Java Spring Series

Java Programming: Beginner to Intermediate

Unlock the World of Java Programming
From Your First Line of Code to Real-World Skills

Gerry Byrne
David Wilson

Java Programming

Beginner to Intermediate

Step by step instructions
for practical hands-on programming

Gerry Byrne and David Wilson

ABOUT THE AUTHORS

Gerry Byrne and David Wilson are Senior Technical Trainers for a Forbes 100 company. They work to upskill and reskill software engineers who develop business-critical software applications. They also help refine the programming skills of returners to the workforce and introduce new graduates to the application of software development within the commercial environment.

Their subject expertise has been developed over multi-decade careers as teachers, lecturers, and technical trainers in a corporate technology environment. They have delivered a range of courses across computer languages and frameworks and understand how to teach skills and impart knowledge to a range of learners. They have taught software engineers in the use of modern technologies and frameworks such as C#, Java, Spring, Android, JavaScript, Node, HTML, CSS, Bootstrap, React, Python, and Test-Driven Development, not to mention legacy technologies such as COBOL and JCL.

Gerry and David have mastered how to teach difficult concepts in a simple way that makes learning accessible and enjoyable. Their delivery and content follow the same philosophy of keeping it simple, while making the instructions detailed and applying concepts to real-world scenarios. They are passionate about software development and believe we can all learn to write code if we are patient, grasp the basic coding concepts, get plenty of hands-on coding, and most of all, persevere through 'thick and thin.'

DEDICATION

Writing a book is a rewarding undertaking, but it requires time, effort and patience. It requires patience from those who help you write the book and those around you in your life.

So, we start by thanking our families for 'facilitating' us as we worked over many hours, days, weeks and months to write this programming book.

We also wish to thank each other, we have learnt so much in writing this book and in delivering many enjoyable programming courses. When we need coding inspiration, we work together, we try things, and we persevere in pursuit of excellence.

ACKNOWLEDGMENTS

We would like to express our deepest gratitude to everyone who contributed to the creation of this book. We are especially grateful to those who offered invaluable guidance and insightful feedback during the writing process.

Special thanks to the open-source community and all those who have contributed to the Java platform and its resources, which have been fundamental in shaping the examples and content presented here.

Finally, to all the readers, thank you for your interest and enthusiasm. We hope this book serves as a valuable resource on your journey with Java programming, and programming in general.

Contents

Introduction

The chapters in the Java Programming: Beginner to Intermediate and Java Programming: Intermediate to Advanced books will cover coding in Java using the IntelliJ IDEA, Integrated Development Environment from JetBrains. Other Integrated Development Environments exist, such as Visual Studio Code and Eclipse. The code from the applications in the chapters will work within any Integrated Development Environment capable of running Java code. Whilst the step-by-step instructions and screenshots in the book are based around the IntelliJ IDEA, Integrated Development Environment, they can still be used by those preferring a different Integrated Development Environment.

The first two chapters of this Java Programming: Beginner to Intermediate will cover Java from the very basics to get the Integrated Development Environment software installed and the Java Development Kit working with it. With the necessary tools installed, we are then introduced to what a computer program is, before we start to write our own computer programs. We then begin to cover the core concepts needed when developing Java code and which can be applied to other programming languages. We cover a wide range of core programming concepts including data types, casting and parsing, arithmetic operations, selection, iteration, arrays, collections, methods, classes and objects. Studying these chapters is more than enough to allow us to develop applications that emulate commercial application code, but they are essential if we want to progress beyond an intermediate level of programming. The core concepts in this Java Programming: Beginner to Intermediate book are required to help us advance through the intermediate to advanced concepts in the Java Programming: Intermediate to Advanced book. Having completed both books, we will have gained all the necessary concepts and skills to make us capable of being competent programmers.

All examples in the chapters are fully commented to ensure we can understand the code and to enhance our knowledge of the Java programming language. Reading the comments within the code examples is essential, they are an integral part of the book and will enhance our understanding of Java and will help explain why the code does something or what the code is doing. Read the comments, do not ignore them. In enterprise application code detailed comments would not be seen as good practice but remember, we are learning to code and need all the help we can in mastering the Java language.

This first book is ideal for beginners, those refreshing their Java skills or those moving from another object-oriented programming language. It is ideally suited for students studying

programming at high school or at university and is an excellent resource for teachers who deliver programming lessons. The book offers detailed explanations and comments to support learning. By using clean code with proper naming, the code is intuitive to read and understand.

Reading the books is one thing, but coding the examples using an Integrated Development Environment is the most important thing if we wish to get the best understanding of the Java language. Hands on experience whilst reading this book is the key to success.

We should think about two things before we begin our programming journey through this book:

Life begins at the edge of our comfort zone and

Think about now and believe.

Often the thought of getting started can make us frightened and uncomfortable. We need to believe in ourselves and understand that whilst there will be lows during the learning and coding, we will survive them and move to the inevitable highs.

Programming can be rewarding and thankfully it is within our ability to write code. The chapters in the two books, Java Programming: Beginner to Intermediate and Java Programming: Intermediate to Advanced, will help us to learn about coding, teach us how to code and make us realize that it is indeed realistic for us to program in the Java language.

As we start learning Java it is important to realize that our target of being able to write computer applications in Java will seem large, as there is a lot to learn, but we should take comfort in the fact that as we complete each chapter and gain experience in writing our applications the target gets closer, and the amount of learning gets smaller. In essence, as we move along our learning pathway, we gain competence in concepts that will be continually used in our application code.

Source code for this book is available to readers on GitHub (https://github.com/gerardbyrne/Java-Programming-Beginners-to-Intermediate.git)

1 Software Installation

Software Installation

Java Development Kit

As we want to write Java code to build applications, we will need to install the required software and tools. We will use the Java SE Development Kit for the Windows operating system, but it is available for the Mac operating system, as well as other operating systems. The software can be downloaded from the website:

https://adoptium.net/temurin/releases or https://adoptium.net/en-GB/temurin/releases/

The version of the **Java Development Kit, JDK**, to be downloaded will depend on

- The release version we wish to have on our computer and use for development. We will use Java 24 for the examples in this book.
- The computer on which it is being installed e.g., Windows, macOS or Linux. In our case it will be for a Windows platform as the chapters in the book use the instructions and screenshots from a Windows installation, but, if you choose to use a different installation the instructions and screenshots will be very similar, and little if any changes will be required. The Windows version, like the MacOS and Linux versions, have different architecture options, and by selecting the architecture we are specifying the device we wish to use with the JDK. The types of architecture include:
    - x64

      This refers to a 64-bit CPU and operating system.
    - x86 (x32)

      This represents a 32-bit CPU and operating system.
    - Aarch64, ARM32, and ARM64

      Aarch64 or ARM64 is the 64-bit Execution state of the ARM architecture family. Traditionally we would have had the x86 32-bit CPU, then we had the x64 64-bit CPU and now we have the ARM32 and ARM64 which are used on a range of devices such as mobile devices and even Internet of Things (IoT) devices.

      ARM32 and x86 are for 32-bit processors, whereas ARM64 and x64 are for 64-bit processors.

To download the required Java software go to the https://adoptium.net/temurin/releases or https://adoptium.net/en-GB/temurin/releases webpage.

- Choose the Operating System you are using e.g. **Windows**, MacOS, Linux, Solaris.

- Choose the required **Architecture,** x64 is the most likely.
- Choose **JDK** as the Package Type. There is also a JRE, but the JDK has the JRE built in.
- Choose **24** as the Version or we could choose a different version.

We will then have the option to download a .msi file or a .zip file, so which will we choose?

- A .msi file is an installer and therefore if we chose this file the software installation would launch automatically.
- A .zip file is simply an archive of the required file and will not do anything without user intervention. When the .zip file is downloaded it needs to be extracted.
- Now click on the .msi or the .zip version.

Figure 1-1 shows the steps for downloading. The webpage you are on may have a different layout, e.g., Figure 1-2, but you will still be able to make the same choices.

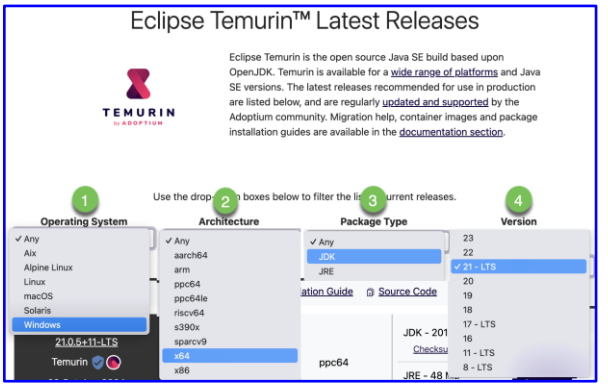

Figure 1-1. Downloading the JDK

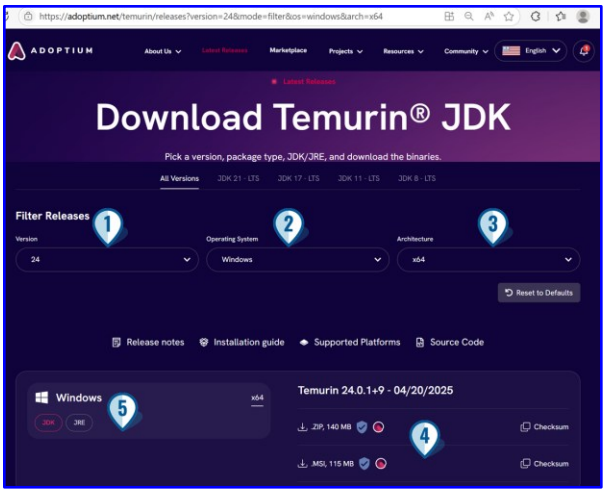

Figure 1-2. Downloading the JDK from Adoptium website

If the .msi was downloaded, follow the instructions during the automatic installation, and if the .zip file was downloaded, extract the files into a Java folder within the Program Files folder. The extraction will create a folder for the JDK, and inside this folder will be the bin folder containing the binary files. Now we need to set the PATH for the Java bin folder.

1. In the Windows search box type **Environment Variables.**
2. Click on the **Edit the system environment variables** option as shown in Figure 1-3.

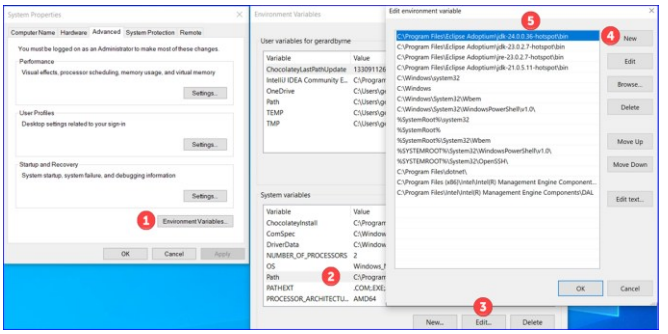

Figure 1-3. Setting the environment variables

3. Now click on the **Environment Variables** button.
4. Click on the **New** button.
5. For the variable name enter **JAVA_HOME**, as shown in Figure 1-4.
6. For the variable value enter the location of the downloaded JDK e.g., C:\Program Files\Eclipse Adoptium\jdk-24.0.0.36-hotspot\bin or click on the Browse Directory and locate the folder.

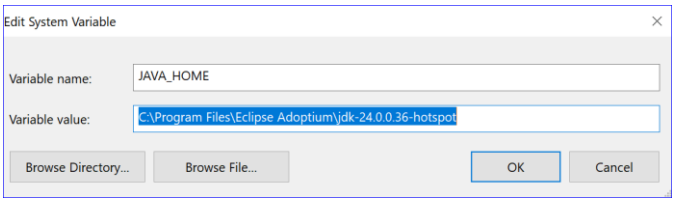

Figure 1-4. Java Home variable

7. Click the **OK** button.
8. Click on the **Path** variable in the System Variables section.
9. Click on the **Edit** button.
10. Click on the **New** button.
11. Add the location **%JAVA_HOME%\bin** as shown in Figure 1-5.

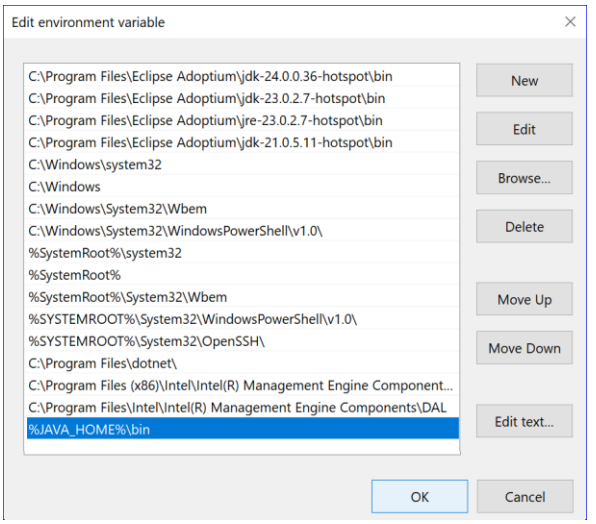

Figure 1-5. Add the %JAVA_HOME%\bin location

12. Click **OK**.
13. Click **OK**.
14. Click **OK**.

We will now check that the installation was successful.

15. In the Windows search box type **Command Prompt** or **Windows PowerShell.**
16. Click on Command Prompt or Windows PowerShell when it appears.
17. In the shell window type the command **java -version**, as shown in Figure 1-6**.**
18. Press the **Enter** key.
19. In the shell window type the command **javac -version**, as shown in Figure 1-6.
20. Press the **Enter** key.

The two commands should confirm that the installation has worked as shown in Figure 1-6.

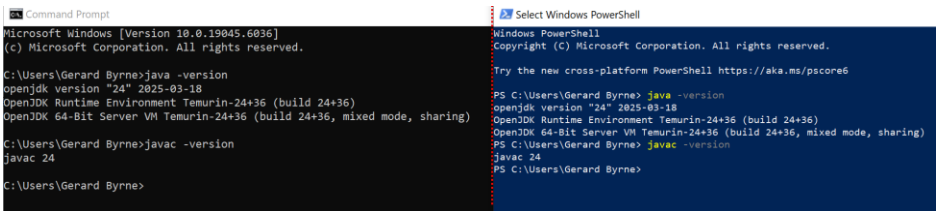

Figure 1-6. Installation and version check

Download and install IntelliJ Community Edition IDEA

When we want to develop a Java application by writing code, we could use a text editor and then compile the code from the command line, but this would be an antiquated methodology, and one that would not fit well in a modern development environment. As developers, we now have a choice of development tools to help us when writing Java code. The Integrated Development Environment, IDE, tool that we choose will generally have a visual environment that has an editor where the code is typed, and it will have a file browser. More importantly it will have tools that assist us through autocompletion of code, help by giving us warnings regarding the code that has been entered etc. There are many Integrated Development Environments available but, in this book, we will use the **JetBrains IntelliJ IDEA Community Edition**, but other editors like NetBeans, Eclipse, and Visual Studio Code can be used. All editors will have the functionality to connect to the Java Development Kit, JDK, we downloaded and can therefore be used to compile the Java we have saved to a location of our choice.

We need to install the JetBrains IntelliJ Community Edition, and the version will depend on the type of operating system we have. The software can be downloaded from the website:

https://jetbrains.com/idea/download

From the downloads we will be provided with two available editions, the free IntelliJ IDEA Community Edition and the paid IntelliJ IDEA Ultimate edition, as shown in Figure 1-7. We will install the IntelliJ IDEA Community Edition.

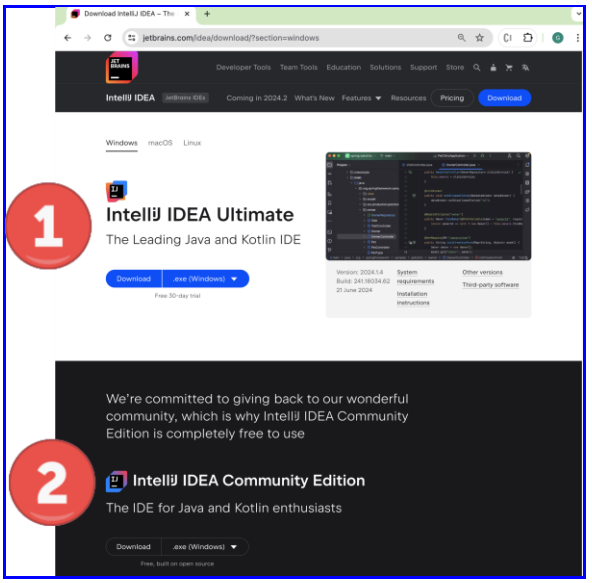

Figure 1-7. Downloading the Integrated Development Environment Software

21. Click on the dropdown in the IntelliJ IDEA Community Edition section as shown in Figure 1-8.
22. Click on the **.exe (Windows)** option, as shown in Figure 1-8.
23. Click on the Download button.

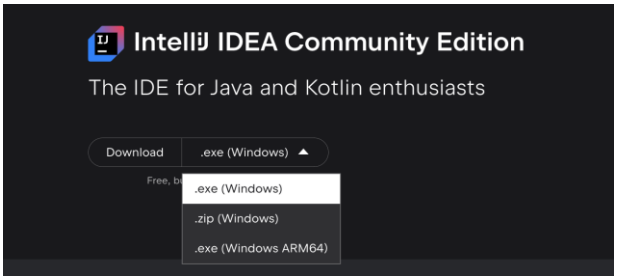

Figure 1-8. Download IntelliJ IDEA Community Edition

24. Locate the downloaded file in the download location.
25. Double click on the .exe to run the installer file.
26. Choose the default folder location.
27. Click the Next buttons until we get to the button that says Finish.
28. Click the Finish button.

Now we can open the IntelliJ IDEA Community Edition from the Programs menu.

Chapter Summary

In this chapter we have learned about the different versions of Java, and where we can access the files required to install both the JDK and JRE. We looked at the installation process and discussed the use of an Integrated Development Environment.

We then looked at the process to download and install the free edition of IntelliJ IDEA Community Edition from JetBrains. This Integrated Development Environment will be our development tool for the Java applications we will use throughout the chapters.

2 About Java

From the seed to Java

Many of the features of Java can be traced back to, and are related to, features in the C++ programming language. In turn C++ is a direct descendant of C and that is why Java is closely related to C and C++. While James Gosling at Sun Microsystems is often credited with developing Java, the language was the product of contributions from multiple individuals. At the time Java was created the existing programming languages involved the program code being written for the purpose of compiling it for a specific architecture. The program would then have to be modified for each new type of architecture it was to be run on. Whilst this was possible it was a costly and time-consuming job, and generally there was little appetite for converting the code for all possible architectures. This meant that the developers of Java had a very clear objective, develop a programming language that was independent of the platform it would run on, or to use fancy terminology, make it architecturally neutral. One big objective was that the software would be capable of running on embedded systems in the consumer electronics market. The 'dream' was a computer program written in a language that could be run on any device e.g., TV, washing machine, microwave, without having to make any changes to the underlying computer code. A write once run anywhere philosophy. When the World Wide Web emerged during the development of Java it added a new concept for the Java developers to solve. How could portability be used within the many systems that made up the World Wide Web?

What is it?

Java is a programming language and a **computing platform** which was first released by Sun Microsystems in 1995. It has evolved and is now one of the most popular programming languages used in modern software development, as shown in Figure 2-1.

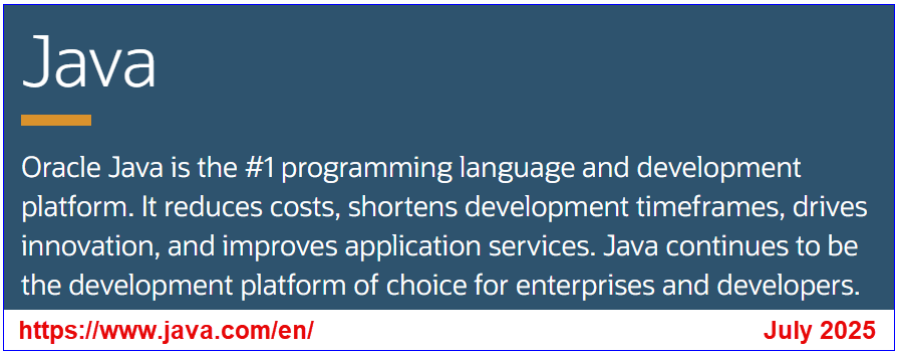

Figure 2-1. Number 1 programming language

3 Program Structure

Computer Program

We will be using Java to write computer programs, just like many programmers in companies around the world who use Java to write programs in the commercial environment. So, a very good starting point before we write programs using Java code, is to fully understand what a computer program is. We can think of a computer program as:

- A sequence of data instructions created by a programmer.
- Instructions that tell the computer what operations it should execute.
- Instructions that tell the computer how it should execute an operation.
- Instructions written in a special programming language, for example, C#, C++, COBOL.

Besides Java, there are many programming languages available to developers. Each programming language will have advantages and disadvantages when compared to the other programming languages, but they will all be useful for writing software applications. It is important to understand that some programming languages are:

- More powerful than others, for example, Java.
- Better for developing applications requiring fast processing, for example, C.
- Better for developing web-based software applications, for example, JavaScript.
- Better for developing computer games, for example, C++.
- Better for data analytics, for example, Python.
- Better for statistical analysis and data visualization, for example, R.
- Better for scripting, for example, Perl.

Once we understand the core concepts of programming, such as variables, loops, conditionals, functions, and data structures, in one language, learning another language becomes significantly easier. This is because most programming languages share fundamental principles, even if their syntax and specific features differ. So, by the time we finish reading this book, entering and running all the example code, and doing all the exercises, we will be in a strong position to recognize and apply constructs in the C# programming language or the C++ language and indeed other programming languages, as well as our focus, the Java language.

