyModifier
public
final
```

15.3. Identifying Class Fields

Application such as a class browser, might want to find out what fields belong to a particular class. This can be identified by invoking the getFields method on a Class object. The getFields method returns an array of Field objects containing one object per accessible public field.

A public field is accessible if it is a member of either:

- this class
- a superclass of this class
- an interface implemented by this class
- an interface extended from an interface implemented by this class

```
 1
 2      import java.lang.reflect.*;
 3
 4      class What {
 5              public int publicVar;;
 6              private int privateVar;;
 7              static int staticVar;;
 8
 9              static void printFieldNames(Object o) {
10                      Class c = o.getClass();
11                      Field[] publicFields = c.getFields();
12                      for (int i = 0; i < publicFields.length; i++) {
13                              String fieldName = publicFields[i].getName
14                              Class typeClass = publicFields[i].getType(
15                              String fieldType = typeClass.getName();
16                              System.out.println("\tName: " + fieldName
17                                  ", Type: " + fieldType);
18                      }
19              }
20
21              public static void main(String[] args) {
22                      String aS = new String();
23                      Thread aT = new Thread();
24                      What aW = new What();
25                      System.out.println("String: ");
26                      printFieldNames(aS);
27
28                      System.out.println("Thread: ");
29                      printFieldNames(aT);
30
31                      System.out.println("What: ");
32                      printFieldNames(aW);
33              }
34      }
```

Source Code: Src/17/What.java

Result:

```
% java What
tring:
        Name: serialVersionUID, Type: long
        Name: CASE_INSENSITIVE_ORDER, Type: java.util.Comparator
Thread:
        Name: MIN_PRIORITY, Type: int
        Name: NORM_PRIORITY, Type: int
        Name: MAX_PRIORITY, Type: int
What:
        Name: publicVar, Type: int
```

15.4. Getting Values

A development tool such as a debugger, must be able to obtain field values. This is a three-step process:

1. Create a Class object.
2. Create a Field object by invoking getField on the Class object.
3. Invoke one of the get methods on the Field object

```
 1
 2      import java.lang.reflect.*;
 3
 4      class Get {
 5              public int publicVar = 42;
 6              private int privateVar;
 7              static int staticVar;
 8
 9              static void getValue(Object o) {
10                      Class c = o.getClass();
11                      Integer value;
12                      try {
13                              Field publicVarField = c.getField("publicV
14                              value = (Integer) publicVarField.get(o);
15                              System.out.println("value: " + value);
16                      } catch (NoSuchFieldException e) {
17                              System.out.println(e);
18                              } catch (SecurityException e) {
19                              System.out.println(e);
20                              } catch (IllegalAccessException e) {
21                              System.out.println(e);
22                              }
23              }
24
25              public static void main(String[] args) {
26                      Get aG = new Get();
27
28                      System.out.println("Get: ");
29                      getValue(aG);
30              }
31      }
```

Source Code: Src/17/Get.java

```
% java Get
Get:
value: 42
```

15.5. Setting Values

Some debuggers allow users to change field values during a debugging session. A tool that has this capability, must call one of the Field class's set methods.

1. To modify the value of a field, perform the following steps: Create a Class object. For more information, see the section Retrieving Class Objects.
2. Create a Field object by invoking getField on the Class object.
3. Class Fields shows you how. Invoke the appropriate set method on the Field object.

The Field class provides several set methods. Specialized methods, such as setBoolean and setInt, are for modifying primitive types. If the field you want to change is an object invoke the set method. It is possible to set to modify a primitive type, but the appropriate wrapper object for the value parameter must be used.

