UNIVERSITY
OF LONDON

Introduction to Java and object-oriented programming Volume 2

S. Danicic

CO1109

2007

Undergraduate study in
Computing and related programmes

Goldsmiths
UNIVERSITY OF LONDON

This guide was prepared for the University of London by:

S. Danicic

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

london.ac.uk

Published by: University of London

© University of London 2007

Contents

Chapter 1

Introduction

This is the second volume of an introductory programming course in Java. It is assumed that the reader has first studied Volume 1 [Dan07]. For a full explanation of how to study this course, the reader is reminded to refer to Chapter 1 of Volume 1.

1.1 What We Cover in this Subject Guide

In this volume, we cover more advanced, but essential topics in Object Oriented Programming. These include:

- Command-line arguments
- Recursion
- Packaging programs
- More about variables
- Bits, types, characters and type casting
- Files and streams
- Sorting arrays and searching
- Defining your own classes
- Inheritance
- Exception handling
- `Vectors`

1.1.1 Suggested Schedule for Volume 2

This schedule is an approximate indication of how much time to spend on each chapter. It assumes that all the material is to be covered in ten weeks. This is a minimum. If you have a longer period of study you can adjust this proportionally.

Week 1: Chapters 2 and 3

Week 2: Chapter 4

Week 3: Chapters 5 and 6

Week 4: Chapter 7

Week 5: Chapter 8

Week 6: Chapter 9

Week 7: Chapter 9

Week 8:	Chapter 10
Week 9:	Chapter 11
Week 10:	Chapter 12

All the example programs given in the text, exercises and solutions, and other useful information will be provided on the accompanying CD and on the course website.

Details on how to access this website will be posted on

http://www.londonexternal.ac.uk/current_students/programme_resources/index.shtml

1.2 Books

I refer to a number of books throughout the text, specifically at the beginning of each chapter. Details of these books can be found in the bibliography in on the last page of this volume (page 189).

Chapter 2

Command-Line Arguments

2.1 Learning Objectives

Chapter 2 explains:

- the purpose of command-line arguments
- how to use command-line arguments.

2.2 Reading

- [CK06] pages 308-310

2.3 Introduction

Now, at last, you will learn the purpose of the line

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

The name, `args`, is a formal parameter to the `main` method. The parameter `args` is of type array of `String`. The use of `args` is straightforward: if we run our program `fred.class` normally we type `java fred`, but if we like, we can write anything else we like after `java fred`, for example:

1. `java fred hello`

   or

2. `java fred 1 2 3 4`

   or

3. `java fred fred.java`

The `Strings` that we type after `java fred` are called *command line arguments*. The command line arguments are passed to the program in the `String` array parameter of the `main` method, namely, `args`. (We could have called `args` anything we liked. Let's stick to `args` though.)
In 1, above, there is one command line argument `args[0]` and its value is the `String` `"hello"`.
In 2, there are four command line arguments:

name	value
`args[0]`	”0”
`args[1]`	”1”
`args[2]`	”2”
`args[3]`	”3”

Note: `"0"`,`"1"`,`"2"`, and `"3"` are all of type `String`. If we want to add them up we first have to convert them to `int`s using the method `Integer.parseInt()`.
In 3, there is one command line argument `args[0]` and its value is the `String` `"fred.java"`.

2.4 The Number of Command Line Arguments

Since `args` is an array, the number of command line arguments is simply given by the expression `args.length`. (Recall that, if `a` is an array then `a.length` is the number of elements of `a`.)

2.5 Exercises on Chapter 2

2.5.1 Add One

Write a program which prints out one more than its command line argument. For example, `java AddOne 5` should output 6, etc. (See page 159 Number 1 for the answer.)

2.5.2 Add

Write a program which adds its two command line arguments i.e. `java Add 5 6` should output 11. (See page 159 Number 2 for the answer.)

2.5.3 Backwards

Write a program that allows you to enter as many words as you like as command line arguments and the program prints them out in reverse order. (See page 159 Number 3 for the answer.)

2.5.4 Add All

Write a program that allows you to enter as many integers as you like as command line arguments and the program prints out their sum. (See page 159 Number 4 for the answer.)

2.5.5 Add All Real

Write a program that allows you to enter as many real numbers as you like as command line arguments and the program prints out their sum. (See page 159 Number 5 for the answer.)

2.5.6 Exercises (no solutions)

1. Write a program that prints out the average of all its command line arguments.
2. Write a program that prints out all the longest of all its command line arguments.
3. Write a program that prints all its command line arguments backwards. So `java back Fred Bloggs` should output `derf sggolB`

2.6 Summary

Having worked on Chapter 2 you will have:

- Understood the purpose of command-line arguments.
- Learned how to use command-line arguments.

Chapter 3

Recursion

3.1 Learning Objectives

Chapter 3 explains:

- recursion
- how to define and use recursive methods.

3.2 Reading

- [Dow03] Chapter 4

3.3 Definition of a Recursive Method

A recursive method is one that calls itself.

3.4 Examples of Recursive Methods

3.4.1 Factorial

```
[Lecture16/FactorialNew.java]

public class FactorialNew
{
    static    int fact(int n)
    {
        if (n==0) return 1;
        else return n*fact(n-1);
    }
    public static void main(String[] args)
    {
        System.out.print(fact(Integer.parseInt(args[0])));
    }
}
```

3.4.2 Greatest Common Divisor

The greatest common divisor of two integers is the largest integer that divides them both exactly. For example gcd(8,12)=4, gcd(7,13)=1 and gcd(16,96)=16.

An algorithm for finding the GCD of two positive integers n and m is as follows:

1. if $n = m$ then return n
2. if $n > m$ then subtract m from n and go to 1
3. if $m > n$ then subtract n from m and go to 1

[Lecture16/gcdNew.java]

```
public class gcdNew
{
    static    int gcd(int n,int m)
    {
        if (n==m) return n;
        else if (n>m)
                     return gcd(n-m,m);
                     else   return gcd(n,m-n);
    }
    public static void main(String[] args)
    {
        System.out.print(gcd(Integer.parseInt(args[0]),
                                      Integer.parseInt(args[1]))
                                 );
    }
}
```

3.5 Exercises on Chapter 3

3.5.1 Fibonacci Numbers

Write a program such that `java fibonacci n` prints out the nth Fibonacci number. The Fibonacci sequence goes 1,1,2,3,5,8,13,21,34,55,89,$\cdots$. (See page 160 Number 6 for the answer.)

3.5.2 Multiplication

Multiplication of non-negative integers can be defined recursively in terms of addition:

$$\begin{aligned} \text{mult}(n,0) &= 0 \\ \text{mult}(n,m+1) &= \text{n+mult}(n,m) \end{aligned}$$

Write a class which has a method `mult` which implements such a function. (See page 160 Number 7 for the answer.)

3.5.3 Exponentiation

Exponentiation of non-negative integers can be defined recursively in terms of Multiplication:

$$\begin{array}{lcl} n^0 & = & 1 \\ (n^{m+1}) & = & \text{n*}(n^m) \end{array}$$

Write a class which has a method `power` which implements such a function. (See page 160 Number 8 for the answer.)

3.5.4 Reversing Input

Without using `Vectors` or arrays write a program which reads in characters from the keyboard and prints them out in the opposite order to which they were typed in. The program should stop when the user presses the *enter* key. (See page 161 Number 9 for the answer.)

3.5.5 The Syracuse Sequence

The *Syracuse Sequence* starting with 14 goes like this:
14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
The rule is as follows: if n is even then the next number in the sequence is $n/2$ and if n is odd the next number is $3n + 1$.
The sequence stops at 1.
Using recursion, write a program not containing the word "`while`", such that for all positive integers, n, `java syr n` prints out the Syracuse sequence starting with n.
(Try running `syr.class` to see how your program should behave.) An interesting fact about the Syracuse sequence is that nobody knows whether it always ends with 1 or not! No-one has proved it and no one has found an example that does not end in 1. (See page 161 Number 10 for the answer.)

3.6 Summary

Having worked on Chapter 3 you will have:

- Understood recursion.
- Learned how to define and use recursive methods.

Chapter 4

Packaging Programs

4.1 Learning Objectives

Chapter 4 explains:

- the purpose of the `CLASSPATH` system variable
- the purpose of the `package` statement
- the purpose of the `import` statement
- how to run a Java program that is part of a package, from the command line.

4.2 Reading

- [CK06] Chapter 13
- [NK05] page 66
- [Kan97] page 33
- [Smi99] Chapter 17

4.3 Introduction

If you have some useful programs that you wish to use over and over again it is useful to package them up so you can refer to them from anywhere (i.e., make them public). There is a statement in Java that comes at the beginning of your program called a `Package Statement`. This is used in conjunction with a system variable called `CLASSPATH`[1] used by your Java System in conjunction with your operating system to tell Java where to look for `.class` files.

[1] See Volume 1 for instructions on setting the CLASSPATH.

Consider [`LecturePackages/egg.java`]

```
class dog
{
     public static void main (String [] args)
     {
       System.out.println(" hello");
       food.veg.radish.me();
     }
}
```

Java will expect to find a directory called `food` at 'the top level' of a directory in `CLASSPATH`.

The directory `food` will contain a directory called `veg`. The directory `veg` will contain a file `radish.class` and `radish.class` will have a static method called `me()`.

```
                   <somewhere in CLASSPATH>
                           |
----------------------------------------------------
|              |          |         |              |
              food
               |
------------------------------------
|        |          |               |
                   veg
                    |
     ----------------------------
    |       |                   |
         radish.class
```

The java code for `radish.class` could be [`LecturePackages/food/veg/radish.java`]

```
package food.veg;
public class radish
{
        public static void me()
        {
          System.out.println(" radish");
        }
}
```

Notice the `package` statement.

4.4 `Public` Classes

We can also now explain the need for the `public` keyword for the method `me` of `radish`. If we remove it and recompile LecturePackages/egg.java we get

```
egg.java:6: No method matching me() found in class food.veg.radish.
       food.veg.radish.me();
                         ^
1 error
```

If you want to be able to call a method from outside the package where it is declared you must declare it as `public`.

4.5 File Names and Public Classes

If your `.java` file contains a public class `fred` then the file must be called `fred.java`. You can therefore see that each java file can only define one public class. It can however define as many non-public classes as you like.

We may wish to put the class dog in a package. Since dog is in a directory called LecturePackages we must call the package LecturePackages as well. [LecturePackages/dog.java]

```
package LecturePackages;
class dog
{
     public static void main (String [] args)
     {
       System.out.println(" hello1");
       food.veg.radish.me();
     }
}
```

4.5.1 Running Programs that are Part of Packages

To run our program we must type

```
java LecturePackages.dog
```

4.6 The import Statement

Now consider [LecturePackages/pig.java]

```
import food.veg.*;
public class pig
{
    public static void main (String [] args)
   {
     System.out.println(" hello");
     radish.me();
   }
}
```

Because of the import statement we do not have to refer to the method me() by its full name food.veg.radish.me() but simply as radish.me().

We have been using the import statement throughout this course. Here is the simple Echo program without an import statement. [Lecture2/EchoNoImportNew.java]

```
class EchoNoImportNew
{
    public static void main(String[] args)
    {
      java.util.Scanner in =new java.util.Scanner(System.in);
      String s =in.nextLine();
      System.out.println(s);
    }
}
```

Notice that we have had to refer to `Scanner` by its full name since it is a class defined in the `java.util` package.

4.7 Laborious but Worthwhile Packaging Task

You may decide to have a new directory `javacourse` to put all the useful programs you have used on this course. Package them up nicely together. The simplest way to do this would be to keep the same directory structure as this course with the different Lectures.

```
                    <your home directory>
                           |
-----------------------------------------------------------
|             |              |          |                 |
                           javacourse
                               |
----------------------------------------------------................
|              |                   |
Lecture1      Lecture2    ...    LecturePackages
    |                                 |
  ---------              ------------------------------------
   |       |             |        |          |              |
      HelloWorld.java                                      veg
                                                            |
                                             -----------------------------
                                             |       |                   |
                                               radish.class
```

and then add the full pathname `/home/you/javacourse` to your `CLASSPATH`. In Unix you would say

```
CLASSPATH=$HOME/javacourse:$CLASSPATH
export CLASSPATH
```

and add a package statement to every program like this [`Lecture1/HHH.java`]

```
//Our First Program
package Lecture1;
public class HHH
{
        public static void main(String[] args)
        {
          System.out.println("Henry");
        }
}
```

You can then run HHH.java wherever you are by typing

```
java Lecture1.HHH
```

provided your CLASSPATH is correctly set. If this seems an onerous task, then simply package up the useful array methods like this: Please see [Dan07], Chapter 1 for more information on the CLASSPATH system variable. [LectureNonVoidMethods/ArraysNew.java]

```
package LectureNonVoidMethods;
import java.util.Scanner;
public class ArraysNew
{
    public static int sum(int [] a)
    {
       int sum=0;
       for (int i=0;i<a.length;i++)sum=sum+a[i];
       return sum;
    }
    public static int largest(int [] a)
    {
       int largest=a[0];
       for (int i=1;i<a.length;i++) if (a[i]>largest) largest=a[i];
       return largest;
    }
    public static int smallest(int [] a)
    {
       int smallest=a[0];
       for (int i=1;i<a.length;i++) if (a[i]<smallest) smallest=a[i];
       return smallest;
    }
    public static double average(int [] a)
    {
      return (sum(a)*1.0)/a.length;
    }
    public static void print(int a[])
    {
      for (int i=0;i<a.length;i++) System.out.println(a[i]);
    }
    public static int [] readintarray()
    {
       Scanner in = new Scanner(System.in);
       System.out.println("How many numbers are you going to enter? ");
       int n = in.nextInt();
       System.out.println("Enter "+n+ " numbers ");
       int[] a; //declares a to be an array of ints
       a =new int[n];  //gives a enough space to hold n ints
       for(int i=0;i<n;i++)  /*read them into an array*/
       {
         a[i]=in.nextInt();
       }
        return a;
     }
}
```

We can now refer to these methods in other programs. Make a new directory and use LectureNonVoidMethods/Arrays.java to write a program which reads some numbers into an array and prints the average. [LectureTest/Test1.java]

```
class Test1
{
 public static void main(String [ ] args)
 {
  int [ ] a;
  a = LectureNonVoidMethods.ArraysNew.readintarray();
  System.out.println("Their average is " + LectureNonVoidMethods.ArraysNew.average(a));
 }
}
```

Or if we use an `import` statement: [`LectureTest/Test2.java`]

```
import LectureNonVoidMethods.*;
class Test2
{
 public static void main(String [ ] args)
 {
  int [ ] a;
  a = ArraysNew.readintarray();
  System.out.println("Their average is " + ArraysNew.average(a));
 }
}
```

For some reason, if you want to test a program that is part of a package and your test program is in the same subdirectory as the package, then you must include a package statement at the beginning of your test program as in [`LectureNonVoidMethods/Test2.java`]

```
package LectureNonVoidMethods;
class Test2
{
 public static void main(String [ ] args)
 {
  int [ ] a;
  a = ArraysNew.readintarray();
  System.out.println("Their average is " + ArraysNew.average(a));
 }
}
```

So you make the test program also part of the package. This is not a problem. Also notice that you do not need to import from the same package that you are currently in. So we do not need

```
import LectureNonVoidMethods.*
```

Java will automatically look in the current package for unresolved class and method names.

4.8 Exercises on Chapter 4

4.8.1 Add Comments

Add your own comments explaining each of the methods in the class `ArraysNew` defined in Section 4.7, page 14.

4.8.2 No Import

Consider the following Java program: `[Lecture2/EchoNew.java]`

```
import java.util.Scanner;
class EchoNew
{
    public static void main(String[] args)
    {
      Scanner in =new Scanner(System.in);
      String s =in.nextLine();
      System.out.println(s);
    }
}
```

How would it have to be rewritten if there was no `import` statement in Java? (See page 161 Number 11 for the answer.)

4.8.3 A Complete Application

See the class `ArraysNew` in Section 4.7, page 14. The methods in class `ArraysNew` can be referred to in other classes. Write a complete Java application (by calling some of the methods in the class `ArraysNew`) which asks the user how many numbers they are going to enter, reads in the numbers and prints their average. (See page 161 Number 12 for the answer.)

4.8.4 Add your Own Method

Add your own method to the class `ArraysNew` in Section 4.7, page 14, which multiplies the elements of an array together. Write a complete application that uses this method to compute factorial.

4.9 Summary

Having worked on Chapter 4 you will have:

- Understood the purpose of the CLASSPATH system variable.
- Understood the purpose of the `package` statement.
- Learned the purpose of the `import` statement.
- Learned how to run a Java program that is part of a package, from the command line.

Chapter 5

More About Variables

5.1 Learning Objectives

Chapter 5 explains:

- about local variables and the scope of variables
- about simple variables
- about reference variables
- more about how the values of these variables are passed as parameters to methods.

5.2 Reading

- [Fla05] Page 150
- [NK05] Page 132
- [Hub04] 5.2
- [Kan97] Q1.10, Q2.12

5.3 Introduction

5.3.1 Local Variables

A local variable is one that is declared inside a method or inside another structure like a `for` loop. Consider the program [`LectureScope/p1.java`]

```
class p1
{
    static void f()
    {
      int x=3;
    }

     public static void main(String [ ] args)
    { int x;
      x=2;
      f();
      System.out.println(x);
    }
}
```

What is its output? (See page 162 Number 13 for the answer.)

The x declared inside the method f() is called a *local variable*.

The program [LectureScope/p3.java]

```
class p3
{

    static int x;
    static void f()
    {
      int x=3;
    }

     public static void main(String [ ] args)
    {
      x=2;
      f();
      System.out.println(x);
    }
}
```

also prints 2 because the assignment in the method f() is to a local variable x and not the global one that gets printed in main().

Now consider [LectureScope/p2.java]

```
class p2
{
    static int x;
    static void f()
    {
       x=3;
    }

     public static void main(String [ ] args)
    {
      x=2;
      f();
      System.out.println(x);
    }
}
```

The output of this is 3 because both main() and p reference the same global variable x.

A common error is illustrated in [LectureScope/ForLocal.java]

```
class ForLocal
{
    public static void main(String [ ] args)
    {
      int sum=0;
      for (int i=1;i<10;i++) sum=sum+i;
      System.out.println(i);
    }
}
```

The compiler will complain with

```
ForLocal.java:8: cannot resolve symbol
symbol  : variable i
location: class ForLocal
      System.out.println(i);
                         ^
1 error
```

The variable `i` is only in scope inside the body of the loop.

5.4 What's Really in a Variable?

There are two kinds of variable in Java:

- Simple variables
- Reference variables

Simple variables are those whose type is simple, like `int` or `char` etc.

All other variables are reference variables (for example `String` variables or array variables).

When you make an assignment to a simple variable, the actual value is stored in the variable. For example, the assignment `int x=79;` puts the value 79 into the variable `x`.

When you make an assignment to a reference variable, what is assigned is the address of the object.

For example `String s= "cat"` will not put the value `"cat"` in the variable `s` but the `String` `"cat"` will be stored somewhere in memory and `s` will contain the address of where `"cat"` has been stored. So `String s ="fred";String t=s; System.out.println(t);` will print `fred` and `String s ="fred";String t=s; s="mary"; System.out.println(t);` will print `fred`. In the second example `String t=s;` makes variable `t` 'point at' the same as what variable `s` points at. The assignment, `s="mary"` makes `s` point at something different, but `t` still points at what `s` pointed at before.

Similarly the assignment `a [ ] = new in[50]` creates an array of 50 `int`s somewhere in RAM and the variable `a` will contain the address of this array.

5.5 Exercises

What is the output of [`LectureRefVars/test1.java`]

```
class test1
{
    public static void main(String[ ] args)
    {
        int[ ] a = new int[1];
        a[0]=5;
        int [ ] b =a;
        a[0]=6;
        System.out.println(b[0]);
    }
}
```

(See page 162 Number 14 for the answer.)

What is the output of [LectureRefVars/test2.java]

```
class test2
{
    public static void main(String[ ] args)
    {
        int  a;
        a=5;
        int  b=a;
        a=6;
        System.out.println(b);
    }
}
```

(See page 162 Number 15 for the answer.)

5.6 Parameters Passed by Value

When parameters are passed in a method call, it is as if the method has some local variables whose names are the same as the formal parameters of the method. Before executing the body of the method these local variables are assigned the values of the corresponding actual parameters.

Consider a [Lecture9/intParams.java]

```
class intParams
{
 public static void main(String [ ] args)
 {
   int n=3;
   p(n);
   System.out.println(n);//This prints 3
 }
 static void p(int m)
 {
   m=m+1;
 }
}
```

The output of this is 3. This is because when we call `p(n)` a copy of the value of `n` is made before we execute `p`. The variable `n` is not changed.

5.7 Exercises on Chapter 5

Consider the following programs and see if you can explain their behaviour.

5.7.1 Arrays (1)

```
[Lecture9/arrayParams.java]

class arrayParams
{
 public static void main(String [ ] args)
 {
   int[]a=new int[1];
   a[0]=1;
   p(a);
   System.out.println(a[0]);//This prints 5
 }
 static void p(int [ ]  m)
 {
   m[0]=5;
 }
}
```

(See page 162 Number 16 for the answer.)

5.7.2 Arrays (2)

```
[Lecture9/arrayParams2.java]

class arrayParams2
{
 public static void main(String [ ] args)
 {
   int[]a= new int[1];
   a[0]=5;
   p(a);
   System.out.println(a[0]);//This prints 5
 }
 static void p(int [ ]  m)
 {
   m= new int[3];
   m[0]=7;
 }
}
```

(See page 162 Number 17 for the answer.)

5.7.3 Strings

```
[Lecture9/StringParams.java]

class StringParams
{
 public static void main(String [ ] args)
 {
   String n="hello";
   p(n);
   System.out.println(n);//This prints hello
 }
 static void p(String m)
 {
   m="goodbye";
 }
}
```

(See page 162 Number 18 for the answer.)

5.7.4 Test Array

```
[Lecture8/TestArray.java]

class TestArray
{
        static void change(int [ ] a)
        {
          a[0]=17;
        }
        public static void main (String [] args)
        {
          int [ ] z = new int [2];
          z[0]=1;z[1]=2;
          change(z);
          Arrays.printarray(z,2);
        }
}
```

(See page 162 Number 19 for the answer.)

5.7.5 Test Int

```
[Lecture8/TestInt.java]

class TestInt
{
        static void change(int a)
        {
          a=17;
        }
        public static void main (String [] args)
        {
          int z=5;
          change(z);
          System.out.println(z);
        }
}
```

(See page 162 Number 20 for the answer.)

5.8 Summary

Having worked on Chapter 5 you will have:

- Learned about local variables and the scope of variables.
- Learned about simple variables.
- Learned about reference variables.
- Learned more about how the values of these variables are passed as parameters to methods.

Chapter 6

Bits, Types, Characters and Type Casting

6.1 Learning Outcomes

Chapter 6 explains:

- that variables of different types have different sizes
- about type casting
- about characters and Unicode
- why the `read` method returns an `int`.

6.2 Reading

- [LO02] Chapter 1
- [Fla05] pages 21-28
- [Kan97] Chapter 2
- [Smi99] Chapter 4

6.3 Introduction

The memory of a computer (Random Access Memory or RAM for short) is made up of a sequence of consecutive bytes. These days (2007) a typical personal computer has about 512 megabytes of RAM. This is roughly 512 million bytes, each consisting of 8 bits. A bit can have only two possible values: 0 or 1. Every time you declare a variable in your program it is temporarily using up some RAM. If you declare a large array it will take up lots of RAM. Try running
`[LectureChar/BigMemory.java]`

```
public class BigMemory
{
     public static void main(String [ ] args)
        {
           int a[] = new int[100000000];
        }
}
```

On my laptop, I get the following error message:

```
Exception in thread "main" java.lang.OutOfMemoryError:
    at BigMemory.main(BigMemory.java:7)
```

6.3.1 Exercise: Maximum Array Size

Experiment to find out what the biggest `int` array you can have is. On my system it's at least 15,000,000.

6.4 Different Types Have Different Sizes

A variable consists of some bytes of RAM. When you declare a variable of a particular type, some bytes of RAM are allocated to hold that variable. Depending on what the type is, different amounts of RAM are allocated. A `boolean` variable, for example, needs only one bit since it has only two possible values: `true` and `false`.

6.5 Characters

Java allows two bytes (=16 bits) to store characters. This means that in Java we can represent 2^{16} different characters.

6.5.1 Exercise

Write a Java program to work out 2^{16}. (See page 162 Number 21 for the answer.)

So Java can represent 65536 different characters. This is because Java tries to be international, so you will be able to write programs to output characters in your own alphabet wherever you live (provided, of course, your operating system is correctly set up to do so).

Unfortunately, on my computer I can only print $256 = 2^8$ different characters. These are the ones whose Unicode value is between zero and 255. For any other Unicode values my computer simply prints out a question mark.

6.5.2 Exercise: Find All Characters

Write a program that prints out the character corresponding to every Unicode value between 0 and $2^{16} - 1$. (See page 162 Number 22 for the answer.)

On my computer, only the first 256 give anything. After that, I just get `'?'`. How is it on your computer?

