BSc (Hons) Computing and Information Systems

CIS109

Introduction to Java and Object Oriented Programming (Volume 1)

Subject guide

Written by Sebastian Danicic, Department of Computing Science, University of London.

First published 2002

This edition published 2007

Printed by Central Printing Service, The University of London

Publisher:
University of London Press
Senate House
Malet Street
London
WC1E 7HU

Contents

Chapter 1

Introduction

1.1 How to Study this Course

This is an introductory programming course in Java. It is intended for students with no previous programming experience.

This is the first volume of two. You should work your way through each chapter in order. It is expected that you spend roughly one week studying each chapter. To study a chapter do the following:

1. Read through the chapter, trying out all examples on your computer as you go along.
2. Having read the chapter, attempt all the exercises at the end of chapter. It is important that you spend a considerable amount of time on each exercise before you look up the solution at the back of the guide.
3. If you cannot understand the solutions, try running them on your computer. If you are still having difficulty, then refer to the reading list at the beginning of each chapter.
4. Read the first item on the reading list for a different explanation of the topic covered in the chapter.

1.1.1 Reading List

The first section of each chapter has suggested reading. For example:

- [Dow03] Chapter 2
- [DD07] pages 53-57
- [Fla05] Chapter 1 and 2

The codes like "[Dow03]" refer to books in the Reading List on page 131.

1.1.2 Suggested Schedule for Volume 1

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 these times proportionally.

Week 1: Chapters 2, 3 and 4

Week 2: Chapters 5 and 6

Week 3: Chapter 7

Week 4: Chapter 8

Week 5: Chapter 8
Week 6: Chapter 9
Week 7: Chapter 10
Week 8: Chapter 11
Week 9: Chapter 12
Week 10: Chapter 12

1.1.3 Practice, Practice, Practice!

Learning to program is a bit like learning a musical instrument. Although theory is important, practice is much, much more important. **The only way to learn to program is to write lots and lots of programs!** The way we judge a good musician is by listening to her playing a piece of music. Similarly we judge a programmer by running her programs. We can also, of course, admire the technique of a musician, but really the technique is just a means to an end. We don't really care how the violinist makes the sound as long as it sounds good to our ear.

Unfortunately, the musical analogy breaks down here. It is not enough that our computer programs work. Although computer programs are primarily meant to be understood by a computer, they also need to be understood by other humans who need to adapt them and improve them. Programs must be easy for humans to understand. Simplicity in programming is the key. The simpler your program, the better it is. Never show off by doing something in a complicated way. Always keep it simple.

1.1.4 The Challenging Problems

The challenging problems in Appendix A, page 105 are central to the course. By attempting to solve these problems you will learn an enormous amount about how to program. Each challenging problem has two numbers, for example [1,5] associated with it. This means you need to have read as far as Volume 1 Chapter 5 before you attempt this problem.

To return to the musical analogy, these problems are equivalent to the pieces you would be expected to perform as a new musician. The problems range from very easy to very difficult. Do not worry if you can't master them all as quickly as your colleagues. Different people learn at different speeds. Just because someone gets there first, it does not mean that they will end up being a better programmer than you.

1.1.5 The Examination

In Volume 2 there is a sample exam paper with no solutions and further past exam questions with solutions. You should start attempting these questions at least two months before your real exam. Try to attempt the sample exam paper in real exam conditions. Give yourself three hours and then mark your exam yourself by referring to the subject guides.

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 of how to access this website will be posted on:

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

1.1.6 Multiple Choice Questions

In the appendix of both Volumes 1 and 2, there are some multiple choice questions with solutions.

1.2 The Course CD

The course is accompanied by a CD containing the following useful material:

1.2.1 Course Material

Clickable CIS109 Subject Guide Volume 1
Clickable CIS109 Subject Guide Volume 2
CIS109 Java Programs and Solutions to Exercises
2006 Exam
2005 Exam

1.2.2 Books and Documentation

Java Documentation From Sun
Free Book: *How to Think Like a Computer Scientist* by Allen B. Downey
Free Book: *Thinking in Java* by Bruce Eckel
Free Book: *Introduction to Programming Using Java* by David J. Eck
Java Elements Documentation

1.2.3 Essential Software

Windows

TextPad Editor for Microsoft Windows
Java Install for Microsoft Windows
Acrobat Reader for Microsoft Windows

Linux

Java Install for Linux
Acrobat Reader for Linux

1.2.4 Extra Software

For Microsoft Windows

bluej for Microsoft Windows
NetBeans for Microsoft Windows

Linux

bluej for Linux
NetBeans for Linux
Eclipse for Linux
Eclipse for Linux

1.3 Topics

The first volume of the Java Subject Guide considers many of the basic concepts of programming. These include:

- 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

In the second 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

1.4 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 on the last page of this volume (page 131). A good book to get started with is *How to Think Like a Computer Scientist* by Allen B. Downey. It is free and can be found on the course CD and at http://greenteapress.com/thinkapjava/ under the Gnu Free Documentation Licence. Thank you very much Allen B. Downey. I strongly recommend that you read chapters 1 to 13 of the book and do all its exercises.

1.5 Installing Java

Before you can usefully study this course, you need Java installed on your computer. The course CD contains an installable version of Java and instructions on how to install it.

Alternatively, go to

1. http://java.sun.com/javase/downloads/index.jsp
   Click on the **JDK 6** download button.
2. Java SE APIs and Documentation from http://java.sun.com/javase/reference/api.jsp.

If you are using Microsoft Windows you may wish to download and install TextPad Programmer's Text Editor from http://www.TextPad.com for editing, compiling, and running your Java programs (this is also provided on the course CD). You may prefer to use BlueJ from http://www.bluej.org/. Alternative programming environments include Netbeans which can be downloaded from http://java.sun.com/javase/downloads/index.jsp and Eclipse which can be downloaded from http://www.eclipse.org/.

1.6 Need Help Installing Java?

There is plenty of online help for installing Java. Try searching for "installing Java" using your favourite Internet search engine. See, for example, http://www.jibble.org/settingupjava.php.

1.7 Preliminaries

Before starting to learn Java, you need to know a few things about using a computer:

- You need some familiarity with a computer operating system. The operating system that you are using is probably one of the following:
  - Microsoft Windows
  - Unix (or Linux)
- You need to know how to create files using a text editor. In Microsoft Windows, we recommend that you use TextPad (Download from www.TextPad.com and on CD)

  In Unix, popular text editors that you might use include:

- vi
- emacs
- xedit
- nedit

It is important that you know how to create directories and subdirectories, copy, delete and move files.

1.8 Learning Outcomes

Having completed this subject guide you will understand programming concepts sufficiently to be able to write Java applications to solve simple programming problems. Topics covered include:

- Simple Output (Chapter 2, page 7)
- Arithmetic Expressions (Chapter 3, page 17)
- Variables (Chapter 4, page 23)
- Calling Methods (Chapters 5 and 9, pages 31 and 65)
- Keyboard Input (Chapter 6, page 37)
- Conditional Statements (Chapter 7, page 43)
- Simple `For` Loops (Chapter 8, page 53)
- One-Dimensional Arrays (Chapter 10, page 73)
- Nested Loops (Chapter 11, page 81)
- Defining Static Methods (Chapter 12, page 89)

Chapter 2

Your First Java Program

2.1 Learning Objectives

Chapter 2 explains:

- how to write Java programs that output messages to the terminal.
- about directory structure and where to put the programs you write during this course.
- about the `CLASSPATH` system variable.
- about the use of comments in a program.
- that Java is case-sensitive.
- about the purpose and syntax of the main method in a Java application.
- how to define `String` constants.
- how to compile and run Java programs.
- how to interpret some common compiler error messages.
- about the difference between `print` and `println`.

2.2 Reading

2.2.1 Main Reading

- Do all the exercises in Chapter 1 [Dow03] after you have read both this chapter and Chapter 1 of [Dow03].

2.2.2 Other Reading

- [Hub04] pages 1-13
- [CK06] pages 4-9

2.3 Directory Structure for the Course

I recommend that you create a directory (folder) for each chapter in the book. See Figure 2.1.

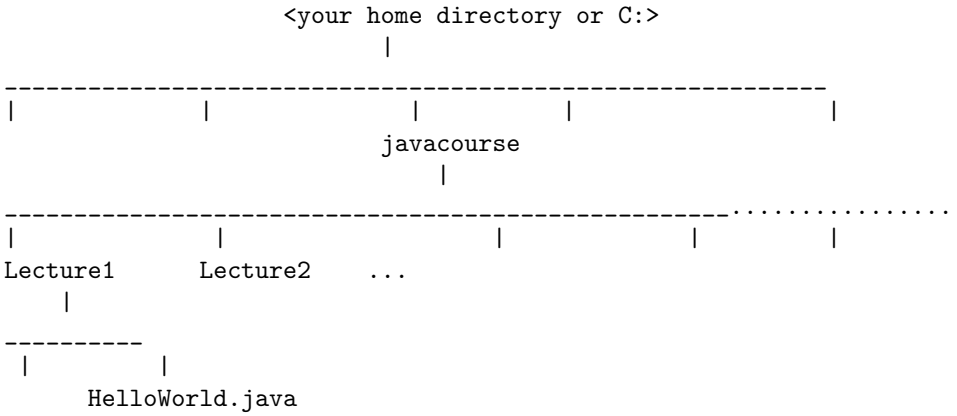

Figure 2.1: Directory Structure

2.4 Task

Read Pages 1-10 of [Dow03].

2.5 Your First Program

Throughout the text, we give suggested file names for each program. We put these file names in square brackets. For example, we write `[Lecture1/HelloWorld.java]`. This means that on the course CD the program can be found in a file called `HelloWorld.java` in a directory(folder) called `Lecture1`. I suggest that you also put your first Java program `HelloWorld` in a file called:

```
HelloWorld.java
```

in a directory(folder) called:

```
Lecture1
```

in the directory called

```
javacourse
```

If you do not put the programs where we suggest you may end up with problems since other programs may be looking in a particular place for another program.

2.5.1 CLASSPATH

There is a system variable called `CLASSPATH` that causes problems to beginners in Java. If you need to, please ask your tutor to help you with this.

This variable contains the set of directories (folders) where the Java system looks for classes (you will learn about classes later in the course).

2.5.2 Setting the `CLASSPATH` on Windows XP

In order to make everything in the course work smoothly you need to set the `CLASSPATH` system variable.

1. click on start → control panel.
2. click on performance and maintenance
3. click on system
4. click on advanced
5. click on Environment Variables
6. click on new
7. for the variable name write `CLASSPATH` and for the value write `c:\cis109\element.jar;c:\javacourse;.\`

If you have trouble with this, I suggest you do an Internet search using Google or some other search engine with *CLASSPATH java XP* as your search term.

2.5.3 Setting the `CLASSPATH` on Unix or Mac

Unix users should type:

```
export CLASSPATH=$HOME/element.jar:$HOME/javacourse:$CLASSPATH
```

Read the first chapter of [Hub04] for more details.

2.6 Editing, Compiling and Running your First Program

First, into TextPad (or the programming environment of your choice) type the example `[Lecture1/HelloWorld.java]` Having typed it in, save it and compile it. To do this using TextPad, you click on `compile Java` under the `tools` menu. If you have typed it in correctly, nothing will happen. If you have not typed it correctly you will get some error messages from the compiler. If you get error messages, then check that every character you have typed is exactly as it appears in the text. If you still get errors, then try reading Section 2.8. This may help you to find your errors. When you have done this, then compile your program again. Repeat this process until you have no errors and then run your program.

To run your program using TextPad, you click on `Run Java Application` under the `tools` menu.

If you are using Unix or MS Windows and do not have TextPad, you can compile and run your Java programs from the command line. Type `javac HelloWorld.java` at the command line to compile your program and type `java HelloWorld` to run it.

What happens when you run the program?

2.6.1 Summary

There are three phases in writing programs:

1. Editing the program
   (a) In Windows, I suggest that you use `TextPad`.
   (b) In Unix use your favourite text editor. I use `nedit`. Other people prefer `vi` or `emacs`.
2. Compiling the program
   (a) In Windows, click on `tools` followed by `compile` in `TextPad`.
   (b) In Unix (or DOS) type `javac` followed by file name, e.g. `javac HelloWorld.java`.
3. Running the program
   (a) In Windows, click on `tools` followed by `run Java application` in `TextPad`.
   (b) in Unix (or Dos) type `java` followed by class name, e.g. `java HelloWorld`.

2.7 Analysis of the `HelloWorld` Program

We will now analyse various aspects of the program: `[Lecture1/HelloWorld.java]` in more detail.

2.7.1 Comments

The very first line `// HelloWorld` is just a comment. After two forward slashes `//` you can write anything you like on that line. It will be ignored by the compiler and have no effect on what your program does when it runs. Comments are very important since when your programs become large the comments help to remind you how and why you wrote your programs the way you did.

2.7.2 The Other Way of Doing Comments

In Section 2.7.1 we saw one way of doing comments. In the program `[Lecture1/HelloWorld2.java]` we have included some text between `/*` and `*/`. This is how we do comments if we want them to last more than one line. We can think of `/*` as meaning “start comment” and `*/` as meaning “end comment”. It is essential that from the beginning of your programming “life” you get into the habit of commenting your programs.

2.7.3 The Program Heading

The next line

```
class HelloWorld
```

tells us the name of the program. All Java programs have a `class` statement very near the beginning. Normally, because our program is called `HelloWorld`, we store it in a file called

```
HelloWorld.java
```

This is not essential however. We could have called the file `anything.java` and it would still have worked. Because of the program heading, after we compile it, we will end up with a file in the current directory called `HelloWorld.class`.

2.7.4 Java is Case-Sensitive

This means that it matters whether we use small or capital letters. If we had written `CLASS` instead of `class`, the compiler would give us an error message and we would have to correct it before being allowed to run the program. Try it and see!

2.7.5 The Program Body

The rest of the file is the *body* of the program.

Matching Brackets

It starts with an open curly bracket { and ends with a closing curly bracket }. When you write programs, brackets must always match: for every opening bracket there must be a corresponding closing bracket and vice-versa.

The Main Method

All Java applications have what is called a *main method* which always starts:

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

This line is called the *heading* of the main method. The code inside the next pair of open and closing curly brackets is called the *body* of the main method.

```
{
```

```
   System.out.println("Hello World");
}
```

This is where we put what we actually want our Java program to do when we run it. In this example, the body of our main method consists of a single statement. In this case, the statement is a call to a *method*[1] whose name is `System.out.println`. We have passed this method the argument `"Hello World"`. When the `System.out.println` method is executed the `String` passed to it is *printed* on the computer screen.

[1] Methods will be studied in more detail in Chapters 5, 9 and 12.

2.7.6 `String`**s**

A `String` is a sequence of characters, with a double quote at either end. Examples of `String`s are

- `"sddffhh*^(_sg"`
- `"3253dssfdgg09231138"`
- `"1"`
- `""` (This one is called the *empty* `String`)

2.8 Some Compiler Error Messages

In Java all statements end with a semi-colon `;`. If we leave the semi-colon out the compiler will complain! Try compiling the program: [`Lecture1/bad.java`] When we try to compile this program we get an error message:

```
bad.java:6: ';' expected.
        System.out.println("Henry")
                                   ^
1 error
```

The Java compiler tells us that it got to line 6 when it realised 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. Another common error is to have the class name different from the file name. This is only a problem if we have the word `public`[2] before `class`. If we compile the program [`Lecture1/bad1.java`] we will get a compiler error message saying:

[2] The use of the word `public` will be explained later in the course.

```
bad1.java:2: Public class Bad1 must be defined in a file called "Bad1.java".
public class  Bad1
              ^
1 error
```

2.8.1 Correcting Compilation Errors

If your programs do not conform exactly to the rules for the syntax of Java, errors will appear when you try to compile your program. When you start writing programs you will have lots of compilation errors. The best way to correct them is just to correct the first one and then recompile. This is because the first error sometimes makes the compiler think there are lots of other errors which are not really there. Note: **Just because your program has no compilation errors it doesn't mean it will do what you want it to do!**

2.9 print vs. println

Consider:

```
public class Name
{
 public static void main(String[] args)
 {
   System.out.println("Sebastian Danicic");
   System.out.println("Sebastian Danicic");
   System.out.println("Sebastian Danicic");
 }
}
```

As you have seen, every time we call the `System.out.println` method it prints its actual parameter (the bit in the brackets after the word `System.out.println`) and then goes on to the next line. The output to the program above is

```
Sebastian Danicic
Sebastian Danicic
Sebastian Danicic
```

If we had written:

```
public class Name
{
 public static void main(String[] args)
 {
   System.out.print("Sebastian Danicic");
   System.out.print("Sebastian Danicic");
   System.out.print("Sebastian Danicic");
 }
}
```

The output would have been:

```
Sebastian DanicicSebastian DanicicSebastian Danicic
```

So, as we have seen, `System.out.println` prints its argument followed by a newline character which makes the cursor go onto the next line.

2.10 Exercises on Chapter 2

2.10.1 Printing your Name

Create a new program that prints your own name instead of Henry's. Don't forget to compile and run the new program.

Notice that the first program, HelloWorld.java starts with

```
class HelloWorld
```

and the second program, Name.java starts with

```
class Name
```

Notice that there is a file in your directory called `HelloWorld.class`.

Delete the file called `HelloWorld.class`. Now try to run it. What happens?

2.10.2 Print your Name Three Times

Write a Program that prints your name 3 times; once per line.

2.10.3 Print your Name Ten Times

Write a program that prints your name 10 times.

2.10.4 Print your Name a Hundred Times

Write a program that prints your name 100 times.

2.10.5 Print your Name a Thousand Times

Write a program that prints your name 1000 times. In Chapter 8, on `For` loops, you will learn a shorter way of programming this!

2.11 Summary

Having worked on Chapter 2 you will have:

- Written Java programs that output messages to the terminal.
- Understood about directory structure and where to put the programs you write during this course.
- Been introduced to the `CLASSPATH` system variable.
- Learned about the use of comments in a program.
- Learned that Java is case-sensitive.
- Understood the purpose and syntax of the main method in a Java application.
- Learned how to define `String` Constants.
- Learned how to compile and run Java programs.
- Understood how to interpret some common compiler error messages.
- Understood the difference between `print` and `println`.

Chapter 3

Arithmetic Expressions

3.1 Learning Objectives

Chapter 3 explains:

- how arithmetic expressions are used in programming to perform calculations.
- an alternative way of writing comments.
- how to use the integer and real types in programming.
- how the division operator gives different types of result depending on its operands.
- how to concatenate `Strings` using `+`.
- about the use of operator precedence in expressions.
- about the use of brackets in computing expressions.

3.2 Reading

- [Dow03] Chapter 2
- [DD07] pages 53-57
- [Fla05] Chapter 1 and 2

3.3 Introduction

Arithmetic expressions are a way of telling a computer to do calculations. Compile and run the program [`Lecture1/OnePlusOne.java`] `1+1` is an example of an arithmetic expression. When we call `System.out.println(1+1)` the arithmetic expression 1+1 is first evaluated to produce 2. When we run this program it prints 2.

3.4 Quotes Make All the Difference

What is the output of [`Lecture1/QuoteOnePlusOne.java`] The quotes round `1+1` make it into a `String`. Without the quotes `1+1` is an integer. As we will see in Chapter 4, an integer is called an `int` in Java.

3.5 Multiplication is Written with an Asterisk *

I am going to America and taking 250 pounds sterling with me. I want to know how much this is in US Dollars. There are 1.51 dollars in each pound. [Lecture1/PoundstoDollars.java] As you can see, 1*2 means 1 times 2.

3.6 Division is Written with a Forward Slash /

Write a program which prints out how many pounds there are in one dollar assuming there are 1.51 dollars in a pound. [Lecture1/DollarsToPounds.java] As you can see, 1/2 means 1 divided by 2.

3.7 Converting Centigrade to Fahrenheit

Here is a program to print a Centigrade to Fahrenheit conversion table where x degrees centigrade is $32 + 9x/5$ degrees Fahrenheit. [Lecture1/CentigradeToFahrenheit.java] The output of this program is

3.8 More About Division

3.8.1 Integer Division

If both the numerator and denominator are integers then Java does **integer division**. [Lecture1/div1.java] prints 1 when we run it. This is because

- 3 and 2 are both ints
- The largest integer which is less than or equal to 3/2 is one.

Notice also that 3/(-2) would give -1.

The general rule for integer division is to work out the largest integer which is less than the absolute value of the expression.

3.8.2 Non-Integer Division

To represent real numbers we simply include a decimal point and at least one digit to the right of the decimal point, for example 1.0 or 1.51. If either the numerator or denominator is real the division is 'what we would expect'. The following programs all print 1.5:

[Lecture1/div2.java] [Lecture1/div3.java] [Lecture1/div4.java]

3.8.3 Concatenating `String`s

As well as for adding numbers, the plus sign can be used for concatenating `Strings`. For example `"Hello" + "fred"` gives `"Hellofred"`
and `"Hello " + "fred"` gives `"Hello fred"` (Note the space at the end of the first `String`).
The program DollarsToPounds.java in Section 3.6 could have been written in a neater way as:
[`Lecture1/BetterDollarsToPounds.java`]

3.9 Operator Precedence

What is the output of [`Lecture1/Precedence.java`] The answer is 6. This is because when the system works out `5*1+1` it does the multiplication before it does the addition. We write

> "times binds more tightly than plus"

or

> "* binds more tightly than +".

3.9.1 Brackets

How would we make the system do the plus first? Answer: Use brackets! `5*(1+1)` would give 10.

You never need to remember operator precedence. Just use brackets to get the expression you want. Expressions inside brackets are always calculated first. For example (3+5)*2 evaluates to 16.

See [Fla05] page 29 for a list of all operators. or [DD07] page 53.

3.10 Exercises on Chapter 3

3.10.1 Pence to Dollars

Look up, on the internet or elsewhere, the exchange rate between UK Sterling and US Dollars. Write a program that works out how many pence in 250 dollars.

3.10.2 Ten Times Table

Write a program that prints out the 10 times table.

3.10.3 One Hundred and Thirty Seven Times Table

Write a program that prints out the 137 times table.

3.10.4 Operator Precedence

Write some programs to test the order in which expressions are evaluated in Java.

Note

To make the following programs work, you have to write the numbers as real numbers with a decimal point. That is for two, write 2.0. For one million write 1000000.0 and so on. This will be explained in Volume 2.

3.10.5 Seconds in a Year

Write a program to work out the number of seconds in 365 days.

3.10.6 Months in a Millennium

Write a program to work out the number of months in a millennium (1000.0 years).

3.10.7 Bits in a Megabyte

Write a program to work out the number of bits in a megabyte. (A byte is 8 bits and a megabyte is 2.0^{20} bytes) To work out 2.0^{10} for now simply write

$$2.0 * 2.0 * 2.0 * 2.0 * 2.0 * 2.0 * 2.0 * 2.0 * 2.0 * 2.0$$

Eventually you will learn a better way of achieving this!

3.10.8 Bits in a Gigabyte

Write a program to work out the number of bits in a gigabyte. (A gigabyte is 2.0^{10} megabytes.)

3.10.9 My Snail

Assume light travels at 299,792,458 metres per second, and the star Proxima Centauri is 4.2 light years away. My snail travels at 48 centimetres an hour. How many years will it take my snail to get to Proxima Centauri and back? Write a Java program to work it out.

3.10.10 Feeding my Snail

My snail eats two grams of lettuce a day. Write a program that works out how many metric tons of lettuce it will have to take with it to Proxima Centauri. There are a million grams in a metric ton.

3.11 Summary

Having worked on Chapter 3 you will have:

- Understood how arithmetic expressions are used in programming to perform calculations.
- Learned an alternative way of writing comments.
- Been introduced to the integer and real types in programming.
- Understood how the division operator gives different types of result depending on its operands.
- Learned how to concatenate `Strings` using `+`.
- Understood operator precedence in expressions.
- Understood the use of brackets in computing expressions.

Chapter 4

Variables

4.1 Learning Objectives

Chapter 4 explains:

- the purpose of variables.
- about the primitive types of Java.
- about the allowable `Strings` used for variable names.
- how to declare variables.
- how to use assignment statements.

4.2 Reading

- [Dow03] Chapter 2
- [DD07] pages 48-49
- [Hub04] pages 19-23
- [CK06] pages 11-21

4.3 Introduction

Variables are very important in all programming languages. Variables are used to store values that we need later on in a computation. Each variable represents some memory inside the computer. Into this memory, values can be stored. In order to use a variable, we first **declare** it with a **variable declaration** and then store a value in it using an **assignment statement**.

Consider [`LectureVariables/Hello1.java`]

4.4 Declaring Variables

In `Hello1.java`, first we declare the variable called `s`. The value `124` is then stored in this variable `s`. The **contents** of the variable `s` (in this case, 124) will be printed. When we run this program we will see

```
124
```

on the screen of our computer. We will not see `s` appearing on the computer screen. `s` is the **name** of the variable, not its contents.

Whenever we declare a variable we must give its **type**. The type of `s`, in this case, is `int`. This means that the only sorts of thing we can store in `s` are integers.

4.4.1 Other Types

Other basic types (usually called *primitive* types) in Java include

```
boolean
char
byte
short
long
float
double
```

Variables of different types are for holding different sorts of values.

Examples of legal declarations are:

```
boolean b; //A boolean variable called b.
char c,d; //Two char variables called c and d.
byte k; //A byte variable called k.
short silly; // A short variable called silly.
int m,n,p; //Three int variables called m, d and p.
long lilliput; // A long variable called lilliput.
float f1,g1,h; //Three float variables called f1, g1 and h.
double q,r; //Two double variables called q and r.
```

4.5 Variable Names

Any sequence of letters and digits that starts with a letter is a legal variable name. Examples of legal variable names are

- `x`
- `x1`
- `banana`
- `Kilimanjaro`
- `Y2K`
- `t3x4y666minush4`
- `ZuZuZu11`

There is absolutely no difference in the behaviour of

```
int x = 1543;
System.out.print(x);
```

and

```
int bananasplit = 1543;
System.out.print(bananasplit);
```

and

```
int BorisYeltsin54 = 1543;
System.out.print(BorisYeltsin54);
```

In each of the three program fragments we store the integer 1543 in a variable and then print out the contents of the variable. In all three

```
1543
```

will be printed out.

4.5.1 Important Fact about Replacing Variable Names

If we replace every occurrence of a variable name in a program by another that doesn't occur already in the program then the program will behave exactly the same.

4.6 Exercise: Boris Yeltsin's Pet Rabbit

Rewrite all the programs in this chapter that contain a variable in such a way as to not change their behaviour but so they all have a variable called `BorisYesltsinAndHisPetRabbit`.

4.6.1 Exercise

What is the output of [`LectureVariables/Hello1Boris.java`] If you think the answer is `BorisYeltsinAndHisPetRabbit` then please re-read this chapter. If you think the answer is `1543`, then carry on reading!

4.7 Wrong Assignments

Consider the program [`LectureVariables/WrongType.java`] When we try to compile this program we get the following error message:

```
WrongType.java:7: Incompatible type for =. Can't convert int to java.lang.String.
          s = 1;          // assignment statement
            ^
1 error
```

This is because we are trying to assign a value of 1 to a `String` variable but 1 is not a `String`, 1 is an integer (called `int` in Java). If we put double quotes round the 1 (i.e. `"1"`) it becomes a `String`.

Now consider Hello2.java:

This program prints `1568`.

So does Hello3.java: So does Hello4.java:

4.7.1 Executing Assignment Statements

When an assignment is executed, first the expression on the right hand side is calculated and the result is put into the variable on the left hand side of the assignment. So in `Hello4.java` when executing the assignment statement `s = s + 25;` first the expression `s + 25` is calculated to give `1568`. The result is then stored in the variable `s`.

4.7.2 A Common Mistake

Consider [`LectureVariables/Hello5.java`] What does it output? The answer is `s`. It does not print `30` because we are asking the system to print the `String` value `"s"` not the value contained in the `int` variable `s`. It is very important that you understand this! The `"s"` is NOT the same as `s`. Again, the quotes make all the difference.

4.7.3 Another Common Mistake

A common mistake made by beginners is to declare the same variable more than once inside the main method (or as we shall see later, in any method). Java does not allow this.

Consider [`LectureVariables/Dec2.java`] When we try to compile this program we get:

```
Dec2.java:6: Variable 's' is already defined in this method.
     int s = 53;
         ^
1 error
```

4.8 Assigning to the Same Variable More Than Once

It is allowed to assign to the same variable more than once, so the following program compiles with no errors.

[`LectureVariables/TwoAssign.java`]

The output of the program above is

```
26
```

because the value 1453 stored in variable s has been overwritten by the value 26. A new assignment to the same variable always causes the previous value in that variable to be thrown away and replaced with the new value.

4.9 A Common Mistake - Forgetting to Declare Variables

A very common mistake is to forget to define variables.

See, for example, [LectureVariables/Undeclared.java] The compiler complains with

```
Undeclared.java:5: Undefined variable: s
          s = 55;
          ^
Undeclared.java:6: Undefined variable: s
          System.out.println(s);
                             ^
2 errors
```

The solution is simply to add the declaration `int s;` as in [LectureVariables/Declared.java] and now there are no errors. Another possible solution is:

[LectureVariables/Declared1.java]

4.10 Shorthand

Instead of `int x; int y; x=1;y=1;`

we can write:

```
int x=1; int y=1;
```

or even

```
int x=1,y=1;
```

4.11 Exercises on Chapter 4

4.11.1 Add One

What is the output of [LectureVariables/AddOne.java]

4.11.2 Double

What is the output of [LectureVariables/DoubleDouble.java]

4.11.3 Arithmetic

What is the output of [LectureVariables/p1.java]

4.11.4 `String` Concatenation

What is the output of [LectureVariables/p2.java]

4.11.5 `String` and `int` Concatenation

What is the output of [LectureVariables/p3.java]

4.11.6 Division by `int`

What is the output of [LectureVariables/p4.java]

4.11.7 Division by Real

What is the output of [LectureVariables/p5.java]

4.11.8 Division by Zero

What is the output of [LectureVariables/p6.java]

4.11.9 Further Exercises (no solutions)

1. What would be the appropriate type for variables that represent each of the following:
   (a) The number of students in your class.
   (b) The average number of students per class in your college.
   (c) The distance from the earth to the moon measured to the nearest centimetre.
   (d) Whether a person has a degree.

4.12 Summary

Having worked on Chapter 4 you will have:

- Understood the purpose of variables.
- Learned the primitive types of Java.
- Learned which `String`s are allowable as variable names.
- Learned how to declare variables.
- Learned how to use assignment statements.

Chapter 5

Calling Methods

5.1 Learning Objectives

Chapter 5 explains:

- what a method is.
- how to call a method.
- about some methods of the class `java.lang.Math`.
- about method signatures.
- about some graphical methods.

5.2 Reading

- [BB99] Chapter 2
- [Hub04] Chapter 2
- [DD07] pages 203-206
- [Bis01] pages 54-60
- [Fla05] 64-70

5.3 Introduction

Java provides many useful methods for helping us to achieve what we want. In this chapter, we explain just a very few of them. It is very important that, as a programmer, you get to know where to find useful methods and how to use them.

5.4 What is a Method?

A *method* is a separate piece of code (also called a procedure or function or sub-routine) that performs a particular task. To make a method perform a task we "call it". To call a method, you just write the name of the method. Very importantly, **we do not need to understand how a method works or even see the code of the method in order to be able to use it. We just need to know what it does.** It may have been written by someone much cleverer and more experienced than ourselves. This chapter shows how to call methods. Later on in the course, but not in this chapter, you will learn how to write your own methods.

5.5 How to Call a Method

We have already seen a few method calls in the programs that we have written. Examples of these are:

- `System.out.print("hello");`

  Here we are calling the method `System.out.print` with one actual parameter `"hello"`.
- `System.out.println(5+2)` Here we are calling the method `System.out.print` with one actual parameter `5+2`.

  Here we are calling the method `System.out.println(5+2)`.
- `System.out.println("hello" + " fred");` Again, here we are calling the method `System.out.println` with one actual parameter `"hello" + " fred"`.
- `System.out.println();` Again, here we are calling the method `System.out.println` with no actual parameter so we use empty brackets `( )`.

We simply give its name together with a (possibly empty) list of *actual parameters* in round brackets.

Documentation for this package can be found at
http://www.cs.williams.edu/~bailey/JavaElements/documentation.html

5.6 Some Simple Methods for Random Numbers

The following program prints out a random number between 0 and 10 each time it is run: [`LectureElements/random.java`] Can this program output 0 or 10? Try running it a few times. Look up the Sun Java Documentation at
http://java.sun.com/j2se/1.5.0/docs/api/java/util/Random.html to find out.

5.7 Some Simple Methods for Graphics

For this section, you will have to copy the file `element.jar` from the course CD and put it in your `CLASSPATH` as described in Chapter 1. This is from the good book, *Java Elements* [BB99]. (Further information about the `Element` package can be found on the CD or at http://www.mhhe.com/javaelements.) Consider the program [`LectureElements/one.java`] This program:

- Creates a drawing window of size 200 pixels by 200 pixels
- Moves the 'pen' to co-ordinate (100,100)
- Draws a line from current position to co-ordinate (100,150)

The numbers in the brackets are called the *actual parameters* or *arguments* of the method. We can use arithmetic expressions in place of values for the arguments as in [`LectureElements/two.java`]

Just to make sure that this is clear, here is another example: [`LectureElements/three.java`] The way it works is that the arguments are worked out (or *evaluated*) before being passed to the method.

5.7.1 Instances of Objects

Every time we write something like `d= new DrawingWindow()` in our program we are creating a new instance of an Object of type `DrawingWindow`. Each Object of type `DrawingWindow` has its own collection of instance methods. So, for example there is now a new `"d.lineTo()"` method and a new `"d.moveTo()"` method. So every time we use `new DrawingWindow()`, we are not only creating the `DrawingWindow` itself, but also copies of all the instance methods such as `d.lineTo()`, etc.

5.8 Method Signatures

5.8.1 The Class `java.lang.Math`

Look up the class in http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Math.html There you will see a long list of methods that may be useful to you. These methods include:

```
public static  int abs(int a)
```

This is called a *method signature*. Ignoring, for the moment, the bit that says `public static`, this tells us that the method `abs` takes a single `int` parameter and returns an `int`. So whenever we call it we get back an `int`. Try [`LectureUsefulMethods/abs.java`] This program contains two calls to `Math.abs`. In the first call, the actual parameter is -5 and in the second call the actual parameter is 5.

The output of this program is

```
5
5
```

What do you think abs does?

```
public static float abs(float a)
```

This says that there is a version of the method `abs` that takes a single float parameter and returns a float. So whenever we call it we get back a float. Try [`LectureUsefulMethods/abs1.java`] The output of this program is

```
5.375
5.824
```

```
public static int round(int float)
```

This says that the method `round` takes a single float parameter and returns an `int`.

So whenever we call it we get back an `int`. Try [`LectureUsefulMethods/round.java`] The output of this program is

```
-5
6
6
```

What do you think `round` does? I am not telling you. You must find out for yourself!

Notice that I have left out the formal parameter name. It is acceptable to do this. Instead of writing

```
public static int abs(int n)
```

we write

```
public static int abs(int)
```

The name, `n`, of the formal parameter is not important. The type of the formal parameter(s) and the return type of the method are important.

5.8.2 Max

```
int max(int,int)
```

This method finds the maximum of two values. Write a program to try it out.

5.9 Exercises on Chapter 5

5.9.1 Square

Using `lineTo` and `moveTo`, from `element.jar`, write a program which draws a square.

5.9.2 Drawing a Cube

Using `lineTo` and `moveTo`, from `element.jar`, write a program which draws a cube.

5.9.3 Drawing a Childish Picture

Try out the programs in
http://www.cs.williams.edu/~bailey/JavaElements/examples/chapter02.elements.

By adapting some of the methods from these examples, try to draw a childish picture. Your picture should have a house with windows and a door and there should be a sun with a smiling face in the sky.

5.9.4 Signatures

Write down the signatures of all the methods in `java.lang.Math`

5.9.5 Exercise

Give the signatures of all the methods in `java.lang.String`.

5.10 Summary

Having worked on Chapter 5 you will have:

- Discovered what a method is.
- Learned how to call a method.
- Been introduced to some methods of the class `java.lang.Math`.
- Been introduced to method signatures.
- Been introduced to some graphical methods.

Chapter 6

Keyboard Input

6.1 Learning Objectives

Chapter 6 explains:

- how to write programs which accept input from the user and then behave in different ways depending on the input.
- how to prompt the user for input.
- how to use the `Scanner` class.
- how to input `int`s and other types.
- how to look up useful Java API information in http://java.sun.com/j2se/1.5.0/docs/ and http://java.sun.com/j2se/1.5.0/docs/api .

6.2 Reading

- [Hub04] pages 14-17
- [Bis01] pages 113-116
- [CK06] pages 21-24

6.3 Introduction

Most useful programs allow the user of the program to input information. One way to input data to a computer program is through the computer keyboard.

Question: How else can a user input information to a computer?

Often, input is achieved simply by typing and then pressing the `enter` key. We now learn how to write programs which accept input from the keyboard and then behave in different ways depending on the input. In Java, there are a few things we must remember to do if our program is to accept input. Study the program: `[Lecture2/EchoNew.java]` This program is waiting for you to type something in although it doesn't tell you. If, eventually, you do type something in, the program simply *copies* what the user typed in to the screen. Try running the program. When you type something in, remember to press the `Return` or `Enter` key when you have finished.

At this point in the course, do not try to understand everything about this program. The following facts may help your understanding:

- When we write programs with keyboard input we must remember to put `import java.util.Scanner;` at the beginning of our program. If we leave it out, like in [Lecture2/EchoNoImportNew.java] then we get the following compilation errors:

```
EchoNoImportNew.java:5: cannot find symbol
symbol  : class Scanner
location: class EchoNoImportNew
      Scanner in =new Scanner(System.in);
      ^
EchoNoImportNew.java:5: cannot find symbol
symbol  : class Scanner
location: class EchoNoImportNew
      Scanner in =new Scanner(System.in);
                      ^
2 errors
```

- If we want our programs to use keyboard input it is normal to include the line:

```
Scanner in =new Scanner(System.in);
```

  Here we are declaring a variable called `in` of type `Scanner`. The `System.in` associates `in` with the keyboard.

  We could equally well have written

```
Scanner calfsbrain =new Scanner(System.in);
```

  and

```
s=calfsbrain.nextLine();
```

  It should now be clear: `in` and `calfsbrain` are just arbitrary variable names.

- To read a line of text from the keyboard we call the method `in.nextLine()`. Here we are calling the `nextLine` method of `Scanner in` that we declared earlier.

The useful thing to remember is that when the method `nextLine()` is called, the program waits for the user to press some keys (ending with the `Enter` key). The key sequence is converted into a `String`, preserving the order in which the characters were typed (not including the `Enter` key). This `String` is returned by the method `nextLine()` and the program execution continues. In program Lecture2/EchoNew.java, the program waits for the user to type something in. Whatever the user types in is then stored in `String` variable `s`. This is done using the assignment statement `String s = in.nextLine();`. The final statement of the program `System.out.println(s);`, simply displays on the screen the value of the `String s`, which contains what the user typed in before the `enter` key was pressed.

6.4 Prompting the User For Input

In `Lecture2/EchoNew.java` in Section 6.3, the user will not know that he is expected to type something in. It is usual, therefore, to display a message first, asking the user to type something in. This is called *Prompting for Input*. Consider the program: [Lecture2/EchoWithPromptNew.java] After the user has pressed `Enter` the computer replies `you typed in` followed by whatever the user has input. This is achieved using `System.out.println("you typed in "+s);`. Here, the `String "you typed in"` and the `String` contained in the variable `s` are *concatenated* (See Section 3.8.3, page 19).

Now we give an example where the user is prompted for two `String`s, a first name and a last name. [Lecture2/BothNamesNew.java] In order to read two values here we simply call `z.nextLine()` twice and store the results in different variables. (Here we have used `s` and `t`.)

Suppose we wanted to print the surname first and then the first name. We would do this by changing the order of s and t in the final System.out.println statement, as in

[Lecture2/BothNamesRevNew.java]

What is wrong with the following example: [Lecture2/BothNamesRevDec2New.java] The answer is that we have declared the same variable twice. The compiler says:

```
BothNamesRevDec2New.java:10: s is already defined in main(java.lang.String[])
       String s =in.nextLine();
              ^
1 error
```

(Recall Section 4.7.3.)

6.5 Inputting ints

Using the Scanner class, we can input values of all different types. So far, the only type we have seen a user input is a String. What if the user wants to input a number to a computer program? In Java, the way to do this is to call the method nextInt().

6.5.1 nextInt()

Each time nextInt() is called, the next int in the input is returned. Every method in Java belongs to a class. You will learn a lot more about how to use methods in Chapter 9. The method, nextInt(), belongs to the class called Scanner. See http://java.sun.com/j2se/1.5.0/docs/api/java/util/Scanner.html for a full list of all the Scanner methods. Look up nextInt() there.

Study the program, [Lecture2/Add1New.java] First, the user's input is read into the int, n. This is achieved by using the statement:

```
int n=in.nextInt();
```

The value of the arithmetic expression n+1 is finally printed out by the program. What happens if you don't enter an integer? Try it! Enter a String like assd. The Java interpreter gives the following error message:

```
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:819)
        at java.util.Scanner.next(Scanner.java:1431)
        at java.util.Scanner.nextInt(Scanner.java:2040)
        at java.util.Scanner.nextInt(Scanner.java:2000)
        at Add1New.main(Add1New.java:8)
```

The nextInt() method has thrown a "java.util.InputMismatchException" because we did not enter a String that it could convert to an int. A similar error message would occur if the user entered a real number like 14.5. An *exception* is thrown when an error occurs. Exceptions will be covered in more detail in Volume 2.

6.6 Exercises on Chapter 6

6.6.1 Double

Write a program which asks the user to enter an integer and prints out double the number.

6.6.2 Add Two Numbers

Write a program which asks the user to enter two integers. The computer adds them up and prints out the answer.

6.6.3 Average

Write a program which asks the user to enter three numbers and then prints out their average.

Notice that the average of three `int`s is not necessarily an `int`.

6.6.4 Question

What would have happened if we had written `3` instead of `3.0`? Remember integer division? See Section 3.8.

6.6.5 Task

Rewrite your answers to the first two questions so that you declare all the variables at the beginning.

6.6.6 Task

Rewrite your answers to the first three questions but this time inputting `double`s instead of `int`s.

6.7 Summary

Having worked on Chapter 6 you will have:

- Learned how to write programs which accept input from the user and then behave in different ways depending on the input.
- Learned how to prompt the user for input.
- Learned how to use the `nextLine()` method.
- Learned how to input `int`s and to use `nextInt()`.
- Learned how to look up useful Java API information in http://java.sun.com/j2se/1.5.0/docs/ and http://java.sun.com/j2se/1.5.0/docs/api

Chapter 7

Boolean Expressions and Conditional Statements

7.1 Learning Objectives

Chapter 7 explains about:

- the `if - else` statement.
- the `if` statement.
- syntax and semantics.
- the sequential statement.
- the empty statement.
- program layout.
- Boolean expressions.

7.2 Reading

- [Hub04] Chapter 3
- [CK06] Chapter 2
- [DD07] pages 117-122
- [Dow03] Sections 4.1-4.5

7.3 Introduction

A lot of work in programming involves the programmer making decisions depending on variables in the program having certain values.

For example, suppose that you had been asked to write a program which asks the user to enter two numbers and prints out only the larger of the two numbers entered. You could not write such a program with just the statements you've seen so far. You need a *conditional statement*. There are at least two kinds of conditional statement:

- The `if - else` statement
- The `if` statement

7.3.1 Example of the `if - else` Statement

This is how you make programs do different things depending on different conditions. Here is an example:

[LectureConditionals/BiggestOfTwoNew.java]

The program reads two integers into variables `x` and `y` and then if `x>y` it prints out the value of `x`, and if `x`$\leq$ `y` it prints out `y`.

7.3.2 Example of the `if` Statement

An `if` statement is like an `if - else` statement except there is no `else` part.

[LectureConditionals/SimpleIfNew.java]

The program above, reads in an integer typed in by the user and prints out

```
the number you entered was too big
```

if this number is greater than 100. If, on the other hand, the number entered is not less than 100, then this program does nothing at all.

7.4 Syntax and Semantics

Before we go any further in our discussion of the conditional statement, we explain two important concepts in computing: *syntax* and *semantics*.

7.4.1 Syntax

The syntax of a statement or a program simply means what it looks like when it is written down. For example the syntax of an *assignment* statement consists of a variable name, like `x` or `borisyeltsinspetrabbit`, followed by an *equals* sign (=), and on the right hand side of the equals sign is some form of *expression*. To fully describe what an expression is would take a lot more rules. We will not do it here. An expression could be something simple like the constant 1 or the variable `y`, or it could be something more complicated like `x+1*100-5/2+(x-1)`.

7.4.2 Semantics

The semantics of a statement or program means what it does when it is run. For example the semantics of the assignment statement `x=e` is to first evaluate the expression `e` and then store the result in the variable `x`.

The semantics of `System.out.print(e)` is first to evaluate `e` and then to display this value on the computer screen. Before we talk about the syntax and semantics of conditional statements, we look at one more example.

7.4.3 Biggest of Three

A program that finds the biggest of three numbers.

[LectureConditionals/BiggestOfThreeNew.java] In Java <= stands for $\leq$ (less than or equal). This program reads in three integers into variables, x, y and z and then prints out the value of the biggest. How does it decide which is the biggest? If x $\geq$ y and x $\geq$ z then it prints out the value of the variable x since x is the biggest. (Notice that && means *and*.) If x is not the biggest, it checks whether y is the biggest, and if it is, it prints the value of y. If neither x nor y is the biggest, the program just prints out the value of z.

7.4.4 Question

What would happen if all the numbers entered by the user were the same? For example, if the user entered the number 20 three times, the program would output:

```
Biggest is 20
```

7.5 The Syntax of the if - else Statement

The if - else statement is the first example of a structured statement that we have seen. The reason that it is called structured is that it can contain more statements itself. The syntax of an if - else is given by

```
if (<boolean expression>) <statement> else <statement>
```

This means that a conditional consists of a *boolean expression* in brackets (we call this *the guard*) and a statement (called the *true part*) followed by the word else and then another statement (called the *false part*).

The true part and the false part can be as complicated as you want them to be. In the previous example, the true part is System.out.println(x); and the false part was itself the conditional statement:

```
if (y>=x && y>=z) System.out.println(y);
else System.out.println(z);
```

7.6 The Semantics of the if - else Statement

The semantics of the if - else statement is given in Figure 7.1. The semantics of an if - else statement is easy to define: when the computer executes a conditional statement, it first works out (evaluates) the guard. If the guard evaluates to be true it then executes the true part. If, on the other hand, the guard evaluates to be false then the computer executes the false part of the conditional. Consider the following program: [LectureConditionals/TestCond1New.java] Let us focus on the code

```
if (x>y) ;
```

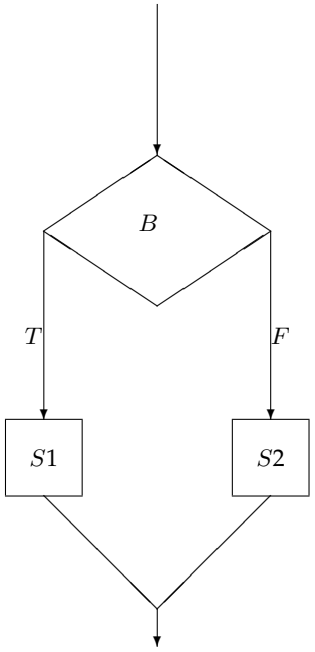

Figure 7.1: A picture of `if` B $S1$ else $S2$.

The guard B is evaluated. If it evaluates to true then $S1$ is executed. If it evaluates to false $S2$ is executed.

```
else System.out.println(y);
```

Here, the true part of the conditional is the *empty statement*. The empty statement is discussed in greater detail in Section 7.6.1. In this case if `x` is greater than `y` the program will do nothing.

7.6.1 The Empty Statement

Look at this program called `Silly.java`: It will compile correctly because it doesn't have any errors. But when we run it, it doesn't do anything. The body of the main method has five `empty` statements. The empty statement is very important. This is surprising as the empty statement does nothing!

Another way of writing the empty statement is $\{\}$ in the program: [`Lecture1/Silly1.java`] also does nothing, as does [`Lecture1/Silly2.java`] and [`Lecture1/Silly3.java`]

Programs that do Nothing

Make up some more Java programs that do nothing. What is the longest Java program that does nothing?

7.7 The Sequential Statement

A very important type of statement that you have already come across without knowing that it had a name is the *sequential statement*. The syntax of a sequential statement is simply a sequence of zero or more statements enclosed in curly brackets. The semantics of a sequential statement is to execute each of its component statements from left to right (i.e. *sequentially*). It is very important to realise that *a sequential statement is a single statement*. You can see a sequential statement in the conditional statements in the solutions to the exercises 7.12. Consider, for example, the conditional statement

```
if (y>z) {
           System.out.println(y);System.out.println(z);
         }
else     {
           System.out.println(z);System.out.println(y);
         }
```

The syntax of `if then else` (see Section 7.5 page 45) tells us that both the true part and the false part must be statements. Here they are both sequential statements.

7.7.1 A Common Mistake is to Leave out Curly Brackets

Quite often, novice programmers will make the following mistake:

```
if (y>z)
           System.out.println(y);System.out.println(z);

else     {
           System.out.println(z);System.out.println(y);
         }
```

The compiler gives an error. The reason why the compiler gives an error is that we are always allowed only a single statement between the guard `(y>z)` and the `else`. In the case above, there are two statements: `System.out.println(y);` and `System.out.println(z);`. To make it into a single statement we simply enclose it in curly brackets like this:

```
{System.out.println(y);System.out.println(z);}
```

We have created a single statement (a sequential statement) from two statements.

Try compiling [`LectureConditionals/ErrorCurliesNew.java`] and watch the compiler complain with

```
ErrorCurlies.java:19: 'else' without 'if'.
                        else     {
                        ^
```

The precise reason for this error message will be explained in Section 7.10.1.

7.7.2 Exercise

Put back the curly brackets and check that the program compiles with no errors. Does it?

7.7.3 Reminder

If you want to do more than one thing in either the true or false part of an `if - else` statement don't forget the curly brackets.

7.8 Program Layout

In the solutions to the exercises 7.12 you see how complicated programs should be laid out. Every closing curly bracket should lie directly underneath the corresponding opening bracket with nothing else in the way. In other words, you should be able to look vertically down the page from an opening curly bracket and the next thing you should see is the corresponding closing curly bracket. It is essential that you do this so that your programs are readable by other humans. Also the layout of conditionals should be of the form:

```
if (....)
    ....
    ....
    ....
else
    ....
    ....
    ....
```

7.9 `if` Statements

`if` statements are simply a shorthand form of the `if - else` statement, where the false part is empty. But because the false part to a conditional is the empty statement, we can leave out the the word `else` as well. So

```
if (n<0)
   {
     x=1;
     y=2;
   }
else ;
```

can be written more simply as:

```
if (n<0)
   {
     x=1;
     y=2;
   }
```

7.9.1 The Syntax of the `if` Statement

The syntax of an `if` statement is given by
`if (<boolean expression>) <statement>`
This means that a conditional consists of a *boolean expression* (we call this *the guard*) and a statement (called the *true part*).

7.10 The Semantics of the `if` Statement

When the computer executes an `if` statement, it first works out (evaluates) the guard. If the guard evaluates to *true* it then executes the *true* part. If, on the other hand, the guard evaluates to *false* then the computer does nothing.

7.10.1 Leaving out the Curlies

So if we mistakenly leave out the curly brackets (see Section 7.7.1) as in:

```
if (n<0)

     x=1;
     y=2;

else ;
```

The compiler will first see the `if` statement

```
if (n<0)  x=1;
```

and then see the assignment statement

```
     y=2;
```

Then it will come across

```
else ;
```

on its own and get confused and say

```
.......: 'else' without 'if'.
                         else  ;
                         ^
1 error
```

7.11 Boolean Expressions

7.11.1 The type `boolean`

There is a type in Java called `boolean`. There are only two values of this type

- `true`
- `false`

Consider [`LectureConditionals/Bool1.java`] If we compile and run this program, it simply prints out `true`.

7.11.2 The Simplest Boolean Expressions

The simplest boolean expressions are `true` and `false`.
What is the output of [`LectureConditionals/Bool2.java`] The answer is
`hello`
Here we have a conditional statement whose guard evaluates to true. This is because the boolean expression, `true`, evaluates to `true`. Hence the program executes the *true* part. What is the output of [`LectureConditionals/Bool3.java`] The answer is `goodbye`

7.11.3 Combining Boolean Expressions using Logical Operators

New boolean expressions can be created from old ones using the logical operators: *and*(`&&`), *or*(`||`), and *not*(`!`) The program [`LectureConditionals/Bool4.java`] outputs `false`, since *not(true)* equals *false*.

7.12 Exercises on Chapter 7

7.12.1 Not Not

What is the output of [LectureConditionals/Bool5.java] Truth Tables are a simple way of defining the behaviour of logical connectives like and,or and =>.

7.12.2 Truth Table for AND

Write a program that prints out the truth table for and like this:

```
p      |    q    |  p and q
--------------------------
true   |   true  |   true
true   |   false |   false
false  |   true  |   false
false  |   false |   false
```

7.12.3 Truth Table for OR

Write a program that prints out the truth table for or like this:

```
p      |    q    |  p or q
-------------------------
true   |   true  |   true
true   |   false |   true
false  |   true  |   true
false  |   false |   false
```

7.12.4 Truth Table for Implication

Write a program that prints out the truth table for implication. Use the fact that

p implies $q = (\neg p)$ or q.

7.12.5 Sorting Two Numbers

Write a program which asks the user to enter two numbers and then prints them out in

1. ascending order
2. descending order

7.12.6 Sorting Three Numbers

Write a program which asks the user to enter three numbers and then prints them out in

1. ascending order.
2. descending order.

7.12.7 Notes on these Exercises

Note that `&&` is java for *and*. Similarly `||` is java for *or*.

7.12.8 Validating One Input

Write a program that asks the user to enter an exam mark. This mark must be between zero and 100. If they do, then print out the number. If they don't, say "Wrong!".

7.12.9 Validating Two Inputs

Write a program that asks the user to enter an exam mark. This mark must be between zero and 100. If the user does not do this, say "Wrong exam" and exit the program. If the user does this correctly, then ask for a coursework mark which must also be between zero and 100. If this is also correct then write out the average of the two marks; otherwise, say "Wrong coursework".

7.12.10 Sorting Four Numbers

Write a program which asks the user to enter four `int`s, and then prints them out in ascending order. Later, we will see how this is easier using loops.

7.13 Summary

Having worked on Chapter 7 you will have learned about:

- The `if - else` statement.
- The `if` statement.
- Syntax and semantics.
- The sequential statement.
- Program layout.
- The empty statement.
- Boolean expressions.

Chapter 8

Simple Loops

8.1 Learning Objectives

Chapter 8 explains about:

- Programming repetition using loops.
- The syntax of `for` and `while` loops.
- The semantics of `for` and `while` loops.

8.2 Reading

- [CK06] Chapter 3
- [Hub04] Chapter 4
- [DD07] Chapter 5

8.3 Introduction

If we want the computer to repeat a task a number of times, the best way to do it is using a loop. One form of loop in Java is called the `for` loop. Compile and run the program `[Lecture4/LineOfTenStars.java]` When we run this program, it produces:

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

8.3.1 Exercise

Rewrite the above program so that it prints 100 stars.

8.4 Syntax of `for` Loops

A `for` loop consists of the word `for` followed by an expression, then a boolean expression and then another expression. These three expressions are enclosed in round brackets separated by semicolons. After this comes the *body* of the `for` loop which is a statement. So the structure of a `for` loop is

```
for (<initialisation expression>;<guard>;<final expression>) <body>
```

The first statement expression we call the *initialisation expression*. The boolean expression in the middle we call the *guard* and the final bit in the brackets is called the *final expression*. Any or all of theses components can be empty. So an extreme, but syntactically correct `for` loop is

```
for (;;);
```

8.4.1 Exercise

Make up a program with `for (;;);` in it and check that it compiles. The body of the `for loop` can be any statement however complex. In

```
for(int i=0;i<10;i=i+1) System.out.println("*");
```

- The initialisation expression is `int i=0`
- The guard is `i<10`
- The final statement is `i=i+1`
- The body is the statement `System.out.println("*");`

8.5 The Semantics of the `for` Loop

See Figure 8.1 to see how a `for` loop is executed.

1. Execute the initialisation expression.
2. Evaluate the guard. If it is false then leave the loop and go on to the next statement. If it is true go to 3.
3. Execute the body.
4. Execute the final expression.
5. Go to 2.

8.5.1 Example

```
for(int i=0;i<3;i=i+1) System.out.println("*");
```

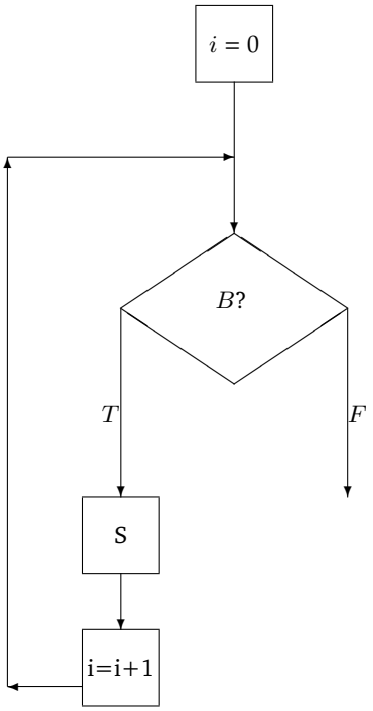

Figure 8.1: The semantics of `for(i=0;B;i=i+1){S}` First, variable i is set to zero. Then guard B is evaluated. If it evaluates to true then the body S of the loop is executed, then the variable i is incremented and B is evaluated again etc. If B evaluates to false we leave the loop and go on to the next statement.

The following steps happen:

- First the initialisation expression is executed. This sets variable `i` to zero.
- Next, the guard, `i<3` is evaluated. This is true since `i` currently has the value zero.
- Because the guard is true, the body of the loop is now executed once. This causes a single asterisk to appear on the screen.
- Execute the final expression. The variable `i` now has the value 1.
- Next, the guard, `i<3` is evaluated. This is true since `i` currently has the value 1.
- Because the guard is true, the body of the loop is now executed once. This causes another asterisk to appear on the screen.
- Execute the final expression. The variable `i` now has the value 2.
- Next, the guard, `i<3` is evaluated. This is true since `i` currently has the value 2.
- Because the guard is true, the body of the loop is now executed once. This causes another asterisk to appear on the screen.
- Execute the final expression. The variable `i` now has the value 3.
- Next, the guard, `i<3` is evaluated. This is false since `i` currently has the value 3, and $(3 < 3)$ is false. So we leave the loop.

So altogether 3 asterisks have been printed.

So what would

```
for(int i=1;i<3;i=i+1) System.out.println("*");
```

do?

8.5.2 Exercises

For each of the following loops, say how many asterisks are printed:

1. `for(int i=0;i<5;i=i+1) System.out.println("*");`
2. `for(int i=1;i<3;i=i+1) System.out.println("*");`
3. `for(int i=-1;i<3;i=i+1) System.out.println("*");`
4. `for(int i=0;i<3;i=i+2) System.out.println("*");`
5. `for(int i=0;i<3000;i=i+2) System.out.println("*");`
6. `for(int i=0;i>=0;i=i+1) System.out.println("*");`
7. `for(int i=m;i>=n;i=i+1) System.out.println("*");`

   You can assume that $n - m > 0$.
8. `for(int i=m;i>=n;i=i+1) System.out.println("*");`

   Where $n - m < 0$.

8.6 Number of Iterations Depending on User Input

So far, all the `for` loops have gone round a **fixed** number of times.
Now consider [`Lecture4/VerticalLineOfStarsNew.java`] This program first asks the user to enter a number. This number is stored in the variable `n`. This variable is then also used in the guard, `i<n`, of the loop. So here, we will go round the loop exactly the number of times that the user entered. Compile and run `Lecture4/VerticalLineOfStarsNew.java`. What happens if you enter -5? Why?

8.6.1 Incrementing and Decrementing Shorthand

`i++` is shorthand for `i=i+1;`.
`i--` is shorthand for `i=i-1;;`

Compile and run [`Lecture4/NumbersUpToNew.java`]

- If you enter 5 what does it do?
- If you enter 0 what does it do?

- If you enter 100 what does it do?
- If you enter -4 what does it do?

8.7 `while` Loops

A `while` loop is another form of loop that is, in fact, simpler than a `for` loop. A `while` loop does not have an initialisation expression or a final expression. It only has a guard and a body.

8.7.1 The Syntax of `while` Loops

A `while` loop consists of the word `while` followed by a boolean expression in round brackets. After this comes the *body* of the `while` loop which is a statement. So the structure of a `while` loop is

```
while (<guard>) <body>
```

8.7.2 The Semantics of a `while` Loop

If you understood how a `for` loop works, then you will have no problem in understanding the semantics of a `while` loop. A `while` loop behaves exactly the same as a `for` loop with an empty initialisation expression and an empty final expression. In Java, there is no need ever to use a `while` loop. We could always use a `for` loop instead. See Figure 8.2 for a further description.

8.7.3 A Program that Goes On for Ever

`[Lecture4/nonTerminate.java]`

8.7.4 Exercise

Explain why this goes on for ever.

8.8 Crude Animations

Consider the program `[LectureElements/animate.java]` This program produces a simple animation. The next frame in the animation is produced by clicking the mouse inside the drawing window.

There is an infinite loop. Inside the loop we draw a black circle. When the mouse is clicked we draw over the circle in white (this erases the circle) and draw a black circle in a different place. This produces the effect of movement.

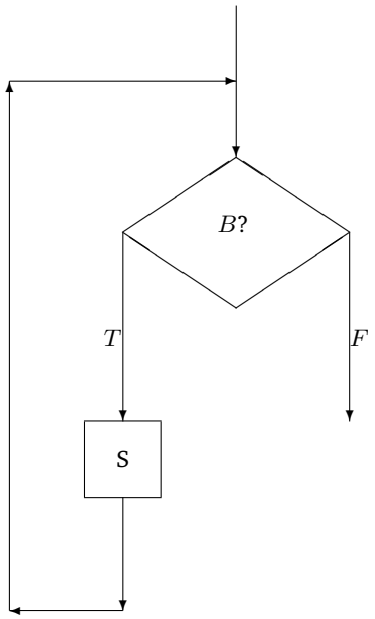

Figure 8.2: The semantics of `while(B){S}` First, the guard B is evaluated. If it evaluates to *true* then the body S of the loop is executed and we go round again. If B evaluates to *false* we leave the loop and go on to the next statement.

Try taking out the call to `d.awaitMouseClick()`. Now it's too fast. Can you think of a way of slowing it down?

8.8.1 Random Animations

Here is a program that produces pretty colours. [`LectureElements/pretty.java`] A colour is defined by three numbers between 0 and 255. To do this animation, we repeatedly define a random colour and draw a random oval at a random place in that colour.

8.9 Exercises on Chapter 8

8.9.1 One to Ten

Change [`Lecture4/NumbersUpToNew.java`] so that if you enter 10, instead of printing out 0 to 9 it prints 1 to 10.

8.9.2 While

Rewrite the program above using a `while` loop.

8.9.3 Non-terminating

Write a non-terminating program using a `for` loop.

8.9.4 Descending Sequence from Ten to One

Write a program, `NumersDownToNew.java`, which is like [`Lecture4/NumbersUpToJava`], but prints the numbers in descending order.

8.9.5 Even Numbers

Write a program, `EvensNew.java`, such that if you enter the number n it outputs the first n even numbers. For example if you enter 5, the output would be:
0
2
4
6
8

8.9.6 Odd Numbers

Write a similar program called `OddsNew.java`. You must be able to read my mind to work out what it does!

8.9.7 Ten Times Table

Write a similar program that does the ten times table as in Section 3.10, question 3.10.2 using a `for` loop.

8.9.8 Multiples of Three

Write a program `MultiplesOfThreeNew.java` such that if you enter the number n it outputs the first n multiples of 3. For example if you enter 4, the output would be
0
3
6
9

8.9.9 Multiples

Write a program `UserEntersMultipleNew.java` where the user enters the multiple and the number. So if the user enters 3 and 7 the output will be
0
7
14
and if the user enters 4 and 101 the output would be
0
101
202
303

8.9.10 Simple Times Table

Write a program `SimpleTimesTable.java`, such that if the user enters two numbers m and n, the program prints out the first m items in the times table for n. For example if the user enters 4 and 5, the program outputs
1 times 5 = 5
2 times 5 = 10
3 times 5 = 15
4 times 5 = 20
and if the user entered 3 and 19 the program outputs
1 times 19 = 19
2 times 19 = 38
3 times 19 = 57

8.9.11 Largest of Ten

Write a program that asks the user to enter 10 integers and then prints out the largest.

8.9.12 Largest (User First Says How Many)

Generalise this so that the program first asks the user how many numbers she is going to enter.

8.9.13 Largest of As Many Numbers as Until Zero is Input

This time, do not ask the user how many numbers they will enter, but allow them to enter as many as they like until they enter a zero. Notice that we have used a boolean variable `more` in the guard of the loop. A common mistake would be to forget to change the value of `more` in the loop. This would cause the loop to go on for ever. Initially this variable is set to `true`. When the user enters zero it is set to `false`. The guard of the loop is the variable, `more`. The program will leave the loop only when `more` is set to false.

8.9.14 A Guessing Game

Write a program that implements the following game: The computer has *thought of* a number (245). The computer says:

> "Try to guess the number I'm thinking of:"

The user must try to guess this number by typing in a number. If the number the user types in is less than the number the computer thought of, it says:

> "too low - guess again:"

and repeats the process. If the number the user types in is greater than the number the computer thought of, it says:

> "too high - guess again:"

and repeats the process. If the user guesses the number correctly, the computer says:

> "Correct! The number of guesses you made was ..."

and the program finishes. This is the only way to finish the program. So if the user never guesses correctly, the game should go on for ever!

8.9.15 Factorial

Write a program that asks the user to enter a number and outputs its factorial. The factorial of n is 1 times 2 times ... times n. The factorial of 0 is 1. What happens if the user enters a negative integer?

8.9.16 Exercise (No Solution)

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. If the computer takes more than ten guesses then the computer loses otherwise the user loses.

8.9.17 Moving Balls

Rewrite `animate.java` so that the ball moves horizontally instead of diagonally.

8.9.18 Random Animation

Make up your own beautiful random animation. It must be unique to you.

8.10 Summary

Having worked on Chapter 8 you will have learned about:

- Programming repetition using loops.
- The syntax of `for` and `while` loops.
- The semantics of `for` and `while` loops.

Chapter 9

More on Calling Methods

9.1 Learning Objectives

Chapter 9 explains:

- the difference between static and non-static methods.
- the difference between the use of void and non-void methods.
- about some methods of the class `java.lang.String`.
- method overloading.
- type-checking.

9.2 Reading

- [Hub04] Chapter 2
- [DD07] pages 203-206
- [Bis01] pages 54-60

9.3 Different Uses of Method Calls

9.3.1 Method Calls as Statements

Often methods are called as statements. An example of this includes

```
System.out.println("hello" + " fred");
```

or

```
d.lineTo(100,100);
```

9.3.2 Method Calls as Expressions

Sometimes calling a method gives a value which we then do something else with. An example of this is

```
int n=in.nextInt();
```

Here `in.nextInt()` is returning a value which we are using in an assignment statement. It would be wrong and meaningless to write

```
int y=System.out.println();
```

This is because `System.out.println()` does not return a value. So there is no value that can be assigned to the variable `y`.

9.3.3 Void and Non-void Methods

A method that can be used as a statement is called a void method. A method that can be used as an expression is called a non-void method. These methods have return type `void`.

9.3.4 More Useful Methods in the Class `java.lang.Math`

```
int max(int,int)
```

This method returns the maximum of two `int`s. We could rewrite the program to find the biggest of three numbers (Section 7.4.3) as [`LectureUsefulMethods/BiggestOfThreeFirstNew.java`] So first we work out the maximum of `x` and `y` and store the result in `a`. We then print out the maximum of `a` and `z`. So we didn't need the conditional statement after all!

In fact, we don't need the variable `a` either:
[`LectureUsefulMethods/BiggestOfThreeNew.java`] To find them maximum of x, y and z, find the maximum of y and z and then find maximum of *that* and x.

We can use this method in the program where the computer guesses a number in Exercise 8.9, Question 8.9.14 to make the computer guess a different value each time. We do this by generating a random (double) number between zero and one using `Math.random()`, multiplying it by 1000 to give a random double between 0 and 1000 and then rounding it to give us a random integer between 0 and 1000. [`LectureUsefulMethods/guessNew.java`]

9.4 Static vs. Instance Methods

Every instance method (i.e. one which doesn't say static in its definition) will be called using dot notation.

If `public int f(int)` is defined in a class called `C` then every call to `f` will be of the form `v.f(e)`, where `v` is an expression of type `C` (i.e. `v` evaluates to an object of type `C`) and `e` is an expression of type `int`.

9.4.1 The Class `java.lang.String`

Look up the Class `java.lang.String` in http://java.sun.com/j2se/1.5.0/docs/api/.

9.4.2 Instances of Objects

Every time we write something like `"hello"` in our program we are creating a new instance of an Object of type `String`. Each instance like `"hello"` has its own collection of instance methods. So, for example there is now a new `"hello".charAt()` method and a new `"hello".length()` method but not a new `valueOf()` method, since that is declared as `static` in `java.lang.String`. So every time we use a `String` of characters with double quotes at either end like `"hello"`, we are not only creating the `String` `"hello"`, but also copies of all the instance methods such as `"hello".charAt()`, etc.

9.4.3 `length()`

The signature of length is `int String.length()`.
The `length()` method can be used to find the length of a `String`. If `s` is a `String`, the expression, `s.length()` will be the length (i.e. the number of characters in the `String` `s`). So `"fred".length()` will return 4 and `"".length()` will return zero. You may ask why don't we write `length(s)` for the length of `String` `s`. This should become clearer later. Consider the program:

[LectureUsefulMethods/LongestOfThreeNew.java]

The user is asked to enter three `String`s and the longest one is printed out.

9.4.4 `charAt()`

The signature of `charAt()` is `char String.charAt(int)`.
The `charAt()` method returns the character at a particular position in a `String`. So `"hello".charAt(0)` gives the character `'h'` and `"hello".charAt(1)` gives the character `'e'` and `"hello".charAt(4)` gives the character `'o'`. Here is a program that asks the user to enter her name and then prints it out backwards. [LectureUsefulMethods/ReverseNew.java]
Notice that here we have written `i--`. This has the same effect as `i=i-1`

9.4.5 `compareTo()`

The signature of `compareTo()` is `int String.compareTo(String)`.
The behaviour of `compareTo()` can be explained by running
[LectureUsefulMethods/AlphabeticOf2New.java]

See how it treats upper case characters and digits.

9.5 Type-Checking

When we call a method, we can call it with *any* expressions as actual parameters, provided they are of the right type. It is very important to understand this. So, for example, `"hello" + " dog"` is a `String` so we can apply the `length()` method to this expression like this:

```
("hello" + " dog").length()
```

This gives us back the `int`, 9. Notice that there is a single space character before the 'd' in "dog". We can therefore use the expression `("hello" + " dog").length()` in any place that an `int` is expected. So for example we can write:

```
("hello" + " dog").length()*5
```

and it will return 45 or

```
Math.max(("hello" + " dog").length(),5).
```

What value does this return? Answer: 9 because the maximum of 9 and 5 is 9.

We can check whether things are right by checking the signatures of the methods we call in the expressions. The compiler does this for us when we compile our programs.

9.6 Parsing `String`s that Represent Integers

9.6.1 `Integer.parseInt()`

The static method, `Integer.parseInt(String s)`, takes a single argument, `s`, which must be a `String` and returns an int whose value is that of the integer represented by `s`. For example, the following program prints out the number 125:

```
class TestParseInt
{
     public static void main(String[] args)
     {
         System.out.println(2 + Integer.parseInt("123"));
     }
}
```

The full signature of the `Integer.parseInt(String s)` method is

```
int Integer.parseInt(String)
```

We can also use a short-cut notation and say that the signature of `Integer.parseInt(String s)` is

```
int (String)
```

There exists another method, `Integer.parseInt` whose signature is

```
int Integer.ParseInt(String s, int i)
```

This method returns the decimal value of the number represented by the `String s` when this string is interpreted as a number in base `i`. For example, the following program prints out the value 38 because the number 123 in base 5 is equal to 38 in base 10:

```
class TestParseInt2
{
    public static void main(String[] args)
    {
       System.out.println(Integer.parseInt("123",5));
    }
}
```

9.7 Method Overloading

When there are two methods with the same name and different signatures (as in the case of `Integer.parseInt`, above) this is known as method **overloading**.

9.8 Exercises on Chapter 9

9.8.1 Exercise

Write a program based on the one in Exercise 7.12.6 which asks the user to enter 3 `Strings` and then prints them in reverse dictionary order. That is, the word that would come nearest the end of the dictionary is printed out first and the one that comes nearest the beginning of the dictionary is printed last.

9.8.2 Exercises – Type Checking

For each expression say whether it type checks correctly and, if possible, give its value. Write a program if necessary.

(i) `Math.abs("hello")`

(ii) `Math.abs("hello".length())`

(iii) `Math.abs("hello".length())+5`

(iv) `"fruit".charAt( Math.abs("hello".length())-3)`

(v) `"boy".compareTo(6)`
.

(vi) `"boy".compareTo("girl")`

(vii) `"boy".compareTo("6")`

(viii) `"boy".compareTo("6")+17`

(ix) `"boy".replace('b',"soup".charAt(0))`

(x) `("boy".replace('b',"soup".charAt(0))).length()`

9.8.3 Trying Methods

(i) Write programs to investigate the behaviour of the following methods:

(a) `String substring(int)`

(b) `String substring(int, int)`

(c) `String replace(char, char)`

(ii) Describe their behaviour.

(iii) Write a program that asks the user to enter a `String` and then outputs the number of occurrences of the character `'a'` in the `String` the user entered. For example, if the user entered `"hello"` the output would be 0, but if the user entered "abracadabra", the answer would be 5.

(iv) The program `LectureUsefulMethods/NumOfasNew.java` produces grammatically incorrect output if the user enters a `String` with exactly one 'a'. It outputs:

> "Your String has 1 occurrences of the character 'a' "

Correct this.

9.8.4 Integer Methods

For each method in the class `Integer` give its signature.

9.8.5 Dictionary Order

Write a program based on `LectureUsefulMethods/DescendingThree.java` which asks the user to enter 3 `String`s and then prints them in dictionary order. That is, the word that comes earliest in the dictionary is printed out first and the one that comes latest in the dictionary is printed last. An added complication of your program is that it should not distinguish between upper and lower case letters, i.e. “dog” and “Dog” should be considered the same as far as comparison is concerned. Convert everything to lower case before comparing. (See `toLowerCase()` in `java.lang.String`.) Big hint: `(x.toLowerCase()).compareTo(y.toLowerCase())`

9.9 Summary

Having worked on Chapter 9 you will have:

- Understood the difference between static and non-static methods.
- Understood the difference between the use of void and non-void methods.
- Been introduced to some methods of the class `java.lang.String`.
- Been introduced to method overloading.
- Understood type-checking.

Chapter 10

One-Dimensional Arrays

10.1 Learning Objectives

Chapter 10 explains:

- how to declare and initialise an array.
- how to used`for` loops to process arrays.
- *array index out of bounds* exceptions.

10.2 Reading

- [Hub04] pages 188-195
- [DD07] Chapter 7
- [Bis01] Chapter 6
- [CK06] Chapter 5

10.3 Introduction

An array is a "place" in the memory of the computer to store lots of data items, all of the same type. For example, remember the program [`Lecture4/LargestOfTenNew.java`] This programs ask the user to enter ten numbers and then prints out the largest. Once a new number is read in, the previous number is lost because the input is achieved through a single variable.

Suppose we wanted to store all the numbers the user entered and then print them out in the opposite order to which they were entered. Without arrays, this could be achieved using ten separate variables. A better alternative is to store the numbers in an array:

[`LectureArrays1D/ReverseTenNew.java`]

First we declare an array variable called `a`:

```
int [ ] a;
```

We then give it space to hold ten `int`s by:

```
a = new int[10];
```

The array a now looks like this:

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

The ten elements of the array, a are called a[0],a[1],a[2] up to a[9]. The first element of a is called a[0]. If we wanted to put 5 into the third element of the array a we would write

```
a[2]=5;
```

The array would then look like this:

		5							
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

We could also have written

```
a[1+1]=5;
```

or

```
int i=2;
a[i]=5;
```

Consider the following program:

[LectureArrays1D/Array1.java] What does it output?
Answer:

```
0
1
2
3
4
```

num is declared to be an array of 5 ints. The first loop has the following effect:

0	1	2	3	4
num[0]	num[1]	num[2]	num[3]	num[4]

If we wanted to fill the whole array up with sevens we could write:

```
for (int i=0;i<10;i++) num[i]=7;
```

The array a would then look like this:

7	7	7	7	7	7	7	7	7	7
num[0]	num[1]	num[2]	num[3]	num[4]	num[5]	num[6]	num[7]	num[8]	num[9]

If we wanted to populate the array num with ten values entered by the user we would write:

```
for(int i=0;i<10;i++)
{
    num[i]=in.nextInt();
}
```

The first time round the loop, the variable `i` has the value 0 and each time round the loop, one is added to `i`. The final value of `i` inside the loop is 9 (it is 10 when the program leaves the loop). This has the same effect as:

```
num[0]=in.nextInt();
num[1]=in.nextInt();
num[2]=in.nextInt();
num[3]=in.nextInt();
num[4]=in.nextInt();
num[5]=in.nextInt();
num[6]=in.nextInt();
num[7]=in.nextInt();
num[8]=in.nextInt();
num[9]=in.nextInt();
```

If we then wanted to print out the values input by the user in the reverse order that they were typed in, we would simply print first `num[9]` then `num[8]` and,. . . , lastly `num[0]`, like this:

```
for(int i=9;i>=0;i--) System.out.println(num[i]);
```

The first time round the loop the variable `i` has the value 9. Each time round the loop, one is subtracted from i. The last value that `i` has in the loop is zero. So, this has the same effect as:

```
System.out.println(num[9]);
System.out.println(num[8]);
System.out.println(num[7]);
System.out.println(num[6]);
System.out.println(num[5]);
System.out.println(num[4]);
System.out.println(num[3]);
System.out.println(num[2]);
System.out.println(num[1]);
System.out.println(num[0]);
```

10.4 Array Index Out of Bounds

Try running `[LectureArrays1D/OutOfBounds.java]` It gives the error: The reason for this is that `num` is defined here to have length 2. The error is that there is no such element of the array `num` as `num[79]`. Any other value apart from 0 and 1 will give this error. So `num[712]` will give the same error, but `num[1]` will be OK. Note that this error is not detected by the compiler, but occurs only when we run the program. It is called a *run time error*. Another run time error that we have already come across is *division by zero*.

10.4.1 Array Limitations

Remember the program `Lecture4/LargestEndWithZeroNew` where we carry on allowing the user to enter as many values as she likes until she enters zero. We could try to do this using an array: `[LectureArrays1D/LargestEndWithZeroNew.java]` but this does not work, because if the user enters more than 1000 numbers the program will crash. Notice that we have used the final value of `i` to tell us exactly how many numbers were entered. Also, notice that in order to do this we had to declare `i` *globally* i.e. not inside the loop. This will be explained in greater detail in Volume 2.

10.5 Exercises on Chapter 10

10.5.1 Reverse

Write a program similar to `LectureArrays1D/ReverseTenNew.java` but first ask the user how many numbers she will enter. If `n` is the number of ints the user will enter then our array must be big enough to hold `n` `int`s and the program must go round each loop `n` times.

10.5.2 Largest

Remember the program [`Lecture4/LargestNew.java`] Rewrite it using an array.

10.5.3 Crash

Alter [`LectureArrays1D/CrashNew.java`] to see if you can make it have a *array index out of bounds* error! When a program has a run time error like this we say

> "the program has crashed".

10.5.4 Bad Input

In [`LectureArrays1D/Input1New.java`] if the user enters `0`, the program does not behave well. Correct it.

10.5.5 Array Largest, Smallest, Sum and Average

Write a program which reads some numbers into an array and prints out:

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

Could we do this without an array?

10.5.6 Backwards

Write a program that asks the user to enter some numbers (first the user says how many), which 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`.

10.5.7 Occurrences

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`.

10.5.8 Longest `String`

Write a program where the user types in a number of `Strings` which are stored in an array of `Strings` and then the program prints out all the longest `Strings` entered by the user.

10.5.9 Exercise (No Solution)

Write a program where the user types in a number of `Strings` stored in an array of `Strings` and then the program prints out all the `Strings` that have the most occurrences of the character 'a'. answer: This one is for you to struggle with!