```
 1
 2      import java.lang.reflect.*;
 3
 4      class Set {
 5              public int publicVar = 42;
 6              private int privateVar;
 7              static int staticVar;
 8
 9              static void setValue(Object o) {
10                      Class c = o.getClass();
11                      Integer value;
12                      try {
13                              Field publicVarField = c.getField("publicV
14                              publicVarField.set(o, new Integer(24) );
15                      } catch (NoSuchFieldException e) {
16                              System.out.println(e);
17                              } catch (SecurityException e) {
18                              System.out.println(e);
19                              } catch (IllegalAccessException e) {
20                              System.out.println(e);
21                              }
22              }
23
24              public static void main(String[] args) {
25                      Set aS = new Set();
26
27                      System.out.println("before: aS.publicVar = "
28                              + aS.publicVar);
29                      setValue(aS);
30                      System.out.println("after:  aS.publicVar = "
31                              + aS.publicVar);
32              }
33      }
```

Source Code: Src/17/Set.java

```
% java Set
before: aS.publicVar = 42
after:  aS.publicVar = 24
```

15.6. Obtaining Method Information

To find out what public methods belong to a class, invoke the method named getMethods. The array returned by getMethods contains Method objects. This can be used to uncover a method's name, return type, parameter types, set of modifiers, and set of throwable exceptions. All of this information would be useful to write a class browser or a debugger. A metod can be called with Method.invoke.

1. It retrieves an array of Method objects from the Class object by calling getMethods.
2. For every element in the Method array, the program:
   a. retrieves the method name by calling getName
   b. gets the return type by invoking getReturnType
   c. creates an array of Class objects by invoking getParameterTypes
3. The array of Class objects created in the preceding step represents the parameters of the method. To retrieve the class name for every one of these parameters, the program invokes getName against each Class object in the array.

```
 1
 2      import java.lang.reflect.*;
 3
 4      class Show {
 5              public int publicVar = 42;
 6              private int privateVar;
 7              static int staticVar;
 8
 9              public static int HPclassM(String aString)    {
10                                      return 1;
11              }
12              public int        HPobjectM(int i, Boolean aBoolean) {
13                                      return 1;
14              }
15
16              void showMethods(Object o) {
17                      Class c = o.getClass();
18                      Method[] theMethods = c.getMethods();
19                      for (int i = 0; i < theMethods.length; i++) {
20                              String methodString = theMethods[i].getNa
21                              System.out.println("Name: " + methodStrin
22                              String returnString =
23                              theMethods[i].getReturnType().getName();
24                              System.out.println("   Return Type: " + re
25                              Class[] parameterTypes = theMethods[i].get
26                              System.out.print("   Parameter Types:");
27                              for (int k = 0; k < parameterTypes.length
28                                      String parameterString = parameter
29                                      System.out.print(" " + parameterSt
30                                      }
31                              System.out.println();
32                      }
33              }
34
```

```
35              public static void main(String[] args) {
36                      Show aS = new Show();
37                      System.out.println("Show: ");
38                      aS.showMethods(aS);
39
40              }
41      }
```

Source Code: Src/17/Show.java

```
% java Show
Show:
Name: HPclassM
   Return Type: int
   Parameter Types: java.lang.String
Name: main
   Return Type: void
   Parameter Types: [Ljava.lang.String;
 ...
Name: wait
   Return Type: void
   Parameter Types: long int
Name: HPobjectM
   Return Type: int
   Parameter Types: int java.lang.Boolean
```

15.7. Invoking Methods

A debugger should hallows a user to select and then invoke methods during a debugging session. Since it is not know at compile time which methods the user will invoke, the method name can not be hardcoded in the source code.

1. Create a Class object that corresponds to the object whose method you want to invoke. See the section Retrieving Class Objects for more information.
2. Create a Method object by invoking getMethod on the Class object. The getMethod method has two arguments: a String containing the method name, and an array of Class objects. Each element in the array corresponds to a parameter of the method you want to invoke. For more information on retrieving Method objects, see the section Obtaining Method Information
3. Invoke the method by calling invoke. The invoke method has two arguments: an array of argument values to be passed to the invoked method, and an object whose class declares or inherits the method.