6.6 Type Casting

To display the character whose Unicode value is 37 we do [`LectureChar/Unicode.java`]

```
public class Unicode
{
    public static void main(String [ ] args)
        {
            System.out.println((char)37);
        }
}
```

(char)37 means the character whose Unicode value is 37. This is called *type casting* or simply *casting*. To do type casting you write a type in brackets followed by an expression. The reason you need type casting is that an expression on its own could be of many different types and Java will treat it differently depending on what type it is. Therefore we have to tell the Java system which type we want the expression to be treated as. Consider [LectureChar/AplusB.java]

```
public class AplusB
{
    public static void main(String [ ] args)
        {
         System.out.println('a'+'b');
         System.out.println((char)('a'+'b'));
        }
}
```

On my computer, this prints out
195
ã

The reason for this is as follows: 'a' + 'b' is interpreted as the Unicode value of the character 'a' + the Unicode value of the character 'b'. These two are added together to give the int 195. Which when we print gives 195. (char)('a' + 'b') will be interpreted as the character whose Unicode value is 195. Apparently, this character is ã.

6.6.1 Quick Question

The Unicode value of 'b' is one more than that of 'a'. What are the Unicode values of a and b respectively? (See page 162 Number 23 for the answer.)

6.7 The Method read()

In this section we discuss the read() method. This method is used for inputting a single character. Consider[LectureChar/Try.java]

```
import java.io.*;
class Try
{
    public static void main(String[] args) throws IOException
        {
           System.out.println("please type someting in ending with 'enter'");
           int c=System.in.read();
           System.out.println("You typed in " +(char)c);
        }
}
```

Here we are inputting just a single character using `read()`, instead of the previous `readLine()` which reads a whole line. The method `read()` returns the Unicode value of the character that the user typed in. So if you want to print out the character the user typed in, the output of `read()` must be cast to `char` otherwise the Unicode value will be written out as in `[LectureChar/Try1.java]`

```
import java.io.*;
class Try1
{
    public static void main(String[] args) throws IOException
        {
           System.out.println("please type something in ending with ’enter’");
           int c=System.in.read();
           System.out.println("You typed in " + c);
        }
}
```

6.7.1 End of File

If you are using Unix, try running LectureChar/Try1.java and typing in CTRL d (hold the control key down and press `d` at the same time). This stands for end of file. The program prints:

```
You typed in -1
```

This means that `read()` returns -1 when end of file is reached. (If you didn’t understand this bit, don’t worry. It will become clear in the chapter on Files.) This is why `read()` returns an `int` rather than a `char`.

6.7.2 A Reason why `read()` Returns an `int`

A variable of type `int` occupies 32 bits. All the 2^{16} values of type `char` are reserved for storing real characters. No special value of type `char` is used to represent special things like representing end of file. We therefore need a bigger type to be the return type of `read()`.

6.8 More Type Casting

6.8.1 Question

Consider `[LectureChar/EOF.java]`

```
import java.io.*;
class EOF
{
    public static void main(String[] args) throws IOException
        {
           System.out.println((int)((char)(-1)));
        }
}
```

What do you think is its output? (See page 163 Number 24 for the answer.)

Now consider [`LectureChar/Try3.java`]

```
import java.io.*;
import java.util.*;
class Try3
{
    public static void main(String[] args) throws IOException
        {
           Scanner in =new Scanner((System.in));
           System.out.println("please type something in ending with 'enter'");
           int c= in.nextInt();
           System.out.println("You typed in " + (int)((char)c));
        }
}
```

If you input integers between 0 and 65535 inclusive you get the same answer back. If you enter 65536 you get 0, if you enter 65537 you get 1, etc. This is because the only whole numbers that can be stored as type `char` lie between 0 and 65535. If we try to cast a number not in this range as a `char`, it will be ‘squashed’ into this range simply by calculating its value mod 65536. This explains why entering -1 gives us back the value 65535, since -1 mod 65536 is 65535.

Another way to think of this is that four bytes are used to store an `int` and only two are used to store a char. When a `int` is cast to a `char` the two most significant bytes are discarded. So this means that the two least significant bytes in the `int` representation of -1 are both 11111111. In fact -1 is represented as the four bytes:
11111111 11111111 11111111 11111111. If you cast a value to a smaller type you may lose information.

6.9 Exercises on Chapter 6

6.9.1 Research

Read pages 23 band 24 of [Fla05] to find out how big each type is.

6.9.2 `Int` to `Boolean`

Write a program to check whether or not an `int` can be cast as a `boolean`. (See page 163 Number 25 for the answer.)

6.9.3 `Boolean` to `Int`

Write a program to check whether or not a `boolean` can be cast as an `int`. (See page 163 Number 26 for the answer.)

6.9.4 `Float` to `Int`

Write a program to check whether or not a `float` can be cast as an `int`. (See page 163 Number 27 for the answer.)

6.9.5 `Int` to `Float`

Write a program to check whether or not an `int` can be cast as a `float`. (See page 163 Number 28 for the answer.)

6.9.6 `Double` to `Float`

Write a program to check whether or not a `double` can be cast as a `float`. (See page 164 Number 29 for the answer.)

6.9.7 `Float` to `Char`

Write a program to check whether or not a `float` can be cast as a `char`. (See page 164 Number 30 for the answer.)

6.9.8 `Int` to `Short`

Write a program to check whether or not an `int` can be cast as a `short`. (See page 164 Number 31 for the answer.)

6.9.9 Next Biggest

What is the next biggest `int`, n, after 0, such that `System.out.println(`n`)` gives 0. (See page 164 Number 32 for the answer.)

6.9.10 What is the Output?

Why is the output of [`LectureChar/IntToShort1.java`]

```
public class IntToShort1
{
    public static void main(String [ ] args)
        {
          int i = 65536;
          for(int j=0;j<10;j++)
          System.out.println((short)(j*i));
        }
}
```

```
0
0
0
0
0
0
0
0
0
0
```

See Question 6.9.9 for the answer.

6.9.11 Largest `Int`

What does the following program output? [`LectureChar/LargestInt.java`]

```
public class LargestInt
{
    public static void main(String [ ] args)
        {
          int i = 2147483647;
          System.out.println(i);
          System.out.println(i+1);
          }
}
```

(See page 164 Number 33 for the answer.)

6.10 Summary

Having worked on Chapter 6 you will have:

- Learned that variables of different types have different sizes.
- Learned about type casting.
- Learned about characters and Unicode.
- Learned why the `read` method returns an `int`.

Chapter 7

Files and Streams

7.1 Learning Objectives

Chapter 7 explains:

- the purpose of a file
- how to read a file a character at a time
- how to read a file a line at a time
- how to write to a file
- how to write a simple spell-checking program, which checks that all words in a file have been spelled correctly.

7.2 Reading

- [CK06] Chapter 20
- [LO02] Chapter 14
- [DD07] Chapter 15

7.3 Introduction

A file is some data that is not in the memory of the computer. The most common place to find files is on a disc. The difference between data on a disc and data in memory is that disc data survives after you have finished executing the program. So if there is any information that needs to be kept as a result of running a program, you had better write it to a file. Similarly, many programs are required to manipulate data produced by other programs. This means that programs are required to read data from files stored on discs.

Java also treats user input as if the data input is coming from a file; similarly, user output is exactly the same as writing to a file.

7.4 Reading Files

Consider: `[Lecture11/cat1.java]`

```
import java.io.*;
import java.util.Scanner;
public class cat1 //This copies from file fff.dat to standard output (screen)
```

```
{
   public static void main(String[] args) throws Exception
   {
     Scanner in =new Scanner(new FileReader("fff.dat"));
     while (in.hasNextLine())
     {
       System.out.println(in.nextLine());
     }
   }
}
```

This program prints out the contents of a file called `fff.dat` on the screen. When I run it

```
my name
is sebastian.
I live in
London.
```

is displayed. This is because that is what is in the file called `fff.dat`.

Look up the class `FileReader` in
http://java.sun.com/j2se/1.5.0/docs/api/java/io/BufferedReader.html. and
http://java.sun.com/j2se/1.5.0/docs/api/java/io/FileReader.html respectively. Notice that we can use the `Scanner` with a FileReader as well as with `System.in`.

7.4.1 End Of File

If we are reading a file using `nextLine()` and we come to the end of the file, `hasNextLine()` will return the value `false`. Therefore the code:

```
while (in.hasNextLine())
     {
       System.out.println(in.nextLine());
     }
```

will repeatedly read a line from `in(fff.dat)` and write it out to the screen until the end of file is reached.

Notice the extra `throws Exception`. This will be explained in Chapter 11. See what happens if you leave it out.

7.5 Reading Files a Character at a Time

Consider `[Lecture11/cat79.java]`

```
import java.io.*;
class cat79
{
      public static void main(String [] args) throws Exception
      {
         FileReader inone = new FileReader("fff.dat");
         int t=inone.read();
         while (t!=-1)
         {
           System.out.print((char)t);
           t=inone.read();
         }
      }
}
```

Here we are reading from file `fff.dat` a single character at a time. As explained in Section 6.7.1, the method `read()` returns an `int`: either the *Unicode* value of the character just read in, or -1 if we have reached the end of file. We must therefore cast the value just read in as a character in order to print it out properly.

7.5.1 Question

If we forget to cast to `char`, as in [`Lecture11/cat79u.java`]

```
import java.io.*;
class cat79u
{
      public static void main(String [] args) throws Exception
      {
         FileReader inone = new FileReader("fff.dat");
         int t=inone.read();
         while (t!=-1)
         {
           System.out.print(t);
           t=inone.read();
         }
      }
}
```

what happens? (See page 164 Number 34 for the answer.)

7.5.2 The Newline Character

The end of a line in a file is achieved using a special character called newline, written `'\n'` in Java. so

```
println("hello");
```

is exactly the same as

```
print("hello");
print('\n');
```

7.6 Writing to Files

Once we have started off with

```
PrintStream out =new PrintStream(new FileOutputStream("hhh.dat"));
```

writing to the file `hhh.dat` is just like outputting to the screen. We can use the methods `out.print()` and `out.println()`.

7.6.1 Closing the File

There is one small difference between writing to a file and writing to the screen; when we have finished writing a file, we must close it with

```
out.close()
```

If you forget to do this it is quite possible that you will find that after you finish executing your program, your file will not have been written to. Here is a program to write to a file called "`hhh.dat`" [Lecture11/cat83.java]

```
import java.io.*;
class cat83
{
      public static void main(String [] args) throws Exception
      {
         PrintStream out =new PrintStream(new FileOutputStream("hhh.dat"));
         out.println("this");
         out.println("file was written by");
         out.println("a Java program");
         out.close();
      }
}
```

After we have run it we will find a new file in the current directory called `hhh.dat` with

```
this
file was written by
a Java program
```

written in it.

[Lecture11/cat.java]

```
import java.io.*; // this copies from anywhere we like to screen
public class cat
{
  public static void main(String[] args) throws Exception
  {
     FileReader inone =new FileReader(args[0]);
     int t=inone.read();
```

```
      while (t!=-1)
      {
        System.out.print((char)t);
        t=inone.read();
      }
    }
  }
```

This program copies the contents of any file to standard output (i.e., it displays any file on the screen). This program uses **command line arguments**. To display a file called `fred`, type

```
java cat fred
```

[Lecture11/copy.java]

```
import java.io.*;
public class copy // this copies from anywhere we like to anywhere we like
{
    public static void main(String[] args) throws Exception
    {
       FileReader inone = new FileReader(args[0]);
       PrintWriter outone =new PrintWriter(new FileOutputStream(args[1]));
       int t=inone.read();
       while (t!=-1)
       {
         outone.print((char)t);
         t=inone.read();
       }
       outone.close();
    }
}
```

This program copies the contents of any file to any other.

7.6.2 Swap all `a`s and `b`s

Write a program called `swapab.java` that is the same as cat.Java except all the ‘a’s and ‘b’s are swapped. (See page 164 Number 35 for the answer.)

7.6.3 Example: Counting the Number of Lines in a File

To count the number of lines we can either simply read the file a line at a time, as in [Lecture11/lineCount2.java]

```
import java.io.*;
import java.util.Scanner;
public class lineCount2
{
  public static void main(String[] args) throws Exception
  {
     int count=0;
     Scanner inone =new Scanner(new FileReader(args[0]));
     while (inone.hasNextLine())
     {
         String t=inone.nextLine();
         count++;
     }
     System.out.println(count + " lines");
  }
}
```

or alternatively, read the file a character at a time and keep a count of the number of newline characters that we encountered, as in [Lecture11/lineCount1.java]

```
import java.io.*;
public class lineCount1
{
  public static void main(String[] args) throws Exception
  {
     int count=0;
     FileReader inone = new FileReader(args[0]);
     int t=inone.read();
     while (t!=-1)
     {
       if (t=='\n') count++;
       t=inone.read();
     }
     System.out.println(count + " lines");
  }
}
```

7.6.4 Example: Counting the Number of Words in a File

To count the number of words in a file, we first need to decide what a *word* is. Let us say that a word is any consecutive string of characters, not containing a newline character '\n', a tab character '\t' or a space ' '. We need to read the file a character at a time and add one to a counter every time we go from *not being in a word* to *in a word*.

- We start off *not in a word*.
- When *not in a word*, we stay *not in a word* if the next character we read is a newline, space or tab.
- When *not in a word*, we go *in a word* if the next character we read is a not a newline, space or tab.
- When *in a word*, we stay *in a word* if the next character we read is not a newline, space or tab.
- When *in a word*, we go *not in a word* if the next character we read is a newline, space or tab.

[Lecture11/wordCount.java]

```
import java.io.*;
public class wordCount
{
  public static void main(String[] args) throws Exception
  {
     boolean inword=false;
     int count=0;
     FileReader inone = new FileReader(args[0]);
     int t=inone.read();
     while (t!=-1)
     {
       if (!inword)
          if (t=='\n' || t=='\t' || t==' ') ;//empty true part
          else {inword=true;count++;}
       else if (t=='\n' || t=='\t' || t==' ')inword=false;
       t=inone.read();
     }
     System.out.println(count + " words");
  }
}
```

7.7 Example: A Simple Spell Checker

There is a file `Lecture11/words` consisting of a big list of words in English (courtesy of Unix), one per line. We can use it to write a simple spell checker. We are using the method

```
java.lang.String.startsWith().
```

The user simply types the beginning of the word he wants to check as a command line argument. For example,

```
java spell freq
```

will print out all English words starting with `freq`. These are:

```
frequencies
frequency
frequent
frequented
frequenter
frequenters
frequenting
frequently
frequents
```

The answer is [`Lecture11/spell.java`]

```
import java.io.*;
import java.util.Scanner;
public class spell
{
  public static void main(String[] args) throws Exception
  {
     Scanner inone =new Scanner(new FileReader("words"));
     while (inone.hasNext())
     {
       String t= inone.nextLine();
       if (t.startsWith(args[0])) System.out.println(t);
     }
  }
}
```

7.8 A Slightly Better Spell Checker

Rather than expecting the user to enter the word they are looking for on the command line, we can give the user a prompt > to enter a word. The program keeps giving the user another go. The program stops if the user simply presses return. [Lecture11/spell1.java]

```
import java.io.*;
import java.util.Scanner;
public class spell1
{
  public static void main(String[] args) throws Exception
  {
     Scanner inone =new Scanner(new FileReader("words"));
     Scanner in =new Scanner(System.in);
     System.out.print("> ");
     String s=in.nextLine();
     while (s.length()>0)
     {
       while (inone.hasNext())
       {
         String t=inone.nextLine();
         if (t.startsWith(s)) System.out.println(t);
       }
       System.out.print("> ");
       s=in.nextLine();
       inone =new Scanner(new FileReader("words"));
      }
   }
}
```

7.9 Example: A Program to Find Anagrams

Here is a program that might be useful for people who do crossword puzzles. The user types in a word and it finds all the anagrams of the word in the English dictionary. An anagram is simply a re-arrangement, for example `taste` is an anagram of `state`. The program is based very closely on the spell checker spell1.java. The only difference is that instead of checking whether each word in the dictionary *starts with* the word the user entered, we check whether the word is an anagram of the word in the dictionary.

We have written a method called `anagram` which returns true if `s` and `t` are anagrams of each other. We do this by converting s and t to arrays of characters, as and at. Then, for each element of at, we see if we can find it in as. If we do, we change the corresponding element of as to zero, so we never find it again. A zero is not a printable character. If at any stage we don't find a letter we are looking for, then it can't be an anagram. [Lecture11/anagrams.java]

```
import java.io.*;
import java.util.Scanner;
public class anagrams
{
  static int posOf(char c, char [ ] a)
  {
     for (int i=0;i<a.length ;i++) if (a[i]==c) return i;
     return -1;
  }
  public static boolean anagram(String t, String s)
  { char [ ] as = s.toCharArray();
    char [ ] at = t.toCharArray();
    if (at.length != as.length) return false;
    for (int i=0; i < at.length;i++)
    {  int k =posOf(at[i],as);
       if (k!=-1) as[k]=0; else return false;
    }
    return true;
  }
  public static void main(String[] args) throws Exception
  {
     Scanner inOne =new Scanner(new FileReader("words"));
     Scanner in =new Scanner(System.in);
     System.out.print("> ");
     String s=in.nextLine();
     while (s.length()>0)
     {
       while (inOne.hasNext())
       {
         String t=inOne.nextLine();
         if (anagram(t,s) && !t.equals(s))
         System.out.println(t);
       }
       System.out.print("> ");
       s=in.nextLine();
       inOne =new Scanner(new FileReader("words"));
      }
   }
}
```

7.10 Exercises on Chapter 7

7.10.1 Third Line

Write a program that prints out the third line of `fff.dat`. (See page 165 Number 36 for the answer.)

7.10.2 Hundredth Line

Write a program that prints out the hundredth line of `ggg.dat`. (See page 165 Number 37 for the answer.)

7.10.3 Odd Lines

Write a program to print out the odd lines of `ggg.dat`. (See page 165 Number 38 for the answer.)

7.10.4 Even Lines

Write a program to print out the even lines of `ggg.dat`. (See page 166 Number 39 for the answer.)

7.10.5 Cat Choose

Write a program that prints out the contents of a file of the user's choice. (See page 166 Number 40 for the answer.)

7.10.6 Cat Command Line

Write a program that prints out the contents of the file whose name is typed on the command line. (See page 167 Number 41 for the answer.)

7.10.7 Third Char

Write a program that prints out the third character of `fff.dat`. (See page 167 Number 42 for the answer.)

7.10.8 Hundredth Char

Write a program that prints out the hundredth character of `ggg.dat`. The file `ggg.dat` contains all the integers from 1 to 100 in ascending order, one integer per line:

```
1
2
3
.
.
.
97
98
99
100
```

(See page 167 Number 43 for the answer.)

7.10.9 Odd Characters

Write a program to print out the ascii values of the odd chars of `ggg.dat`. (See page 167 Number 44 for the answer.)

The output is

```
1234567891
11
31
51
71
92
12
32
52
72
93
13
33
53
73
94
14
34
54
74
95
15
35
55
75
96
16
```

```
36
56
76
97
17
37
57
77
98
18
38
58
78
99
19
39
59
79
910
```

Make sure you understand the output of this program.

7.10.10 Even Characters

Write a program to print out the ascii values of the even characters of `ggg.dat`. (See page 168 Number 45 for the answer.) The output is

```
01
21
41
61
81
02
22
42
62
82
03
23
43
63
83
04
24
44
64
84
05
25
45
65
85
```

```
06
26
46
66
86
07
27
47
67
87
08
28
48
68
88
09
29
49
69
89
0
```

Make sure you understand the output of this program. Don't forget - the newline character counts as a character.

7.10.11 Swapchars

Write a program called `swapchars.java` using command line arguments that allows any two characters to be swapped. For example if the program was called `swapchars` then to swap all 'a's and 'b's in a file `fred` we would type `java swapchars fred ab`. (See page 168 Number 46 for the answer.)

7.10.12 Manycat

Write a program called `manycat.java` that is the same as `cat.Java` except many files can be printed out one after the other. For example to print `fred1`, `fred2`, and `fred3` we would type `java manycat fred1 fred2 fred3`. (See page 168 Number 47 for the answer.)

7.10.13 Swapcase

Write a program called `swapcase.java` that is the same as `cat.Java` except that all lower case letters are swapped for upper case and vice versa. (Hint: see Character class in [Fla05, Inc].) (See page 169 Number 48 for the answer.)

7.10.14 ASCII

Write a program called `ascii.java` that is the same as `cat.Java` except that as well as printing out each character, it also prints its Unicode value. (See page 169 Number 49 for the answer.)

What is the Unicode value for the newline character?

7.10.15 Remnewline

Write a program called `remnewline.java` that is the same as `cat.java` except that it doesn't print out the newline characters in the file. (See page 170 Number 50 for the answer.)

7.10.16 Tenperline

Write a program called `tenperline.java` that is the same as `remline.java` except that it prints 10 characters on each line. (See page 170 Number 51 for the answer.)

7.10.17 List Words

Write a program that prints out every word in a file, one word per line. (See page 170 Number 52 for the answer.)

7.10.18 Assignment – Spell Checker

Write a spell checker that goes through a file and prints out all the words it finds that aren't in the dictionary.

7.10.19 Hard Assignment – A Better Spell Checker

Write a spell checker that goes through a file and every time it finds a word not in the dictionary it prompts the user either to accept the word or enter a replacement.

7.11 Summary

Having worked on Chapter 7 you will have:

- Understood the purpose of a file.
- Learned how to read a file a character at a time.
- Learned how to read a file a line at a time.
- Learned how to write to a file.
- Written a simple spell-checking program, which checks that all words in a file have been spelled correctly.

Chapter 8

Sorting Arrays and Searching

8.1 Learning Objectives

Chapter 8 explains:

- how to sort elements as they get put into an array
- about one method of sorting an array
- both linear and binary searching
- about complexity analysis
- that sorting arrays of `ints` and sorting arrays of `Strings` is essentially the same.

8.2 Reading

- [LO02] Chapter 10
- [DD07] Chapter 5
- [Wu06] Chapter 15 Sections 15.1 and 15.2
- [Bis01] Chapter 6.4

8.3 Introduction

The reason why it is good to sort things into some order is that it makes things easier to find. Imagine a telephone directory where the names were just in any old order – impossible! Before we sort any array of things, we need to decide what order to sort them in. For example if they are `ints`, we may want ascending or descending order. If the things we are sorting are names, then we may want to sort them in alphabetical order.

8.4 Ways of Sorting

8.4.1 Sorting as you Create

One way of making sure an array is sorted is to insert things in the correct place as we create the array. As you can probably imagine, this will involve a lot of shuffling things around. Imagine ‘in our hand’ we have a partially filled array consisting of the numbers we have so far read in and sorted, and a new number that we have just read in. We want to put this new number into the correct place in the array so that the array is still sorted. We need 3 methods:

1. A method to find where to put the new number in the array.
2. A method to shuffle to the right every element of the array from that point on.
3. And finally, a method that does all the work (by calling the other two methods), namely, finding where to put the new number, shuffling everything one to the right from that point on and updating the array with the new number to be inserted.

Each method has an extra parameter, which tells it where the next free element of the array is. This is very useful because it tells us how far we need to shuffle, etc.
`[Lecture8/SortOnInput.java]`

```
package Lecture8;
import java.util.Scanner;
import java.io.*;
public class SortOnInput
{

        public static void shuffle(int from, int to, int a[ ])
        {
         for (int i=to;i>=from;i--)a[i+1]=a[i];
        }
        public static int findWhereToInsertAscending(int x,int [ ] a, int firstfree)
        { int pos=0;
          while(x>a[pos] && pos<firstfree)pos++;
          return pos;
        }
        public static void insertAscending(int x, int [ ] a, int firstfree)
        { int pos=0;
          int z=findWhereToInsertAscending(x,a,firstfree);
          shuffle(z,firstfree-1,a);
          a[z]=x;
        }
        public static int[] readintsAscending() throws IOException
        {
          Scanner in =new Scanner(System.in);
          System.out.print("How many numbers do you want to enter? ");
          int k=in.nextInt();
          int num[] = new int[k];
          System.out.println("Now type in the numbers");
          for (int i=0;i<k;i++)insertAscending(in.nextInt(),num,i);
          return num;
        }

}
```

8.4.2 Exercise

Write a program that tests `Lecture8/SortOnInput.java`. (See page 171 Number 53 for the answer.)

8.5 Sorting an Array

Suppose we have read n numbers into an array a. To sort a we do $n-1$ passes of the array. After i passes we are sure that the first $i+1$ elements of the array are in order. For the first pass we compare the zeroth element of the array with all the later elements of the array (i.e. with the first, second, ..., up to the last element of the array). If any of these elements is less than the zeroth, we swap them. At the end of doing this we are guaranteed to have the smallest element in the zeroth element of the array. We then move on to the first element of a and repeat the process on the later elements of the array, namely, comparing $a[1]$ with $a[2]$, $a[3]$ etc. and swapping if necessary. At the end of this pass we have $a[0]$ the smallest and $a[1]$ the next smallest. After $n-1$ such passes the array will be completely sorted.

The code to do this is a loop within a loop and is shown in [Lecture8/SortArray.java]

```
public static void SortArray(int[] a, int size)
{
  for (int i=0;i<size-1;i++)
     for (int j=i+1;j<size;j++)
        if (a[i]>a[j]) {int temp=a[i]; a[i]=a[j]; a[j]=temp;}
}
```

For the outer loop we have variable `i` going from 0 to `size-2`. The inner loop has variable `j` going from `i+1` to `size-2`. This ensures that `j` always stays ahead of `i` and that all comparisons are made. if we find that `a[i]>a[j]` we must swap `a[i]` and `a[j]`.

8.5.1 Swapping

Swapping two memory locations, `a[i]` and `a[j]`, for example, involves a temporary storage place.

```
  {int temp=a[i]; a[i]=a[j]; a[j]=temp;}
```

If we simply wrote

```
a[i]=a[j];
a[j]=a[i];
```

the locations `a[i]` and a[j] would both end up with the same value. There are many other sorting algorithms not considered here. Please see other reference books for further information.

8.6 Searching

Having sorted our array we then want to check whether or not it contains certain items. What is an efficient way of doing this?

8.6.1 Linear Searching

How do we look for someone's name in a telephone directory? We do not just start at page 1, and then 2, etc. until we find it. That is what we would have to do if the telephone directory was not in order. The code to do this is: `[Lecture8/LinearSearch.java]`

```
public static boolean linearsearch(int [ ] a, int size, int thing)
{
 int i;
 for (i=0;i<size&&a[i]!=thing;i++);
 return (a[i-1]==thing);
}
```

We are searching for the `int` called `thing` in array `a`. This method returns a boolean, `true` if we find it and `false` if we don't. This code is interesting as we have a `for` loop with an empty body. The way we know that we've found `thing` is that when we leave the `for` loop the last thing we found was `thing`. So we return true if this was the case and false otherwise.

8.6.2 Binary Searching

One way to look for a name is to open the directory in the middle and pick a name n, say. If, by some miracle, n is what we are looking for then we have finished! If the name we are looking for comes before n we can repeat the process, thinking of the first half of the directory as the whole directory (as we never have to look in the second half), or if the name we are looking for comes after n we can repeat the process on the second half. Each time round the loop we are, in effect, cutting the directory in half. This is a very quick way of finding what we want. For example, if the dictionary had 2^n entries, it would take at most n repetitions of the above process before we either found what we want or discovered that it was not there. How do we know if something isn't there? We end up looking in a dictionary containing only one word and it's not the one we are looking for. We implement binary searching by having two `int` variables `first` and `last` to tell us which part of the array to search in for the thing we are looking for. Initially we look in the whole array. As we proceed, `first` and `last` get closer to each other.
`[Lecture8/BinarySearch.java]`

```
public static boolean binarysearch(int [ ] a, int size, int thing)
{
  int first=0,last=size-1;
  int mid;
  while (first <=last)
  {
      mid=first + (last-first)/2;
      if (thing ==a[mid]) return true;
      else if (thing < a[mid]) last = mid-1;
           else first=mid+1;
  }
  return false;
}
```

Don't forget, we are doing integer division.

8.6.3 Exercises on Binary Searching

1. Suppose the array a has 3 elements and thing is at $a[0]$, what are the successive values of mid?

   (See page 171 Number 54 for the answer.)
2. Suppose the a has 4 elements and thing is at a[1], what are the successive values of mid?

   (See page 171 Number 55 for the answer.)
3. Suppose the a has 17 elements and thing is bigger than everything in a, what are the successive values of mid?

   (See page 171 Number 56 for the answer.)

A more useful method would be to say where it found the thing rather than say true it's there or false it's not. We can use -1 to mean that it's not there. [Lecture8/BinarySearchPos.java]

```
public static int binarysearchpos(int [ ] a, int size, int thing)
{
  int first=0,last=size-1;
  int mid;
  while (first <=last)
  {
      mid=first + (last-first)/2;
      if (thing ==a[mid]) return mid;
      else if (thing < a[mid]) last = mid-1;
           else first=mid+1;
  }
  return -1;
}
```

8.7 Efficiency of Different Algorithms: Complexity Analysis

An important question is how fast different algorithms are for searching and sorting. The speed of an algorithm is measured not in number of seconds but rather in terms of the ratio of the number of elements being searched through or sorted and the time. The sensible kinds of question that can be asked are:

> If we doubled the number of elements how much longer would it take?

or

> If we trebled the number of elements how much longer would it take?

8.7.1 Linear Searching

In linear searching, we just start at the beginning and go through from 'left to right' until we find what we are looking for. If we have say, 100 items, it will take on average 100/2=50 comparisons before we find what we are looking for. If, on the other hand, we have a million items, it will take on average 500,000 comparisons. The average number of comparisons is clearly proportional to the number of elements that we start with. If we double the number of elements we will double the average number of comparisons before we find the element we are looking for. If we treble the number of elements we (roughly) treble the amount of time taken to find the element we are looking for. Such an algorithm is said to be of order n, written $O(n)$.

8.7.2 Binary Searching

In binary searching, on the other hand, after each comparison, we halve the search space. If we start off with 32 elements it will take at most five comparisons to find what we are looking for ($32 = 2^5$, alternatively we can say $\log_2 64 = 6$), if we have 64 elements it will take at most 6 comparisons ($64 = 2^6$, alternatively we can say $\log_2 32 = 5$), if we have 4096 elements it will take at most 12 comparisons, if we have 16 million elements it will take at most 24 comparisons! (Since $2^{24} > 16000000$). Clearly this is much more efficient than linear searching. $\log_2(n)$ is the number that 2 must be raised to the power of to give n. The number of comparisons in binary searching is proportional to $\log_2(n)$. We say that binary searching is $O(\log_2(n))$. See [LO02] pages 358-374 for a discussion of complexity analysis.

8.8 Exercises on Chapter 8

8.8.1 A Class for Searching and Sorting Arrays

Consider the class [Lecture8/ArraysOfIntsNew.java]

```
package Lecture8;
import java.util.Scanner;
public class ArraysOfIntsNew
{
        public static Scanner in = new Scanner(System.in);
//
        public static int[] readintArray()
        {
                int k;String s;
                System.out.print("How many ints do you want to enter? ");
                k=in.nextInt();
                int a[] = new int[k];
                System.out.println("Now type in the ints");
                for (int i=0;i<k;i++)
                {
                  a[i]=in.nextInt();
                }
                return a;
        }
//
        static boolean after(int s,int t)
        {
               return (s>t);
        }
//
        public static void printArray(int [ ] a)
        {
           for (int i=0;i<a.length;i++) System.out.println(a[i]);
        }
```

```
//
        public static int [] readAndSortintArrayAscending()
        {
                int k;String s;
        System.out.print("How many ints do you want to enter? ");
        k=in.nextInt();
                int a[] = new int[k];
                System.out.println("Now type in the ints");
                for (int i=0;i<k;i++)
                {
                int z=in.nextInt();
            insertAscending(z,a,i);
                }
        return a;
        }
//
        static void shuffle(int from, int to, int a[ ])
        {
         for (int i=to;i>=from;i--)a[i+1]=a[i];
        }
//
        static int findWhereToInsertAscending(int x,int [ ] a, int firstfree)
        {
          int pos=0;
          while(pos<firstfree && after(x,a[pos]) )pos++;
          return pos;
        }
//
        static void insertAscending(int x, int [ ] a, int firstfree)
        {
          int pos=0;
          int z=findWhereToInsertAscending(x,a,firstfree);
          shuffle(z,firstfree-1,a);
          a[z]=x;
        }
//
        public static void ascendingBubbleSort(int[] a, int size)
        {
          for (int i=0;i<size-1;i++)
          for (int j=i+1;j<size;j++)
          if (after(a[i],a[j])) {int temp=a[i]; a[i]=a[j]; a[j]=temp;}
        }
//
        public static void descendingBubbleSort(int[] a, int size)
        {
          for (int i=0;i<size-1;i++)
          for (int j=i+1;j<size;j++)
          if (after(a[j],a[i])) {int temp=a[i]; a[i]=a[j]; a[j]=temp;}
        }
//
        public static boolean linearSearch(int [ ] a, int size, int thing)
        {
         int i;
         for (i=0;i<size && a[i]!=thing;i++);
         return (a[i-1]==thing);
        }
//
        public static boolean binarySearch(int [ ] a, int size, int thing)
        {
          int first=0,last=size-1;
          int mid;
          while (first <=last)
          {
              mid=first + (last-first)/2;
              if (thing==(a[mid])) return true;
              else if (after(a[mid],thing)) last = mid-1;
                   else first=mid+1;
          }
          return false;
        }
//
        public static int binarySearchPos(int [ ] a, int size, int thing)
        {
          int first=0,last=size-1;
          int mid;
          while (first <=last)
          {
              mid=first + (last-first)/2;
              if (thing==(a[mid])) return mid;
              else if (after(a[mid],thing)) last = mid-1;
                   else first=mid+1;
          }
          return -1;
        }
}
```

8.8.2 Test the Class

Write a program for testing your program. (See page 171 Number 57 for the answer.)

8.8.3 Sorting Arrays of `String`s

Write a similar class which works on arrays of `Strings`. (See page 171 Number 58 for the answer.)

8.8.4 Test your Class

Test your program by asking the user to enter some `Strings` and your program should sort them and print a different message depending whether or not it can find the `String` `"telephone"`.

(See page 173 Number 59 for the answer.)

Notice the only differences between the two are superficial – purely how we compare `Strings` as opposed to `ints`.

8.8.5 Exercises (No Solutions)

1. Write a program that sorts its command line arguments into alphabetical order.

8.9 Example Assignment

Here is an extract from a course handbook:
For calculating the degree classifications
Each student does:

- n1 first year half-units
- n2 second year half-units
- n3 third year half-units

The candidate's overall mark is calculated as $\frac{ux+vy+wz}{ub+wc+vd}$ where

- x is the total marks of the best b first year half units.
- z is the total marks of the best c third year half units.
- y is the total marks of the remaining best d second and third year half units.
- u, v and w are constants which vary from degree to degree. The Values you should use are:

```
static int n1=8;
static int n2=8;
static int n3=8;
static int b=6; /*Number of 1st year marks that count*/
static int c=6; /*number of 3rd year marks that count*/
static int d=8; /*number of remaining 2nd and 3rd year units that count */
static int u=1;
static int v=3;
static int w=5;
```

The degree classification is as follows:

Below 35 is a Fail

35– is a Pass Degree

40– is a Third Class Honours Degree

50– is a Lower Second Class Honours Degree

60– is an Upper Second Class Honours Degree

70– is a First Class Honours Degree

Write a Java program which inputs n1 first year half-unit marks, n2 second year half-unit marks, n3 third year half-unit marks, and then outputs the degree classification.

(See page 174 Number 60 for the answer.)

8.10 Summary

Having worked on Chapter 8 you will have:

- Learned how to sort elements as they get put into an array.
- Learned one method of sorting an array.
- Understood both linear and binary searching.
- Had a brief introduction to complexity analysis.
- Learned that sorting arrays of `ints` and sorting arrays of `Strings` is essentially the same.

Chapter 9

Defining Classes

9.1 Learning Objectives

Chapter 9 explains:

- how to define your own classes
- how to use constructors
- about instance variables
- about instance methods
- how to defined classes in terms of other user-defined classes.

9.2 Reading

- [LO02] Chapter 9
- [Smi99] Chapter 5
- [CK06] Chapters 6 and 7
- [Hub04] Chapter 6
- [Fla05] Chapter 3
- [Kan97] Chapter 1

9.3 Introduction

We have already used objects often in this course. For example, when we wrote

```
DrawingWindow d = new DrawingWindow(500,500);
```

we were declaring a variable of type `DrawingWindow` and assigning it the value `new DrawingWindow(500,500)`.

How are things like `DrawingWindow` actually defined? They are defined as a `class`. The name of the class will be `DrawingWindow`. The `DrawingWindow` class has a *constructor* with two parameters. We can see this because of the expression `new DrawingWindow(500,500)`.

9.4 An Example of Defining your own Class

We now give some examples of how to define and use classes.

9.4.1 WARNING!

For these examples to work, make sure you have got the directory (folder) which contains all the Lecture directories in your CLASSPATH.

To run your programs in a sensible operating system like Unix you would type

```
java LectureSimpleObjects.printHouse
```

at the command line.

9.4.2 A `Date` Class

A Date consists of three fields: a day, a month and a year, all of which are `int`s: `[LectureSimpleObjects/Date.java]`

```
package LectureSimpleObjects;
public class Date
{
    public int day;
    public int month;
    public int year;

    public Date(int d, int m, int y)
    {
      day=d;
      month=m;
      year=y;
    }
}
```

The class `Date` has three fields: `day`, `month`, and `year` all of type `int`. We create an object of type `Date` by giving values to fields using `new`. For example `Date d = new Date(30,12,2009)` or `Date e = new Date(17,1,2010)`. We are calling a *constructor* of the `Date` class. Inside our `Date` class we have:

```
    public Date(int d, int m, int y)
    {
      day=d;
      month=m;
      year=y;
    }
```

This is a constructor. A constructor looks exactly like a method except that it does not have a return type and **its name is the same as the class.** In Java, fields are often referred to as *instance variables*.

9.5 Using a Class that you have Defined

9.5.1 Using the `Date` Class

```
[LectureSimpleObjects/printDate.java]

package LectureSimpleObjects;
class printDate
{
      public static void main(String[] args)
      {
           Date d= new Date(1,12,2003);
           System.out.println(d.day+" "+d.month+" "+d.year);
      }
}
```

Here we define a date `d`, assign it a value, and print out its fields (instance variables).

Objects and Instances

We say variable `d` references (or points to) an object of type `Date`. We have created an *instance* of the class `Date`. It is an instance of `Date` where the values of the fields are 1, 12 and 2003 respectively. When we call a constructor `Date` by saying `new Date(1,12,2003)` we are creating an object and assigning values to its fields.

We can reassign to `d` in exactly the same way that we reassign to any other variable. For example if we said `Date d = new Date(17,1,2010)`, then a new instance of `Date` would be created and this would now be 'pointed to' by variable `d`.

9.5.2 Dot Notation

To access the fields of an object, *dot notation* is used. In

```
package LectureSimpleObjects;
class printDate
{
      public static void main(String[] args)
      {
           Date d= new Date(1,12,2003);
           System.out.println(d.day+" "+d.month+" "+d.year);
      }
}
```

`d.day` corresponds to the `day` field of `d` so has the value 1, `d.month` corresponds to the `month` field of `d` so has the value 12 and `d.year` corresponds to the `year` field of `d` so has the value 2003.

9.6 Using Classes in other Classes

9.6.1 A `Person` Class

A person has a first name, a middle name and a family name, which are all Objects of type `String`; a date of birth which is an Object of type `Date` (as defined in Section 9.4.2); and a sex which is of type `boolean`.
`[LectureSimpleObjects/Person.java]`

```
package LectureSimpleObjects;
public class Person
{
    public String firstname;
    public String lastname;
    public Date dob;
    public boolean sex; //true= female, false = male

    public Person(String f, String l, Date birth , boolean s)
    {
      firstname=f;
      lastname=l;
      dob=birth;
      sex=s;
     }
    public Person(String f, String l, Date birth, String s)
    {
      firstname=f;
      lastname=l;
      dob=birth;
      if (s.charAt(0)=='f' || s.charAt(0)=='F') sex=true;
      else sex=false;
    }
}
```

For a person there are two possible values for sex. We have arbitrarily decided that true means female and false means male. For person, we have defined two constructors. We can have as many constructors as we like provided that they all have a different signature. In the second constructor however, we pass a `String` to give the sex. If the `String` starts with an `'f'` or an `'F'` then it means female, otherwise male.

Using the `Person` Class

`[LectureSimpleObjects/printPerson.java]`

```
package LectureSimpleObjects;
class printPerson
{
      public static void main(String[] args)
      {
           Date d= new Date(1,12,2003);
           Person p  = new Person("John","Smith",d,false);
           System.out.println(p.firstname+" "+ p.lastname+
           " was born on "+p.dob.day+" "+p.dob.month+" "+p.dob.year);
      }
}
```

Here we are creating a Date d and a Person p whose date of birth is d. Notice how we get at the year that d was born using dot notation : `p.dob.year` . We could have defined p without declaring d as follows:

```
 Person p  = new Person("John","Smith",new Date(1,12,2003),false);
```

9.6.2 A `House` Class

A house has a number, an array of persons and a number of rooms. [LectureSimpleObjects/House.java]

```
package LectureSimpleObjects;
public class House
{
    public int number;
    public int rooms;
    public Person [ ] people;

    public House(int number, int rooms, Person [ ] p)
    {
      this.number=number;
      this.rooms=rooms;
      people=p;
    }
}
```

Here we have illustrated the use of the special keyword `this`. Because we couldn't think of different names, we have given the first two parameters, the same names as the fields. To distinguish the formal parameter from the field name we say `this.number`. This is the field of *this* Object called number.

9.6.3 A `Street` Class

A street has a name and an array of houses. [LectureSimpleObjects/Street.java]

```
package LectureSimpleObjects;
public class Street
{
    public String name;
    public House [ ] houses;

    public Street(String name, House [ ] h)
    {
      this.name=name;
      houses=h;
    }
}
```

9.7 An Aside: Expressions for Arrays

We can assign an array explicitly, without giving it a name, as in
[LectureSimpleObjects/ArrayTest.java]

```
public class ArrayTest
{
     public static void main(String [] args)
     {
       System.out.println( (new String[ ]{"my" ,"name","is","fela","kuti"})[3]);
        /* we have created an array with out giving it a name */
       /* This prints fela*/
     }
}
```

The elements of the array are written as a list of expressions between curly brackets and separated by commas. Using this we could define a class Houses.
[LectureSimpleObjects/Houses.java]

```
public class Houses
{
     public static void main(String [] args)
     {
        Date x1 = new Date(2,3,2001);
        Person x2 = new Person("Joseph","Boteju",x1,false);
        Person x3 = new Person("Pushpa","Kumar",new Date(17,2,1942),"male");
        Person x4 = new Person("Ola","Olatunde",new Date(27,8,1983),"female");
        House x5 = new House (5,3,new Person [ ]{x2,x3,x4});
    }
}
```

9.8 Define your House in a Single Expression

Your house can be defined ‘in one go’ as follows: [LectureSimpleObjects/MyHouse.java]

```
public class MyHouse
{

  public static House  it  =
  new House (
            4,5,new Person [ ]
            { new Person("Sebastian", "Danicic",new Date(5,2,1956),"m"),
              new Person("The", "Cat",new Date(27,12,1955),"f")
             }
            );
}
```

LectureSimpleObjects.MyHouse.it is now a constant object standing for my house. It can now be used anywhere.

9.9 A More Complex Example with Instance Methods

Suppose you want to write a method sumandav that returns both the sum and average of an array of ints. Clearly sumandav would take one int array parameter, but what would be the return type of such a method? It has to return something that has two values:

1. An int to hold the sum
2. and a double to hold the average.

We cannot use an array in any obvious way as the return type, because all the elements of an array must be of the same type and here we want two elements of different type: an int and a double. To get round this problem we have to define our own class as follows:
[LectureSimpleObjects/IntAndDouble.java]

```
package LectureSimpleObjects;
public class IntAndDouble
{
    public int i;
    public double d;

    public IntAndDouble(int x,double y) /*A Constructor*/
    {
      i=x;
      d=y;
    }
    public String toString() /*An Instance Method*/
    {
        return "(" + i + "," + d + ")";
    }
}
```

As we will see, defining a class is like defining a new type. Having defined a class we can then declare variables of that type in exactly the same way that we declare variables of existing types. These variables can be assigned values corresponding to Objects of the type.

9.9.1 The Package Statement

We have already studied the package statement in Chapter 4. The package statement at the beginning tells us that IntAndDouble is part of a package called LectureSimpleObjects. Because we have done this we will be able to refer to a type that we have just invented, called

LectureSimpleObjects.IntAndDouble. We will be able to use this type wherever we want. The type LectureSimpleObjects.IntAndDouble has, in essence, become part of the Java programming language for us. If we do not have this package statement we will not be able to use this type anywhere apart from the directory that contains it.

9.9.2 Instance Variables

LectureSimpleObjects.IntAndDouble contains two variable declarations:

```
    public int i;
    public double d;
```

These declare two instance variables called i and d. These are the components (also called *attributes*) of an Object of type LectureSimpleObjects.IntAndDouble. The reason they are public is that, having created an Object of type LectureSimpleObjects.IntAndDouble, we want to be able to refer to its two components separately. The components are often called *fields* or *members*.

9.9.3 A Constructor

```
 public IntAndDouble(int x,double y)
    {
      i=x;
      d=y;
    }
```

Whenever we define a new type like LectureSimpleObjects.IntAndDouble we will also want to define a method which, when called, creates objects of this type. This is called a *constructor*. A constructor always has the same name as the class containing it. In this case, all the constructor does is assign values to the instance variables of the class. The values of the actual parameters in a call to this constructor determine the values of the instance variables in any given instance of the class. This is a very typical use of a constructor in Java.

9.9.4 Creating Objects with `new`

An expression like `new LectureSimpleObjects.IntAndDouble(5,1.0)` creates an instance of an object of type LectureSimpleObjects.IntAndDouble with its i field set to 5 and its r field set to 1.0. Here we are calling the constructor of LectureSimpleObjects.IntAndDouble with actual parameters 5 and 1.0. Later we will see that a class can have arbitrarily many constructors.

Here is the code containing our method sumandav that we initially required. [LectureSimpleObjects/SumAndAv1.java]

```
package LectureSimpleObjects;
public class SumAndAv1
{
     public static IntAndDouble sumAndAv(int [] a)
     {
       int sum=0;
```

```
        double av;
        for (int i=0;i<a.length;i++) sum=sum+a[i];
        av=sum*1.0/a.length;
        IntAndDouble i;//Declare a variable i
                                    // of type LectureSimpleObjects.IntAndDouble
        i = new IntAndDouble(sum,av);
                                    // assign it a value by calling the
                                    //constructor method
        return i;
      }
}
```

We work out the sum and average of the `ints` in our array parameter `a` and then create an instance of an object of type `LectureSimpleObjects.IntAndDouble` with `sum` and `av` as the values of its two fields, and then return this object as the result of our method `sumandav`.

We don't really need the variable `i`. We can simply return the `new` expression itself, as in `[LectureSimpleObjects/SumAndAv4.java]`

```
package LectureSimpleObjects;
public class SumAndAv4
{
     public static IntAndDouble sumAndAv(int [] a)
     {
       int sum=0;
       double av;
       for (int i=0;i<a.length;i++) sum=sum+a[i];
       av=sum*1.0/a.length;
       return new IntAndDouble(sum,av);
     }
}
```

Again, if we want to make `sumandav` usable everywhere we had better include a package statement at the beginning:

`[LectureSimpleObjects/SumAndAv3.java]`

```
package LectureSimpleObjects;
import LectureSimpleObjects.IntAndDouble;
public class SumAndAv3
{
     public static IntAndDouble sumAndAv(int [] a)
     {
       int sum=0;
       double av;
       for (int i=0;i<a.length;i++) sum=sum+a[i];
       av=sum*1.0/a.length;
       return new IntAndDouble(sum,av);
     }
}
```

Here is a program that can use `SumAndAv3`: `[LectureSimpleObjects/UseSumAndAv.java]`

```
package LectureSimpleObjects;
import java.io.*;
import LectureNonVoidMethods.*;
public class UseSumAndAv
{
     public static void main(String [] args) throws IOException
     {
       int [ ] a = Arrays.readintarray();
       IntAndDouble x=SumAndAv3.sumAndAv(a);
       System.out.println("The sum of the numbers you entered is "+x.i);
       System.out.println("The average of the numbers you entered is "+x.d);
     }
}
```

We are using the method `LectureNonVoidMethods.Arrays.readintarray()` to read some tt ints into an array and then calling `LectureSimpleObjects.SumAndAv3.SumAndAv()` to return the sum and average. `x.i` is the `int` field of the `IntAndDouble x` where we store the sum, and `x.d` is the `double` field where we store the average. To run this program you must type:

```
java LectureSimpleObjects.UseSumAndAv
```

9.9.5 Instance Methods

Our class `IntAndDouble` has another method called `toString`. This is an example of an *instance* method. We can tell that it is an instance method because

- It does not have the word `static` at the beginning of its definition.
- Its name is not the same as the name of the class. This means that it is not a constructor.

To see how an instance method is called, consider
`[LectureSimpleObjects/UseSumAndAv2.java]`

```
package LectureSimpleObjects;
import java.io.*;
import LectureSimpleObjects.*;
import LectureNonVoidMethods.*;
public class UseSumAndAv2
{
     public static void main(String [] args) throws IOException
     {
       int [ ] a = Arrays.readintarray();
       IntAndDouble x=SumAndAv3.sumAndAv(a);
       System.out.println("The sum and average of the numbers you entered are"
                        +x.toString()
                       );
     }
}
```

The call to `toString` is achieved using 'dot' notation. `x.toString()` is the `toString()` method of the Object `x`. Every time an object is created, new copies of of the instance variables and instance methods are created. A class can have many instance methods. The purpose of the `toString()` method is to convert `IntAndDoubles` into `Strings` so that they can be printed out. We have chosen to print `IntAndDoubles` in between parentheses and separated by a comma, first the `int` and then the `double`. This was completely our choice.

9.9.6 Leaving out `toString()`

If we include an Object in a `System.out.print()` statement we can leave out the calls to `toString()`. If `x` is an Object with its own `toString()` method then `System.out.print(x)` and `System.out.print(x.toString())` will behave in exactly the same way.

9.10 Exercises on Chapter 9

9.10.1 Exercise on `IntAndDouble`

Rewrite `IntAndDouble.java` (call it `IntAndDouble1.java`) so that it prints an `IntAndDouble` object like this:

```
Double: 2.4367
Int: 6
```

(See page 175 Number 61 for the answer.)

Here is a program to test your answer. [`LectureSimpleObjects/TestIntAndDouble1.java`]

```
package LectureSimpleObjects;
import java.io.*;
import LectureNonVoidMethods.*;
public class TestIntAndDouble1
{
     public static void main(String [] args) throws IOException
     {
       IntAndDouble1 x= new IntAndDouble1(3,4.5);
       IntAndDouble1 y= new IntAndDouble1(2,6.15);
       System.out.println(x.toString());
       System.out.println(y.toString());
     }
}
```

The output should be:

```
Double 4.5
Int 3

Double 6.15
Int 2
```

9.10.2 Expressions For Objects

Write a program that contains

1. An assignment to variable `x1` of a value of type Date, representing 2 March 2001.
2. An assignment to variable `x2` of a value of type Person, representing the man Joseph Boteju born on 2 March 2001.
3. An assignment to variable `x3` of a value type Person, representing the man Pushpa Kumar born on 17 May 1942.
4. An assignment to variable `x4` a value of type Person, representing the woman Ola Olatunde born on 27 August 1983.

(See page 175 Number 62 for the answer.)

Notice that for `x2` and `x3` we have chosen to use the different constructors for `Person`. We could equally well have written:

```
Person x3 = new Person("Pushpa","Kumar",new Date(17,2,1942),false)
```

9.10.3 `toString()` Methods

Add `toString()` methods to each of the classes, `Date`, `House` and `Street`.

9.10.4 Exercises (no solutions)

1. Write `toString()` instance methods for the classes:
   (a) `Person`
   (b) `House`
   (c) `Street`
2. Write a program for testing out your methods.

9.11 Summary

Having worked on Chapter 9 you will have:

- Learned how to define your own classes.
- Learned how to use constructors.
- Learned about instance variables.
- Learned about instance methods.
- Defined classes in terms of other user-defined classes.

Chapter 10

Inheritance

10.1 Learning Outcomes

Chapter 10 explains:

- how to extend a class
- the use of the keyword `super`
- more about instance methods
- about method overriding

10.2 Reading

- [LO02] Chapter 11
- [Smi99] Chapter 12
- [CK06] Chapter 8
- [Hub04] Chapter 7
- [Fla05] Chapter 3
- [Kan97] Chapter 1
- [NK05] Chapter 6
- [DD07] Chapter 7
- [Bis01] Chapter 9

10.3 Introduction

In the class `LectureSimpleObjects.Person`, a person has a first name and a surname. Suppose we decided that we also wanted to include a person's middle name as well as all the other information like date of birth and sex. We could rewrite the whole class `LectureInheritance.person` as in `[LectureInheritance/Person1.java]`

```
package LectureInheritance;
import LectureSimpleObjects.*;
public class Person
{
    public String firstname;
    public String middlename;
    public String lastname;
    public Date dob;
    public boolean sex; //true= female, false = male

    public Person(String f, String m,String l, Date birth , boolean s)
    {
      firstname=f;
      middlename=m;
      lastname=l;
      dob=birth;
      sex=s;
     }
    public Person(String f, String m, String l, Date birth, String s)
    {
      firstname=f;
      middlename=m;
      lastname=l;
      dob=birth;
      if (s.charAt(0)=='f' || s.charAt(0)=='F') sex=true;
      else sex=false;
    }
}
```

Notice that, as well as adding the extra field `middlename`, we have had to change the constructors by giving them an extra parameter to accommodate this extra field. If we hadn't done this, then we would not have been able to give a person a middle name when creating him or her using `new`. We have kept the import statement so that we can refer to the class `LectureSimpleObjects.Date` simply as `Date`. A simpler way of achieving exactly the same goal is using *inheritance*:
[LectureInheritance/Person2.java]

```
package LectureInheritance;
import LectureSimpleObjects.*;
public class Person extends LectureSimpleObjects.Person
{
    public String middlename;

    public Person(String f, String m,String l, Date birth , boolean s)
    {
     super(f,l,birth,s);
     middlename=m;
    }
    public Person(String f, String m, String l, Date birth, String s)
    {
      super(f,l,birth,s);
      middlename=m;
    }
}
```

10.4 The `extends` keyword

By saying `Person extends LectureSimpleObjects.Person` we are saying that the new class

```
 LectureInheritance.person
```

automatically possesses all the fields (but not the constructors) of the class

```
 LectureSimpleObjects.person
```

10.5 The super Keyword

In the new constructors for our new class LectureInheritance.person we refer to a method called super. This simply refers to the corresponding constructor of the class that we are extending. Java will know which one we mean by the types of the parameters we pass it. This one call to super will assign values to all the fields except middlename which we then update separately. Here is a test program to check that the compiler understands our new class

LectureInheritance.person

[LectureInheritance/Test1.java]

```
package LectureInheritance;
import LectureInheritance.*;
import LectureSimpleObjects.*;
class Test1
{
 public static void main(String [ ] args)
 {
   Person a = new Person("fred","mark","Jones",new Date(21,1,1991),"m");
 }
}
```

Because we have a package statement at the beginning, Java assumes that any references to the class Person refer to the one in LectureInheritance and not the one in LectureSimpleObjects. Don't forget, to run it we must type

```
java LectureInheritance.Test1
```

We can still refer to the **old** type of person by giving the full name of the class, as in [LectureInheritance/Test2.java]

```
package LectureInheritance;
import LectureInheritance.*;
import LectureSimpleObjects.*;
class Test2
{
 public static void main(String [ ] args)
 {
   Person a = new Person("fred","mark","Jones",new Date(21,1,1991),"m");
   LectureSimpleObjects.Person p;
   p=new LectureSimpleObjects.Person("fred","Jones",new Date(21,1,1991),"m");;
 }
}
```

10.6 More about Instance Methods

In Chapter 9 we introduced instance methods. Further examples are now given. Suppose we wanted to work out a person's age. To calculate a person's age we define a method called age in the class person. This method takes today's date as a parameter. In order to work out someone's age we need today's date (the parameter we are passing) and the person's date of birth, this.dob.
[LectureInheritance/Person.java]

```
package LectureInheritance;
import LectureSimpleObjects.*;
public class Person extends LectureSimpleObjects.Person
{
    public String middlename;

    public Person(String f, String m,String l, Date birth , boolean s)
    {
     super(f,l,birth,s);
     middlename=m;
    }
    public Person(String f, String m, String l, Date birth, String s)
    {
      super(f,l,birth,s);
      middlename=m;
    }
   public int age(Date today)
   {
     int age=today.year - this.dob.year;
     if (today.month > this.dob.month) return age;
     if (today.month ==this.dob.month && today.day >=this.dob.day) return age;
     return age-1;
   }
}
```

The method age does not have the word static in front of it. This is how we know that it is an instance method. We can only use this method in conjunction with an instance of the class person that has been created with new. Here we test out our new instance method:
[LectureInheritance/Test3.java]

```
package LectureInheritance;
import LectureInheritance.*;
import LectureSimpleObjects.*;
class Test3
{
 public static void main(String [ ] args)
 {
   Person a = new Person("fred","mark","Jones",new Date(21,1,1991),"m");
   System.out.println("age is "+a.age(new Date(30,8,2001)));
 }
}
```

We pass the age method of variable a, a date. This date is 30 August 2001. The program prints 10, since that is Fred's age on 30 August 2001.

10.7 Shapes Revisited

Here is a class for drawing a horizontal line of stars:
[LectureInheritance/HorizLine.java]

```
package LectureInheritance;
import java.io.*;
class HorizLine
{
   int length; /* instance variable of HorizLine*/

   public HorizLine(int l) /*constuctor for HorizLine*/
   {
    length=l;
   }
   public void Draw()    /*class method of HorizLine*/
   {
     for (int i=0;i<length;i++)System.out.print("*");
   }
   public void Drawln()
   {
     Draw();System.out.println();
   }
}
```

Here is a program to test it: [LectureInheritance/Test4.java]

```
package LectureInheritance;
import LectureInheritance.*;
class Test4
{
 public static void main(String [ ] args)
 {
   (new BetterLine(15,'!','-')).Drawln();
   (new HorizLine(7,'*',' ')).Draw();
 }
}
```

The output is:

```
***************
*******
```

10.7.1 Extending `HorizLine` (Method Overriding)

We may want to be able to draw horizontal lines with different characters apart from stars. In fact we want to be able to specify different characters to occur at the ends of the line from the middle. So we can draw lines like

```
&......&
&............&
```

which have an & at either end and dots in the middle, or like

```
*           *
*   *
```

which have asterisks at either end and spaces in the middle.
Consider [LectureInheritance/BetterLine.java]

```
package LectureInheritance;
import java.io.*;
class BetterLine extends HorizLine
{  /*inheritance*/
    char endchar;
    char middlechar;
    public BetterLine(char end,char middle, int len)
    {
      super(len);   /* this calls the constructor of HorizLine */
      endchar=end;
      middlechar=middle;
    }
    public void Draw()   /* method overriding */
    {
     for (int i=0;i<length;i++)
       if (i==0 || i == length-1)
          System.out.print(endchar);
       else
          System.out.print(middlechar);
    }
}
```

Since `BetterLine` extends `HorizLine` it already has a `length` field and two instance methods `draw()` and `drawln()`. We add two new fields of type `char` for specifying the end and middle characters of the lines we want to draw. However, we do not want to use the `draw` and `drawln` methods of `HorizLine` since they only draw asterisks. We want to draw `endchar`s at either end and `middlechar`s in the middle. So we *override* the `draw` method of `HorizLine`. That is, we define a method which has exactly the same name and signature in the new class as in the class we are extending. Interestingly, we do not need to explicitly override `Drawln` (though we can if we want) since that calls `Draw` which has been overridden. To test it we can use [LectureInheritance/Test5.java]

```
package LectureInheritance;
class Test5
{
 public static void main(String [ ] args)
 {
   (new BetterLine('-',':',15)).Drawln();
   (new BetterLine('*',' ',7)).Drawln();
   (new BetterLine('*',' ',7)).Drawln();
 }
}
```

The output is

```
-:::::::::::::
*     *
*     *
```

For drawing our hollow rectangles and triangles we can use a `HollowLine`. This can be defined by extending `LectureInheritence.BetterLine` as follows:
`[LectureInheritance/HollowLine.java]`

```
package LectureInheritance;
import java.io.*;
public class HollowLine extends BetterLine
{
   public HollowLine(int n)
   {
      super('*',' ',n);
   }
}
```

A `Hollowline` of length n is a `BetterLine` of length n with the end character being a `'*'` and the middle characters a `' '`. Here, the call to `super` is a call to `BetterLine`.

10.7.2 Rectangles of Stars

We can use `LectureInheritence.HorizLine` to make a class `LectureInheritence.Rectangle` as follows: `[LectureInheritance/Rectangle.java]`

```
package LectureInheritance;
public class Rectangle
{
  public int height;
  public int width;
  public Rectangle(int h, int w)
  {
    height=h;
    width=w;
  }
  public void Draw()
  {
    HorizLine m= new HorizLine(width);
    for(int j=0;j<height;j++) m.Drawln();
  }
}
```

This has two `int` fields, `height` and `width`. In its `Draw` method, first we create an instance `m` of `HorizLine` of length equal to the width of the rectangle. Then we have a loop which draws this horizontal line `height` times. Clearly, this will produce a solid rectangle of stars. A program to test our rectangle class is:
[LectureInheritance/Test6.java]

```
package LectureInheritance;
class Test6
{
 public static void main(String [ ] args)
 {
   (new Rectangle(3,7)).Draw();
   (new Rectangle(7,4)).Draw();
 }
}
```

The output is:

```
*******
*******
*******
****
****
****
****
****
****
****
```

10.7.3 Better Rectangles

A better rectangle is one made up of `BetterLines`.
[LectureInheritance/BetterRectangle.java]

```
package LectureInheritance;
public class BetterRectangle extends Rectangle
{
   char ends;
   char middle;
   public BetterRectangle(int h, int w, char m,char e)
   {
     super(h,w);
     ends=e;
     middle=m;
   }
   public void Draw()
   {
     BetterLine topandbottomline = new BetterLine(ends,ends,width);
     BetterLine  middleline = new BetterLine(ends,middle,width);
     for(int j=0;j<height;j++)
```

```
          if (j==0 || j==height-1)
             topandbottomline.Drawln();
          else
             middleline.Drawln();
    }
}
```

We have deliberately made a *better rectangle* a generalisation of a *hollow rectangle*. For the first and last (top and bottom) lines, we draw a *betterline* with the end and middle characters both set to `ends`. For the middle lines we draw a `BetterLine` with the end character set to `ends` and the middle character set to `middle`:
`[LectureInheritance/Test7.java]`

```
package LectureInheritance;
class Test7
{
 public static void main(String [ ] args)
 {
   (new BetterRectangle(5,7,'.','!')).Draw();
   (new BetterRectangle(7,4,' ','*')).Draw();
   (new BetterRectangle(7,4,'8','=')).Draw();
 }
}
```

The output is:

```
!!!!!!!
!.....!
!.....!
!.....!
!!!!!!!
****
*  *
*  *
*  *
*  *
*  *
****
====
=88=
=88=
=88=
=88=
=88=
====
```

10.7.4 Hollow Rectangles

We can extend the class `BetterRectangle` to produce Hollow Rectangles as follows:
`[LectureInheritance/HollowRectangle.java]`

```
package LectureInheritance;
public class HollowRectangle extends BetterRectangle
{
  public HollowRectangle(int h, int w)
  {
    super(h,w,' ','*');
  }
}
```

10.8 Exercises on Chapter 10

10.8.1 Test `HollowRectangle`

Write a program using HollowRectangle whose output is

```
******************
*                *
*                *
*                *
*                *
******************
*******
*     *
*     *
*     *
*     *
*     *
*     *
*     *
*     *
*     *
*     *
*******
```

(See page 175 Number 63 for the answer.)

10.8.2 Extend `Rectangle` to `Square`

Extend `Rectangle` to produce a class `Square` (See page 175 Number 64 for the answer.)

10.8.3 Extend `BetterRectangle` to `BetterSquare`

Extend the class `BetterRectangle` to produce a class `BetterSquare`.

(See page 176 Number 65 for the answer.)

10.8.4 Left Bottom Triangles

1. Use a `BetterLine` etc. to define a class `HollowLeftBottomTriangle`.
   (Do it in a similar way by extending a class called `BetterLeftBottomTriangle`.) (See page 176 Number 66 for the answer.)
2. Similarly define a class called `SolidLeftBottomTriangle` which is entirely made up of stars. (See page 176 Number 67 for the answer.)
3. Write a program to test your classes. (See page 176 Number 68 for the answer.)

10.9 Exercises (no solutions)

10.9.1 Left Top Triangles

Repeat exercise 10.8.4 for left top triangles.

10.9.2 Right Triangles

Repeat exercise 10.8.4 for Right top and right bottom triangles.

10.10 Summary

Having worked on Chapter 10 you will have:

- Learned how to extend a class.
- Learned the use of the keyword `super`.
- Learned more about instance methods.
- Learned about method overriding.

Chapter 11

Exception Handling

11.1 Learning Objectives

Chapter 11 explains:

- how to handle exceptions using `try` and `catch`
- what it means to throw an exception.

11.2 Reading

- [CK06] Chapter 15
- [Smi99] Chapter 13
- [Hub04] page 230
- [Fla05] pages 56-60
- [NK05] pages 103-115
- [Kan97] Q2.22-Q2.26
- [DD07] Chapter 12

11.3 Introduction

An exception is a signal that an error has occurred. To *throw* an exception means to signal an error. To *catch* an exception means to `handle` the error, i.e. to take action to recover from the error.
Consider the program [`LectureExceptions/test1.java`]

```
package LectureExceptions;
import java.io.*;
class  test1
{
        public static void main(String[] args)
        {
          int x=Integer.parseInt("dl;fkas;lkf");
        }
}
```

This program compiles correctly but when we run it we get:

```
Exception in thread "main" java.lang.NumberFormatException: dl;fkas;lkf
at java.lang.Integer.parseInt(Integer.java:405)
at java.lang.Integer.parseInt(Integer.java:454)
at LectureExceptions.test1.main(test1.java:7)
```

A `java.lang.NumberFormatException` has been thrown because we are trying to parse the `String` `dl;fkas;lkf` as an `int`.

Now consider the program [`LectureExceptions/test2.java`]

```
package LectureExceptions;
import java.io.*;
class  test2
{
 public static void main(String[] args)
 {
   int s =System.in.read();
 }
}
```

This program gives the compilation error:

```
test2.java:7: unreported exception java.io.IOException; must be caught or declared to be thrown
   int s =System.in.read();
                       ^
1 error
```

A cure is to add `throws IOException` as in [`LectureExceptions/test3.java`]

```
package LectureExceptions;
import java.io.*;
class  test3
{
 public static void main(String[] args) throws IOException
 {
   int s =System.in.read();
 }
}
```

An alternative is to catch the `IOException` as in [`LectureExceptions/test4.java`]

```
package LectureExceptions;
import java.io.*;
class  test4
{
 public static void main(String[] args)
 {
   try {int s  =System.in.read();}
   catch (IOException e)
   {
   }
 }
}
```

Here we use `try` and `catch`.
Now consider [`Lecture2/Add1ForceOkLoop.java`]

```
//This program stays in a loop until the user enters an integer
//
import java.io.*;
import java.util.*;
class  Add1ForceOkLoop
{
  public static void main(String[] args) throws IOException
  {
      Scanner in =new Scanner(System.in);
      boolean finished=false;
      while (!finished)
      {
        System.out.print("Enter Number>");
        String s =in.nextLine();
        try
        {
            int x=Integer.parseInt(s);
            System.out.println("The answer is " + (x+1));
            finished=true;
          }
        catch(Exception e)
        {
            System.out.println("You made a mistake-try again");
          }
      }
  }
}
```

In this program we have a `try` clause containing `int x=Integer.parseInt(s);` if the `String s` does not parse correctly to an integer (i.e. it contains characters which are not digits), then an *exception* will be thrown. The exception is caught by the `catch` clause which displays a message telling the user that they have made a mistake. We could write a method with one input parameter of type `String` which returns `true` if the `String` represents an `int` and `false` otherwise:
[`LectureExceptions/IntException.java`]

```
package LectureExceptions;
public class  IntException
{
   public static boolean isInt(String s)
   {
     try
     {
       Integer.parseInt(s); return true;
     }
     catch(Exception e)
     {
      return false;
      }
    }
}
```

We can use this method to produce the same behaviour as in `Lecture2/Add1ForceOkLoop.java`. See `[LectureExceptions/test5.java]`

```
package LectureExceptions;
import java.util.*;
//This program stay in a loop until the user enters an integer
//
import java.io.*;
class  test5
{
  public static void main(String[] args) throws IOException
  {
   Scanner in =new Scanner(System.in);
   boolean finished=false;
   while (!finished)
   {
       System.out.print("Enter Number>");
       String s =in.nextLine();
       if  (IntException.isInt(s))
          {
            int x=Integer.parseInt(s);
            System.out.println("The answer is " + (x+1));
            finished=true;
           }
       else
        {
            System.out.println("You made a mistake-try again");
        }
   }
  }
}
```

This code is slightly inefficient because we are parsing the `String` **twice** when it is correct. Explain this!

11.4 'File Not Found' Exceptions

Consider the program:
`[LectureExceptions/fileNotFound.java]`

```
package LectureExceptions;
import java.io.*;
class  fileNotFound
{
        public static void main(String[] args) throws IOException
        {
            FileReader in = new FileReader("silly.dat");
            int c;
            while ((c=in.read())!=-1) System.out.print((char)c);
        }
}
```

Because the file `silly.dat` does not exist, when we run this program, it says:

```
Exception in thread "main" java.io.FileNotFoundException: silly.dat (No such file or directory)
at java.io.FileInputStream.open(Native Method)
at java.io.FileInputStream.<init>(FileInputStream.java:64)
at java.io.FileReader.<init>(FileReader.java:31)
at LectureExceptions.propag1.main(propag1.java:9)
```

Now consider the program [`LectureExceptions/catchFileNotFound.java`]

```
package LectureExceptions;
import java.io.*;
class  catchFileNotFound
{
        public static void main(String[] args) throws IOException
        {
          try
          {
              FileReader in = new FileReader("silly.dat");
              int c;
              while ((c=in.read())!=-1) System.out.print((char)c);
          }

          catch (IOException f)
          {
               System.out.println("file not found ");
          }
        }
}
```

Now the program, when it runs, prints out

```
File not found
```

because we are catching the `FileNotFoundException`. Now we rewrite the program so that it asks the user to enter the file name.
[`LectureExceptions/catchUserFileName.java`]

```
package LectureExceptions;
import java.io.*;
import java.util.*;
class  catchUserFileName
{
        public static void main(String[] args) throws IOException
        {
          System.out.print("Enter File Name ");
          Scanner kbin =new Scanner(System.in);
          String s = kbin.nextLine();
          try
          {
              FileReader in = new FileReader(s);
              int c;
              while ((c=in.read())!=-1) System.out.print((char)c);
          }

          catch (IOException f)
          {
               System.out.println("file "+s+" not found ");
          }
        }
}
```

Now the program asks the user to enter the file name. If the user enters a non-existent file name then the error is *caught* with a message saying that the file doesn't exist. We now rewrite the program to force the user to enter an existing file name.
[LectureExceptions/forceUserFileName.java]

```
package LectureExceptions;
import java.io.*;
import java.util.*;
class  forceUserFileName
{
        public static void main(String[] args) throws IOException
        {
          System.out.print("Enter File Name ");
          Scanner kbin =new Scanner(System.in);
          String s = null;
          FileReader in=null;
          boolean exists=false;
          while (!exists)
          {
            try
            {
              s = kbin.nextLine();
              in = new FileReader(s);
              exists=true;
            }

            catch (IOException f)
            {
               System.out.print("File "+s+" not found. Please Re-enter: ");
            }
          }
          int c;
          while ((c=in.read())!=-1) System.out.print((char)c);
       }
}
```

Here we stay in a loop until an existing file name is entered.

11.5 Throwing Exceptions

We can handle conditions by throwing exceptions when conditions are encountered, and catching them. Consider [LectureExceptions/catchEOF.java]

```
package LectureExceptions;
import java.io.*;
import java.util.*;
class  catchEOF
{
        public static void main(String[] args) throws IOException
        {
          System.out.print("Enter File Name ");
          Scanner kbin =new Scanner(System.in);
          String s = null;
          FileReader in = null;
          boolean exists=false;
          while (!exists)
          {
            try
            {
              s = kbin.nextLine();
              in =new FileReader(s);
              exists=true;
            }

            catch (IOException f)
            {
               System.out.print("File "+s+" not found. Please Re-enter: ");
            }
          }
          int c;
          boolean end=false;
             try
             {
                 while (true)
                 {
                     c=in.read();
                     if (c==-1) throw new EOFException();
                     System.out.print((char)c);
                 }
              }
            catch (EOFException e)
            {
               end=true;
            }

     }
}
```

This handles the ‘End of file’ condition.

11.6 Exercises on Chapter 11

11.6.1 'File Not Found' Exceptions

Re-do each exercise in Chapter 7, but this time catch the 'file not found' exceptions; where the file name is input by the user, force the user to re-enter a file name until it is found otherwise print a suitable error message.

11.7 Summary

Having worked on Chapter 11 you will have:

- Learned how to handle exceptions using `try` and `catch`.
- Learned what it means to throw an exception.

Chapter 12

Vectors

12.1 Learning Objectives

Chapter 12 explains:

- the similarities and differences between a `Vector` and an array
- how `Vectors` are used for storing arbitrarily large amounts of data.
- how to sort `Vectors`
- how to manipulate the contents of files by storing them in a `Vector` and then manipulating the `Vector`
- a way of writing a complete system for processing student marks.

12.2 Reading

- [Smi99] Chapter 21 and Chapter 7
- [Hub04] Chapter 8
- [Fla05] Page 160-178, 824
- [Kan97] Q2.13-Q2.19
- [DD07] Chapter 18
- [NK05] Chapter 8

12.3 Introduction

A `Vector` (see `java.util.Vector`) is a structure a bit like an array for storing many Objects. The main differences between `Vectors` and arrays are:

1. `Vectors` are *dynamic*: they can grow and shrink in size (number of elements) while the program is running.
2. All elements of a `Vector` are of type `Object`. Arrays can have elements of type `int`, `char` etc. `Vectors` cannot.

`Vectors` can be used whenever you would have used an array but do not know in advance how many elements you need to store. The most useful instance methods of class `Vector` are:

1. `void addElement()` Adds an Object to the end of a `Vector`
2. `Object elementAt(int i)` Returns the ith element
3. `int Size()` Returns the number of elements

4. `void removeElementAt(int i)` Removes the ith element
5. `void setElementAt(Object x, int i)` Changes the ith element to `x`.

12.4 Autoboxing, Unboxing and Generics

Before Java 1.5, handling `Vectors` and other similar types was much more difficult than it is now. With Autoboxing, Unboxing and Generics, things have become much easier.

Consider [`Lecture9/LargestInVec2.java`]

```
package Lecture9;
import java.util.*; /*always needed if you use vectors*/
import java.util.Scanner;
public class LargestInVec2
{
    public static void main(String [ ] args)
    {
        Vector <Integer>  v = new Vector();
        v= readInVectorOfIntegers();
        System.out.println("The largest you typed in was " + largest(v));
    }


    public static Vector <Integer> readInVectorOfIntegers()
    {
       Scanner in = new Scanner(System.in);
       Vector <Integer> w = new  Vector ();
       while(in.hasNextInt()) w.addElement(in.nextInt());
       return w;
    }

    public static int largest(Vector <Integer> v)
    {
        int largestsofar;
        if (v.size()>0)
        {
            largestsofar= v.elementAt(0);
            for (int i=0;i<v.size();i++)
            if (largestsofar< v.elementAt(i))
                       largestsofar= v.elementAt(i);
            return largestsofar;
        }
                return 0;
    }
}
```

This program allows us to store arbitrary amounts of data into the `Vector`. We can go on entering integers for as long as we like. The input terminates when the user types a non-integer value. The program then prints out the largest value entered. It is exactly the same algorithm that we used for finding the largest number in an array.

