JAVA PROGRAMMING

FOR BEGINNERS

THEO HOULE

JAVA PROGRAMMING

JAVA

PROGRAMMING

1

TABLE OF CONTENTS

5

INTRODUCTION

Java is a popular programming language on the Internet and in computer applications. It's a free download solution that lets users access the most recent versions and apply updates. This programming language can be found in the vast majority of today's web applications and computing technologies. Java's scalability makes it suitable for use in a wide variety of applications, including apps for small electronic devices such as cell phones and software solutions for large-scale operations such as data centers. Java's growing popularity can be attributed to its robust functional features and solid security credentials.

Java is a programming language developed by Sun Microsystems, which was later acquired by Oracle Corporation. It is intended to run on any Java-enabled operating system. This is what made the language so popular: the developer only had to write the program once, and it could then run on any operating system without the programmer having to change the code.

The Java programming language is used to create the majority of modern applications around the world. The Java programming language is used to build the majority of the server-side and business logic components of major applications.

You will learn how to write programs like the one above throughout this book, as well as advanced concepts that will allow you to begin writing complete application programs.

Some of Java's design goals are listed below:

The language is designed to be written only once and run on any operating system.

The language should support several software engineering principles. Portability is an important consideration. This is why Java can run on the Windows, Linux, and macOS operating systems.

Internationalization requires a lot of support.

Java is designed to be used for developing applications for both hosted and embedded systems.

Other design objectives are discussed next:

Type checking is extremely important.

Java is a powerful type of language. Every variable that is defined must be associated with a data type.

You don't need to understand the entire program right now, so let's just look at two lines of code.

7

1) int i=5;

In this case, we're defining a variable, which is used to store a value. The amount of data that can be stored is determined by the data type. In this example, I am of the type 'int' or Integer, which is a numeric data value.

Array Bounds Validation

Java will check whether the array has the required number of values at runtime. An exception will be thrown if a value is accessed that is outside the bounds of the array.

You don't need to understand the entire program right now, so let's just look at the following lines of code.

1) int[] array1 = new int[2];

In this case, we're declaring an array, a collection of integer values. The value of ‘2’ means that we can only store two values in the array.

2) array1[0] = 1; array1[1] = 2; array1[2] = 3;

We can see from this code that we are assigning three values to the array. When you run this program, you will receive an error because the array has exceeded its maximum allowable bounds of 8

two. At runtime, you will receive the error shown below. Exception in "main" thread

At

HelloWorld.main,

java.lang.ArrayIndexOutOfBoundsException: 2 (HelloWorld.java:8)

What is the significance of Java?

Next, Java's syntax and features are similar to those of other programming languages such as C and C++. Learning Java will be a breeze if you have any prior programming experience. Even if you are completely new to programming, Java is designed to be a relatively simple language to learn. Most programmers find Java easier to learn than C or C++.

Java is also intended to be platform-agnostic. As previously stated, Java code is first compiled into bytecode, which can then be run on any machine that has the Java Virtual Machine.

As a result, with Java, you can write the code once and run it anywhere.

Why is Java used?

One of the primary reasons to use Java is its emphasis on object-oriented programming.

Object-oriented programming, also known as "OOP," is a programming language model that allows program code to be 9

organized around data rather than functions and logic, as in procedural programming.

These "data clusters" are organized into "objects," thus the term "object-oriented programming."

These objects are created by "classes," which are understood here in the traditional sense: types of objects that allow the programmer to "classify" them based on two major criteria: attributes and methods.

The attributes of a class are the raw data that will be used to create the object: these are its descriptors, such as the values it possesses, and other relevant data that will be used to create the object. The object's "method" is the second criterion.

This "method" is the class's behavior or the logical sequences that describe how it interacts or can be interacted with natively.

CHAPTER 1.

JAVA FUNDAMENTALS

One of the primary reasons to use Java is its emphasis on object-oriented programming. Object-oriented programming, also known as "OOP," is a programming language model that allows program code to be organized around data rather than functions and logic, as in procedural programming.

These objects are created by "classes," which are understood in the traditional sense of how classes are types of objects that allow the programmer to "classify" them based on two major criteria: attributes and methods.

The attributes of a class are the raw data that will be used to create the object: these are its descriptors, such as the values it possesses, and other relevant data that will be used to create the object.

The object's "method" is the second criterion.

To illustrate, suppose there is a class called "Human." Height, weight, gender, and race will be characteristics of this "class."

Methods in the "human" class can include "run," "walk," and "talk."

These theoretical components comprise the "human" class, which serves as a blueprint for an object.

Now that the class has been defined, the programmer can use the "human" class as a blueprint to create an object.

They can use the class "Human" to "populate" its attributes, such as height, weight, gender, and race. Furthermore, the object already has built-in functions such as "run," "walk," and "talk," so when an object named "Mike" from the "Human" class is created, it already contains the functions to run, walk, and talk, without the need for the programmer to code those specific functions again, as they are already "inherent" in the created object.

In a nutshell, that is what object-oriented programming is intended to be: a programming style that allows the programmer to draw on pre-defined classes to make it easier to describe them and use their internal, or built-in, functions to operate them.

Assuming the reader is not a complete novice to programming and has been introduced to the world of programming using C or another procedural-heavy language, the next logical question is: why use object-oriented programming at all?

One of its primary benefits is that it saves time in the long run.

In simpler algorithms and programs, procedural programming is usually much faster and more straightforward; rather than having to construct and define a class, and then create an object based on that class, all the programmer has to do is declare the necessary 12

variables, write the functions, and create the algorithm to solve the problem that the code is supposed to address.

However, when it comes to more complex programs that require more complex solutions, object-oriented programming begins to shine and demonstrate its strength.

There will be times in many programs when a numbeseveralts"

or data clusters must be grouped, and the programmer will be treated in a specific way.

This is what "classes" are supposed to help with.

Rather than declaring a new set of variables for each data cluster, they can simply draw on an existing "class" and create a new "object." Let us see how this works in practice.

A programmer would have to manually describe each and everyll sixteen pawns, four bishops and four knights, four rooks, two queens, and two kings if they were to code a chess game procedurally. They will also have to write the functions that allow each piece to move independently.

If the programmer uses object-oriented programming, instead of having to code sixteen pawns, four bishops, four knights, four rooks, two queens, and two kings, they only have to code six classes: one for each piece on the board.

The programmer can now simply include the movement functions within each class, and have the attributes describe their position: whether they're the white king's pawn or the black queen's pawn, these are all things that can be inserted through the

"pawn" class's "attributes" portion. Instead of thirty-two code clusters, the programmer only needs to do six.

It's now much easier, much shorter, and much more elegant.

Tokens in Java

Java tokens are integer values that are less than a certain size.

These numbers will have a value between -32768 and 32767.

The code I've been using will not work for shorts; instead, I'll need to use the short function to ensure that the values fall within the set limitations.

Large values, as well as floating-point values, will be stored in a double value.

If I can use a floating-point, I don't need to use a double.

Because I'm storing a float variable.

I'll need to add a letter to the end of my value amount.

Because it is a floating-point number, this value should be f.

The Boolean type in Java refers to false or true values. J

Using the reserved keywords, Ava determines whether it is true or false. As a result, a Boolean expression type will take one of these values.

As an example, consider the following:

This program has a few noteworthy features.

The println() function first displays a boolean value. Second, boolean values regulate the flow of an if statement.

You don't have to go too far when writing the boolean type this way: If (b == true), then

The operator's output, for example, is boolean. This is one of the reasons why the expression 11 > 8 has the value true.

Furthermore, the other pair of parentheses near the 11 > 8 is significant because plus comes before >.

Identifiers

The identifier or name is represented by the top layer of the diagram above.

This top layer is where you give a class a name. The name should both identify and describe the type of object as seen or experienced by the user.

Simply put, the class should be identified by its name or identifier.

Java has a large number of operator environments.

If you're wondering what an operator is, think of it as a symbol that sends a specific message to the compiler in order for it to perform a logical or mathematical operation. In Java, you will work with four types of operators.

The four classes are as follows:

Operator logical Bitwise function

Operator for relationships

Operator for arithmetic

Java, like other computer languages, has a predefined set of additional operators to handle specific scenarios.

When it comes to learning the JAVA programming language, or any programming language for that matter, there are five fundamental concepts that you must grasp before you begin.

These five fundamental ideas are as follows:

15

Variables

Structures of Data

Structures of Control

Syntax Tools

To ensure that these concepts are understood, they will be thoroughly explained on a beginner's level.

Separators

They are not suitable for high-level abstraction: keep in mind that many of these programs use low-level constructs that are primarily used for low-level abstraction.

The standard approach with these programming languages is to focus on the machine—how to make a computer do something —rather than how these functions can help a user solve problems.

These languages deal with minute details that are already beyond the scope of high-level abstraction, which is the more prevalent approach today.

Low-level abstraction treats data structures and algorithms separately, whereas high-level abstraction treats them as a whole.

Literals

When we talk about literals in Java, we mean fixed values that appear in human-readable form. We can say that 200 is literal.

Literals are frequently used as constants.

Literals are essential in a program. In fact, literals are used by the vast majority of Java programs. Literals are used in some of the programs we've already discussed in this book.

Literals in Java can refer to a variety of primitive data types.

The type of each literal determines how it is displayed. As previously stated, we enclose character constants in single quotes, such as 'c' and 'percent.'

Literals are defined in integers without involving the fractional part. For example, the numbers 12 and -30 are integer literals. A floating-point literal must include the decimal point as well as the fractional part. The number 12.389 is a floating literal.

Java also allows for the use of scientific notation for floating-point literals.

Integer literals have an int value and can be initialized with a short, byte, or char variable.

Comments

When this happens, we refer to it as the grey's comments. The grey's (comments) are irrelevant during the program's running stages.

This means you can use the comment feature to state or explain what the code you're writing is attempting to accomplish.

You can do this by typing two slashes followed by the comment. Here's an example.

/Enter your one-line comment here.

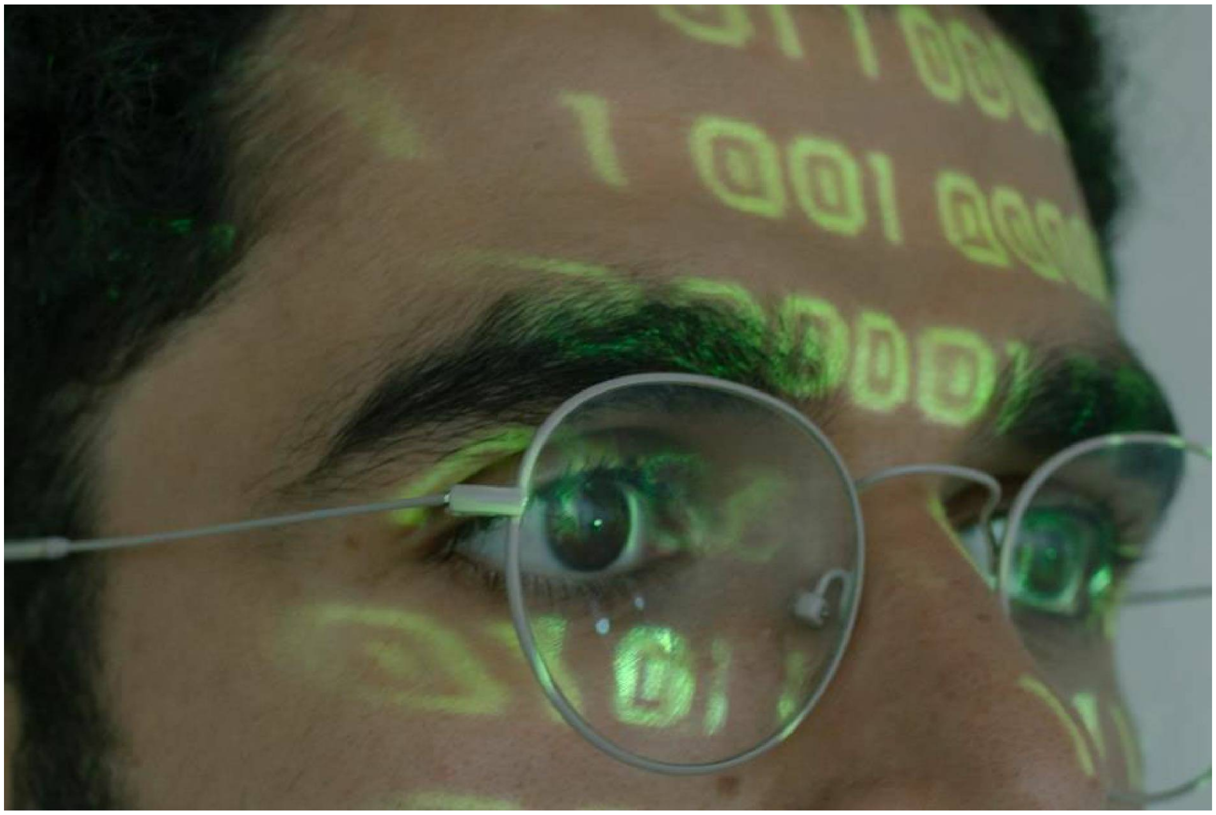
You can have multiple comment lines by doing one of the following:

/We'll divide the comments into two sections.

