How to Program Using Java

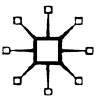
Tony Jenkins
Graham Hardman

How to Program Using Java

Tony Jenkins
Graham Hardman

Illustrations by Christine Jopling

palgrave
macmillan



First published 2004 by
PALGRAVE MACMILLAN
Houndmills, Basingstoke, Hampshire RG21 6XS and
175 Fifth Avenue, New York, N.Y. 10010
Companies and representatives throughout the world

PALGRAVE MACMILLAN is the global academic imprint of the Palgrave Macmillan division of St. Martin's Press, LLC and of Palgrave Macmillan Ltd. Macmillan® is a registered trademark in the United States, United Kingdom and other countries. Palgrave is a registered trademark in the European Union and other countries.

ISBN 978-1-4039-1223-7 ISBN 978-0-230-80243-8 (eBook)

DOI 10.1007/978-0-230-80243-8

This book is printed on paper suitable for recycling and made from fully managed and sustained forest sources.

A catalogue record for this book is available from the British Library.

10 9 8 7 6 5 4 3 2 1
13 12 11 10 09 08 07 06 05 04

This book is respectfully dedicated to all SysAdmins.
Now can we have some more disk quota, please?

Using this book

Deciding what to read

This book is meant to be read by someone who is learning to program. It is not meant to be a reference. The first chapter explains what's in each of the following chapters in some detail, but briefly:

If you're approaching programming as a complete novice you should aim to work through these chapters in this order. Don't be tempted to skip straight to the chapters with Java in! It's extremely important that you understand what you're trying to achieve and the best ways of achieving that before you go anywhere near any programs. This might seem odd, but please bear with us!

If you've already programmed in some other language (particularly something like Pascal or C, or definitely if you've met C++) and just want a flavour of Java you're probably safe to skip on to Chapter 6 or 7. It would still be a good plan to skim through the earlier chapters, though.

At the end of the book you'll find a quick Java Reference, a Glossary and an Index. This is where to look when you realise that you need that little bit of information but can't remember where it was.

Understanding what you read

There are some conventions that you need to keep in mind as you read.

In the text, anything in `this font` is a Java statement or name. Anything in this font is not.

All programs and classes also appear in `this font`. Like this:

```
/*
  Duck.java
  A simple Duck class.

  AMJ
  22nd January 2003
*/
public class Duck
{
  private String name;
```

```
    public Duck ()
    {
    }
}
```

Anything not in `that font` is not a program. Fragments of programs also appear in `this font`:

```
System.out.println ("Quack");
```

Anything in `this font` is correct Java.

Where a user is entering a value, the user's typing is shown in **`bold`**:

```
Enter your name: Tony
```

Sometimes it is necessary to show just the general format of the Java statement. This appears like this:

format of a Java statement

For example:

<type> <identifier>;

Anything that appears between < and > in these examples is a description of what is required in the program. Examples presented like this are not valid Java!

Finally, sometimes there are things that need to be laid out in a way that looks like a program, but isn't a program. This appears in the same font as a program, but in *`italics`*, like this:

```
IF the value of "sold to" is blank THEN
  Set the value of "sold to" to the Reserve object
OTHERWISE
  Display an error message – the Duck is already sold
END IF
```

This is not valid Java.

There are many definitions in the book. Words that are defined in the Glossary at the end appear *like this* in the text.

A note on programming style

You will learn that programming style is an important element of writing a program. Style refers to the way in which a programmer lays out a program, what names are chosen for various things in the program, and what else might be added. Programming style is a very individual thing, and develops in any programmer over many years; it's very much like handwriting.

The programs and examples in this book were written by two people. We each have rather different programming styles, so we've negotiated and agreed to adopt a "house style". You cannot imagine the bloodshed. We think that the style we have finally agreed to adopt is a reasonable compromise and that it should be reasonably clear.

Develop your own style as you learn to program. Copy ours for the moment if you want to, but if you find things that you don't like don't follow them. Just be consistent.

A note on persons

You will discover soon that the chapters in this book make use of the first person, and the first person singular in particular. This might seem odd in a book that has been written by two people. The thing to remember is that each chapter was, in fact, written by just one person. You can amuse yourself, if you like, by trying to work out who wrote which one … There is the occasional clue.

To the Student

Hi.

Welcome to the book. We hope you like it.

If you're a student just starting out on your first programming course, this book is for you. This book contains what we think you'll need to know as you go through your course. We very much hope you'll enjoy reading it and come to enjoy programming.

Because we want to get something absolutely clear before we go any further; programming *is* enjoyable. It's a creative pastime, and has been called by some a craft. Writing a program is the process of creating something from nothing – the process of creating something that solves a real problem and hopefully makes the world a better place. A complete and correct program can be a source of great satisfaction to a programmer. Even the appearance of the lines of a program laid out on a sheet of paper can be a thing of beauty, almost like a poem.

But enough of these fine words. We would be lying to you if we didn't admit that many people do not especially enjoy learning to program. Many people do find it difficult, but just as many take to it quickly and easily; we suppose that you'll be finding out which one you are in the next few weeks. Whichever turns out to be you, just keep in mind that anyone can get there eventually; there's nothing special about people who can write computer programs.

This book is not like many of the other books on programming that you can see on shelves in the bookshops or libraries. For a start there's no chapter on the history of computers, gloriously illustrated with highly amusing photos and hairstyles from the 1950s. No. We'll be assuming that if you want to look at that sort of stuff you know where to find it, and you'll go and seek it out. No. This is a book about *programming*.

You notice that we say *programming* and not *Java*? That's important. What we are about here is learning to program, and the programming language that we have chosen to use is something called Java. The skills and techniques that you'll find in this book, and which you'll learn, can be applied to many, if not most, other programming languages. A mechanic does not learn to repair just one kind of car, and a chef does not learn to cook just one kind of pie. So a programmer does not learn to program in only one language.

Just reading this book won't turn you overnight into a programmer. Reading this book will help turn you into a programmer, but you're going to have to do other things. You're going to have to write your own programs. You're going to have to practise. There are plenty of our programs in this book, and the first stage for you is to look at these and understand them. Then you're going to have to take the leap of starting to write your own programs. This transition – from understanding a program that someone else has written, to writing your own programs – is difficult, and let no one tell you otherwise. When you've managed it you'll have achieved something that should make you very proud.

The chapters that you're going to read are quite short. This is quite deliberate. With any luck you'll be able to find time to read them between classes or on the bus home, or something. As you read, doodle on the book, or make notes. Read *actively* and think about what you're reading. Learning is about thinking and reflecting, not just about reading.

Right. Sermon over. There's one last practical thing. You'll find out soon that we're going to assume that you've been given a copy of something that we're going to call your *Local Guide*. This should explain how the Java system that you're going to use works, and should fill you in on any other little local details. If you've got that (and there's no need to look inside it just yet), all that remains to be said is …

… let's go and do some programming!

To the Teacher

Hello. Welcome to a book about learning to program.

Before we go any further, you need to be absolutely clear about what it is that you are holding in your hand. This is possibly a book with the name of a programming language in the title that is unlike any book about with the name of a programming language in the title that you have encountered before. And you have probably encountered many. Too many.

A big claim, that. But this is not a book about Java. This is not a book that seeks to explain all the minute details of the Java language.[1] This book contains no UML and none of whatever the flavour of the month is at the moment in systems development. This is not a book that an experienced programmer, working in industry, would use as a reference while working on some commercial project. There are lots of books like that, and lots of books written for experienced programmers, and this is not one of them.

