

Java for Students

Visit the *Java for Students*, sixth edition Companion Website at www.pearsoned.co.uk/bell to find valuable student learning material including:

- How to download Java 6.0
- Programs from the book
- An extra chapter on Java network programming

The Pearson logo consists of the word "PEARSON" in a white, sans-serif, uppercase font, centered within a dark rectangular background. A thin, white, curved line is positioned directly beneath the text, arching slightly upwards at both ends.

We work with leading authors to develop the strongest educational materials in computing, bringing cutting-edge thinking and best learning practice to a global market.

Under a range of well-known imprints, including Prentice Hall, we craft high quality print and electronic publications which help readers to understand and apply their content, whether studying or at work.

To find out more about the complete range of our publishing, please visit us on the World Wide Web at:
www.pearsoned.co.uk

Java

for Students

DOUGLAS BELL
MIKE PARR

Sixth edition



Harlow, England • London • New York • Boston • San Francisco • Toronto • Sydney • Dubai • Singapore • Hong Kong
Tokyo • Seoul • Taipei • New Delhi • Cape Town • São Paulo • Mexico City • Madrid • Amsterdam • Munich • Paris • Milan

Pearson Education Limited

Edinburgh Gate
Harlow
Essex CM20 2JE
England

and Associated Companies throughout the world

Visit us on the World Wide Web at:

www.pearsoned.co.uk

Sixth edition published 2010

© Prentice Hall Europe 1998

© Pearson Education Limited 2001, 2010

The rights of Douglas Bell and Mike Parr to be identified as authors of this work have been asserted by them in accordance with the Copyright, Designs and Patents Act 1988.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without either the prior written permission of the publisher or a licence permitting restricted copying in the United Kingdom issued by the Copyright Licensing Agency Ltd, Saffron House, 6–10 Kirby Street, London EC1N 8TS.

All trademarks used herein are the property of their respective owners. The use of any trademark in this text does not vest in the author or publisher any trademark ownership rights in such trademarks, nor does the use of such trademarks imply any affiliation with or endorsement of this book by such owners.

ISBN: 978-0-273-73122-1

British Library Cataloguing-in-Publication Data

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

Library of Congress Cataloging-in-Publication Data

Bell, Doug, 1944–

Java for students / Douglas Bell, Mike Parr. – 6th ed.

p. cm.

Includes index.

ISBN 978-0-273-73122-1 (pbk.)

1. Java (Computer program language) I. Parr, Mike, 1949– II. Title.

QA76.73.J38B45 2010

005.13'3–dc22

2009051149

10 9 8 7 6 5 4 3 2 1

14 13 12 11 10

Typeset in 9.75/12pt Galliard by 35

Printed in Great Britain by Henry Ling Ltd., at the Dorset Press, Dorchester, Dorset

The publisher's policy is to use paper manufactured from sustainable forests.

Contents

Detailed contents	vii
Introduction	xix
Guided tour	xxiv
1 The background to Java	1
2 First programs	8
3 Using graphics methods	22
4 Variables and calculations	35
5 Methods and parameters	60
6 Using objects	88
7 Selection	115
8 Repetition	152
9 Writing classes	171
10 Inheritance	194
11 Calculations	210
12 Array lists	228
13 Arrays	242
14 Arrays – two dimensional	265
15 String manipulation	278
16 Exceptions	301
17 Files and console applications	318
18 Object-oriented design	348
19 Program style	369

20	Testing	383
21	Debugging	397
22	Threads	406
23	Interfaces	416
24	Programming in the large – packages	426
25	Polymorphism	432
26	Java in context	441
Appendices		454
Index		522

Detailed contents

Introduction	xix
Guided tour	xxiv
1 The background to Java	1
The history of Java	1
The main features of Java	2
What is a program?	3
Programming principles	5
Programming pitfalls	5
Summary	6
Exercises	6
Answers to self-test questions	7
2 First programs	8
Introduction	8
Integrated development environments	9
Files and folders	9
Creating a Java program	10
The libraries	13
Demystifying the program	14
Objects, methods: an introduction	15
Classes: an analogy	16
Using a text field	17
Programming principles	19
Programming pitfalls	19
Grammar spot	20

New language elements	20
Summary	20
Exercises	21
Answers to self-test questions	21
3 Using graphics methods	22
Introduction	22
Events	22
The button-click event	24
The graphics coordinate system	25
Explanation of the program	25
Methods for drawing	27
Drawing with colours	28
Creating a new program	28
The sequence concept	29
Adding meaning with comments	31
Programming principles	31
Programming pitfalls	32
Grammar spot	32
New language elements	32
Summary	32
Exercises	32
Answers to self-test questions	33
4 Variables and calculations	35
Introduction	35
The nature of <code>int</code>	36
The nature of <code>double</code>	36
Declaring variables	37
The assignment statement	41
Calculations and operators	41
The arithmetic operators	42
The <code>%</code> operator	45
Joining strings with the <code>+</code> operator	46
Converting between strings and numbers	47
Message dialogs and input dialogs	49
Formatting text in dialogs with <code>\n</code>	51
Converting between numbers	52
Constants: using <code>final</code>	53
The role of expressions	54
Programming principles	55
Programming pitfalls	55
Grammar spot	56
New language elements	56
Summary	57

Exercises	57
Answers to self-test questions	59
5 Methods and parameters	60
Introduction	60
Writing your own methods	61
A first method	62
Calling a method	64
Passing parameters	64
Formal and actual parameters	66
A triangle method	67
Local variables	70
Name clashes	71
Event-handling methods and <code>main</code>	72
<code>return</code> and results	73
Building on methods: <code>drawHouse</code>	76
Building on methods: <code>areaHouse</code>	78
<code>this</code> and objects	79
Overloading	80
Programming principles	81
Programming pitfalls	82
Grammar spot	82
New language elements	83
Summary	83
Exercises	83
Answers to self-test questions	86
6 Using objects	88
Introduction	88
Instance variables	89
Instantiation: using constructors with <code>new</code>	92
The <code>Random</code> class	92
The <code>main</code> method and <code>new</code>	97
The Swing toolkit	98
Events	98
Creating a <code>JButton</code>	99
Guidelines for using objects	101
The <code>JLabel</code> class	101
The <code>TextField</code> class	103
The <code>JPanel</code> class	104
The <code>Timer</code> class	104
The <code>JSlider</code> class	106
The <code>ImageIcon</code> class – moving an image	109
Programming principles	111
Programming pitfalls	112

Grammar spot	112
New language elements	112
Summary	112
Exercises	112
Answers to self-test questions	114
7 Selection	115
Introduction	115
The <code>if</code> statement	116
<code>if...else</code>	118
Comparison operators	121
Multiple events	129
And, or, not	131
Nested <code>ifs</code>	134
<code>switch</code>	136
Boolean variables	139
Comparing strings	143
Programming principles	143
Programming pitfalls	143
Grammar spot	145
New language elements	146
Summary	146
Exercises	147
Answers to self-test questions	149
8 Repetition	152
Introduction	152
<code>while</code>	153
<code>for</code>	158
And, or, not	159
<code>do...while</code>	161
Nested loops	163
Combining control structures	164
Programming principles	165
Programming pitfalls	165
Grammar spot	166
New language elements	166
Summary	167
Exercises	167
Answers to self-test questions	169
9 Writing classes	171
Introduction	171
Designing a class	172

Classes and files	175
<code>private</code> variables	177
<code>public</code> methods	177
The <code>get</code> and <code>set</code> methods	179
Constructors	180
Multiple constructors	181
<code>private</code> methods	182
Scope rules	184
Operations on objects	185
Object destruction	186
<code>static</code> methods	186
<code>static</code> variables	187
Programming principles	188
Programming pitfalls	189
Grammar spot	190
New language elements	190
Summary	191
Exercises	191
Answers to self-test questions	193
10 Inheritance	194
Introduction	194
Using inheritance	195
<code>protected</code>	196
Scope rules	197
Additional items	197
Overriding	198
Class diagrams	198
Inheritance at work	199
<code>super</code>	200
Constructors	200
<code>final</code>	203
Abstract classes	204
Programming principles	205
Programming pitfalls	206
New language elements	207
Summary	207
Exercises	208
Answers to self-test questions	209
11 Calculations	210
Introduction	210
Library mathematical functions and constants	211
Formatting numbers	211
Case study – money	214

Case study – iteration	217
Graphs	218
Exceptions	222
Programming principles	223
Programming pitfalls	223
Summary	223
Exercises	224
Answer to self-test question	227
12 Array lists	228
Introduction	228
Creating an array list and generics	229
Adding items to a list	229
The length of a list	230
Indices	231
Displaying an array list	231
The enhanced <code>for</code> statement	232
Using index values	233
Removing items from an array list	234
Inserting items within an array list	235
Lookup	235
Arithmetic on an array list	236
Searching	238
Programming principles	239
Programming pitfalls	240
New language elements	240
Summary	240
Exercises	241
Answers to self-test questions	241
13 Arrays	242
Introduction	242
Creating an array	244
Indices	245
The length of an array	247
Passing arrays as parameters	247
The enhanced <code>for</code> statement	248
Using constants with arrays	249
Initializing an array	250
A sample program	251
Lookup	253
Searching	254
Arrays of objects	256
Programming principles	257
Programming pitfalls	258