```
 1
 2      import java.lang.reflect.*;
 3
 4      class Invoke {
 5              public int publicVar = 42;
 6              private int privateVar;
 7              static int staticVar;
 8
 9              public String objectM(Boolean aBoolean, Integer i) {
10                      System.out.println("objectM: i        = "
11                              + i);
12                      System.out.println("objectM: aBoolean = "
13                              + aBoolean);
14                      return "Alles Verloren, alles Geboren ...";
15              }
16
17              void invokeMethod(Object o) {
18                      Class c = Invoke.class;
19                      Class[] parameterTypes = new Class[] {Boolean.clas
20                                                           Integer.clas
21                      Method objectM;
22                      Object[] arguments = new Object[] {new Boolean(tru
23                                                         new Integer(2)
24                      try {
25                              objectM = c.getMethod("objectM", parameter
26                              System.out.println(
27                                      (String) objectM.invoke((Invoke)o,
28                      } catch (NoSuchMethodException e) {
29                              System.out.println(e);
30                      } catch (IllegalAccessException e) {
31                              System.out.println(e);
32                      } catch (InvocationTargetException e) {
33                              System.out.println(e);
34                      }
35
36              }
37
```

```
38              public static void main(String[] args) {
39                      Invoke aS = new Invoke();
40                      System.out.println("Invoke: ");
41                      aS.invokeMethod(aS);
42
43              }
44      }
```

Source Code: Src/17/Invoke.java

```
% java Invoke
Invoke:
objectM: i        = 2
objectM: aBoolean = true
Alles Verloren, alles Geboren ...
```

```
 1
 2      import java.lang.reflect.*;
 3
 4      class Stubs {
 5              public int publicVar = 42;
 6              private int privateVar;
 7              static int staticVar;
 8
 9              public String objectM(Boolean aBoolean, Integer i) {
10                      System.out.println("objectM: i        = "
11                              + i);
12                      System.out.println("objectM: aBoolean = "
13                              + aBoolean);
14                      return "Alles Verloren, alles Geboren ...";
15              }
16
17              static void invokeMethod(Object o) {
18                      System.out.println("invokeMethod: ");
19                      Class c = o.getClass();
20                      System.out.println("class: " + c );
21                      Class[] parameterTypes = new Class[] {String.class
22
23                      Method objectM;
24                      Object[] arguments = new Object[] {new String("Mer
25                      try {
26                              objectM = c.getMethod("sayHello", paramete
27                              System.out.println(
28                                      (String) objectM.invoke((HelloInte
29                      } catch (NoSuchMethodException e) {
30                              System.out.println(e);
31                      } catch (IllegalAccessException e) {
32                              System.out.println(e);
33                      } catch (InvocationTargetException e) {
34                              System.out.println(e);
35                      }
36
37              }
```

```
38
39              public static void main(String[] args) {
40                      try {
41                              HelloInterface aHello = new HelloImplement
42                              invokeMethod(aHello);
43                              System.exit(0);
44                      } catch ( Exception e )         {
45                              e.printStackTrace();
46                      }
47
48              }
49      }
```

Source Code: Src/17/Stubs.java

16. Under Construction

17. Under Construction

18. Under Construction

19. Under Construction

20. Under Construction

21. Homework 1

Posted: May/10/2018
Due: September/3/2018 24.00

All homework solutions are due September/3/2018 24.00. I recommend to submit at least one version of all homework solutions long before due date.

This course uses the try command to allow students to electronically submit their work.

IMPORTANT: EVERY STUDENT has to register.

All submission and registrations must be done from **glados.cs.rit.edu**.

21.1. Registering Course Database

Open a terminal window and from a Unix prompt and type the following commands: This is an example from my desktop. My computer is named *spiegel*, and my account name is *guest*. You need to type in the text in bold.

```
spiegel% ssh hpb@glados.cs.rit.edu
Password: # i typed in my password followed by a return
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-116-generic x86_64)
  ... more text was printed
hpb[41]% try hpb-grd register /dev/null
```

If *glados.cs.rit.edu* does not work try *queeg.cs.rit.edu*.

The program should print out something like this: