UNIVERSITY
OF LONDON

Graphical object-oriented and internet programming in Java Volume 2

T. Blackwell

CO2220

2009

Undergraduate study in
Computing and related programmes

Goldsmiths
UNIVERSITY OF LONDON

This guide was prepared for the University of London by:

Tim Blackwell

This guide was produced by:

Sarah Rauchas, Department of Computing, Goldsmiths, University of London.

This is one of a series of subject guides published by the University. We regret that due to pressure of work the author is unable to enter into any correspondence relating to, or arising from, the guide. If you have any comments on this subject guide, favourable or unfavourable, please use the form at the back of this guide.

In this and other publications you may see references to the 'University of London International Programmes', which was the name of the University's flexible and distance learning arm until 2018. It is now known simply as the 'University of London', which better reflects the academic award our students are working towards. The change in name will be incorporated into our materials as they are revised.

University of London
Publications Office
32 Russell Square
London WC1B 5DN
United Kingdom

london.ac.uk

Published by: University of London

Contents

Preface

Introduction

The course is split into two parts, with a separate volume for each part. This volume constitutes the second part of the course. Part I (Volume 1) covers Object-Oriented programming in Java, graphical user interfaces and event-driven systems. Part II, in this volume, is concerned with the principles of client-server computing, techniques of interconnectivity in Java and interactive web-based computing systems.

Aims

The course as a whole aims to give students an insight into the object-oriented approach to the design and implementation of software systems. The course also considers specific features of the programming language Java, in particular, graphical interfaces and event driven applications.

The second part of the course, which is covered in this volume, is intended to give students the necessary background to understand the technical software aspects of how computers communicate across the internet. Students will be introduced to the underlying principles of client-server computing systems and will gain the required conceptual understanding, knowledge and skills to enable them to produce simple web-based computing systems in Java.

Objectives

1. To re-enforce students' knowledge of object-oriented programming in Java. (Part I)
2. To introduce students to the notion of graphical user interfaces. (Part I)
3. To introduce students to the notion of event-driven systems. (Part I)
4. To teach students the principles of client-server computing. (Part II)
5. To introduce the main techniques for interconnectivity in Java. (Part II)
6. To produce students able to develop rudimentary interactive web-based computing systems. (Part II)

Learning outcomes

On completion of this course students should be able to:

1. Analyse and represent problems in the object-oriented programming paradigm. (Part I)
2. Design and implement object-oriented software systems. (Part I)
3. Build an event-driven graphical user interface. (Part I)
4. Explain the main principles for client-server programming. (Part II)
5. Design and implement rudimentary client side system. (Part II)
6. Design and implement a rudimentary server-side system. (Part II)
7. Integrate his or her knowledge and skills to produce a rudimentary web-based application. (Part II)

Assessment

Coursework contributes 20 per cent of the final mark on the complete unit. An unseen examination paper will contribute 80 per cent of the final mark.

Important note. The information given above is based on the examination structure used at the time this guide was written. Please note that subject guides may be used for several years. Because of this we strongly advise you to always check the current Regulations for relevant information about the examination. You should also carefully check the rubric/instructions on the paper you actually sit and follow those instructions.

How to use this subject guide

This subject guide is not a self-contained account, but is a companion to the course text *Head First Java* 2nd edition (*HFJ*) by Kathy Sierra and Bert Bates. **It is essential that you obtain this book**.

It will also be helpful to have access to *Java Network Programming* (*JNP*) by Elliotte Rusty Harold.

There are a number of other books listed in the section below called **Reading**, which expand on a number of topics and you are advised to deepen your understanding by referring to these additional texts where directed.

There are eleven chapters in this guide.Most chapters are in two parts. The first part is based on a number of readings from *HFJ*. The second part is devoted to programming. Here you will find programming examples and activities based on the material covered in the first part of the chapter.

In short, the chapters are comprised of

- readings from *HFJ* (not Chapter 3), followed by a summary of the key points from each reading

- a bulleted chapter summary
- programming
- learning outcomes.

It is important that you read *HFJ* when directed, and then read the commentary to check that you have understood the main points. The commentaries are not sufficient in themselves. You must refer to *HFJ* and you are recommended to engage with the many interesting activities that the authors suggest. The chapter summaries collect together the main points. All chapter summaries are reproduced in the revision chapter. These summaries can be used to ensure that you are on top of the material, and as a revision guide.

The programming sections are an integral part of the course. Aside from the program examples, you will find programming activities. You **must** attempt these activities before reading on. The activities are followed by a programming solution. Please realise that there is rarely a unique solution to a programming exercise, so do not feel disheartened if your solution differs from mine. However you should read my program, and the accompanying commentary, to understand my solution. Moreover, new material, especially concerning Java graphics, will be found in the commentaries.

The CD-ROM The accompanying CD-ROM provides all the source code and compiled programs. Many activities are centred on making a drawing or an animation. You should run these programs before attempting the activity so that you can see what to aim for (but do not peek at the code!).

The CD-ROM contains the following:

- an Index which serves to navigate through the folders
- demonstration programs
- source and compiled code for all programming examples and exercises
- source and compiled code for Goo, a special animation package developed for this course
- the Goo API, which is the reference document for the Goo code
- the Java API, which is the reference document for the Java code.

You may wish to develop your programs with an integrated program development environment (IDE) such as Eclipse. It is beyond the scope of this course to show you how to use Eclipse but it is well worth investing some time in learning to use this valuable programming tool yourself. Eclipse can be downloaded for free from `http://www.eclipse.org/`. This guide tells you how to write code in a simple editor and compile and run from the command line. However an IDE such as Eclipse simplifies many programming tasks and greatly helps with debugging.

At the end of each volume, there is a revision guide that summarises the main concepts you should have acquired from each chapter, and also gives you some sample examination papers that can guide some of your study.

Reading

The following is a list of essential and supplementary reading.

Essential reading

Head First Java (second edition), Kathy Sierra and Bert Bates (Sebastopol, Calif.: O'Reilly 2005) [ISBN 0596009208 (pbk)].

Recommended reading

Java in a Nutshell, David Flanagan (Sebastopol, Calif.: O'Reilly, 2005) [ISBN 0596007736].
Learning Java, Patrick Niemeyer and Jonathon Knudsen (O'Reilly, 2005).
Effective Java (second edition), Joshua Bloch (Upper Saddle River, NJ; Harlow: Addison Wesley, 2008) [ISBN 0321356683 (pbk)].
Java Network Programming, Elliotte Rusty Harold (Sebastopol, CA; Farnham: O'Reilly, 2005) [ISBN 0596007213 (pbk)].
Java Cookbook, Ian F. Darwin (O'Reilly Media Inc., 2004) [ISBN 0596007019; 978-05960007010].

Notation

`Java keywords, source code, variable names, method names and other source code` are printed in typewriter font. **Filenames, directories and the command line** in bold type. Three dots, . . . , denotes omitted source code in a code excerpt. Concepts and things, where they are distinguishable from their representative Java names (classes , interfaces, method names . . .), are printed in normal type.

Before you do anything else

Insert the CD-ROM into your computer, open the **demo** folder, and run ManyWorldsApp.

You are watching the flight of clouds of blobs (left hand world) and flapping polygons (right hand world). The two worlds are connected by two 'worm holes'. The exit and entrances to the worm holes are shown by the dark grey discs. Objects from either world, when flying over the exit disc in their world will 'fall' into the other world, appearing at the exit disc.

You will observe that the objects are mini-goo animations, similar to those you coded in Volume 1, except that these mini animations move across the screen to form a larger animation. You can think of each type of mini-animation as a creature.

In this demo, the two worlds are launched from a single application on a single computer.

Imagine instead that the worlds are running on separate machines, connected by worm holes that extend across the Earth.

Imagine that there are many worlds connected by a tangle of wormholes, and many different species of goo creature populating the worlds. You are watching your world, awaiting the arrival of an exotic species. Perhaps you are designing your own species, introducing creatures of this species into your world, and letting them disperse throughout the goo'niverse.

Chapter 1

Advantages

Essential reading

JNP Chapter 1.

1.1 Introduction

Java is the first language designed with networks in mind. Java provides solutions to many network problems such as platform independence, security and international character sets. And thanks to the extensive API, little code is needed even for full-scale applications. In brief, some of the peculiar advantages of Java are:

1. Java applications are safer than off-the-shelf software.
2. Network programs in Java are (relatively) easy to write.
3. Java uses threads rather than processes; this is important for scalable web servers.
4. Java has an exception-handling mechanism; this is important for web applications that need to run continuously.
5. Java is object oriented, and all the software engineering principles that you saw in Volume I can be employed.
6. Java is a full language, rather than a scripting language, and is therefore very versatile. Java has only a small execution-speed disadvantage compared to C++, which is another object language.
7. Java has an extensive Java API, especially for networking. The important packages in this regard are:
   - java.io
   - java.nio
   - java.net
   - java.applet
   - java.rmi
   - javax.servlet
   - javax.servelt.http
   - java.lang.Thread.

1.2 Important networking features of the Java Programming Language

There are four important parts of the JPL which we need to find out about: Threads, Java IO, Serialization and Exception Handling. These features play a vital role in Java network programming. Additionally, some further work on Swing will enhance your graphical interfaces.

This volume begins therefore with these Java techniques (which aren't restricted to networks, but are applicable to all Java programming). The course then continues with some fundamental networking: how to connect to a host, writing an internet browser, a spider and peer-to-peer messaging. The course ends with an introduction to distributed computing.

But before we can do any of that, there are some general programming aspects that we need to look at.

1.3 Learning outcomes

By the end of this chapter, the relevant reading and activities, you should be able to:

- describe the advantages of Java as a programming language
- describe the main networking features of Java: threads, Java IO, serialisation and exception handling.

Chapter 2

Statics

Essential reading

HFJ Chapter 10 (not number formatting or static imports).

2.1 Introduction

This chapter covers some aspects of 'general programming' in Java. By this we mean useful, non-object Java techniques. Many useful programs can be written as a sequence of procedures (method calls) and do not need sophisticated data structures. Top-down programming is sufficient in many situations.

2.2 Statics

Reading: pp. 273–281 of *HFJ*.

A mathematical function such as $ROUND$ does the same thing every time it is invoked. There is no associated state. These functions are pure behaviour. Such functions are represented by static methods in Java.

A *static* method can be invoked without any instances of the method's class on the heap. Static methods are used for utility methods that do not depend on a particular instance variable value. This means that static methods are not associated with any particular instance of that class.

A static method (such as `main`) cannot access a non-static (i.e. instance) method. If you write a class with only static methods then it usually makes no sense to instantiate objects of that class. You can prevent instantiating by marking the constructor as `private`.

There is only one copy of a static variable and it is shared by all members of the class. A static method can access a static variable.

2.3 Final

Reading: pp. 282–284 of *HFJ*.

Some quantities such as the speed of light, or the value of pi just do not change. Constants such as these are marked `static final`.

A final static variable is either initialised when it is declared, or in a static initializer block,

```
static{
    SPEED_OF_LIGHT = 299792458;
}
```

Static finals are conventionally named in uppercase with underscores separating words. Note that this is a convention, rather than contributing to the semantics of the language.

A final variable cannot be changed after it has been initialised. An instance variable can also be marked final. It can only be initialised in the constructor or where it is declared.

Methods and classes may also be final. A final method cannot be overridden and a final class cannot be extended.

2.4 Maths

Reading: pg. 286 of *HFJ*.

`java.lang.Maths` has two static final variables (constant variables, that is), and various static methods. The methods correspond to mathematical functions such as $SQUAREROOT$, COS and $ROUND$.

2.5 Wrapping a Primitive

Reading: pg. 287 of *HFJ*.

All primitive values can be wrapped into objects, and wrapper reference classes can be unwrapped:

```
Integer intObj = new Integer(i);
...
int j = Integer.intValue(intObj);
```

2.6 Autoboxing

Reading: pp. 288–291 of *HFJ*.

Java 5 and beyond supports autoboxing: the automatic wrapping and unwrapping of values in method arguments, return values, Boolean expressions, operations on numbers, assignments, list insertion and removal . . . in fact almost anywhere a primitive or a wrapper type is expected.

For example,

```
Integer i = 42;
i++;
```

```
...
Integer j = 5;
...
Integer k = i + j;
```

Note that `+, ++` operators are NOT defined to act on objects. In fact the compiler will convert wrapper objects to their primitive types before the operations are applied.

2.7 Wrapper methods

Reading: pp. 292–293 of *HFJ*.

Wrappers have static utility methods e.g.

```
String s = "2997792458";
int c = Integer.parseInt(s);
...
double kmPerSec = c / 1000.0;
s = Double.toString(kmPerSec);
```

2.8 Number formatting

Reading: pp. 294–301 of *HFJ*.

In a gesture of conciliation towards C programmers, Java 5 has introduced number formatting in print statements. The syntax is similar to C and C++'s `printf`.

2.9 Dates

Reading: pp. 302–306 of *HFJ*.

`Date()` is fine for getting today's date, as in `Date today = new Date();` but use `Calendar` for date manipulation. In fact `Calendar` is abstract. You can obtain an instance of a concrete subclass by calling a static method:

```
Calendar c = Calendar.getInstance();
```

2.10 Static imports

Reading: pp. 307–309 of *HFJ*.

A static class or a static variable can be imported in an effort to save typing. This can make code easier to read, but may create name conflicts. Only use a static import if the static member is called often from within your class, and you are sure there are no naming collisions.

2.11 Summary

- A static method can be called using the class name rather than an object reference variable.
- A static method can be invoked without any instances of the method's class on the heap.
- Static methods are used for utility methods that do not depend on a particular instance variable value.
- Static methods are not associated with any particular instance of that class.
- A static method (such as `main`) cannot access a non-static (i.e. instance) method.
- Mark the constructor as `private` if you wish to prevent clients instantiating the class.
- There is only one copy of a static variable and it is shared by all members of the class.
- A static method can access a static variable.
- Java constants are marked `static final`.
- A final static variable is either initialised when it is declared, or in a static initializer block.
- Static finals are conventionally named in uppercase with underscores separating words.
- A final variable cannot be changed after it has been initialised.
- An instance variable can also be marked final. It can only be initialised in the constructor or where it is declared.
- A final method cannot be overridden.
- A final class cannot be extended.
- `java.lang.Math` only has static methods. Some very useful methods in this class are: `random(), abs(), round(), min(), max()`. These, and others are listed in the API.
- All primitive values can be wrapped into objects (`Integer intObj = new Integer(i);` and unwrapped: `(int i = Integer.intValue(intObj);`.
- Java 5 and beyond supports autoboxing: the automatic wrapping and unwrapping of values.
- Wrappers have static utility methods e.g. `Integer.parseInt(s)` and `Double.toString(kmPerSec);`
- A format string uses its own special little language; the format string enables precise control of number printing.
- Date is fine for getting today's date, but use `Calendar` for date manipulation.
- Calendar is abstract: get an instance of a concrete subclass like this: `Calendar c = Calendar.getInstance();`
- Static imports save typing, but can lead to name collisions.
- Static methods encourage the procedural style of programming.

2.12 Programming

The Math class has many useful operations on numbers. Suppose we wish to define a similar utility class for *vectors*. Here are some common vector functions:

An n-dimensional vector **a** is a list of real (components), $\mathbf{a} = [a_1, a_2 \ldots a_n]$.

A vector can be multiplied by a number, s, $s\mathbf{a} = [sa_1, sa_2 \ldots sa_n]$.

The vector *dot product* is $\mathbf{a} \cdot \mathbf{b} = [a_1b_1 + a_2b_2 + \ldots + a_nb_n]$;

and the *cross product* in three dimensions is defined as
$\mathbf{c} = \mathbf{a} \times \mathbf{b} = [a_2b_3 - a_3b_2,\ a_3b_1 - a_1b_3,\ a_1b_2 - a_2b_1]$.

Vectors can be *added* and *subtracted* (component by component).

The `length` of a vector is calculated from Pythagoras' formula: $|\mathbf{a}| = \sqrt{\mathbf{a}.\mathbf{b}}$

and vectors can be *normalised* to unit length. If $\hat{\mathbf{a}}$ is the normalised form of **a** then $|\hat{\mathbf{a}}| = 1$.

Given point $A = (a_1, a_2 \ldots a_n)$ then $\mathbf{a} = [a_1, a_2 \ldots a_n]$ connects O to A i.e. $\mathbf{a} =$ `OA`. The *distance* between A and B is therefore $|\mathbf{b} - \mathbf{a}|$.

The *angle* θ between **a** and **b** can calculated from the relation: $\mathbf{a} \times \mathbf{b} = |\mathbf{a}||\mathbf{b}|cos\theta$

Learning activity

Write a vector utility class, `VecMath`. The class should implement the vector functions listed above.

2.13 Vector Maths

```
package numbersandstatics;

import static java.lang.Math.*;

public class VecMath {

  private VecMath() {
  }

  public static double length(double[] a) {

    return sqrt(dot(a, a));
  }

  public static void normalise(double[] a) {

    double mag = length(a);
    for (int i = 0; i < a.length; i++) {
      a[i] = a[i] / mag;
    }
```

```
}

public static double distance(double[] a, double[] b) {

  return length(subtract(a, b));
}

public static double[] add(double[] a, double[] b) {

  int numCpts = a.length;
  double[] c = new double[numCpts];
  for (int i = 0; i < numCpts; i++) {
    c[i] = a[i] + b[i];
  }
  return c;
}

public static double[] subtract(double[] a, double[] b) {

  int numCpts = a.length;
  double[] c = new double[numCpts];
  for (int i = 0; i < numCpts; i++) {
    c[i] = a[i] - b[i];
  }
  return c;
}

public static double[] mult(double scalar, double[] a) {

  double[] b = new double[a.length];
  for (int i = 0; i < a.length; i++) {
    b[i] = scalar * a[i];
  }
  return b;
}

public static double[] cross(double[] a, double[] b) {

  if (a.length == b.length && b.length == 3) {
    double[] c = new double[3];
    c[0] = a[1] * b[2] - a[2] * b[1];
    c[1] = a[2] * b[0] - a[0] * b[2];
    c[2] = a[0] * b[1] - a[1] * b[0];
    return c;
  } else
    return new double[0];
}

public static double dot(double[] a, double[] b) {

  int numCpts = a.length;
  double result = 0.0;
  for (int i = 0; i < numCpts; i++) {
    result += a[i] * b[i];
  }
  return result;
}

public static double angle(double[] a, double[] b) {
```

```
    return acos(dot(a, b) / (length(a) * length(b)));
  }
}
```

The methods are a straightforward implementation of the common vector operations. The constructor has been marked private to prevent instantiations of `VecMaths`, and `java.lang.Math` has been statically imported. (This static import is only of marginal benefit since Math is only called twice.)

Learning activity

Write a test class for your VecMath, and test each method. Numerical output should contain four decimal places. Use static imports wherever appropriate.

2.14 Vector maths test program

```
package numbersandstatics;

import static java.lang.Math.*;
import static java.lang.System.out;
import static java.lang.String.format;
import static numbersandstatics.VecMath.*;

public class VecMathTest {

  public static String vecFormat(String formatStr, double[] a) {

    String s = "(";
    for (int i = 0; i < a.length - 1; i++) {
      s += format(formatStr, a[i]) + ", ";
    }
    s += format(formatStr, a[a.length - 1]) + ")";
    return s;
  }

  public static double toDegrees(double radians) {

    return (180 / PI) * radians;
  }

  public static void main(String[] args) {

    // initialise two vectors
    double[] a = { 5, 0, 0 };
    double[] b = { 5 * cos(PI / 6), 5 * sin(PI / 6), 0 };

    // format String
    String formatStr = "%.4f";

    // print a = OA and b = OB
    out.println("a = " + vecFormat(formatStr, a) + ", " + "b = "
        + vecFormat(formatStr, b));

    // print lengths of a and b
```

```
        out.println("|a| = " + format(formatStr, length(a)) + ", |b|
            = "
             + format(formatStr, length(b)));

        // find unit vector pointing along b
        double[] c = { 3, 4 };
        normalise(c);
        out.println("c hat = " + vecFormat(formatStr, c));

        // is it really a unit vector?
        out.println("length of c hat = " + format(formatStr, length(c
            )));

        // vector addition and subtraction
        c = add(a, b);
        out.println("a + b = " + vecFormat(formatStr, c));
        c = subtract(a, b);
        out.println("a - b = " + vecFormat(formatStr, c));

        // scalar multiplication
        c = mult(0.5, b);
        out.println("0.5b = " + vecFormat(formatStr, c));

        // dot product
        out.println("a.b = " + format(formatStr, dot(a, b)));

        // distance between A and B
        out.println("dist AB = " + format(formatStr, distance(a, b)))
            ;

        // cross product
        c = cross(a, b);
        out.println("c = a x b = " + vecFormat(formatStr, c));

        // is c at 90 degrees to a and b?
        out.println("a.c = " + dot(a, c) + ", b.c = " + dot(b, c));

        // angle AOB
        double angle = toDegrees(angle(a, b));
        out.println("angle AOB  = " + format(formatStr, angle));
    }
}
```

`main` initialises two test vectors and systematically calls each method in VecMath. Some static imports shorten the code; probably justifiable in a small program such as this. However one problem did become apparent when coding: how to format the elements of a numerical array? The solution used here is to write a utility method that applies String.format to each element in turn. In order to prevent naming ambiguity, this method was not named `format`.

Learning activity

The above tests were too conservative. There are two problems with the implementations within VecMath which could easily cause a client program to crash. Can you spot them? Write more tests to demonstrate which methods are unsafe and what happens when they are called by a careless client.

2.15 Better test program

```
package numbersandstatics;

import static java.lang.Math.*;
import static java.lang.System.out;
import static java.lang.String.format;
import static numbersandstatics.VecMath.*;

public class VecMathUnsafeTest {

  final static int TEST_0 = 0;
  final static int TEST_1 = 1;
  final static int TEST_2 = 2;

  public static String vecFormat(String formatStr, double[] a) {

    String s = "(";
    for (int i = 0; i < a.length - 1; i++) {
      s += format(formatStr, a[i]) + ", ";
    }
    s += format(formatStr, a[a.length - 1]) + ")";
    return s;
  }

  public static double toDegrees(double radians) {

    return (180 / PI) * radians;
  }

  public static void main(String[] args) {

    int test = TEST_0;
    if (args.length > 0)
      test = Integer.parseInt(args[0]);

    double[] a = new double[] { 5, 0, 0 };
    double[] b = new double[] { 5 * cos(PI / 6), 5 * sin(PI / 6),
        0 };

    switch (test) {
    case TEST_0:
      b = new double[] { 5 * cos(PI / 6), 5 * sin(PI / 6), 0 };
      break;
    case TEST_1:
      a = new double[] { 5, 0, 0, 1 };
      break;
    case TEST_2:
      b = new double[] { 0, 0, 0 };
      break;
    default:
    }

    // format String
    String formatStr = "%.4f";

    // print a = OA and b = OB
```

```
    out.println("a = " + vecFormat(formatStr, a) + ", " + "b = "
        + vecFormat(formatStr, b));

    // print lengths of a and b
    out.println("|a| = " + format(formatStr, length(a)) + ", |b|
        = "
        + format(formatStr, length(b)));

    // find unit vector pointing along b
    double[] c = {3, 4};
    normalise(c);
    out.println("c hat = " + vecFormat(formatStr, c));

    // is it really a unit vector?
    out.println("length of c hat = " + format(formatStr, length(c
        )));

    // vector addition and subtraction
    c = add(a, b);
    out.println("a + b = " + vecFormat(formatStr, c));
    c = subtract(a, b);
    out.println("a - b = " + vecFormat(formatStr, c));

    // scalar multiplication
    c = mult(0.5, b);
    out.println("0.5b = " + vecFormat(formatStr, c));

    // dot product
    out.println("a.b = " + format(formatStr, dot(a, b)));

    // distance between A and B
    out.println("dist AB = " + format(formatStr, distance(a, b)))
        ;

    // cross product
    c = cross(a, b);
    out.println("c = a x b = " + vecFormat(formatStr, c));

    // is c at 90 degrees to a and b?
    out.println("a.c = " + dot(a, c) + ", b.c = " + dot(b, c));

    // angle AOB
    double angle = toDegrees(angle(a, b));
    out.println("angle AOB  = " + format(formatStr, angle));
  }
}
```

2.15.1 Test report

Three tests were performed by supplying *0, 1, 2* as an argument to the Java interpreter (i.e. by typing **(java numbersandstatics/VecMathUnsafeTest 0** at the command line).

1. TEST 0

```
a = (5.0000, 0.0000, 0.0000), b = (4.3301, 2.5000, 0.0000)
|a| = 5.0000, |b| = 5.0000
b hat = (0.8660, 0.5000, 0.0000)
```

```
length of b hat = 1.0000
a + b = (9.3301, 2.5000, 0.0000)
a - b = (0.6699, -2.5000, 0.0000)
0.5b = (2.1651, 1.2500, 0.0000)
a.b = 21.6506
dist AB = 2.5882
c = a x b = (0.0000, 0.0000, 12.5000)
a.c = 0.0, b.c = 0.0
angle AOB  = 30.0000
```

Everything is working fine.

2. TEST 1

```
a = (5.0000, 0.0000, 0.0000, 1.0000), b = (4.3301, 2.5000, 0.0000)
|a| = 5.0990, |b| = 5.0000
b hat = (0.8660, 0.5000, 0.0000)
length of b hat = 1.0000
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
at numbersandstatics.VecMath.add(VecMath.java:35)
at numbersandstatics.VecMathUnsafeTest.main(VecMathUnsafeTest.java:70)
```

This test crashes the program. The bug is traced back to VecMaths line 35 where the code assumes that the two vectors have the same number of components.

3. TEST 2

```
a = (5.0000, 0.0000, 0.0000), b = (0.0000, 0.0000, 0.0000)
|a| = 5.0000, |b| = 0.0000
b hat = (NaN, NaN, NaN)
length of b hat = NaN
a + b = (5.0000, 0.0000, 0.0000)
a - b = (5.0000, 0.0000, 0.0000)
0.5b = (0.0000, 0.0000, 0.0000)
a.b = 0.0000
dist AB = 5.0000
c = a x b = (0.0000, 0.0000, 0.0000)
a.c = 0.0, b.c = 0.0
angle AOB  = NaN
```

Some numerical values have been set at `NaN` (not-a-number). This 'value' is the result of dividing by zero. Ensuing calculations will reveal surprising results. The divide by zero error happens whenever a `normalise` and `angle` is called with a vector of zero length.

We shall see one way of fixing these runtime errors in the next chapter.

2.16 Learning outcomes

By the end of this chapter, the relevant reading and activities, you should be able to:

- describe the advantages of Java as a programming language
- describe the main networking features of Java: threads, Java IO, serialisation and exception handling
- explain how a static method can be called or invoked
- describe what static methods are used for

- explain the term **static final**
- describe how static final variables are initialised and how they are usually named
- understand that a final method cannot be overridden and that a final class cannot be extended
- describe how primitive values can be wrapped into objects and how wrapper reference type classes can be unwrapped
- explain what autoboxing is
- understand that + and ++ operators are **not** defined to act on objects
- understand that wrappers have static utility methods
- describe how number formatting works
- understand how a static class or a static variable can be imported, the advantages of this, as well as the limitations
- describe how a static method can be called using the class name, or invoked without any instances of the method's class on the heap
- understand that static methods are used for utility methods that do not depend on a particular instance variable value
- understand that static imports can lead to name collisions
- understand that static methods encourage the procedural style of programming.

Chapter 3

Exceptions

Essential reading

HFJ Chapter 11.

3.1 Introduction

Most of the time you should aim to write safe code so that clients of your classes are not surprised by the messages they get back, and so that your classes do not crash their program.

However, certain programming activities such as asking the operating system to do something (e.g. pause execution for 100 ms) or interfacing with a device (print a file), or connecting to a network are inherently dangerous because your program cannot know exactly what may happen.

Flaws in your own code are known as bugs and we hope that OO techniques can minimise these. But some runtime errors are outside your control. Luckily Java includes a mechanism so that you can prepare for the unexpected and unusual at runtime with special *exception handling* code.

3.2 Catch!

Reading: pp. 319–328 of *HFJ*.

The compiler makes sure that checked-exceptions are caught by your code. For example, the API tells us that the method prototype for `read(byte[] b)` of `java.io.InputStream` is:

```
public int read(byte[] b) throws IOException
```

This means that read throws an exception object which the calling method must 'catch':

```
try{
   int numRead = inputStream.read(byte[] b);
}
catch (IOExeption e){
   System.out.println(e);
}
```

The risky call is placed in a try block. The accompanying catch block contains emergency code in case something goes wrong. The JVM will execute this code in that eventuality. Here, the catch just prints the exception to the terminal, but normally you would want to do something (e.g. wait a while and then try and read from the input stream again). Occasionally you may wish to write a `finally` block, which will run regardless of an exception.

3.3 Multiple exceptions

Reading: pp. 329–324 of *HFJ*.

A method can throw more than one exception and they must all be caught, preferably by one catch block after another. Exceptions are polymorphic and it is possible to catch all the exceptions with a single supertype catch. Certainly a `catch (Exception e)` block will catch everything because all exceptions subclass `java.lang.Exception`, but this is not advisable because your handler will not have precise information about what went wrong. Since the JVM works its way down the list of catch blocks, it is better to place more specific catch calls (i.e. lowest subclasses in the inheritance tree) higher up the list.

An important subclass of Exception is `java.lang.RuntimeException`. Subclasses of this, such as `java.lang.ArrayIndexOutOfBoundsException` (which is thrown to indicate that an array has been accessed with an illegal index), are ignored by the compiler. Runtime exceptions are usually due to faulty code logic, rather than unpredictable or unpreventable conditions that arise when the code is running.

Exceptions should only be used for very unusual circumstances caused by interactions with the outside world, and not by the internal logic of your code.

3.4 Duck!

Reading: pp. 335–357 of *HFJ*.

The thrown exception can be ducked by your code by declaring it, as in

```
public int riskyMethod() throw IOException{

      return inputStream.read(byteArray);
  }
```

The stack diagrams on pg. 336 of *HFJ* show how the duck mechanism works, and if main ducks as well, the uncaught exception will shut the JVM down.

3.5 Relevance to network programming

Network programs must function for many hours without supervision, and are in contact with the outside world. Therefore the exception handling mechanism of Java is invaluable.

3.6 Summary

- A method can throw an exception object if something goes wrong at runtime.
- All exceptions subclass java.lang.Exception.
- The compiler does not check that possible runtime exceptions are handled in your code.
- However the compiler does care about **checked exceptions**, and insists that they are declared or wrapped in a try/catch block.
- A method throws an exception with the keyword throw followed by a new exceptions object.
- If your code calls a checked exception throwing method, you must either duck the exception or enclose the call in a try/catch block.

3.7 Programming

We noticed in the last chapter that our VecMath class could be dangerous if a client called a method with two double[]'s of different size; and if normalise() or angle() is called with a vector of zero length.

We saw from the test report that if an array is accessed outside its defined range a `java.lang.ArrayIndexOutOfBounds exception` is thrown. Division by zero (double) does not raise any exceptions, but sets the result to NaN.

Learning activity

Copy and paste length(), normalise() and dot() methods from VecMaths into a new class, VecMathException. Alter normalise() so the method throws an ArithmeticException if a division by zero is attempted. (Consult the API to find out about ArithmeticException and its superclass, RuntimeException.)

Write a test class which catches any arithmetic and array index out of bounds exceptions thrown by VecMathException.

3.8 Vector maths with exception throwing

```
package exceptionhandling;

import static java.lang.Math.*;

public class VecMathException {

  private VecMathException() {
  }

  public static double length(double[] a) {

    return sqrt(dot(a, a));
```

```
  }

  public static void normalize(double[] a) throws
      ArithmeticException {

    double mag = length(a);
    if (mag == 0)
      throw new ArithmeticException(
          "SafeVecMath.java 18: Attempt to divide by zero");
    for (int i = 0; i < a.length; i++) {
      a[i] = a[i] / mag;
    }
  }

  public static double dot(double[] a, double[] b) {

    int numCpts = a.length;
    double result = 0.0;
    for (int i = 0; i < numCpts; i++) {
      result += a[i] * b[i];
    }
    return result;
  }
}
```

3.9 Catching VecMath exceptions

```
package exceptionhandling;

import static java.lang.System.out;
import static java.lang.String.format;

import static exceptionhandling.VecMathException.*;

public class VecMathExceptionTest {

  public static String vecFormat(String formatStr, double[] a) {

    String s = "(";
    for (int i = 0; i < a.length - 1; i++) {
      s += format(formatStr, a[i]) + ", ";
    }
    s += format(formatStr, a[a.length - 1]) + ")";
    return s;
  }

  public static void main(String[] args) {

    // format String
    String formatStr = "%.4f";

    // dot product with different size vectors
    try {
      double[] a = new double[] { 1, 1, 0 };
      double[] b = new double[] { 1, 1 };
      out.println("a.b = " + format(formatStr, dot(a, b)));
    } catch (ArrayIndexOutOfBoundsException e) {
      out.println(e);
```

```
    }

    // normalise a zero vector?
    try {
      double[] c = { 0, 0, 0 };
      normalize(c);
      out.println("c hat = " + vecFormat(formatStr, c));
    } catch (ArithmeticException e) {
      out.println(e);
    }

  }
}
```

In fact the exception handling mechanism is rather clumsy for this example. VecMath throws only runtime exceptions, which are unchecked by the compiler; hence the programmer must remember to include try/catch blocks without helpful compiler reminders. Furthermore, runtime exceptions are generally used to escape unknowable problems. It is far better to handle internal errors with code (i.e. not exception throwing) which corrects program logic.

For example we might decide that an attempt to normalise a zero vector should not alter the vector in any way and attempts to dot product unequal length arrays result in a returned NaN.

Learning activity

Write a new class, SafeVecMath, with length(), dot() and normalise() methods which check for unequally sized arrays and zero-length vectors. Write a test class to show what happens if a client calls dot and normalise carelessly. Also demonstrate how to write safe client code for calls to these methods.

3.10 Safe vector maths

```
package exceptionhandling;

import static java.lang.Math.*;

public class SafeVecMath {

  private SafeVecMath() {
  }

  public static double length(double[] a) {

    return sqrt(dot(a, a));
  }

  public static void normalize(double[] a) {

    double mag = length(a);
    if (mag == 0)
      return;

    for (int i = 0; i < a.length; i++) {
      a[i] = a[i] / mag;
```

```
    }

  }

  public static double dot(double[] a, double[] b) {

    int numCpts = a.length;
    if (b.length != numCpts)
      return Double.NaN;

    double result = 0.0;
    for (int i = 0; i < numCpts; i++) {
      result += a[i] * b[i];
    }
    return result;
  }
}
```

3.11 Safe test

```
package exceptionhandling;

import static java.lang.System.out;
import static java.lang.String.format;

import static exceptionhandling.SafeVecMath.*;

public class SafeVecMathTest {

  public static String vecFormat(String formatStr, double[] a) {

    String s = "(";
    for (int i = 0; i < a.length - 1; i++) {
      s += format(formatStr, a[i]) + ", ";
    }
    s += format(formatStr, a[a.length - 1]) + ")";
    return s;
  }

  public static void main(String[] args) {

    // format String
    String formatStr = "%.4f";

    // dot product with different size vectors
    double[] a = new double[] { 1, 1, 0 };
    double[] b = new double[] { 1, 1 };
    out.println("a.b = " + format(formatStr, dot(a, b)));

    // normalise a zero vector?
    double[] c = { 0, 0, 0 };
    normalize(c);
    out.println("c hat = " + vecFormat(formatStr, c));

    // how to write safe client code
    if (a.length != b.length) {
      out.println("unequal sized vectors");
    } else {
```

```
      if (dot(a, b) == 0) {
        // carry on...
      }
    }
    if (length(c) == 0) {
      out.println("zero length vector");
    } else {
      normalize(c);
      // carry on...
    }

  }
}
```

SafeVecMathTest shows how the good programmer should prepare for runtime exceptions due to coding errors by what might be termed 'defensive programming'.

In Volume I we saw a situation where we just had to catch an exception; this was the checked InterruptedException, thrown by the JVM if the operating system could not service the request for program execution pause.

Checked exceptions are an important and valuable aspect of internet Java and we shall be using them often in the remainder of this volume.

3.12 Learning outcomes

By the end of this chapter, the relevant reading and activities, you should be able to:

- explain why it is important to write safe code
- understand when it might not be possible to write safe code, and the implications of this
- describe how a method can throw an exception object if something goes wrong at runtime
- understand how all exceptions subclass `java.lang.Exception`
- understand that the compiler does not check that possible runtime exceptions are handled in code
- understand that the compiler does care about **checked exceptions**, and insists that they are declared or wrapped in a `try/catch` block
- describe how a method throws an exception with the keyword `throw` followed by a new exceptions object
- understand that if your code calls a checked exception throwing method, you must either `duck` the exception or enclose the call in a `try/catch` block.

Chapter 4

Swing

Essential reading

HFJ Chapter 13.

You first encountered Java's user interface technology, Swing, in Chapter 11 of Volume 1. The emphasis in that chapter was in understanding the fundamentals of events and listeners. Here you will explore Swing in more detail, learning how to have more (or less) control of where user interface components are placed on your UI, and of the extensive range of available components.

The value of Swing is revealed in the programming tasks at the end of the chapter. Here you will discover how to build a rudimentary web browser.

4.1 Layout managers

Reading: pp. 400–402 of *HFJ*.

Text fields, buttons, scrollable lists, radio buttons, frames, panels... are all examples of what Swing calls *components* (more informally, widgets). Swing's components all extend `javax.swing.JComponent`.

Almost all swing components can be inserted inside other components. However, the usual technique is to insert *user interactive* components such as buttons into *background* components such as frames and panels. (In fact all components, except `JFrame` can be either interactive, or background.)

Remember, the steps to making a GUI are:

```
// Make a window (JFrame)
JFrame frame  = new JFrame();

// Make a component
JButton button = new JButton("Click me");

// Add the component to a frame
frame.getContentPane().add(BorderLayout.EAST, button);

// Display it
frame.setSize(500, 500);
frame.setVisible(true);
```

The code above uses a *layout manager* in order to control where the button will be placed, and what size it will have. Different managers have different policies. The default frame manager (i.e. when you do not specify one) is `BorderManager`, and the default panel manager is `FlowLayout`.

4.2 Border layouts

Reading: pp. 403–407 of *HFJ*.

The BorderLayout manager adds components to one of five background regions. Components in the north and south get their preferred heights, but not width. Components of the east and west get their preferred widths. The component in the middle gets whatever is left over.

4.3 Flow and Box layout

Reading: pp. 408–412 of *HFJ*.

The default layout manager for a panel is FlowLayout. Components are added, in order, left to right and top to bottom. Box layout, on the other hand, makes sure that components remain stacked even if the window is resized.

4.4 Other components

Reading: pp. 413–417 of *HFJ*.

Other commonly used components are `JTextField`, `JTextField`, `JTextArea`, `JCheckBox`, `JList` and `JScrollBar`. The reading shows some code examples.

4.5 Summary

- Layout managers control the size and location of nested components.
- The layout manager of the background component determines the size and location of the added component.
- The layout manager uses the preferred size of the added component for its calculations, but whether or not the size is respected depends on the layout manager's policies.
- Use BorderLayout manager to add components to one of five background regions.
- In this case components in the north and south get their preferred heights, but not width. The situation is reversed for the east/west regions. The component in the middle gets whatever is left over, unless you use `pack()`.
- pack() guarantees that the central component gets its preferred size. The size of the other regions depends on how much space is left over.
- Other managers include Flow and Box.

- BorderLayout is the default manager for a frame; FlowLayout is the default for a panel.
- Use setLayout() to change a panel's manager.

4.6 Programming

You are about to write your first internet Java program! The following activities ask you to generate a dynamic HTML page, and to make a simple web browser. But before we start these activities, there are three very useful library classes that you should get to know:

4.6.1 JEditorPane

```
class javax.swing.JEditorPane
public JEditorPane(String url)
public JEditorPane(String type, String text)
public JEditorPane(URL initialPage)
public final setContentType(String type)
public void setPage(String url) throws IOException
public void setPage(URL url) throws IOException
public void setText(String t)

Inherited from javax.swing.text.JTextComponent
public void setEditable(boolean b)
```

JEditorPane, along with JTextArea and JTextField, subclass the abstract `javax.swing.text.JTextComponent`. You met text areas and field in the reading for this chapter. All text components are intended for text manipulation and display. However JEditorPane can handle plain text, rich text format, **and HTML**. The particular type of text is fixed by calling `setContentType(String textType)` where textType is text/plain, text/html or text/rtf (refer to the Java API for more information).

4.6.2 URL

```
class java.net.URL
public URL(String spec) throws MalformedURLException
public final InputStream openStream() throws IOException
public String getHost()
```

The class `java.net.URL` represents a Uniform Resource Locator i.e. a pointer to a resource on the World Wide Web. We shall mainly use it to open a *stream* so that we can read data. (Java streams are covered in the next chapter.)

4.6.3 StringBuffer/StringBuilder

```
class java.lang.StringBuffer/Builder
```

```
public StringBxxxx()
public StringBxxxx(String str)
public StringBxxxx append(char c)
public StringBxxxx append(String str)
public StringBxxxx delete(int start, int end)
public int indexOff(String str)
public insert(int offset, Srring str)
public int length()
public String toString()
```

Reading: pg. 669 of *HFJ*.

Bxxxx refers to either StringBuffer of StringBuilder.

StringBuffer's and StringBuilder's (new to Java 5) are like strings, except that characters can be added and deleted. Java strings are immutable which means that string manipulation requires the generation of new objects. The compiler uses StringBuffers for string concatenation. For example, the code:

```
String str = "H" + 2 + "O";
```

is compiled to the equivalent of:

```
String str = new StringBuffer().append("H").append(2).append("O").toString();
```

We mention here that StringBuilder methods are not *synchronized* and should be used in preference to StringBuffers in single thread situations, or where thread safety is not an issue (you will find out about threads and synchronization in a later chapter).

StringBxxx's should be used wherever there is a lot of string concatenation, as in the following activity.

Learning activity

This activity demonstrates how to generate an HTML web page from within Java. Your dynamically generated HTML page will display the first 1000 Fibonacci numbers as HTML. These numbers will be calculated and then displayed without first saving to an HTML file.

The first Fibonacci number is 0, and the second is 1; subsequent numbers are the sums of the two immediate predecessors. The series starts: $0, 1, 1, 2, 3, 5, 8, \ldots$.

The Fibonacci numbers quickly get very large, larger even than the longest Java `long`. Investigate how java.math.BigInteger class can hold integers of arbitrary size. Generate the numbers one by one, adding them to a StringBuilder for efficiency. Convert the StringBuilder to a String with HTML formatting and display using JEditorPane.

4.7 Dynamic HTML

```
package usingswing;

import java.math.BigInteger;
import javax.swing.JEditorPane;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.WindowConstants;

public class Fibonacci {

  public static void main(String[] args) {

    StringBuilder result = new StringBuilder(
        "<html><body><h1>Fibonacci Numbers</h1><ol>");

    BigInteger fib1 = BigInteger.valueOf(0);
    BigInteger fib2 = BigInteger.valueOf(1);

    for (int i = 0; i < 1000; i++) {
      result.append("<li>");
      result.append(fib1);
      BigInteger fib3 = new BigInteger(fib2.toByteArray());
      fib2 = fib2.add(fib1);
      fib1 = fib3;
    }
    result.append("</ol></body></html>");

    JEditorPane jep = new JEditorPane("text/html", result.
        toString());
    jep.setEditable(false);

    JScrollPane scrollPane = new JScrollPane(jep);
    JFrame f = new JFrame("Fibonacci Numbers");
    f.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
    f.getContentPane().add(scrollPane);
    f.setSize(512, 342);
    f.setVisible(true);
  }
}
```

Learning activity

This activity has more than a flavour of internet Java; you are about to build your own internet browser! You will be surprised at how easy this task is with Swing. The secret is that JEditorPane already contains the necessary networking code to download and display a website. The API tells us that `setPage(String url)` will display the page referred to by the given url. The only snag is the full HTML specification is not recognised by JEditorPane. The capabilities are rather limited; JEditorPane may not display the page as you would normally see it.

Use JEditorPane to display a URL provided at the command line. You will need to construct a JScrollPane with JEditorPane as a parameter. Then add the JScrollPane object to the JFrame's content pane, and display.

4.8 Page loader

```
package usingswing;

import java.io.IOException;
import javax.swing.JEditorPane;
import javax.swing.JFrame;
import javax.swing.JScrollPane;

public class PageLoader {

  public static void main(String[] args) {

    JEditorPane jep = new JEditorPane();
    jep.setEditable(false);

    try {
      jep.setPage(args[0]);
    } catch (IOException e) {
      jep.setContentType("text/html");
      jep.setText("<html>" + e + "</html>");
    }

    JScrollPane scrollPane = new JScrollPane(jep);

    JFrame f = new JFrame();
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.getContentPane().add(scrollPane);
    f.setSize(512, 342);
    f.setVisible(true);
  }
}
```

Page loader is rather limited; it is fixed at a single web page. The user cannot interact with the application; in particular (s)he cannot click on a link and navigate to a new page or site.

Learning activity

Hyperlinks are activated on a JEditorPane if a HyperlinkListener is added to the pane. Consult the API for JEditorPane.addHyperlinkListener. Write an inner class which implements HyperlinkListener and include it with your code from PageLoader; save as SimpleBrowser.

4.9 Simple browser

```
package usingswing;

import java.io.IOException;
import javax.swing.JEditorPane;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.event.HyperlinkEvent;
import javax.swing.event.HyperlinkListener;
```

```
public class SimpleBrowser {

  JEditorPane pane;

  class Hyperactive implements HyperlinkListener {

    public void hyperlinkUpdate(HyperlinkEvent e) {

      if (e.getEventType() == HyperlinkEvent.EventType.ACTIVATED)
          {
        try {
          pane.setPage(e.getURL());
        } catch (Throwable t) {
          t.printStackTrace();
        }
      }
    }
  }

  public SimpleBrowser(String home) {

    pane = new JEditorPane();
    pane.addHyperlinkListener(new Hyperactive());
    pane.setEditable(false);

    try {
      pane.setPage(home);
    } catch (IOException e) {
      pane.setContentType("text/html");
      pane.setText("<html>" + e + "</html>");
    }

    JScrollPane scrollPane = new JScrollPane(pane);
    JFrame f = new JFrame();
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.getContentPane().add( scrollPane);
    f.setSize(512, 342);
    f.setVisible(true);
  }
  public static void main(String[] args){
    new SimpleBrowser(args[0]);
  }
}
```

Learning activity

SimpleBrowser can be improved still further by adding forward and back buttons, enabling the user to navigate to previously visited (in that session) sites, and by adding an address filed so that the user can type a new URL.

Use your knowledge of Swing to implement these and any further improvements.

4.10 Better browser

```
package usingswing;

import java.awt.*;
import java.awt.event.*;
import java.io.IOException;
import java.net.*;
import java.util.ArrayList;
import javax.swing.*;
import javax.swing.event.*;

public class BetterBrowser {

  private JEditorPane pane;
  private JTextField address;
  private int currentIndex;
  private String home = "http://doc.gold.ac.uk/~mas01tb/index.
      html";
  private ArrayList<URL> history = new ArrayList<URL>();

  class Hyperactive implements HyperlinkListener {

    public void hyperlinkUpdate(HyperlinkEvent e) {

      if (e.getEventType() == HyperlinkEvent.EventType.ACTIVATED)
        newPage(e.getURL());
    }
  }

  class AddressListener implements ActionListener {

    public void actionPerformed(ActionEvent e) {
      try {
        URL url = new URL(address.getText());
        newPage(url);
      } catch (IOException ex) {
        printHTML(ex.toString());
      }
    }
  }

  class BackButtonListener implements ActionListener {

    public void actionPerformed(ActionEvent e) {

      setPage(currentIndex - 1);
    }
  }

  class ForwardButtonListener implements ActionListener {

    public void actionPerformed(ActionEvent e) {
      setPage(currentIndex + 1);
    }
  }
```

```
private void newPage(URL url) {
  try {
    pane.setPage(url);

    // clear history for this point forwards
    int i = history.size() - 1;
    while (i > currentIndex)
      history.remove(i--);

    history.add(url);
    currentIndex++;
    address.setText(url.toString());
  } catch (IOException e) {
    printHTML(e.toString());
  }
}

private void setPage(int desiredIndex) {

  if (desiredIndex >= 0 && desiredIndex < history.size()) {
    try {
      URL url = history.get(desiredIndex);
      pane.setPage(url);
      currentIndex = desiredIndex;
      address.setText(url.toString());
    } catch (IOException e) {
      printHTML(e.toString());
    }
  }
}

private void printHTML(String message) {

  pane.setContentType("text/html");
  pane.setText("<html>" + message + "</html>");
}

public BetterBrowser() {

  home = "http://www.goldsmiths.ac.uk/computing/";
  int width = 900;
  int height = 500;
  pane = new JEditorPane();
  pane.addHyperlinkListener(new Hyperactive());
  pane.setEditable(false);
  pane.setPreferredSize(new Dimension(width, height));

  JScrollPane scrollPane = new JScrollPane(pane);
  scrollPane
      .setVerticalScrollBarPolicy(ScrollPaneConstants.
          VERTICAL_SCROLLBAR_ALWAYS);
  scrollPane
      .setHorizontalScrollBarPolicy(ScrollPaneConstants.
          HORIZONTAL_SCROLLBAR_ALWAYS);

  JButton backButton = new JButton("<<");
  backButton.addActionListener(new BackButtonListener());

  JButton forwardButton = new JButton(">>");
```

```
    forwardButton.addActionListener(new ForwardButtonListener());

    address = new JTextField(50);
    address.setText(home);
    address.addActionListener(new AddressListener());

    currentIndex = -1;
    try {
      newPage(new URL(home));
    } catch (MalformedURLException e) {
      printHTML(e.toString());
    }

    JPanel panel = new JPanel();
    panel.add(backButton);
    panel.add(forwardButton);
    panel.add(address);

    JFrame frame = new JFrame(home);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.getContentPane().add(BorderLayout.NORTH, panel);
    frame.getContentPane().add(BorderLayout.CENTER, scrollPane);
    frame.setSize(width, height + 50);
    frame.setVisible(true);
  }

  public static void main(String[] args) {
    new BetterBrowser();
  }
}
```

The code listing seems long, but much space is taken up by the three inner classes. The functionality has been improved with an address bar and forward and back buttons. Perhaps your browser has even more features.

4.11 Learning outcomes

By the end of this chapter, the relevant reading and activities, you should be able to:

- describe how layout managers control the size and location of nested components
- understand that the layout manager of the background component determines the size and location of the added component
- describe how the layout manager uses the preferred size of the added component for its calculations
- understand that whether or not the size is respected depends on the layout manager's policies
- use BorderLayout manager to add components to one of five background regions
- understand that in this case components in the north and south get their preferred heights, but not width; the situation is reversed for the east/west regions; and the component in the middle gets whatever is left over, unless you use `pack()`
- describe how `pack()` guarantees that the central component gets its preferred size, and that the size of the other regions depends on how much space is left over
- describe other managers such as Flow and Box
- understand that `BorderLayout` is the default manager for a frame and `FlowLayout` is the default for a panel
- use `setLayout()` to change a panel's manager.

Chapter 5

Streams

Essential reading

HFJ Chapter 14.

5.1 Introduction

The Java IO (input-output) API represents connections to destinations outside the JVM (files on the hard disc, serial ports, network sockets, the sound card . . .) by streams. The idea is that the destination doesn't matter – you use the same classes. In other words you, as the application programmer, work with a common interface. The acronym API is normally used to mean *Application Programming Interface*.

5.2 Data streams

Reading: pg. 433 of *HFJ*.

There are two types of stream: connection streams for the connection itself, and chain streams. The latter can only connect to other streams.

Connection streams are low level – they can only read/write bytes – but we would prefer to read/write strings, characters, numbers, and even objects. That's where the chain streams come in.

Why is this the way? According to the principles of Object programming, each class should have one job or function. By separating out low and high level work, the API is very flexible and maintainable.

Traditionally (i.e. Unix and C) desktops have a standard input (keyboard and terminal), a standard output (the terminal), and an output for error messages (usually also the terminal).

The System class continues this tradition.

5.3 Reading and writing to a text file

Reading: pg. 447 of *HFJ* ; pp. 452–454 of *HFJ* ; pp. 458–459 of *HFJ*.

Each call to a file reader or writer will result in a visit to the disk, a very expensive

operation compared to manipulating data in memory. Chaining BufferedReaders and Writers greatly improves the efficiency of reading and writing since the operation will only take place when the buffer is full. But if you wish to send data before the buffer is full, call `flush()`.

`String.split(String separator)` can be used to *parse* text. The text is split, according to the positions of of the separators into a String array.

5.4 Reading bytes

InputStream

```
public abstract class java.io.InputStream
public abstract int read() throws IOException
public int read(byte[] b) throws IOException
public int read(byte[] b, int off, int len)
```

The other implemented read methods place the data in an array of bytes, returning the number of bytes read, or -1 if no bytes were available (i.e. the stream is at the end of the file).

java.lang.System

```
public static PrintStream err
public static PrintStream in
public static PrintStream out
public static void exit(int status)
public static long currentTimeMillis()
public static void setErr(PrintStream err)
public static void setIn(PrintStream err)
public static void setOut(PrintStream err)
```

`System.in` is an InputStream reference. `System.in` actually points to a concrete subclass object, one that implements `int read()`.

```
try{
   int val = System.in.read();
   byte b = (byte)val;
}
catch(IOEXception e){
   System.out.println(e);
}
```

Notice that each read returns a byte and (once more according to the C convention) returns -1 if no bytes are available. Integer values from 0 to 255 are returned for normal reads, leaving -1 available as an end of stream flag.

`System.in.read()`, in common with other reads in the IO API, *blocks*. This means that execution hangs at this line of code until a byte is available to be read. (Another package, Java NIO, allows non-blocking reading, and this can be better for network applications.)

Reading bytes is rather inefficient, and besides which we may want to read Unicode characters. `InputStreamReaders` provide a bridge between bytes and characters.

5.5 Summary

- Chain a FileWriter connection stream to a BufferedWriter for efficient writing.
- A File object represents the path to a file, not the contents of the file.
- Use a File object in preference to a String filename when reading/writing to a stream.
- Chain a FileReader connection stream to a BufferedReader for efficient reading.
- If you need to parse a text file, then you will need to recognise different elements of the file, for example by using a separator.
- String.split() is a convenient way of splitting a string into its individual tokens.
- Use a serial version ID if a class definition may change and render serialised objects of that class incompatible.

5.6 Programming

Here is another useful java.net class:

InetAddress

```
class java.net.InetAddress
public static InetAddress getByName(String host) throws UnknownHostException
public String getHostName()
public static InetAddress getLocalHost() throws UnknownHostException
public String getHostAddress()
public static InetAddress getLocalHost() throws UnknownHostException
```

Java represents an internet address by the class `InetAddress`. This class has no public constructors but you can obtain an InetAddress object by a call to one of the static methods. These methods actually make connections to the local DNS server to acquire the information that an InetAddress object needs. This is a good example of how the Java API does a lot of tedious networking coding for us. Once we have an InetAddress object, we can call various instance methods to get the address or the hostname.

Learning activity

Write a program that reads from System.in and prints to System.out.

5.7 Terminal Input

```
package io;

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

public class TerminalInput {

  public static void main(String[] args) {
    try {
      BufferedReader inputStream = new BufferedReader(
          new InputStreamReader(System.in));
      String inputLine = null;
      while ((inputLine = inputStream.readLine()) != null) {
        System.out.println(inputLine);
      }
      inputStream.close();
    } catch (java.io.IOException e) {
      System.err.println(e);
    }
  }
}
```

TerminalInput chains the InputStreamReader to a buffer. This is for efficiency. Rather than making 1024 trips of a single byte, make one trip of 1024 bytes. This technique is common. Notice how the read operation happens inside the condition of the while loop.

The input stream is closed after use. This is very important, since it frees up that destination for other parts of your program (and indeed for other programs running on your machine). All streams will in fact be closed when the JVM exits. A status value is then returned to your operating system. Usually the value 0 means successful execution of the program, and 1 is some error.

System.out is a PrintStream. Unlike other streams, this never throws an IOException.

"`PrintStreams` are evil and should be avoided like the plague" says the author of *JNP*. They are OK for terminal printing, but bad for writing to network clients because:

i println is platform dependent;

ii it assumes the default encoding of the platform it is running on; and

iii it eats all exceptions.

If you're worried about all this, use a `BufferedWriter`. `BufferedWriters` have a `newLine()` method that inserts a new line character, according to the local platform.

Learning activity

Write a program which reads a specified number of characters typed at the command line and then outputs the byte value of each character.

5.8 Read Bytes

```
package io;

import java.io.*;
import java.util.Arrays;

public class ReadBytes {

  InputStream in;
  int bytesToRead;
  byte[] byteArray;

  public ReadBytes(InputStream in, int numBytes) {

    this.in = in;
    bytesToRead = numBytes;

    byteArray = new byte[bytesToRead];

    try {
      go();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  public byte[] getByteArray(){

    return byteArray;
  }

  public void go() throws IOException {

    int bytesRead = 0;
    int bytesAvailable = 0;
    while (bytesRead < bytesToRead && bytesAvailable != -1) {

      bytesAvailable = in
          .read(byteArray, bytesRead, bytesToRead - bytesRead);
      bytesRead += bytesAvailable;
    }
  }

  public static void main(String[] args) {

    InputStream in = System.in;
    int numBytesToRead = 32;

    ReadBytes rb = new ReadBytes(in, numBytesToRead);

    byte[] byteArray = rb.getByteArray();
    System.out.println(byteArray.length + " bytes read");
    System.out.println(Arrays.toString(byteArray));
  }
}
```

Read Bytes invokes input stream's read(byte[], int, int) to read directly into a byte array. Notice how the while loop does not assume that all 32 bytes will be available in a single read. The difference between the total number of bytes to read and the number of bytes actually read gives the required number to read at each pass through the loop.

Learning activity

Write a short program to print the contents of a web page at the terminal. You will need to instantiate a URL object at a website and then obtain a stream for reading. The output should preserve the original formatting of the HTML file.

The program should run with the command:

java SourceViewer http://www.gold.ac.uk

5.9 Source Viewer

```
package io;

import java.io.*;
import java.net.*;

public class SourceViewer {

  public static void main(String args[]) {

    try {

      URL u = new URL(args[0]);
      Reader r = new BufferedReader(new InputStreamReader(u.
          openStream()));

      int c = -1;
      while ((c = r.read()) != -1) {
        System.out.print((char) c);
      }

    } catch (MalformedURLException e) {
      System.err.println(e);
    } catch (IOException e) {
      System.err.println(e);
    }
  }
}
```

Source Viewer reads bytes and writes characters. This ensures that characters such as newlines, tabs, etc., are used for formatting instructions.

Learning activity

Write a Java program which reads its own source code. The Java source file should be displayed as text in a window. Add a button so that the code can be edited and saved.

5.10 Mirror

```
package io;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;

public class Mirror {

  JTextArea text;

  public Mirror() {

    show(read("./src/io/Mirror.java"));
  }

  class ButtonListener implements ActionListener {

    public void actionPerformed(ActionEvent arg0) {
      write(text.getText(), "MeModified.java");
    }
  }

  private void show(String str) {

    text = new JTextArea();
    text.setText(str);

    JScrollPane scroller = new JScrollPane(text);
    scroller
        .setVerticalScrollBarPolicy(ScrollPaneConstants.
            VERTICAL_SCROLLBAR_ALWAYS);
    scroller
        .setHorizontalScrollBarPolicy(ScrollPaneConstants.
            HORIZONTAL_SCROLLBAR_ALWAYS);
    scroller.setPreferredSize(new Dimension(800, 800));

    JPanel panel = new JPanel();
    panel.add(scroller);

    JButton button = new JButton("Save");
    button.addActionListener(new ButtonListener());

    JFrame frame = new JFrame();
    frame.getContentPane().add(BorderLayout.CENTER, panel);
    frame.getContentPane().add(BorderLayout.SOUTH, button);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(900, 900);
    frame.setVisible(true);
  }

  private String read(String file) {

    String fileAsString = null;
    try {
```

```
      BufferedReader reader = new BufferedReader(new FileReader(
          file));

      StringBuffer stringBuff = new StringBuffer();
      String line = null;
      while ((line = reader.readLine()) != null)
        stringBuff.append(line).append("\n");

      reader.close();
      fileAsString = stringBuff.toString();
    } catch (IOException e) {
      System.out.println("Could not read file");
    }
    return fileAsString;
  }

  private void write(String str, String file) {

    try {
      BufferedWriter out = new BufferedWriter(new FileWriter(file
          ));
      out.write(str, 0, str.length());
      out.close();
    } catch (IOException e) {
      System.out.println("Could not write to file");
    }
  }

  public static void main(String[] args) {

    new Mirror();
  }
}
```

With just a little work this program can serve as a simple text editor. By exploring the methods of `java.lang.Runtime` you may even be able to code a Java development environment by including compile and run functionalities.

Learning activity

Write a Java application, Viewer.java, that displays the contents of a file (e.g. file.txt) with the command **java io/Viewer io/file.txt**

5.11 File viewer

```
package io;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;

public class Viewer {

  JTextArea text;
```

```
class ButtonListener implements ActionListener {

  public void actionPerformed(ActionEvent arg0) {
    write(text.getText(), "MeModified.java");
  }
}

public void go(String file){
  show(read(file));
}

private void show(String str) {
  text = new JTextArea();
  text.setText(str);
  text.setEditable(false);

  JScrollPane scroller = new JScrollPane(text);
  scroller
      .setVerticalScrollBarPolicy(ScrollPaneConstants.
          VERTICAL_SCROLLBAR_ALWAYS);
  scroller
      .setHorizontalScrollBarPolicy(ScrollPaneConstants.
          HORIZONTAL_SCROLLBAR_ALWAYS);
  scroller.setPreferredSize(new Dimension(800, 800));

  JPanel panel = new JPanel();
  panel.add(scroller);

  JButton button = new JButton("Save");
  button.addActionListener(new ButtonListener());

  JFrame frame = new JFrame();
  frame.getContentPane().add(BorderLayout.CENTER, panel);
  frame.getContentPane().add(BorderLayout.SOUTH, button);
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  frame.setSize(900, 900);
  frame.setVisible(true);
}
private String read(String file) {

  String fileAsString = null;
  try {
    BufferedReader reader = new BufferedReader(new FileReader(
        file));

    StringBuffer stringBuff = new StringBuffer();
    String line = null;
    while ((line = reader.readLine()) != null)
      stringBuff.append(line).append("\n");

    reader.close();
    fileAsString = stringBuff.toString();
  } catch (IOException e) {
    System.out.println("Could not read file");
  }
  return fileAsString;
}
```

```
  private void write(String str, String file) {

    try {
      BufferedWriter out = new BufferedWriter(new FileWriter(file
          ));
      out.write(str, 0, str.length());
      out.close();
    } catch (IOException e) {
      System.out.println("Could not write to file");
    }
  }

  public static void main(String[] args) {

    new Viewer().go(args[0]);
  }
}
```

Learning activity

Write a program which, when given a host name, finds the corresponding IP address, as exemplified by this terminal session:

java io/HostInfo www.gold.ac.uk
www.gold.ac.uk 158.223.1.86

5.12 Host info

```
package io;

import java.net.*;

public class HostInfo {

  public static void main(String[] args) {
        try {
            InetAddress inetAddress = InetAddress.getByName(args
                [0]);
            System.out.println(inetAddress.getHostName() + "\t"
                    + inetAddress.getHostAddress());
        } catch (UnknownHostException e) {
            System.out.println(e);
        }
    }
}
```

Learning activity

Write a program which reports on the IP address of the current machine.

5.13 Where am I?

```
package io;

import java.net.*;

public class MyAddress {

  public static void main(String[] args) {

        try {
            InetAddress inetAddress = InetAddress.getLocalHost();
            System.out.println(inetAddress.getHostAddress());
        } catch (UnknownHostException e) {
            System.out.println(e);
        }
    }

}
```

Learning activity

Write a program which navigates to a given website and downloads the HTML contents of the site to a file. The program then displays the newly created file in a window to confirm that the task has been completed successfully.

5.14 SourceSaver

```
package io;

import java.io.BufferedReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.Reader;
import java.net.MalformedURLException;
import java.net.URL;

public class SourceSaver {

  private static void writeToFile(String fileName, String text) {

    try {
      FileWriter writer = new FileWriter(fileName);
      writer.write(text);
      writer.close();
    } catch (IOException e) {
      System.out.print(e);
    }
  }

  public static void main(String args[]) {
```

```
    StringBuilder builder = new StringBuilder();
    try {
      URL u = new URL("http://" + args[0]);
      Reader r = new BufferedReader(new InputStreamReader(u.
          openStream())); // IO

      int c = 0;
      while ((c = r.read()) != -1) { // throws IO exception
        builder.append((char) c);
      }

      String site = args[0];
      if (site.contains("http://")) {

      }
      String file = args[0] + ".html";
      writeToFile(file, builder.toString());
      new Viewer().go("./" + file);
    }

    catch (MalformedURLException e) { // a subclass of
        IOException
      System.err.println(e);
    } catch (IOException e) {
      System.err.println(e);
    }

  }

}
```

Learning activity

The final activity of this chapter will be the construction of a webspider. Webspiders roam endlessly over the internet gathering URLs. Since there is no directory of the internet, the only way to find out what is out there is by exploration.

In fact a webspider doesn't move at all. The webspider is an application which downloads HTML from a site, finds HTML tags within the source, looks for possible URLs within the tags, and then downloads source from each of these URLs, and so on.

We need two small programs before we build the spider: a tag extractor which hunts for HTML tags in a document and a URL extractor which searches a tag for a URL.

Write a utility class with two static methods:

```
public static ArrayList<String>extractTags(URL u)
```

and

```
public static String extractURL(String tag).
```

5.15 Tag and URL extractor

```
package io;

import java.io.*;
import java.net.*;
import java.util.ArrayList;

public class Util {

    public static String extractURL(String tag) {

        String host = null;
        tag.trim();
        if (tag.startsWith("<a") || tag.startsWith("<A")) {

            for (int i = 0; i < tag.length();) {

                String sub = tag.substring(i);
                if (sub.startsWith("http://") || sub.startsWith("
                    HTTP://")) {

                    int j = 7;
                    for (; j < sub.length() && sub.charAt(j) != '
                        /'
                            && sub.charAt(j) != '"'; j++)
                        ;

                    host = sub.substring(0, j);
                    i += 7;
                } else
                    i++;
            }
        }
        return host;
    }

    public static ArrayList<String> extractTags(URL u) throws
        IOException {

        ArrayList<String> tags = new ArrayList<String>();

        Reader r = new BufferedReader(new InputStreamReader(u.
            openStream()));
        StringBuffer tag = new StringBuffer();

        int c = 0;
        boolean inTag = false;
        while ((c = r.read()) != -1) {

            if (c == '<')
                inTag = true;

            if (inTag)
                tag.append((char) c);

            if (c == '>') {
```

```
                tags.add(tag.toString());
                tag = new StringBuffer();
                inTag = false;
            }
        }
        return tags;
    }

    public static void main(String args[]) {

        try {
            URL u = new URL("http://www.gold.ac.uk");

            ArrayList<String> tags = extractTags(u);
            for (int i = 0, n = tags.size(); i < n; ++i) {
                String host = extractURL(tags.get(i));
                if (host != null)
                    System.out.println(host);
            }
        } catch (MalformedURLException e) {
        } catch (IOException e) {
        }
    }
}
```

Learning activity

Code a webspider using the above extractors. The aim of the spider is to catalogue as many websites as possible. Think carefully of how you will store the incoming URLs, and how you will ensure that a previously visited website is not revisited (or else the spider is in danger of getting caught in loops).

5.16 Webspider

```
package io;

import java.io.*;
import java.net.*;
import java.util.ArrayList;

public class SimpleSpider {

    ArrayList<String> set = new ArrayList<String>();

    public void search(String host) {

        try {
            URL u = new URL(host);

            set.add(host);

            ArrayList<String> tags = Util.extractTags(u);
            for (int i = 0; i < tags.size(); ++i) {

                String nexthost = Util.extractURL(tags.get(i));
```

```
                if (nexthost != null && !set.contains(nexthost))
                    {
                     System.out.println(nexthost);
                     search(nexthost);
                }
            }
        } catch (MalformedURLException e) {
        } catch (IOException e) {
        }
    }

    public static void main(String args[]) {

        SimpleSpider spider = new SimpleSpider();
        spider.search("http://www.gold.ac.uk");
    }
}
```

5.17 Learning outcomes

By the end of this chapter, the relevant reading and activities, you should be able to:

- describe the two types of stream: connection and chain; their functions and how they work
- chain a FileWriter connection stream to a BufferedWriter for efficient writing
- understand that a File object represents the path to a file, not the contents of the file
- use a File object in preference to a String filename when reading/writing to a stream
- chain a FileReader connection stream to a BufferedReader for efficient reading
- understand that if you need to parse a text file, then you will need to recognise different elements of the file
- understand that `String.split()` is a convenient way of splitting a string into its individual tokens
- use a serial version ID if a class definition may change and render serialised objects of that class incompatible
- write a program that reads from `System.in` and prints to `System.out`
- write a program which reads a specified number of characters typed at the command line and then outputs the byte value of each character
- write a short program that prints the contents of a web page at the terminal
- write a Java program that reads its own source code
- write a Java application that displays the contents of a file
- write a program which, when given a host name, finds the corresponding IP address
- write a program which reports on the IP address of the current machine
- write a program which navigates to a given website and downloads the HTML contents of the site to a file
- construct a webspider.

Chapter 6

Serialisation

Essential reading

HFJ Chapter 14.

Supplementary reading

JNP Chapter 4.

6.1 Introduction

This chapter explains how to save the state of a Java object (serialisation). We begin by learning how easy it is to save an object to a file, and reconstitute it at a later date.

6.2 Saving state

Reading: pp. 431–432 of *HFJ*.

There are two options for saving state (i.e. the values of the instance variables): (i) serialise or (ii) write to a plain text file using some protocol that allows you to initialise state from text.

Either solution might make sense, but generally serialisation is the easier way.

The steps are:

```
// Make a FileOutputStream
FileOutputStream fileStream = new FileOutputStream("MyObject.ser"");

// Make an object output stream
ObjectOutputStream os = new ObjectOutputStream(fileStream);

// Write the object
os.write(myObject);

// Close the stream
os.close();
```

Reading: pp. 434–440 of *HFJ*.

Serialised objects save the values of their primitive instance variables. If an instance variable is a reference variable, then the object it refers to is serialised as well; this rule applies to all objects in the entire object graph.

Objects, or their super class, must implement `Serializable` if they are to be serialised. This is a simple marker interface, announcing to the compiler that you are happy for objects of this class to be serialised. The whole object graph must contain serialisable objects, or else the process will fail.

Sometimes you may wish an instance variable to be skipped by the serialisation process. A transient instance variable will not be serialised. It will assume the default/null value when the containing object is restored.

6.3 Restoring state

Reading: pp. 441–443 of *HFJ*.

The steps are:

```
// Make a FileInputStream
FileInputStream is = new FileInputStream("gooables.ser");

// Make an ObjectInputStream
ObjectInputStream os = new ObjectInputStream(is);

// Read the object
Object obj = (Gooable) os.readObject();

// Cast the object
Gooable gooable = (Gooable) obj;

// Close the streams
os.close();
```

6.4 Version ID

One issue with object deserialisation is a possible change of class definition which may break the deserialisation process. A serial version ID can solve this problem. This enables the JVM to assess if the class is compatible with the serialised object.

6.5 Summary

- Serialisation saves an object's state.
- Streams are either connection or chain streams.
- Connection streams represent a connection to a source or destination and must be chained to another stream.
- Chain streams cannot connect to a source or destination.

- An object is serialised to a file by chaining a FileOutputStream to an ObjectOutputStream.
- Then call `writeObject(theObject)` on the ObjectOutputStream.
- Objects, or their super class, must implement Serializable if they are to be serialised.
- All objects in an object's entire object graph are serialised at the same time. An exception will be thrown at runtime if any of these other objects does not implement Serializable.
- A transient instance variable will not be serialised. It will assume the default/null value when the containing object is restored.
- The class of all objects in the graph must be available to the JVM during deserialisation.
- Objects are read using `readObject()` in the order in which they were serialised.
- readObject returns an Object reference which should be subsequently cast to the objects type (or supertype).
- Static variables are not serialised.
- Use a serial version ID if a class definition may change and render serialised objects of that class incompatible.

6.6 Programming

The programming section of this chapter is devoted to a single project: an animation that can be saved and then restarted. How can we save and reload an animation? By serialising the state of the animation program at a given time and writing to disk. And then subsequently resuming the animation by deserialisation.

We start with an extended Goo, GooWorld.

6.7 GooWorld

```
package serialisation;

import goo.Goo;

import java.awt.*;
import java.util.ArrayList;

public class GooWorld extends Goo {

  ArrayList<Gooable> gooables;

  public GooWorld(int w, int h, boolean b) {

    super(w, h, b);
    gooables = new ArrayList<Gooable>();
  }

  public void addGooable(Gooable g) {

    gooables.add(g);
```

```
  }


  public void draw(Graphics g) {

    for (Gooable obj : gooables) {

      obj.move(getWidth(), getHeight());
      obj.draw(g);
    }
  }

  public ArrayList<Gooable> getGooables() {

    return gooables;
  }

  public void addGooables(ArrayList<Gooable> g){

    gooables = g;
  }
}
```

```
package serialisation;

import java.awt.Graphics;

public interface Gooable {

  public void draw(Graphics g);

  public void move(int width, int height);


}
```

GooWorld is a simple extension of Goo. The new feature is the overridden draw which loops through an `ArrayList` of `Gooables`. We see from the interface that a gooable is an object that responds to move and draw messages. We can conceive of two sorts of movement; an internal movement that causes a drawing to change its shape and form (as in MovingPolygon) and a movement that causes a shape to redraw itself at a different screen location (as in MovingHoop).

Run the demo program GooWorldApp to see an animation with both types of movement.

Learning activity

Write an animation that consists of a number of gooables. The gooables should move across the screen, changing shape in some way as they do so.

Begin by writing a gooable object, for example a 'flapping polygon' as in the demonstration.

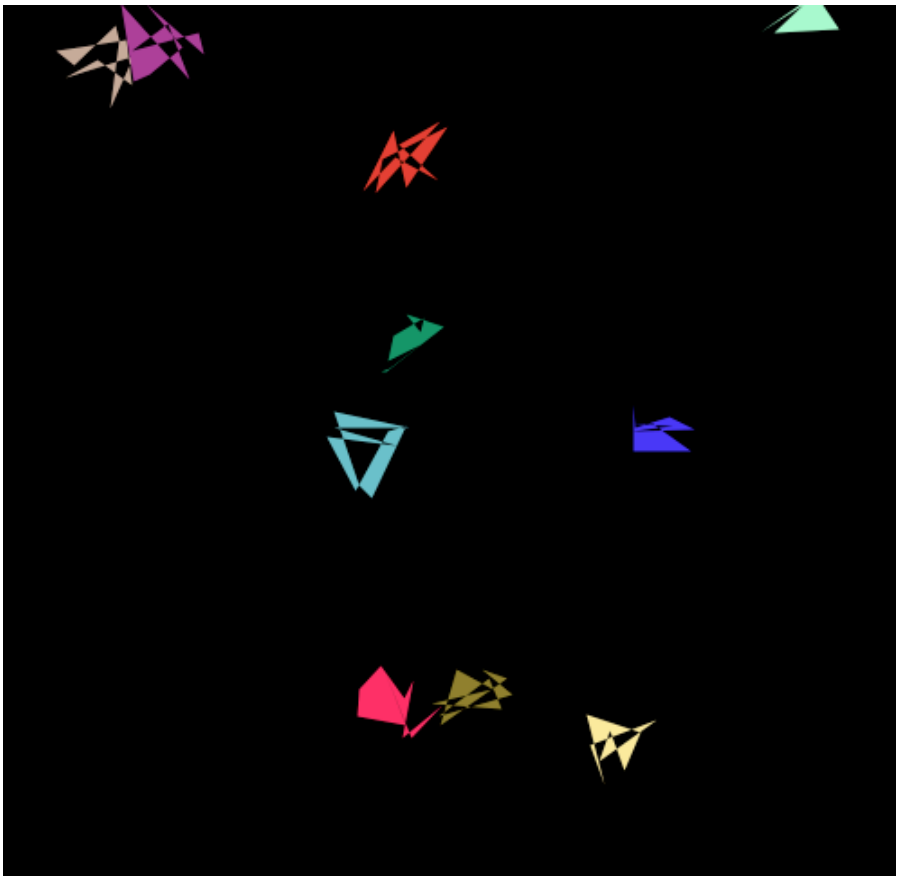

Figure 6.1: Goo world populated with flapping polygons

6.8 Flapping polygon

```
package serialisation;

import java.awt.Color;
import java.awt.Graphics;
import java.io.Serializable;
import java.util.Random;

public class FlappingPolygon implements Gooable, Serializable {

  private static final long serialVersionUID = 1L;

  // External position, velocity and size
  int ox, oy, vox, voy, maxVox, sizeX, sizeY;

  // Internal position and velocities
  int[] x, y, vx, vy;
  int maxV;

  // Color of interior fills and lines
  Color color;

  // Num interior points
  int nVertices, maxVertices;

  // Random object
  Random random;

  public FlappingPolygon(int width, int height, long seed) {

    random = new Random(seed);

    // External param init
    sizeX = 50;
    sizeY = 50;

    ox = random.nextInt(width);
    oy = random.nextInt(height);

    maxVox = 10;

    vox = 1 + random.nextInt(maxVox);
    if (random.nextDouble() > 0.5)
      vox *= -1;

    voy = 1 + random.nextInt(maxVox);
    if (random.nextDouble() > 0.5)
      voy *= -1;

    // Internal param init
    randomisePolygon();
  }

  void randomisePolygon() {

    // Internal param init
```

```
    maxV = Math.min(sizeX, sizeY) / 3;
    maxVertices = 20;
    nVertices = maxVertices / 2;

    x = new int[nVertices];
    y = new int[nVertices];
    vx = new int[nVertices];
    vy = new int[nVertices];

    for (int i = 0; i < nVertices; i++) {

      x[i] = ox + random.nextInt(sizeX);
      y[i] = oy + random.nextInt(sizeY);

      vx[i] = 1 + random.nextInt(maxV - 1);
      if (random.nextDouble() > 0.5)
        vx[i] *= -1;

      vy[i] = 1 + random.nextInt(maxV - 1);
      if (random.nextDouble() > 0.5)
        vy[i] *= -1;
    }

    color = new Color(random.nextInt(256), random.nextInt(256),
        random
        .nextInt(256));
  }

  public void drawOutline(Graphics g) {

    g.setColor(color);

    for (int i = 0; i < nVertices - 1; i++)
      g.drawLine(x[i], y[i], x[i + 1], y[i + 1]);

    g.drawLine(x[nVertices - 1], y[nVertices - 1], x[0], y[0]);
  }

  public void draw(Graphics g) {

    g.setColor(color);
    g.fillPolygon(x, y, nVertices);
  }

  public void move(int width, int height) {

    // Position update
    ox += vox + vox * (random.nextDouble() - 0.5);
    oy += voy + voy * (random.nextDouble() - 0.5);

    // Circular boundary conditions
    int tempX = ox % width;
    if (tempX < -sizeX)
      tempX += width;
    int tempY = oy % height;
    if (tempY < -sizeY)
      tempY += height;

    ox = tempX;
```

```
    oy = tempY;

    flap();
  }

  private void flap() {

    for (int i = 0; i < x.length; i++) {

      x[i] += vx[i];
      y[i] += vy[i];

      // circular boundary conditions
      int relativeX = x[i] - ox;
      relativeX %= sizeX;
      if (relativeX < 0)
        relativeX += sizeX;
      x[i] = ox + relativeX;

      int relativeY = y[i] - oy;
      relativeY %= sizeY;
      if (relativeY < 0)
        relativeY += sizeY;
      y[i] = oy + relativeY;
    }
  }
}
```

Notice that FlappingPolygon implements both Gooable and Serializable. The serializable interface has been included since we expect to need to serialise all the flapping polygons when we close the program down.

Learning activity

Write an application class that populates the window with a few gooables and launches the animation.

6.9 A Goo World application

```
package serialisation;

import java.awt.Color;

public class GooWorldApp {

  public static void main(String[] args) {

    boolean FSE = false;
    GooWorld gooWorld = new GooWorld(500, 500, FSE);

    long seed = System.currentTimeMillis();
    for (int i = 0; i < 10; i++)
      gooWorld.addGooable(
        new FlappingPolygon(gooWorld.getWidth(),
          gooWorld.getHeight(), seed++));
```

```
    gooWorld.background(Color.BLACK);
    gooWorld.smooth();
    gooWorld.go();
  }
}
```

Learning activity

The animation ends when the window is closed. We wish to serialise all the flapping polygons at this moment.

Goo itself is a WindowListener. This means that it implements the WindowListener interface. Various methods are called when the state of a window is changed; amongst these is `windowClosing(WindowEvent e)`, called whenever the window's shutdown icon is pressed.

Extend GooWorld and override windowClosing so that all the gooables are written to disk when the user tries to close the window. Make a call to Goo's windowClosed to terminate the application after windowClosing has accomplished its task.

Also add a method `public void load()` which reads in the saved state of all the gooables, and resumes the animation.

6.10 Start again Goo World

```
package serialisation;

import java.awt.event.WindowEvent;
import java.io.*;
import java.util.ArrayList;

public class RestartableGooWorld extends GooWorld {

  public RestartableGooWorld(int w, int h, boolean b) {

    super(w, h, b);
  }

  public void windowClosing(WindowEvent e) {

    try {
      FileOutputStream fs = new FileOutputStream("gooables.ser");
      ObjectOutputStream os = null;
      os = new ObjectOutputStream(fs);
      os.writeObject(gooables);
      os.close();
    } catch (FileNotFoundException ex) {
      ex.printStackTrace();
    } catch (IOException ex) {
      ex.printStackTrace();
    }

    windowClosed(e);
  }

  public void load() throws FileNotFoundException, IOException,
```

```
      ClassNotFoundException {

    FileInputStream is = new FileInputStream("gooables.ser");
    ObjectInputStream os = new ObjectInputStream(is);

    gooables = (ArrayList<Gooable>) os.readObject();

    os.close();
  }
}
```

The window closing and load methods follow the serialisation/deserialisation recipe from *HFJ*. Notice that since the entire object graph is serialised, it is enough to serialise the Gooables ArrayList.

6.11 Goo World restarted

```
package serialisation;

import java.awt.Color;

public class RestartableGooWorldApp {

  public static void main(String[] args) {

    boolean FSE = false;
    RestartableGooWorld gooWorld = new RestartableGooWorld(500,
        500, FSE);

    // Attempt to load saved objects
    try {
      gooWorld.load();
    } catch (Exception e) {
      // If *anything* goes wrong
      long seed = System.currentTimeMillis();
      for (int i = 0; i < 10; i++)
        gooWorld.addGooable(new FlappingPolygon(gooWorld.getWidth
            (),
            gooWorld.getHeight(), seed++));
    }

    gooWorld.background(Color.BLACK);
    gooWorld.smooth();
    gooWorld.go();
  }
}
```

This application attempts to load the gooables ArrayList from file. If anything goes wrong, a new configuration of flapping polygons is created.

6.12 Learning outcomes

By the end of this chapter, the relevant reading and activities, you should be able to:

- understand that serialisation saves an object's state.
- describe the two different kinds of stream: connection or chain stream, and the differences between them
- understand that chain streams cannot connect to a source or destination
- describe how an object is serialised to a file by chaining a `FileOutputStream` to an `ObjectOutputStream`
- understand that objects, or their super class, must implement `Serializable` if they are to be serialised
- understand that all objects in an object's entire object graph are serialised at the same time
- understand that an exception will be thrown at runtime if any of these other objects does not implement `Serializable`
- understand that a transient instance variable will not be serialised but that it will assume the default/null value when the containing object is restored
- understand that the class of all objects in the graph must be available to the JVM during deserialisation
- understand that objects are read using `readObject()` in the order in which they were serialised
- understand that `readObject` returns an `Object` reference which should be subsequently cast to the objects type (or supertype)
- understand that static variables are not serialised
- use a serial version ID if a class definition may change and render serialised objects of that class incompatible.

Chapter 7

Networking

Essential reading

HFJ Chapter 15.

7.1 Introduction

We are now ready to look at the most fundamental aspect of internet Java: how a Java program can connect to another machine on a network (a client); and how a Java program can sit there waiting for other Java programs to contact it (a server).

7.2 Clients

Reading: pp. 473–477 of *HFJ*.

A `Socket` object represents a TCP connection between two applications: the server and the client application. These applications may exist on separate machines.

The client must know the IP address of the server, and the TCP port of the server application. A TCP port is a 16-bit unsigned number that is assigned to a specific server application. Technically speaking, up to 65536 server applications might be running on a host. The first 1024 ports are reserved for well known services such as FTP, HTTP, SMTP ...

7.3 Sockets

Reading: pp. 478–482 of *HFJ*.

A client connects to port 4200 of the machine at 127.0.0.1 by instantiating a socket,

```
Socket socket = new Socket("127.0.0.1", 4200);
```

and obtains low level connection streams by calling `socket.getInputStream()` and `socket.getOutputStream();`.

These streams can be chained to readers and writers in the usual way.

7.4 Servers

Reading: pp. 483–485 of *HFJ*.

The server application starts listening on a specific port,

```
ServerSocket serverSock = new ServerSocket(4200);
```

and a connection is made to a client by the blocking `accept()` method:

```
Socket socket = serverSocket.accept();
```

This method returns a `Socket` object (remember that all connections are handled with `Sockets`). This socket is on a different port so that the serverSocket can return to listening and handle a new client connection request. The sockets know each others' IP addresses and ports.

7.5 Summary

- A `Socket` object represents a TCP connection between two applications: the server and the client application.
- These applications may exist on separate machines.
- The client must know the IP address of the server, and the TCP port of the server application.
- A TCP port is a 16-bit unsigned number that is assigned to a specific server application. Technically speaking, up to 65536 server applications might be running on a host.
- The first 1024 ports are reserved for well known services such as FTP, HTTP and SMTP.
- A client connects like this: `Socket socket = new Socket("127.0.0.1", 4200);`
- and obtains low level IO connection streams by calling `socket.getInputStream()` and `socket.getOutputStream();`
- Remember to chain these streams to readers and writers in the usual way.
- The server application starts listening on a specific port like this: `ServerSocket serverSock = new ServerSocket(4200);`
- A connection is made to a client by the blocking `accept()` method: `Socket socket = serverSocket.accept();`
- This method returns a `Socket` object (remember that all connections are handled with `Sockets`).
- This socket is on a different port so that the serverSocket can return to listening and handle a new client connection request. The sockets know each other's IP addresses and ports.

7.6 Programming

We shall begin by writing a very simple client/server pair. Although basic, such programs are used to confirm that remote machines are connected. We will then move on to writing an object server that can deliver the serialised state of an object to a remote location. This more useful programming task demonstrates how objects can be passed from machine to machine on the internet.

Learning activity

Suppose we wish to contact a Hello Client sever on port 6006. Such a server replies to connection requests with the message “Hello Client” and then terminates the connection. Write a class, `SimplestClient` that attempts to contact a Hello Client server, running on port 6006 on a machine at a specified IP address.

7.7 Simplest client

```
package networking;

import java.io.*;
import java.net.*;

public class SimplestClient {

  private int port = 6006;

  public void go(String host) {
    try {
      System.out.println("Contacting " + host + " on port " +
          port);

      Socket socket = new Socket("localhost", port);
      BufferedReader reader = new BufferedReader(new
          InputStreamReader(
           socket.getInputStream()));

      System.out.println(reader.readLine());

      socket.close();
    } catch (UnknownHostException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  public static void main(String[] args) {
    new SimplestClient().go(args[0]);
  }
}
```

Learning activity

Now write the Hello Client server, `SimplestServer`. If you are working in a networked lab, or know someone who is: (a) connected to the internet, and (b) willing to run your software on their machine, install your server on this second machine. Find the IP address of the server's machine by running **io.MyAddress.java**. Write the address down or have it emailed to you.

Now start the server.

Test your server and client by starting the client, remembering to point the client at the IP address of the server.

If you are working alone, then start the server on your machine, and point the client at "localhost" or "127.0.0.1" , which your computer will recognise as the IP address of itself. Alternatively, if your machine is connected to the internet, start the server and point the client at the IP address of your machine (which you can find by running MyAddress).

7.8 Simplest server

```
package networking;

import java.io.*;
import java.net.*;

public class SimplestServer {

  private int port = 6006;

  public void go() {
    try {

      ServerSocket serverSocket = new ServerSocket(port);
      System.out.println("Server started on port " + port);

      Socket socket = serverSocket.accept();

      PrintWriter writer = new PrintWriter(
          socket.getOutputStream());
      writer.println("Hello Client");
      writer.flush();
      serverSocket.close();

      socket.close();

      System.out.println("Server closed");
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

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

Learning activity

Write a Gooables server.

Such a server replies to connection requests by sending a serialised ArrayList of gooables. You have already serialised some gooables in a file **gooables.ser** which your server can read and then write to the output stream.

7.9 Gooables server

```
package networking;

import java.io.*;
import java.net.*;

public class GooablesServer {

  private int port = 6006;
  public void go() {

    try {

      ServerSocket serverSocket = new ServerSocket(port);
      System.out.println("Gooable Server started on port " + port
          );

      Socket socket = serverSocket.accept();

      ObjectInputStream ois = new ObjectInputStream(new
          FileInputStream(
          "gooables.ser"));

      ObjectOutputStream oos = new ObjectOutputStream(socket
          .getOutputStream());

      Object obj = ois.readObject();
      oos.writeObject(obj);

      ois.close();
      oos.close();
      serverSocket.close();
      socket.close();

      System.out.println("Server closed");

    } catch (IOException e) {
      e.printStackTrace();
    } catch (ClassNotFoundException e) {
      e.printStackTrace();
    }
  }

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

Learning activity

Finally write a Gooables client. This client connects to the gooables server. If the connection is successful, an ArrayList of gooables is received in a serialised form and the client instantiates a RestartableGooWorld, setting GooWorld's gooable list to the gooables that have just been recovered from a remote site.

7.10 Gooables client

```
package networking;

import java.awt.Color;
import java.io.*;
import java.net.*;
import java.util.ArrayList;

import serialisation.Gooable;
import serialisation.RestartableGooWorld;

public class GooablesClient {

  private int port = 6006;

  public void go(String host) {

    try {
      System.out.println("Contacting " + host + " on port " +
          port);

      Socket socket = new Socket("localhost", port);

      ObjectInputStream ois = new ObjectInputStream(socket.
          getInputStream());
      ArrayList<Gooable> gooables = (ArrayList<Gooable>)ois.
          readObject();

      boolean FSE = false;
      RestartableGooWorld gooWorld = new RestartableGooWorld(500,
           500, FSE);

      gooWorld.addGooables(gooables);
      gooWorld.background(Color.BLACK);
      gooWorld.smooth();
      gooWorld.go();

      ois.close();
      socket.close();
    } catch (UnknownHostException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
    catch(ClassNotFoundException e){
      e.printStackTrace();
    }
  }
```

```
  public static void main(String[] args) {
    new GooablesClient().go(args[0]);
  }
}
```

7.11 Learning outcomes

By the end of this chapter, the relevant reading and activities, you should be able to:

- describe how a `Socket` object represents a TCP connection between two applications: the server and the client application
- understand that these applications may exist on separate machines
- understand that the client must know the IP address of the server, and the TCP port of the server application
- understand that a TCP port is a 16-bit unsigned number that is assigned to a specific server application
- understand how a client connects and how it obtains low-level IO streams
- describe how a connection is made to a client by the blocking `accept()` method and that this method returns a `Socket` object
- understand that this socket is on a different port so that the serverSocket can return to listening and handle a new client connection request, and that the sockets know each other's IP addresses and ports
- write a server and a client for serving and receiving Strings
- write a server and a client for serving and receiving Objects.

Chapter 8

Threads

Essential reading

HFJ Chapter 15.

8.1 Introduction

So far in this volume we have discovered three key aspects of Java programming: streams, exceptions and serialisation. There is one further Java technology to master: threading.

All modern machines are multi-threaded, meaning that many programs can appear to be running at the same time. This has tremendous time saving potential because, for example, if a program is waiting for something to happen (receive data, a button press), a single-threaded machine would become idle. But in a multi-threaded environment, other programs can usefully run some of their instructions while other programs are blocked.

Luckily the JVM is multi-threaded, and the Java language gives you the chance to create your own threads (think of these as streams of execution).

Threading is a very common internet programming technique since network connections are inherently unpredictable. All servers are multi-threaded, enabling many client requests to be dealt with simultaneously.

8.2 Multi-threading in Java

Reading: pp. 490–494 of *HFJ*.

A thread (small 't') is a single executing process. Multi-threaded computers can run many threads (apparently) simultaneously. A thread is represented in Java by `Thread` (capital 'T').

Each thread has its own call stack and a `Runnable` object which defines the task. A Runnable object must have a run() method because this is defined by the Runnable interface. This method is placed on the bottom of the call stack.

For example, `main` starts in the main thread. The main method is placed at the foot of the call stack. main starts a new thread,

```
Runnable myRunnable = new MyThreadJob();
```

```
Thread myThread  = new Thread(myRunnable);
myThreadstart();
Dog dog = new Dog();
...
```

When start is called on the Thread object, the Java Virtual Machine (JVM) prepares a new call stack and places t's `run()` method at the bottom of the new stack. The JVM switches between the two call stacks, executing instructions from both stacks until the threads are complete.

Study the diagrams on pages 491 and 492 of *HFJ*.

A thread is launched by passing a Runnable object to the Thread's constructor. The thread object initialises a Runnable instance variable to point at the object you have provided. When your code calls start on the Thread object, the Thread object invokes run() on its Runnable instance variable.

Carefully study the code and diagram on page 494.

8.3 States of thread

Reading: pp. 495–496 of *HFJ*.

A thread can be in one of four states: NEW, RUNNABLE, RUNNING and BLOCKED. A Thread is NEW when it has been instantiated but its start() method has not yet been called. A thread becomes RUNNABLE when start() is called. Eventually the JVM's thread scheduler will select s RUNNABLE thread and begin execution of the Runnable's run, moving this thread to the state RUNNING. Only one thread can run at any moment on a single-processor machine. The scheduler may later return the currently executing thread to RUNNABLE so that another thread can have a chance to execute some code. However a thread may move to BLOCKED if, for example, the thread is waiting for data, is sleeping for a while, or is trying to access a locked object.

8.4 The thread scheduler

Reading: pp. 497–503 of *HFJ*.

The JVM's thread scheduler makes all the decisions about which thread moves from runnable to running, and from there to blocked or runnable. Thread scheduling is not predictable so you cannot write your code assuming any particular order of execution. The diagrams on pg. 499 illustrate how the same doubly threaded program (one thread for main, plus one more) can produce different results.

You can make a multi-threaded program a little more predictable by putting your threads to sleep every now and then. It is very unlikely that the thread's sleep will be interrupted; however, you cannot guarantee that the thread will reawaken exactly at the specified interval.

8.5 Concurrency problems

Thread programming is potentially hazardous. Concurrency in this context means ensuring that data has only one current value from each thread's perspective. Concurrency problems may arise when two or more threads access a method on a single object.

Reading: pp. 504–508 of *HFJ*.

The Monica and Ryan scenario shows just how serious it can be if a thread is placed to runnable somewhere in the middle of a method. The problem can be cured if only one thread is allowed to access a method at any one time.

Reading: pp. 509–511 of *HFJ*.

The keyword `synchronized` can be used to make methods (and even statements) thread-safe. A thread, when attempting to enter a synchronized method of an object, must obtain the object's 'key' to unlock the object. There is only one object key, so only one thread can enter a synchronized method at any one time. In fact all the synchronized methods of an object become locked.

Reading: pp. 512–515 of *HFJ*.

```
int i = balance;
balance = i + 1;
```

The rather innocuous piece of code can cause major concurrency problems. Each thread accessing this code will add 1 to whatever `i` was when it was previously read by the thread, which is rather different from adding 1 to `balance`'s current value.

The lost update scenario is again solved with locks and keys. However, synchronization should be used sparingly because: (i) method access is slowed down; (ii) concurrency is restricted because other threads are forced to wait in line; and (iii) potential deadlock.

Reading: pp. 516–517 of *HFJ*.

Deadlock can happen with just two threads and two objects. Each thread is awaiting to obtain the key that the other thread holds. The result is that both threads will freeze. There is no mechanism within Java to avoid this deadly problem.

8.6 Other techniques

A thread can also be made by subclassing Thread and overriding `run()`. Calling start on your Thread object will then invoke the overridden run method. Although this will work perfectly well, it does not make good OO sense unless you subclass IS-A Thread. Besides which, since it is only possible to extend a single class, the subclassed Thread cannot fit into any inheritance tree you have designed.

The other technique is to use an anonymous class.

Reading: pg. 666 of *HFJ*.

Creating anonymous classes is often a good design choice because the class is defined right where it is needed. It wouldn't be such a good choice if objects of that class were needed elsewhere in your code, and if the class definition is long.

8.7 Summary

- A Java thread is a single program execution and is represented in Java by the class java.lang.Thread
- Each thread has its own call stack.
- Each thread has a Runnable object which defines the task.
- A Runnable object must have a run() method. This method is placed on the bottom of the call stack.
- The thread is launched by passing a Runnable object to the Thread's constructor and calling start() on the Thread object.
- A thread can be in one of four states: NEW, RUNNABLE, RUNNING and BLOCKED.
- A thread is NEW when it has been instantiated but its start() method has not yet been called.
- A thread becomes RUNNABLE when start() is called.
- Eventually the JVM's thread scheduler will select a RUNNABLE thread and begin execution of the Runnable's run, moving this thread to the state RUNNING. Only one thread can run at any moment on a single-processor machine.
- The scheduler may later return the currently executing thread to RUNNABLE so that another thread can have a chance to execute some code.
- A thread may move to BLOCKED if, for example, the thread is waiting for data, is sleeping for a while, or is trying to access a locked object.
- Thread scheduling is not predictable so you cannot write your code assuming any particular order of execution.
- Thread programming is hazardous. For example, if two threads access the same object, data may become corrupted if the thread is moved to RUNNABLE while manipulating the critical state.
- The keyword `synchronized` can be used to make methods (and even statements) thread-safe.
- A thread, when attempting to enter a synchronized method of an object, must obtain the object's 'key' to unlock the object. There is only one object key, so only one thread can enter a synchronized method at any one time. In fact, all the synchronized methods of an object become locked.
- Synchronization should be used sparingly because: (i) method access is slowed down; (ii) concurrency is restricted because other threads are forced to wait in line; and (iii) because of the hazards of potential deadlock.

8.8 Programming

This programming section starts with a simple clock program, threaded in three different ways. You will then write a threaded Object server and client. As we have mentioned, threaded servers and clients are essential for networked Java, since it enables our programs to get on with other things if there is a delay. In fact threading your program can be useful in other situations as well. For example, the GUIs that you have been using and writing are automatically threaded by the JVM. This means that user input can occur at any time; we don't need to put the program on hold while waiting for someone to press a button.

The final, and most ambitious program, is a thread pool server, similar in principle to the very popular Apache server. This will be the most sophisticated program that we have written to date.

Learning activity

Write three threaded Java clocks that print the time at one second intervals at the command line. You should use the three thread techniques: implementing Runnable, subclassing Thread and using an anonymous class. Write a clock application that launches all three clocks by invoking the instance method `public void go()` on each Clock object.

Study the output carefully to see how the scheduler handles the three threads.

8.9 Tick tock

```
package threads;

import static java.util.Calendar.*;
import java.util.Calendar;

public class Clock {

  class MyRunnable implements Runnable {

    public void run() {

      int i = 0;
      while (i++ < 60) {

        Calendar now = Calendar.getInstance();
        now.setTimeInMillis(System.currentTimeMillis());

        int h = now.get(HOUR);
        int m = now.get(MINUTE);
        int s = now.get(SECOND);

        System.out.println("Clock 1\t" + h + ":"
            + (m < 10 ? "0" + m : m) + ":"
            + (s < 10 ? "0" + s : s));
```

```
        try {
          Thread.sleep(1000);
        } catch (InterruptedException e) {
        }
      }
    }
  }

  public void go() {

    Thread t = new Thread(new MyRunnable());
    t.start();
  }
}
```

```
package threads;

import static java.util.Calendar.*;
import java.util.Calendar;

public class Clock2 extends Thread {

  public void run() {

    int i = 0;
    while (i++ < 60) {

      Calendar now = Calendar.getInstance();
      now.setTimeInMillis(System.currentTimeMillis());

      int h = now.get(HOUR);
      int m = now.get(MINUTE);
      int s = now.get(SECOND);

      System.out.println("Clock 2\t" + h + ":" + (m < 10 ? "0" +
          m : m) + ":"
          + (s < 10 ? "0" + s : s));

      try {
        Thread.sleep(1000);
      } catch (InterruptedException e) {
      }
    }
  }

  public void go(){

    start();
  }
}
```

```
package threads;

import static java.util.Calendar.*;
import java.util.Calendar;

public class Clock3 {

  public void go() {
```

```
    Thread t = new Thread(new Runnable() {

      public void run() {

        int i = 0;

        while (i++ < 60) {

          Calendar now = Calendar.getInstance();
          now.setTimeInMillis(System.currentTimeMillis());

          int h = now.get(HOUR);
          int m = now.get(MINUTE);
          int s = now.get(SECOND);

          System.out.println("Clock 3\t" + h + ":"
              + (m < 10 ? "0" + m : m) + ":"
              + (s < 10 ? "0" + s : s));

          try {
            Thread.sleep(1000);
          } catch (InterruptedException e) {
          }
        }
      }

    });
    t.start();
  }

  public static void main(String[] args) {

    new Clock3().go();
  }
}
```

```
package threads;

public class ClockApp {

  public static void main(String[] args){

    new Clock().go();
    new Clock2().go();
    new Clock3().go();
  }
}
```

The three clock and their application are a straightforward implantation of thread techniques. Make sure you understand these programs.

Learning activity

Write a threaded Gooables server. The server should reply to the connection request by sending a serialised array list of Gooables. Use an inner ClientHandler class to deal with the transaction.