Grammar spot	259
Summary	259
Exercises	260
Answers to self-test questions	263
14 Arrays – two dimensional	265
Introduction	265
Declaring an array	266
Indices	267
The size of an array	268
Passing arrays as parameters	269
Using constants with two-dimensional arrays	269
Initializing an array	270
A sample program	271
Programming principles	272
Programming pitfalls	273
Summary	273
Exercises	274
Answers to self-test questions	277
15 String manipulation	278
Introduction	278
Using strings – a recap	279
The characters within strings	280
A note on the <code>char</code> type	280
The <code>String</code> class	281
The <code>String</code> class methods	281
Comparing strings	283
Amending strings	285
Examining strings	286
String conversions	289
String parameters	291
An example of string processing	291
String case study – <i>Frasier</i>	292
Programming principles	296
Programming pitfalls	297
Grammar spot	297
New language elements	297
Summary	298
Exercises	298
Answer to self-test question	300
16 Exceptions	301
Introduction	301
Exceptions and objects	303

When to use exceptions	304
The jargon of exceptions	304
A try-catch example	304
try and scopes	307
The search for a catcher	308
Throwing – an introduction	309
Exception classes	310
Compilation and checked exceptions	310
Catching – the common cases	312
Using the exception class structure	314
Programming principles	314
Programming pitfalls	315
Grammar spot	315
New language elements	315
Summary	316
Exercises	316
Answers to self-test questions	317
17 Files and console applications	318
Introduction	318
File access: stream or random?	319
The essentials of streams	319
The Java I/O classes	320
The <code>BufferedReader</code> and <code>PrintWriter</code> classes	320
File output	321
File input	324
File searching	327
The <code>File</code> class	331
The <code>JFileChooser</code> class	333
Console I/O	336
The <code>System</code> class	336
Using <code>JOptionPane</code>	338
A console example: <code>Finder</code>	338
Reading from a remote site	340
Command-line arguments	342
Programming principles	344
Programming pitfalls	344
Grammar spot	344
New language elements	344
Summary	345
Exercises	346
Answers to self-test questions	347
18 Object-oriented design	348
Introduction	348
The design problem	349

Identifying objects and methods	349
Case study in design	354
Looking for reuse	360
Composition or inheritance?	361
Guidelines for class design	365
Summary	366
Exercises	367
Answers to self-test questions	368
19 Program style	369
Introduction	369
Program layout	370
Names	370
Classes	371
Comments	372
Javadoc	373
Constants	373
Methods	374
Nested <code>ifs</code>	375
Nested loops	378
Complex conditions	379
Documentation	381
Consistency	381
Programming pitfalls	382
Summary	382
Exercises	382
20 Testing	383
Introduction	383
Program specifications	384
Exhaustive testing	385
Black box (functional) testing	385
White box (structural) testing	388
Inspections and walkthroughs	390
Stepping through code	391
Incremental development	391
Programming principles	392
Summary	392
Exercises	393
Answers to self-test questions	394
21 Debugging	397
Introduction	397
Debugging without a debugger	399
Using a debugger	400
Common errors – compilation errors	401

Common errors – run-time errors	402
Common errors – logic errors	403
Common errors – misunderstanding the language	403
Summary	405
Answer to self-test question	405
22 Threads	406
Introduction	406
Threads	407
Starting a thread	411
Thread dying	412
<code>join</code>	412
The state of a thread	412
Scheduling, thread priorities and <code>yield</code>	413
Programming principles	414
Summary	414
Exercises	415
Answers to self-test questions	415
23 Interfaces	416
Introduction	416
Interfaces for design	416
Interfaces and interoperability	419
Interfaces and the Java library	420
Multiple interfaces	421
Interfaces versus abstract classes	423
Programming principles	423
Programming pitfalls	423
Grammar spot	424
New language elements	424
Summary	424
Exercises	424
Answers to self-test questions	425
24 Programming in the large – packages	426
Introduction	426
Using classes and the <code>import</code> statement	426
Creating packages using the <code>package</code> statement	427
Packages, files and folders	428
Scope rules	429
The Java library packages	429
Programming pitfalls	430
New language elements	430
Summary	430
Exercise	430
Answers to self-test questions	431

25 Polymorphism	432
Introduction	432
Polymorphism in action	433
Programming principles	437
Programming pitfalls	438
New language elements	438
Summary	439
Exercises	439
26 Java in context	441
Introduction	441
Simple	442
Object oriented	442
Platform independence (portability)	442
Performance	443
Security	444
Open source	446
The versions of Java	446
Java capabilities	447
Java libraries	447
Java beans	447
Databases – JDBC	448
Java and the Internet	449
Java and the World Wide Web	450
The opposition: Microsoft's .NET platform	451
JavaScript	452
Conclusion	453
Summary	453
Exercises	453
Appendices	
A Java libraries	454
B The Abstract Window Toolkit	496
C Applets	500
D Glossary	504
E Rules for names	506
F Keywords	507
G Scope rules (visibility)	508
H Bibliography	511
I Installing and using Java	513
 Index	 522

Supporting resources

Visit www.pearsoned.co.uk/bell to find valuable online resources:

Companion Website for students

- How to download Java 6.0
- Programs from the book
- An extra chapter on Java network programming

For instructors

- PowerPoint slides
- How to use this book as part of a course

For more information please contact your local Pearson Education sales representative or visit www.pearsoned.co.uk/bell

Introduction

What this book will tell you

This book explains how to write Java programs that run either as independent applications or as applets (part of a web page).

This book is for novices

If you have never done any programming before – if you are a complete novice – this book is for you. This book assumes no prior knowledge of programming. It starts from scratch. It is written in a simple, direct style for maximum clarity. It is aimed primarily at first-year undergraduates at universities and colleges, but it is also suitable for novices studying alone.

Why Java?

Java is probably one of the best programming languages to learn and use because of the following features.

Java is small and beautiful

The designers of Java have deliberately left out all the superfluous features of programming languages; they cut the design to the bone. The result is a language that has all the necessary features, combined in an elegant and logical way. The design is lean and mean. It is easy to learn, but powerful.

Java is object oriented

Object-oriented languages are the latest and most successful approach to programming. Object-oriented programming is the most popular approach to programming. Java is

completely object oriented from the ground up. It is not a language that has had object-orientedness grafted onto it as an afterthought.

Java supports the Internet

A major motivation for Java is to enable people to develop programs that use the Internet and the World Wide Web. Java applets can easily be invoked from web browsers to provide valuable and spectacular facilities. In addition, Java programs can be easily transmitted around the Internet and run on any computer.

Java is general purpose

Java is a truly general-purpose language. Anything that C++, Visual Basic, etc., can do, so can Java.

Java is platform independent

Java programs will run on almost all computers and mobile phones and with nearly all operating systems – unchanged! Try that with any other programming language. (You almost certainly can't!) This is summed up in the slogan ‘write once – run anywhere’.

Java has libraries

Because Java is a small language, most of its functionality is provided by pieces of program held in libraries. A whole host of library software is available to do graphics, access the Internet, provide graphical user interfaces (GUIs) and many other things.

You will need

To learn to program you need a computer and some software. A typical system is a PC (personal computer) with the Java Software Development Kit (JDK). This is also available for Unix, GNU/Linux and Apple systems. This kit allows you to prepare and run Java programs. There are also more convenient development environments. See Chapter 2.

Exercises are good for you

If you were to read this book time and again until you could recite it backwards, you still wouldn't be able to write programs. The practical work of writing programs and program fragments is vital to becoming fluent and confident at programming.

There are exercises for the reader at the end of each chapter. Please do some of them to enhance your ability to program.

There are also short self-test questions throughout the text with answers at the end of the chapter, so that you can check you have understood things properly.

What's included?

This book explains the fundamentals of programming:

- variables;
- assignment;
- input and output;
- calculation;
- graphics and windows programming;
- selection using `if`;
- repetition using `while`.

It also covers integer numbers, floating-point numbers and character strings. Arrays are also described. All these are topics that are fundamental, whatever kind of programming you go on to do.

This book also thoroughly addresses the object-oriented aspects of programming:

- using library classes;
- writing classes;
- using objects;
- using methods.

We also look at some of the more sophisticated aspects of object-oriented programming, like:

- inheritance;
- polymorphism;
- interfaces.

What's not included

This book describes the essentials of Java. It does not explain the bits and pieces, the bells and whistles. Thus the reader is freed from unnecessary detail and can concentrate on mastering Java and programming in general.

Applications or applets?

There are two distinct types of Java program:

- a distinct free-standing program (this is called an application);
- a program invoked from a web browser (this is called an applet).

In this book we concentrate on applications, because we believe that this is the main way in which Java is being used. (We explain how to run applets in Appendix C.)

● Graphics or text?

Throughout the text we have emphasized programs that use graphical images rather than text input and output. We think they are more fun, more interesting and clearly demonstrate all the important principles of programming. We haven't ignored programs that input and output text – they are included, but they come second best.

● Graphical user interfaces (GUIs)

The programs we present use many of the features of a GUI, such as windows, buttons, scrollbars and using the mouse in lots of different ways.

● AWT or Swing?

There are two Java mechanisms for creating and using GUIs – AWT and Swing. The Swing set of user-interface components is more complete and powerful than the AWT set. This book adopts the Swing approach because it is being used more widely.

● The sequence of material

Programming involves many challenging ideas, and one of the problems of writing a book about programming is deciding how and when to introduce new ideas. We introduce simple ideas early and more sophisticated ideas later on. We use objects from an early stage. Then later we see how to write new objects. Our approach is to start with ideas like variables and assignment, then introduce selection and looping, and then go on to objects and classes (the object-oriented features). We also wanted to make sure that the fun element of programming is paramount, so we use graphics right from the start.

● Bit by bit

In this book we introduce new ideas carefully one at a time, rather than all at once. So there is a single chapter on writing methods, for example.

● Computer applications

Computers are used in many different applications and this book uses examples from all these areas:

- information processing;
- games;
- scientific calculations.

The reader can choose to concentrate on those application areas of interest and spend less time on the other areas.

Different kinds of programming

There are many different kinds of programming – examples are procedural, logic, functional, spreadsheet, visual and object-oriented programming. This book is about the dominant type of programming – object-oriented programming (OOP) – as practised in languages like Visual Basic, C++, C#, Eiffel and Smalltalk.

Which version of Java?

This book uses Java 6.

Have fun

Programming is creative and interesting, particularly in Java. Please have fun!

Visit our website

All the programs presented in this book are available on our website, which can be reached via: www.pearsoned.co.uk/bell

Changes to this edition

If you have used earlier editions of this book, you might like to know what is different about this edition.

The latest version of Java is version 6. This book accords with version 6. There are no changes to the Java language or to the library classes that we use. All the programs in the book work with version 6. This has actually meant no changes to the programs from the last edition.

The main changes for this 6th edition are:

- Chapter 2, ‘First programs’, and Appendix I. We have enhanced the explanation to include some treatment of integrated development environments (IDEs).
- The CD. In an era of broadband, we have eliminated the CD. Everything, and more, is on the website.
- Chapter 26 on the role of Java in the world is thoroughly updated.
- There are light-touch improvements throughout to enhance readability

We hope you like the changes.

SELF-TEST QUESTION

5.7 Here is a method named `twice`, which returns the doubled value of its `int` parameter:

```
private int twice(int n) {
    return 2 * n;
}
```

Here are some calls:

```
int n = 3;
int r;
r = twice(n);
r = twice(n + 1);
r = twice(n) + 2;
r = twice(1 + 2 * n);
r = twice(twice(n));
r = twice(twice(n) + 1);
r = twice(twice(n) + 1);
r = twice(twice(twice(n)));
```

For each call, state the returned value.

Building on methods: `drawHouse`

As an example of methods which make use of other methods, let us create a method which draws a primitive “lean-to” house with a cross-section shown in Figure 5.7. The height of the roof is the same as the height of the walls, and the width of the walls is the same as the width of the roof. We will choose the `int` parameters to be:

- the horizontal position of the top right point of the roof;
- the vertical position of the top right point of the roof;



Figure 5.7 House with width of 100 and roof height of 50.

Numerous **self-test questions** throughout the book, and **exercises** at the end of every chapter allow the student to practise with the concepts until they fully understand them. The answers to the self-test questions appear at the end of each chapter.

Exercises

- Movie theatre (cinema) price** Write a program to work out how much a person pays to go to the cinema. The program should input an age from a slider or a text field and then decide on the following basis:
 - under 5, free;
 - aged 5 to 12, half price;
 - aged 13 to 54, full price;
 - aged 55, or over, free.
- The elevator** Write a program to simulate a very primitive elevator. The elevator is represented as a filled black square, displayed in a tall, thin, white panel. Provide two buttons – one to make it move 20 pixels up the panel and one to make it move down. Then enhance the program to make sure that the elevator does not go too high or too low.
- Sorting** Write a program to input numbers from three sliders, or three text fields, and display them in increasing numerical size.
- Betting** A group of people are betting on the outcome of three throws of a die. A person bets \$1 on predicting the outcome of the three throws. Write a program that uses the random number method to simulate three throws of a die and displays the winnings according to the following rules:
 - all three throws are sixes: win \$20;
 - all three throws are the same (but not sixes): win \$10;
 - any two of the three throws are the same: win \$5.
- Digital combination safe** Write a program to act as the digital combination lock for a safe. Create three buttons, representing the numbers 1, 2 and 3. The user clicks on the buttons, attempting to enter the correct numbers (say 331121). The program remains unhelpfully quiet until the correct buttons are pressed. Then it congratulates the user with a suitable message. A button is provided to restart. Enhance the program so that it has another button which allows the user to change the safe's combination, provided that the correct code has just been entered.
- Deal a card** Write a program with a single button on it which, when clicked on, randomly selects a single playing card. First use the random number generator in the library to create a number in the range 1 to 4. Then convert the number to a suit (heart, diamond, club and spade). Next, use the random number generator to create a random number in the range 1 to 13. Convert the number to an ace, 2, 3, etc., and finally display the value of the chosen card. Hint: use `switch` as appropriate.
- Rock, scissors, paper game** In its original form, each of the two players simultaneously chooses one of rock, scissors or paper. Rock beats scissors, paper beats rock and scissors beats paper. If both players choose the same, it is a draw. Write a program to play the game. The player selects one of three buttons, marked rock, scissors or paper. The

New language elements reiterate the new syntax features introduced by the chapter.

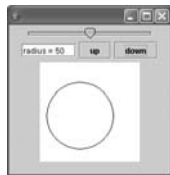


Figure 18.1 The balloon program.

Here, for example, is the specification for the simple balloon program:

Write a program to represent a balloon and manipulate the balloon via a GUI. The balloon is displayed as a circle in a panel. Using buttons, the position of the balloon can be changed by moving it a fixed distance up or down. Using a slider, the radius of the balloon can be altered. The radius is displayed in a text field.

The window is shown in Figure 18.1.

We look for verbs and nouns in the specification. In the above specification, we can see the following nouns:

```
GUI, panel, button, slider, text field, balloon, position, distance, radius
```

The GUI provides the user interface to the program. It consists of buttons, a slider, a text field and a panel. The GUI is represented by an object that is an instance of the class `JFrame`. The button, slider, text field and panel objects are available as classes in the Java library.

The GUI object:

1. Creates the buttons, slider, text field and panel on the screen.
2. Handles the events from mouse-clicks on the button and slider.
3. Creates any other objects that are needed, such as the balloon object.
4. Calls the methods of the balloon object.

The next major object is the balloon. It makes use of information to represent its position (*x* and *y* coordinates), the distance it moves and its radius. One option would be to create completely distinct full-blown objects to represent these items. But it is simpler to represent them as `int` variables.

Programs that use graphical images (particularly GUIs) rather than text input–output programs are used throughout. This demonstrates the creative and exciting side of programming which helps the student learn concepts faster.

New language elements

- **extends** – means that this class inherits from another named class.
- **protected** – the description of a variable or method that is accessible from within the class or any subclass (but not from elsewhere).
- **abstract** – the description of an abstract class that cannot be created but is provided only to be used in inheritance.
- **abstract** – the description of a method that is simply given as a header and must be provided by a subclass.
- **super** – the name of the superclass of a class, the class it inherits from.
- **final** – describes a method or variable that cannot be overridden.

Summary

Extending (inheriting) the facilities of a class is a good way to make use of existing parts of programs (classes).

A subclass inherits the facilities of its immediate superclass and all the superclasses above it in the inheritance tree.

A class has only one immediate superclass. (It can only inherit from one class.) This is called *single inheritance* in the jargon of OOP.

A class can extend the facilities of an existing class by providing one or more of:

- additional methods;
- additional variables;
- methods that override (act instead of) methods in the superclass.

A variable or method can be described as having one of three types of access:

- **public** – accessible from any class.
- **private** – accessible only from within this class.
- **protected** – accessible only from within this class and any subclass.

A class diagram is a tree showing the inheritance relationships.

The name of the superclass of a class is referred to by the word **super**.

An abstract class is described as **abstract**. It cannot be instantiated to give an object, because it is incomplete. An abstract class provides useful variables and methods for inheritance by subclasses.

Summaries offer a concise round-up of the key concepts covered by each chapter. They tie in with the objectives listed at the beginning of the chapter and are a great reference and revision aid.



Figure 7.11 The shop sign.

When the Closed button is clicked on:

```
open = false;
```

When the On button is clicked on, the value of `open` is tested with an if statement and the appropriate sign displayed:

```
if (open) {
    textField.setText("Open");
} else {
    textField.setText("Closed");
}
```

The complete program is:

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

public class ShopSign extends JFrame implements ActionListener {
    private JButton onButton, offButton, openButton, closeButton;
    private JTextField textField;

    private boolean on = false, open = false;

    public static void main(String[] args) {
        ShopSign demo = new ShopSign();
        demo.setSize(250, 200);
        demo.createGUI();
        demo.setVisible(true);
    }
}
```

```
private void createGUI() {
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    Container window = getContentPane();
    window.setLayout(new FlowLayout());
    onButton = new JButton("On");
    window.add(onButton);
    onButton.addActionListener(this);
    offButton = new JButton("Off");
    window.add(offButton);
    offButton.addActionListener(this);
    textField = new JTextField(4);
    textField.setText(" ");
    textField.setFont(new Font(null, Font.BOLD, 60));
    window.add(textField);
    openButton = new JButton("Open");
    window.add(openButton);
    openButton.addActionListener(this);
    closeButton = new JButton("Closed");
    window.add(closeButton);
    closeButton.addActionListener(this);
}

public void actionPerformed(ActionEvent event) {
    Object source = event.getSource();
    if (source == onButton) {
        handleOnButton();
    } else if (source == offButton) {
        handleOffButton();
    } else if (source == openButton) {
        handleOpenButton();
    } else handleCloseButton();
    drawSign();
}

private void handleOnButton() {
    on = true;
}

private void handleOffButton() {
    on = false;
}

private void handleOpenButton() {
    open = true;
}

private void handleCloseButton() {
    open = false;
}
```

Programming pitfalls highlight common programming mistakes and how to avoid them.

Example code is included in the text – this edition uses Swing throughout.

Programming pitfalls

- The method header must include type names. The following is wrong:

```
private void methodName(x) { // wrong
```

 Instead we must put, for example, the following:

```
private void methodName(int x) {
```
- A method call must not include type names. For example, rather than:

```
methodName(int y); //
```

 we put:

```
methodName(y);
```
- When calling a method, we must supply the correct number of parameters and the correct types of parameters.
- We must arrange to consume a returned value in some way. The following style of call does not consume a return value:

```
sumMethod(a, b); //
```

Grammar spot

- The general pattern for methods takes two forms. Firstly, for a method that does not return a result, we declare the method by:

```
private void methodName(formal parameter list) {
    ... body
}
```

 and we call the method by a statement, as in:

```
methodName(actual parameter list);
```
- For a method which returns a result, the form is:

```
private type methodName(actual parameter list) {
    ... body
}
```

 Any type or class can be specified as the returned type. We call the method as part of an expression. For example:

```
a = methodName(a, b);
```

APPENDIX

C



Applets

Introduction

This book has been about writing 'applications'. They run under the control of your operating system and the Java code and corresponding class files are stored on your computer. Applets are different. The term means a small program. Compiled applet class files are uploaded to a web server computer, in the same folder as you might store your web pages. It is possible to specify that a web page links up to an applet. When a user downloads such a web page, the Java class code comes with it, and the applet runs in an area of the web browser window.

An applet example

Here we will look at the process of creating and running an applet. We will use the `SumTextFields` program of Chapter 6. Note that, in Appendix B, we provided an AWT version of this program, and this is the version we will convert to an applet. The reason for this choice is because AWT applets will work with all browsers, but Swing support in browsers is not as widespread. Figure C.1 shows the applet running within a web browser. Here is the code:

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class SumTextFieldsApplet
    extends Applet implements ActionListener {
    private TextField number1Field, number2Field, sumField;
    private JLabel equalsLabel;
    private JButton plusButton;
```

Grammar spot identifies the correct way to write code, reinforcing the student's understanding of Java syntax.