This book came about like this. We'll let Tony explain.

I've been to a few conferences on teaching computing, and I've given a few presentations and so on describing some of my ideas on what's wrong and right with the way we teach programming. I think I've come to the conclusion that there's rather more that's wrong than right. A publisher's rep came up to me at one of these happy events and started to pester me to write my own Java book. I declined, since there were already far too many Java books about and I saw no need to add to this needlessly large pile of paper. More to the point, I didn't know Java, even if it was flavour of the month at the time.

The problem that then emerged was that this was a persistent publisher's rep. I kept finding that she kept popping up in my office. I will admit to having been bought a beer, but despite advice from other authors, I always seemed to miss the free lunch. Eventually, during ITiCSE 2001 at Canterbury, I cracked and agreed to write something. But only on my terms. I was not going to write another totally unnecessary book about Java. I was, in fact, going to write a book about programming and C++. I knew C++, you see. It was sort of last month's flavour that was still quite good.

The C++ book has been and gone. You might have seen it; it's the one with all the sheep. Now we come to the Java version. First, let's be very clear that this is not just the C++ book rewritten in Java. Some of the chapters are similar, yes. The style is not completely different, even if the sheep have mysteriously become ducks. But the whole basic approach and structure have been revisited. The main change is that Java demands a much earlier and deeper discussion of objects (which is present and correct), and that objects need to be used more and throughout. The rest of the material has all been revisited too, and changed where needed. Underneath, though, the approach is the same – the underlying belief is that students need to learn to program, they do not need to learn Java or C++ or some other language. And they need to understand that.

That is why this is a book about learning to program. Specifically this is a book that is intended to support a student following an introductory programming course in further or higher education. There is sufficient Java in this book to be included in such a course; there are also some pointers in the final chapter that would be of interest in the

1 You can probably tell that from the size!

more ambitious courses.[2] *My hope is that after reading this book, and after following your course, a student would be able to write some reasonably complex Java programs and make sensible use of one of the many other Java books that are available.*

Now let me explain why this book is like this. I have taught programming for many years in what is probably one of the most respected university computing departments in the UK. Every year I have some successes, and every year there are failures. I see students struggle with this topic; they are struggling with something that lies at the very heart of our discipline. I often see students suffer as they attempt to come to terms with programming; often I have seen them drop out of their degree simply to avoid more programming. I have certainly seen them carefully choosing course options in future years to avoid anything that resembles programming.[3] *Your students might be different, but somehow I doubt it (and if you think they are I respectfully recommend a second, closer, look). This sad state of affairs just cannot be right.*

One aspect of this problem (or at least one issue that contributes to the problem) is the nature of the programming textbooks available. These are often weighty tomes indeed, and many run to well over 1000 pages. You know the ones I mean. Most contain far more than could ever be learned effectively in a single course that is, after all, only one part of what a student is expected to study during the year. These books (there are a few honourable exceptions, of course) simply do not meet the needs of our students.

There was one last thing that I had to do before starting on the Java book. I had to learn some Java. Oddly enough, I did not seek out a Java programming course. I did not sit in a room with 200 or more other people trying to learn Java. No. I found someone who knew some Java (so a welcome to Graham!), I got him to tell me the basics, and then I had some fun writing some programs. Isn't it odd that we still expect students to learn to program from attending our lectures?

Now, this book works like this. It not our intention, or our place, to try and replace your lectures. The place of this book is to support your lectures by providing something that your students will actually read, hopefully before your lecture. Our job is to explain to them what's coming up, why it's important, and why it's useful. Each chapter should occupy about a week in a 20-week course; for a student this week should probably include a couple of lectures, some supervised practical time, and opportunities for plenty of practice. There are some exercises at the end of each chapter; please add in some of your own to fit your own local system or needs.

You might think that sometimes our explanations are a little simplistic. Sometimes we admit that we are, in Civil Servant terms, "economical with the truth". This approach is essential with a language as complex as the language that Java has now become. There are so many little details that can tend to get in the way of the real business of the day, which is to learn to program. Sometimes we've added a more complete explanation as a footnote; you might well want to go into more detail, particularly with your more experienced or advanced students. It's up to you.

As for the technical details, all the programs in this book have been written and tested using, at the earliest, version 1.4.0 of the JDK running on a Linux platform. We believe that all the programs work (except where stated otherwise),

---

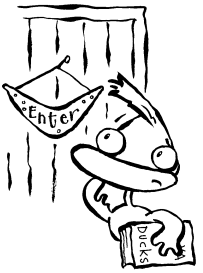
2 But those teaching more ambitious courses would do well to ponder whether it is better for a student to understand a little Java or to be totally baffled by a lot of Java.

3 At Leeds there is a second year course on *Linear Programming*. They avoid that too. Just in case.

and should work unchanged with other comparable Java systems. One issue might be that we've chosen to use the `ArrayList` structure, which only appeared in JDK version 1.4.

There's one last thing we need you to do before we start. We don't know what Java system you're planning to use. We know nothing about your editor-of-choice, and we don't even know what operating system you're using. To be honest, we don't much care. We've ignored all these issues, since we want this book to be useful to everyone. Obviously, though, there are some things your students need to know. As you read through the book you'll see that we've told them about something called the *Local Guide*.[4] We need you to put this together for us. Here you can describe how your system works, you can set down coding or layout standards if you want to, and you can pass on any other local wisdom. You could probably combine it with some existing set of notes. We hope that's not too much work for you. Thanks. The exercises at the end of Chapter 2 include most of the things that we need you to make sure that your students know.

Finally, we hope you agree with our reasons for writing this book. If you do, we're ready to go and teach some programming!



4 If you were to suspect that we've "borrowed" this idea from Leslie Lamport's LaTeX book, you'd be quite correct.

The web site of the book

This book is accompanied by a web site. The address is:

http://www.comp.leeds.ac.uk/tony/book/java/

The site is mirrored here:

http://www.palgrave.com/htpuj/

On the web site you can find:

- All the code that you'll need to complete the exercises.
- An interactive Java reference using the examples from the book.
- All the solutions to the exercises, with some extra details.
- Additional exercises for every chapter.
- All the programs from the book, so that you can download them, try them out, and adapt them.
- Links to free Java compilers, editors, and other development tools.

There are also forms to submit comments and useful links, and much more.

About the authors

Tony Jenkins is a Senior Teaching Fellow in the School of Computing at the University of Leeds. His hobby is teaching introductory programming. He is lucky that his hobby is also his job. He has given many presentations and written many papers about the ways in which programming is taught.

Tony gained his BSc from the University of Leeds in Data Processing (back when computers were *real* computers) in 1988. Five years spent writing programs in what many call the "real world" convinced him that fun was more important than money, and he returned to the University of Leeds in 1993. He has been teaching since then, and teaching introductory programming since 1995. In 2002 the University of Kent at Canterbury saw fit to award Tony an MSc for research into the experience and motivation of students learning to program.

When not at work, Tony can generally be found with Dave and Wallace in the Grove or the Eldon, probably after two hours on the terraces at Headingley, observing the antics of the Tykes or the Rhinos. He has been known to drink beer.

Tony approves of cats, but is allergic to them. He thinks that ducks are alright, but he has never been closely acquainted with one.

Anyone wanting to contact Tony about this book (or just to have a chat) can send email to `tony@tony-jenkins.co.uk`. There are rumours of some sort of web page at `http://www.comp.leeds.ac.uk/tony/`, but `http://www.leeds-camra.com/` is probably a better bet.

Graham Hardman works as a computer support officer in the School of Computing at the University of Leeds, and has done so since graduating from there with a BSc in Computer Science in 2001. During his time as an undergraduate, he acquired the *nom de plume* Mr Gumboot, for reasons lost in the mists of time, and probably best left there.

In the course of his job, Graham writes programs in many languages, including C, Java, Perl and Python. He can usually be found helping to maintain the large number of Linux workstations and servers in the School, but has been known to touch Windows machines on occasion.

Outside the hallowed corridors of Yorkshire academia, Graham maintains an avid interest in an organisation known as Everton, apparently a popular gentlemens' sporting establishment in his home town. He can sometimes be found in the Spellow House on Dane Street enjoying 5.68cl of draught Irish stout with various gentlefolk in royal blue attire. He also chases inflated spherical objects on artificial grass surfaces, is trying (and struggling) to teach himself Greek, and possesses a drumkit and several guitars.

Graham currently lives in Armley, Leeds with his wife Tanya, two guinea pigs named Arthur and Geraldine, a fish named Colin, and several dozen unnamed feral pigeons. He would rather like a dog, but feels that one would get decidedly bored in their 6 m^2 back yard.

Graham can be contacted at `gph@mrgumboot.co.uk`, whether to discuss Java, the 2008 European Capital of Culture, or the Modern Greek verb system.

Acknowledgements

There are, as always, many people that we need to thank for their help in preparing this book. This has indeed been something of an experience.

Thanks are due to all at Palgrave for their support and advice. Tracey Alcock started it going all those years ago (because she *really* wanted a Java book and not a C++ one), and Becky Mashayekh and Anna Faherty have kept us going more recently.

Christine Jopling once again drew the pictures. We only wish there was space for all the ones she drew before we changed the chapter titles. As always, Chris never seemed to mind if we needed just one more picture, or some really subtle and annoying change.

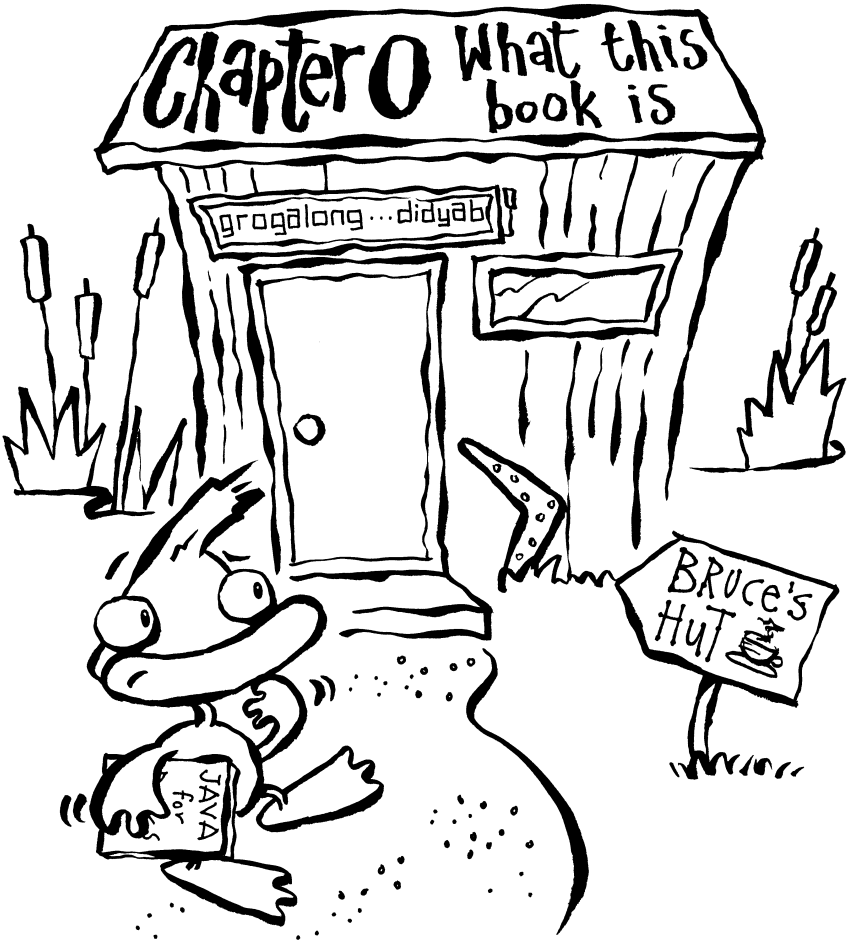
We are very grateful to the anonymous reviewers of the first version of this text. We know that one of you was David Barnes of the University of Kent because of some of the things you (quite correctly) said. The second reviewer remains anonymous, even if we do have our suspicions. And once again our colleague Nick Efford volunteered to go through a more complete version making many useful suggestions, most of which have hopefully found their way into the final version.

Respect and thanks to Mukesh and the team in Chennai for turning the Word files into a book.

Elvis the Duck is a bit of a mystery. Flossy the Sheep (as seen in the C++ book (and indeed as seen on Elvis's wall)) could be explained, but Elvis is a mystery. The best theory is that Elvis is Christine's idea, probably because she can draw ducks. Chris also produced Zoot the Coot. Tony would like to claim some of the credit for Don the Swan and Bruce the Goose.[1] Graham produced Mr Martinmere, because Graham is from that part of the world.

We are also grateful to Walrus Gumboot.

1 Yes. This is a very bad joke to do with Goslings.


Chapter 0 What this book is
grogalong···didyab
BRUCE'S HUT
JAVA for

Good Evening. Glad you could make it. Pull up a chair. Just let me put some more logs on the fire. Move the papers off the stool. Help yourself to a drink from the cabinet; there's some orange juice at the back, and maybe something stronger somewhere. I'm afraid I've run out of toast and pretzels. Would you like a chocolate digestive?

Ah, you've got a copy of the new book, "How to Program Using Java". A wise choice if I may say so. While you have your drink and digestive just let me explain what all this is about. There are some things that you're going to need to know before we start.

This is Chapter 0 and that is your first lesson. Most books have Chapter 1 as the first but this one has Chapter 0. I'm afraid that computers are like that. You're going to have to get out of the habit of starting to count from 1. Computers have a nasty habit of starting to count from 0 and if we're going to write programs to make a computer do useful things for us we're going to have to get into that habit too.

Before we go any further allow me to explain what this book *is* and more importantly what this book is *not*.

What this book is not

This is not a book about Java. There are plenty of books about Java available and it would have been very foolish of Graham and me to spend our precious leisure time trying to add to their number. This is not a Java reference book, although there is a handy Java reference in the back. It is not a book to read if you are already confident about programming and can program well in some other language. Oh no. This book is for people who can't program. It is for people who don't have much idea of what the whole thing is about. It is perhaps even a book for people who are starting a course in programming and are just a little bit worried about it.

What this book is

This is a book about learning to program. There are not many books about learning to program even if some books about programming languages claim to be about learning to program. These claims are normally wrong. This is a book about learning to program, and the language we will use to write programs as we learn is Java. This means of course that you will also learn some Java, but don't confuse the two things. If you can write programs in Java you can quickly learn to write programs in lots of other similar languages.

Computer programs are written in computer programming languages. Deep down most programming languages are basically the same. They use the same concepts and ideas and much the same constructions. Sometimes the way of achieving something is exactly the same in several languages. If you can learn to program in one language you can normally pick up another language without too much bother. I originally learned to program in a language called BASIC. Then I learned another called Pascal and then something else called C. Eventually I arrived at a development of C called C++, and finally I came to Java, the language in this book. It's learning the first language that's the difficult bit; it's much, much easier after that!

After they've managed the first language a lot of people find that they enjoy learning new ones. Graham and I have a colleague who claims to make a point

of learning a new language every year. There are many programming languages. Some have been designed for a particular purpose and are very good in that special area. Others, like Java, have been designed so that they can be used for almost anything. As you learn more and more programming, and more and more languages, you'll come to be able to pick the right language for the job. But you have to start somewhere, and you're starting with Java.

This is a book about learning to program. It is not a book about a particular programming language. This means that this book is rather unusual. There is not surprisingly a lot in this book about learning to program but not very much just about Java. When you've read this book and when you've done all the exercises you'll be able to get a different book about Java and learn about all the extra fiddly little details that we've missed off as and when you need them. Java certainly has a lot of these, and the last chapter will point you in the right direction. Our job in this book is to put you in a position where you can use one of the other books if you want to learn more Java or another language.

You are learning to program. You are not learning Java. Promise me that you'll remember that.

Who this book is for

This book is mainly for students following a first programming course in Java as part of a further or higher education course. This will probably be in the first year of the course. It is especially for those students who have never done any programming before. It doesn't matter what your main course is, whether it's computing or something totally different. If you've not done any programming before this is the book for you.

It really doesn't matter at all if you've done no programming before. In fact, about half of the students I meet every year before their programming course starts have never done any. If you've done a bit before you might want to let your attention wander in a few places. I'll trust you to spot when these are.

Of course other readers are welcome to join us. If you just want to know something about programming or object-oriented programming in particular you're in the right place. The more the merrier, that's what I say.

Why this book is like this

I've been teaching people to program for a very long time. Every year I teach many students and every year almost all of them succeed. When they find things difficult they often go away to read their book. Sadly their book has almost always been written for people who can already program or for people who find the whole thing very easy or by people who are very good at programming and don't understand what it's like to learn to program. Most of my students do not find the whole thing very easy, and I hope I understand why. This book is for them.

Often my students tell me that they don't understand something or that they're stuck with some programming topic. They tell me that they've been away and read their book but that it didn't seem to help. This doesn't surprise me because I've seen their books. This is the book I wish they'd had.

Without further ado let me explain what is in this book.

What is in this book

After this introduction every chapter[1] in this book has five sections. The sections work like this:

In brief This section introduces you briefly to the topic of the chapter. It tells you what the chapter is all about and tells you what you should understand after reading it. Think of it as an enticing appetiser. Our friends who have done some programming before might want to read this to help them to decide whether they need to read more.

The idea This is the main part of the chapter. It explains the new idea that is being introduced in some detail and illustrates it with sample programs. This is the section you should pay the most attention to. It is the hearty main course.

Examples Most chapters also have some examples. These use the new ideas from the chapter and apply them to a new problem. I'll show you the correct solutions and working programs and explain why they are correct and why they work. This is the dessert and is something to look forward to.

Exercises There are always exercises or tasks for you to try yourself. They are not optional! You should work through these on your own, and don't start looking at the answers until you have. You should linger a while over this. Think of it as the cheese board.

Summary Finally you'll find a brief summary of the chapter. This explains what you should have learned and what you should now be able to do or what you should understand. This is what you should be able to do before going on to the next chapter. The coffee and after-dinner mints.

You should try to read each chapter in one sitting; they're not very long. Hopefully you'll be able to read one on the bus on the way home or something. Then try the exercises later on. Don't be tempted to read the book while sitting at a computer. Read the chapter. Then think about it. And then go to a computer and try the exercises.

This book uses an approach to object-oriented programming called "objects first". We'll start by looking at what "objects" are and how we might recognise them in the real world. Then we'll move on to see how we can implement programs using objects in Java. Some Java books don't deal with objects first; many years of experience have convinced me (and others) that they're wrong.

The chapters

The chapters are short and so there are quite a few of them. Here's what's in them. Read this now and you'll see where we're going.

Chapter 0 What this book is You are here! I want you to understand why this book is like this. You need to know what's going to happen in the rest of the book. Most importantly you need to realise that learning to program and learning Java are not the same thing.

---

1 Alright, I admit it. I mean "almost every". Sometimes in this book I will stray slightly from the exact truth and tell you what you need to know at the moment rather than what's strictly true. I'll talk in footnotes when that happens.

Chapter 1
Programming

The book starts with a bit of background and a quick history lesson. Before you start to learn to program it's important that you understand what a computer program really is. You'll learn that programs are all around us in all sorts of surprising places. Programming has developed over many years. Over this time there have been many different approaches and many different languages. Java is the programming language you'll use. In order that you appreciate why it is as it is we'll also have a brief history of programming and of Java itself.

Chapter 2
The mechanics

You are going to learn to write programs. You are also going to have to learn how to make your computer execute your programs to make them actually do something. You are going to have to know how to find the results that your programs produce.

What precisely you have to do will depend a lot on the computer system that you're going to use; you might be using Microsoft Windows, Unix or even something else. I don't really mind.

This chapter gives you the general idea of how to create and execute a program and will tell you what you need to find out and where to find it. It explains what goes on behind the scenes when you prepare to run one of your programs.

Chapter 3
Before you start

There are some important things that you will need to get hold of before starting to learn to program. This chapter explains what these are and points you in the right direction to find them.

There is also a very quick introduction to a Java programming environment that many new programmers have found meets their needs – *BlueJ*.

As well as physical things (like a computer!) this chapter also goes through some of the problems that some people have when they learn to program and explains what you need to do to avoid them.

Chapter 4
Objects. The building block

Java is an example of something called an *object-oriented* (and not object-*orientated* – let's get that right from the start!) programming language. In fact everything in Java (even the program itself, in a way) is an object. This chapter explains what this means by explaining precisely what an object is and how you might identify one.

Objects have special properties called *attributes* and *methods*. This chapter also explains what these are and how you can choose the correct set for the types of object in your programs.

Chapter 5
A word on analysis and design

Before anyone can write a program someone must decide precisely what the program should do. This involves *analysis* of the problem that the program is going to address and *design* of the solution.

This is really only a book about programming but you will also need to be able to do some basic analysis and design. You will need, for example, to analyse a problem you have been given to solve and then design a solution. This chapter explains what you need to know to do just that.

Chapter 6 A first look

Without further ado, we arrive at some computer programs and some Java. This chapter gives you a first look at some programs in Java, and shows you how the example from the previous chapter might look in a program.

This points you in the right direction for the rest of the book.

Chapter 7 Programming (don't panic!)

Programming is a tricky business. The title of this chapter gives the best advice of all for new programmers – Don't Panic! Programming is a structured activity that requires and even demands a structured, methodical approach.

Many new programmers do indeed panic when faced with a new problem. They make mistakes that an experienced programmer would never make. They make mistakes and work themselves into a hopeless state from which they can never recover. This is why many books on programming don't really help people as they learn.

This chapter describes the process of writing a program and some of the common pitfalls and mistakes that new programmers make. Hopefully after reading it you won't make them! Or at least you'll realise when you do make them so you'll only make them once.

Chapter 8 The basics

This is over a third of the way through the book and this is the first chapter that includes a detailed explanation of any Java! Don't be tempted to skip straight to it though; the chapters before are there for a purpose and contain essential background that you will need to have read and understood.

This chapter introduces the ideas of a *variable*, the most basic component of a program, and of course of the program itself. It shows how variables are used to store and manipulate values held in a program and explains how to display the values held in the variables. After reading this chapter you should be able to write your first Java program.

Chapter 9 Input

To write properly useful Java programs you need to be able to get values from a user. You need a user to *input* these values, usually from the computer's keyboard.

There are a couple of ways to do this and this chapter explains the Java you need to do it. After reading it you'll be able to write some useful Java programs.

Chapter 10 A word on testing

For a program to be truly useful we need to have confidence in the results that it produces. We need to believe that they're accurate and correct. This means that we have to *test* all our programs very thoroughly. Testing is another structured process that requires a plan.

This chapter explains how to build up a series of *test cases* into a *test plan*. This plan will allow you to test your programs so that you can be confident that they work. If you are confident in the results produced by your program your users should be too.

After reading this chapter you should also understand why it is never possible to be completely sure that any program works in every possible case!

Chapter 11
A first class

Objects are implemented in Java as something called *classes*. An object in a Java program is in fact an instance of a Java class.

This chapter introduces a very simple class to show the basic ideas of how this works. After reading this chapter you should be able to write programs that make use of other simple classes, and you should have a basic idea of how to write such classes yourself.

Chapter 12
Classes and objects

This chapter looks in more detail at how classes and objects are defined in Java. There are quite a few fiddly little details that you need to understand!

At the end of this chapter you should be able to write simple Java classes and programs that make use of them. You should be able to identify the classes in a problem area and design and write a program using them to solve the problem.

Chapter 13
Get your hands off my data!

Java classes have a public face and a private face. The public part, called the *interface*, is available to all programs that use the class. The private part, on the other hand, is available only to the class itself. Classes are not unlike people.

This is a very powerful mechanism that allows for *data hiding*. It is one of the key ideas that make Java programs portable between different computer systems and one of the properties of Java that makes it one of today's most popular programming languages.

A well-written Java program written on one computer should be able to run unaltered on another computer system. It should be obvious why this is a good thing.

After reading this chapter you should understand the difference between the public and private parts of a Java class. You should be able to design classes with suitable public and private parts and you should understand why this means that Java programs can be portable and reusable.

Chapter 14
Making things happen. Sometimes

This chapter moves on to deal with the rest of the basics of Java. It describes the two *conditional control statements*. Sometimes there are parts of a program that we want to execute only if some condition is true; a word processor program should print a document only if the print button is pressed, for example.

This chapter explains what a *condition* is and how it can either be *true* or *false*. It explains how to combine single conditions into more complex expressions using Boolean logic. Finally it explains how you can use such expressions to control how your programs behave in certain situations.

After reading this chapter you should be able to write more complex Java programs that can carry out different tasks depending on the user's input and on other values. Your programs will be able to deal sensibly with unexpected input values and you will be able to implement simple menu-based systems.

Chapter 15 Making things happen. Again and again

Sometimes parts of a program must be executed many times. This might be for some number of times that is always known (*determinate*) or for an unknown number of times until some event happens (*indeterminate*).

This is achieved in Java with *program loops*. This chapter describes the different kinds of loops available in Java and explains how to use them.

At this point you should be able to write many useful Java programs. You will have learned most of the basic Java you will need and will hopefully have had a good amount of practice. You will be a Java programmer!

Chapter 16 More methods

The new Java in the previous two chapters will enable us to write some more complex methods in our classes.

This chapter provides some examples of doing that and also fills in some of the final details in how methods are written and used.

Chapter 17 Collections

Often programs must process collections of data. The final part of Java that you need is the ability to store such collections. You might want to write a program to store the details of all the books on Java in a library, for example. This program might need to search for a particular book or display a list of all the books.

This chapter explains two ways to do this in Java using *array lists* and *arrays*. These two ideas will allow you to design and write complex and useful Java programs. You will indeed be a Java programmer!

Chapter 18 A case study

As you get near to the end of the book you have learned many things. This chapter ties them all together with one final big example.

This chapter shows you a problem and explains how to analyse it, design a solution, and then how to write a Java program to solve the problem. This is exactly what Java programmers do and now you should be able to do it too.

Chapter 19 More on testing

With bigger programs you need bigger and better testing if you are to be sure that the programs work. This chapter explains how to test more complicated programs and how to test Java classes so that they can be used elsewhere in other programs and by other programmers.

Chapter 20 Onward!

And this is the end of the book! You won't have seen all the Java that there is (very few people have) but you'll have seen, used, and practised a fair bit. This chapter introduces you to some of the more important and useful aspects of Java that you have not seen and explains how to find out more about them.

Java reference and examples

As you write your programs you will often want quick access to a short summary and some simple examples of how the basic Java commands and ideas work. You will want to check the details of the syntax of some command or you will need to be reminded of some detail.

This is what you need!

Further reading

After reading this book you will probably want to move on to more programming. This section points the way to where you might start and gives you some references both on the World-Wide Web and in books to follow up if something has particularly interested you.

Some languages that you might want to move on to sooner rather than later are C++, Python, and C#. If this is what you want to do, you'll find some handy pointers at the end of this chapter.

The book is rounded off with a handy glossary and index.

The characters

All good books on programming have a cast of lovable characters; it has to be said that in some books they're the best bit. Not wanting this book to be an exception to this fine tradition[2] I too have assembled a suitable cast of suitably lovable characters to guide you on your way through the book.

Here they are.

Elvis

The star of the show. Elvis lives on a wild-fowl reserve with his friends near a nice big pond. He used to be just a typical duck whose day generally consisted of looking out for the odd person who throws bread in the pond and going "quack" a lot.

This was not enough for Elvis, and he failed to find fulfilment in this role. So he has decided to branch out. Computers seem to be quite big at the moment, so Elvis has decided to see what they can offer duck-kind.

Elvis is a duck that is going to learn Java.

Buddy

Buddy is Elvis's best friend. Buddy is always keen to try new things, so he has decided to join Elvis.

Cilla

Cilla makes up the happy band of duck programmers.

Cilla is an especially good friend of Buddy.

2 There are many fine traditions that should be observed by all books on programming. I hope that I haven't missed any; I'll be pointing them out as we go along.

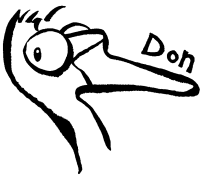
Bruce

Bruce is just a typical goose. He recently arrived at the reserve from Australia, a country where all the geese are skilled Java programmers.

Bruce has agreed to teach Elvis and his friends all he knows.

Zoot

Zoot is a coot. He is not entirely convinced that computer programming has much to offer coot-kind, but he has decided to join in. Just in case.



Don

Don is the only swan at the reserve. He finds that the size of his wings makes typing difficult and has little time for computers.

He sometimes gets in the way.

Mr Martinmere

Mr Martinmere runs the nature reserve where Elvis and the others live.

He knows little about Java, but is not afraid of using the skills of his birds to make the management of his reserve easier.

The end of the start

This is the end of the start of the book. Hopefully you now know what's coming up and why it's coming up in that way. It's time to read on.

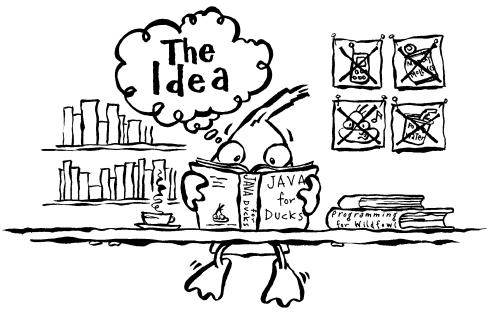
Right. Put the dog back on the chair and I'll put the fire out. Leave the empty mug on the table. Mind the crumbs.

Now, it's on with the show. Remember that we're in this together and that I'm right behind you!

Chapter 1
Programming

In the modern world computer programs are all around us. They are sometimes hidden away in the most unexpected and unlikely places. All of these computer programs have at some time been written by a human programmer using the processes and skills that you are about to learn. This programmer has gone through the stages that you are going to learn about in this book and has probably used a programming language not unlike Java. At some point in the past this programmer learned to write programs. Before you start learning to write programs it is essential that you know what a program is.

After reading this chapter you should understand what a computer program is, and you should understand what programming is all about. You should be able to identify the various programs that run on the computers you use and you should be able to make a reasonable guess at how they were written. You should understand something about Java and why it is a popular programming language. You should know about some of the people who have contributed to the development of programming and to other currently popular programming languages and to Java in particular.



Computer *programs* are everywhere. I am using a computer program to write these words. These words will be processed by many more computer programs before you read them; an electronic mail program will be used to send them to an editor, a backup and compression program will be used to keep a safe copy, and the typesetter will use yet more programs. It is entirely likely that a computer program was involved when you bought this book; there would have been one in the till when you paid, you may have used a computer-based catalogue to find the book you wanted, and you may have paid with a credit card that was authorised by a computer. All these things rely on computer programs, and all these programs have been developed by human computer *programmers*.

While I type these words, my computer is running many other programs. The *operating system* itself (Microsoft Windows as it happens) is a program. I am using yet another program to choose which track to play on a CD; this program is using another program buried deep inside my CD drive to translate the data stored on the CD into sound. The CD itself was probably recorded on a machine that used many computer programs. Many of the instruments played to make the recording also contained computer programs.

Computer programs can be found in devices ranging from mobile telephones through dishwashers to sports cars and fighter aircraft. All of these programs have one thing in common; a human programmer wrote them and went through much the same processes to write each one.

Computers are pretty useless things without programs. On its own a computer is little more than a collection of wires, plastic, and tiny bits of silicon. On its own a computer would make a handy doorstop, and little more. It is only when a program is added to this collection of components that a computer becomes a useful tool. The development of computer programs is a fundamental part of the development of computing.

A program

Here is a simple program. It is not the sort of program that can be used on a computer, but it *is* a program. It is, in fact, a recipe for chocolate cake.[1]

```
Ingredients
  4 oz soft margarine
  4 oz caster sugar
  2 size 3 eggs
  3 oz self-raising flour
  1 oz cocoa powder

Method
  Pre-heat the oven to gas mark 4, 325°F, 160°C.
  Beat the margarine and sugar together until the mixture is
    almost white.
  Beat the eggs one at a time into the mixture.
  Sift the flour together with the cocoa powder and carefully fold
    it into the mixture.
  Divide the mixture into two 6 inch round tins.
  Bake on middle shelf for 20 minutes.
  If desired, cover with melted chocolate.
```

(Source: Home Recipes with Be-Ro Self Raising Flour, *37th Edition, Rank Hovis MacDougall, 1982)*

Although this may appear to have little to do with computer programming, this recipe shares many features with computer programs:

- It follows a format dictated by convention – all recipes look like this, with the ingredients listed first and then the method described step-by-step.
- It is written in a specialised form of language.
- This language has a vocabulary that someone reading it must understand – beat, sift, and fold are all terms that the person making the cake would need to understand.

---

1 You can try it, if you like. It's really rather good.

- After the ingredients have been listed the recipe consists of a sequence of steps that the user must follow in the correct order to achieve the desired result – it would be foolish to put the cake tins in the oven before making the mixture.
- Some steps require the user to make comparisons and take decisions based on what they find – in this case the user must repeat the action "beat" until the mixture is white.
- Some steps require the user to make choices – here the cake is covered in chocolate only if desired.

It would also be possible (but admittedly unlikely) for a cook to follow all the steps of this recipe without any knowledge of what the final result was going to be. That is exactly how a computer interacts with a program; it follows the instructions that it is given without any knowledge or understanding of the result that the programmer is trying to achieve.

A computer program is simply a "recipe", called an *algorithm*, presented to a computer in order to allow it to carry out some task. The recipe is written by a programmer and is expressed in a programming language. The programming language has a vocabulary from which the programmer constructs the instructions that the computer can understand and execute. These instructions form a computer program.

Programming languages

Early "computers" were designed to fulfil a single purpose. The devices generally considered the earliest computers, the Difference Engine and Analytical Engine designed and partially built by Charles Babbage, were designed to carry out the calculation of mathematical tables. This extremely tedious and repetitive task was an obvious candidate for automation, especially as the many errors in the human-generated tables could prove very costly.

Babbage never completed his machines. He was misunderstood and ridiculed during his lifetime and died unknown. One of the few to recognise the potential and importance of his work was Lady Ada Lovelace, now generally recognised as the first computer programmer. Lady Lovelace developed a mathematical notation for representing different ways to "program" Babbage's Analytical Engine. Sadly, as the machine was never built, it was never programmed. Today there is a replica of Babbage's Difference Engine in the Science Museum in London; it is the first exhibit in the gallery tracing the history of computing. Any self-respecting programmer should take a moment when in London to examine the Difference Engine, especially now that admission is free.

The first electronic computers were programmed in what we would now consider very "low-level" ways. The programming involved changing physical components in the computers; levers were set, wires were moved, and switches were changed. This was a slow and error-prone job. This was also the time when the first computer program "bugs" were encountered; these were moths that crawled into the computers, promptly died, and stopped them working[2] properly.

---

2 The moth normally stopped working properly too. There is actually a school of thought that this story about the moths is an "Urban Legend". Why not spend a bit of time looking in to this on the web?

It became apparent that a different way of programming computers was needed. The existing way of working was too close to the machine's way of operating; the programmer was being forced to think at too low a level. This led to the development of the first programming languages expressed in a form that was more convenient to human programmers.

This was an improvement but it was not ideal. Every computer had its own language and programming any computer was still a complex and time-consuming task. A language was needed that would be easier to use and that could be used on any computer. A "high level" language was needed.

This idea had been seen before. At the end of the Second World War Konrad Zuse had developed a high level language that he called Plankalkul. This was never implemented, but it remains as the first high level programming language ever described.

The first language to be implemented and used was FORTRAN, first released in 1957. FORTRAN was designed mainly for mathematical applications and was highly *portable*; FORTRAN programs would run unaltered on many different types of computers. Languages based closely on the original version of FORTRAN are still popular for many mathematical applications today.

Many more languages followed. Some were designed to be truly general-purpose and others were designed for particular types of application. Notable languages include:

Lisp — Lisp was designed specifically for List Processing applications. It is still used today in some areas of artificial intelligence.

Algol — Algol offered programmers the ability to express algorithms in a neat and concise way. In many ways it is the earliest ancestor of Java.

