1 Introduction

Objectives

This chapter sets the scene by introducing the background concepts needed to begin programming using Java.

Keywords

abstraction *humour* *jargon*
history

1.1 The Start

This book, as may be inferred from the title, is about programming with Java. We assume that you, the reader, are basically computer literate, being familiar with using a computer for tasks such as word processing, World Wide Web browsing, etc. but have little or no programming experience, and, in particular, little or no *object-oriented programming* experience. Object-oriented? It's the technical term for the kind of programming method that Java embodies and any competent programmer needs to know about. This book is all about object-oriented programming using Java.

In this chapter, we are going to set out the background information needed to give context. In the rest of the book, we will proceed through not just a description of what Java is but, rather more importantly, how to use it effectively and efficiently. By the end of the book, you should be ready to write some serious and useful programs. That is one of the key reasons we wrote this book: we want to leave readers ready to tackle real problems and get professional results. This book is not a Java language primer but is a description of the object-oriented approach to programming using Java.

To set the scene for those totally new to programming, we need to define a few basic terms. A *program* is a (usually long and complex) series of instructions that are carried out (or *executed*, or *run*) by a computer. The instructions define what the program does, whether it is a word processor, spreadsheet, game, or any other kind of program. The role of the *programmer* is to create, or write, programs and this is usually done using a *programming language*, one of which is called Java. A*pplication programming* (sometimes termed *application development*) is the process of programming in order to create a program (the *application*) to be used by people (*end users*) in order to get something useful done. S*ystems programming*, on the other hand, is the activity of writing, amending or extending the *operating system*, which is the program that manages a computer's resources in order to allow applications to be run.

As can be seen from the last paragraph, talking about programming and Java involves becoming familiar with quite a lot of technical terms and jargon. We explain the jargon as we begin to use it but if you find that you don't understand a particular term try looking in the Glossary (Appendix A, page 821).

1.2 A (Very!) Short History of Java

When Java first 'hit the streets' in late 1995–early 1996, a lot of people started to use it as a way of creating dynamic Web content, *applets*. Web pages were, prior to this time, constructed using only HTML, which did not then allow sophisticated pages to be built. In the last few years, the whole Web content creation scene has changed dramatically with the arrival of XML, XHTML, XSL, CSS, Flash, audio and video streaming, etc. Although in its early days Java was often described as the World Wide Web programming language, its use changed quickly and its main use is now as the language of first choice within server-side Web applications (*servlets*)—rather than browser-side applications, applets. Also, Java, in various forms, is becoming increasingly used in embedded systems. Traditionally, embedded systems have been written using assembly languages or C, but the properties of Java give it a place alongside these languages (not

replacing them). This is an interesting phenomenon since when Java was first being developed it was intended for embedded systems. Java's usage is coming full circle.

Given the statement that Java's usage is coming full circle, the actual origins of Java are interesting. The development of what we now call Java started out as work on a programming language called Oak. Oak was designed during the early 1990s by a team of developers at Sun Microsystems building on the object-oriented systems revolution then under way. (The name Oak was inspired, reputedly, by a large oak tree visible from an office window.) Oak was intended to be used for programming the embedded systems in consumer electronics devices, with a particular emphasis on areas such as interactive cable TV control devices and electronic home management systems. Among other features, these applications needed new kinds of user interface, easy networking and a programming system to support system development.

note

For a detailed history of the development of Java, visit Sun's Java website at http://java.sun.com/.

As it turned out, the market for programmable control devices didn't develop at anything like the scale or rate originally anticipated. However, as work on Oak continued, the Internet and, in particular, the Web exploded into widespread use. In early 1994, a decision was made to adapt the Oak language to the needs of the Internet and provide a programming language for Web-based applications. As Oak included many features that were relevant for the Internet environment, including the idea of being *architecturally neutral* (meaning that the same program can run on a wide variety of different machines without having to be rewritten), it was easy to make this decision.

By January 1995, Oak had been evolved into a robust programming language suitable for building Web-based applications and was renamed Java (mainly because the name Oak was already in commercial use elsewhere). In May 1995 Sun released the initial Java Development Kit (usually referred to as the JDK) via the Internet, which allowed developers all over the world to download it and start using Java. The JDK was supported by the release of a Web browser called HotJava, which was capable of running Java programs in the form of applets embedded in Web pages, a feature then taken up by browsers from Microsoft and Netscape. From these beginnings, Java became increasingly popular and is now widely supported by commercial developers and software companies, who are busy providing development tools and many other facilities.

Late in 1998, the Java 2 Platform was released along with Sun's implementation of it, Java 2 Platform, Standard Edition Software Development Kit (J2SE SDK). Over the subsequent years Sun developed the Java 2 Platform by enhancing the basic J2SE SDK and adding many additional features to extend the environment. Some of these features were concerned with new ways of constructing programs while others come in the form of *class libraries*, or as they are often referred to, APIs (Application Programmer Interfaces). APIs provide the programmer with a wide range of additional facilities in a ready-to-use form, saving much time and effort.

In the second half of 2004, Sun released a critical and significant evolution of the Java 2 Platform in the form of J2SE v5.0, which included generics, annotations, and various other highly desirable features. With this release, Sun reverted to using the term JDK—J2SE Development Kit—to label their realization of the Java 2 Platform. This evolution of Java required a significant change in the way that people worked with Java, especially when using and writing class libraries—which is all the time when using Java! This means that books and programs that do not use J2SE v5.0 are 'out of date' and could lead people into using Java in the wrong way. In this book we believe we present no out-of-date idioms of use of Java.

An important consequence of the evolution of Java is that it has grown into a full-scale, general development system, quite capable of being used for developing large applications that exist outside of the browser-based Web environment. In fact, it can be used to create all kinds of programs, with a particular emphasis on those that make extensive use of networking and communications. As mentioned earlier, Java is now having its greatest success as a language for developing what are known as *middleware* applications, which form the core component of distributed applications running on networks. Java is also being widely adopted in universities as a language for teaching programming. Moreover, it is now seeing use in embedded systems: many systems, for example mobile phones, set-top boxes and other consumer products, are now Java-enabled using a variation of the Java 2 Platform called Java 2 Platform, Micro Edition (J2ME). Java really is now becoming the embedded systems programming language that Oak started out to be.

1.3 Being at the Right Place at the Right Time

One of the most notable features of Java is that it became very popular in a short space of time. The growth of interest was spectacular and took many people by surprise. It is worth asking why.

Java is not a perfect language but it does incorporate almost all of the important, well-established ideas about programming languages and programming methods. This was and continues to be important, but there were a number of other factors which enabled Java to achieve mega-stardom:

- It looks familiar to users of other popular languages, notably C, C++ and Smalltalk. Familiarity avoids potential users seeing a new language as 'too different' and, therefore, not worth the time to learn.
- It was closely connected to the Web, which was becoming very popular. This association gave it a huge kick start.
- Programming 'gurus' liked it. Gurus are early adopters of new programming languages, even if they (the languages, not the gurus) are still immature, and can have a big impact on how quickly information about the language is disseminated.
- You can get it for free! Very important this, as it allows potential users to try out a new language with very little start-up cost (at least, in direct financial terms).

Interestingly, this list shows that the most important thing about a new language is the social aspect—people must want to use it because it does something they want to do *now*. Technical considerations often come a distant second!

Overall, Java was launched in the right place (on the Web), at the right time (just as the Web was getting really popular), with the right features (it is dynamic and object-oriented) and at the right cost (free).

1.4 What is Java?

The name Java is typically used as a generic label encompassing the Java programming language, the tools and environment of the development kit and all the class libraries and APIs that are delivered as standard. To a large extent all these elements are inseparable, so learning

about Java is learning about them all. Sometimes you hear references to the 'Java Programming System' or more usually (and properly) the 'Java 2 Platform', emphasizing that Java is not just a programming language on its own.

Java provides an *object-oriented programming language*. The chapters in Part 1 of this book will describe what this means. For now we need only know that a programming language allows you to express programs in a textual form, and that object-oriented development is *the* modern approach to designing and building systems, which is best supported by programming languages that embody directly the key object-oriented concepts.

Object-oriented development is now widely used as it addresses a large number of the problems that arise during software development, providing viable and practical solutions. This is of enormous significance as the track record for successful software development is perceived to be very poor. This is actually something of a misconception. It is true that: there is a lot of software that is written but never used; there have been some very high-profile failures of software development projects; and there is a lot of software that is delivered very late and of very poor quality. Despite this there is a large amount of very good and very reliable software in the world. Don't assume though that 'being object-oriented' is like waving a magic wand that solves all your software development problems. What it does give you is a conceptual framework and an approach to software development that is based on a great deal of experience of what works best. Java embodies much of this experience and this book aims to put that across while you learn how to use Java. In the end, though, you have to do the hard work of learning to use Java efficiently and effectively.

Programming is a learnt skill. Practice is essential.

So, Java is a programming language based on a particular way of doing software development. As must be the case with any programming language, Java is defined so that programmers can write programs and understand what they mean. The written form of a programming language is specified by its *syntax*, whereas the meaning of what has been written (i.e. what the program does) is specified by its *semantics*. Parts 1–4 of this book will introduce much of the syntax and semantics of the Java programming language in the context of actually developing software, while Part 4 presents detailed information in a more reference-like style.

1.5 Abstraction: The Critical Core of Programming

Programming, and especially object-oriented programming, centres on identifying and working with *abstractions*. An abstraction allows a concept or idea to be expressed and then repeatedly used without all the detail getting in the way. As an analogy consider a chair. There are all sorts of different kinds of chair but people learn to recognize the range of properties of what makes a chair—'chairness' if you like—allowing them to deal with particular instances of chairs without having to continually describe all the detail. Java supports the use of abstraction via the mechanism of *class* and *object*, where a class allows the properties of an abstraction to be expressed and objects are instances of classes allowing the abstraction to be used within a program: objects are instances of the abstraction described by a class that can be used without getting side-tracked by all the detail.

***Abstraction** is **THE** key concept in computing.*

Many basic abstractions are widely and repeatedly used, so there is no point in having to continually recreate them. As noted earlier, the Java system comes with an integral library of basic abstractions in the form of a large class library. This library is intimately interconnected with the programming language to the extent that neither can be used without the other.

Hence, the Java programmer must not only learn a programming language but must also become familiar with a large collection of classes. Whilst this learning task may seem very daunting initially, the advantage of making this commitment comes from having a large number of ready-made building blocks available when writing programs. Instead of having to describe everything in full detail, abstractions can be used, saving much time and effort.

There is an infinite amount of knowledge. However much is known, there is always more to learn.

The use of abstractions does not stop with those that come from the standard Java library, the programmer needs to find, learn and use other appropriate class libraries that can be added to the development environment or, indeed, build new ones themselves. Finding and building abstractions is the key to successful systems development, particularly object-oriented development. Failure to use and build abstractions really misses the point and is going to lead to poorly designed, unstable programs.

Abstractions are built up in layers. At the bottom is the programming language, next the abstractions from the standard library, then one or more layers of abstractions provided by the programmer. Each layer of abstraction provides a higher-level perspective on the system being implemented. For any non-trivial program, matching the level of abstraction with the way people think and reason about systems is critical. If the programmer is forced to deal with too much detail at any one time, mistakes will be made and progress will grind to a halt. Using the chair analogy again, say we are trying to describe a chair and its location in a room. Using abstraction, we describe the chair and then, treating the chair as a whole, just name a spot in the room where it is located. Without abstraction we have to describe where each of the parts (legs, seat, back, etc.) is exactly in the room, hoping that we have placed the parts in such a way that they make a complete chair. Clearly not a sensible thing to do.

Unsurprisingly then, most of this book is about abstraction. Part 1 introduces the Java technology of class-based abstraction. Part introduces the process of creating programs, which essentially means using and creating abstractions whilst designing the overall structure of the program, building and putting together the pieces, and testing the result to see that it actually works as required. Part 3 of the book presents some case studies of the design and implementation of programs, showing the process that was followed to create them and demonstrating how to go about using pre-existing abstractions and creating new abstractions in the context of building a program.

1.6 The Java 2 Platform

To use Java and get some programming done, development tools are needed. Java is distributed freely by Sun Microsystems as the Java 2 Platform, Standard Edition Development Kit (JDK) which is obtained from Sun's Java website (http://java.sun.com).

note

The Eclipse consortium have released a very good IDE *which can be used for Java development. Look at http://www.eclipse.org/.*

The JDK provides a set of command line tools, meaning that the tools are invoked by typing in commands to a command interpreter (in an xterm, gnome-terminal or equivalent if using Linux, Solaris or some other flavour of UNIX or a terminal window if using Microsoft Windows). Many vendors supply Java integrated development environments (IDEs), with all the tools integrated into sophisticated graphical programming environments. Since each IDE is different and this is a book about programming rather than IDE usage, we will simply assume the use of the standard JDK but it should be easy to use any JDK-compatible programming environment with this book.

At the time of writing this book, the current version of Sun's Java system is J2SE v5.0. The original version of Sun's Java system was called the Java Development Kit (JDK) 1.0. It went through a number of revisions and stabilized with version 1.0.2. The next major version of the JDK was 1.1. This also went through a number of revisions, ending up at 1.1.8. With the release of the Java 2 Platform and version 1.2, Sun changed the labelling so we had J2SE System Development Kit v1.2 (J2SDK). There were two major revisions of the system (J2SE v1.3 and J2SE v1.4), which continued the positive evolution of Java. Just as the next major revision was ready to come out, Sun made the decision to change the numbering and the labelling so instead of having J2SE SDK v1.5 we have J2SE v5.0 and a reversion to the label JDK to describe the Java 2 Platform, Standard Edition Development Kit.

Although every attempt is made for Java to be backwardly compatible so that existing programs can still be used, there is a policy of allowing deprecation (i.e. removal) of outdated features. Thus, programmers will occasionally encounter warnings about 'deprecated' features, most of which are replaced with new features. Since a deprecated feature can be removed in the next release of the JDK, it is wise to change code to remove the deprecated features immediately: the good programmer never leaves deprecated features in their code.

To use the JDK, it must first be installed on your computer—following the instructions provided. The JDK comes with an extensive set of tools and features. The core elements are:

- The Java development tools.
- The Java class libraries, organized into a collection of *packages*.
- A number of demonstration programs.
- Various supporting tools and components, including the *source code* of most of the classes in the libraries.

A full set of documentation is available separately from the main JDK and should be installed as a matter of priority when the JDK is installed. In particular, the documentation describes all the classes in the class libraries, providing essential information for the programmer. The documentation is supplied in the form of Web pages so it is easy to browse when using a computer.

A first priority of the Java programmer is to become familiar with the main development tools. Programs, written by the programmer in textual form, need to be transformed into a format that can be executed by a computer. This is done by a tool called a *compiler*. The Java compiler supplied in the JDK is named **javac**. To execute or run a Java program, a *run-time environment* containing a Java *interpreter* is needed and this is provided by a tool called **java.** Another tool called a *debugger* is provided for *debugging* programs. Debugging is the process of locating and fixing errors in programs, something programmers tend to spend a lot of time doing, even the good ones!

1.7 Java is Architecture Neutral

Java is a *compiled language* but instead of producing instructions for a specific hardware processor, a Java compiler generates Java *bytecodes*. When a Java program is executed the bytecodes are interpreted by an implementation of the *Java Virtual Machine* (JVM), which is an idealized computer optimized to run Java programs. Providing a computer or workstation has an implementation of the JVM, any Java program can be run on that machine. A Java program is a

program that is run by another program, the implementation of the JVM, running on a real processor.

A consequence of this is that a Java program can be compiled once, stored on a server, transferred over a network to a client machine and then executed on that machine without worrying about what kind of machine it is. A Java program is said to be *architecture neutral* as its executable representation is independent of both the physical architecture of the machine and the operating system it is running on.

These portability issues have important commercial considerations. Since a Java program can be executed on any machine with any operating system (such as Linux, Solaris, Microsoft Windows, etc.), as long as it has an implementation of the JVM, there is no longer any need to be concerned about which operating system a machine is running: operating systems become less important. Java developers do not have to worry about targeting a particular operating system since the JVM is the platform written for.

Figure 1.1 presents an overview of the Java execution environment. At the bottom we have the host computers that support the execution of an implementation of the JVM. At the top we have the applications that are the programs that need to be executed using the JVM. In between we have the Java execution system which comprises the Java Runtime Environment (JRE) and below that the JVM itself. The Just in Time compiler (JIT) is an optional part of the JVM that can considerably speed up execution of Java code. The JRE comprises various internal tools for loading and verifying applications as well as the class library. The JVM not only has the bytecode interpreter, it also has the peer interface which is used to connect Java graphics to the graphics system of the operating system of the host computer.