8.10 Threaded Gooables server

```
package threads;

import java.io.*;
import java.net.*;

public class ThreadedGooablesServer {

  final int PORT = 6006;

  public class ClientHandler implements Runnable {

    Socket socket;

    public ClientHandler(Socket clientSocket) {
      socket = clientSocket;
    }

    public void run() {

      try {
        ObjectOutputStream oos = new ObjectOutputStream(socket
            .getOutputStream());

        ObjectInputStream ois = new ObjectInputStream(
            new FileInputStream("gooables.ser"));

        oos.writeObject(ois.readObject());

        ois.close();
        oos.close();
        socket.close();

      } catch (IOException e) {
      } catch (ClassNotFoundException e) {
      }
    }
  }


  public void go() {

    boolean running = true;

    try {

      ServerSocket serverSocket = new ServerSocket(PORT);
      System.out.println("Gooable Server started on port " + PORT
          );

      while (running) {

          Socket clientSocket = serverSocket.accept();
          Thread t = new Thread(new ClientHandler(clientSocket));
          t.start();
      }
```

```
      serverSocket.close();
      System.out.println("Server closed");

    } catch (IOException e) {
    }
  }



  public static void main(String[] args) {

    ThreadedGooablesServer server = new ThreadedGooablesServer();
    server.go();
  }
}
```

go() starts the server. A ServerSocket is instantiated and accept() is called inside a while loop.

Each request is handled by a handler object in a new thread. This enables the server to carry on listening while servicing current client requests which might take a while to complete (such as reading from disc, as in this case).

Here, and in the network applications that follow, exception catch blocks have been left empty in order to reduce the size of the code. In practice various diagnosis printouts would be called, or other action taken to maintain the functionality of the application.

Learning activity

Write a class UserIn which reads command strings entered on the command line. The UserIn object should then inform all registered listeners.

8.11 Command line control

```
package threads;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.SocketException;
import java.util.ArrayList;

public class UserIn {

  ArrayList<CommandLineListener> listeners;

  public UserIn() {
    listeners = new ArrayList<CommandLineListener>();
    go();
  }

  public UserIn(CommandLineListener l) {
    listeners = new ArrayList<CommandLineListener>();
```

```
    listeners.add(l);
    go();
  }

  public void addListener(CommandLineListener l) {
    listeners.add(l);
  }

  public void go() {

    final BufferedReader reader = new BufferedReader(new
        InputStreamReader(
        System.in));

    Thread t = new Thread(new Runnable() {
      public void run() {
        try {
          boolean running = true;
          while (running) {
            String s = reader.readLine();
            for (CommandLineListener l : listeners)
              l.userIn(s);
            if (s.equalsIgnoreCase(".close")){
              running = false;
            }
          }
        } catch (SocketException e) {
        } catch (IOException e) {
        }
      }
    });
    t.start();
  }
}
```

UserIn listens at the command line, informing listeners of typed input. `readLine()`–which blocks–is threaded, which means that an application can employ a UserIn object to read from the command line, and get on with other activities at the same time.

In this coding, we have decided that special input strings commencing with '.' mean that the user has typed a command to UserIn. In this case '.close' will set `running` to false, hence terminating the `while` loop; `run` then completes and terminates the thread.

We decide that listeners should implement a common interface so that we can be sure that the call back method exists. Such an interface, CommandLineListener, is listed below.

```
package threads;

public interface CommandLineListener {
  public void userIn(String s);
}
```

Learning activity

Modify ThreadedGooablesServer so that it registers with UserIn. The server *and* UserIn should terminate upon receipt of the command string '.close'.

8.12 Threaded Gooables server with control

```
package threads;

import java.io.*;
import java.net.*;

public class InteractiveThreadedGooablesServer implements
    CommandLineListener {

  final int PORT = 6006;
  private boolean running = false;
  final int TIME_OUT = 1000;

  public class ClientHandler implements Runnable {

    Socket socket;

    public ClientHandler(Socket clientSocket) {
      socket = clientSocket;
    }

    public void run() {

      try {
        ObjectOutputStream oos = new ObjectOutputStream(socket
            .getOutputStream());

        ObjectInputStream ois = new ObjectInputStream(
            new FileInputStream("gooables.ser"));

        oos.writeObject(ois.readObject());

        ois.close();
        oos.close();
        socket.close();

      } catch (IOException e) {
      } catch (ClassNotFoundException e) {
      }
    }
  }

  public void go() {

    running = true;

    try {

      ServerSocket serverSocket = new ServerSocket(PORT);
      serverSocket.setSoTimeout(TIME_OUT);
```

```
      System.out.println("Gooable Server started on port " + PORT
          );

      while (running) {

        try {
          Socket clientSocket = serverSocket.accept();
          Thread t = new Thread(new ClientHandler(clientSocket));
          t.start();
        } catch (SocketTimeoutException e) {
        }
      }

      serverSocket.close();
      System.out.println("Server closed");

    } catch (IOException e) {
    }
  }

  public void close(){
    running = false;
  }

  public void userIn(String s){
    if(s.equalsIgnoreCase(".close"))
      close();
  }

  public static void main(String[] args) {

    InteractiveThreadedGooablesServer server = new
        InteractiveThreadedGooablesServer();
    new UserIn(server);
    server.go();
  }
}
```

`main` instantiates an InteractiveThreadedGooablesServer, and then a userIn object. The server is registered with the UserIn object by passing a reference in the constructor call. UserIn starts a new thread for command line input, allowing `main` to continue. The server is then started.

`accept()` is blocking, which means that execution stalls until a connection request is made. This means that the while loop cannot be terminated during a call to accept (except of course by closing down the JVM with `System.exit()`). However, a timeout interval is set on the ServerSocket. This means that `accept` will cease waiting for a connection after the required interval and program flow will move to the next line of code. A SocketTimeoutException is immediately thrown. Without this timeout, setting `running` to false would not terminate the `while` until after the next connection.

Learning activity

Write a threaded Gooables client which runs five `serialisation.RestartableGooWorlds`. Each GooWorld contacts the threaded Gooables server in order to acquire the initial population of FlappingPolygons.

8.13 Threaded Gooables client

```
package threads;

import java.awt.Color;
import java.io.*;
import java.net.*;
import java.util.ArrayList;

import serialisation.Gooable;
import serialisation.RestartableGooWorld;

public class ThreadedGooablesClient {

  private int port = 6006;

  public void go(String host) {

    try {
      System.out.println("Contacting " + host + " on port " +
          port);

      Socket socket = new Socket("localhost", port);
      ObjectInputStream ois = new ObjectInputStream(socket.
          getInputStream());
      ArrayList<Gooable> gooables = (ArrayList<Gooable>)ois.
          readObject();
      ois.close();
      socket.close();

      boolean FSE = false;
      RestartableGooWorld gooWorld = new RestartableGooWorld(500,
           500, FSE);

      gooWorld.addGooables(gooables);
      gooWorld.background(Color.BLACK);
      gooWorld.smooth();
      gooWorld.go();
    } catch (UnknownHostException e) {
    } catch (IOException e) {
    }
    catch(ClassNotFoundException e){
    }
  }

  public static void main(final String[] args) {

    int numClients = 5;
    for(int i = 0; i < numClients; i++){

      Thread t = new Thread(new Runnable(){
        public void run(){
          new ThreadedGooablesClient().go(args[0]);
        }
      });
      t.start();
```

```
      try {
        Thread.sleep(1000);
      } catch (InterruptedException e) {
      }
    }
  }
}
```

Learning activity

You will build a threaded object server in this activity. This server operates rather like a chat server; it distributes objects sent by a client to all other currently connected clients. You can base your program on the very simple chat server on pages 520–521 of *HFJ*.

8.14 Object server

```
package threads;

import java.io.*;
import java.net.*;
import java.util.ArrayList;

public class ObjectServer implements CommandLineListener {

  ArrayList<ClientHandler> clientHandlers;
  private boolean running = false;
  final int TIME_OUT = 1000;

  public class ClientHandler implements Runnable {

    ObjectInputStream objInputStream;
    ObjectOutputStream objOutputStream;
    SocketAddress address;
    Socket clientSocket;

    public ClientHandler(Socket sckt) {
      clientSocket = sckt;
      address = clientSocket.getRemoteSocketAddress();
    }

    public void close() {

      try {
        System.out.println("Closing connection to " + address);
        objInputStream.close();
        objOutputStream.close();
      } catch (IOException e) {
      }
    }

    public void run() {
      try {
        objInputStream = new ObjectInputStream(clientSocket
            .getInputStream());
        objOutputStream = new ObjectOutputStream(clientSocket
```

```
            .getOutputStream());

        boolean running = true;

        while (running) {
          Object obj = objInputStream.readObject();

          System.out.println("Object received from "
              + clientSocket.getRemoteSocketAddress());
          tellEveryone(obj, this);
        }
      } catch (EOFException e) {
      } catch (IOException e) {
      } catch (ClassNotFoundException e) {
      }

    }
}

public void go() {

    clientHandlers = new ArrayList<ClientHandler>();

    try {

      ServerSocket serverSock = new ServerSocket(5000);
      serverSock.setSoTimeout(TIME_OUT);

      running = true;
      while (running) {

        try {
          Socket clientSocket = serverSock.accept();
          System.out.println("Connected to "
              + clientSocket.getRemoteSocketAddress());

          ClientHandler ch = new ClientHandler(clientSocket);
          clientHandlers.add(ch);

          new Thread(ch).start();
        } catch (SocketTimeoutException e) {
        }

      }
      serverSock.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
}

public void close() {

    running = false;
    for (ClientHandler ch : clientHandlers) {
      ch.close();
    }
}

public void userIn(String s) {
```

```
    if (s.equalsIgnoreCase(".close"))
      close();
  }

  public void tellEveryone(Object object, ClientHandler sender) {

    for (ClientHandler handler : clientHandlers) {

      if (handler != sender) {

        ObjectOutputStream oos = handler.objOutputStream;

        try {
          oos.writeObject(object);
          oos.flush();

        } catch (SocketException e) {
          e.printStackTrace();
        } catch (IOException e) {
          e.printStackTrace();
        }
      }
    }
  }

  public static void main(String[] args) {

    ObjectServer server = new ObjectServer();
    new UserIn(server);
    server.go();
  }

}
```

ObjectServer largely follows *HFJ*'s `VerySimpleChatServer` except the server deals in objects rather than Strings. However, since a String is an object, our ObjectServer could be used as a chat server. There are a few differences between this code and VerySimpleChatServer. The client handlers are stored in a list, rather than the output streams, and an extra method `close` enables the input/output streams associated with a connection to be closed.

Learning activity

Write an object client that can send, and receive, objects to/from your object server. Use a `UserIn` object to send Strings to all connected clients.

8.15 Object client

```
package threads;

import java.io.*;
import java.net.*;

public class ObjectClient implements CommandLineListener {
```

```
ObjectOutputStream objOutputStream;
InetSocketAddress address;
String DEFAULT_HOST = "localhost";
int DEFAULT_PORT = 5000;
boolean isConnected = false;

class IncomingReader implements Runnable {

  ObjectInputStream objInputStream;

  public IncomingReader(ObjectInputStream ois) {

    objInputStream = ois;
  }

  public void run() {

    try {
      boolean running = true;
      while (running) {

        Object obj = objInputStream.readObject();
        receive(obj);
      }
    } catch (EOFException e) {
      System.out.println("Lost connection to " + address);
    } catch (SocketException e) {
    } catch (IOException e) {
    } catch (ClassNotFoundException e) {
    }
  }
}

public void go(){
  go(DEFAULT_HOST, DEFAULT_PORT);
}

public void go(String host, int port) {

  address = new InetSocketAddress(host, port);
  try {
    Socket sock = new Socket(host, port);
    objOutputStream = new ObjectOutputStream(sock.
        getOutputStream());
    ObjectInputStream objInputStream = new ObjectInputStream(
        sock
        .getInputStream());
    System.out.println("Connected to " + address + " from port
        "
        + sock.getLocalPort());

    isConnected = true;

    Thread readerThread = new Thread(new IncomingReader(
        objInputStream));
    readerThread.start();

  } catch (UnknownHostException e) {
```

```
      isConnected = false;
    } catch (IOException e) {
      System.out.println("Could not connect to " + address);
      isConnected = false;
    }
  }

  public void userIn(String inString) {

    if(inString.equalsIgnoreCase(".close"))
      System.exit(0);

    send(inString);
  }

  public boolean send(Object obj) {

    try {
      objOutputStream.writeObject(obj);
      objOutputStream.flush();
      return true;

    } catch (IOException e) {
      System.out.println("Could not send object");
    } catch (NullPointerException e) {
      System.out.println("Could not send object");
    }
    return false;
  }

  public void receive(Object obj) {

    System.out.println(obj);
  }

  public boolean isConnected(){

    return isConnected;
  }

  public static void main(String[] args) {

    ObjectClient client = new ObjectClient();
    new UserIn(client);

    if (args.length < 2) {
      client.go();
    }

    try {
      client.go(args[0], Integer.parseInt(args[1]));
    } catch (NumberFormatException e) {
    }
  }
}
```

The code listing seems long but it is based on our previous examples and should look familiar.

An inner `IncomingReader` class deals with incoming objects. `readObject` in `run` awaits an incoming object, sending it to `receive`. Here we just print the object to the terminal, but in general any object processing could take place here.

Outgoing objects are sent via `send`. Once more, any object could be sent, but here we are sending strings typed at the command line and passed to `userIn()`.

A Boolean variable `isConnected` monitors the state of the client; this allows any clients of ObjectClient to check that a connection does indeed exist before attempting to send an object.

Learning activity

Each client request to the above servers is dealt with in a separate thread, but the total number of threads is unlimited. A thread pool can be used to limit the number of simultaneous connections.

Write a thread pool Gooables server.

8.16 Thread pool Gooables server

```
package threads;

import java.io.*;
import java.net.*;

public class ThreadPoolGooablesServer {

  private int port = 6006;
  ServerSocket serverSocket;
  ClientHandler[] handlers;
  final int NUM_THREADS = 10;

  public class ClientHandler implements Runnable {

    ServerSocket serverSocket;
    int threadNumber;

    public ClientHandler(ServerSocket s, int num) {

      serverSocket = s;
      threadNumber = num;
    }

    public void run() {

      while (true) {

        try {
          Socket clientSocket;

          synchronized (serverSocket) {

            System.out.print("Thread: " + threadNumber
                + " waiting...");
            clientSocket = serverSocket.accept();
```

```
          }
          System.out.println("responding  ");

          ObjectOutputStream oos = new ObjectOutputStream(
              clientSocket.getOutputStream());

          ObjectInputStream ois = new ObjectInputStream(
              new FileInputStream("gooables.ser"));

          oos.writeObject(ois.readObject());

          ois.close();
          oos.close();
        } catch (SocketException e) {
          System.out.println(threadNumber + " has been terminated
              ");
        } catch (IOException e) {
          e.printStackTrace();
        } catch (ClassNotFoundException e) {
          e.printStackTrace();
        }
      }
    }
  }

  public void go() {

    try {

      serverSocket = new ServerSocket(port);
      System.out.println("Gooable Server started on port " + port
          );

      handlers = new ClientHandler[NUM_THREADS];

      for (int i = 0; i < NUM_THREADS; i++) {

        handlers[i] = new ClientHandler(serverSocket, i);
        Thread t = new Thread(handlers[i]);
        t.start();
      }

    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  public static void main(String[] args) {

    ThreadPoolGooablesServer server = new
        ThreadPoolGooablesServer();
    server.go();
  }
}
```

go() creates NUM_THREADS ClientHandlers and places them in an array. Each ClientHandler's thread is started and each thread tries to get a key for serverSocket.accept(). Notice how the serverSocket.accept() statement is placed in a

synchronized code block,

```
synchronized (serverSocket) {
    System.out.print("Thread: " + threadNumber + " waiting...");
    clientSocket = serverSocket.accept();
}
```

so that only one of the ClientHandler threads can access the serverSocket and run `accept()` at any one time. This lucky thread then waits until a client request comes in. Program flow then moves outside the synchronized block, allowing another ClientHandler thread (we cannot know which one) from the pool of handlers to obtain the key.

8.17 Learning outcomes

By the end of this chapter, the relevant reading and activities, you should be able to:

- understand that Java thread is a single program execution
- understand that each thread has its own call stack
- understand that each thread has a Runnable object which defines the task
- describe how a Runnable object must have a run() method, and that this method is placed on the bottom of the call stack
- describe how a thread is launched
- describe the four states in which a thread can be: NEW, RUNNABLE, RUNNING and BLOCKED
- understand that a thread is NEW when it has been instantiated but its start() method has not yet been called
- describe when a thread is in the state RUNNING
- describe how a thread may move to the state BLOCKED
- understand that thread scheduling is not predictable so you cannot write your code assuming any particular order of execution
- understand that thread programming is hazardous, and give examples of why this is the case
- understand that the keyword `synchronized` can be used to make methods (and even statements) thread-safe
- understand how a thread, when attempting to enter a synchronized method of an object, must obtain the object's 'key' to unlock the object
- understand that there is only one object key, so only one thread can enter a synchronized method at any one time
- understand the reasons why synchronization should be used sparingly
- understand in principle how to build a server using a thread pool.

Chapter 9

Distributed computing

Essential reading

HFJ Chapters 17, 18.

9.1 Introduction

This chapter covers various aspects of distributed Java. Java programs may run dynamically on servers, generating web content on the fly (servlets); small applications may be downloaded from an HTTP server to a browser (applets); and finally, objects can talk to other objects on different heaps using the Remote Method Invocation (RMI) mechanism.

9.2 RMI

Reading: pp. 607–610 of *HFJ*.

The JVM will only allow objects to communicate within the same heap. However, you might wish to split your programming task between different machines. Maybe you have a small handheld Java device and you want to contact a more powerful machine to run your program; or contact a database at a remote location. RMI is designed to allow a call to a method on a different heap, running under a different instance of the JVM, usually on a different computer.

9.2.1 Helpers

Reading: pp. 611–614 of *HFJ*.

An object on one heap cannot get a normal Java reference to an object on another heap. Java RMI fools the JVM into thinking it is running a method on a local object, when in fact the object is at a remote location. A client calls a method on a proxy (stub). The stub is a client helper, taking care of the networking detail (sockets, streams, serialisation, etc.).

9.2.2 Making the remote service

Reading: pp. 615–618 of *HFJ*.

The remote service must implement your remote interface. It must extend `java.rmi. Remote`, and all methods must declare RemoteException. The remote service should extend UnicastRemoteObject. The remote service must have a constructor which declares a RemoteException and it must be instantiated and the object registered with the RMI registry using the static `Naming.rebind()`. The RMI registry must be running on the same machine as the remote service. The client looks up the remote service using the static `Naming.lookup()`.

9.2.3 Example code

Reading: pp. 619–623 of *HFJ*.

Some example code for the remote interface, remote service and client is provided in HFJ. Notice that the client must have the stub class definition in order to deserialise the stub. The class file can be distributed by hand, although there is a mechanism known as dynamic class downloading that can automate the process. The stub object is stamped with a URL which enables the RMI to download the class file from an HTTP server.

9.3 Servlets

Reading: pp. 625–628 of *HFJ*.

Servlets are Java classes that run on an HTTP server. They are used for running code on the server as a result of interaction with a web page.The servlet classes are not part of the standard distribution but are obtainable from the Java website, `http://java.sun.com`. Servlets can only be run by servlet-enabled servers such as Tomcat.

A typical servlet extends HttpServlet and overrides one or more servlet methods such as `doGet()` or `doPost()`. The web server starts the servlet and calls the appropriate method based on the client's request. The servlet sends back a message by getting a PrintWriter output stream from the response parameter of the `doGet()` method.The servlet then writes the HTML page, complete with tags.

The Java Servlet API is a framework for writing servlets (Java programs): application components for web services. Compare this to applets which are application components for browsers. The API lies in `java.servlet` package, a standard extension (i.e. not part of core Java).

In the dark ages, there were other server-side applications such as the common gateway interface or CGI, used in conjunction with a scripting language such as Perl. But now most web servers have Java servlet containers (engines). Servlets are becoming more and more popular.

Why is this the case?

It is because you can write web applications (apps) in Java and derive all the benefits of a powerful language and VM environment. Also, Java is faster than scripting languages, especially in server-side environments where long-running apps can be optimised by the VM. And servlets have an additional speed advantage over CGI programs because they can use threads within one instance of the JVM. Java has

unique run-time safety, very important for servers, where an errant transaction could crash the server.

But above all, large and complex apps are much more manageable in a full-blown language such as Java. Servlets are easier to update than scripts (although harder to write) and scale better for high volume applications. The servlet can access all the standard Java APIs within the VM **and** continue to handle requests. So database connections can be 'live' within JDBC, and other networking services can be handled (in CGI this feature was a hack).

9.3.1 Servlet lifecycle

Firstly `init()` is called by the server. This initialises all the data and other objects required by the servlet.

Then, `service()` handles requests.

Finally `destroy()` is called when the server is closed down, in order to clean up.

The `service()` method accepts two parameters: a servlet 'request' object and a servlet 'response' object.

Servlets are expected to handle multi-threaded requests which means that you cannot store client-related data in instance variables of your servlet object. You must use a client session object such as a cookie.

9.3.2 HTTP requests

HttpServlet is the base class of the `javax.servlet.http` package. This is an abstract servlet that provides some basic implementation related to handling a HTTP request. It overrides the generic service request and breaks it out into several HTTP-related methods, including `doGet()` (corresponding to the HTTP GET operation). This is the standard request for retrieving a file or a document at a specified URL. The query string is attached to the URL and sent to the server.

`doPost()` corresponds to a POST operation. This is for sending an arbitrary amount of data to the server. The file is sent via an output stream.

`doPut()` and `doDelete()`: these two are obscure and allow files to be placed and removed. Not widely used.

Study the code examples on page 627 of *HFJ*.

9.4 Relationship with JSP

What is a Java Server Page (JSP), and how does it relate to servlets? A servlet is a Java class that contains HTML in output statements but a JSP is an HTML page that contains Java code. JSP are dynamic web pages written in normal HTML, but with embedded Java that is triggered by tags at runtime. The main advantage is that it is easier to write Java inside HTML, than to write HTML in the servlet's print statements.

9.5 Applets

An applet is a Java application that runs inside a web browser. In contrast, a servlet is a Java application that runs on a server. The applet code is downloaded with the web page.

One advantage that Java applets have over Java applications is that applets can be easily deployed over the web while Java applications require installation. Moreover, since applets are downloaded from the Internet, by default they run in a restricted security environment, called the 'sandbox'.

The early hopes for applets have not been fully realised because: (i) not all browsers are shipped with Java plug-ins; and (ii) due to the popularity of web scripting languages such as Flash. Furthermore, the Java Web Start (JWS) technology provides an alternative to applets (see below).

9.5.1 Applets are safe

If asked if you wish to download an applet, you probably shouldn't worry too much. They have been designed with security in mind. Applets **cannot**:

- access arbitrary memory addresses
- access the local file system
- launch other programs on the client
- load native libraries or native method calls
- use System.getProperty() to reveal information
- manipulate any Thread that is not in its own ThreadGroup
- open network connections to any host other than the originating host
- listen to ports below 1024.

9.5.2 Applications and applets

An application is a standalone program, started by calling a main method. Applets do not have a main method. Instead, several methods are called at different times in the execution of an applet.

An applet is made by creating a subclass of java.applet.Applet in which you override the init method to initialise your applet's resources. init might be called more than once and should be designed accordingly. GUI components are added directly to an applet (they are added to a frame in applications).

9.5.3 Applets and HTML

You have an applet, archived in `MyApplet.jar` and you wish to deploy it in an HTML page. You also wish the local browser to display an image `loading.gif` while the applet is loading. This is how you do it:

```
<applet code=MyApplet.class width = "500" height = "500">
<param name = "image" value="loading.gif">
</applet>
```

If you want to use any Swing components in an applet, you should subclass JApplet.

9.6 Lifecycle

A web browser will download an applet when it sees an applet tag. The server downloads binary data which is checked by the byte code verifier. The class file is instantiated using its no-args constructor.

Then, `init()` and `start()` are invoked in turn.

Applets have four lifecycle methods:

1. `init()` Initialise the applet (like a constructor).
2. `start()` Invoked when the page is loaded, reloaded or revisited by the Back button.
3. `stop()` The user is leaving this page, or the applet is scrolled off-screen.
4. `destroy()` The applet is being removed from memory. Here you can close files, network connections, etc.

9.7 Deployment

Reading: pp. 581–595 of *HFJ*.

You have worked hard on your code and eventually you have an application that you wish to share (or even sell!). But how do you go back to actually deploying – or distributing – your code? We have explored some techniques of network distributed code in the above sections. But now we need to understand how to distribute your code, which may consist of many classes and several packages, as a single file.

A Java application may be local (deployed as an executable jar) or remote (e.g. servlets); or it may be a combination of the two (Java Webstart, RMI). Whatever the nature of the application, the compiled class files need to be organised and bundled-up.

We may wish to separate source from compiled files. The Java compiler can be instructed, with the use of the **-d** flag, to place the compiled classes in a separate directory, perhaps named **classes** or **bin**. If you have been using an IDE such as Eclipse, this might have been happening automatically.

The next step is to archive your classes and packages in what is called a jar (Java Archive) file. The jar file is executable. On some operating systems, the application can be launched by double clicking on the jar. *HFJ* pp. 585-586 and pg. 592 take you through the necessary steps.

We have already been carefully organising our Java code in packages, as explained in *HFJ* pp. 588-591.

9.8 Java Web Start

Reading: pp. 597–601 of *HFJ*.

Java Web Start enables you to deploy a stand-alone client application from the web. The application runs as a stand-alone without the constraints of a browser. JWS is able to detect changes in the application on the server and automatically update the client's previously downloaded JWS application.

JWS technology includes a helper app that must be installed, along with Java, on the client machine. A JWS application has an executable jar, and a .jnlp file.