Appendices broaden the student's understanding of Java programming.

CHAPTER 1



The background to Java

This chapter explains:

- how and why Java came into being;
- the main features of Java;
- the introductory concepts of programming.

The history of Java

A computer program is a series of instructions that are obeyed by a computer. The point of the instructions is to carry out a task, e.g. play a game, send an email, etc. The instructions are written in a particular style: they must conform to the rules of the programming language we choose. There are hundreds of programming languages, but only a few have made an impact and become widely used. The history of programming languages is a form of evolution, and here we will look at the roots of Java. The names of the older languages are not important, but we provide them for completeness.

Around 1960, a programming language named Algol 60 was created. ('Algol' from the term 'algorithm' – a series of steps that can be performed to solve a problem.) This was popular in academic circles, but its ideas persisted longer than its use. At this time, other languages were more popular: COBOL for data processing, and Fortran for scientific work. In the UK, an extended version of Algol 60 was created (CPL – Combined Programming Language), which was soon simplified into basic CPL, or BCPL.

We then move to Bell Laboratories USA, where Dennis Ritchie and others transformed BCPL into a language named B, which was then enhanced to become C,

around 1970. C was tremendously popular. It was used to write the Unix operating system, and, much later, Linus Torvalds used it to write a major part of Unix – named Linux – for PCs.

The next step came when C++ ('C plus plus') was created around 1980 by Bjarne Stroustrup, also at Bell Labs. This made possible the creation and reuse of separate sections of code, in a style known as 'object-oriented programming'. (In C, you could use ++ to add one to an item – hence C++ is one up from C.) C++ is still popular, but hard to use – it takes a lot of study.

Now we move to Sun Microsystems in the USA. In the early 1990s, James Gosling was designing a new language named Oak, intended to be used in consumer electronics products. This project never came to fruition, but the Oak language became renamed Java (after the coffee).

In parallel, the Internet was becoming more popular, and a small company called Netscape had created a web browser.

After discussions with Microsoft, Netscape agreed to provide support for Java in its web browser, with the result that Java programs could be downloaded alongside web pages. This provided a programming capability to enhance static pages. These programs were known as 'applets'. Netscape decided to allow users to download its browser for free, and this also spread the word about Java.

● The main features of Java

When James Gosling designed Java, he didn't create something from nothing. Rather, he took existing concepts, and integrated them to form a new language. Here are its main features:

- Java programs look similar to C++ programs. This meant that the C++ community would take it seriously, and also meant that C++ programmers can be productive quickly.
- Java was designed with the Internet in mind. As well as creating conventional programs, applets can be created which run 'inside' a web page. Java also had facilities for transferring data over the Internet in a variety of ways.
- Java programs are portable: they can run on any type of computer. In order for this to happen, a Java 'run-time system' has to be written for every type of computer, and this has been done for virtually all types of computer in use today. Java is also available for cell or mobile phones, so, in a sense, the abandoned Oak project has come to fruition.
- Java applets are secure. Computer viruses are widespread, and downloading and running programs over the Internet can be risky. However, the design of Java applets means that they are secure, and will not infect your computer with a virus.

- Though not a technical issue, Sun's marketing of Java is worthy of note. All the software needed to create and run Java programs was made available free, as an Internet download. This meant that Java became popular quickly. In addition, the Netscape web browser supported Java from its early days, and Microsoft provided similar facilities in its Internet Explorer web browser.

Java was very well received in industry because of its portability and Internet features. It was also well received in education, as it provides full object-oriented facilities in a simpler way than in C++.

Although Java is relatively new, it was influential in the design of Microsoft's C# (C Sharp) language. From an educational point of view, familiarity with Java will enable you to move to C# relatively easily.

What is a program?

In this section we try to give the reader some impression of what a program is. One way to understand is by using analogies with recipes, musical scores and knitting patterns. Even the instructions on a bottle of hair shampoo are a simple program:

```
wet hair
apply shampoo
massage shampoo into hair
rinse
```

This program is a list of instructions for a human being, but it does demonstrate one important aspect of a computer program – a program is a sequence of instructions that is obeyed, starting at the first instruction and going on from one to the next until the sequence is complete. A recipe, musical score and a knitting pattern are similar – they constitute a list of instructions that are obeyed in sequence. In the case of a knitting pattern, knitting machines exist which are fed with a program of instructions, which they then carry out (or 'execute'). This is what a computer is – it is a machine that automatically obeys a sequence of instructions, a program. (In fact, if we make an error in the instructions, the computer is likely to do the wrong task.) The set of instructions that are available for a computer to obey typically includes:

- input a number;
- input some characters (letters and digits);
- output some characters;
- do a calculation;
- output a number;
- output some graphical image to the screen;
- respond to a button on the screen being clicked on by the mouse.

The job of programming is one of selecting from this list those instructions that will carry out the required task. These instructions are written in a specialized language called a programming language. Java is one of many such languages. Learning to program means learning about the facilities of the programming language and how to combine them so as to do something you want. The example of musical scores illustrates another aspect of programs. It is common in music to repeat sections, e.g. a chorus section. Musical notation saves the composer from duplicating those parts of the score that are repeated and, instead, provides a notation specifying that a section of music is repeated. The same is true in a program; it is often the case that some action has to be repeated: for example, in a word-processing program, searching through a passage of text for the occurrence of a word. Repetition (or iteration) is common in programs, and Java has special instructions to accomplish this.

Recipes sometimes say something like: ‘if you haven’t got fresh peas, use frozen’. This illustrates another aspect of programs – they often carry out a test and then do one of two things depending on the result of the test. This is called selection, and, as with repetition, Java has special facilities to accomplish it.

If you have ever used a recipe to prepare a meal, you may well have got to a particular step in the recipe only to find that you have to refer to another recipe. For example, you might have to turn to another page to find out how to cook rice, before combining it with the rest of the meal: the rice preparation has been separated out as a sub-task. This way of writing instructions has an important analogue in programming, called methods in Java and other object-oriented languages. Methods are used in all programming languages, but sometimes go under other names, such as functions, procedures, subroutines or sub-programs.

Methods are sub-tasks, and are so called because they are a method for doing something. Using methods promotes simplicity where there might otherwise be complexity.

Now consider cooking a curry. A few years ago, the recipe would suggest that you buy fresh spices, grind them and fry them. Nowadays, though, you can buy ready-made sauces. Our task has become simpler. The analogy with programming is that the task becomes easier if we can select from a set of ready-made ‘objects’ such as buttons, scrollbars and databases. Java comes with a large set of objects that we can incorporate in our program, rather than creating the whole thing from scratch.

To sum up, a program is a list of instructions that can be obeyed automatically by a computer. A program consists of combinations of:

- sequences;
- repetitions;
- selections;
- methods;
- ready-made objects;
- objects you write yourself.

All modern programming languages share these features.

SELF-TEST QUESTIONS

- 1.1 Here are some instructions for calculating an employee's pay:

```
obtain the number of hours worked
calculate pay
print pay slip
subtract deductions for illness
```

Is there a major error?

- 1.2 Take the instruction:

```
massage shampoo into hair
```

and express it in a more detailed way, incorporating the concept of repetition.

- 1.3 Here are some instructions displayed on a roller-coaster ride:

```
Only take the ride if you are over 8 or younger than 70!
```

Is there a problem with the notice? How would you rewrite it to improve it?

Programming principles

- Programs consist of instructions combined with the concepts of sequence, selection, repetition and sub-tasks.
- The programming task becomes simpler if we can make use of ready-made components.

Programming pitfalls

Human error can creep into programs – such as placing instructions in the wrong order.

Summary

- Java is an object-oriented language, derived from C++.
- Java programs are portable: they can run on most types of computer.
- Java is integrated with web browsers. Applet programs can be executed by web browsers.
- A program is a list of instructions that are obeyed automatically by a computer.
- Currently the main trend in programming practice is the object-oriented programming (OOP) approach, and Java fully supports it.

Exercises

- 1.1** This question concerns the steps that a student goes through to wake up and get to college. Here is a suggestion for the first few steps:

```
wake up  
dress  
eat breakfast  
brush teeth  
...
```

- (a) Complete the steps. Note that there is no ideal answer – the steps will vary between individuals.
 - (b) The 'brush teeth' step contains repetition – we do it again and again. Identify another step that contains repetition.
 - (c) Identify a step that contains a selection.
 - (d) Take one of the steps, and break it down into smaller steps.
- 1.2** You are provided with a huge pile of paper containing 10000 numbers, in no particular order. Write down the process that you would go through to find the largest number. Ensure that your process is clear and unambiguous. Identify any selection and repetition in your process.
- 1.3** For the game of Tic Tac Toe (noughts and crosses), try to write down a set of precise instructions which enables a player to win. If this is not possible, try to ensure that a player does not lose.

Answers to self-test questions

1.1 The major error is that the deductions part comes too late. It should precede the printing.

1.2 We might say:

keep massaging your hair until it is washed.

or:

As long as your hair is not washed, keep massaging.

1.3 The problem is with the word 'or'. Someone who is 73 is also over 8, and could therefore ride.

We could replace 'or' by 'and' to make it technically correct, but the notice might still be misunderstood. We might also put:

only take this ride if you are between 8 and 70

but be prepared to modify the notice again when hordes of 8 and 70 year olds ask if they can ride!

CHAPTER 2



First programs

This chapter explains:

- how to create, compile and run Java programs;
- the use of an integrated development environment;
- the ideas of classes, objects and methods;
- how to display a message dialog;
- how to place text in a text field.

Introduction

To learn how to program in Java you will need access to a computer with Java facilities, but fortunately the Java language has been designed to run on any operating system. Currently, the most widely used operating systems are Microsoft's Windows systems on PCs, but other operating systems in use are GNU/Linux on PCs and OS X on Apple Mac. Java can run on any of these systems. This is a major benefit, but it means that the detailed instructions for using Java will vary. Here we provide general information only. Appendix I provides more details about how to obtain free Java systems.

When Java has been installed, there are four stages involved:

- creating a new file or project;
- entering/modifying the program with an editor;
- compiling the program;
- running the program.

Integrated development environments

There are two main ways to create and run your programs. Firstly, you might choose an integrated development environment (IDE). This is a software package designed to help with the complete process of creating and running a Java program. If you use an IDE, it is still a good idea to understand the ideas of files, editing, compilation and running, as described below.

There are several IDEs. One of the most popular free ones is Eclipse (itself written in Java). Refer to Appendix I for more information. Alternatively, you can use a text editor (rather like a simple word processor) to create your programs. Some of the more powerful ones (such as Textpad) can be configured to link up Java software, allowing you to initiate compiling and running by clicking on a menu option.

Files and folders

The programs that are automatically loaded and run when the computer is switched on are collectively called the operating system. One major part of an operating system is concerned with storing files, and here we provide a brief introduction.

Information stored on a computer disk is stored in files, just as information stored in filing cabinets in an office is stored in files.

Normally you set up a file to contain related information. For example:

- a letter to your mother;
- a list of students on a particular course;
- a list of friends, with names, addresses and telephone numbers.

Each file has its own name, chosen by the person who created it. It is usual, as you might expect, to choose a name that clearly describes what is in the file. A file name has an *extension* – a part on the end – that describes the type of information that is held in the file. For example, a file called `letter1` that holds a letter and is normally edited with a word processor might have the extension `.doc` (short for document) so that its full name is `letter1.doc`. A file that holds a Java program has the extension `.java`, so that a typical file name might be `Game.java`.

A group of related files is collected together into a folder (sometimes called a directory). So, in a particular folder you might hold all letters sent to the bank. In another folder you might store all the sales figures for one year. Certainly you will keep all the files that are used in a single Java program in the same folder. You give each folder a name – usually a meaningful name – that helps you to find it.

Normally folders are themselves grouped together in a folder. So you might have a folder called `Toms` within which are the folders `myprogs`, `letters`.

You might think that this will go on for ever, and indeed you can set up folders of folders *ad infinitum*. Your computer system will typically have hundreds of folders and thousands of files. Some of these will be yours (you can set them up and alter them) and some of them will belong to the operating system (leave them alone!).

So, a file is a collection of information with a name. Related files are collected together into a folder, which also has a name.

To actually see lists of folders and the files they contain, we make use of a program known as *Windows Explorer* on Microsoft Windows systems. Clicking on a folder reveals the files it contains. GNU/Linux and Apple Macs have similar facilities.

SELF-TEST QUESTION

- 2.1 (a) What is the difference between a folder and a directory?
(b) What is a folder?
(c) Is it possible to create two folders with the same name?

● Creating a Java program

Whether you use a powerful IDE or a simpler text editor, there are a number of steps that need to be worked through.

Creating a new file

If you use a text editor, you will create a new file, which needs to have a `.java` extension. In an IDE, you will create a project, which consists of a number of files. The only file you will modify is the `.java` file.

Editing the file

This involves typing in and modifying the program. (We provide an example program below.) Obviously, a text editor can be used, but all IDEs contain a text editor component. The editor is where you will spend a lot of time, so it is important to explore its advanced facilities, such as those for searching and replacing text.

Compiling the program

A compiler is a program that converts a program written in a language like Java into the language that the computer understands. So a compiler is like an automatic translator, able to translate one (computer) language into another. Java programs are converted into bytecode. Bytecode is not exactly the same as the language that a computer understands (machine code). Instead, it is an idealized machine language that means that your Java program will run on any type of computer. When your program is run, the bytecode is interpreted by a program called the Java Virtual Machine (JVM).

We will click on a button or menu option to begin the compilation process.

As it compiles your program, the Java compiler checks that the program obeys the rules of programming in Java and, if something is wrong, displays appropriate error messages. It also checks that the programs in any libraries that you are using are being employed correctly. It is rare (even for experienced programmers) to have a program compile correctly first time, so don't be disappointed if you get some error messages. Here is an example of an error message:

```
Hello.java:9: ';' expected
```

This message provides the name of the file, the line number of the error (9 in this case) and a description of the error. We need to understand the error (this is not always obvious) and then return to the editor to correct the text.

Running the program

When we have removed any compilation errors, the program can now be run (executed, interpreted). To initiate the JVM, we will click on a button or use a menu option. We now see the effect of the program as it does its job.

Note that with some IDEs, clicking to start the compilation process will also initiate a run of the program if no compilation errors were found.

SELF-TEST QUESTION

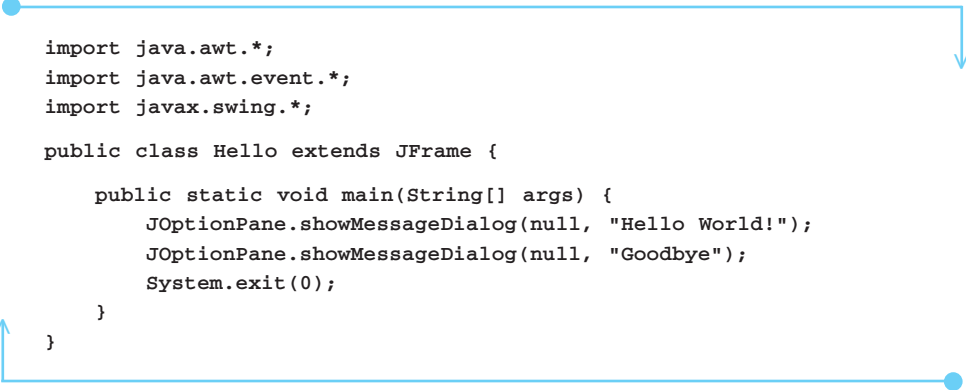
- 2.2 (a) Find out how to start and use your editor/IDE.
- (b) In your editor, enter some text containing the word "he" several times. Find out how your editor can be used to replace every occurrence of "he" by "she" with a single command.

A first program

Using the editor or IDE, create a new file (named `Hello.java`) or project, then key in the small Java program shown below.

A file that holds a Java program must have the extension `java`. The first part of the name must match the name that follows the words `public class` in the Java code. This name can be chosen by the programmer.

Do not worry about what it means, at this stage. You will see that the program contains certain unusual characters and three different kinds of bracket. You might have to search for them on your keyboard. The text that you have entered is known as the Java *code*.



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

public class Hello extends JFrame {

    public static void main(String[] args) {
        JOptionPane.showMessageDialog(null, "Hello World!");
        JOptionPane.showMessageDialog(null, "Goodbye");
        System.exit(0);
    }
}
```

Undoubtedly you will make mistakes when you key in this program. You can use the editor to correct the program.

When the program looks correct, try to compile it, and observe any error messages. Fix them by comparing your code with our version

One of the standing jokes of programming is that error messages from compilers are often cryptic and unhelpful. The compiler will indicate (note: not pinpoint) the position of the errors. Study what you have keyed in and try to see what is wrong. Common errors are:

- semicolons missing or in the wrong place;
- brackets missing;
- single quotes (') rather than double quotes (").

Identify your error, edit the program and recompile. This is when your patience is on test! Repeat until you have eradicated the errors.

SELF-TEST QUESTION

- 2.3** Make an intentional error in your code, by omitting a semicolon. Observe the error message that the compiler produces. Finally, put the semicolon back.

After editing and compiling with no errors, we can run (or execute) the program. The compiler creates a file on disk with the extension **.class**. The first part of the name matches the Java program name – so in this example the file name is:

Hello.class

This is the file that the JVM will run. A button-click or menu option will start the process.



Figure 2.1 Screenshot of `Hello.java`.

The program now runs. First, it displays the message:

`Hello World!`

Click on **OK** to close the message and to display the next message:

`Goodbye`

Click on **OK** again. The program terminates. Figure 2.1 shows a screenshot of the first message.

The libraries

As we saw, the output from the compiler is a `.class` file, which we execute with the JVM. However, the class file does not contain the complete program. In fact, every Java program needs some help from one or more pieces of program that are held in libraries. In computer terms, a library is a collection of already-written useful pieces of program, kept in files. Your small sample program needs to make use of such a piece of program to display information on the screen. In order to accomplish this, the requisite piece of program has to be linked to your program when it is run.

The libraries are collections of useful parts. Suppose you were going to design a new motor car. You would probably want to design the body shape and the interior layout. But you would probably want to make use of an engine that someone else had designed and built. Similarly, you might well use the wheels that some other manufacturer had produced. So, some elements of the car would be new, and some would be off the shelf. The off-the-shelf components are like the pieces of program in the Java library. For example, our example program makes use of a pop-up message dialog from a library.

Things can go wrong when the compiler checks the links to library software and you may get a cryptic error message. Common errors are:

- the library is missing;
- you have misspelled the name of something in the library.

The libraries are incorporated into your program when it runs.

SELF-TEST QUESTION

2.4 (a) Find two errors in this code:

```
JOptionPane.showMessageDialog(null, "Hello world");
```

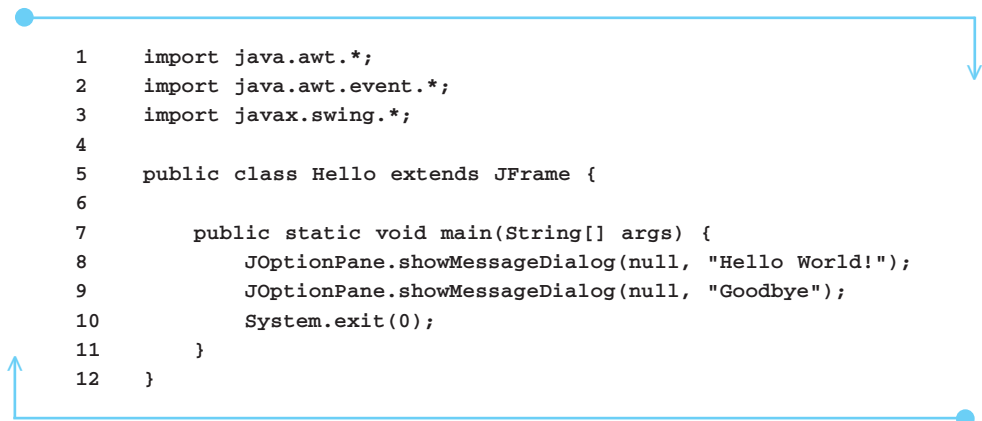
(b) Which error will prevent the program from running?

Demystifying the program

We will now provide an overview of the Java program. Even though it is quite small, you can see that the program has quite a lot to it. This is because Java is a real industrial-strength language, and even the smallest program needs some major ingredients. Note that at this early stage, we do not cover every detail. This comes in the following chapters.

We show the code of the program here again, in Figure 2.2. This time it has line numbers to help with the explanation. (Line numbers must not be part of a real program.) Lines 8 and 9 are the most important pieces of this program. They instruct the computer to display some text in a pop-up rectangle known as a message dialog, from the `JOptionPane` class. Line 8 displays the text `Hello World!`, which must be enclosed in double quotes. Text in quotes like this is called a string. The line ends with a semicolon, as do many lines in Java. Similarly, line 9 displays `"Goodbye"`. In Java, the letter `J` (standing for Java, of course) precedes many names (such as `JOptionPane`).

Figure 2.1 shows the effect of line 8. When OK is clicked, the program proceeds to line 9. This time the string `"Goodbye"` is displayed. Note that the message dialogs are displayed in sequence, working down the program.



```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class Hello extends JFrame {
6
7      public static void main(String[] args) {
8          JOptionPane.showMessageDialog(null, "Hello World!");
9          JOptionPane.showMessageDialog(null, "Goodbye");
10         System.exit(0);
11     }
12 }
```

Figure 2.2 The `Hello` program.

SELF-TEST QUESTION

- 2.5** Alter the program so that it displays a third message dialog, showing the string `"Finishing now"`.

At the top, lines 1, 2 and 3 specify information about the library programs that the program uses. The word `import` is followed by the name of a library that is to be used by the program. This program uses the AWT (Abstract Window Toolkit) and the Swing library in order to display a message dialog.

Line 4 is a blank line. We can use blank lines anywhere, to make a program more readable.

Line 5 is a heading which announces that this code is a program named `Hello`.

The program itself is enclosed within the curly brackets. The opening `{` in line 5 goes with (matches) the closing `}` on line 12. Within these lines, there are more curly brackets. The `{` at line 7 goes with the `}` at line 11.

Line 10 causes the program to stop running.

Later, we will present a longer program which displays text on the screen in a different way. Before we do, let us look at the use of ‘objects’ in Java.

Objects, methods: an introduction

One of the reasons for Java’s popularity is that it is *object oriented*. This is the book’s main theme, and we cover it at length in future chapters. But here we will introduce the concept of objects via an analogy.

Firstly, consider a home with a CD player in the kitchen and an identical one in the bedroom. In the Java jargon, we regard them as ‘objects’. Next, we consider what facilities each CD player provides for us. In other words, what buttons does a CD player provide? In the Java jargon, each facility is termed a ‘method’. (For example, we might have a start and a stop method, and a method for skipping to a track via its number.)

The term *method* is rather strange, and it comes from the history of programming languages. Imagine it as meaning ‘function’ or ‘facility’, as in ‘This CD player has a start and a stop method.’

Now consider the task of identifying each button on each player. It is not enough to state:

```
stop
```

because there are two players. Which player do we mean? Instead, we must identify the player as well. Moving slightly closer to Java, we use:

```
kitchenCD.stop
```

Note the use of the dot. It is used in a similar way in all object-oriented languages. We have two items:

- The name of an object; this usually corresponds to a noun.
- A method which the object provides; this is often a verb.

Note that you can imagine that the dot means 's in English, as in:

```
kitchenCD's stop button
```

Later, when we discuss methods in more detail, you will see that the exact Java version of the above is:

```
kitchenCD.stop();
```

Observe the semicolon and the brackets with nothing between them. For some other methods, we might have to supply additional information for the method to work on, such as selecting a numbered track:

```
bedroomCD.select(4);
```

The item in brackets is known as a 'parameter'. (Again a traditional programming term rather than an instantly meaningful one.)

In general terms, the way we use methods is:

```
object.method(parameters);
```

If the particular method does not need parameters, we must still use the brackets. We cover parameters and methods in Chapter 5.

SELF-TEST QUESTION

- 2.6** Assume that our kitchen and bedroom CD players have facilities (methods) for stopping, starting and selecting a numbered track. Here is an example of using one method:

```
kitchenCD.select(6);
```

Give five examples of using the methods.

Classes: an analogy

The concept of a class is extremely important in object-oriented programming. Recall our analogy: we have two identical CD player objects in our house. In object-oriented jargon, we have two 'instances' of the CD player 'class'. A class is rather like a production line which can manufacture new CD players.

Let us distinguish between a class and instances of a class. The house has two instances of the CD player class, which really exist: we can actually use them. A class is a more abstract concept. Though the CD player production line possesses the design (in some form or another) of a CD player, an actual instance does not exist until the

machine manufactures one. In Java, we use the word ‘new’ to instruct a class to manufacture a new instance. To summarize:

- objects are instances of a class;
- a class can produce as many instances as we require.

It is worth repeating that these concepts are the main ones of this book, and we cover them later in much more detail. We do not expect you to be able to create Java programs with objects and classes in this chapter.

Using a text field

The first program we saw used a message dialog. The second program we will introduce is rather longer, but it forms the basis for many of the programs in this book. It uses a *text field* to display a single line of text – the string “Hello!” in this case. Figure 2.3 shows a screenshot, and Figure 2.4 shows the code with line numbers.

At this stage, we need to remind you again that the details of Java really begin in the next chapter. For now, we are showing you some programs, and providing a general explanation of what they do. We do not (yet) expect you to be able to look at a line of code, and say precisely what it does.

As before, we provide line numbers to assist in our explanations, but the numbers should not be typed in. The name following the `public class` words is `Greeting`, so the program must be saved in a file named:

```
Greeting.java
```

Compile and run the program. To stop the program, click on the cross at the top right of the window, or click on the Java icon at the top left, then select **Close** from the menu.

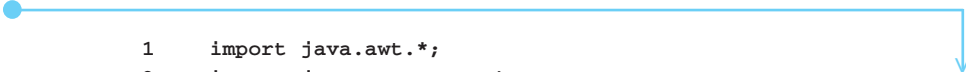
We will now look at some of the uses of instances, methods and parameters.

Recall our analogies. We mentioned the use of the ‘dot’ notation for objects and their associated methods. Locate line 11:

```
frame.setSize(300, 200);
```



Figure 2.3 Screenshot of `Greetings.java`.