COBOL — COBOL was designed for applications in business. It offered sophisticated features for processing large files of data to produce the sorts of reports that managers needed. Decisions made in the development of COBOL programs in the early years of computing led to most of the panic surrounding the "Y2K Crisis" at the start of this century.

BASIC — As programming languages developed there was a need to train new programmers. BASIC was a language designed for beginners; it included all the features of most other languages and was a fine learning tool. Most of the early home microcomputers in the 1980s provided a version of BASIC as their programming language.

Simula — Simula was a special-purpose language, designed to carry out simulations in applications such as queuing theory. In a way it is also an ancestor of Java as it introduced the concept of an object.

Java

We come now to Java. Java is the latest in a proud family of languages. Its origins can probably be traced back to a language called BCPL. BCPL was developed at Cambridge University in the late 1960s as an alternative to FORTRAN. BCPL was developed by Martin Richards while he was visiting the Massachusetts Institute of Technology (MIT); it became widely used by those working on the new Unix operating system there.

A more direct ancestor can be found in the B language also developed at MIT in the late 1960s and early 1970s. B was designed as a programming language for developing systems on Unix, which was yet another project being developed at MIT.[3] B was developed, extended, and refined and eventually evolved into a new language, imaginatively named C. C was first fully described in "The C Programming Language" by Brian Kernighan and Dennis Ritchie in 1978. C became one of the most popular programming languages, if not the most popular, in the 1980s and 1990s. It is still widely used today.

In the early 1980s an extension to C was proposed by Bjarne Stroustrup; this was C++ which is essentially a development of C with extensions to support something called *object-oriented programming*. Stroustrup was influenced by Simula and wanted to extend C to include object-oriented facilities. C++ offers significant advantages over C for many applications, although even Stroustrup himself has remarked that C is still to be preferred in some areas. The key is choice; C++ can be used as an object-oriented language or it can be used in a more traditional, C-like, way. This flexibility means that C++ became one of the most popular programming languages in a whole range of applications.

Java can in a way be seen as a further development of C++, although it was developed from a "clean slate". The history of C++ and its development means that it is effectively a hybrid language; in some ways it is an object-oriented language while in others it is more traditional. Java is a much more pure object-oriented language.

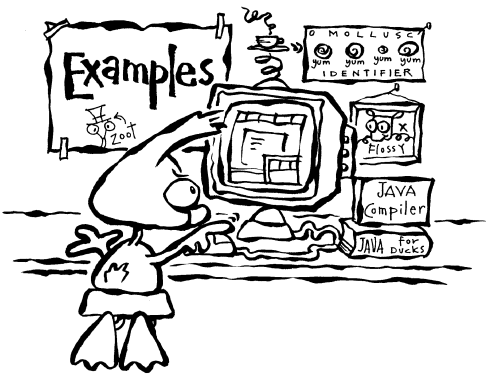
Java was developed by a group led by James Gosling at Sun Microsystems. It started out as part of something called the "Green Project", a research and development project started by Sun to try to look ahead to future trends in computing. The project developed various electronic gadgets aimed at the consumer market. These gadgets needed to be programmed and so a new language, originally called Oak, was developed. Oak is the language that eventually became Java; rumour has is it that the name was changed when the Green Project discovered that there was already a programming language called Oak. The name "Java" has a lot to do with coffee, a very popular drink among computer programmers; the Java logo of a steaming coffee cup can be see on Sun's Java web pages to this day (and versions of it are liberally spread around this book!).

Java is now one of the most popular programming languages around. Its heritage and design means that programs written in Java can run unaltered on all sorts of computers and devices (I have a mobile phone that can run small Java programs!). This design is based around something called the *Java Virtual Machine* (or JVM), which we will meet later on. The JVM is, incidentally, written in C.

The history of programming languages is quite fascinating in its own right. There are pointers to help you find out more in the Further Reading section at the back of the book. You might want to check out some more details of the history of Java, and perhaps even see what James Gosling himself has to say about it. If you understand how a language has developed you can often understand why it works the way it does.

---

3 You may well have met or heard of the most popular modern incarnation of Unix, the free operating system Linux.



There are no real examples in this chapter (I did say that some chapters wouldn't have any!), but here's an illustration of something that you should remember while you learn to program:

Many programming languages are basically the same.

This is simple to illustrate.[4] A first example is something called a *loop*, and in particular a "for" loop. Look at these for loops written in a few programming languages:

```
Pascal   for i := 1 to 10 do
BASIC    for i = 1 to 10
C        for (i = 0; i < 10; i ++)
C++      for (i = 0; i < 10; i ++)
Java     for (i = 0; i < 10; i ++)
```

Take it from me that each of these five has pretty much exactly the same effect in a program. They are examples of the same type of command in five languages designed for quite different purposes. Even if you don't yet understand what a `for` loop achieves, you should be able to see the similarities (yes, the C, C++, and the Java are all exactly the same).

Computer programming languages evolve. When someone designs a new language, they include all the basic features that are found in other languages, and it makes sense to use a way of expressing them that is similar to these other languages; for one thing it makes the new language easier to learn and therefore more likely to be adopted.

Here's another example. This is a conditional statement (a statement that lets a program make a choice), in this case testing whether some value (called *x*) is less than 10:

```
Pascal   if x < 10 then
Python   if x < 10:
C        if (x < 10)
C++      if (x < 10)
Java     if (x < 10)
```

4 It is also a fine example of some of the things that I'll tell you that are not *strictly* true. It is true that the vast majority of the languages that you are likely to meet for the time being are all the same. These are called *procedural* languages. You might in the future meet *functional* languages such as ML or Lisp which do work in a rather different way.

Once again the C, C++, and Java are the same, which is to be expected as C++ is a development of C (in fact all valid C programs are also valid C++ programs) and Java is based closely on C++. But the examples in the other two languages look very similar, as would examples in many more languages.

Remember this. You are not just learning Java. You are learning to program and you are learning principles that you will be able to apply in many other languages.

1.1 If you have access to the World-Wide Web look up some information about some of the main figures in the development of C, C++, Java, and object-oriented programming. Dennis Ritchie, Brian Kernighan, Bjarne Stroustrup, and James Gosling all have homepages that can be found quickly and easily. There's a list of web search engines at the back of the book.

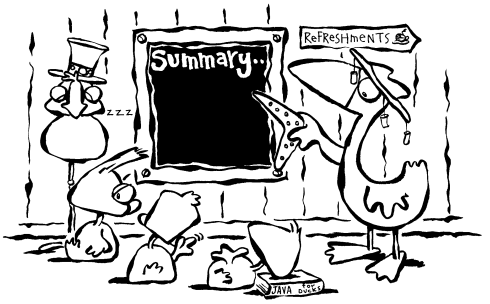
1.2 An important part of your development as a computer programmer is that you know something about other figures whose work has been fundamental to the development of programming. Use the web to find out more about Charles Babbage, Lady Ada Lovelace, Alan Turing, and Grace Hopper.

1.3 In this chapter a recipe for chocolate cake was shown as an example of an everyday thing that has some of the same characteristics as a computer program. List at least two other everyday things that have the same characteristics.

1.4 Look around you know. What things around you have computer programs inside them? Who do you imagine wrote them, and how?

1.5 If you have access to a computer running Microsoft Windows press `Control-Alt-Delete` and click on `Task Manager`. Under the "Applications" tab you'll find a list of the programs running on the computer. See if you can discover what some of them do.