Listing 3-1 shows Java code, which will ask the user to input two values, and then totals the values. The program is like our recipe; it is a set of instructions. We will not write this code because it is being used to illustrate the point that code is like a recipe.

Listing 3-1. Sample Java program code

```
package labs.chapter3;

import java.util.Scanner;

public class Example1
{
  public static void main(String[] args)
  {
   Scanner myScanner = new Scanner(System.in);
   // Declare the variables
   int counter = 0;
   int totalofallclaims = 0;
   int inputnumber = 0;

   while (counter < 2)
   {
    System.out.println("What is the value of the claim: -- ");
    inputnumber = myScanner.nextInt();
                                                    statements (list of statements)
    // Add the number to the total
    totalofallclaims = totalofallclaims + inputnumber;

    //Add one to the value of count
    counter = counter + 1;

    // Print out the total of the claims that have been entered
    System.out.println("The total of the claims is " + totalofallclaims);
   } // End of while loop
  } // End of main() method
} // End of Example1 class
```

Listing 3-2 shows Python code, which will ask the user to input two values, and then totals the values. The program is like our recipe; it is a set of instructions. We will not write this code because it is being used to illustrate the point that code is like a recipe.

Listing 3-2. Sample Python program code

```
counter = 0
totalofallclaims = 0
                                    variables (list of ingredients)
while counter < 2:
                                        statements (list of statements)
  #Input a number
  inputnumber = int(input("What is the value of the claim:-- "))

  #Add the number to the total
  totalofallclaims = totalofallclaims + inputnumber
```

```
    #Add one to the value of count
    counter = counter + 1

# Print out the total of the claims that have been entered
print("The total of the claims that have been input is ", totalofallclaims)
```

4 Input and Output

We learned in Chapter 3 that under the direction of a program, written in a programming language, and converted to machine-readable code, the computer can perform tasks, as shown in Table 4-1.

Table 4-1. Input, output and process

Input	The computer can accept user input from the keyboard.
Process	The computer can perform arithmetic calculations and other types of processing.
Output	The computer can display a message or a result on the screen.

This chapter will concentrate on how to **output to the console**. We will also use a Java method to **read from the console**, which is an example of **input**. It is important to understand that what we learn by completing the examples in this chapter will:

- Help us build more complex code examples in future chapters.
- Show us commands that are used in real-world applications.
- Get us started with two important aspects of any programming language - **input** and **output** .

Looking back at Figure 3-2 from Chapter 3, we can think of the console as a black and white screen where input from the user is accepted and output from the computer program is displayed as shown in Figure 4-1. The console colors can be changed as we can see from the lower part of Figure 4-1.

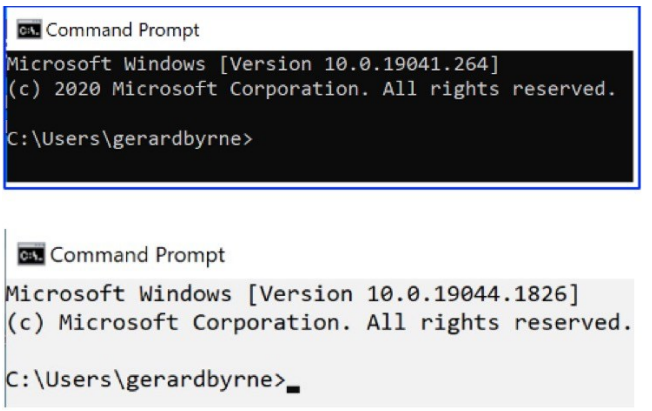

Figure 4-1. Console in black and white and alternative color

In using the Scanner class, we will use **System. in**, which represents the standard input stream.

1. Open the **IntelliJ** Integrated Development Environment.
2. Click on the **New Project** button or go to the File menu and choose New Project, as shown in Figure 4-6.
3. Enter the name **CoreJava** for the project name in the **Name** text box.
4. In the Location text box enter the location where the project is to be stored, or select the location, e.g. the location might be **C:\CoreJava.**
5. Select the Build system as **IntelliJ**.
6. Choose the **JDK** from the dropdown list, e.g. Temurin-24
7. Leave the Module name as **CoreJava**.
8. Leave the Content Root as it appears.
9. Leave the Module file location as it appears.
10. Click on the **Create** button.

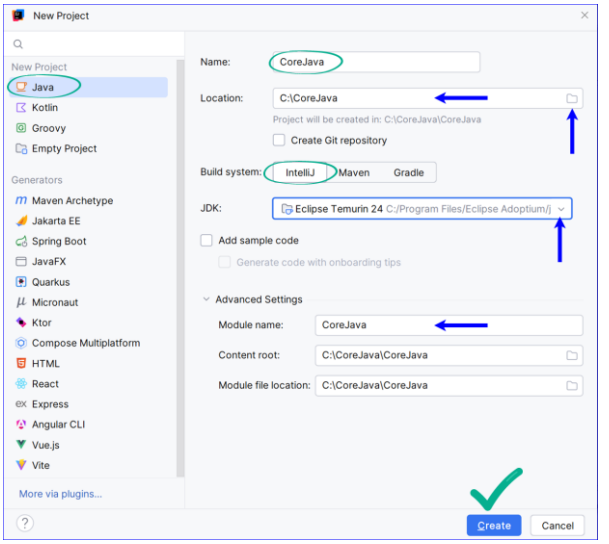

Figure 4-6. New Project

11. Click on the **File** menu.
12. Choose **Settings**.
13. Expand the **Appearance & Behavior** option.
14. Click on the **Appearance** option.
15. Choose a **Theme** from the drop-down list, as shown in Figure 4-7.
16. Change the font and font size as required.