10.6 Summary

Having worked on Chapter 10 you will have:

- Learned how to declare and initialise an array.
- Used `for` loops to process arrays.
- Learned about *array index out of bounds* exceptions.

Chapter 11

Nested Loops

11.1 Learning Objectives

Chapter 11 explains:

- how to solve problems using loops within loops.

11.2 Reading

- [Hub04] pages 78-88
- [DD07] page 280

11.3 Squares and Rectangles

As we said earlier, the body of a loop can be any Java statement. It is quite usual for the body of a loop to contain another loop. Consider [`Lecture5/FullSquareOfStarsNew.java`] In this program there is an outer loop:

```
for(int j=0;j<x;j++)
{
                    ...
}
```

The body of this loop contains two statements, a `for` loop (the inner loop) and a `println` statement. The body of the outer loop will be executed `x` times where `x` is the value typed in by the user. The body of the inner loop prints a horizontal line of `x` stars and then moves the cursor onto the next line (`System.out.println()`). So, the outer loop will print `x` horizontal lines each containing `x` stars. In other words, it will draw a *square*[1] of stars. The reason that it is a square rather than a rectangle is that the same value `x` is used to define the number of times to go round the outer loop and also the number of times round the inner loop.

[1] We call it a *square* because the number of horizontal and vertical stars is the same. In fact because the horizontal and vertical spacing between the stars is not the same, it will not look like a square because the true height and width are different.

11.3.1 Exercise

A trace of the values of `i` and `j` as the program is executed if the user enters 3 is given below:

j	i
0	
0	0
	1
	2
	3
1	
	0
	1
	2
	3
2	
	0
	1
	2
	3
3	

If these two values were different, then a rectangle would be printed:

[Lecture5/FullRectangleOfStarsNew.java]

Here, the user enters two values, the height `y`, and the width `x`. Clearly the number of times round the outer loop will govern the height of the shape and the number of times round the inner loop will govern the width. The guard of the outer loop should depend on `y` and the guard of the inner loop should depend on `x`.

11.3.2 Exercise

Trace the values of `i` and `j` as the program is executed if the user enters 2 for the height and 3 for the width.

11.4 Non-rectangular Shapes

We now further investigate loops within loops using shapes. We will only get a rectangle if the number of times gone round the inner loop is constant. If we vary the number of times round the inner loop, we will not get a rectangle. For example to get a right-angled triangle with the right angle at the bottom left:

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

the number of times through the inner loop starts at one and goes up by one each time. A program that achieves this is

[Lecture5/LeftBottomTriangleNew.java]

Notice that the number of times round the inner loop depends on `j`, the outer loop counter; that is, the first time we go once round the inner loop, the second time twice, and so on. If we trace the values of `i` and `j` when the user enters 3 we will get:

j	i
1	
1	0
	1
2	
	0
	1
	2
3	
	0
	1
	2
	3
4	

11.4.1 Exercise: Left Top Triangles

Write a program that prints this

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

if the user enters 4, and

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

if the user enters 5, etc. In this case, the number of times to go round the inner loop starts at x and goes down by one each time. So, the jth time round the inner loop we want to go round it $x - j$ times, for all values of j from 0 to $x - 1$. Study the differences between this program and `Lecture5/LeftBottomTriangleNew.java`. The only difference is the guard of the inner loop.

11.4.2 Exercise: Right Top Triangles

Write a program that prints this

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

if the user enters 4 and

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

if the user enters 5, etc.

In this case, the number of stars on each line is identical to the number of stars on each line in the previous program `Lecture5/LeftTopTriangleNew.java`. The difference is that on each line we have to print some spaces before we start doing stars. The zeroth line has zero spaces before the stars, the first line has one space before the stars, the second line has two spaces before the stars, etc. The program goes once round the outer loop for each horizontal line that is drawn. The outer loop will contain two inner loops: the first to do the spaces and the second to do the stars. For the jth time round the outer loop we want j spaces and $x - j$ stars.

11.4.3 Exercise: Hollow Squares

Write a program that does this

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

if the user enters 4 and

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

if the user enters 5 etc.

This time, the zeroth and last line (the $(x - 1)$st) are different from all the others. They are complete lines of x stars. All the other lines, from the first to the $(x - 2)$nd, have a single star followed by $x - 2$ spaces followed by another single star. Another, neater way of thinking of the problem is that if we are at the boundary of the square we print a star and otherwise we print a space. We are at the boundary of the square, at the zeroth and last time of the outer loop and the zeroth and last times of the inner loop. Here, the body of the inner loop first contains a conditional statement. The guard of this conditional is

```
(i==0 || i==x-1 || j==0 || j==x-1)
```

Remembering that `||` means or, this condition is `true` if we are at the boundary of the square and `false` otherwise.

11.5 Exercises on Chapter 11

Write the following programs:

11.5.1 RightBottomTriangleOfStars

11.5.2 HollowRectangleOfStars

11.5.3 HollowLeftBottomTriangleOfStars

11.5.4 HollowLeftTopTriangleOfStars

11.5.5 HollowRightTopTriangleOfStars

11.5.6 HollowBottomRightTriangleOfStars

11.5.7 Producing Multiplication Tables

Write a program to print out all the times tables. The user should enter how many tables and also how many to go up to for each table. For example:

```
How Many Numbers for each table? 7
Up to which times table 3
1 times 1 = 1
2 times 1 = 2
3 times 1 = 3
4 times 1 = 4
5 times 1 = 5
6 times 1 = 6
7 times 1 = 7

1 times 2 = 2
2 times 2 = 4
3 times 2 = 6
4 times 2 = 8
5 times 2 = 10
6 times 2 = 12
7 times 2 = 14

1 times 3 = 3
2 times 3 = 6
3 times 3 = 9
4 times 3 = 12
5 times 3 = 15
6 times 3 = 18
7 times 3 = 21
```

11.5.8 Multiplication and Exponentiation in Terms of Addition

11.5.9 Multiplication in Terms of Addition

Suppose Java didn't have multiplication. Write a program that asks the user to enter two whole integers m and n and then outputs m times n. Your program should use a loop and not contain a multiplication sign, `*`, anywhere!

11.5.10 Exponentiation in Terms of Addition

Suppose Java didn't have multiplication. Write a program that asks the user to enter two whole numbers m and n, and outputs m^n (m to the power of n). Your program should use a loop within a loop.

11.5.11 A Clock Animation

Here is a program that emulates the big hand of a clock: [`LectureElements/clock.java`] Notice, again, how we rub out the previous line and redraw to give the effect of movement. Click the mouse to move on to the next minute. Change it to produce an animation that has both hands of the clock. You will have to use a nested loop for this. Be careful when the hands are on top of each other. You will have to redraw the hour hand after you've moved the minute hand. The reason we have to write expressions like

```
(int)Math.round(origX+bighandSize*Math.cos(angle))
```

is because `lineTo` has `int` parameters. The expression `origX+bighandSize*Math.cos(angle)` is of type `double`, so we need to use `Math.round` to round it up to the nearest `int` in order to make it of the right type for `lineTo`. Java is very fussy. `Math.round` does not have a return type of `int`. It has a return type of `long`. So we need to write `(int)` at the beginning to tell Java to think of the `long` value as an `int`. This is called `type casting`.

11.6 Summary

Having worked on Chapter 11 you will have:

- Learned how to solve problems using loops within loops.

Chapter 12

Defining Your Own Methods

12.1 Learning Objectives

Chapter 12 explains:

- why methods are useful.
- how to define void static methods.
- how to use parameters.
- how to define non-void static methods.
- more about return types.

12.2 Reading

- [CK06] Chapter 4
- [Bis01] Chapter 3
- [LO02] Chapter 12
- [AW01] Chapter 11
- [DD07] Chapters 4 and 6
- [Hub04] 5.4

12.3 Introduction

You are now going to learn how to write your own methods. Successfully writing and using methods is one of the keys to being able to program effectively. As we have already seen, methods are useful collections of statements that can be *called* over and over again. The usefulness of methods has already been illustrated in Chapters 5 and 9.

We refer again to the shapes that we were drawing in Chapter 11. We often used loops to draw horizontal lines of different numbers of stars.

The first method we are going to define will be for drawing such a line of stars. The method `Lines.LineOfStars` will have one `int` parameter, which enables us to use the method to draw horizontal lines of different numbers of stars. We refer to the method as `Lines.LineOfStars` because the class in which it is defined is called `Lines` and the method's name is `LineOfStars`. If the method is defined in the class that we are calling it from, then we could refer to it simply as `LineOfStars`.

12.3.1 The Purpose of Parameters

Parameters allow us to use the same general method to perform different tasks.

`Lines.LineOfStars(5)` will draw 5 stars, `Lines.LineOfStars(52)` will draw 52 stars and `Lines.LineOfStars(1000)` will draw 1000 stars in a line etc.

Our program to draw the square is now [`Lecture6/FullSquareOfStarsNew.java`] We have put the `LineOfStars` method definition and other definitions in a different file called `Lines.java` and which looks like this: [`Lecture6/Lines.java`]

`Lines.java` contains three separate method definitions:

1. `LineOfStars`
2. `HollowLineOfStars`
3. `LineOfSpaces`

When we call our method `LineOfStars` we also have to tell the Java system where to find it. We say `Lines.LineOfStars` because we have put our method in a class called `Lines` . For convenience we have put our method in a file called `Lines.java`. We will put all useful methods for drawing shapes in this file.

12.4 The Structure of a Method Definition

Let us study the code for the definition of `LineOfStars` in a bit more detail. The first line:

12.4.1 The Method Heading

```
static void LineOfStars(int n)
```

is called the heading of the method. It gives us the signature of the method and tells us whether it is static or not. The return type `void` means it doesn't return anything: it just *does* something (i.e. it draws a line of stars).

The Formal Parameters

The method heading tells us that this method has one *formal parameter* called `n` of type `int`. Later we will see methods with more than one parameter. Methods do not have to have parameters. In this case, there will be nothing written between the brackets at the end of the heading.

12.4.2 The Body of the Method

Everything apart from the first line is called the *body* of the method. The body looks very familiar; it is just the same old loop for drawing some stars in a horizontal line.

How to Use the Parameters

But how many stars does it do? How many times does it go round the loop? Clearly it goes `n` times round the loop. It is very important that we have used the same identifier `n` both for the formal parameter and for the boolean in the guard of the loop. This is why calling the method with different values will give different results. When we call the method by saying `Lines.LineOfStars(5)` it is as if the *formal* parameter `n` is replaced by the *actual* parameter 5 and then the body of the method is executed (calling methods has already been discussed in Section 5.5 Chapter 5). So in this case it is as if the body is

```
for (int i=0;i<5;i++)System.out.print("*");
```

The `n` has been replaced by 5, so 5 stars will be drawn.

12.5 Calling Methods

Having defined our `LineOfStars` method we can now use it. [`Lecture6/SimpleCallNew.java`] There are four calls to the method `Lines.LineOfStars`, each with a different actual parameter. Notice that the actual parameter can be any expression that evaluates to an `int`. This is because in the method heading, the formal parameter was declared to be of type *int*. Having defined a method, it can be called as often as we require.

The output of this program is:

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

We can also use the method to draw rectangles of stars:

[`Lecture6/FullRectangleOfStarsNew.java`]

12.6 Hollow Lines of Stars

Let us rethink how to draw a hollow shape, say, for example, a hollow square. We always have a loop where each time round the loop we draw a horizontal line of the hollow shape. The zeroth and last time round the loop, we will draw a solid line of stars and all the other times round the loop we draw a *hollow line* of stars. A hollow line consists of a star at either end with spaces in between. We can write a method for drawing a hollow line (of stars) like this:

```
static void HollowLineOfStars(int n)
{
   for (int i=0;i<n;i++)
   {
      if (i==0 || i==n-1) System.out.print("*");
      else System.out.print(" ");
   }
}
```

Notice, this method is already in `Lines.java`. So the program to draw the hollow square of stars is:

[Lecture6/HollowSquareOfStarsNew.java]

The zeroth and last time round the loop we call `Lines.LineOfStars` and all the other times we call `Lines.HollowLineOfStars`.

12.7 Some Nice Things about Methods

12.7.1 Readability

Using methods makes our programs more readable and understandable, both by us and other people. This is only true if we use methods sensibly and give them sensible names that reflect what they actually do.

12.7.2 Reusability

If we have defined a useful method then it will be used both by us and other people over and over again.

12.7.3 Breaking Down Problems into Smaller Ones

If we *think* using methods, then we can describe a big problem in terms of slightly smaller problems each of which we can break down further.

12.8 Right Triangles Using Methods

In order to write methods to print triangles with their right angle on the right, we need to print spaces before we start doing stars. So we add a method for drawing n spaces to our ever growing portfolio of methods in `LinesNew.java`.

```
static void LineOfSpaces(int n)
{
  for (int i=0;i<n;i++)System.out.print(" ");
}
```

12.9 Methods Calling Other Methods

In order to break problems down into smaller and smaller ones it is usual for method A to call method B and method B to call method C etc. The following exercise illustrates this.

12.9.1 Easy Exercise

What is the output of [Lecture6/Easy1New.java]

Methods Calling Other Methods

We can make each shape into a method and put them in a class called Shapes: [Lecture6/Shapes.java]

Notice how FullSquareOfStars is defined in terms of FullRectangleOfStars. A square is simply a rectangle where the width and height are the same.

12.10 The Names of Formal Parameters

Although methods f and g, below, have different names for their formal parameters, there is absolutely no difference in behaviour between methods them.

```
static void f(int n)
{
  System.out.print(n+1);
}


static void g(int banana)
{
  System.out.print(banana+1);
}
```

Further Example

The calls f(5); and g(5); will both print 6 . As will the program fragments:

```
int apple=5;
f(apple);

int apple=5;
g(apple);

int n=5;
f(n);

int n=5;
g(n);

int banana=5;
f(banana);
```

```
int banana=5;
g(banana);
```

The names of formal parameters are usually chosen to be as suggestive of their use as possible.