The Java execution environment provides not just a Java execution engine, it provides a safe execution environment. Every time Java code is loaded it is verified to check that it has not been tampered with, that the compiled code is internally self-consistent and consistent with the Java standard. Also each Java program is associated with a security manager which verifies that,

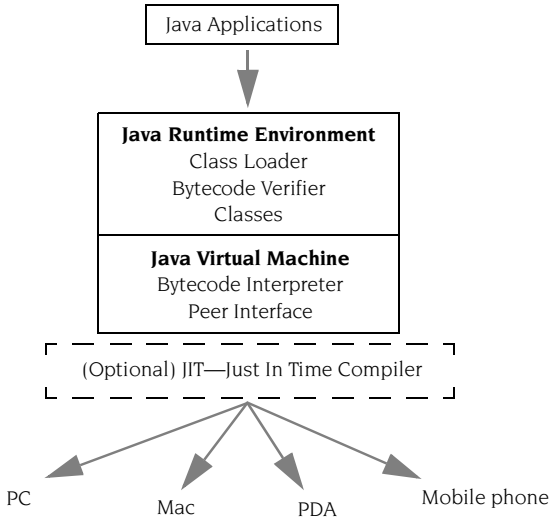

Figure 1.1 *Overview of the Java environment.*

when executing, the program has permission to access the resources such as files and network connections that it attempts to use. This ability to control resources available to executing programs is crucially important as it allows untrusted programs to be downloaded and executed with confidence that they cannot do any damage to the computer on which they are executing.

1.8 Java and its Jokes

It is reputed that the name Java is an acronym for Just Another Virtual Architecture.

Java certainly is the name of an island in the Indian Ocean that is (at the time of writing anyway) part of Indonesia. Java, the island not the acronym, is famous for many reasons. To its east is Bali and to its west Sumatra and between Java and Sumatra in the Sunda Strait (i.e. to the west of Java not 'East of Java' as in the 'geographically challenged' film title) is an island (Rakata) with a very famous small volcano (Krakatau)—well, it is small now compared to what it used to be before the 1883 explosion that gave us the song Blue Moon (allegedly) and the previous larger (!) explosion of 416 (chronicled in 'Pustaka Raja') which turned Kapi the volcanic island into Krakatau the volcano on Rakata the island with a few other surrounding islands as left-overs.

Of course, the name Java really became famous in 1890 when some bones of a very (very) long dead person (about 800,000 years to be more or less exact) were discovered, giving us Java Man (now believed to be *homo erectus* not ancient *homo sapiens*).

Having said all this, most people have heard of Java mainly because it grows a shrub (coffee) which can be used to create a drink (coffee) that contains a substance (caffeine), copious quantities of which are supposedly required for programmers to be able to function. In fact, in our experience, programmers require tea much more than coffee but this may be a very British thing. Either that or programmers require tannin as well as caffeine to function.

In any event, so legendary are the stories of programmers' use and abuse of coffee, that the arrival of Java as a programming language enabled vendors, and authors, to construct huge humorous edifices and, indeed, some actual jokes. The originators of Java, or someone associated with their publishing activity, created the coffee cup logo that appears on all of Sun's Java products, amongst other icons now an integral part of the culture of Java (the programming system anyway).

In the early days, many authors made use of the coffee theme, extending it to include related food imagery. In fact, so extreme did some of the coffee-related humour become, from vendors in particular, that many authors went for a backlash and completely eschewed any reference to coffee. Indeed, going so far as to invent many anti-coffee allegories.

All of this is good natured and, sometimes, very humorous. However, amidst our appreciation of the humour, we must not forget that Java, the programming language, is used for real, serious, large-scale application development. Do not let any of the humour of presentation detract from the very serious purpose of Java.

1.9 Summary

In this chapter we have presented Java as a modern programming system which is very much the 'right thing at the right time'. We believe that Java is also very suitable for teaching and for learning to work with object-oriented development.

This chapter has really just set the scene by providing some appropriate background. We are now ready to start exploring what programming is all about and how it can be done with Java.