5 Commenting Code

We learned in Chapter 4 that our application code can involve input and output, and that the input and output can be displayed in the console window. The output from our application is the visible part for an end user, and it is important they have a good experience when looking at the output. The user experience is referred to as the UX, and can involve the developer making use of colors, emphasis, layout and more, to make the application readable and pleasing to look at.

Java single line comments

Single-line comments in Java, and other programming languages, are preceded by two forward slash symbols, **//**. The // indicates a single-line comment which is used for brief comments. Some developers will write the comments above the code, whilst others will use the comment on the same line as the code. Both types are shown in Listing 5-1, Listing 5-2, and Listing 5-3

Example one

Listing 5-1 shows an example of three single-line comments, which could be used at the start of a program to give the user information about the program, the developer, and the creation date. The comments are at the start of the program code before any Java statements are entered.

Listing 5-1. Single line comments

```
// Program:   A simple Java program to output text and read input
// Authors:   Gerry Byrne and David Wilson
// Date of creation: 01/09/2027
```

Example two

Listing 5-2 shows a single-line comment which gives the user information about the class called Program that follows it. The single-line comment appears above the code statement.

Listing 5-2. Single line comment above code

```
// This is our only class and it will contain the main method
public class ConsoleV1
{
```

6 Data Types

We learned in Chapter 5 that whilst we can use single-line and multiple-line comments, they should not be a replacement for self-documenting code. Comments are added to help the reader of the code, but when the code is written expressively with proper package names, class names, variable names, etc., there is a limited need for comments. We should set an objective of zero need for comments.

In this chapter we will use code which is well documented for the purposes of helping us understand and read the code. It is not how we would do it in a real application, and if it was enterprise code, we would be breaking the objective of zero need for comments and the concept of self-documenting code.

We will now learn about the very important concepts of **data types and variables**. We will use data types and variables in all the Java programs in this book, that is how crucial they are to Java programming. We should also be aware that data types and variables exist in all programming languages and are a core building block for the code we will write.

The Java programming language is said to be **statically typed**, which means that when variable is declared we must declare the **data type** of the variable along with the **name** of the variable. There are different data types in Java, but we will now look at the category called **Value Types.** The data type of a variable will determine the value that it can contain, and the operations that can be performed on it. In Java there are built in data types, and these are called the **primitive data types**. In Java, primitive types are also **value types**. This means that variables of primitive types directly contain their values. The **eight primitive data types** supported by the Java programming language are shown in Table 6-1.

Table 6-1. Primitive data types (value types) in Java

boolean	char		
Arithmetic integral types (integer)			
byte	short	int	long
Arithmetic floating-point types (floating-point)			
float	double		

Notice that **char**, which holds one character, is a data type but there is no **String,** more than one character. Java still supports strings, but strings are a **reference** to an **instance of the**

class String. We will not worry about this statement too much, for now, we will use strings and characters in our programming. **However, note that the String type has a capital S, and the other data types have lower-case letters.**

In Java, and indeed in other languages, the value types are referred to as primitive types and primitive types are predefined by Java, with their names being **reserved keywords**. When we declare a data type, we are reserving memory to store a value. Each data type will have a particular size of memory that needs to be set aside. The primitive data types in Java along with their memory size are shown in Table 6-2.

Table 6-2. Value types in Java

Java data type	default	size	description
boolean	false	1 byte	Contains either true or false
char	\u0000	2 bytes	Contains a single character
Integral types			
byte	0	1	May contain integers from -128 to 127
short	0	2	Ranges from 32,768 to 32,767
int	0	4	Ranges from -2,147,483,648 to 2,147,483,647
long	0	8	Ranges from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
Floating-point types			
float	0.0	4	unlimited
double	0.0	8	unlimited

7 Casting and Parsing

Parsing

In Java, **parsing** is a method used to convert a **String to a primitive data type.** With parsing we use methods of type parseDATATYPE(), where DATATYPE will be replaced by the data type it is being converted to. Examples are, **parseInt(), parseDouble(), parseByte(), parseFloat().** The operation performed using the specific method is called parsing and the parsing methods are available through **Wrapper classes**. For now, we will not worry about this idea of wrapper classes, we will concentrate on the fact that we can do conversions using parsing and using casting.

Now we will look at using casting in different ways. Remember to read the comments carefully as they fully explain what we are doing.

Create a new package

1. Right click on the **code** package.
2. Choose **New**.
3. Choose **Package**.
4. Enter the new name for the package, we will call it **chapter07.**
5. Press the Enter key.
6. Expand the click **chapter06** package.
7. Right click on the **Program** file.
8. Choose **Copy**.
9. Right click on the **chapter07** package.
10. Choose **Paste**.
11. Name the class **Program**, the same name as was copied.

We will amend the Java code to use a casting from int to short, and to do this we will add two new variables, one of data type int called maximumAmountForRepairCosts, and the other of data type short called minmumAmountForRepairCosts.

12. Amend the code as shown in Listing 7-1.

Listing 7-1. Two variables, one of type int, one of type short with initial values

```
    int vehicleCurrentMileage;
    String dateOfBirthOfMainDriver;

/*
The maximum value of an int is 2,147,483,647
We could use Integer.MAX_VALUE to find it.
*/
    int maximumAmountForRepairCosts = 32767;

/*
The maximum value of a short is 32,767
We could use Short.MAX_VALUE to find it.
*/
    short maximumAmountForCarHire = 0;

    System.out.println();
    System.out.println("---- Car Quotation Application ----");
```

13. Click on the **File** menu.
14. Choose **Save All**.

We will now assign the value of the maximumAmountForRepairCosts variable to the variable called maximumAmountForCarHire.