The .jnlp file describes your JWS application with tags for the name and location of the jar, and the name of the class with the main method. A browser starts up the JWS helper after downloading the .jnlp file. The browser then requests the executable jar from the web server. The client can start JWS at a later time and relaunch the application without the use of a browser.

9.9 Summary

- An object on one heap cannot get a normal Java reference to an object on another heap.
- Java Remote Method Invocation (RMI) fools the JVM into thinking it is running a method on a local object, when in fact the object is at a remote location.
- A client calls a method on a proxy (stub).
- The stub is a client helper, taking care of the networking detail (sockets, streams, serialisation, etc.).
- The remote service must implement your remote interface.
- The remote interface must extend java.rmi. Remote, and all methods must declare RemoteException.
- The remote service should extend UnicastRemoteObject.
- The remote service must have a constructor which declares a RemoteException.
- The remote service must be instantiated and the object registered with the RMI registry using the static Naming.rebind().
- The RMI registry must be running on the same machine as the remote service.
- The client looks up the remote service using the static Naming.lookup().
- RemoteExceptions are checked by the compiler. They are thrown at registration, at look up, and by all remote calls from the client to the stub.
- Servlets are Java classes that run on an HTTP server.
- Servlets are used for running code on the server as a result of interaction with a web page.
- The servlet classes are not part of the standard distribution but are obtainable from java.sun.com.
- Servlets can only be run by servlet-enabled servers such as Tomcat.
- A typical servlet extends HttpServlet and overrides one or more servlet methods such as doGet() or doPost().

- The web server starts the servlet and calls the appropriate method based on the client's request.
- The servlet sends back a message by getting a PrintWriter output stream from the response parameter of the doGet() method.
- The servlet then writes the HTML page, complete with tags.
- Applets are Java programs running inside a browser.
- Applets have very restricted terms of operation.
- Swing components are added directly to the applet.
- The applet does not have a constructor, but it has four life cycle methods corresponding to the four actions on a web page: initialise, display, leave or hide page and clean-up.
- Your Java application can be bundled into a single archive file, known as a jar.
- The jar contains a manifest file which points to the main() method.
- A jar can be executed by typing **java - jar MyJar.jar** at the command line, or by double clicking.
- Java Web Start enables you to deploy a stand-alone client application from the web.
- JWS technology includes a helper app that must be installed, along with Java, on the client machine.
- A JWS application has an executable jar, and a .jnlp file.
- The .jnlp file describes your JWS application with tags for the name and location of the jar, and the name of the class with the main method.
- A browser starts up the JWS helper after downloading the .jnlp file.
- The browser then requests the executable jar from the web server.
- The client can start JWS at a later time and relaunch the application without the use of a browser.

9.10 Programming

Run manyworlds.ManyWorldApp from the CD-ROM.

You are watching the flight of clouds of blobs (left hand world) and flapping polygons (right hand world). The two worlds are connected by two 'worm holes'. The exit and entrances to the worm holes are shown by the dark grey discs. Objects from either world, when flying over the exit disc in their world will 'fall' into the other world, appearing at the exit disc.

You will observe that the objects are mini-goo animations, similar to those you coded in Volume 1, except that these mini-animations move across the screen to form a larger animation. You can think of each type of mini-animation as a creature.

In this demo, the two worlds are launched from a single application on a single computer.

Imagine now that worlds are running on separate machines, connected by worm holes that extend across the planet.

Imagine that there are many worlds connected by a tangle of wormholes, and many different species of goo creature populating the worlds. You are watching your

world, awaiting the arrival of an exotic species. Maybe you are designing your own species, introducing creatures of this species into your world, and letting them disperse throughout the goo'niverse.

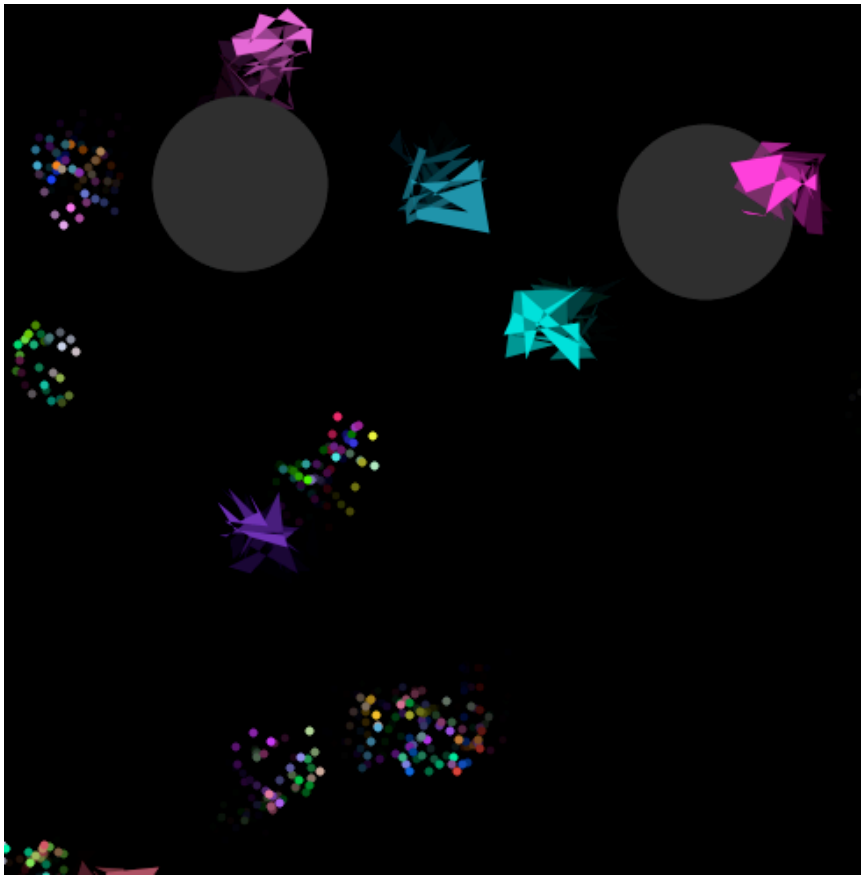

Figure 9.1: Screen shot of a Goo World showing butterflies and snow drops. The grey worm holes connect this world to others on the Internet

---

Learning activity

Write a GooWorlds class. A GooWorld has two discs: exit and arrival. The GooWorld is populated by Gooables. Gooables move through the world, and have internal motion as well; each Gooable is a mini-animation.

Include an inner class, `ManyWorldsObjectClient extends threads.ObjectClient` and override `receive()` so that incoming objects (i.e. from the internet) can be added to the list of Gooables in your world. Also invoke `ObjectClient.send()` whenever one of your Gooables moves over the exit disc. Remember to remove this Gooable from your animation.

9.11 Many Goo worlds

```
package manyworlds;

import goo.Goo;

import java.awt.*;
import java.util.ArrayList;
import java.util.Random;

import threads.ObjectClient;

public class GooWorld extends Goo {

  ArrayList<Gooable> gooables;
  ArrayList<Gooable> departures;
  ArrayList<Gooable> arrivals;
  ObjectClient objectClient;
  boolean connected;

  int exitHoleSize;
  int exitHoleX, exitHoleY, exitCentreX ,exitCentreY;
  double exitRadiusSquared;

  int entryHoleSize;
  int entryHoleX, entryHoleY;

  Color entryHoleColor, exitHoleColor;

  Random random;

  long then = System.currentTimeMillis();

  class ManyWorldsObjectClient extends ObjectClient {

    public void receive(Object obj) {
      arrivals.add((Gooable) obj);
    }
  }

  public GooWorld(int w, int h, boolean b, long seed) {

    super(w, h, b);

    background(new Color(0, 0, 0, 96));

    gooables = new ArrayList<Gooable>();
    departures = new ArrayList<Gooable>();
    arrivals = new ArrayList<Gooable>();

    objectClient = new ManyWorldsObjectClient();
    objectClient.go();

    random = new Random(seed);
    exitHoleSize = 100;
    exitHoleX = random.nextInt((getWidth() - exitHoleSize));
    exitHoleY = random.nextInt((getHeight() - exitHoleSize));
```

```
    exitCentreX = exitHoleX + exitHoleSize / 2;
    exitCentreY = exitHoleY + exitHoleSize / 2;
    exitRadiusSquared = exitHoleSize * exitHoleSize / 8.0;

    entryHoleSize = 100;
    entryHoleX = random.nextInt((getWidth() - exitHoleSize));
    entryHoleY = random.nextInt((getHeight() - exitHoleSize));

    exitHoleColor = new Color(50, 50, 50, 128);
    entryHoleColor = new Color(50, 50, 50, 128);
  }

  public void addGooable(Gooable g) {

    gooables.add(g);
  }

  private void tunnelOut() {

    if (!departures.isEmpty()) {
      for (Gooable g : departures) {
        if (objectClient.send(g))
          gooables.remove(g);
        else{
          connected = false;
        }
      }
      departures.clear();
    }
  }

  private void tunnelIn() {

    if (!arrivals.isEmpty()) {
      for (Gooable g : arrivals) {
        g.setX(entryHoleX + entryHoleSize / 2);
        g.setY(entryHoleY + entryHoleSize / 2);
        gooables.add(g);
      }
      arrivals.clear();
    }
  }

  public void draw(Graphics g) {

    g.setColor(exitHoleColor);
    g.fillOval(exitHoleX, exitHoleY, exitHoleSize, exitHoleSize);

    g.setColor(entryHoleColor);
    g.fillOval(entryHoleX, entryHoleY, entryHoleSize,
        entryHoleSize);

    if (objectClient.isConnected()) {

      tunnelOut();
      tunnelIn();
    }

    for (Gooable gooable : gooables) {
```

```
      gooable.move(getWidth(), getHeight());

      int x = gooable.getX() + gooable.getWidth() / 2;
      int y = gooable.getY() + gooable.getHeight() / 2;

      int a = x - exitCentreX;
      int b = y - exitCentreY;

      if(a*a + b * b < exitRadiusSquared)
        departures.add(gooable);

      gooable.draw(g, 0, 0);
    }
  }

  public ArrayList<Gooable> getGooables() {

    return gooables;
  }

  public void addGooables(ArrayList<Gooable> g) {

    gooables = g;
  }
}
```

Separate lists are maintained for the inhabiting Gooables, and for any departing and arriving Gooables. This is to avoid concurrency problems in the Gooables move/draw loop. tunnelOut is invoked first (if a connection exists) and Gooables are removed from the list; then tunnelIn is invoked and any arrivals may enter the world.

The draw method just draws the world, which just consists of the exit and entry wormholes. The Gooables are asked to draw themselves.

The following interface defines what a Gooable is, extending somewhat on our previous definition, `serialisation.Gooable`.

9.12 Gooable

```
package manyworlds;

import java.awt.Graphics;

public interface Gooable {

  public void draw(Graphics g, int x, int y);

  public void move(int extWidth, int extHeight);

  public void setX(int i);

  public int getX();

  public void setY(int i);
```

```
  public int getY();

  public int getWidth();

  public void setWidth(int i);

  public int getHeight();

  public void setHeight(int i);
}
```

A Gooable has external coordinates (x, y). `draw(Graphics g, int x, int y)` draws a Gooable at (x, y) and `move(int extWidth, int extHeight)` updates (x, y) in a rectangle of left top corner $(0, 0)$, right bottom corner $(extWidth, extHeight)$. In other words, the Gooable keeps track of its own position, (x, y), but the calling object passes in the extent $(extWidth, extHeight)$ of the space so that `move()` can take suitable action at the boundaries.

In addition a Gooable contains other internal Gooables that move and are drawn in a smaller rectangle, $(0, 0) \rightarrow (width, height)$. Once more this Gooable stores the boundaries of the internal space.

The following class:

`Box implements Gooable, Serializable`

represents the internal space of the mini-animation. A box moves at constant speed through a GooWorld. When it leaves one edge of the world it reappears on the opposite edge (circular boundary conditions). The Box contains a list of Gooables that move within the dimensions of the Box, constituting the inner-animation.

9.13 Box

```
package manyworlds;

import java.awt.Color;
import java.awt.Graphics;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.Random;

public class Box implements Gooable, Serializable {

  private static final long serialVersionUID = 1L;

  // The width and height of this gooable
  int width, height;

  // Position and velocities
  int x, y, vx, vy, vMax;

  // Colour of interior fills and lines
  Color color;

  // Random object
  Random random;
```

```
// List of any contained gooables
ArrayList<Gooable> gooables;

public Box(int outerWidth, int outerHeight, int vm, int mWidth,
    int mHeight, long seed) {

  random = new Random(seed);
  init(outerWidth, outerHeight, vm, mWidth, mHeight);
}

public Box(int outerWidth, int outerGeight, int vm, int mWidth,
    int mHeight, Random r) {

  random = r;
  init(outerWidth, outerGeight, vm, mWidth, mHeight);
}

public void init(int outerWidth, int outerHeight, int vm, int w
    , int h) {

  x = random.nextInt(outerWidth);
  y = random.nextInt(outerHeight);

  vMax = vm;

  width = w;
  height = h;

  vx = 1 + random.nextInt(vMax);
  if (random.nextDouble() > 0.5)
    vx *= -1;
  vy = 1 + random.nextInt(vMax);
  if (random.nextDouble() > 0.5)
    vy *= -1;

  color = new Color(random.nextInt(256), random.nextInt(256),
      random
       .nextInt(256));

  gooables = new ArrayList<Gooable>();
}

public void addGooable(Gooable g) {
  gooables.add(g);
}
// Coordinates (x, y) of this gooable
// (ox, oy) coordinate origin of caller
public void draw(Graphics g, int ox, int oy) {

  g.setColor(color);

  // Uncomment if you want to see the bounding box
  // g.drawRect(ox + x, oy + y, width, height);

  // Draw any contained gooables
  for (Gooable gooable : gooables)
    gooable.draw(g, ox + x, oy + y);
}
```

```
  // This gooable moves inside a rectangle of size outerWidth x
      outerHeight
  public void move(int outerWidth, int outerHeight) {

    // Move the box
    x += vx + vx * (random.nextDouble() - 0.5);
    y += vy + vy * (random.nextDouble() - 0.5);
    x = x % outerWidth;
    if (x < -width)
      x += outerWidth;
    y = y % outerHeight;
    if (y < -height)
      y += outerHeight;

    // Move any contained gooables
    for (Gooable gooable : gooables)
      gooable.move(width, height);
  }

  public int getX() {
    return x;
  }

  public int getY() {
    return y;
  }

  public void setX(int i) {
    x = i;
  }

  public void setY(int i) {
    y = i;
  }

  public void setBoxWidth(int i) {
    width = i;
  }

  public void setBoxHeight(int i) {
    height = i;
  }

  public int getHeight() {
    return height;
  }

  public int getWidth() {
    return width;
  }

  public void setHeight(int i) {
    height = i;
  }

  public void setWidth(int i) {
    width = i;
  }
}
```

Box is a long class, although the getters and setters occupy much lineage. The important methods are move and draw. The former moves the box within a rectangle $(0,0) \rightarrow (outerWidth, outerHeight)$. It then asks any contained Gooables to move inside the box itself, of size $(0,0) \rightarrow (width, height)$. `draw` optionally draws the box, but more importantly asks each contained Gooable to draw itself.

Learning activity

Write two Gooables that might form the constituents of a mini-animation. For example, a Gooable might be a simple blob.

Think carefully about the appearance of the Gooables, what internal coordinates they need, how they move and what happens when they hit an edge.

9.14 Blob

```
package manyworlds;

import java.awt.Graphics;
import java.util.Random;

public class Blob extends Box{

  private static final long serialVersionUID = 1L;

  public Blob(int outerWidth, int outerHeight, int vmax, int
      miniWidth, int miniHeight,
       long seed) {
    super(outerWidth, outerHeight, vmax, miniWidth, miniHeight,
        seed);

  }

  public Blob(int outerWidth, int outerHeight, int vmax, int
      miniWidth, int miniHeight,
       Random r) {

    super(outerWidth, outerHeight, vmax, miniWidth, miniHeight, r
        );
  }

  public void draw(Graphics g, int ox, int oy) {

    g.setColor(color);
    g.fillOval(ox + x, oy + y, width, height);
  }

  public void move(int outerWidth, int outerHeight) {

    // Position update
    x += vx;
    y += vy;

    // Reflected boundary conditions
    if (x + width > outerWidth) {
```

```
      x = 2 * outerWidth - x - 2 * width;
      vx = -vx;
    } else if (x < 0) {
      x = -x;
      vx = -vx;
    }
    if (y + height > outerHeight) {
      y = 2 * outerHeight - y - 2 * height;
      vy = -vy;
    } else if (y < 0) {
      y = -y;
      vy = -vy;
    }
  }
}
```

This is a very simple Gooable. The object bounces at a boundary (reflected boundary conditions). Blob extends Box which stores the blob coordinates.

There are many other animations you could make based on this pattern.

9.15 Butterflies

```
package manyworlds;

import java.awt.Graphics;
import java.util.Random;

public class Butterfly extends Box{

  private static final long serialVersionUID = 1L;


  int[] xP, yP, vxP, vyP;
  int vPMax;

  int maxVertices, nVertices;


  public Butterfly(int outerWidth, int outerHeight, int vmax, int
       miniWidth, int miniHeight,
      long seed) {
    super(outerWidth, outerHeight, vmax, miniWidth, miniHeight,
        seed);

  }

  public Butterfly(int outerWidth, int outerHeight, int vmax, int
       miniWidth, int miniHeight,
      Random r) {

    super(outerWidth, outerHeight, vmax, miniWidth, miniHeight, r
        );
  }

  public void init(int outerWidth, int outerHeight, int vm, int w
      , int h) {
```

```
    super.init(outerWidth, outerHeight, vm, w,h);

    // Internal param init
    vPMax = Math.min(width, height) / 3;
    maxVertices = 20;
    nVertices = maxVertices / 2;

    xP = new int[nVertices];
    yP = new int[nVertices];
    vxP = new int[nVertices];
    vyP = new int[nVertices];

    for (int i = 0; i < nVertices; i++) {

      xP[i] = random.nextInt(width);
      yP[i] = random.nextInt(height);

      vxP[i] = 1 + random.nextInt(vPMax - 1);
      if (random.nextDouble() > 0.5)
        vxP[i] *= -1;

      vyP[i] = 1 + random.nextInt(vPMax - 1);
      if (random.nextDouble() > 0.5)
        vyP[i] *= -1;
    }
  }

  public void draw(Graphics g, int ox, int oy) {

    super.draw(g, ox, oy);

    g.setColor(color);


    int[] xx = new int[nVertices];
    int[] yy = new int[nVertices];
    for(int i = 0; i < nVertices; i++){
      xx[i] = ox + x + xP[i];
      yy[i] = oy  + y + yP[i];
    }

    g.fillPolygon(xx, yy, nVertices);
  }

  public void move(int outerWidth, int outerHeight) {

    super.move(outerWidth, outerHeight);
    for (int i = 0; i < nVertices; i++) {

      xP[i] += vxP[i];
      yP[i] += vyP[i];

      // circular boundary conditions
      xP[i] %= width;
      if (xP[i] < 0)
        xP[i] += width;

      yP[i] %= height;
      if (yP[i] < 0)
```

```
        yP[i] += width;
    }
  }
}
```

This Butterfly is essentially a flapping polygon. Since the internal structure is more complicated than a blob, the Butterfly has internal coordinate arrays, as well as a position that is stored in its super class.

Learning activity

Finally write an application that populates two Worlds. Run an object server on localhost and launch your application.

9.16 Many Worlds application

```
package manyworlds;

import goo.Goo;
import threads.ObjectServer;
import threads.UserIn;

public class ManyWorldsApp {

  public static void main(String[] args) {

    Thread t = new Thread(new Runnable() {

      public void run() {
        ObjectServer server = new ObjectServer();
        new UserIn(server);
        server.go();
      }

    });

    t.start();

    final int screenWidth = Goo.screenWidth();
    final int screenHeight = Goo.screenHeight();
    final int gooWidth = 9 * screenWidth / 20;
    final int gooHeight = 9 * screenHeight / 10;
    final int horizMargin = (screenWidth - 2 * gooWidth) / 3;
    final int vertMargin = (screenHeight - gooHeight) / 2;

    t = new Thread(new Runnable() {

      public void run() {
        boolean FSE = false;

        long seed = System.currentTimeMillis();

        GooWorld gooWorld = new GooWorld(gooWidth, gooHeight, FSE
            ,
            seed++);
```

```
      gooWorld.setLocation(horizMargin, vertMargin);

      for (int i = 0; i < 10; i++) {

        int vMax = 5;
        int width = gooWorld.getWidth();
        int height = gooWorld.getHeight();
        int boxWidth = 50;
        int boxHeight = 50;

        Box box = new Box(width, height, vMax, boxWidth,
            boxHeight,
            seed++);

        vMax = 10;
        int blobWidth = 5;
        int blobHeight = 5;
        int numGooables = 10;
        for (int j = 0; j < numGooables; j++) {
          box.addGooable(new Blob(boxWidth, boxHeight, vMax,
              blobWidth, blobHeight, seed++));
        }
        gooWorld.addGooable(box);
      }

      gooWorld.smooth();
      gooWorld.go();
    }

  });
  t.start();

  t = new Thread(new Runnable() {

    public void run() {
      boolean FSE = false;

      long seed = 1066;

      GooWorld gooWorld = new GooWorld(gooWidth, gooHeight, FSE
          ,
          seed++);
      gooWorld.setLocation(2 * horizMargin + gooWidth,
          vertMargin);

      for (int i = 0; i < 10; i++) {

        int width = gooWorld.getWidth();
        int height = gooWorld.getHeight();
        int vMax = 5;
        int butterflyWidth = 50;
        int butterflyHeight = 50;
        Box butterfly = new Butterfly(width, height, vMax,
            butterflyWidth, butterflyHeight, seed++);
        gooWorld.addGooable(butterfly);

      }

      gooWorld.smooth();
```

```
        gooWorld.go();
      }

    });
    t.start();

  }
}
```

Notice that there are three threads; one for each world and one for the server (alternatively you could run the object server from another terminal).

A blob cloud is made by adding ten blobs to a box. There are ten clouds in all and each is added to a GooWorld. Ten butterflies are added to another world.

Learning activity

If you are working with a group of other students you can run the fully networked Goo'niverse.

One of you decides to run the object server. You will all need to know the address and port of the server.

Make a species of Gooable, either by following the example patterns of Blob or Butterfly, or simply by implementing Gooable.

Populate a GooWorld with your species, aim your client at the server, launch, sit back and watch what happens.

You may have to guard against over population and over-sized Gooables; GooWorld is a workable framework which you can adapt and improve as necessary.

9.17 Learning outcomes

By the end of this chapter, the relevant reading and activities, you should be able to:

- understand that an object on one heap cannot get a normal Java reference to an object on another heap
- understand that the Java Remote Method Invocation (RMI) fools the JVM into thinking it is running a method on a local object, when in fact the object is at a remote location
- describe how a client calls a method on a proxy (stub) and what a stub is
- understand that the remote service must implement your remote interface and that the RMI registry must be running on the same machine as the remote service
- describe how the client looks up the remote service using the static `Naming.lookup()`
- describe how remote exceptions are checked by the compiler, and how they are thrown at registration, at look up, and by all remote calls from the client to the stub
- understand that servlets are Java classes that run on an HTTP server
- describe what servlets are used for
- understand that the servlet classes are not part of the standard distribution

- understand that servlets can only be run by servlet-enabled servers
- understand that a typical servlet extends HttpServlet and overrides one or more servlet methods
- understand how the web server starts the servlet and calls the appropriate method
- understand how the servlet sends back a message and writes the HTML page
- understand that applets are Java programs running inside a browser and that they have very restricted terms of operation
- understand that Swing components are added directly to the applet.
- explain that the applet does not have a constructor, but it has four lifecycle methods corresponding to the four actions on a web page: initialise, display, leave or hide page and clean-up
- understand that a Java application can be bundled into a single archive file, known as a jar
- describe how a jar can be executed
- describe how Java Web Start enables you to deploy a stand-alone client application from the web
- describe how to use JWS technology
- explain how the client can start JWS at a later time and relaunch the application without the use of a browser.

Chapter 10

Finally . . .

Essential reading

HFJ Chapter 16, Appendix B.

We have come a long way in these two volumes. We began with Eliza, the amusing computer psychoanalyst, coded in a single static method of less than twenty lines of code. We ended with `ManyGooWorlds`, a fully networked and object oriented application.

The two volumes have followed the material and order of development of *Head First Java* fairly closely. A special set of programming projects, including our own Goo animation framework, has been especially developed for this course, which you have hopefully worked through in turn. These projects are integral to the course, since programming is principally a practical rather than a theoretical activity.

An important feature of this course has been a linking of the Java language to the workings of the Java Virtual Machine. It is my belief (shared by the authors of *Head First Java*) that a deeper understanding of how Java actually works will help you with Java programming.

The two main themes of the course, graphical programming and network programming, reflect the more exciting aspects of the Java language. These themes have also been a vehicle for demonstrating the object programming paradigm.

Although we have covered much ground, much of course remains unsaid. However, you are now equipped with the necessary background to continue developing your Java skills and knowledge. You can do this by exploring the extensive Java library and by reading other people's programs.

One important part of the Java library, the Collections framework, is covered in *HFJ* Chapter 16, and various topics are listed in Appendix B. I urge you to read this material; however, it does not form part of the examinable syllabus.

Programming cannot be taught in a single course, or even in a degree program. Programming is a skill, even an art, and it takes many years to become an accomplished programmer.

However, you have made an excellent start.