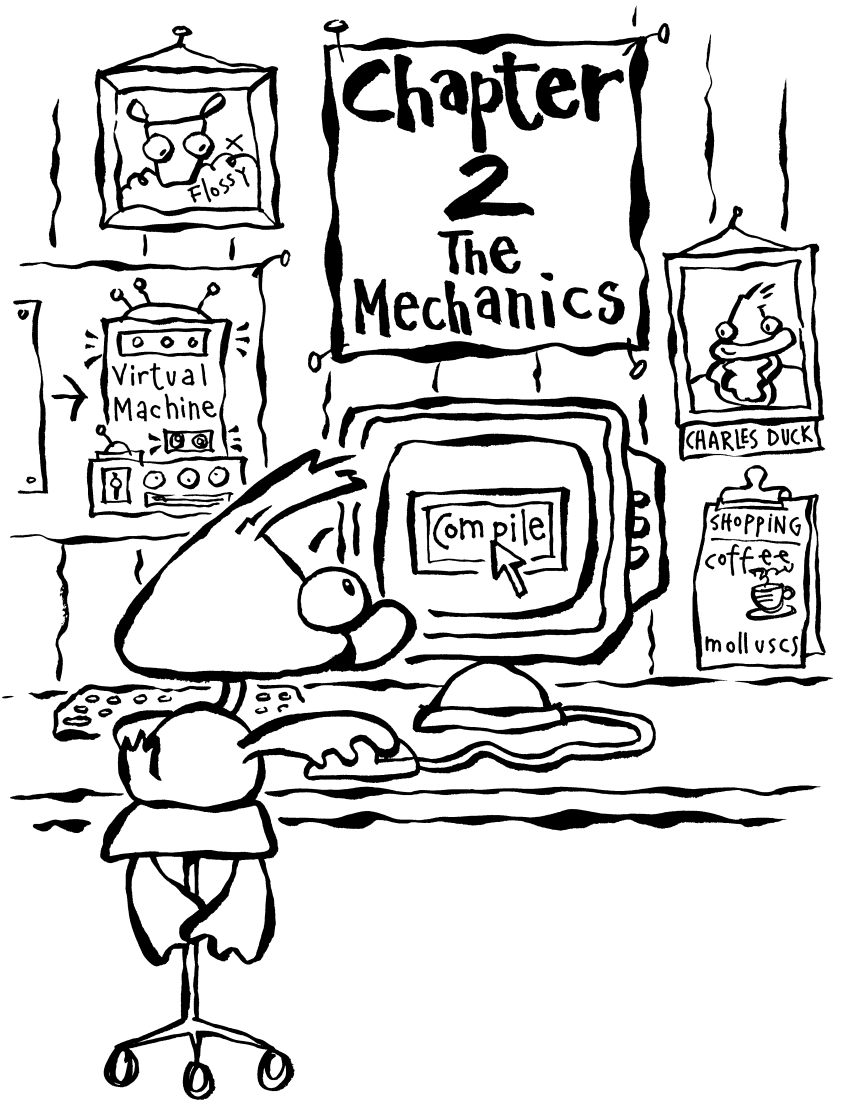
1.6 Alternatively, if you have a Unix system, type `top` at the shell prompt and look at the output. This shows you all the processes (programs) running on the computer. See if you can find out what some of them actually do.



Computers are all around us and so, therefore, are computer programs. Any computer that is carrying out any useful task has somewhere inside it a program. In fact it probably has many programs. All these programs were written by human programmers using a programming language.

As computers have become more sophisticated so have the languages that are used to program them. Early computers were programmed with switches and levers and then by low-level languages. This was so time-consuming and error-prone that higher-level languages were developed. Over the years some of these languages have died out while others have been refined and developed further.

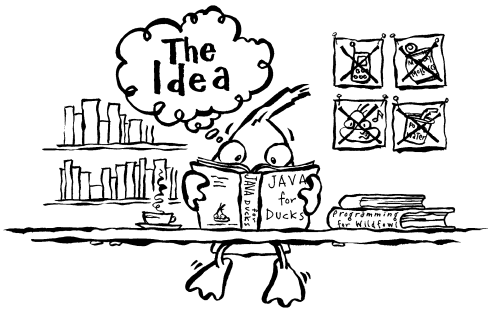
Java is one of the latest languages in this evolution.


Chapter 2
The Mechanics
Flossy
Virtual Machine
CHARLES DUCK
Compile
SHOPPING
coffee
molluscs

Humans and computers do not speak the same language. Humans communicate in spoken language, a very complex system of nouns, verbs, adjectives, and so on. Computers on the other hand are much simpler. Their "language" has only two symbols, 0 and 1. Obviously some translation must take place if humans and computers are going to be able to communicate effectively. Something will have to happen if a computer is going to understand what a programmer is asking it to do.

A computer program is a description of a way to solve a problem or carry out a task; we might call it a procedure for solving a problem. It is written in a form that humans can use and understand. This chapter introduces you to the processes that take place when a computer program expressed in this form is translated from this representation into a form that computers can use and understand. Both these forms are actually the same program in the same way that this book translated into Japanese would still be this book; all that has changed is the way it is represented.

After reading this chapter you should understand what a *compiler* is and what the process of *compilation* involves. You should understand how your programs will be linked with the Java *system libraries* to produce versions that can be executed by your computer. You should know what the *Java Virtual Machine* is and you should understand that role it plays in executing your programs. You should also understand how to compile and run your programs in whatever programming environment you are going to use.



The previous chapter explained that a computer programming language is essentially no more than a way for a human programmer to communicate with a computer. Humans find it very hard to express themselves using the 1 and 0

binary language of computers[1] and computers find it equally difficult to understand natural human language. A computer programming language is something of a halfway house between the two; it's something that both humans and computers can understand. In many ways the language is not quite exactly halfway; it's usually rather closer to the human's side than to the computer's. This means that some special software is needed to translate the program into a form that the computer can execute. This translation is a process called *compilation*. This is the process that is explained in this chapter.

But first a word about the computer environment that you're going to use.

The "Local Guide"

Computer systems are all different. This is a book about programming and Java and I don't want all the details of many different computer systems to get in the way. I certainly don't want this book to be useful only to people using a particular type of computer system.

If you read the introduction, you'll remember that I'm going to have to assume that you have a document that I'm going to call the *Local Guide*. This guide might be a manual that came with the system or it might be something specially written for you as part of a course. I'm going to assume that it will explain to you how to create and run programs on the computer system that you're using. I don't mind what that system is just as long as it's capable of running standard Java. What exactly it is doesn't matter at all in the rest of this book.

What I'm going to describe now is the process that happens whenever any program is created and run on any computer. This process is essentially the same on every platform.

Creating and naming a program

A program exists in a file on a computer system. Each file on the system has a unique name,[2] consisting of a memorable name for the file and sometimes an extension:

```
memorable.ext
```

The *file extension* normally indicates what type the file is. You may well be familiar with `.doc` files that contain Microsoft Word documents, for example, or `.jpg` files that contain images. The extension for a Java source file must be `.java` (hence these are often simply referred to as *.java files*), and using any other extension for a Java program is likely to produce a compiler error.

These are possible names for Java programs:

```
Program.java
Ducks.java
Elvis.java
```

The extension should never be left off. Doing so will probably cause errors. Ideally the name of the file should give some indication of the purpose of the

1 Although this is precisely what the users of early computers had to do. A switch that was on was a 1, and one that was off was a 0, for example.

2 Well, unique in its own directory or folder.