15. Amend the code as shown in Listing 7-2.

Listing 7-2. Assign one variable value to another variable

```
        } catch (ParseException e) {
            System.out.println("EXCEPTION - enter date in yyyy-MM-dd format");
            throw new RuntimeException(e);
        }

    /*
    We are trying to put the maximumAmountForRepairCosts variable
    of data type int into the variable maximumAmountForCarHire
    which is of data type short. This is not possible without
    something being changed and this is where the error message
    will appear and a cast comes into play
    */
        maximumAmountForCarHire = maximumAmountForRepairCosts;

        // Close the Scanner object we created earlier
        myScanner.close();
    } // End of main() method
} // End of Program class
```

8 Arithmetic Operations

1. Right click on the **code** package.
2. Choose **New**.
3. Choose **Package**.
4. Name the package **chapter08**.
5. Press the Enter key.
6. Right click on the **chapter08** package.
7. Choose **New**.
8. Choose **Java Class**.
9. Name the class **Arithmetic**.

The class code in the editor window will be similar that shown in Listing 8-2.

Listing 8-2. Basic class code

```
package code.chapter08;

public class Arithmetic
{
} // End of Arithmetic class
```

We will now rename the class in the Project panel and ensure that the class is also renamed in the editor window.

10. Right click on the **Arithmetic** class in the Project panel.
11. Choose **Refactor**.
12. Choose **Rename**.
13. Change the name to **QuoteArithmetic**.
14. Click the **Refactor** button.
15. The refactoring should have amended the class name in the editor window, as shown in
16. Listing 8-3:

Listing 8-3. Renamed class code

```
package code.chapter08;

public class QuoteArithmetic
```

```
{

} // End of QuoteArithmetic class
```

The class has also been renamed in the Project panel. Now that we have renamed the class using the refactor option, we can see that this process could be useful when renaming packages, classes, variables etc. Refactoring will be a useful tool to have as we develop our coding skills.

We will now set up variables that will be used in the mathematical calculation to produce a vehicle insurance quotation. We will declare variables to hold user input and variables that will be used in calculating the quote. We will use a base mileage of 10000 kilometers and a base rate of $100 for the quotation. We will also use a base age of 10 years for the vehicle. We will then calculate the age factor, mileage factor, and their respective premiums before we calculate a discount based on the age of the vehicle and determine the final quotation value.

In the code in Listing 8-4, and future listings, there are detailed comments to help us get a full understanding of the code. In this example all code is contained within the opening and closing curly braces, , of the main() method.

First, we will add the variables we will use in our insurance quotation application code.

17. Amend the code as shown in Listing 8-4.

Listing 8-4. Declaring the variables in the main() method

```
// Program:      Performing arithmetical operations
// Authors:      Gerry Byrne and David Wilson
// Date of creation:  01/09/2027

package code.chapter08;

public class QuoteArithmetic
{
    public static void main(String[] args) {
    /*
    Setup our variables that will be used in the mathematical
    calculation used to produce a vehicle insurance quotation.
    First we will setup the variables that will hold the user
    input and that will be used in calculating the quote */
        int vehicleAgeInYears;
        int vehicleCurrentMileage;
```

```
/*
For the quotation we will use 10000 kilometers as a base line
for calculating a mileage factor. If the average kilometers
travelled per year is above the  base mileage of 10000 the
mileage factor will be above 1, if the average kilometers
travelled per year is the lower than the base mileage of
10000 the mileage factor will be below 1 */
    double quoteAverageExpectedKilometers = 10000;

/*
For the quotation we will use £100 as a base figure (this is
just an example) and this figure will be multiplied by the
mileage and age factors */
    double quoteBaseRate = 100.00;

/*
For the quotation we will use 10 as a base figure for the age
of the vehicle (this is just an example). If the vehicle is
older than 10 years, the age factor will be above 1. If the
vehicle is less than 10 years the age factor will be below 1.*/
    int quoteBaseAge = 10;

/*
This variable will be used to hold the value of
the age factor */
    double quoteAgeFactor;

/*
This variable holds the quote amount based on the age
factor and the base rate */
    double quoteAgeFactorPremium;

/*
This variable holds the quote mileage factor based on the
number of kilometers travelled each year and how the
kilometers per year is a ratio of the average expected 10000
kilometers as decided by the insurance company */
    double quoteMileageFactor;

/*
This variable will hold the amount for the quote based only
on the mileage factor. The quote also has to take into
account the age of the vehicle */
    double quoteMileageFactorPremium;

/*
This variable will hold the discount amount. A discount will
be applied to the quote based on the age of the vehicle.
The age of the vehicle is divided into 1 to get the discount.

The decimal value is a representation of the discount and
will then be multiplied by the quote value to get the actual
discount in terms of £s */
    double quoteDiscount;

/*
This variable holds the total of the age factor premium and
the mileage factor premium and will be used by the discount
calculation to get the discount amount */
    double quoteAmountForPremium;
```

```
    /*
    This variable holds the final quotation value, the premium */
        double quoteFinalAmountForPremium;
    } // End of main() method

} // End of QuoteArithmetic class
```

Now we will write some information to the console and ask for user input.

9 Selection

Selection is an important concept and will be used in many commercial applications. However, the concept of selection should be familiar to us through our everyday life. Many of the things we do in everyday life require us to make decisions and often we will be directed down one path or another. Figure 9-1 illustrates such a scenario.