```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class Greeting extends JFrame {
6
7      private JTextField textField;
8
9      public static void main (String[] args) {
10         Greeting frame = new Greeting();
11         frame.setSize(300, 200);
12         frame.createGUI();
13         frame.setVisible(true);
14     }
15
16     private void createGUI() {
17         setDefaultCloseOperation(EXIT_ON_CLOSE);
18         Container window = getContentPane();
19         window.setLayout(new FlowLayout() );
20         textField = new JTextField("Hello!");
21         window.add(textField);
22     }
23 }
```




Figure 2.4 The `Greeting` program.

This uses the same notation – object, dot, method, parameters.

Imagine the frame object as the outer edge of the screenshot of Figure 2.3. The `setSize` method takes two parameters – the required width and height of the frame in units known as pixels. Here, we have used an object-oriented approach to setting the size of the frame.

SELF-TEST QUESTION

2.7 Modify the `Greeting` program so that the frame is half as wide.

Here is another use of objects. Locate the following lines in the program:

```
textField = new JTextField("Hello!");
window.add(textField);
```

Firstly, a text field is being created, using the word `new`. At this stage, we can choose the text that will appear in the text field, though this can also be overtyped by the user when the program runs. Next, the window object has the text field added to it. When the program runs, the text field is displayed, and is centred automatically.

SELF-TEST QUESTIONS

2.8 Modify the program to display the text:

`Some text in a text field`

2.9 Run the **Greeting** program and experiment with resizing the frame by dragging it with the mouse. What happens to the position of the text field?

Finally, a general point about our second program. Most of the instructions are concerned with stating which libraries are needed, and setting up the visual appearance of the screen, i.e. the ‘graphical user interface’ or GUI.

Imagine the GUI of your favourite word processor. Across the top of its window, you will see a large number of menus and buttons. When you run the word processor, they all appear instantly, so it might surprise you to learn that, behind the scenes, it starts with a totally blank window, and laboriously adds each menu and button to the window, one by one. Because of the speed of the computer, this process seems instantaneous.

When you write larger programs, the initial setup of the screen still has to be done, but that part of the code becomes less dominant in proportion to the code concerned with making the program carry out a task when a button or menu item is clicked on.

Programming principles

- A major feature of Java is the widespread use of classes.
- The ‘dot’ notation for using objects is:

`object.method(parameters)`

- Instructions are obeyed in sequence, from the top of the program to the bottom.

Programming pitfalls

- When you are editing a program, save it every 10 minutes or so to guard against losing your work should the computer fail.
- Make sure that when you key in a program, you copy the characters exactly, with capitals as shown.
- Make sure that the name of the file matches the name of the class in the file. For example:

```
public class Hello extends JFrame {
```

Here, the name following **class** is **Hello**. The file must be saved in **Hello.java**. The capital letter of the class is important.

- You will almost certainly make a mistake when you key in a program. The compiler will tell you what the errors are. Try not to get too frustrated by the errors.

Grammar spot

Java programs contain a number of opening and closing brackets. There must be the same number of closing brackets as opening brackets.

New language elements

A message dialog can display a text string along with an **OK** button, as in:

```
JOptionPane.showMessageDialog(null, "Hello World!");
```

Summary

- Java programs can be created and executed on most types of computer.
- An editor or IDE is used to create and modify your Java source code.
- A Java program must be compiled prior to running.
- Compilation errors must be corrected before a program can run.
- The compiler produces a file with the same name as the original Java file, but with the extension **class**. This file contains bytecode instructions.
- The JVM (Java Virtual Machine) is used to run (execute) programs.
- Much of the power of Java comes from its libraries, which are linked in as the program runs.
- Java is object oriented. It uses the concepts of classes, instances and methods.
- The ‘dot’ notation occurs throughout Java programs. Here is an example:

```
frame.setSize(300, 200);
```

- Methods (such as **setSize**) cause tasks to be performed on the specified object (such as **frame**).
- Things can go wrong at any stage, and part of the programmer’s job is identifying and correcting the errors. Don’t forget: it is rare for everything to work smoothly first time. Be careful, be relaxed.

Exercises

- 2.1 Ensure that you know how to compile and run Java programs on your computer. Compile and run the two programs from this chapter.
- 2.2 In the **Hello** program, add a message dialog to display your name.
- 2.3 In the **Greeting** program, make the text field display your name.

Answers to self-test questions

- 2.1 (a) No difference. The terms mean the same thing.
(b) A folder contains a number of files and/or other folders.
(c) Yes. As long as the two identically named folders are not within the same folder. (For example, the folders **Work** and **Home** might each contain a **letters** folder.)
- 2.2 (a) This depends on the editors available on your computer. If you use an IDE, the editor is contained within it.
(b) This depends on your editor. Many editors have a **Find...Replace** facility, which scans all of the text.
- 2.3 The error message will vary, depending on which semicolon you omitted.
- 2.4 (a) There is an incorrect 'p'. It should be 'P' as in **JOptionPane**. There is a misspelling of **wirld**.
(b) The 'p' error will prevent the program from compiling, hence it cannot run. The 'wirld' error will not prevent the program running, but the result will not be as you intended.

- 2.5 Insert the following line immediately below line 9, which displays **"Goodbye"**:

```
JOptionPane.showMessageDialog(null, "Finishing now");
```

Compile and run the modified program.

- 2.6

```
kitchenCD.start();  
kitchenCD.stop();  
bedroomCD.start();  
bedroomCD.stop();  
bedroomCD.select(3);
```

- 2.7 Replace **300** by **150** in line 11, then recompile and run the program.

- 2.8 Replace **"Hello!"** in line 20 by:

```
"Some text in a text field"
```

Compile and run the program.

- 2.9 It remains centred near the top of the frame.

CHAPTER

3



Using graphics methods

This chapter explains:

- the nature of events;
- how to draw shapes with graphics methods;
- the use of parameters;
- how to comment programs;
- how to use colours;
- the sequence concept.

Introduction

The term *computer graphics* conjures up a variety of possibilities. We could be discussing a computer-generated Hollywood movie, a static photo, or a simpler image made up of lines. In this chapter we restrict ourselves to still images built from simple shapes, and focus on the use of library methods to create them. Our programs also introduce the use of a button to allow user interaction.

Events

Many programs are built in such a way to allow user interaction via a GUI. Such programs provide buttons, text fields, scrollbars, etc. In Java terms, the user manipulates the mouse and keyboard, creating ‘events’ which the program responds to. Typical events are:

- a mouse-click;
- a key press;
- using a slider to scroll through some values.

The Java system regards events as falling into several categories. For example, scrolling through a page is regarded as a ‘change’ event, whereas clicking on a button is regarded as an ‘action’ event.

When you write a Java program, you must ensure that the program will detect the events – otherwise nothing will happen. The transmission of an event (such as a mouse-click) to a program does not happen automatically. Instead, the program has to be set up to ‘listen’ for types of event. Fortunately, the coding for this is standard, and you will reuse it from program to program rather than creating it anew for every program.

Responding to an event is known as ‘handling’ the event.

Here is a program which provides a button. Figure 3.1 shows the screenshot.

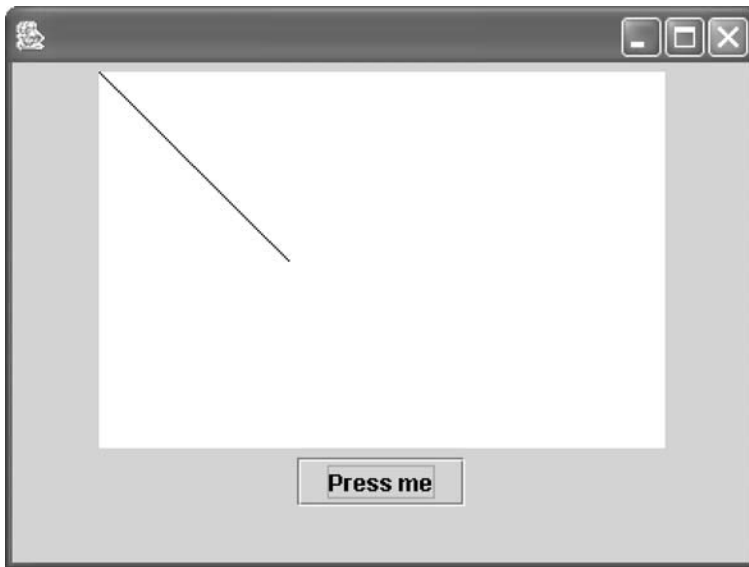


Figure 3.1 Screenshot of `DrawExample` program, after clicking on the button.

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

public class DrawExample extends JFrame
    implements ActionListener {

    private JButton button;
    private JPanel panel;
```

```

public static void main(String[] args) {
    DrawExample frame = new DrawExample();
    frame.setSize(400, 300);
    frame.createGUI();
    frame.setVisible(true);
}

private void createGUI() {
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    Container window = getContentPane();
    window.setLayout(new FlowLayout() );

    panel = new JPanel();
    panel.setPreferredSize(new Dimension(300, 200));
    panel.setBackground(Color.white);
    window.add(panel);

    button = new JButton("Press me");
    window.add(button);
    button.addActionListener(this);
}

public void actionPerformed(ActionEvent event) {
    Graphics paper = panel.getGraphics();
    paper.drawLine(0, 0, 100, 100);
}
}

```

The user clicks on the button, and a diagonal line is drawn. For the purpose of this chapter, we are mainly interested in this instruction:

```
paper.drawLine(0, 0, 100, 100);
```

As you might expect, this instruction actually draws the line. The rest of the code sets up the GUI, and we will discuss it only in general terms. Setting up the GUI involves:

- adding a button to the window, in a similar way that we added a text field in Chapter 2;
- adding a ‘panel’ to be used for drawing;
- stating that the program will listen for mouse-clicks (categorized as action events).

The following point is very important: in later chapters, we cover the creation of user interfaces. For now, treat the above GUI code as standard.

● The button-click event

The main event in this program is created when the user clicks on the **"Press me"** button. A button-click causes the program to execute this section of program:

```
public void actionPerformed(ActionEvent event) {  
    Graphics paper = panel.getGraphics();  
    paper.drawLine(0, 0, 100, 100);  
}
```

This section of program is a *method*, named `actionPerformed`. When the button is clicked on, the Java system calls up (invokes) the method, and the instructions between the opening `{` and the closing `}` are executed in sequence. This is where we place our drawing instructions. We will now look at the details of drawing shapes.

The graphics coordinate system

Java graphics are based on pixels. A pixel is a small dot on the screen which can be set to a particular colour. Each pixel is identified by a pair of numbers (its coordinates), starting from zero:

- the horizontal position, often referred to as x in mathematics (and in the Java documentation) – this value increases from left to right;
- the vertical position, often referred to as y – this value increases downwards. Note that this differs from the convention in mathematics.

We use this system when we request Java to draw shapes. Figure 3.2 shows the approach. The top left of the drawing area is $(0, 0)$, and we draw relative to this point.