Question

What will the following program fragment output?

```
int banana=5;
g(banana);
System.out.println(banana);
```

Answer:

```
65
```

The call to g does not affect the value of its parameter.

12.11 Hard Exercise: Drawing a Hollow Diamond

- If the user enters 0 the program should do nothing.
- If the user enters 1 the program should print

```
*
```

- If the user enters 2 the program should print

```
 *
* *
 *
```

- If the user enters 3 the program should print

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

- If the user enters 4 the program should print

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

  and so on. (Work out the formulas and translate into loops.)

12.12 Non-void Static Methods

So far all the methods we have written have return type `void`. Their headings have all started with

```
static void
```

This means that they don't return a value. In this section we will look at methods that return values (i.e. their return type is not `void`). Consider [`Lecture6/Easy2New.java`]

Here, we have declared a method called `Bake`. Its signature is `String Bake(String)`. This means it has one parameter of type `String` and its return type is `String`. This means that we can put a call to `Bake` in our program wherever we would use a `String`.

This program prints

```
fish pie
```

Why? Simply because `Bake("fish")` returns the `String`, `"fish pie"` which we then print out.

Consider [`Lecture6/Easy3New.java`] This program does nothing. There is no print statement so when we execute the program we will not see anything although the `Bake` method is called.

In `Bake`, we have used a `return` statement. The `return` statement does two things.

1. It tells Java what value to return after the method is called.
2. When the `return` statement is encountered, the system leaves the method containing it and goes back to the place where the method was called from. Any code not yet reached in a method after the `return` statement will not get executed in this call.

12.13 Differences between Void and Non-void Methods

The main difference between void and non-void methods is that calls to non-void methods can be used in place of *expressions* and void methods can't. Calls to both can be used in place of statements. If a call to a non-void method is used in place of a statement, then the return value of the method is simply ignored. (in `Lecture6/Easy3New.java` above.)

It is very important that the reader understands what is an expression and what is a statement. An easy way to remember is that:

- **an expression is anything that can go on the right hand side (after the `=`) in an assignment statement.**
- **A statement is anything that can go in place of the dots in the conditional `if (condition) ... else`.**

In general, a statement does something and an expression returns a value (although an expression can also do something as a side-effect.)

Consider

[LectureNonVoidMethods/FactorialNew.java]

Example - factorial

The method, `factorial`, is a non-void method that has the signature `int factorial (int)`. In other words it takes an `int` and returns an `int`. If we have programmed it properly it will return the factorial of the `int` we give it. We use a call to factorial in the print statement so it prints out the factorial of the input in the correct place.

12.14 The Clock Using Methods

Remember the Clock program of Chapter 11: [LectureElements/clock.java] We could have simplified it a lot if we had known how to define our own methods at the time: [LectureElements/clockMeths.java] We have introduced three methods: `makeLine`, `drawLine` and `eraseLine`. The method, `makeLine`, does all the 'hard work'. The other two methods both call `makeLine`. The only difference between the two is that one draws the line and the other erases it. The important thing to note is how easy and self-explanatory the main method now becomes. We could guess what it was doing even if we didn't understand how the other methods work.

Having defined these methods, the program to animate the two-handed clock based on `LectureElements/bigClock.java` will now be much much easier, so it is left as a challenge to the reader.

Most programming problems become a lot easier once you have mastered how to define and use methods. We strongly suggest that you rewrite the solutions to all the challenging problems that you have tackled so far using methods. From now on, you should always define your own methods wherever you think they are necessary. This is *the* art of programming.

12.15 Exercises on Chapter 12

12.15.1 Left Bottom Triangle of Stars

Rewrite the program `Lecture5/LeftBottomTriangleNew.java` so it uses `Lines.LineOfStars`.

12.15.2 Left Top Triangle of Stars

Rewrite the program `Lecture5/LeftTopTriangleNew.java` so it uses `Lines.LineOfStars`.

12.15.3 Hollow Left Bottom Triangle

Rewrite the program to draw a Hollow Left Bottom Triangle of Stars using methods.

12.15.4 Hollow Left Top Triangle

Rewrite the program to draw a Hollow Left Top Triangle of Stars using methods.

12.15.5 Right Bottom Triangle

Rewrite the program to draw a Right Bottom Triangle of Stars using methods.

12.15.6 Right Top Triangle

Rewrite the program to draw a Right Top Triangle of Stars using methods.

12.15.7 Hollow Right Bottom Triangle

Rewrite the program to draw a Hollow Right Bottom Triangle of Stars using methods.

12.15.8 Hollow Right Top Triangle

Rewrite the program to draw a Hollow Right Bottom Triangle of Stars using methods.

12.15.9 Complete `Shapes`

Add static methods for all the other hollow shapes to the class `Shapes`.

12.15.10 Use Shapes

Write a program using shapes that asks the user to enter an integer and then draws a solid bottom right triangle of the size they enter.

12.15.11 Re-do Times Table Exercise

Redo the question in Section 11.5.7 using methods.

12.15.12 Addall

Write a program which asks the user to enter a number n, say, and it outputs $1 + \cdots + n$. For example, if the user entered 4, the output would be 10. Another possible answer you may give if you know the formula is

`[LectureNonVoidMethods/Addall1New.java]`

12.15.13 Array Methods

Write non void methods to work out:

1. The sum of all the elements in an array
2. The largest of all the elements in an array
3. The smallest of all the elements in an array
4. The average of all the elements in an array

Put all the methods in a class called `ArrayUseful`.

12.15.14 Rewrite Array Sum Average etc.

Rewrite the exercise in Section 10.5.5 using the methods you have just defined.

12.15.15 Rewrite Array Assignment

Remember the exercise in (Section 10.5.9): Write a program where the user types in a number of `Strings` stored in an array of `Strings` and then the program prints out all the `Strings` that have the most occurrences of the character 'a'. This is much easier using methods.

12.15.16 Mult in Terms of Add

Redo Exercise 11.5.8 using methods.

12.15.17 Power in Terms of Add

Redo Exercise 11.5.10 using methods.

12.16 Summary

Being able to define and use methods is a key to successful programming. Having worked on Chapter 12 you will have:

- Understood why methods are useful.
- Learned how to define void static methods.
- Learned how to use parameters.
- Learned how to define non-void static methods.
- Learned about return types.

Chapter 13

Conclusion

You have now come to the end of the first volume of the Java Subject Guide. By now you should be familiar with a lot of the basic concepts of programming.

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

In the second volume, we will 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

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`] 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, 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]` 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`] 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

Multiple Choice Questions

QUESTION 1

Consider the following program:

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

Which of the following statements is correct?

(a) The program prints `hello`

(b) The program waits forever for the user to input something and then outputs whatever they typed in.

(c) The program waits forever for the user to input something and then gives an error message.

(d) The program prints `s`

(e) None of the above.

QUESTION 2

Consider the following program:

```
import java.util.Scanner;
public class Add1
{
   public static void main(String[] args)
   {
    Scanner in =new Scanner(System.in)
    String s =in.nextLine();
    System.out.println(x+1);
   }
}
```

This program has a compilation error.
The error is:

(a) Missing '}'

(b) Variable `x` is not declared

(c) Undefined variable, class, or package name: `System.out.println(x+1);`

(d) Undeclared variable: `s`

(e) None of the above.

QUESTION 3

Consider the following program:

```
public class A
{
  public static void main(String[] args)
  {
   System.out.println("35" + 1);
  }
}
```

Which of the following statements is correct?

(a) This program crashes
(b) This program compiles correctly and outputs nothing
(c) This program compiles correctly and outputs 36
(d) This program compiles correctly and outputs 351
(e) None of the above.

QUESTION 4

Consider the following program:

```
import java.util.Scanner;
public class B
{
  public static void main(String[] args)
  {
   Scanner in =new Scanner(System.in);
   int x=in.nextInt();
   System.out.println("Finished");
  }
}
```

Which of the following statements is correct?

(a) The program will print `hello`
(b) If the user inputs 23 the program will crash.
(c) If the user inputs `hello` the program will crash.
(d) If the user inputs `hello` the program will print `Finished`
(e) None of the above.

QUESTION 5

What is the value of the following expression: `(1<2 && (2<1 || 3==4))`

(a) 1
(b) true
(c) 2
(d) false
(e) None of the above.

QUESTION 6

Consider the following statement:
`boolean b = (3==4);`
Which of the following is true?

(a) This produces a compilation error
(b) The value of `b` will be true
(c) The value of `b` will be false
(d) The value of `b` will be 3
(e) The value of `b` will be 4

QUESTION 7

Consider the following statement:
`boolean b=3=4;`
Which of the following statements is true?

(a) This produces a compilation error
(b) The value of `b` will be true
(c) The value of `b` will be false
(d) The value of `b` will be 3
(e) The value of `b` will be 4

QUESTION 8

Consider the following program:

```
import java.util.Scanner;
public class C
{
   public static void main(String[] args)
   {
    Scanner in =new Scanner(System.in);
    System.out.print("Enter Number>");
    int x=in.readInt();
    for(int i=1;i<x;i++) System.out.print("*");
    System.out.println();
   }
}
```

Which of the following best describes its behaviour?

(a) A horizontal line of n stars will be displayed, where n is the number entered by the user.

(b) A vertical line of n stars will be displayed, where n is the number entered by the user.

(c) A vertical line of $n + 1$ stars will be displayed, where n is the number entered by the user.

(d) A horizontal line of $n - 1$ stars will be displayed, where n is the number entered by the user.

(e) None of the above.

QUESTION 9

Consider the following program:

```
import java.util.Scanner;
public class D
{
   public static void main(String[] args)
   {
    Scanner in =new Scanner(System.in);
    System.out.print("Enter Number>");
    int x=in.nextInt();
    int t=0;
    for(int i=0;i<x;i++) t=t+i;
    System.out.println(t);
   }
}
```

The above program has no compilation errors. Which of the following describes its behaviour when the user enters 3?

(a) The program will output

```
1
2
3
```

(b) The program will output

```
0
1
2
```

(c) The program will output

```
0
1
```

(d) The program will output

```
6
```

(e) None of the above.

QUESTION 10

Consider the following program:

```
import java.util.Scanner;
public class D
{
   public static void main(String[] args)
   {
    Scanner in =new Scanner(System.in);
    System.out.print("Enter Number>");
    int x=in.nextInt();
    int t=0;
    if (t==0 && t==3) System.out.println("hello");
   }
}
```

Which of the following describes its behaviour when the user enters 3?

(a) The program will output `hello`

(b) The program will output nothing at all

(c) The program will output `0`

(d) The program will not compile

(e) None of the above.

QUESTION 11

Consider the following program:

```
public class T
{
   public static void main(String[] args)
   { x=4;
     for(int j=1;j<=x;j++)
     {  for(int i=0;i<ZZZ;i++) System.out.print("*");
        System.out.println();
     }
   }
}
```

What expression should we replace ZZZ with so that the program outputs

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

(a) i

(b) j

(c) j+1

(d) 4

(e) None of the above.

QUESTION 12

Consider the method p.

```
static void p(int n)
{System.out.print(n+1);
}
```

Which of the following is a legal call to p?

(a) `p();`

(b) `p(3+1);`

(c) `p(3,2);`

(d) `p("hello")`

(e) None of the above.

QUESTION 13

What is the output of the following program?

```
public class F
{
  static void L(int n)
  {for (int i=0;i<n;i++)System.out.print("*");
  }

  public static void main(String[] args)
  {
   L(4);
  }
}
```

(a) `***`

(b) `4`

(c) It will not compile

(d)
```
****
****
****
****
```

(e) None of the above.

QUESTION 14

Consider the following program:

```
class Array1
{
   public static void main(String[] args)
   {
    int[] num =new int[3];
    num[0]=1; num[1]=1; num[2]=2;
   }
}
```

Which one of the following statements is correct?

(a) The program will not compile

(b) The program will compile but when it is run it will crash with an 'array out of bounds' exception.

(c) The program will compile and run but output nothing.

(d) The program will compile and run and output `0 1 2`

(e) None of the above.

QUESTION 15

Consider the following program:

```
class Array2
{
   public static void main(String[] args)
   {
    int[] num =new int[2];
    num[0]=1; num[1]=1; num[2]=2;
   }
}
```

Which one of the following statements is correct?

(a) The program will not compile

(b) The program will compile but when it is run it will crash with an 'array out of bounds' exception.

(c) The program will compile and run but output nothing.

(d) The program will compile and run and output `0 1 2`

(e) None of the above.

QUESTION 16

Consider the following program:

```
class Array3
{
    public static void main(String[] args)
   {  int k=5;
     int[] num= new int[k];
     for(int i=0;i<k;i++)num[i]=i+1;
     for(int i=0;i<k;i++)System.out.println(num[i]);
   }
}
```

Which of the following statements is correct?

(a) The program will not compile

(b) The program will compile but when it is run it will crash with an 'array out of bounds' exception

(c) The program will compile and run but output nothing

(d) The program will compile and run and output

```
0
1
2
3
4
5
```

(e) None of the above.

Appendix C

Reading List

[AW01] David Arnow and Gerald Weiss. *Introduction to Programming using Java*. Addison-Wesley, 2001.

[BB99] Duane A. Bailey and Duane W. Bailey. *Java Elements*. McGraw–Hill International Editions, October 1999. http://www.cs.williams.edu/~bailey/JavaElements/.

[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.

[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.

[LO02] Kenneth A. Lambert and Martin Osborne. *Java - A Framework for Programming and Problem Solving*. Brookes-Cole, 2002.