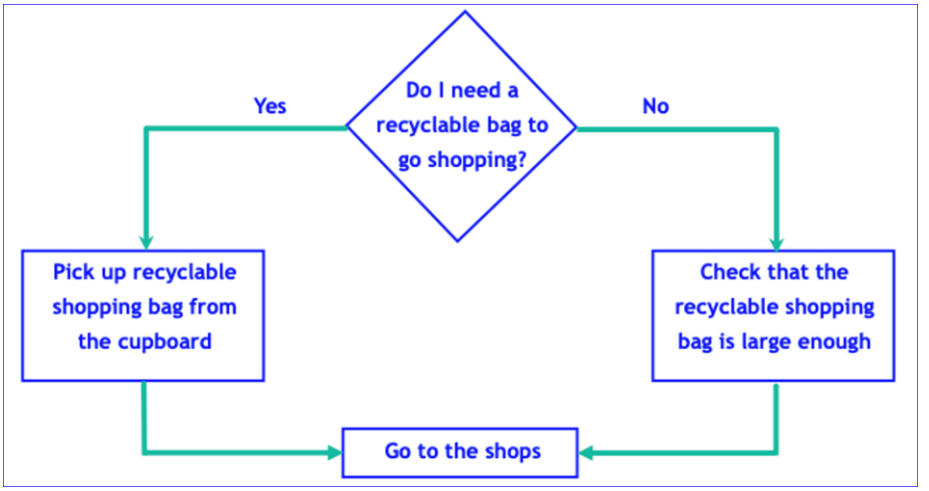

Figure 9-1. Everyday decision using a Yes or No scenario

In a similar manner the programs we, and every developer, write will normally require the use of decisions. Decisions in our code will change the execution flow depending on the decision made. Figure 9-1 shows the decision-making process where:

- a **yes** decision changes execution down the yes path and
- a **no** decision changes execution down the no path

In Figure 9-1 the execution eventually returns to a common path, **Go to the shops.** In programming, making decisions can be achieved in different ways and we will now look at the use of the **SELECTION** statements within Java.

operators, and their corresponding symbols used in Java.

10 Classes

Classes and Objects

In Chapter 13, we explored methods, blocks of reusable code designed to perform specific tasks. Methods help organize code, reduce redundancy, and improve program efficiency. As applications grow in size and complexity, methods alone are no longer sufficient for organizing code. That's where classes come in, grouping related data and behaviors into cohesive units. This leads us to classes, the foundation of object-oriented programming (OOP) in Java.

Understanding Classes

A class serves as a blueprint for creating objects, defining their properties (fields or attributes) and behaviors (methods). In simpler terms, a class is like a template, and objects are the real-world instances formed from the template.

OOP Pillars

- **Encapsulation** means keeping related data (fields, attributes) and behavior (methods) bundled together inside a class and controlling access to that data through visibility modifiers like private, protected, and public.

  We can think of a class as a protective shell. In an insurance application we could have a Policy class that stores client details, some of which is sensitive, such as coverage limits or medical history. By making the fields private and controlling access through getter and setter methods, we protect the integrity of the data.

Modeling a Physical Object

Figure 14-1 shows an image of a car which we can say is a physical object. When we look at a car we probably think of its properties subconsciously e.g., the make, the model, the age, the color, and what its top speed is.

Figure 14-1. A vehicle which is a physical object

When modeling a car, or the vehicle in Figure 14-1, in a Java class we could define these properties and behaviors

Properties (Attributes)

1. **make (String)** The manufacturer of the car e.g., Toyota, Audi, Ford.
2. **model (String)** The specific model e.g., Rav4, Q5 Mustang.
3. **year (int)** The car's year of manufacture e.g., 2025.
4. **Color (String)** The car's color e.g., blue.
5. **mileage (double)** The odometer reading for the total distance travelled e.g., 10000.

Behaviors (Methods)

1. **startEngine()** Starts the car's engine.
2. **stopEngine()** Turns off the engine.
3. **accelerate()** Increases the cars's speed.
4. **brake()** Slows the car down.
5. **displayInfo()** Prints details about the car.

The methods reflect how a car behaves or interacts with the world; all neatly bundled into one class. Once the class is defined, we can use it to create actual cars, known as **objects** e.g., we could create an object for a specific car like a Toyota Rav4 or an Audi Q5 or a Ford Mustang or any other object as shown in Figure 14-2. Each object is unique, but they all follow the structure of the car class.

Figure 14-2. Specific car types

The way forward

Now that we've covered the core concepts in Java, including classes and objects, we are in a great position to develop and read enterprise application code. But, like many things in life, there is always more to learn. Java programming is no different; there is still plenty to explore and master.

This is just the beginning. In this book **Java Programming: Beginner to Intermediate**, we have reached an intermediate level. But the journey does not end here. In the next book in the series **Java Programming: Intermediate to Advanced**, we will dive into the more advanced topics that will elevate our Java programming skills to new heights.

Get ready to explore:

- Inheritance and Polymorphism
- Interfaces
- Records, Sealed Classes, and Interfaces
- Exception Handling
- File Handling
- Serialization
- Reflection and Annotations
- Enumerations
- Generics
- Lambda Expressions
- Streams - Map, Reduce, Filter
- Multithreading
- String Handling

These are crucial topics that will significantly enhance our programming capabilities. By mastering these advanced concepts, we will be able to tackle complex programming challenges with confidence and finesse.

So, are we ready to take our Java skills to the next level? There is a lot to learn, so let's get started with **Java Programming: Intermediate to Advanced** and unlock more complex aspects of Java programming!