Chapter 11

Revision

11.1 Overview

11.1.1 Statics

Summary

- A static method can be called using the class name rather than an object reference variable.
- A static method can be invoked without any instances of the method's class on the heap.
- Static methods are used for utility methods that do not depend on a particular instance variable value.
- Static methods are not associated with any particular instance of that class.
- A static method (such as `main`) cannot access a non-static (i.e. instance) method.
- Mark the constructor as `private` if you wish to prevent clients instantiating the class.
- There is only one copy of a static variable and it is shared by all members of the class.
- A static method can access a static variable.
- Java constants are marked `static final`.
- A final static variable is either initialised when it is declared, or in a static initializer block.
- Static finals are conventionally named in uppercase with underscores separating words.
- A final variable cannot be changed after it has been initialised.
- An instance variable can also be marked final. It can only be initialised in the constructor or where it is declared.
- A final method cannot be overridden.
- A final class cannot be extended.
- `java.lang.Math` only has static methods. Some very useful methods in this class are: `random(), abs(), round(), min(), max()`. These, and others are listed in the API.
- All primitive values can be wrapped into objects (`Integer intObj = new Integer(i);` and unwrapped: `(int i = Integer.intValue(intObj);`.
- Java 5 and beyond supports autoboxing: the automatic wrapping and unwrapping of values.
- Wrappers have static utility methods e.g. `Integer.parseInt(s)` and `Double.toString(kmPerSec);`

- A format string uses its own special little language; the format string enables precise control of number printing.
- Date is fine for getting today's date, but use `Calendar` for date manipulation.
- Calendar is abstract: get an instance of a concrete subclass like this: `Calendar c = Calendar.getInstance();`
- Static imports save typing, but can lead to name collisions.
- Static methods encourage the procedural style of programming.

Programs

- VecMath
- VecMathTest
- VecMathUnsafeTest

11.1.2 Exceptions

Summary

- A method can throw an exception object if something goes wrong at runtime.
- All exceptions subclass java.lang.Exception.
- The compiler does not check that possible runtime exceptions are handled in your code.
- However the compiler does care about **checked exceptions**, and insists that they are declared or wrapped in a try/catch block.
- A method throws an exception with the keyword throw followed by a new exceptions object.
- If your code calls a checked exception throwing method, you must either duck the exception or enclose the call in a try/catch block.

Programs

- SafeVecMath
- SafeVecMathTest
- VecMathException
- VecMathExceptionTest

11.1.3 Swing

Summary

- Layout managers control the size and location of nested components.
- The layout manager of the background component determines the size and location of the added component.

- The layout manager uses the preferred size of the added component for its calculations, but whether or not the size is respected depends on the layout manager's policies.
- Use BorderLayout manager to add components to one of five background regions.
- In this case components in the north and south get their preferred heights, but not width. The situation is reversed for the east/west regions. The component in the middle gets whatever is left over, unless you use `pack()`.
- pack() guarantees that the central component gets its preferred size. The size of the other regions depends on how much space is left over.
- Other managers include Flow and Box.
- BorderLayout is the default manager for a frame; FlowLayout is the default for a panel.
- Use setLayout() to change a panel's manager.

Programs

- BetterBrowser
- Fibonacci
- PageLoader
- SimpleBrowser

11.1.4 Streams

Summary

- Chain a FileWriter connection stream to a BufferedWriter for efficient writing.
- A File object represents the path to a file, not the contents of the file.
- Use a File object in preference to a String filename when reading/writing to a stream.
- Chain a FileReader connection stream to a BufferedReader for efficient reading.
- If you need to parse a text file, then you will need to recognise different elements of the file, for example by using s separator.
- String.split() is a convenient way of splitting a string into its individual tokens.
- Use a serial version ID if a class definition may change and render serialised objects of that class incompatible.

Programs

- HostInfo
- Mirror
- MyAddress
- ReadBytes
- SimpleSpider

- SourceSaver
- SourceViewer
- TerminalInput
- Util
- Viewer

11.1.5 Serialisation

Summary

- Serialisation saves an object's state.
- Streams are either connection or chain streams.
- Connection streams represent a connection to a source or destination and must be chained to another stream.
- Chain streams cannot connect to a source or destination.
- An object is serialised to a file by chaining a FileOutputStream to an ObjectOutputStream.
- Then call `writeObject(theObject)` on the ObjectOutputStream.
- Objects, or their super class, must implement Serializable if they are to be serialised.
- All objects in an object's entire object graph are serialised at the same time. An exception will be thrown at runtime if any of these other objects do not implement Serializable.
- A transient instance variable will not be serialised. It will assume the default/null value when the containing object is restored.
- The class of all objects in the graph must be available to the JVM during deserialisation.
- Objects are read using `readObject()` in the order in which they were serialised.
- readObject returns an Object reference which should be subsequently cast to the objects type (or supertype).
- Static variables are not serialised.
- Use a serial version ID if a class definition may change and render serialised objects of that class incompatible.

Programs

- FlappingPolygon
- Gooable
- GooWorld
- GooWorldApp
- RestartableGooWorld
- RestartableGooWorldApp

11.1.6 Networking

Summary

- A `Socket` object represents a TCP connection between two applications: the server and the client application.
- These applications may exist on separate machines.
- The client must know the IP address of the server, and the TCP port of the server application.
- A TCP port is a 16-bit unsigned number that is assigned to a specific server application. Technically speaking, up to 65536 server applications might be running on a host.
- The first 1024 ports are reserved for well known services such as FTP, HTTP and SMTP.
- A client connects like this: `Socket socket = new Socket("127.0.0.1", 4200);`
- and obtains low level IO connection streams by calling `socket.getInputStream()` and `socket.getOutputStream();`
- Remember to chain these streams to readers and writers in the usual way.
- The server application starts listening on a specific port like this: `ServerSocket serverSock = new ServerSocket(4200);`
- A connection is made to a client by the blocking `accept()` method: `Socket socket = serverSocket.accept();`
- This method returns a `Socket` object (remember that all connections are handled with `Sockets`).
- This socket is on a different port so that the serverSocket can return to listening and handle a new client connection request. The sockets know each others' IP addresses and ports.

Programs

- GooablesClient
- GooablesServer
- SimplestClient
- SimplestServer

11.1.7 Threads

Summary

- A Java thread is a single program execution and is represented in Java by the class java.lang.Thread
- Each thread has its own call stack.
- Each thread has a Runnable object which defines the task.
- A Runnable object must have a run() method. This method is placed on the bottom of the call stack.

- The thread is launched by passing a Runnable object to the Thread's constructor and calling start() on the Thread object.
- A thread can be in one of four states: NEW, RUNNABLE, RUNNING and BLOCKED.
- A Thread is NEW when it has been instantiated but its start() method has not yet been called.
- A thread becomes RUNNABLE when start() is called.
- Eventually the JVM's thread scheduler will select s RUNNABLE thread and begin execution of the Runnable's run, moving this thread to the state RUNNING. Only one thread can run at any moment on a single-processor machine.
- The scheduler may later return the currently executing thread to RUNNABLE so that another thread can have a chance to execute some code.
- A thread may move to BLOCKED if, for example, the thread is waiting for data, is sleeping for a while, or is trying to access a locked object.
- Thread scheduling is not predictable so you cannot write your code assuming any particular order of execution.
- Thread programming is hazardous. For example, if two threads access the same object, data may become corrupted if the thread is moved to RUNNABLE while manipulating the critical state.
- The keyword `synchronized` can be used to make methods (and even statements) thread-safe.
- A thread, when attempting to enter a synchronized method of an object, must obtain the object's 'key' to unlock the object. There is only one object key, so only one thread can enter a synchronised method at any one time. In fact, all the synchronized methods of an object become locked.
- Synchronization should be used sparingly because: (i) method access is slowed down; (ii) concurrency is restricted because other threads are forced to wait in line; and (iii) because of the hazards of potential deadlock.

Programs

- Clock
- Clock2
- Clock3
- CommandLineListener
- InteractiveThreadedGooablesServer
- ObjectClient
- ObjectServer
- ThreadedGooablesClient
- ThreadedGooablesServer
- ThreadPoolGooablesServer
- UserIn

11.1.8 Distributed Computing

Summary

- An object on one heap cannot get a normal Java reference to an object on another heap.
- Java Remote Method Invocation (RMI) fools the JVM into thinking it is running a method on a local object, when in fact the object is at a remote location.
- A client calls a method on a proxy (stub).
- The stub is a client helper, taking care of the networking detail (sockets, streams, serialisation, etc.).
- The remote service must implement your remote interface.
- The remote interface must extend java.rmi. Remote, and all methods must declare RemoteException.
- The remote service should extend UnicastRemoteObject.
- The remote service must have a constructor which declares a RemoteException.
- The remote service must be instantiated and the object registered with the RMI registry using the static Naming.rebind().
- The RMI registry must be running on the same machine as the remote service.
- The client looks up the remote service using the static Naming.lookup().
- RemoteExceptions are checked by the compiler. They are thrown at registration, look up and by all remote calls from the client to the stub.
- Servlets are Java classes that run on an HTTP server.
- Servlets are used for running code on the server as a result of interaction with a web page.
- The servlet classes are not part of the standard distribution but are obtainable from java.sun.com.
- Servlets can only be run by servlet-enabled servers such as Tomcat.
- A typical servlet extends HttpServlet and overrides one or more servlet methods such as doGet() or doPost().
- The webserver starts the servlet and calls the appropriate method based on the client's request.
- The servlet sends back a message by getting a PrintWriter output stream from the response parameter of the doGet() method.
- The servlet then writes the HTML page, complete with tags.
- Applets are Java programs running inside a browser.
- Applets have very restricted terms of operation.
- Swing components are added directly to the applet.
- The applet does not have a constructor, but it has four lifecycle methods corresponding to the four actions on a web page: initialise, display, leave or hide page and clean-up.
- Your Java application can be bundled into a single archive file, known as a jar.
- The jar contains a manifest file which points to the main() method.
- A jar can be executed by typing **java - jar MyJar.jar** at the command line, or by double clicking.

- Java Web Start enables you to deploy a stand-alone client application from the web.
- JWS technology includes a helper app that must be installed, along with Java, on the client machine.
- A JWS application has an executable jar, and a .jnlp file.
- The .jnlp file describes your JWS application with tags for the name and location of the jar, and the name of the class with the main method.
- A browser starts up the JWS helper after downloading the .jnlp file.
- The browser then requests the executable jar from the web server.
- The client can start JWS at a later time and relaunch the application without the use of a browser.

Programming

- Blob
- Box
- Butterfly
- Gooable
- ManyWorlds
- ManyWorldsApp

11.2 Sample examination questions, answers and appendices

Important note. The information and advice given in the following section are based on the examination structure used at the time this guide was written. Please note that subject guides may be used for several years. Because of this we strongly advise you to always check the current Regulations for relevant information about the examination. You should also carefully check the rubric/instructions on the paper you actually sit and follow those instructions.

Using the sample examination material. The examination is in two parts: Part A (Volume 1) Graphical programming and Part B (Volume 2) Internet programming. These correspond to Part 1 and Part 2 of this course. You will have to answer two questions from Part A and two questions from Part B of the examination.

Attached are the four Part B papers and the Appendices from recent exams. The Appendices (found at the end of the examination paper) contain materials needed for the questions such as lengthy class definitions, and some summaries from the Java API. The order is as follows: Papers 1 and 2 Part B, Appendices for 1 and 2, Answers to 1 and 2, Papers 3 and 4 Part B, Appendices for 3 and 4, Answers to 3 and 4.

Paper 1 Part B

QUESTION 4

(a) What is a thread?

[1 Marks]

(b) Describe how threads are organised and managed by the Java Virtual Machine (JVM).

[4 Marks]

(c) Describe how threads are created and launched in the Java programming language.

[5 Marks]

(d) Illustrate your answer above by writing a Java programme to create and launch a thread.

[5 Marks]

(e) Why are threads so important in server programming?

[5 Marks]

(f) The programme below shows a simple server application, `SimpleServer`. This server assigns a new thread to each client request, and responds by printing the client address to the command terminal. However the inner handler class has not been completed. Add code to `Handler` so that the server functions as described.

[5 Marks]

Paper 1 Part B

```
// imports...
public class SimpleServer {

    boolean keepGoing = true;

    public SimpleServer() {

        try {
            ServerSocket serverSocket = new ServerSocket(7005);

             while (keepGoing) {

                Socket clientSocket = serverSocket.accept();
                Thread t = new Thread(new Handler(clientSocket));
                t.start();

            }
            serverSocket.close();
        } catch (IOException e) {
            System.out.println(e);
        }
    }

    public static void main(String[] args) {

        new SimpleServer();
    }

    class Handler implements Runnable {

        Socket socket;
        // ADD YOUR CODE HERE
    }
}
```

Paper 1 Part B

QUESTION 5

(a) Write a few lines of Java code to show how a client might attempt to make a TCP connection to a server at 158.223.1.108 on port 5001.

[5 Marks]

(b) Extend your code from the above answer so that the client can read textual messages sent from the server and print them to the command line

[5 Marks]

(c) Write a few lines of code to demonstrate how a Java programme might read from the command line.

[5 Marks]

(d) Write a few lines of code to show how a server can wait for a connection request on a designated port.

[5 Marks]

(e) Explain how you could engineer a peer-to-peer chat application in Java.

[5 Marks]

Paper 1 Part B

QUESTION 6

(a) URLs are represented in Java by the `URL` class. Show, in a few lines of code, how to instantiate a URL object for a given address.

[5 Marks]

(b) Show, in a few lines of code, how a `URL` object `u` can be used to download html source into a `String`.

[5 Marks]

(c) Show, in a few lines of code, how a `URL` object `u` can be used to display a web page using Swing.

[5 Marks]

(d) Extend your answer to the previous question by adding code that allows the user to click hyperlinks on the displayed page and therefore navigate to a new web page. (Hint: Write a LinkFollower class that implements the `HyperlinkListener` interface by supplying a `public void hpyerlinkUpdate(Hyperlinkevent e)` method.)

[10 Marks]

Paper 2 Part B

QUESTION 4

(a) Write a few lines of Java code to show how a client might attempt to make a TCP connection to a server at 158.223.1.108 on port 7005.

[2 Marks]

(b) Extend your code from the above answer so that the client can read and print to the command line textual messages sent from the server

[5 Marks]

(c) Write a few lines of Java code to show how to set-up a TCP server. The server should listen for incoming connection requests on port 7005. Show how the server will make a non-threaded connection to the client, but do not include code to handle the request.

[5 Marks]

(d) In order to deal with many client requests, the server should be threaded. Add a few lines of code to your answer above to demonstrate how this might be done. You should assume the existence of a handler class `Handler implements Runnable`.

[3 Marks]

(e) Now write the Handler class. This class should implement the `Runnable` interface and should respond by sending a textual representation of the current date and time.

[10 Marks]

Paper 2 Part B

QUESTION 5

(a) URLs are represented in Java by the `URL` class. Show, in a few lines of code, how to instantiate a URL object for a given address.

[3 Marks]

(b) Write a few more lines of code to show how a stream can be obtained from a URL object and how characters can be read from this stream.

[5 Marks]

(c) Use your answer above to write a method with the signature `public static ArrayList getTags(URL u) throws IOException` which returns a list of tags from the web page represented by `u`.

[10 Marks]

(d) Web spiders are used by search engines such as Google to compile a directory of web sites. Write, in pseudocode, an algorithm for a web spider. Indicate where in your algorithm you would use the `getTags(URL u)` method.

[7 Marks]

Appendices to Paper 1 and Paper 2

Appendix 2: Class summaries

```
class java.awt.Graphics
abstract void dispose()
abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)
abstract void drawLine(int x1, int y1, int x2, int y2)
abstract void drawOval(int x, int y, int width, int height)
abstract void drawString(String str, int x, int y)
abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)
abstract void fillOval(int x, int y, int width, int height)
abstract void fillRect(int x, int y, int width, int height)
abstract void setColor(Color c)

class java.awt.color
static final black
static final BLACK
static final white
static final WHITE
static final red
static final RED
static final green
static final GREEN
static final blue
static final BLUE

class java.util.Date
public Date()
public boolean after(Date when)
public boolean equals(Object obj)
public boolean before(Date when)
public String toString();

class java.net.InetAddress
public static InetAddress getByName(String host) throws UnknownHostException
public String getHostName()
public String getHostAddress()
public static InetAddress getLocalHost() throws UnknownHostException

class java.net.URL
public URL(String spec) throws MalformedURLException
public final InputStream openStream() throws IOException
public String getHost()

class java.io.InputStream
public abstract int read() throws IOException
public int read(byte[] b) throws IOException
public int read(byte[] b, int off, int len) throws IOException
```

Appendices to Paper 1 and Paper 2

```
public void close() throws IOException

class java.io.OutputStream
public abstract int write() throws IOException
public int write(byte[] b) throws IOException
public int write(byte[] b, int off, int len) throws IOException
public int write(int b) throws IOException
public void close() throws IOException

java.net.Socket
public Socket(InetAddress address, int port) throws IOException
public InputStream getInputStream() throws IOException
public OutputStream getOutputStream() throws IOException
public void close() throws IOException

java.net.ServerSocket
public ServerSocket(int port) throws IOException
public void close() throws IOException
public Socket accept() throws IOException

java.applet.Applet
public URL getCodebase()
public AudioClip getAudioClip(URL u)

java.applet.AudioClip
pubic void play()
pubic void stop()
```

Answers to Paper 1 Part B

QUESTION 4

(a) What is a thread?

[1 Marks]

A thread is a separate line of execution within one JVM

(b) Describe how threads are organised and managed by the Java Virtual Machine (JVM).

[4 Marks]

Every thread has its own call stack

A run() method is placed on a new call stack and begins when the scheduler is ready

Only one thread of execution on a single-processor machine

If a thread becomes blocked for whatever reason, the JVM scheduler will continue execution with another thread (if there is another one)

(c) Describe how threads are created and launched in the Java programming language.

[5 Marks]

You can implement the `Runnable` interface

by suppling a `public void run()` method.

To launch a new thread, pass a `Runnable` to the `Thread`'s constructor

and call `start()` on the `Thread` object

and this will place `run()` on a new stack

(Also award marks for extending Thread, and for showing how to use an adapter)

(d) Illustrate your answer above by writing a Java programme to create and launch a thread.

[5 Marks]

```
class SayGoodbye implements Runnable {

    public void run() {
        System.out.println("Goodbye");
}

    public static void main(String[] args) {

        Thread t = new Thread(new SayGoodbye());
        t.start();
    }
}
```

Answers to Paper 1 Part B

(e) Why are threads so important in server programming?

[5 Marks]

A server can only handle one client at a time

However a request might take a while to complete (for example due to a slow connection) rendering the server inactive

If each client is handled on a separate thread

the thread handling the slow request can become blocked

enabling the scheduler to choose another client-thread and the server can resume useful work

(f) The programme below shows a simple server application, `SimpleServer`. This server assigns a new thread to each client request, and responds by printing the client address to the command terminal. However the inner handler class has not been completed. Add code to `Handler` so that the server functions as described.

[5 Marks]

```
// imports...
public class SimpleServer {

    boolean keepGoing = true;

    public SimpleServer() {

        try {
            ServerSocket serverSocket = new ServerSocket(7005);

             while (keepGoing) {

                Socket clientSocket = serverSocket.accept();
                Thread t = new Thread(new Handler(clientSocket));
                t.start();

            }
            serverSocket.close();
        } catch (IOException e) {
            System.out.println(e);
        }
    }

    public static void main(String[] args) {

        new SimpleServer();
    }

    class Handler implements Runnable {
```

Answers to Paper 1 Part B

```
        Socket socket;
        // ADD YOUR CODE HERE
    }
}
```

Answers to Paper 1 Part B

```
// imports...
public class SimpleServer {

    ...

    class Handler implements Runnable {

        Socket socket;

        public Handler(Socket s) {
            socket = s;
        }

        public void run() {

            System.out.println("Connection from; " + socket);
        }
    }
}
```

Answers to Paper 1 Part B

QUESTION 5

(a) Write a few lines of Java code to show how a client might attempt to make a TCP connection to a server at 158.223.1.108 on port 5001.

[5 Marks]

```
try{
   Socket socket = new Socket("158.223.1.108", 5001);
   //...
   socket.close();
}
catch(IOException e){
   System.out.println(e);
}
```

(b) Extend your code from the above answer so that the client can read textual messages sent from the server and print them to the command line

[5 Marks]

```
try{
   Socket socket = new Socket("158.223.1.108", 5001);

   // code for part (b)
   BufferedReader reader = new BufferedReader(
                              new InputStreamReader(
                                 socket.getInputStream()));
   String line = null;
   while((line = reader.readLine()) != null){
      System.out.println(line);
   }
   reader.close();

   socket.close();

} catch (IOException e) {
   System.out.println(e);
}
```

(c) Write a few lines of code to demonstrate how a Java programme might read from the command line.

[5 Marks]

Answers to Paper 1 Part B

```
String line = null;
try {
  java.io.BufferedReader inputStream = new java.io.BufferedReader(new java.io.InputStreamReader(System.in))
  line = inputStream.readLine();
  inputStream.close();
} catch (java.io.IOException e) {
  System.err.println(e);
}
```

(d) Write a few lines of code to show how a server can wait for a connection request on a designated port.

[5 Marks]

```
try{
   ServerSocket ss = new ServerSocket(5001);
   Socket sock = ss.accept();
   // deal with connection
   sock.close();
   ss.close();
}
catch(IOException e){
   System.out.println(e);
}
```

(e) Explain how you could engineer a peer-to-peer chat application in Java.

[5 Marks]

The application would listen on a designated port using a ServerSocket object

When accept() returns, open an input and an output stream on the socket

These would preferentially be in separate threads

Read and write to these streams from the terminal

Close the streams, sockets and server socket when the conversation is over

Answers to Paper 1 Part B

QUESTION 6

(a) URLs are represented in Java by the `URL` class. Show, in a few lines of code, how to instantiate a URL object for a given address.

[5 Marks]

```
try{
   URL u = new URL("http://www.gold.ac.uk");
   //...
}
catch(MalformedURLException e){
   System.out.println(e);
}
```

(b) Show, in a few lines of code, how a `URL` object `u` can be used to download html source into a `String`.

[5 Marks]

```
try{
   Reader r = new BufferedReader(new InputStreamReader(u.openStream()));
   StringBuffer sb = new StringBuffer();
   int c = 0;
   while ((c = r.read()) != -1) {
      sb.add(c);
   }
   String thePage = sb.toString();
}
catch(IOException e){
   System.out.println(e);
}
```

(c) Show, in a few lines of code, how a `URL` object `u` can be used to display a web page using Swing.

[5 Marks]

```
try {
   JEditorPane jep = new JEditorPane();
   jep.setEditable(false);
   jep.setPage(u);
   JFrame f = new JFrame(homePage);
   f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   f.getContenttPane().add(scrollPane);
   f.setSize(512, 512);
```

Answers to Paper 1 Part B

```
   f.show();
}
catch(IOException e){
   System.out.println(e);
}
```

(d) Extend your answer to the previous question by adding code that allows the user to click hyperlinks on the displayed page and therefore navigate to a new web page. (Hint: Write a LinkFollower class that implements the `HyperlinkListener` interface by supplying a `public void hpyerlinkUpdate(Hyperlinkevent e)` method.)

[10 Marks]

```
/*
 * Display a web page on a swing JEditorPane
 */
package ipj;

import java.io.IOException;
import java.net.URL;

import javax.swing.JEditorPane;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.event.HyperlinkEvent;
import javax.swing.event.HyperlinkListener;

public class SimpleBrowser {

    public static class LinkFollower implements HyperlinkListener {

        private JEditorPane pane;

        public LinkFollower(JEditorPane jep) {
            pane = jep;
        }

        public void hyperlinkUpdate(HyperlinkEvent evt) {
            try {
                if(evt.getEventType() == HyperlinkEvent.EventType.ACTIVATED)
                    pane.setPage(evt.getURL());
            } catch (IOException e) {
                System.out.println(e);
            }
        }
    }

    public static void main(String[] args) {

        JEditorPane jep = new JEditorPane();
        jep.setEditable(false);
        jep.addHyperlinkListener(new LinkFollower(jep));
```

Answers to Paper 1 Part B

```
        try {
            URL u = new URL("http://www.gold.ac.uk");
            jep.setPage(u);
        } catch (IOException e) {
            System.out.println(e);
        }

        JScrollPane scrollPane = new JScrollPane(jep);
        JFrame f = new JFrame();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.getContentPane().add(scrollPane);
        f.setSize(512, 512);
        f.show();
    }
}
```

Answers to Paper 2 Part B

QUESTION 4

(a) Write a few lines of Java code to show how a client might attempt to make a TCP connection to a server at 158.223.1.108 on port 7005.

[2 Marks]

```
try{
   Socket socket = new Socket("158.223.1.108", 7005);
   //...
   socket.close();
}
catch(IOException e){
   System.out.println(e);
}
```

(b) Extend your code from the above answer so that the client can read and print to the command line textual messages sent from the server

[5 Marks]

```
try{
   Socket socket = new Socket("158.223.1.108", 7005);

   // code for part (b)
   BufferedReader reader = new BufferedReader(
                              new InputStreamReader(
                                 socket.getInputStream()));
   String line = null;
   while((line = reader.readLine()) != null){
      System.out.println(line);
   }
   reader.close();

   socket.close();

} catch (IOException e) {
   System.out.println(e);
}
```

(c) Write a few lines of Java code to show how to set-up a TCP server. The server should listen for incoming connection requests on port 7005. Show how the server will make a non-threaded connection to the client, but do not include code to handle the request.

[5 Marks]

Answers to Paper 2 Part B

```
try{
   ServerSocket serverSocket = new ServerSocket(13);
   while(true){
      Socket clientSocket = serverSocket.accept();
      //...
   }
   serverSocket.close();
}
catch(IOException e){
   System.out.println(e);
}
```

(d) In order to deal with many client requests, the server should be threaded. Add a few lines of code to your answer above to demonstrate how this might be done. You should assume the existence of a handler class `Handler implements Runnable`.