Here we are declaring v to be a Vector that only holds ints. Autoboxing and unboxing allow us to think of the type Integer as int. Really Vectors can only hold Objects but with autoboxing and unboxing we can *pretend* that they can hold simple types like int and char.

If we try and put a non-integer value in a Vector <Integer> we get a compilation error: [Lecture9/errorVec.java]

```
import java.util.*;
class errorVec
{
  public static void main(String[] args)
  {

    Vector <Integer> v = new Vector();
    v.addElement("www");
  }
}
```

```
errorVec.java:8: addElement(java.lang.Integer) in java.util.Vector<java.lang.Integer>
cannot be applied to (java.lang.String) v.addElement("www");
```

12.5 Untyped Vectors

Consider [Lecture9/Vector1.java]

```
package Lecture9;
import java.util.*;
class Vector1
{
   public static void main(String[] args)
   {
     int k=5;
     Vector nums = new Vector();
     for (int i=0;i<k;i++) nums.addElement(i);
     nums.addElement('z');
     nums.addElement("zz");
     nums.addElement("Goldsmiths is great");
     for (int i=nums.size()-1;i>=0;i--)System.out.println(nums.elementAt(i));
   }
}
```

Here, since we have not specified the type of the elements that we can put in the Vector we can use it to store all types of Object. Here we are adding five Integers, a Character and two Strings.

12.5.1 Exercise

Work out the output of `Lecture9/Vector1.java` above. (See page 177 Number 69 for the answer.)

12.6 Sorting `Vector`s

In the program `SortVector` below, the algorithm we use for sorting `Vectors` is exactly the same as the one we used for arrays. The sort method, `sort(v)` returns a `Vector` which has the same elements as `v` but is sorted in ascending order. The parameter to `sort` is of course not changed by `sort`. The first thing that the method `sort` does is make a copy of `v` using its `clone` method, `w= v.clone()`. Note the use of the method `Vector.setElementAt`. `[Lecture9/SortVector.java]`

```
package Lecture9;
import java.util.*;
class SortVector
{
    public static void main(String[] args)
    {
            System.out.println("Enter some integers terminated by a non-integer:");
            Vector <Integer> vec=LargestInVec2.readInVectorOfIntegers();
            Vector <Integer> answer = sort(vec);
            System.out.println("Sorted gives:");
            for (int i=0;i<answer.size();i++) System.out.println(answer.elementAt(i));
            System.out.println("The orginal order was:");
            for (int i=0;i<vec.size();i++) System.out.println(vec.elementAt(i));

    }

    public static void swap (Vector w, int i, int j)
    {
        Object temp=w.elementAt(i);
        w.setElementAt(w.elementAt(j),i);
        w.setElementAt(temp,j);
    }


    public static Vector <Integer> sort(Vector <Integer> v)
    {
        Vector <Integer> w= new Vector();
        for (int i=0;i<v.size();i++)w.addElement(v.elementAt(i));
        for (int i=0;i<w.size()-1;i++)
           for (int j=i+1;j<w.size();j++)
               if (w.elementAt(i) > w.elementAt(j)) swap(w,i,j);
        return w;
    }
}
```

What does the following program do? `[Lecture9/RefParams.java]`

```
package Lecture9;
```

```
import java.util.*;
class RefParams
{
    public static void main(String[] args)
    {
            System.out.println("Enter some integers terminated by a non-integer:");
            Vector <Integer> vec=LargestInVec2.readInVectorOfIntegers();
            Vector <Integer> answer = sort(vec);
            System.out.println("Sorted gives:");
            for (int i=0;i<answer.size();i++) System.out.println(answer.elementAt(i));
            System.out.println("The orginal order was:");
            for (int i=0;i<vec.size();i++) System.out.println(vec.elementAt(i));

    }

    public static void swap (Vector w, int i, int j)
    {
        Object temp=w.elementAt(i);
        w.setElementAt(w.elementAt(j),i);
        w.setElementAt(temp,j);
    }


    public static Vector <Integer> sort(Vector <Integer> v)
    {
        for (int i=0;i<v.size()-1;i++)
           for (int j=i+1;j<v.size();j++)
               if (v.elementAt(i) > v.elementAt(j)) swap(v,i,j);
        return v;
    }
}
```

(See page 177 Number 70 for the answer.)

12.6.1 Exercises on Chapter 12

Write a program that reads some numbers (terminated by a non-integer), stores them in a `Vector` and prints out:

1. the smallest
2. the sum
3. the average.

12.6.2 Exercises (no solutions)

1. Write a program that asks the user to enter some numbers (terminated by a non-integer) and then prints them out in the opposite order to which they were entered. For example, if the user types `1 4 3 5` the output should be `5 3 4 1`.

2. Write a program that asks the user to enter some numbers and then prints them out in the opposite order to which they were entered, and then prints them out in the same order they were entered. For example, if the user types `1 4 3 5` the output should be `5 3 4 1 1 4 3 5`.
3. Write a program where first the user enters some numbers and then the program asks the user to pick a number. The program tells the user how many of these numbers the user entered. For example, if the user entered `7 4 4 3 1 7` and then the number chosen was `7`, then the program would output `2` because the user entered two sevens. If the number chosen by the user was `8`, then the program would output `0`.
4. Write a program where the user types in a number of `Strings` stored in a `Vector` of `Strings` and then the program prints out the longest `String`.
5. Write a method `readSort` which is like `Input3.readInVectorOfIntegers` but it sorts them into ascending order as it puts them into the `Vector`.
6. Write a method that sorts `Vectors` in descending order.
7. Write a method that does a binary search for an `Integer` in a `Vector`. It must return the position of the `Integer` in the `Vector` and -1 if it is not there. Assume the `Vector` is sorted in ascending order.
8. Write a method for removing all duplicates from a `Vector` of Objects.

12.7 Manipulating Files Using `Vector`s

We now show how problems involving file manipulation can be tackled by storing the contents of the file in a `Vector`, then manipulating the `Vector` and finally outputting the resulting `Vector` back to the file.

12.7.1 Character by Character

Consider [`Lecture12/fileToVector.java`]

```
package Lecture12;
import java.io.*;
import java.util.*;
public class fileToVector
{
    public static Vector <Character> fileToVector(String s) throws Exception
    {
        Vector <Character> w=new Vector();
        FileReader inone =new FileReader(s);
        int t=inone.read();
        while (t!=-1)
        {
            w.addElement((char)t);
            t=inone.read();
         }
         return w;
     }
    public static void main(String[] args) throws Exception
    {
        Vector <Character> v=fileToVector(args[0]);
        for (int i=0;i<v.size();i++)
        System.out.print(v.elementAt(i));
    }
}
```

Here, the method `fileToVector.fileToVector(s)` reads from the file, `s`, a character at a time and returns the `Vector` consisting of the same sequence of `Characters` that were in the file `s`. Note the Vector contains `Characters` not `chars`.

12.7.2 Line by Line

Now consider [`Lecture12/lines.java`]

```
package Lecture12;
import java.io.*;
import java.util.*;
public class lines
{

    public static Vector <String> fileToVector(String s) throws Exception
    {
        Vector v=new Vector();
        Scanner inone =new Scanner(new FileReader(s));
        while (inone.hasNextLine()) v.addElement(inone.nextLine());
        return v;
    }


    public static void main(String[] args) throws Exception
    {
        Vector <String> w=fileToVector(args[0]);
        for (int i=0;i<w.size();i++) System.out.println(w.elementAt(i));
    }
}
```

Here we are reading the file a line at a time. Each element of the resulting `Vector` will be a `String`. So there will be as many elements of the resulting `Vector` as there are lines in the original file. When we print out the contents of the `Vector` we must therefore use `System.out.println` if we are to preserve the lines of the original file. If the original file contained no end of line characters then the whole file would end up being stored in the zeroth element of the `Vector`.

12.7.3 Exercises

12.7.4 Longest Line in a File

Write a program that reads a file into a `Vector` of `Strings`, one element per line and then finds the longest line. (See page 177 Number 71 for the answer.)

Longest Word in the English Language

Use the program above to find the longest word in the English language.

12.7.5 Occurrences of Printable Characters in a File

Write a program `occ1.java` which prints out the number of occurrences of each printable character in a file. Assume the printable characters are those whose Unicode values lie between 30 and 126 inclusive. For example, `java Lecture12.occ1.java longestline.java` should produce:

```
  24
" 4
( 16
) 16
* 2
+ 3
. 12
0 2
1 1
2 1
; 9
< 1
= 4
> 1
A 2
E 1
L 4
S 6
T 1
V 2
[ 2
\ 1
] 2
a 20
b 2
c 10
d 1
e 32
f 7
g 18
h 4
i 32
j 2
k 1
l 20
m 10
n 29
o 31
p 7
r 21
s 23
```

```
t 45
u 6
v 3
w 1
x 1
y 1
z 1
{ 3
} 3
```

(See page 177 Number 72 for the answer.)

12.7.6 Longest Word in the English Language

Use `Lecture11/words` and `Lecture12/longestline.java` to find the longest word in the English language. (See page 178 Number 73 for the answer.)

12.7.7 Occurrences, Most Popular First

Using a `Vector`, write another program like `Lecture12/occ1.java` which prints out the occurrences of each character in a file but this time prints them in order of popularity. Order the output so that the most frequently occurring character comes first etc. (See page 178 Number 74 for the answer.)

12.7.8 Do the Same Without a `Vector`

Write a program that does the same as `Lecture12/occ2.java` but this time store the information from the file directly into the array `occs`. (See page 178 Number 75 for the answer.)

12.7.9 Frequency

Write a program `freq.java` which you can use to find the frequency as a percentage of each character in a piece of text. (See page 179 Number 76 for the answer.)

12.7.10 Finding Words in Dictionary

Write a program `find.java` so that `java find ../Lecture11/words fred` will say 'yes' if fred is an English word (i.e. if it finds it in the dictionary) and 'no' if it isn't. (See page 180 Number 77 for the answer.)

12.7.11 Exercise (no solution)

Write a program `firstlast.java` that can be used to print out all English words that start and end with the same letter. The choice of letter should be a command line argument. So, for example, `java firstlast /usr/dict/words w` should print all English words that start and end with the letter 'w'.

12.8 A System for Processing Student Marks

We now develop a user-defined class for holding information about students on the Java course. For each student we hold the following information:

- name
- exam mark
- coursework mark
- total mark
- grade.

The student's grade and total mark is worked out from the exam mark and the coursework mark. The total mark is 60% of the exam mark plus 40% of the coursework mark.

The grades are calculated in using the total mark as follows:

A: total $\geq$ 70%

B: 60% $\leq$ total $<$ 70%

C: 50% $\leq$ total $<$ 60%

D: 40% $\leq$ total $<$ 50%

E: 35% $\leq$ total $<$ 40% and both exam and coursework are not less than 35

F: 0% $\leq$ total $<$ 35% or either exam or coursework less than 35

12.9 The Class `Student`

We define a class called `Student` which has five fields, one constructor, `Student`, and two static methods, called `grade` and `total` which work out the grade and total for a student. These two static methods are called by the constructor `Student`. The constructor`Student` has three parameters: the name, coursework mark and exam mark of the student. `Student` also has an instance method called `toString()` which converts a `Student` to a `String`. Its output is a `String` consisting of the five fields of a `Student`. Here is the class [`Lecture13/Student.java`]

```
//ww
package Lecture13;
/*Student.java*/
public class Student
{
    public String name;
    public int exam;
    public int cwk;
    public int total=0;
    public String grade;
    public Student(String name, int exam, int cwk)
    {
        this.name=name;
        this.exam=exam;
        this.cwk=cwk;
        this.total=total(this);
        this.grade=grade(this);
    }
    public static String grade(Student s)
    {     if (s.total<35||s.exam<35||s.cwk<35) return ("F");
          if (s.total<40) return ("E");
          if (s.total<50) return ("D");
          if (s.total<60) return ("C");
          if (s.total<70) return ("B");
          return ("A");
    }
    public static int total(Student s)
    {
    return  (int)Math.round(0.6*s.exam+(1.0-0.6)*s.cwk);
    }
    public String toString()
    {    return name + " " + " " + exam
         + " " + cwk  + " " + total + " " + grade;
    }
}
```

12.9.1 Printing Objects

If we try to print an Object, first Java tries to apply the Object's `toString()` method to convert it to a `String` for printing. Try `System.out.println(new Student("fred",60,45));`

12.9.2 The Raw Data File `marks`

The result of each student has been stored in a text file called `marks` which is a file containing the following information about a student:

1. name
2. exam mark
3. coursework mark.

Each of these fields is on a separate line.
Here is a program called `Results`. It is a program which I used to process the `marks` file. It prints out each student's name, exam mark, coursework mark, total and grade.
`[Lecture13/Results.java]`

```
//Results.java
package Lecture13;
import java.io.*;
import java.util.*;
public class Results
{
    public static void main(String[] args) throws IOException
    {
        String name;
        int exam;
        int cwk;
        Scanner in =new Scanner(new FileReader(args[0]));
        while(in.hasNextLine)
        {
           name =in.nextLine();
           cwk = Integer.parseInt(in.nextLine());
           exam = Integer.parseInt(in.nextLine());
           System.out.println( new Student(name,exam,cwk));
        }
    }
}
```

Each time round the loop we read three `Strings` from the file:

1. The student's name
2. The student's exam mark
3. The student's coursework mark

We then create a `Student` out of these using `new Student(name,exam,cwk)` and print this out.

12.10 Exercises

12.10.1 Students in `Vector`

Write a class `studentsinVector` which has one field called `Students` of type `Vector` (each element of which is a `Student`) and a constructor which reads from a file and then stores all the `Students` in the `Vector`, `Students`. (See page 180 Number 78 for the answer.)

12.10.2 Print Students in `Vector`

Write a program `printstudentsinvector.java` which stores all the students in a `Vector` and prints out the results in the `Vector`. (See page 181 Number 79 for the answer.)

12.10.3 Print Sorted Students in `Vector`

Write a program `printsorrtedstudentsinvector.java` which stores all the students in a `Vector` and prints out the results in the `Vector` but this time sorted in descending order of total mark. (See page 181 Number 80 for the answer.)

12.11 Exercises (no Solutions)

12.11.1 Finding Students whose Name Starts with a Particular Prefix

Write a program `findstudent.java` so that `java findstudent marks Jones` will print out the information for all the students whose name starts with Jones. The file `marks` is where the raw data is kept.

12.11.2 Sorting as you Input

Rewrite `studentsinVector` so that it sorts by ascending order of mark by putting students in the right place in the `Vector` as they are read from the file.

12.12 Summary

Having worked on Chapter 12 you will have:

- Learned the similarities and differences between a `Vector` and an array.
- Seen how `Vectors` are used for storing arbitrarily large amounts of data.
- Learned how to sort `Vectors`.
- Learned how to manipulate the contents of files by storing them in a `Vector` and then manipulating the `Vector`.
- Written a complete system for processing student marks.

Chapter 13

Conclusion

You have now come to the end of the both volumes of the Java Subject Guide. By now you should be familiar with many Java programming concepts.

13.1 Topics

The first volume of the Java Subject Guide considered many of the basic concepts of programming. These included:

- Arithmetic and Boolean Expressions
- Variables and Types, Declarations and Assignments
- Input and Output
- Conditional Statements
- Loops: Simple and Nested
- Useful Built-in Methods
- Arrays
- Defining and Using Methods

and the second volume of the Java Subject Guide introduced:

- Command-line rguments
- Recursion
- Packaging Programs
- More about Variables
- Bits, Types, Characters and Type Casting
- Files and Streams
- Sorting Arrays and Searching
- Defining Your Own Classes
- Inheritance
- Exception Handling
- Vectors

13.2 Complete All the Challenging Problems!

In order to make sure that you are prepared for second year programming, make sure you attempt and complete **all** the challenging problems on page 113 before you start year two of the programme. This is the best way to prepare!

Part II

Appendices

Appendix A

Challenging Problems

We learn to program, not only by reading books or subject guides, but mainly by trying to solve programming problems. This is why I have provided you with some challenging problems. For each problem I will give you some hints as to how I would go about solving it. I hope you find these hints useful, but feel free to solve the problems your own way!

Each challenging problem has two numbers, for example [1,5] associated with it. This means that you need to have studied as far as Volume 1 Chapter 5 before you attempt this problem.

A.1 Try out a Program [1,2]

Here is a program that produces pretty colours: [`LectureElements/pretty.java`]

```
import java.awt.Color;
import java.util.Random;
import element.*;
public class pretty
{
    public static void main(String args[])
    {
        Random r = new Random();
        DrawingWindow d = new DrawingWindow();
        while (true)
        {
         d.setForeground(new Color(r.nextInt(255),r.nextInt(255),r.nextInt(255)));
         Oval o = new Oval(r.nextInt(200),r.nextInt(200),r.nextInt(20),r.nextInt(20));
         d.fill(o);
        }
     }
}
```

Type it in and then compile and run it on you computer. Make sure you set the `CLASSPATH` correctly!

A.2 Rolling a Die [1,5] (dice.class)

Write a program which emulates rolling a die. Every time the program is run, it outputs a random number between 1 and 6.

A.2.1 Hint

This program will have just a simple main method that prints out a random number between 1 and 6. You need to find out how to generate a random number between 1 and 6 and simply print the number out. Look in the Sun Java documentation for the class `java.util.Random`. See if you can find an instance method for generating a random number.

A.3 Leap Years [1,7]

Write a program in which the user enters a year and the program says whether it is a leap year or not.

A.3.1 Hint

Look up the rules for deciding whether a year is a leap year. Try typing *rules for leap year* into Google or some other search engine. Part of the rule will say the year, n, must be divisible by 4. Having found the rules you need to think of a boolean expression involving the year n which is true if n is a leap year and false otherwise. It will be of the form

```
n%4==0 && ...
```

The program will first ask the user to enter an integer. You will store the input in an `int` variable, n. Then you will use the above boolean expression in an `if` statement, to decide whether to print `yes` or `no`.

You do not, at this stage, need to worry about handling illegal input from the user.

A.4 Drawing a Square [1,7]

Using `lineTo` and `moveTo`, from `element.jar`, write a program that asks the user to enter an integer size which draws a square of that size.

A.4.1 Hint

Assuming the square starts at co-ordinate $(origX, origY)$, you need to work out (not very difficult!) the co-ordinates of the three other corners of the square assuming its side has length n. Four calls to `lineTo` is more or less all you need.

A.5 How Old Are You? [1,7] (age.class)

Try out this program:

```
import java.util.Calendar;
class age
```

```
{

 public static void main( String []  args)
 {
   Calendar rightNow = Calendar.getInstance();
   int year =rightNow.get(rightNow.YEAR);
   int month =rightNow.get(rightNow.MONTH);
   int day =rightNow.get(rightNow.DAY_OF_MONTH);
   System.out.println(year);
   System.out.println(month);
   System.out.println(day);
 }

}
```

Write a program which asks the user for their date of birth and then tells them how old they are.

A.5.1 Hint

Having input the user's date of birth, you will have three integers `day`, `month` and `year` from the user and three integers from the system (see above). You then subtract this year from the year entered by the user and then subtract one if the month entered by the user is after this month or the months are the same and the day entered by the user is after today's day. (Careful about how the months are represented!)

A.6 Guessing Game [1,8]

Write a program that tries to guess the number thought of by the user. The number is between 0 and 1000. If the computer's guess is too high, the user should enter 2. If the computer's guess is too low, the user should enter 1. If the computer's guess is correct, the user should enter any integer except 1 or 2. Print out how many guesses it took the computer. Also print out if the user cheated!

A.6.1 Hint

You need a loop. You can use a boolean variable `finished` to get out of the loop. Before you enter the loop set finished to `false`. The loop should look like this:

```
while(!finished)
{


}
```

When the game is over, set `finished` to true. Then the loop will terminate.

Store the lowest and highest possible values. Each time choose half way in between. Use integer division by two to achieve this. Half way in between will be $(highest + lowest)/2$. Depending on whether the user enters 1 or 2 there will either be a new highest or a new lowest. If the computer doesn't guess by chance, eventually the highest and the lowest will become the same value. If this is not the right answer then the user must have cheated!

A.7 Mouse Motion [1,8] (mouseInRect.class)

Write a Java program which displays a small square of a colour of your choice. The left hand corner of the square must have one co-ordinate equal to the day of the month you were born, the other co-ordinate of the left hand corner must be the integer corresponding to the month you were born (Jan=1, Feb=2, etc.). The side of the square must correspond to your age in years. The square must change to a different colour when the mouse is inside it and back to the original colour when the mouse is not inside it. The program must keep responding to the mouse in this way. You should use the element package and the Drawing window described in Chapter 3.

A.7.1 Hint

- Your program should contain an infinite loop, so it goes on for ever.
- You will probably use the following methods from the element package:
  - getMouse()
  - contains
  - setForeground
  - fill
- You will need to work out some boolean expressions to test whether the mouse is inside the square that you have drawn. See the `contains` method for this.

A.8 Maze [1,8] (maze.class)

Write a Java program that represents a maze. The maze must have a start and a finish. The idea is to move the mouse from the start to the finish without going outside the maze. The program should give an error message if the mouse goes off the path of the maze and force the user to start again by ending the program. If the user gets from the start to the finish successfully the program should display to the user how long it took in seconds.

A.8.1 Hint

First make a simple shape for the maze. Mine was like this:

```
DrawingWindow d = new DrawingWindow(500,500);
               Text s =  new Text("start");
               Text f = new Text("end");
               s.center(new Pt(250,400));
               f.center(new Pt(255,200));
               Circle start= new Circle(250,400,30);
               Rect mid1 = new Rect(240,200,10,200);
               Circle finish= new Circle(255,200,30);
               d.fill(start);
```

```
                d.fill(mid1);
                d.fill(finish);
                d.setForeground(Color.white);
                d.draw(s);
                d.draw(f);
```

Then have a loop which gets the position of the mouse and checks where it is. Use the `contains()` method for this. For example, `start.contains(p1)` will be true if and only if point `p1` is inside the start circle. etc. Use `long b=System.currentTimeMillis();` to get the current time.

A.9 Hangman [1,9] (hangman.class)

The computer thinks of a word. (In fact, 'hard-wire' the word into your program.) The user tries to guess the word by trying a letter at a time. If the letter is in the word, then the computer shows the user where it fits. Carry on in this way until either the user runs out of goes (say 9) or the user guesses the word.

A.9.1 Hint

This is an exercise in using the methods in `java.lang.String`. I start off with two Strings, `orig` which is the computer's guess and another one, `user`, which is simply a String of the same length consisting of dashes. `"-------"`. Every time the user has a guess, if the character they input is in `orig` I replace the corresponding dash in `user` by the input letter. The game is over when the two Strings are equal or the user has used up all the goes.

Again, you need a loop. You must read in a character typed by the user. The way I did this was with:

```
in.nextLine().charAt(0)
```

i.e. the first character typed in by the user.

The other methods that I needed were `length` and `compareTo`. To make it easier I defined two methods:

```
 static char getGo()
```

which prompts the user for input and returns the character the user entered, and

```
 static String upDate(String sofar, String orig, char g)
```

which returns the new String for `sofar` assuming the original string is `orig` and the character guessed by the user is `g`. To compute this we loop through `orig` looking for `g`. If we find `g` we update `sofar`.

A.10 Roman Numerals [1,9] (Roman.class)

Romans used a strange way of representing numbers which we call *roman numerals*. In roman numeral notation, M stands for 1000, D for 500, C for 100, L for 50, X for 10, V for 5 and I for one. In order to compute the integer value of a roman numeral, you first look for consecutive characters where the value of the first is less than the second. Take, for example XC and CM. In these cases you subtract the first from the second, so for example XC is 90 and CM is 900. Having done that, you simply add up all the values of such pairs and then to this add the values of the remaining individual roman numerals so, for example, MMMCDXLIX is 3449 and MCMXCIX is 1999.

Write a program which allows the user to enter a roman numeral and then displays its decimal value.

A.10.1 Hint

Write a method `static int value(char a)` which for each single character roman numeral returns its decimal value. This method will use a sequence of `if` statements (or a switch statement if you like).

You can then have another method `static int value(String a)` which returns the value of a complete roman numeral. All you have to do is loop through the `String`, a character at a time. Every time you must look at the next character (if there is one) as well. If the next character is greater than the current one then you must subtract the values and 'jump' two ahead. Otherwise, add the value of the current numeral and jump one ahead.

I do not want you to do any error checking. As long as the program correctly calculates the values of proper roman numerals you will have completed this challenge.

A.11 Shuffling a Pack of Cards (1) [1,10] (deal1.class)

Write a program that shuffles 52 cards randomly. Your program should output the 52 having been shuffled. You may assume that the cards are numbered 1 to 52.