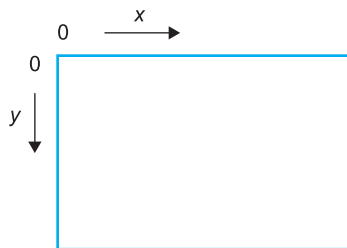


Figure 3.2 The pixel coordinate system.

Explanation of the program

The only section we are concerned with is the small part which does the drawing:

```
1 public void actionPerformed(ActionEvent event) {  
2     Graphics paper = panel.getGraphics();  
3     paper.drawLine(0, 0, 100, 100);  
4 }
```

Line 1 introduces the section of program which is executed when the button is clicked on. It is a method. Any instructions we place between the `{` of line 1 and the `}` of line 4 are executed in sequence, down the page.

Line 2 provides a graphics area for drawing shapes – we have chosen to name it **paper**. Recall Chapter 2, where we stated that Java is object oriented. Objects provide facilities for us. We considered a CD player, which provides a range of facilities, such as:

```
kitchenCD.select(4);
```

In fact, our drawing area is not just a blank sheet of paper – it is more like a drawing kit which comes with paper together with a set of tools, such as a ruler and protractor.

In line 3, the **paper** uses its **drawLine** method to draw a line on itself. The four numbers in brackets specify the position of the line.

The **drawLine** method is one of the many methods provided by the Java system in a library. Line 3 is a call (also known as an invocation) of the method, asking it to carry out the task of displaying a line.

When we make use of the **drawLine** method, we supply it with some values for the start and finish points of the line, and we need to get these in the correct order, which is:

1. the horizontal value (x) of the start of the line;
2. the vertical value (y) of the start of the line;
3. the horizontal value of the end of the line;
4. the vertical value of the end of the line.

The items are known as parameters in Java – they are inputs to the **drawLine** method. Parameters must be enclosed in brackets and separated by commas. (You may encounter the term *argument*, which is an alternative name for a parameter.) This particular method requires four parameters, and they must be integers (whole numbers). If we attempt to use the wrong number of parameters, or the wrong type, we get an error message from the compiler. We need to ensure that:

- we supply the correct number of parameters;
- we supply the correct type of parameters;
- we arrange them in the right order.

Some methods do not require any parameters. In this case, we must still use the brackets, as in:

```
frame.createGUI();
```

There are two kinds of method at work in our example:

- Those that the programmer writes, such as **actionPerformed**. This is called up by the Java system when the button is clicked on.
- Those that are pre-written in the libraries, such as **drawLine**. Our program calls them.

A final point – note the semicolon ‘;’ at the end of the **drawLine** parameters. In Java, a semicolon must appear at the end of every ‘statement’. But what is a statement? The answer is not trivial! As you can see from the above program, a semicolon does not occur at the end of every line. Rather than provide intricate formal rules here, our advice

is to base your initial programs on our examples. However, the use of a method followed by its parameters is in fact a statement, so a semicolon is required.

Methods for drawing

As well as lines, the Java library provides us with facilities for drawing:

- rectangles;
- ovals (hence circles).

Here we list the parameters for each method, and provide an example program which uses them.

`drawLine`

- the horizontal value of the start of the line;
- the vertical value of the start of the line;
- the horizontal value of the end of the line;
- the vertical value of the end of the line.

`drawRect`

- the horizontal value of the top left corner;
- the vertical value of the top left corner;
- the width of the rectangle;
- the height of the rectangle.

`drawOval`

Imagine the oval squeezed inside a rectangle. We provide:

- the horizontal value of the top left corner of the rectangle;
- the vertical value of the top left corner of the rectangle;
- the width of the rectangle;
- the height of the rectangle.

The following shapes can also be drawn, but require additional Java knowledge. We will omit their parameter details, and won't use them in our programs.

- arcs (sectors of a circle);
- raised (three-dimensional) rectangles;
- rectangles with rounded corners;
- polygons.

Additionally, we can draw solid shapes with `fillRect` and `fillOval`. Their parameters are identical to those of the draw equivalents.

Drawing with colours

It is possible to set the colour to be used for drawing. There are 13 standard colours:

<code>black</code>	<code>blue</code>	<code>cyan</code>	<code>darkGray</code>
<code>gray</code>	<code>green</code>	<code>lightGray</code>	<code>magenta</code>
<code>orange</code>	<code>pink</code>	<code>red</code>	<code>white</code>
<code>yellow</code>			

(`cyan` is a deep green/blue, and `magenta` is a deep red/blue).

Take care with the spellings – note the use of capitals in the middle of the names. Here is how you might use the colours:

```
paper.setColor(Color.red);
paper.drawLine(0, 0, 100, 50);
paper.setColor(Color.green);
paper.drawOval(100, 100, 50, 50);
```

The above code draws a red line, then a green unfilled oval. If you don't set a colour, Java chooses black.

Creating a new program

In the above `DrawExample` program, we concentrated on its `actionPerformed` method, which contained a call of the `drawLine` method. Our focus was to learn about calling and passing parameters to the drawing methods. But what about the other lines of code? They are concerned with such tasks as:

- creating the outer frame (window) for the program;
- setting the size of the frame;
- adding the drawing area and button to the user interface.

These tasks are accomplished by calling methods. We explain the details in later chapters.

In fact, for every program in this chapter, the setting up of the user interface is identical. All the programs use a drawing area and a single button to initiate the drawing. However, we cannot use the identical code for each program, because the file name that we choose must match the name of the `public class` within the program. Look at the `DrawExample` program. It is stored in a file named `DrawExample.java`, and it contains the line:

```
public class DrawExample extends JFrame
```

The class name must start with a capital letter, and the name can only contain letters and digits. It cannot contain punctuation such as commas, full stops and hyphens, and cannot contain spaces. Choosing the class name fixes the file name we must use to contain the program.

There is an additional line:

```
DrawExample frame = new DrawExample();
```

This is contained within the **main** method of the program. When we run a Java program, the very first thing that happens is an automatic call of the **main** method. The first task of **main** is to create a new instance (an object) of the appropriate class (**DrawExample** here). In Chapter 6 we will examine the use of **new** to create new instances. For now, note that the **DrawExample** program contains three occurrences of the **DrawExample** name:

- one after the words **public class**;
- two in the **main** method.

When we create a new program, these three occurrences must be changed.

Here is an example. We will create a new program, named **DrawCircle**. The steps are:

1. Open the existing **DrawExample.java** file in any editor, select all the text, and copy it to the clipboard.
2. Open the software that you use to create and run Java programs (an IDE such as Eclipse, or a text editor).
3. If you are using an IDE, you will need to create a new project at this stage. The name of the project could be **DrawCircleProject**. (The name need not be related to **DrawCircle**, but it makes project identification easier.)
4. Create a new blank Java file, named **DrawCircle.java**. If you use an IDE, you may need to delete some code that the IDE creates for you.
5. Paste in the copied code, and change the three occurrences of **DrawExample** into **DrawCircle**.

You can now focus on the main topic of this chapter, and place appropriate calls of the drawing methods within the **actionPerformed** method.

SELF-TEST QUESTION

- 3.1** Create a new program named **DrawCircle**. When the single button is clicked on, it should draw a circle 100 pixels in diameter.

The sequence concept

When we have a number of statements in a program, they are performed from top to bottom, in sequence (unless we specify otherwise using the later concepts of selection and repetition). Here is a program which draws a variety of shapes. Figure 3.3 shows the resulting output. In the following listing we have omitted the code which creates the user interface, as this part is exactly the same as the previous program.

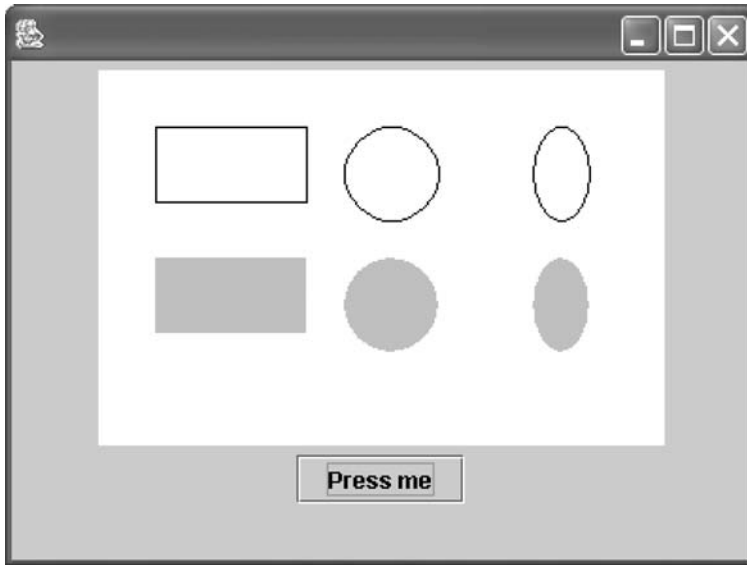


Figure 3.3 Screenshot of the `SomeShapes` program.

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

public class SomeShapes extends JFrame
    implements ActionListener {

    //      GUI code omitted here...

    public void actionPerformed(ActionEvent event) {
        Graphics paper = panel.getGraphics();
        paper.drawRect(30, 30, 80, 40);
        paper.drawOval(130, 30, 50, 50);
        paper.drawOval(230, 30, 30, 50);
        paper.setColor(Color.lightGray);
        paper.fillRect(30, 100, 80, 40);
        paper.fillOval(130, 100, 50, 50);
        paper.fillOval(230, 100, 30, 50);
    }
}
```

The statements are obeyed (executed, performed, . . .) from top to bottom, down the page – though this is impossible to observe because of the speed of the computer. In future chapters, you will see that we can repeat a sequence of instructions over and over again.