[3 Marks]

```
while(true){
      ...
      Thread t = new Thread(new Handler(clientSocket));
      t.start();
}
```

(e) Now write the Handler class. This class should implement the `Runnable` interface and should respond by sending a textual representation of the current date and time.

[10 Marks]

```
class Handler implements Runnable {
   Socket socket;

   public Handler(Socket s) {
      socket = s;
   }

   public void run() {

      System.out.println("Connection from; " + socket);
      try {
         BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
         bw.write(new Date() + "\n");
         bw.close();
         socket.close();
       } catch (IOException e) {
          System.out.println(e);
       }
   }
```

Answers to Paper 2 Part B

QUESTION 5

(a) URLs are represented in Java by the `URL` class. Show, in a few lines of code, how to instantiate a URL object for a given address.

[3 Marks]

```
try{
   URL u = new URL("http://www.gold.ac.uk");
   //...
}
catch(MalformedURLException e){
   System.out.println(e);
}
```

(b) Write a few more lines of code to show how a stream can be obtained from a URL object and how characters can be read from this stream.

[5 Marks]

```
try{
   ...
   Reader r = new BufferedReader(new InputStreamReader(u.openStream()));
   int c = 0;
   while ((c = r.read()) != -1) {
      ...
   }
}
catch(IOException e){System.out.println(e));
```

(c) Use your answer above to write a method with the signature `public static ArrayList getTags(URL u) throws IOException` which returns a list of tags from the web page represented by `u`.

[10 Marks]

```
public static ArrayList getTags(URL u) throws IOException {

   ArrayList tags = new ArrayList();
   Reader r = new BufferedReader(new InputStreamReader(u.openStream()));
   StringBuffer tag = new StringBuffer();

   int c = 0;
   boolean inTag = false;
   while ((c = r.read()) != -1) {
```

Answers to Paper 2 Part B

```
      if (c == '<')
         inTag = true;

      if (inTag)
         tag.append((char) c);

      if (c == '>') {
         tags.add(tag.toString());
         tag = new StringBuffer();
         inTag = false;
      }
   }
   return tags;
}
```

(d) Web spiders are used by search engines such as Google to compile a directory of web sites. Write, in pseudocode, an algorithm for a web spider. Indicate where in your algorithm you would use the `getTags(URL u)` method.

[7 Marks]

```
0. begin with a url of a known site
1. visit url
2. add url to list
3. download html
4. search html for tags using getTags()
5. extract url's from tags
5. for each url found
6.    if url is not in list and url is valid
7.       go to 1.
```

Answers to Paper 2 Part B

QUESTION 6

(a) Explain why an exception handling mechanism is invaluable for network programming.

[5 Marks]

Exceptional things can happen when any computer programmme interacts with the external world

This is because the external world is impossible to predict in detail

Network programmes interact continuously, and run for long periods of time without attention, so exceptions are bound to happen

Applications such as servers should be robust

The handling mechanism enables the programme to recover from the situation by invoking a code block that can respond to the problem

(b) Explain how Java represents exceptions, and how they are thrown and caught. Illustrate your answer with an example.

[10 Marks]

All exception objects subclass java.lang.Exception

Runtime exceptions such as an illegal array index are not checked by the compiler

But all compiler checked exceptions must be handled in a try-catch block

The try block is for normal execution, the catch block is called when something goes wrong:

```
try{
   int b = inputStream.read();
   // ...
}
catch (IOException e){
   b = 0;
   System.out.println(e);
}
```

A method might throw a number of different exceptions

write a different catch block for each exception

starting with the lowest exception

(c) The Java i/o API represents connections to destinations outside the Java Virtual Machine, such as the hard disk, the display, the keyboard, parallel and serial ports and network sockets by streams. How are streams manipulated? What is the advantage in using streams?

[5 Marks]

Answers to Paper 2 Part B

Low level connection streams read and write to the device/port/socket in bytes

High level chain streams deal with higher abstractions such as numbers, characters, Strings and even objects

These are chained together as in

```
InputStream is;
InputStreamReader isr = new InputStreamReader(System.in);
```

The advantage is that we (the applications programmer) use a common interface

And the disntiction between high and low level streams means the API is very flexible and extensible.

(d) Suppose that we wish to read 1024 bytes from an input stream `instream` that has been obtained from a network socket. Write a method

`public static byte[] readBytes(InputStream instream) throws IOException`

that will accompish this, sending back to the caller a reference to the array of 1024 read bytes.

[5 Marks]

```
public byte[] readBytes(InputStream instream) throws IOException{

   int bytesRead = 0;
   int bytesToRead = 1024;
   byte[] input = new byte[bytesToRead]; // 1 mark

   while (bytesRead < bytesToRead) { // 1 mark

      int result = instream.read(input, bytesRead, bytesToRead - bytesRead); // 2 marks
      if (result == -1) // 1 mark
         break;
         bytesRead += result;
       }
   }
   return input;
}
```

Paper 3 Part B

QUESTION 4

(a) Explain, with reference to call stacks, how Java threads give the impression that many processes are happening at once.

[5 Marks]

(b) Once a thread is running, it can move back and forth between one of three states. What are these states and under what circumstances do they occur?

[5 Marks]

(c) Outline the steps in launching a new thread. Include some lines of code to illustrate your answer.

[5 Marks]

(d) Java Servers are usually threaded. Why is this? Outline how a server might use threads.

[5 Marks]

(e) There are a number of dangers in thread programming. What are they?

[5 Marks]

Paper 3 Part B

QUESTION 5

(a) Java uses streams to handle the transfer of data between a programme and the environment (for example, the internet or a printer).

(i) There are two categories of streams. What are they?

(ii) Give an example of a stream from each category.

(iii) Explain briefly how the two categories of stream are used to read or write data.

[5 Marks]

(b) Write a program `SourceSaver.java` that can download the HTML source from a website into an object of type `String`. The programme should be invoked by the command

```
java SourceSaver http://www.gold.ac.uk
```

and must take error handling into account. (Your attention is drawn to the list of class outlines in the Appendix of this paper.)

[10 Marks]

(c) Write a method with the signature

```
private static void writeToFile(String fileName, String text)
```

which saves a `String` to file. Indicate where this method should be called from `SourceSaver` in order to save the downloaded HTML to file.

[5 Marks]

(d) Alter your code in your answer to part (b) so that all HTML tags are stripped from the source code before saving to file.

[5 Marks]

Paper 3 Part B

QUESTION 6

(a) Explain the purpose of `Socket`s in Java internet programming and illustrate, in a couple of lines of code, how they are used.

[5 Marks]

(b) Explain the purpose of `ServerSocket`s in Java internet programming and briefly demonstrate how they are used in a few lines of code.

[5 Marks]

(c) What is object serialisation? Write a few lines for your answer, explaining what parts of an object can be serialized, and how the JVM knows that an object can be serialised.

[5 Marks]

(d) Write code for an object server. This server responds to connection requests by sending a serialized object to the client. The server should not be threaded.

[10 Marks]

Paper 4 Part B

QUESTION 4

(a) Write an account of the reasons why many software developers consider that Java is a good language for internet programming.

[10 Marks]

(b) Provide code examples to illustrate the Java exception handling mechanism.

[5 Marks]

(c) Explain the difference between a runtime exception and a checked exception? Give an example of each type.

[5 Marks]

(d) The methods of `PrintStream` are not recommended for internet applications; one reason for this is that `PrintStream` methods do not throw exceptions. However a class `PrintWriter` implements all the print methods of `printStream` and does throw exceptions. An excerpt from the Java API is reproduced below:

```
-------------------------------------------------------------------------
java.io.PrintWriter

public PrintWriter(OutputStream out, boolean autoFlush)
   Create a new PrintWriter from an existing OutputStream.
   This convenience constructor creates the necessary intermediate
   OutputStreamWriter, which will convert characters into bytes
   using the default character encoding.

Parameters:
   out - An output stream
   autoFlush - A boolean; if true, the println() methods will flush
   the output buffer

public void println(String x)
   Print a String and then terminate the line.
   This method behaves as though it invokes print(String) and then println().
-------------------------------------------------------------------------
```

Modify the common output statement

`System.out.println(String str);`

so that printing to the standard output is managed by a `PrintWriter`.

[5 Marks]

Paper 4 Part B

QUESTION 5

(a) Write a program `WordSaver.java` that can download the HTML source of a website into an object of type `String`. The program should be invoked by the command

```
java WordSaver http://www.gold.ac.uk
```

and should handle any errors. Your attention is drawn to the list of class outlines in the Appendix of this paper.

[10 Marks]

(b) Alter your class `WordSaver` so that all the *words* from the HTML source are added one by one to an `ArrayList<String>`. A word is defined for the purposes of this program as a sequence of letters, where a letter is determined by the method `Character.isLetter(char c)`. Words are separated by any character that is not a letter.

[5 Marks]

(c) (i) Add code to sort the list of words into alphabetical order and print to the command line.

(ii) The list will probably contain numerous identical words, for example "a" and "Goldsmiths". What type of collection will prevent the addition of duplicates?

(iii) Show how to instantiate such a collection from an `ArrayList<String>` of words.

[5 Marks]

(d) Can you think of a practical application for the program you have developed in this question? Explain your idea in a few lines.

[5 Marks]

Paper 4 Part B

QUESTION 6

(a) Show, in a few lines of code, how a client can make a TCP connection with a server.

[5 Marks]

(b) Extend your answer to (a) above by showing how a client can obtain input and output streams from/to the server. What kind of streams are these?

[5 Marks]

(c) Extend your answer to (b) above by showing how would a client could receive an object that has been serialized and despatched by a server.

[5 Marks]

(d) Write code for an object client. The client should contact an object server at 192.168.0.1 and on port 4321, and receive and display the returned object, using the object's `toString()` method.

[10 Marks]

Appendix to Paper 3 and Paper 4

Appendix: Class summaries

```
class java.awt.Graphics
abstract void dispose()
abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)
abstract void drawLine(int x1, int y1, int x2, int y2)
abstract void drawOval(int x, int y, int width, int height)
abstract void drawString(String str, int x, int y)
abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)
abstract void fillOval(int x, int y, int width, int height)
abstract void fillRect(int x, int y, int width, int height)
abstract void setColor(Color c)

class java.awt.color
static final black
static final BLACK
static final white
static final WHITE
static final red
static final RED
static final green
static final GREEN
static final blue
static final BLUE

class java.util.Date
public Date()
public boolean after(Date when)
public boolean equals(Object obj)
public boolean before(Date when)
public String toString();

class java.lang.StringBuffer
public StringBuffer()
public StringBuffer(String str)
public StringBuffer append(char c)
public StringBuffer append(String str)
public StringBuffer delete(int start, int end)
public int indexOff(String str)
public insert(int offset, Srring str)
public int length()
public String toString()

class java.net.InetAddress
public static InetAddress getByName(String host) throws UnknownHostException
public String getHostName()
public String getHostAddress()
```

Appendix to Paper 3 and Paper 4

```
public static InetAddress getLocalHost() throws UnknownHostException

class java.net.URL
public URL(String spec) throws MalformedURLException
public final InputStream openStream() throws IOException
public String getHost()

class java.io.InputStream
public abstract int read() throws IOException
public int read(byte[] b) throws IOException
public int read(byte[] b, int off, int len) throws IOException
public void close() throws IOException

class java.io.OutputStream
public abstract int write() throws IOException
public int write(byte[] b) throws IOException
public int write(byte[] b, int off, int len) throws IOException
public int write(int b) throws IOException
public void close() throws IOException

java.net.Socket
public Socket(InetAddress address, int port) throws IOException
public InputStream getInputStream() throws IOException
public OutputStream getOutputStream() throws IOException
public void close() throws IOException

java.net.ServerSocket
public ServerSocket(int port) throws IOException
public void close() throws IOException
public Socket accept() throws IOException

java.applet.Applet
public URL getCodebase()
public AudioClip getAudioClip(URL u)

java.applet.AudioClip
pubic void play()
pubic void stop()
```

Answers to Paper 3 Part B

QUESTION 4

(a) Explain, with reference to call stacks, how Java threads give the impression that many processes are happening at once.

[5 Marks]

The computer can only deal with a single instruction at any one time.

This is the instruction on the top of the current call stack.

A single Java program can be threaded so that different lines of execution are placed on different call stacks.

Only one is active, the others are frozen.

The Thread scheduler switches execution from the current call stack to one of the frozen ones so quickly it gives the impression that many processes are running at once.

(b) Once a thread is running, it can move back and forth between one of three states. What are these states and under what circumstances do they occur?

[5 Marks]

Runnable: the call stack has been prepared and the thread is ready to run.

Running: the thread is active

The thread scheduler moves the thread between these two states as it tries to optimise performance of the machine

Blocked: a running thread becomes blocked when it is waiting for data

or if it has been put to sleep with a call to sleep()

or the thread is trying to call a locked method on an object

(c) Outline the steps in launching a new thread. Include some lines of code to illustrate your answer.

[5 Marks]

Make a `runnable` object `Runnable job = new MyRunnable();`

Make a Thread object (the worker) and give it a runnable (the job)

```
Thread thread = new Thread(job);
```

Start the thread

```
thread.start();
```

which places the `Runnable`'s run() method onto a new call stack.

(d) Java Servers are usually threaded. Why is this? Outline how a server might use threads.

[5 Marks]

Answers to Paper 3 Part B

Because, if otherwise, a blocked transaction will cause then many other connection requests to also become blocked.

The solution is to use a thread for each connection.

A server operates a pool of threads.

Incoming requests are handled by a pool thread, if there are any, otherwise it is placed in a queue.

Threads that are actively dealing with a request are removed from the pool

and replaced when the transaction is over.

(e) There are a number of dangers in thread programming. What are they?

[5 Marks]

Thread programming is difficult because threads may share memory.

This may lead to concurrency problems if two threads access the same object.

If thread 1 becomes runnable before leaving a method of a shared object, and thread 2 then accesses this object and changes an instance variable, when thread 1 resumes, the rest of its computations may be based on an out-of-date value of that variable.

Deadlock is another problem.

Here, two threads hold keys that the other threads wants. The threads will remain blocked for ever.

Answers to Paper 3 Part B

QUESTION 5

(a) Java uses streams to handle the transfer of data between a programme and the environment (for example, the internet or a printer).

(i) There are two categories of streams. What are they?

(ii) Give an example of a stream from each category.

(iii) Explain briefly how the two categories of stream are used to read or write data.

[5 Marks]

(i) Connection and chain

(ii) Connection: FileOutputStream; Chain: ObjectOutputStream

(iii) The chain stream deals with higher level representations of data; the connection stream acts as a helper and converts the data into bytes for the actual reading/writing operation.

(b) Write a program `SourceSaver.java` that can download the HTML source from a website into an object of type `String`. The programme should be invoked by the command

`java SourceSaver http://www.gold.ac.uk`

and must take error handling into account. (Your attention is drawn to the list of class outlines in the Appendix of this paper.)

[10 Marks]

The following code is just one solution. Award marks for any other sensible program.

```
import java.io.BufferedReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.Reader;
import java.net.MalformedURLException;
import java.net.URL;

class SourceSaver {

    public static void main(String args[]) {

        if (args.length > 0) {
            StringBuffer buff = new StringBuffer();
            try {

                URL u = new URL(args[0]);
                Reader r = new BufferedReader(new InputStreamReader(u
                        .openStream()));
                int c = 0;
                while ((c = r.read()) != -1) {
                    buff.append((char) c);
```

Answers to Paper 3 Part B

```
                }

                String source = buff.toString();
            }

            catch (MalformedURLException e) {
                System.err.println(args[0] + " is not a parseable URL");
            } catch (IOException e) {
                System.err.println(e);
            }

        }
    }
}
```

(c) Write a method with the signature

`private static void writeToFile(String fileName, String text)`

which saves a `String` to file. Indicate where this method should be called from `SourceSaver` in order to save the downloaded HTML to file.

[5 Marks]

```
    .
    .
  String source = buff.toString();
  writeToFile("source.txt", source);
  .
  .
}

    private static void writeToFile(String fileName, String text) {

        try {
            FileWriter writer = new FileWriter(filename + ".html");
            writer.write(text);
            writer.close();
        } catch (IOException e) {
            System.out.print(e);
        }
    }
```

(d) Alter your code in your answer to part (b) so that all HTML tags are stripped from the source code before saving to file.

[5 Marks]

```
int c = 0;
boolean inTag = false;
while ((c = r.read()) != -1) {
   if (c == '<')
      inTag = true;
```

Answers to Paper 3 Part B

```
  if (!inTag)
     buff.append((char) c);

 if (c == '>')
   inTag = false;


}
```

Answers to Paper 3 Part B

QUESTION 6

(a) Explain the purpose of `Socket`s in Java internet programming and illustrate, in a couple of lines of code, how they are used.

[5 Marks]

Sockets represent the connection between two applications
which may or may not be on two different machines (hosts).
Sockets are used for TCP communication.
A client connects to a server like this: `Socket sock = new Socket("127.0.0.1", 4200);`
Once connected, data is transferred by streams: `InputStream str = sock.getInputStream();`

(b) Explain the purpose of `ServerSocket`s in Java internet programming and briefly demonstrate how they are used in a few lines of code.

[5 Marks]

Servers use ServerSockets
to wait for client requests on a particular port.
A ServerSocket object is made `ServerSocket serverSock = new ServerSocket(4200);`
and waits for a connection `serverSocket.accept();`
returning a socket `Socket sock = serverSocket.accept();`

(c) What is object serialisation? Write a few lines for your answer, explaining what parts of an object can be serialized, and how the JVM knows that an object can be serialised.

[5 Marks]

Serialisation is a way of saving an object's state
so that the entire object graph can be read back into a Java program.
Only object variables not modified by the keyword `transient` are serialised (2 marks).
An object must implement the `Serializable` interface.

(d) Write code for an object server. This server responds to connection requests by sending a serialized object to the client. The server should not be threaded.

[10 Marks]

```
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Date;

public class SimpleObjectServer {
```

Answers to Paper 3 Part B

```
    public static int port = 4321;

    public static void main(String[] args) {

        try {
            ServerSocket serverSocket = new ServerSocket(port);

            boolean finished = false;
            while (!finished) {

                Socket socket = serverSocket.accept();

                ObjectOutputStream oos = new ObjectOutputStream(socket
                        .getOutputStream());

                oos.writeObject(new Date());

                oos.close();

            }
            serverSocket.close();
        } catch (IOException e) {
        }
    }
}
```

(e) [**10 Marks**]

Answers to Paper 4 Part B

QUESTION 4

(a) Write an account of the reasons why many software developers consider that Java is a good language for internet programming.

[10 Marks]

Java is was designed with networks in mind.

Java provides solutions to some network problems: platform independence, security, international character sets.

And thanks to the extensive API, little code is needed even for full-scale applications.

Java applets are safer than shrink-wrapped software.

Network programs in Java are easy to write (compared to C).

Java uses threads rather than processes - this is important for scalable web servers.

Java has an exception-handling mechanism - this is important for web applications that need to run continuously.

Java is object oriented, and all software engineering principles can be employed.

Java is a full language, rather than a scripting language, and is therefore very versatile.

Java has only a small execution-speed disadvantage compared to C++ (another object language).

Java has an extensive networking library.

(b) Provide code examples to illustrate the Java exception handling mechanism.

[5 Marks]

```
try{
   inputStream.write(b);
}
catch(IOException e){
   System.out.println(e);
}
```

Alternatively the enclosing method can throw the exception, leaving exception handling to the calling method

```
void sendData(byte[] data) throws IOException{
   .
   .
   .
    inputStream.write(b);
    .
    .
    .
}
```

(c) Explain the difference between a runtime exception and a checked exception? Give an example of each type.

Answers to Paper 4 Part B

[5 Marks]

A runtime exception does not need to be caught.

These exceptions occur through faults of logic, not due to unforeseen errors.

For example, accessing an array beyond its last element.

A checked exception is spotted by the compiler

which insists that the programmer include try/catch blocks.

(d) The methods of `PrintStream` are not recommended for internet applications; one reason for this is that `PrintStream` methods do not throw exceptions. However a class `PrintWriter` implements all the print methods of `printStream` and does throw exceptions. An excerpt from the Java API is reproduced below:

```
------------------------------------------------------------------------
java.io.PrintWriter

public PrintWriter(OutputStream out, boolean autoFlush)
   Create a new PrintWriter from an existing OutputStream.
   This convenience constructor creates the necessary intermediate
   OutputStreamWriter, which will convert characters into bytes
   using the default character encoding.

Parameters:
   out - An output stream
   autoFlush - A boolean; if true, the println() methods will flush
   the output buffer

public void println(String x)
   Print a String and then terminate the line.
   This method behaves as though it invokes print(String) and then println().
------------------------------------------------------------------------
```

Modify the common output statement

`System.out.println(String str);`

so that printing to the standard output is managed by a `PrintWriter`.

[5 Marks]

```
PrintWriter writer = new PrintWriter(System.out, true);
try {
    writer.println("Me too");
} catch (IOException e) {
}
```

Answers to Paper 4 Part B

QUESTION 5

(a) Write a program `WordSaver.java` that can download the HTML source of a website into an object of type `String`. The program should be invoked by the command

`java WordSaver http://www.gold.ac.uk`

and should handle any errors. Your attention is drawn to the list of class outlines in the Appendix of this paper.

[10 Marks]

```
class WordSaver {

    public static void main(String args[]) {

        if (args.length > 0) {
            StringBuffer buff = new StringBuffer();
            try {

                URL u = new URL(args[0]);
                Reader r = new BufferedReader(new InputStreamReader(u
                        .openStream()));
                int c = 0;
                while ((c = r.read()) != -1) {
                    buff.append((char) c);
                }

                String source = buff.toString();
            }

            catch (MalformedURLException e) {
                System.err.println(args[0] + " is not a parseable URL");
            } catch (IOException e) {
                System.err.println(e);
            }

        }
   }
}
```

(b) Alter your class `WordSaver` so that all the *words* from the HTML source are added one by one to an `ArrayList<String>`. A word is defined for the purposes of this program as a sequence of letters, where a letter is determined by the method `Character.isLetter(char c)`. Words are separated by any character that is not a letter.

[5 Marks]

Many answers are possible, for example:

Answers to Paper 4 Part B

```
String word = "";
ArrayList<String> list = new ArrayList<String>();
boolean inWord = false;
int c = 0;
while ((c = r.read()) != -1) {

    if (Character.isLetter((char) c)) {
        inWord = true;
        word += (char) c;
    }

    else {
        if (inWord) {
            list.add(word);
            word = "";
        }
        inWord = false;
    }

}
```

(c) (i) Add code to sort the list of words into alphabetical order and print to the command line.

(ii) The list will probably contain numerous identical words, for example "a" and "Goldsmiths". What type of collection will prevent the addition of duplicates?

(iii) Show how to instantiate such a collection from an `ArrayList<String>` of words.

[5 Marks]

(i)
```
Collections.sort(list);
System.out.println(list);
```

(ii) A Set will not allow duplicates

(iii) (2 marks)
```
Set<String> set = new TreeSet<String>(list);
```

(d) Can you think of a practical application for the program you have developed in this question? Explain your idea in a few lines.

[5 Marks]

A rudimentary search engine . . . associate words with sites . . . find all sites with a target word. Feed the set of words from WordSaver into a hashmap where the key is a word and the value is a list of webpages which contain that word. Use a webspider to find new sites, then employ WordSaver.

Other answers are possible - award 5 marks for a sensible idea and explanation.

Answers to Paper 4 Part B

QUESTION 6

(a) Show, in a few lines of code, how a client can make a TCP connection with a server.

[5 Marks]

```
String host = "localhost";
int port = 5000;
 try {
    Socket socket = new Socket(host, port);
   .
   .
   .
   } catch (IOException e) {
   System.out.println("\nCould not connect to " + server + " on port "
     + port + "\n");
}
```

(b) Extend your answer to (a) above by showing how a client can obtain input and output streams from/to the server. What kind of streams are these?

[5 Marks]

The client can get connection (low-level) streams
by calling `socket.getInputStream()`
and `socket.getOutputStream()`
these methods throw `IOException`'s
which must be caught.

(c) Extend your answer to (b) above by showing how would a client could receive an object that has been serialized and despatched by a server.

[5 Marks]

Use an ObjectInputStream
and chain it to a connection stream
then use `readObject()`

```
  try {
            ObjectInputStream ois = new ObjectInputStream(socket
                    .getInputStream());
            Object obj = ois.readObject();
            ois.close();
   } catch (IOException e) {
}
```

Answers to Paper 4 Part B

(d) Write code for an object client. The client should contact an object server at 192.168.0.1 and on port 4321, and receive and display the returned object, using the object's `toString()` method.

[10 Marks]

```
/*
 * This client makes a connection to local host on prt 7005, prints the response
 * from the server and closes
 */
package ipj;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.net.Socket;

public class SimpleObjectClient {

    static String host = "localhost";

    static int port = 4321;

    public static void main(String[] args) {

        String server = "localhost";

        try {
            Socket socket = new Socket(host, port);
            ObjectInputStream ois = new ObjectInputStream(socket
                    .getInputStream());

            Object obj = ois.readObject();
            System.out.println(obj);

            ois.close();

        } catch (IOException e) {
            System.out.println("\nCould not connect to " + server + " on port "
                    + port + "\n");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Notes

Comment form

We welcome any comments you may have on the materials which are sent to you as part of your study pack. Such feedback from students helps us in our effort to improve the materials produced for the External System.

If you have any comments about this guide, either general or specific (including corrections, non-availability of essential texts, etc.), please take the time to complete and return this form.

Title of this subject guide: .....

Name .....

Address .....

.....

Email .....

Student number .....

For which qualification are you studying? .....

Comments

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Please continue on additional sheets if necessary.

Date: .....

Please send your comments on this form (or a photocopy of it) to:
Publishing Manager, External System, University of London, Stewart House, 32 Russell Square, London WC1B 5DN, UK.