A.11.1 Hint

The way I did it was as follows:
Create an array a of 52 integers. For each i store i at position i in the array. Generate a random number k between 1 and however many cards left in the array (use `java.util.Random`). Print out $a[k]$. Then move all the elements of the array which are to the right of k one to the left. ($a[i] = a[i + 1]$), in effect deleting $a[k]$. Subtract one from the total number of cards left in the array. Repeat this 52 times.

A.12 Shuffling a Pack of Cards (2) [1,10] (deal2.class)

Write a program that shuffles 52 cards randomly. Your program should output the 52 having been shuffled. This time you must output Strings like “five of clubs” or “ace of spades” instead of just a number.

A.12.1 Hint

I used two arrays to store the names values and suits of cards:

```
String [ ] val = {"ace","two","three","four",
                              "five","six","seven","eight","nine", "ten",
                              "jack", "queen","king"};

String [ ] suit ={"clubs","diamonds","hearts","spades"};
```

We then need a way of converting numbers from 1 to 52 into these. To do this I used arithmetic; dividing by thirteen for the suit and taking the remainder for the value.

A.13 Noughts and Crosses (1) [1,11] (tictac.class)

Write a program that allows two people to play noughts and crosses on the computer. Your program should stop illegal moves and detect when the game is over and output who has won.

A.13.1 Hint

I represent the noughts and crosses board as an array of nine integers. I use 0, 1 and 2 to represent empty squares, noughts and Xs respectively. I have written some useful methods including:

```
static void initialise() //initialises the board

static boolean boardFull(int [] b) //returns true iff b is full

static boolean lineOfThree(int [] b,int x, int y, int z) // returns true iff pos
x,y,z are all the same but not empty


static boolean isWon(int [] b) // check whether someone has won

static boolean isFree(int [] b, int x) // checks whether x is a free square in b

static int [ ] userGo(int [] b,int xoro) accepts input from user and returns
updated board

static void drawBoard(int[] b) // draws the board on the screen
```

I'm feeling generous at the moment, so I'll even give you my main method!

```
public static void main(String [] args)
{
     initialise(); int xoro=1;
     for(int i=1;i<10;i++){System.out.print(i); if (i%3==0)System.out.println();  }
     draw(board);
     while(!boardFull(board) && !isWon(board))
     {
        board=userGo(board,xoro);
        if (xoro==1) xoro=2; else xoro=1;
        draw(board);
     }
    if (isWon(board)) System.out.println(xoro==1?"x":"o" + " has won");
    else System.out.println("draw");
}
```

A.14 Mastermind [1,11] (mastermind.class)

Implement the well-known game with coloured pegs. You can use digits instead of colours. The computer's pattern of n digits is hard-wired into the program. The user tries to guess the pattern. The computer responds, telling the user two values:

1. how many digits are right and in the right place (the black pegs)
2. how many digits are right but in the wrong place (the white pegs)

Carry on until the user gets it. Tell the user how many guesses it took.

A.15 Noughts and Crosses (2) [1,11] (tictac2.class)

This time, the computer plays against the user. The user should be allowed to go first and the computer must respond each time with a random legal move.

A.15.1 Hint

The way I have done this is very crude. For the computer's move I repeatedly generate a random number between 1 and 9 until I get a free square. That's the move the computer makes.

A.16 Noughts and Crosses (3) [1,11] (tictac3.class)

This time the computer plays against the user. The user should be allowed to go first and the computer must respond each time with a random legal move. But this time if the computer spots an immediate win it goes for it!

A.16.1 Hint

To help me with this I have written the following ugly method:

```
static boolean winning(int [ ]b, int xoro)
  {
      return (b[0]==xoro && b[0]==b[1] && b[1]==b[2]) ||
             (b[3]==xoro && b[3]==b[4] && b[4]==b[5]) ||
             (b[6]==xoro && b[6]==b[7] && b[7]==b[8]) ||
             (b[0]==xoro && b[0]==b[3] && b[3]==b[6]) ||
             (b[1]==xoro && b[1]==b[4] && b[4]==b[7]) ||
             (b[2]==xoro && b[2]==b[5] && b[5]==b[8]) ||
             (b[0]==xoro && b[0]==b[4] && b[4]==b[8]) ||
             (b[2]==xoro && b[2]==b[4] && b[4]==b[6]);
  }
```

A.17 Nim [1,11] (nim.class)

The computer plays against the user. Implement a winning strategy. In nim we have n piles of matches. The user and the computer take it in turns picking up matches. The rules are that on each go you can pick up as many matches as you like from exactly one pile. The person who is left with no matches is the loser.

A.17.1 Hint

This is quite a hard problem. If you can do this then you must be a super geek! The way I did it relies on two observations:

1. Suppose we have some non-negative integers which when XORed together gives a non-zero value (condition 1). There is a way of subtracting a positive amount from one of the numbers to leave a set of non-negative numbers which when XORed together will give zero.

2. Suppose we start with some non-negative integers which when XORed together gives zero (condition 2). If we subtract a non-negative amount from any one of these numbers to leave a set of non-negative integers, this set of numbers when XORed together will always give a non-zero value.

If we start by offering the user some piles satisfying condition 2 then we can always win because the number of matches is being reduced at each go and when all the piles are zero, they satisfy condition 2.

So, the only problem is to work out how to convert the piles from condition 1 to condition 2. This is one way of doing it:

1. First I XOR all the piles together to produce `xorall`.

2. You then look for a with pile n matches such that when XORed with `xorall`, the result is less than n. Try all piles until you find one satisfying this. That's the computers turn.

A.18 Clock [1,12]

Simplify the following program for animating two clock hands
[LectureElements/bigClock.java]

```
import element.*;
public class bigClock
{
 public static void main(String args[])
 {
  int bigHandSize=30,windowSize=300,origX=100,origY=100,smallHandSize=20;
  double bigAngle=Math.PI*3/2, smallAngle=Math.PI*3/2;
  DrawingWindow d = new DrawingWindow(windowSize,windowSize);
  int hour=0;
  while (true)
  {
   d.paintMode();
   d.moveTo(origX,origY);
   d.lineTo((int)Math.round(origX+smallHandSize*Math.cos(smallAngle)),
            (int)Math.round(origY+smallHandSize*Math.sin(smallAngle)));
   int min=0;
   while (min<60)
   {
    d.paintMode();
    d.moveTo(origX,origY);
    d.lineTo((int)Math.round(origX+bigHandSize*Math.cos(bigAngle)),
             (int)Math.round(origY+bigHandSize*Math.sin(bigAngle)));
    d.awaitMouseClick();
    d.invertMode();
    d.moveTo(origX,origY);
    d.lineTo((int)Math.round(origX+bigHandSize*Math.cos(bigAngle)),
             (int)Math.round(origY+bigHandSize*Math.sin(bigAngle)));
    if (hour*5==min) //redraw the small hand when the big hand goes past
    {
      d.paintMode();
      d.moveTo(origX,origY);
      d.lineTo((int)Math.round(origX+smallHandSize*Math.cos(smallAngle)),
               (int)Math.round(origY+smallHandSize*Math.sin(smallAngle)));
    }
    bigAngle+=Math.PI/30;
    min++;
   }
   d.invertMode();
   d.moveTo(origX,origY);
   d.lineTo((int)Math.round(origX+smallHandSize*Math.cos(smallAngle)),
            (int)Math.round(origY+smallHandSize*Math.sin(smallAngle)));
   smallAngle+=Math.PI/6;
   hour=(hour+1)%12;
  }
 }
}
```

LectureElements/bigClock.java using methods you have defined yourself .

A.19 Spell-Checker [2,7]

1. Find out about the *Soundtex* algorithm.
2. Write a method that implements it.
3. Write a spell-checker that goes through a file and every time it finds a word not in the dictionary it offers the user a list of *similar* words using Soundtex to choose from. It prompts the user either to accept the word or to enter a replacement. New words entered by the user should be stored in a local dictionary.

A.20 Diary Program [2,9]

A diary consists of a number of events. Each event has three associated bits of information.

1. The date when the event takes place.
2. The number of days before they wish to be reminded of the event.
3. The text of the event.

You must write a system that gives the user the following choices:

1. Add an event.
2. See all today's events (you must find today's date). Each event must be displayed one at a time, each time asking the user if they want to:
   (a) Delete the event.
   (b) Change the date of the event.
   (c) Change the number of days before they wish to be reminded of the event.
   (d) Continue.
3. See all today's reminders. Each event must be displayed one at a time, each time asking the user if they want to:
   (a) Delete the event.
   (b) Change the date of the event.
   (c) Change the number of days before they wish to be reminded of the event.
   (d) Continue.
4. See all events on a particular day. Each event must be displayed one at a time, each time asking the user if they want to:
   (a) Delete the event.
   (b) Change the date of the event.
   (c) Change the number of days before they wish to be reminded of the event.
   (d) Continue.
5. See all events on a particular time interval (forget leap years). Each event must be displayed one at a time, each time asking the user if they want to:
   (a) Delete the event.
   (b) Change the date of the Event.

   (c) Change the number of days before they wish to be reminded of the Event.
   (d) Continue.
6. See all Events, each time asking the user if they want to:
   (a) Delete the Event.
   (b) Change the date of the Event.
   (c) Change the number of days before they wish to be reminded of the Event.
   (d) Continue.
7. Update the diary file.
8. Exit the system.

You must design a `Diary` class and an `Entry` class. Your `Diary` class should contain a Vector of Objects of type `Entry`. Your diary should be stored in a file. This file should be read into memory when you start the system and it should be updated when you leave the system. All changes should be done in memory.

Consider the program [`LectureSimpleObjects/diary.java`]

```
import java.io.*;
import java.util.*;
public class diary
{
    static Scanner in = new  Scanner (System.in);

    static void displaymenu()
    {
        System.out.println("Enter 1 to add an event");
        System.out.println("Enter 2 to see all today's events");
        System.out.println("Enter 3 to see all today's reminders");
        System.out.println("Enter 4 to see all events on a particular day");
        System.out.println("Enter 5 to see all events on a particular time interval");
        System.out.println("Enter 6 to see all events");
        System.out.println("Enter 7 to update the diary file");
        System.out.println("Enter 8 to exit the system");
    }

    static void AddAnEvent()
    {
    /* Stub */
    }


    static void seeAllTodaysEvents()
    {
    /* Stub */
    }

    static void seeAllTodaysReminders()
    {
    /* Stub */
    }

    static void seeAllEventsInInterval()
```

```
{
/* Stub */
}
static void seeAllEventsOnParticularDay()
{
/* Stub */
}

static void seeAllEvents()
{
/* Stub */
}

static void updateDiaryFile()
{
/* Stub */
}

static void ExitSystem()
{
/* Stub */System.out.println("bye bye");
}


static boolean goodChoice(int i)
{
    return(i<9 && i>0);
}
static int readUserChoice() throws IOException
{
    int input= Integer.parseInt(in.nextLine());
    boolean continu  = !goodChoice(input);
    while (continu)
    {
        System.out.println("bad input please re-enter");
        input= Integer.parseInt(in.nextLine());
        continu = !goodChoice(input);


    }

     return input;

}

public static void main(String [ ] args) throws IOException
{
Calendar rightNow = Calendar.getInstance();

    System.out.println(rightNow.getTime());

    System.out.println(rightNow.getTime());
```

```
        boolean stop=false;
        while(!stop)
        {
            displaymenu();
            int n=readUserChoice();
            switch (n)
            {
                case 1:{AddAnEvent();break;}
                case 2:{seeAllTodaysEvents();break;}
                case 3:{seeAllTodaysReminders();break;}
                case 7:{updateDiaryFile();break;}
                case 8:{ExitSystem();stop=true;}
            }
        }
    }

}
```

Add the extra options to the diary user interface.

A.20.1 Hints

You need three classes:

- A Date class with three fields:
  - int day
  - int month
  - int year
- An Event class with 3 fields:
  - String text
  - Date date
  - int reminder
- A Diary class with 2 fields:
  - Person owner
  - Vector events

A.20.2 Methods needed for Date Class

- `public boolean Equals(Date d)` returns true if and only if this date is same as d.date like this:

  ```
  public boolean Equals(Date d)
  {
   return (d.day==day && d.month==month && d.year==year);
  }
  ```
- `public boolean Before(Date d)` returns true if and only if this date is before `d`.
- `public boolean After(Date d)` returns true if and only if this date is after `d`.

- `public boolean withinRange(int n,Date d)` returns true if and only if this date is less than n days before `d`.
- `public static Date read()` prompts user for date and returns whatever user enters.
- `public String toString()` converts `Date` to `String`.

A.20.3 Methods needed for Event Class

- `public boolean Equals(Date d)` returns `true` if and only if the date of this Event is same as `d.date` like this:

  ```
  public boolean Equals(Date d)
  {
   return (date.equals(d));
  }
  ```
- `public boolean Before(Date d)` returns `true` if and only if this event's date is before `d`.
- `public boolean After(Date d)` returns `true` if and only if this event's date is after `d`.
- `public boolean Before(Date d)` returns `true` if and only if this event's date is before `d`.
- `public boolean withinRange(int n,Date d)` returns `true` if and only if this event's date is less than `n` days before Date `d`.
- `public static Event read()` prompts user for Event and returns whatever user enters.
- `public String toString()` converts Event to String.

A.20.4 Methods needed for Diary Class

- `public Diary addEvent(Event e)`
  like this:

  ```
  public Diary addEvent(Event e)
  {
   events.addElement(e);
   return new Diary(events,owner);
  }
  ```

Appendix B

Exams

Important: the information and advice given in the following sections are based on the examination structure used at the time this guide was written. However, the University can alter the format, style and requirements of an examination paper without notice. Because of this, we strongly advise that you check the instructions on the paper you actually sit.

B.1 Exam Guidelines

B.1.1 Useful Information

Make sure that you have understood the rubric. Take your time deciding which questions to attempt. The exam lasts for three hours, so you have forty-five minutes per question. Notice that each question is broken into three sections, worth 9, 8 and 8 marks respectively. These are in increasing order of difficulty. The first two sections of each question are usually either book work or an exercise very similar to those in the subject guides. This should be an extra incentive for you to attempt all the exercises in the subject guides.

The third section of each question usually allows you to demonstrate your programming skill.

Do NOT answer more than four questions.

B.1.2 Java 1.5 Changes

The exam paper below was written before the introduction of Java 1.5. Future exams will have the same format as the one below. We now, however, use Java 1.5. The main difference is that the class `java.util.Scanner` is now usually used, as in these guides, instead of the older `java.io.BufferedReader`.

B.1.3 Time Allowed

This exam lasts three hours.

B.2 The Exam

There are six questions in this paper. You should answer no more than FOUR questions. Full marks will be awarded for complete answers to a total of FOUR questions. Each question carries 25 marks. The marks for each part of a question are indicated at the end of the part in [.] brackets.

There are 100 marks available on this paper.

No calculators should be used.

QUESTION 1

1. (a) Write three assignment statements whose effect is to swap the contents of variables x and y.

   (b) Briefly explain why the following program has a compilation error:

```
class f
{
   int x=false;
}
```

   (c) What is the output of the following Java program?

```
class f
{
   public static void main(String [ ]  args)
   {
    System.out.println(7+2*3);
   }
}
```

   Justify your answer.

[9 Marks]

2. (a) What is the output of the following?

```
class H
{
   public static void main(String [ ]  args)
   {
        i=0;
        while (i<5) {System.out.println(i);i=i+1;}
   }
}
```

   (b) Rewrite program H above using a *for* loop.

[8 Marks]

3. Write a method which sorts an array of `ints` into ascending order and discuss the time complexity of your method.

[8 Marks]

QUESTION 2

1. (a) Given that the ASCII code for the character `b` is 98, what is the ASCII code for the character `a`?

   (b) What is the output of the following Java program?

```
class ascii
{
        public static void main(String[] args)
        {
           System.out.print((int) 'a');
        }
}
```

   (c) What is the output of the following Java program?

```
class ascii
{
        public static void main(String[] args)
        {
           System.out.print((char) 98);
        }
}
```

   (d) What does the following program do?

```
import java.io.*;
class one
{
        public static void main(String args[]) throws Exception
        {
                FileReader f =  new FileReader("aaa");
                int x; x=f.read(); x=f.read();
                System.out.print((char)x);
        }
}
```

[9 Marks]

2. Write two fragments of code that illustrate how to output the contents of a file
   (a) by reading it character by character.
   (b) by reading it line by line.

[8 Marks]

3. Write a method which prints out the number of occurrences of each letter in a file called "aaa". You should distinguish between upper and lower case letters.

[8 Marks]

QUESTION 3

1. Define a class called *Date* which has three fields *day*, *month* and *year* all of type *int* and one constructor with three *int* parameters.

   [9 Marks]

2. Write two instance methods, *isEqual* and *isBefore* for the class *Date*. Both methods should have one parameter d of type *Date*. The method *isEqual* should return *true* if this date is equal to d and *false* otherwise. The method *isBefore* should return *true* if this date is before d and *false* otherwise.

   [8 Marks]

3. Define a class called *Person* consisting of a name which is of type *String* and a date of birth which is of type *Date*. Write a constructor with two parameters (one of type *String* and the other of type *Date*) for *Person* and an instance method *isYounger*, with one parameter p of type *Person*, for checking whether this person is younger than p. (You should use the *isBefore* method of the last question.)

   [8 Marks]

QUESTION 4

1. (a) Give a boolean expression which evaluates to `true` if the variable `x` has the value 3 and which evaluates to `false` otherwise.

   (b) Give a boolean expression which evaluates to `true` if the variable `x` has the value 7 or the value 9 and which evaluates to `false` otherwise.

   (c) Give a boolean expression which evaluates to `true` if the variable `x` has the value 1 and the variable `y` has the value 2 and the variable `z` is even and which evaluates to `false` otherwise.

   (d) Give a boolean expression which evaluates to `true` if the variables `x`, `y` and `z` all have the same value and which evaluates to `false` otherwise.

[9 Marks]

2. (a) Here is the body of a method:

```
{
     return 3;
}
```

   What is its return type?

   (b) Here is the body of a method:

```
{
        int x;
        if (x==k) return "hello"+" world";
        else return "";
}
```

   What is its return type?

   (c) Here is the body of a method:

```
{
        System.out.println(3);
}
```

   What is its return type?

   (d) Here is the body of a method:

```
{
        return("hello".charAt(2));
}
```

   What is its return type?

[8 Marks]

3. Write a method whose heading is `static Vector convert(Object [ ] a)` which given an array of Objects, returns a Vector of the same Objects in the same order.

[8 Marks]

QUESTION 5

1. (a) Explain briefly the purpose of packages in Java.
   (b) Explain briefly the purpose of the `import` statement in Java.
   (c) Rewrite the following Java class so it does not have any `import` statements.

```
import java.io.*;
class Echo
{
    public static void main(String[] args) throws IOException
    {
      BufferedReader in =
      BufferedReader(new InputStreamReader(System.in));
      String s =in.readLine();
      System.out.println(s);
    }
}
```

[9 Marks]

2. Given the class C defined as follows

```
class C
{
        int f()
        {
            return 5;
        }
}
```

Define a class D which extends C and which has one method which *overrides* f and another which *overloads* f.

[8 Marks]

3. (a) The class *Object* has one constructor with no parameters. How would you declare a variable, v, of type *Object* and then assign a value to variable, v, by using the constructor of the class *Object*?
   (b) Define a class called *Array* with one field of type *array of Object* and one instance method *toString()* which returns a String containing the elements of the array separated by commas. Do not define a constructor for the class *Array*.

[8 Marks]

QUESTION 6

1. (a) What is the output of the following program?

```
public class A
{
   public static void main(String[] args)
   {
    try
     {
        Integer.parseInt("rabbit");
        System.out.println("cat");
     }
     catch(Exception e)
     {
        System.out.println("fish");
     }
   }
}
```

   (b) There will be an error when we compile the program below. What is it? Give two different ways of correcting it.

```
import java.io.*;
public class cat1
{
        public static void main(String[] args)
        {
            FileReader f =new FileReader("words");
        }
}
```

[9 Marks]

2. (a) What is the output of the following program?

```
public class A
{
   int f(int n)
   {
     if (n==0) return 1;
     else return n*f(n-1);
   }

   public static void main(String[] args)
   {
    System.out.println(f(3));
   }
}
```

   (b) Write a recursive method `static int fibonacci(int n)` which returns the nth fibonacci number.

[8 Marks]

3. Using exceptions, write a method `static boolean find(String f)` which returns *true* if the file whose name is *f* is found and *false* if it is not found.

[8 Marks]

Appendix C

Previous Exam Questions (With Answers)

QUESTION 1

1. (a) What is the purpose of the *main method* in a Java application? (See page 182 Number 81 for the answer.)

   (b) When we compile the program below for the first time, what is the name of the new file that will appear in the current directory?

```
//Our First Program
class HelloWorld
{
        public static void main(String[] args)
        {
          System.out.println("Hello World");
        }
}
```

   (See page 182 Number 82 for the answer.)

[4 Marks]

2. (a) Briefly describe what methods are and why they are useful in programming. (See page 182 Number 83 for the answer.)

   (b) What is a recursive method? Give an example of a recursive method. (See page 182 Number 84 for the answer.)

[6 Marks]

QUESTION 2

1. Briefly explain the purpose of comments in a Java program and briefly describe two different ways of writing comments. (See page 182 Number 85 for the answer.)

[4 Marks]

2. (a) Describe the syntax of a `for` loop in Java. (See page 182 Number 86 for the answer.)
   (b) Give an example of a `for` loop that never terminates. (See page 182 Number 87 for the answer.)

[6 Marks]

QUESTION 3

1. Briefly explain the purpose of *Exceptions* in a Java program. (See page 182 Number 88 for the answer.)

   [4 Marks]

2. (a) List three *primitive types* in Java (See page 183 Number 89 for the answer.)
   (b) Give an example of a *variable declaration* and explain what it does. (See page 183 Number 90 for the answer.)
   (c) Give an example of an *assignment statement* and explain what it does. (See page 183 Number 91 for the answer.)

   [6 Marks]

QUESTION 4

1. (a) What are the possible values that *boolean expression* evaluate to. (See page 183 Number 92 for the answer.)

   (b) List two boolean operators in Java and, with examples, briefly explain their meaning. (See page 183 Number 93 for the answer.)

[4 Marks]

2. (a) Describe the similarities and differences between *arrays* and *Vectors* in Java. (See page 183 Number 94 for the answer.)

   (b) How do you refer to the third element of array `x`? (Reminder: The third element of an array has exactly two elements before it) (See page 183 Number 95 for the answer.)

   (c) How do you refer to the third element of `Vector v`? (Reminder: The third element of a `Vector` has exactly two elements before it) (See page 183 Number 96 for the answer.)

[6 Marks]

QUESTION 5

1. Give an example of a *compilation error* and explain how the compiler informs us of the error. (See page 183 Number 97 for the answer.)

[4 Marks]

2. (a) Write a complete Java program that tests whether *times* binds more tightly than *plus*. Explain how the output of your program will enable you to decide. (See page 183 Number 98 for the answer.)

   (b) Explain why it is not necessary to remember the precedence of operators when writing programs. (See page 183 Number 99 for the answer.)

[6 Marks]

QUESTION 6

1. (a) What does it mean to say that Java is *case sensitive*? (See page 183 Number 100 for the answer.)

   (b) Give an example of a Java program with an error caused by the fact that Java is case sensitive. (See page 183 Number 101 for the answer.)

[4 Marks]

2. (a) Consider the two programs below. Explain what each of their outputs is and why.

```
class div1
{
    public static void main(String[] args)
    {
      System.out.println(3/2);//int divided by int
    }
}
```

```
class div2
{
    public static void main(String[] args)
    {
      System.out.println(3/2.0);//int divided by real
    }
}
```

   (See page 183 Number 102 for the answer.)

   (b) Explain the output of the following program:

```
class abs
{
 public static void main(String[] args)
 {
   System.out.println(Math.abs(-5));
   System.out.println(Math.abs(5));
 }
}
```

   (See page 183 Number 103 for the answer.)

[6 Marks]

QUESTION 7

1. Write a program that prints out every other character of a file. i.e The first, third, fifth etc. character. The name of the file should be passed to the program as a command line argument. (See page 184 Number 104 for the answer.)

[7 Marks]

QUESTION 8

Consider the class Person

```
package LectureSimpleObjects;
public class Person
{
    public String firstname;
    public String lastname;
    public Date dob;
    public boolean sex; //true= female, false = male

    public Person(String f, String l, Date birth , boolean s)
    {
      firstname=f;
      lastname=l;
      dob=birth;
      sex=s;
     }
    public Person(String f, String l, Date birth, String s)
    {
      firstname=f;
      lastname=l;
      dob=birth;
      if (s.charAt(0)=='f' || s.charAt(0)=='F') sex=true;
      else sex=false;
    }
}
```

Extend the class Person, to a class NatPerson so that a person also has a nationality which is a String. We require two new constructors corresponding to each of the two constructors of Person. (See page 184 Number 105 for the answer.)

[7 Marks]

QUESTION 9

1. What is the output of the program below and why?

```
class test1
{
    public static void main(String[ ] args)
    {
        int[ ] a = new int[1];
        a[0]=5;
        int [ ] b =a;
        a[0]=6;
        System.out.println(b[0]);
    }
}
```

   (See page 184 Number 106 for the answer.)

2. What is the output of the program below and why?

```
class p1
{
    static void f()
    {
      int x=3;
    }

     public static void main(String [ ] args)
    { int x;
      x=2;
      f();
      System.out.println(x);
    }
}
```

   (See page 184 Number 107 for the answer.)

[7 Marks]

QUESTION 10

1. Given the class Arrays:

```
package LectureNonVoidMethods;
import java.io.*;
public class Arrays
{
    public static int sum(int [] a)
    {
       int sum=0;
       for (int i=0;i<a.length;i++)sum=sum+a[i];
       return sum;
    }
    public static int largest(int [] a)
    {
       int largest=a[0];
       for (int i=1;i<a.length;i++) if (a[i]>largest) largest=a[i];
       return largest;
    }
    public static int smallest(int [] a)
    {
       int smallest=a[0];
       for (int i=1;i<a.length;i++) if (a[i]<smallest) smallest=a[i];
       return smallest;
    }
    public static double average(int [] a)
    {
      return (sum(a)*1.0)/a.length;
    }
    public static void print(int a[])
    {
      for (int i=0;i<a.length;i++) System.out.println(a[i]);
    }
    public static int [] readintarray() throws IOException
    {
       BufferedReader in =new BufferedReader(new InputStreamReader(System.in));
       System.out.println("How many numbers are you going to enter? ");
       String s =in.readLine();
       int n = Integer.parseInt(s);
       System.out.println("Enter "+n+ " numbers ");
       int[] a; //declares a to be an array of ints
       a =new int[n];  //gives a enough space to hold n ints
       for(int i=0;i<n;i++)  /*read them into an array*/
       {
         s =in.readLine();
         a[i]=Integer.parseInt(s);
       }
        return a;
     }
}
```

The methods in class Arrays can be referred to in other classes. Write a complete Java application which calls some of the methods in the class Arrays which asks the user how many numbers they are going to enter, reads in the numbers and prints their average.

[7 Marks]

(See page 184 Number 108 for the answer.)

QUESTION 11

1. Describe the *Bubble Sort* Algorithm.

   [2 Marks]

2. Write a method whose heading is

   ```
   public static void ascendingBubbleSort(int[] a, int size)
   ```

   which sorts an array of `ints` into ascending order. (See page 185 Number 109 for the answer.)

   [5 Marks]

QUESTION 12

Using `try` and `catch` and the method `Integer.parseInt`, write a static method, `isInt` that takes a `String` as a parameter and returns `true` if the `String` contains only digits and `false` otherwise. (See page 185 Number 110 for the answer.)

[7 Marks]

QUESTION 13

Write a simple spell checker. You can assume the existence of a file called `words` in the current directory consisting of a big list of English words (courtesy of Unix), one per line. The user simply types the beginning of the word he wants to check as a command line argument. For example,

```
java spell freq
```

will print out all English words starting with `freq`

```
frequencies
frequency
frequent
frequented
frequenter
frequenters
frequenting
frequently
frequents
```

You may also use the `String` method `startsWith`. (See page 185 Number 111 for the answer.)

[8 Marks]

QUESTION 14

Write a complete program that asks the user to enter two numbers and then prints out the Greatest Common Divisor of the two numbers. Your solution may be iterative or recursive. (See page 185 Number 112 for the answer.)

[8 Marks]

QUESTION 15

1. Define a class `house` that consists of three fields:
   (a) an `int` which gives its number
   (b) another `int` which gives the number of rooms in the house
   (c) an array of `Persons` living in the house.

   Your definition should include a constructor with three parameters.
2. Define another class called `Street` consisting of two fields
   (a) A `String` representing the name of the street.
   (b) An array of `Houses` in the street.

   Your definition of `Street` should contain a constructor with two parameters and an instance method called `UpMarket` which returns true if every house in the street contains more rooms than people.

(See page 186 Number 113 for the answer.)

[8 Marks]

QUESTION 16

A *palindrome* is a `String` that reads the same backwards as forwards. Write a complete Java program that asks the user to enter a `String`. The program outputs 'Yes' if the `String` entered by the user is a palindrome and 'no' otherwise. The program repeatedly asks the user if he wants another go to enter further `Strings`. (See page 186 Number 114 for the answer.)

[8 Marks]

QUESTION 17

Consider the following program:

```
import java.io.*;
public class dog
{
        public static void cat(String s) throws Exception
        {
                BufferedReader inone =new BufferedReader(new FileReader(s));
                int t=inone.read();
                while (t!=-1)
                {       System.out.print((char)t);
                        t=inone.read();
                }
        }
        public static void main(String[] args) throws Exception
        {cat(args[0]);}
}
```

1. What happens if we type in `java dog dog.java`? (See page 187 Number 115 for the answer.)

   [2 Marks]

2. When does the variable `t` get the value -1? (See page 187 Number 116 for the answer.)

   [2 Marks]

3. What would happen if the input file was empty? Why? (See page 187 Number 117 for the answer.)

   [2 Marks]

4. What is the value of the expression, `"hello".charAt(0)` ? (See page 187 Number 118 for the answer.)

   [2 Marks]

5. Let `s` be a `String`. Write an expression which yields the last character in `s`. (See page 187 Number 119 for the answer.)

   [4 Marks]

6. Write a complete Java program called `swapab` that writes the contents of a file to standard output swapping all 'a's for 'b's and all 'b's for 'a's. For example to run the program on a file called `fred`, we type: `java swapab fred`. (See page 187 Number 120 for the answer.)

   [13 Marks]

QUESTION 18

1. Briefly describe the class `Vector` and how it compares with an array.

   [4 Marks]

   (See page 187 Number 121 for the answer.)
2. Assuming v is an object of class `Vector`, what is wrong with the boolean expression v.elementAt(0)==1? (See page 188 Number 122 for the answer.)

   [2 Marks]
3. Briefly explain the need for the Java classes such as Integer, Character, and Boolean.

   [3 Marks]

   (See page 188 Number 123 for the answer.)
4. Assuming that the zeroth and first elements of the `Vector` v are Integers, write a statement whose effect is to add a new element to v. The value of this new element is the sum of the first two elements of v. Fully explain your answer.

   [8 Marks]

   (See page 188 Number 124 for the answer.)
5. Write a method whose heading is
   `static double average(Vector v)`
   which returns the average of all elements in a non-empty `Vector` of Integers.

   [8 Marks]

   (See page 188 Number 125 for the answer.)

QUESTION 19

1. What is a recursive method?

   [2 Marks]

   (See page 188 Number 126 for the answer.)
2. Write a method `int power(int n, int m)`, which implements exponentiation (n to the power m) recursively in terms of multiplication (Assume $m \geq 0$).

   [5 Marks]

   (See page 188 Number 127 for the answer.)
3. Write a method `int power(int n, int m)`, which implements exponentiation iteratively. (See page 188 Number 128 for the answer.)

   [5 Marks]

Appendix D

Multiple Choice Questions

Consider the following program and then answer the questions that occur on the next pages.

```
class HorizLine
{
   int length;

   public HorizLine(int l)
   {length=l;}

   public void Draw()
   {for (int i=0;i<length;i++)System.out.print("*");
   }

   public void Drawln()
   {Draw();
    System.out.println("");
   }
}


class BetterLine extends HorizLine
{
  char endchar;
  char middlechar;

  public BetterLine(char end,char middle, int len)
  {
   super(len);
   endchar=end;
   middlechar=middle;
  }

  public void Draw()
  {for (int i=0;i<length;i++)
   if (i==0 || i == length-1)  System.out.print(endchar);
   else System.out.print(middlechar);
  }
}
```

QUESTION 20

Consider the program on page 156. How many instance variables does the class `BetterLine` have? (including the instance variables it inherits)

(a) 0

(b) 1

(c) 2

(d) 3

(e) 4

(See page 188 Number 129 for the answer.)

QUESTION 21

Consider the program on page 156.

How would an instance of `HorizLine` be declared?

(a) `b HorizLine;`

(b) `HorizLine b;`

(c) `HorizLine(b);`

(d) `HorizLine();`

(e) None of the above.

(See page 188 Number 130 for the answer.)

QUESTION 22

Consider the program on page 156.

How would an instance of `HorizLine` be created?

(a) `b=new HorizLine;`

(b) `new HorizLine(b);`

(c) `b=new HorizLine(5)`

(d) `b=HorizLine(i);`

(e) None of the above.

(See page 188 Number 131 for the answer.)

QUESTION 23

Consider the program on page 156.

How many of the statements below are true?

- `HorizLine` is a constructor
- `Draw` is a constructor
- `Drawln` is a constructor
- `BetterLine` is a constructor

(a) 0
(b) 1
(c) 2
(d) 3
(e) 4

(See page 188 Number 132 for the answer.)

QUESTION 24

Consider the program on page 156.

Which one of the statements below is true?

(a) The method `Draw` in class `BetterLine` is an example of *method overriding*
(b) The method `Draw` in class `BetterLine` is an example of *method overloading*
(c) The method `Draw` in class `BetterLine` is an example of *method overcasting*
(d) The method `Draw` in class `BetterLine` has two parameters
(e) None of the above.

(See page 188 Number 133 for the answer.)

Appendix E

Answers to Exercises

1. [LectureCommandLine/AddOneNew.java]

   ```
   class AddOneNew
   {
      public static void main(String[] args)
      {
        int x=Integer.parseInt(args[0]);
        System.out.println(x+1);
      }
   }
   ```

2. [LectureCommandLine/AddNew.java]

   ```
   class AddNew
   {
      public static void main(String[] args)
      {
        int x=Integer.parseInt(args[0]);
        int y=Integer.parseInt(args[1]);
        System.out.println(x+y);
      }
   }
   ```

3. [LectureCommandLine/BackwardsNew.java]

   ```
   class BackwardsNew
   {
      public static void main(String[] args)
      {
        for (int i = args.length-1;i>=0;i--)
        System.out.print(args[i]+" ");
        System.out.println();
      }
   }
   ```

4. [LectureCommandLine/AddAllNew.java]

   ```
   class AddAllNew
   {
      public static void main(String[] args)
      {
        int total=0;
        for (int i = 0;i< args.length;i++)
        total=total+Integer.parseInt(args[i]);
        System.out.println(total);
      }
   }
   ```

5. [LectureCommandLine/AddAllRealNew.java]

```
class AddAllRealNew
{
   public static void main(String[] args)
   {
     double total=0.0;
     for (int i = 0;i< args.length;i++)
     total=total+Double.parseDouble(args[i]);
     System.out.println(total);
   }
}
```

6. [Lecture16/answers/fibonacciNew.java]

```
public class fibonacciNew
{
    static    int fib(int n)
    {
        if (n==0||n==1) return 1;
        else return fib(n-1)+fib(n-2);
    }
    public static void main(String[] args)
    {
        System.out.print(fib(Integer.parseInt(args[0])));
    }
}
```

7. [Lecture16/answers/multNew.java]

```
public class multNew
{
    static int     mult(int n,int m)
    {
        if (m==0) return 0;
        else return n+mult(n,m-1);
    }
    public static void main(String[] args)
    {
        System.out.print( mult(Integer.parseInt(args[0]),
                               Integer.parseInt(args[1]))
                        );
    }
}
```

8. [Lecture16/answers/powerNew.java]

```
public class powerNew
{
    static int     power(int n,int m)
    {
        if (m==0) return 1;
        else return n*power(n,m-1);
    }
    public static void main(String[] args)
    {
        System.out.print( power(Integer.parseInt(args[0]),
                                Integer.parseInt(args[1]))
                        );
    }
}
```

9. [Lecture16/answers/reverseNew.java]

```
import java.io.*;
class reverseNew
{
    static BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

    static void rev() throws Exception
    {
        int t=in.read();
        if (t=='\n') return;
        rev();
    System.out.print((char)t);
    }

    public static void main(String args[]) throws Exception
    {
        rev();
    }
}
```

10. [Lecture16/answers/syrNew.java]

```
public class syrNew
{
    static void syr(int n)
    {       System.out.println(n);
        if (n==1) return;
        if (n%2==0) syr(n/2);
                else syr(3*n+1) ;
    }
    public static void main(String[] args)
    {
              syr(Integer.parseInt(args[0])) ;
    }
}
```

11. [Lecture2/EchoNoImportNew.java]

```
class EchoNoImportNew
{
    public static void main(String[] args)
    {
      java.util.Scanner in =new java.util.Scanner(System.in);
      String s =in.nextLine();
      System.out.println(s);
    }
}
```

12. [LectureTest/Test2.java]

```
import LectureNonVoidMethods.*;
class Test2
{
 public static void main(String [ ] args)
 {
  int [ ] a;
  a = ArraysNew.readintarray();
  System.out.println("Their average is " + ArraysNew.average(a));
 }
}
```

13. 2 – because the two `xs` are different.
14. The output is 6 because `a` and `b` both point at the same piece of RAM.
15. The output is 5 because `a` and `b` are different integers. Changing `a` can't affect `b`
16. In the call to method `p` the address of `a` is stored in parameter `m`. So the assignment `m[0]=5;` will change the value of the array `a`. Therefore 5 will be printed.
17. In the call to method `p` the first address of `a` is stored in parameter `m`. Then the assignment `m=new int[3];` will change the value of `m` to be a new array and so the assignment to `m` will not affect the array `a` and therefore 5 will be printed.
18. In the call to method `p` the first address of `n` is stored in parameter `m`. Then the assignment `m= "goodbye";` will change the value of `m` to be a new `String` and so the assignment to `m` will not affect the `String` `n` and therefore `hello` will be printed.
19. 17
    2
20. This prints out 5.
21. [LectureChar/TwoToTheSixteen.java]

```
public class TwoToTheSixteen
{
     public static void main(String [ ] args)
        {
           int i=1;
      for (int j=0;j<16;j++) i=2*i;
      System.out.println(i);
        }
}
```

22. [LectureChar/IntToChar.java]

```
public class IntToChar
{
        static int TwoTo16()
        { int tot=1;
          for (int i=1;i<=16;i++) tot=tot*2;
          return tot;
        }
    public static void main(String [ ] args)
        {
           int max = TwoTo16();
           for (int i=0;i<max;i++) System.out.println(i + " " + (char)i);
        }
}
```

23. $2x + 1 = 195$
    So $x = 97$
    So the Unicode value of `a` is 97 and that of `b` is 98.

24. The answer is 65535 which you should now know is $2^{16} - 1$.
25. [LectureChar/IntToBoolean.java]

```
public class IntToBoolean
{

    public static void main(String [ ] args)
        {
           System.out.println((boolean)0);
        }
}
```

    The answer is no. If we run this program we get:

```
IntToBoolean.java:7: Invalid cast from int to boolean.
           System.out.println((boolean)0);
                                   ^
1 error
```

26. [LectureChar/BooleanToInt.java]

```
public class BooleanToInt
{
    public static void main(String [ ] args)
        {
           System.out.println((int)true);
        }
}
```

    The answer is no. If we run this program we get:

```
BooleanToInt.java:7: Invalid cast from boolean to int.
           System.out.println((int)true);
                                   ^
1 error
```

27. [LectureChar/FloatToInt.java]

```
public class FloatToInt
{
    public static void main(String [ ] args)
        {
           System.out.println((int)13425.79);
        }
}
```

    The answer is yes. If we run this program we get: 13425. How do you think it is done?

28. [LectureChar/IntToFloat.java]

```
public class IntToFloat
{
    public static void main(String [ ] args)
        {
           System.out.println((float)1);
        }
}
```

The answer is yes. If we run this program we get: 1.0 How do you think it is done?

29. `[LectureChar/DoubleToFloat.java]`

```
public class DoubleToFloat
{
    public static void main(String [ ] args)
        {
           System.out.println((float)1.7976931348623);
        }
}
```

The answer is yes. If we run this program we get:
1.7976931
How do you think it is done?

30. `[LectureChar/FloatToChar.java]`

```
public class FloatToChar
{
    public static void main(String [ ] args)
        {
           System.out.println((char)75.79);
        }
}
```

The answer is yes. If we run this program we get:
`K`
Why k? How do you think it is done?

31. `[LectureChar/IntToShort.java]`

```
public class IntToShort
{
    public static void main(String [ ] args)
        {
           System.out.println((short)32769);
        }
}
```

The answer is yes. If we run this program we get:
-32767
How do you think it is done?

32. 65536
What is special about this number? What is the next biggest?

33. It prints out:
2147483647
-2147483648

34. We get the Unicode value of each character in `fff.dat` printed out:

```
10912132110971091011010511532115101989711511610597110461073321081051181013210
511010761111101001111104610
```

35. `[Lecture11/swapab.java]`

```
import java.io.*;
public class swapab
{
  public static void main(String[] args) throws Exception
  {
     FileReader inone = new FileReader(args[0]);
     int t=inone.read();
     while (t!=-1)
     {
       if (t=='a')System.out.print('b');
       else if (t=='b')System.out.print('a');
            else System.out.print((char)t);
       t=inone.read();
     }
  }
}
```

36. [Lecture11/ThirdLinefff.java]

```
import java.io.*;
import java.util.Scanner;
public class ThirdLinefff // This program prints out the third line of fff.dat
{
   public static void main(String[] args) throws Exception
   {
     Scanner in =new Scanner(new FileReader("fff.dat"));
     in.nextLine();
     in.nextLine();
     System.out.println(in.nextLine());
   }
}
```

37. [Lecture11/HundredthLineggg.java]

```
import java.util.Scanner;
import java.io.*;
public class HundredthLineggg
// This program prints out the hundredth line of ggg.dat
{
   public static void main(String[] args) throws Exception
   {
     Scanner in =new Scanner(new FileReader("ggg.dat"));
     for (int i=0;i<99;i++)in.nextLine();

     System.out.println(in.nextLine());
   }
}
```

38. [Lecture11/OddLinesggg.java]

```
import java.util.Scanner;
import java.io.*;
public class OddLinesggg
// This program prints out the Odd lines of ggg.dat
{
   public static void main(String[] args) throws Exception
   {
     Scanner in =new Scanner(new FileReader("ggg.dat"));

     while (in.hasNextLine())
     {
      System.out.println(in.nextLine());
      if (in.hasNextLine()) in.nextLine();
     }
   }
}
```

39. [Lecture11/EvenLinesggg.java]

```
import java.util.Scanner;
import java.io.*;
public class EvenLinesggg
// This program prints out the even lines of ggg.dat
{
   public static void main(String[] args) throws Exception
   {
     Scanner in =new Scanner(new FileReader("ggg.dat"));

     while (in.hasNextLine())
     {
      in.nextLine();
      if (in.hasNextLine()) System.out.println(in.nextLine());
     }
   }
}
```

40. [Lecture11/catChoose.java]

```
import java.util.Scanner;
import java.io.*;
public class catChoose
{
   public static void main(String[] args) throws Exception
   {
        Scanner in1 =new Scanner(System.in);
        System.out.print("Enter the name of the file you want to display: ");
        String s= in1.nextLine(); //read file name from keyboard
        Scanner in = new Scanner(new FileReader(s));
        while (in.hasNextLine()) System.out.println(in.nextLine());
   }
}
```

Notice that we have two inputs `in1`, representing input from the keyboard, and `in`, representing input from the file the user chooses. There is nothing special about the names `in1` and `in`. They are just any old variable names. This works just as well!: `[Lecture11/catChooseSilly.java]`

```
import java.util.Scanner;
import java.io.*;
public class catChooseSilly
{
   public static void main(String[] args) throws Exception
   {
        Scanner elephant =new Scanner(System.in);
        System.out.print("Enter the name of the file you want to display: ");
        String s= elephant.nextLine(); //read file name from keyboard
        Scanner monkey = new Scanner(new FileReader(s));
        while (monkey.hasNextLine()) System.out.println(monkey.nextLine());
   }
}
```

41. [Lecture11/catCommand.java]

```
import java.io.*;
import java.util.Scanner;
public class catCommand
{
   public static void main(String[] args) throws Exception
   {
        Scanner in = new Scanner(new FileReader(args[0]));
        while(in.hasNextLine()) System.out.println(in.nextLine());
   }
}
```

42. [Lecture11/ThirdCharfff.java]

```
import java.io.*;
public class ThirdCharfff // This program prints out the third line of fff.dat
{
   public static void main(String[] args) throws Exception
   {
     FileReader in = new FileReader("fff.dat");
     in.read();
     in.read();
     System.out.println((char)in.read());
   }
}
```

43. [Lecture11/HundredthCharggg.java]

```
import java.io.*;
public class HundredthCharggg
{
   public static void main(String[] args) throws Exception
   {
     FileReader in =new FileReader("ggg.dat");
     for (int i=0;i<99;i++)in.read();
     System.out.println((char)in.read());
   }
}
```

44. [Lecture11/OddCharsggg.java]

```
import java.io.*;
public class OddCharsggg
{
   public static void main(String[] args) throws Exception
   {
     FileReader in =new FileReader("ggg.dat");
     int s=in.read();
     while (s!=-1)
     {
      System.out.print((char)s);
      in.read();
      s=in.read();
     }
   }
}
```

45. [Lecture11/EvenCharsggg.java]

```
import java.io.*;
public class EvenCharsggg
{
   public static void main(String[] args) throws Exception
   {
     FileReader in =new FileReader("ggg.dat");
     in.read();
     int s=in.read();
     while (s!=-1)
     {
      System.out.print((char)s);
      in.read();
      s=in.read();
     }
   }
}
```

46. [Lecture11/swapchars.java]

```
import java.io.*;
public class swapchars
{
        public static void main(String[] args) throws Exception
        {
       String w=args[1];
           FileReader inone =new FileReader(args[0]);
           int t=inone.read();
           while (t!=-1)
           {
              if (t == w.charAt(0))System.out.print(w.charAt(1));
              else if (t==w.charAt(1))System.out.print(w.charAt(0));
                   else System.out.print((char)t);
              t=inone.read();
           }
        }
}
```

47. [Lecture11/manycat.java]

```
import java.io.*;
public class manycat
{
   public static void cat(String s) throws Exception
   {
      FileReader inone =new FileReader(s);
      int t=inone.read();
      while (t!=-1)
      {
        System.out.print((char)t);
        t=inone.read();
      }
   }
   public static void main(String[] args) throws Exception
   {
     for (int i=0;i<args.length;i++) cat(args[i]);
   }
}
```

48. [Lecture11/changecase.java]

```
import java.io.*;
public class changecase
{
   public static void main(String[] args) throws Exception
   {
     FileReader inone =new FileReader(args[0]);
     int t=inone.read();
     while (t!=-1)
     {
       if (Character.isUpperCase((char)t))
                   System.out.print(Character.toLowerCase((char)t));
       else  if (Character.isLowerCase((char)t))
                   System.out.print(Character.toUpperCase((char)t));
             else System.out.print((char)t);
       t=inone.read();
     }
   }
}
```

49. [Lecture11/ascii.java]

```
import java.io.*;
public class ascii
{
   public static void main(String[] args) throws Exception
   {
     BufferedReader inone =new BufferedReader(new FileReader(args[0]));
     int t=inone.read();
     while (t!=-1)
     {
       System.out.print((char)t);
       System.out.print(t);
       t=inone.read();
     }
   }
}
```

50. [Lecture11/remnewline.java]

```
import java.io.*;
public class remnewline
{
  public static void main(String[] args) throws Exception
  {
    FileReader inone =new FileReader(args[0]);
    int t=inone.read();
    while (t!=-1)
    {
      if (t !=10)System.out.print((char)t);
      t=inone.read();
    }
  }
}
```

51. [Lecture11/tenperline.java]

```
import java.io.*;
public class tenperline
{
 public static void main(String[] args) throws Exception
 {
   FileReader inone = new FileReader(args[0]);
   int t=inone.read();
   while (t!=-1)
   {
     int i=0; while (i<10 && t != -1)
     {
       if (t !=10){i++;System.out.print((char)t);}
       t=inone.read();
     }
     System.out.println();
   }
 }
}
```

52. [Lecture11/Words.java]

```
import java.io.*;
public class Words
{
  public static void main(String[] args) throws Exception
  {
     boolean inword=false;
     String newWord="";
     FileReader inone = new FileReader(args[0]);
     int t=inone.read();
     while (t!=-1)
     {
       if (!inword)
          if (t=='\n' || t=='\t' || t==' ') ;//empty true part
          else {newWord=String.valueOf((char)t);inword=true;}
               //String.valueOf((char)t) converts char t to a String
       else if (t=='\n' || t=='\t' || t==' '){System.out.println(newWord);inword=false;}
            else newWord=newWord+String.valueOf((char)t);
       t=inone.read();
     }
   }
}
```

53. [Lecture8/TestSortOnInput.java]

```
import java.io.*;
class TestSortOnInput
{
    public static void main(String [ ] args) throws IOException
    {
      int a [ ] = Lecture8.SortOnInput.readintsAscending();
      System.out.println("In Ascending Order, the numbers are:");
      LectureNonVoidMethods.Arrays.print(a);
    }
}
```

54. 1,0
55. 1
56. 8,12,14,15,16
57. [LectureTest/TestArraysOfInts.java]

```
import Lecture8.*;
import java.io.*;
public class TestArraysOfInts
{
        public static void main(String[] args) throws IOException
        {
          int [ ] s = ArraysOfInts.readAndSortintArrayAscending();
          System.out.println("Ascending order gives:-");
          ArraysOfInts.printArray(s);
          System.out.println("Descending order gives:-");
          ArraysOfInts.descendingBubbleSort(s, s.length);
          ArraysOfInts.printArray(s);
          ArraysOfInts.ascendingBubbleSort(s, s.length);
          if (ArraysOfInts.binarySearch(s,s.length,71))
             System.out.println("You entered 71");
          else System.out.println("71 not found");
        }
}
```

58. [Lecture8/ArraysOfStrings.java]

```
package Lecture8;
import java.io.*;
import java.util.Scanner;
public class ArraysOfStrings
{
        public static Scanner in =new Scanner(System.in);

        public static String[] readStringArray() throws IOException
```

```
{
        System.out.print("How many Strings do you want to enter? ");
        int k=in.nextInt();in.nextLine();
        String a[] = new String[k];
        System.out.println("Now type in the Strings");
        for (int i=0;i<k;i++)
        a[i]=in.nextLine();
        return a;
}
static boolean after(String s,String t)
{
       return (s.compareTo(t)>0);
}

public static void printArray(String [ ] a)
{
          for (int i=0;i<a.length;i++)
        System.out.println(a[i]);
}

public static String [] readAndSortStringArrayAscending() throws IOException
{
        System.out.print("How many Strings do you want to enter? ");
        int k=in.nextInt();in.nextLine();
        String a[] = new String[k];
        System.out.println("Now type in the Strings");
          for (int i=0;i<k;i++)
        insertAscending(in.nextLine(),a,i);
        return a;
}

static void shuffle(int from, int to, String a[ ])
{
 for (int i=to;i>=from;i--)a[i+1]=a[i];
}
static int findWhereToInsertAscending(String x,String [ ] a, int firstfree)
{
  int pos=0;
  while(pos<firstfree && after(x,a[pos]) )pos++;
  return pos;
}
static void insertAscending(String x, String [ ] a, int firstfree)
{
  int pos=0;
  int z=findWhereToInsertAscending(x,a,firstfree);
  shuffle(z,firstfree-1,a);
  a[z]=x;
}
public static void ascendingBubbleSort(String[] a, int size)
{
  for (int i=0;i<size-1;i++)
  for (int j=i+1;j<size;j++)
  if (after(a[i],a[j])) {String temp=a[i]; a[i]=a[j]; a[j]=temp;}
}
```

```
        public static void descendingBubbleSort(String[] a, int size)
        {
          for (int i=0;i<size-1;i++)
          for (int j=i+1;j<size;j++)
          if (after(a[j],a[i])) {String temp=a[i]; a[i]=a[j]; a[j]=temp;}
        }
        public static boolean linearSearch(String [ ] a, int size, String thing)
        {
         int i;
         for (i=0;i<size&&!a[i].equals(thing);i++);
         return (a[i-1].equals(thing));
        }

        public static boolean binarySearch(String [ ] a, int size, String thing)
        {
          int first=0,last=size-1;
          int mid;
          while (first <=last)
          {
              mid=first + (last-first)/2;
              if (thing.equals(a[mid])) return true;
              else if (after(a[mid],thing)) last = mid-1;
                   else first=mid+1;
          }
          return false;
        }

        public static int binarySearchPos(String [ ] a, int size, String thing)
        {
          int first=0,last=size-1;
          int mid;
          while (first <=last)
          {
              mid=first + (last-first)/2;
              if (thing.equals(a[mid])) return mid;
              else if (after(a[mid],thing)) last = mid-1;
                   else first=mid+1;
          }
          return -1;
        }
}
```

59. [LectureTest/TestArrayOfStrings.java]

```
import Lecture8.*;
import java.io.*;
public class TestArrayOfStrings
{
        public static void main(String[] args) throws IOException
        {
          String [ ] s = ArraysOfStrings.readAndSortStringArrayAscending();
          System.out.println("Ascending order gives:-");
          ArraysOfStrings.printArray(s);
          System.out.println("Descending order gives:-");
          ArraysOfStrings.descendingBubbleSort(s, s.length);
          ArraysOfStrings.printArray(s);
          ArraysOfStrings.ascendingBubbleSort(s, s.length);
          if (ArraysOfStrings.binarySearch(s,s.length,"telephone"))
             System.out.println("You did enter telephone");
          else System.out.println("telephone not found");
        }
}
```

60. [Lecture8/ass1.java]

```
package Lecture8;
import Lecture8.*;
import java.io.*;
public class ass1
{
    static int n1=8;
    static int n2=8;
    static int n3=8;
    static int b=6; /*Number of 1st year marks that count*/
    static int c=6; /*number of 3rd year marks that count*/
    static int d=8; /*number of remaining 2nd and 3rd year units that count */
    static int u=1;
    static int v=3;
    static int w=5;
    static int firstyear[]=new int[n1];
    static int secondyear[]=new int[n2+n3-c];
    static int thirdyear[]=new int[n3];
    static int x,y,z;



     static int sumarray(int a[],int upto)
     {int total=0;
      for(int i=0;i<upto;i++)total=total+a[i];
      return total;
     }

     static String degree (double score)
     { if (score<35.0) return("Fail");
       if (score<40.0) return("Pass");
       if (score<50.0) return("Third");
       if (score<60.0) return("Lower Second");
       if (score<70.0) return("Upper Second");
       return("First");
     }

     public static void main(String[] args) throws IOException
     {int a[] =new int[3];
      System.out.print("Enter First Year: ");
      Arrays.readintarray(firstyear,n1);
      System.out.print("Enter Second Year: ");
```

```
        Arrays.readintarray(secondyear,n2);
        System.out.print("Enter Third Year: ");
        Arrays.readintarray(thirdyear,n3);
        Arrays.descendingbubblesort(firstyear,n1);
        Arrays.descendingbubblesort(thirdyear,n3);
        x=sumarray(firstyear,b);
        z=sumarray(thirdyear,c);
        for (int i=0;i<n3-c;i++)secondyear[n2+i]=thirdyear[c+i];
        Arrays.descendingbubblesort(secondyear,n2+n3-c);
        y=sumarray(secondyear,d);
        System.out.println(degree((u*x+v*y+w*z)/(u*b+w*c+v*d)));

        }
}
```

61. [LectureSimpleObjects/IntAndDouble1.java]

```
package LectureSimpleObjects;
public class IntAndDouble1
{
    public int i;
    public double d;

    public IntAndDouble1(int x,double y) /*A Constructor Method*/
    {
      i=x;
      d=y;
    }
    public String toString() /*An Instance Method*/
    {
        return "Double: " + d + "\nInt: " + i + "\n";
    }
}
```

62. [LectureSimpleObjects/People1.java]

```
package LectureSimpleObjects;
import LectureSimpleObjects.*;
public class People1
{
     public static void main(String [] args)
     {
        Date x1 = new Date(2,3,2001);
        Person x2 = new Person("Joseph","Boteju",x1,false);
        Person x3 = new Person("Pushpa","Kumar",new Date(17,2,1942),"male");
        Person x4 = new Person("Ola","Olatunde",new Date(27,8,1983),"female");
     }
}
```

63. javadoubleLectureInheritance/Test9.java

64. [LectureInheritance/Square.java]

```
package LectureInheritance;
public class Square extends Rectangle
{
  public Square(int n)
  {
    super(n,n);
  }
}
```

65. [LectureInheritance/BetterSquare.java]

```
package LectureInheritance;
public class BetterSquare extends BetterRectangle
{
   public BetterSquare(int n,char m, char e)
   {
    super(n,n,m,e);
   }
}
```

66. [LectureInheritance/BetterLeftBottomTriangle.java]

```
package LectureInheritance;
public class BetterLeftBottomTriangle extends BetterSquare
{
    public BetterLeftBottomTriangle(int n,char m, char e)
    {
        super(n,m,e);
    }

    public void Draw()
    {
        for(int i=1;i<this.height;i++)
        (new BetterLine(ends,middle,i)).Drawln();
        (new BetterLine(ends,ends,height)).Drawln();
    }
}
```

67. [LectureInheritance/SolidLeftBottomTriangle.java]

```
package LectureInheritance;
public class SolidLeftBottomTriangle extends BetterLeftBottomTriangle
{
  public SolidLeftBottomTriangle (int h)
  {
    super(h,'*','*');
  }
}
```

68. [LectureInheritance/Test8.java]

```
package LectureInheritance;
class Test8
{
 public static void main(String [ ] args)
 {
   (new HollowLeftBottomTriangle(5)).Draw();
   (new SolidLeftBottomTriangle(5)).Draw();
 }
}
```

69.
    ```
    Goldsmiths is great
    zz
    z
    4
    3
    2
    1
    0
    ```
    This is because we are printing the contents of the `Vector` in reverse order.
70. It simply prints out the sorted numbers twice. We have lost the original order. This is because `Vector` parameters are passed by Reference and not by Value.
71. [Lecture12/longestline.java]
    ```
    package Lecture12;
    import java.io.*;
    import java.util.*;
    public class longestline
    {

        static String longestLine(Vector <String> tomato)
        {
            String longestsofar="";
            for(int i=0;i<tomato.size();i++)
            if (tomato.elementAt(i).length() >= longestsofar.length())
                longestsofar=tomato.elementAt(i);
            return longestsofar;
        }
        public static void main(String[] args) throws Exception
        {
            System.out.println("the Longest line is\n" +longestLine(lines.fileToVector(args[0])))
        }
    }
    ```
72. [Lecture12/occ1.java]
    ```
    package Lecture12;
    import java.io.*;
    import java.util.*;
    public class occ1
    {
        public static int occurrences(char x, Vector <Character> v)
        {
             int total=0;
             for (int i=0;i<v.size();i++)
            if (v.elementAt(i)==x) total=total+1;
            return total;
        }
        public static void main(String[] args) throws Exception
    ```

```
    {
        Vector <Character> v= fileToVector.fileToVector(args[0]);
        for (int i=30;i<127;i++)
        {
            int n= occurrences((char)i,v);
            if (n>0) System.out.println((char) i + " "+ n);
        }
    }
}
```

Note: We have defined a method for finding the number of occurrences of a particular char in a `Vector` of Characters

73. `java longestline Lecture11/words`
74. `[Lecture12/occ2.java]`

```
package Lecture12;
import java.io.*;
import java.util.*;
public class occ2
{
    public static int biggest(int[] a)
    {
        int biggestsofar=0, position=0;
        for(int i=0; i<a.length;i++)
        if (a[i]> biggestsofar)
            {
                biggestsofar=a[i];
                position=i;
            }
          return position;
    }
    public static void main(String[] args) throws Exception
    {

        final int max = 512;
        int[] occs = new int[max];
        Vector v= fileToVector.fileToVector(args[0]);
        for (int i=0;i<max;i++) occs[i]=occ1.occurrences((char)i,v);
        int huge=biggest(occs);
        while (occs[huge] !=0)
           {
            System.out.println((char) huge + " "+ occs[huge]);
            occs[huge]=0;
            huge=biggest(occs);
           }
    }
}
```

75. `[Lecture12/occ3.java]`

```
package Lecture12;
import java.io.*;
import java.util.*;
public class occ3
```

```
{

    public static void cat(String s, int[] a) throws Exception
    {
        FileReader inone = new FileReader(s);
        int t=inone.read();
        for(int i=0; i<a.length;i++)a[i]=0;
        while (t!=-1)
        {
            a[t]=a[t]+1;;
            t=inone.read();
        }
    }
    public static void main(String[] args) throws Exception
    {
           final int max = 255;
           int[] occs = new int[max];
           cat(args[0],occs);
           int huge=occ2.biggest(occs);
           while (occs[huge] !=0)
           {
             System.out.println((char) huge + " "+ occs[huge]);
             occs[huge]=0;
             huge=occ2.biggest(occs);
           }
    }
}
```

76. [Lecture12/answers/freq.java]

```
package Lecture12.answers;
import java.io.*;
import java.util.*;
public class freq
{
    static final int firstletter = 40;// A
    static final int max = 122;//z
    static int[] occs = new int[256];

    static int sum(int[] a)
    {
        int totalsofar=0,
        position=firstletter;
        for(int i=firstletter; i<max;i++)
            totalsofar=a[i]+totalsofar;
        return totalsofar;
    }
    public static void main(String[] args) throws Exception
    {
        Lecture12.occ3.cat(args[0],occs);
        int huge=Lecture12.occ2.biggest(occs);
        float total=sum(occs);
        while (occs[huge] !=0)
        {
            System.out.println((char) huge + " "+
            100* (occs[huge]/total) + "%");
            occs[huge]=0;
            huge=Lecture12.occ2.biggest(occs);
        }
    }
}
```

77. [Lecture12/answers/find.java]

```
import java.io.*;
import java.util.*;
public class find
{
    static Vector v=new Vector();
    public static void cat(String s) throws Exception
    {
        Scanner inone =new Scanner(new FileReader(s));
        while (inone.hasNextLine()) v.addElement(inone.nextLine());
    }

    public static void main(String[] args) throws Exception
    {
        cat(args[0]);
        int i=0;
        while (i<v.size())
        if (((String)v.elementAt(i)).equals(args[1]))
            {
                System.out.println("yes");
                return;
            }
            else i=i+1;
        System.out.println("no");
    }
}
```

78. [Lecture13/studentsinvector.java]

```
package Lecture13;
import java.io.*;
import java.util.*;
public class studentsinvector
{
    public Vector <Student> students =new Vector();

    public studentsinvector(String s) throws IOException
    {
        String name;
        int exam;
        int cwk;
        Scanner in =new Scanner(new FileReader(s));
        while(in.hasNextLine())
        {
          name =in.nextLine();
```

```
            cwk = Integer.parseInt(in.nextLine());
            exam = Integer.parseInt(in.nextLine());
            students.addElement( new Student(name,exam,cwk));
          }
      }
     public String toString()
     {
         String t="";
          for (int i=0;i<students.size();i++) t=t+students.elementAt(i) +"\n";
          return t;
     }
  }
```

79. [Lecture13/printstudentsinvector.java]

```
package Lecture13;
import java.io.*;
import java.util.*;
public class printstudentsinvector
{
    public static void main(String [ ] args) throws IOException
    {
        studentsinvector V = new studentsinvector("marks");
        for (int i=0;i<V.students.size();i++)
             System.out.println(V.students.elementAt(i));
    }
}
```

80. [Lecture13/printsortedstudentsinvector.java]

```
package Lecture13;
import java.io.*;
import java.util.*;
public class printsortedstudentsinvector
{
    static Vector <Student> students= new Vector();

    static int positionofbiggest(Vector v)
    {
          double biggest = ((v.elementAt(0)).total;
          int position = 0;
          for (int i=1;i<v.size();i++)
          if ((((v.elementAt(i))).total > biggest)
          {
              position=i;
              biggest=(v.elementAt(i)).total;
          }
          return position;
    }

    public static void main(String[] args) throws IOException
    {
         studentsinvector V = new studentsinvector("marks");
         while (V.students.size()>0)
```

```
            {
                int k = positionofbiggest(V.students);
                System.out.println(V.students.elementAt(k));
                V.students.removeElementAt(k);
            }
        }
    }
```

I've done it by first storing them all in a `Vector` as before and then repeatedly finding the biggest, printing it, and deleting it, until there are no more left in the `Vector`.

81. It is the place where the program starts executing.
82. HelloWorld.class
83. A Method is a chunk of program code (also called a procedure or function or sub-routine) that performs a particular task. To make a method perform a task simply 'call it'. Very importantly: **We don't need to understand or even see the code of the method in order to be able to use it.** It may have been written by someone much cleverer and more experienced than ourselves. We can call methods over and over again. By giving methods meaningful names this helps us break down a programming problem into smaller well defined subproblems. Use parameters enables us to generalise our solutions.
84. A recursive method is one that calls itself. For example:

```
public class Factorial
{
    static    int fact(int n)
    {
        if (n==0) return 1;
        else return n*fact(n-1);
    }
    public static void main(String[] args)
    {
        System.out.print(fact(Integer.parseInt(args[0])));
    }
}
```

85. To explain your code // one line /* */ many lines
86. A for loop consists of the word `for` followed by a expression statement, then a boolean expression and then another expression statement enclosed in round brackets separated by semicolons and then a body which is a statement. The first statement expression we call the *initialisation expression*. The boolean expression in the middle we call the *guard* and the final statement in the brackets called the *increment expression*. So the structure of a `for` loop is

```
for (<initialisation expression>;<guard>;<increment expression>) <body>
```

Any or all of theses components can be empty. So an extreme, but syntactically correct `for` loop is
for (;;);

87. `[LectureExtras/for.java]`

```
for(int i=0;i>-1;i=i+1) System.out.println("*");
```

88. An Exception is a sometimes signal that an error or a special condition has occurred. To *throw* an exception means to signal an error. To *catch* an exception means to handle the error, i.e. to take action to recover from the error.

89. int, bool char.
90. `int x` This allocates four bytes in memory to store an integer value. This value will be refereed to as `x`.
91. `x =5` This stores the value 5 in the address corresponding to the variable `x`.
92. true,false
93. `||` means or
`&&` means and
for example
(2¡1) && (2¿1) evaluates to false but
(2¡1) —— (2¿1) evaluates to true
94. `Vectors` and Arrays are both used to store data in memory. `Vectors` can only store Objects but Arrays can store any type. Arrays are fixed in size but `Vectors` can grow or shrink. Elements of arrays are accessed using `a[i]`. The elements of `Vectors` are accessed using the instance method `elementAt`.
95. x[2]
96. v.elementAt(2)
97. In Java all statements end with a semi-colon `;`. If we leave the semi-colon out the compiler will complain! If we try compiling the program:

```
class  bad
{
        public static void main(String[] args)
        {
          System.out.println("Henry")
        }
}
```

When we try to compile this program we get an error message telling us there was a semicolon missing.

The Java compiler tells us that it got to line 6 when it realized that there was an error. In fact the error is on line 5. It puts a little caret pointing at where the error might be.

98.

```
class Precedence
{
        public static void main(String[] args)
        {
          System.out.println(5*1+1);
        }
}
```

If the output is 6 then times binds more tightly. If the output is 10 then plus binds more tightly.

99. Because of brackets.
100. The compiler distinguishes between upper and lower case letters.
101. CLASS p { }
102. The first program prints 1 because it does integer division. The remainder is thrown away. The second program does proper division so the answer is 1.5.
103. The answer is 5
5 Because the abs of -5 is 5

104. [Lecture11/OddChar.java]

```
import java.io.*;
public class OddChar
{
   public static void main(String[] args) throws Exception
   {
     BufferedReader in =new BufferedReader(new FileReader(args[0]));
     int s=in.read();
     while (s!=-1)
     {
      System.out.print((char)s);
      in.read();
      s=in.read();
     }
   }
}
```

105.

```
public class NatPerson extends Person
{
    public String nationality;

    public NatPerson(String f, String n,String l, Date birth , boolean s)
    {
     super(f,l,birth,s);
     nationality=n;
    }
    public NatPerson(String f, String n, String l, Date birth, String s)
    {
      super(f,l,birth,s);
      nationality=n;
    }
}
```

106. The output is 6.
The variables a and b, because they refer to arrays are reference variables. So the assignment `int[ ] b =a` makes b point at the same bit of memory as a. So when a is changed this will change b.

107. The output of this program is 2. The variable x declared inside the method `f()` is local to `f()` and a different variable to the one that is printed at the end of the program.

108.

```
import LectureNonVoidMethods.*;
class Test2
{
 public static void main(String [ ] args)
 {
  int [ ] a;
  a = ArraysNew.readintarray();
  System.out.println("Their average is " + ArraysNew.average(a));
 }
}
```

109. [LectureExtras/bubble.java]

```
     public static void ascendingBubbleSort(int[] a, int size)
        {
          for (int i=0;i<size-1;i++)
          for (int j=i+1;j<size;j++)
          if ((a[i]>a[j]) {int temp=a[i]; a[i]=a[j]; a[j]=temp;}
        }
```

110.

```
package LectureExceptions;
public class  IntException
{
   public static boolean isInt(String s)
   {
     try
     {
       Integer.parseInt(s); return true;
     }
     catch(Exception e)
     {
      return false;
     }
   }
}
```

111.

```
import java.io.*;
import java.util.Scanner;
public class spell
{
  public static void main(String[] args) throws Exception
  {
     Scanner inone =new Scanner(new FileReader("words"));
     while (inone.hasNext())
     {
       String t= inone.nextLine();
       if (t.startsWith(args[0])) System.out.println(t);
     }
  }
}
```

112.

```
import java.io.*;
class GCD
{
        static int gcd(int m,int n)
        {
                 for (;!(m==n);)
                    if (m>n) m=m-n;
                    else n=n-m;
                 return n;
        }
        public static void main(String[] args) throws IOException
        {
          BufferedReader in =new BufferedReader(new InputStreamReader(System.in));
          System.out.print("Enter two whole numbers ");
          String s =in.readLine();
          int m=Integer.parseInt(s);
          s =in.readLine();
          int n=Integer.parseInt(s);
          System.out.println("The GCD of "+ m+" and " +n +" is " + gcd(m,n));
        }
}
```

113.

```
package LectureSimpleObjects;
public class House
{
    public int number;
    public int rooms;
    public Person [ ] people;

    public House(int number, int rooms, Person [ ] p)
    {
      this.number=number;
      this.rooms=rooms;
      people=p;
    }
}
```

```
package LectureSimpleObjects;
public class Street
{
    public String name;
    public House [ ] houses;

    public Street(String name, House [ ] h)
    {
      this.name=name;
      houses=h;
    }
}
```

114.

```
import java.io.*;
class Palindrome
{
    public static void main(String[] args) throws IOException
    {
      BufferedReader in =new BufferedReader(new InputStreamReader(System.in));
      boolean finished=false;
      while (!finished)
```

```
        {
            System.out.println();
            System.out.print("Enter String> ");
            String s =in.readLine();
            String t="";
            for (int i=0;i<s.length();i++)
            t=s.substring(i,i+1)+t;
            boolean right = t.equals(s);
          if (right) System.out.print("YES! ");
          else System.out.print("No ");
          System.out.print("the string " + s + " is ");
          if (!right) System.out.print("not ");
          System.out.println("a palindrome.");
            System.out.println();
            System.out.print("another go (y/n)? ");
            finished=((in.readLine()).charAt(0)!='y');
       }
     }
  }
```

115. The contents of file dog.java appears on the screen.
116. At end of file.
117. Nothing. The condition t==-1 would be true at the beginning.
118. ’h’
119. s.charAt(s.length() - 1)
120. [Lecture11/answers/swapab.java]

```
import java.io.*;
public class swapab
{
    public static void cat(String s) throws Exception
    {
        BufferedReader inone =new BufferedReader(new FileReader(s));
        int t=inone.read();
        while (t!=-1)
        {
            if (t =='a')System.out.print('b');
            else
                if (t=='b') System.out.print('a');
                else System.out.print((char)t);
              t=inone.read();
        }

    }
    public static void main(String[] args) throws Exception
    {
        cat(args[0]);
    }
}
```

121. Arrays fixed in length indexed using square brackets `Vectors` are dynamic elements can be added at the end or removed from the middle. Elements of `Vector` are of type Object.

122. 1 is not an Object
123. In order handle the basic types as objects 1 is equiv to new Integer(1).
124. v.addElement(new Integer(((Integer)v.elemenAt(0)).intValue()+
((Integer)v.elemenAt(1)).intValue())
v.elementAt(0)
is of type Object so we must cast it to Integer before we can apply `intValue()` which returns its value as an int. similarly for v.elementAt(1) We can then add together these two int values using `+`. Before we add the result to the `Vector` we must convert it back to Integer. using new Integer(...)
125. LectureExtras/av.java
126. One that calls itself.
127. `[LectureExtras/power.java]`

```
static int power(int n,int m)
        {
         if (m==0) return 1;
         else return n*power(n,m-1);
        }
```

128. `[LectureExtras/power2.java]`

```
       static int power(int n,int m)
       {     int total=1;
              while (m!=0)
              {  total=total*n;m=m-1;
               }
              return total;
        }
```

129. d
130. b
131. c
132. c
133. a

Appendix F

Reading List

Reading List

[Bis01] Judith Bishop. *Java Gently - Third Edition*. Addison-Wesley, 2001.

[CK06] Quentin Charatan and Aaron Kans. *Java - In Two Semesters - Second Edition*. McGraw-Hill, East London Business School, 2006.

[Dan07] Sebastian Danicic. *CIS109 Subject Guide Volume 1*. University of London, Goldsmiths College, 2007.

[DD07] Harvey Deitel and Paul Deitel. *Java - How to Program - 7/e*. Prentice Hall International, 2007.

[Dow03] Allen B. Downey. *How to Think Like a Computer Scientist - Java Version*. Green Tea Press, 2003. A free book – see http://greenteapress.com/thinkapjava/.

[Fla05] David Flanagan. *Java in a Nutshell, Fifth Edition*. O'Reilly, 2005.

[Hub04] John R. Hubbard. *Schaums:Outlines - Programming with Java*. McGraw-Hill, University of Richmond, 2004.

[Inc] Sun Microsystems Inc. http://java.sun.com/javase/reference/api.jsp. This is where you can look up information about Java classes and methods.

[Kan97] Jonni Kanerva. *The Java FAQ*. Addison-Wesley, 1997.

[LO02] Kenneth A. Lambert and Martin Osborne. *Java - A Framework for Programming and Problem Solving*. Brookes-Cole, 2002.

[NK05] Patrick Niemeyer and Jonathan Knudsen. *Learning Java - Third Edition*. O'Reilly, 2005.

[Smi99] Michael Smith. *Java - An Object Oriented Language*. McGraw-Hill, 1999.

[Wu06] C. Thomas Wu. *An Introduction to Object-Oriented Programming with Java, 4/e*. McGraw-Hill, Naval Postgraduate School, 2006.

Notes

Notes

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 University of London.
If you have any comments about this guide, either general or specific (including corrections, non-availability of Essential readings, 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 completed form (or a photocopy of it) to:
Publishing Manager, Publications Office, University of London, Stewart House, 32 Russell Square, London WC1B 5DN, UK.