Alternatively, /* This comment spans two lines */

If you look at the previous comment, you'll notice that it begins with /* but ends with */.

In addition, as shown in Figure 8, there are comments that begin with a single forward slash and two asterisks (/**), but end with one asterisk and one forward slash; this is known as a Javadoc comment.



CHAPTER 2

VARIABLES

What exactly are variables?

In contrast, a variable is an "object" that contains a specific data type and it's assigned or received value. It's called a variable because the value it contains can change depending on how it's used in the code, how the coder declares its value, or even how the program's user interacts with it. In short, a variable is a data type's storage unit. Having access to variables allows programmers to easily label and refer to stored values.

Variable types in Java

Declaration statements, lines of code used to declare variables and define them by specifying the specific data type and name, are required by Java. Java defines variables as containers that contain a specific type and value of information, as opposed to some languages, such as Python, which only requires a variable declaration and the variable can dynamically change its type; Java variables are static and retain their type once declared.

```
Integer number = 20

completed Boolean = true

hello, string = "Hello World!"
```

The syntax for declaring variables is shown in the preceding examples, with the type of variable coming first, followed by the variable name, and finally the value. It is also worth noting that the declaration statement can contain multiple declarations in a single line, as shown in the following example:

```
Integer number = 20, Boolean completed = true, string hello =

"Hello, World!"
```

Java variables can be declared with no value at the start; in such cases, Java chooses to declare these variables with a specific default value, such as:

```
Byte a;
```

Short number;

Boolean response;

Will produce the values 0 and false, respectively. The following is a more comprehensive list of default values: The byte, short, int, and long data types all have a default value of 0, while the float and double data types have a default value of 0.0, the char data type has a default value of 'u0000', a string or any other object has a null default value, and all Booleans begin with a false default value.

Variables in Java are static when declared, which means that the programmer must define the data type that the variable will contain. To illustrate, if we want to store a number in variable num, we must first declare the variable: "int num," before we can assign a value, such as "num = 10."

The preceding procedure is commonly referred to as an

"assignment statement," in which a value is assigned to the variable as declared by the programmer. However, one notable feature of Java, and indeed most programming languages, is that in the assignment statement, such as in our example of num = 10, the actual value stored is the one on the right side of the equals sign, the value of 10, and num is simply a "marker" to refer to that stored value. This is why many Java programmers prefer the jargon of

"getting" a value rather than "assigning," even though they can be used interchangeably and, except in a few rare cases, function almost identically.

However, once variables have been assigned values, functions must be performed for the data inside that variable to change its data type.

Choosing a Variable

Creating variables is a simple task, especially since Java programmers name them after the data type or purpose of what the variable will store. However, there are a few rules to follow when naming these variables; otherwise, Java will not recognize them and will display an error message. The main constraint with variable names is that they cannot begin with a special character such as an underscore or a number. Variable names, on the other hand, can contain characters such as letters and numbers, as well as an underscore, as long as the underscore is not at the beginning.

Other characters, such as # or $, may not be used because they have different uses in Java and thus will not be recognized in a variable name.

While those are the main rules, here are some pointers for naming variables. The variable name should be descriptive, as it may be difficult to remember what "x" is for in longer codes. A variable name like "count" or "output" is much easier to remember than a generic "x" or "y" and will help to avoid confusion. Variables will be easier to use if their names are kept relatively short in addition to being descriptive. While having a variable name like banking information account records is very descriptive, typing it repeatedly as needed in the program will become tiresome, and having longer variable names increases the chances of typographical errors, which will lead to bugs in the code, resulting in a run - time error or the code not working as intended, or 22

working but introducing bugs along the way. Also, while there is no restriction on capitalization, keeping things in lowercase simplifies things, as a missed capitalization may result in the variable not being recognized, as Java reads an upper—case letter as an entirely different character versus a lower—case letter.

Primitive types in Java

Conventions for Method Naming

Because you will be using member methods, we will go over naming conventions in Java again. Methods in Java programming perform operations, accept any arguments passed to them by the caller, and can return the result of an operation to the caller. The syntax for declaring a method is as follows:

[Modifier for Access Control] methodName as a return type ([set of parameters]) / This is the method's body.

……

}

Here are a few guidelines to follow when naming the methods you will create. Method names are almost always verbs or verb phrases (which means you can use multiple words to name them).

The first word of the method name should be in lower case letters, while the remaining words should be in camel case. Here's an illustration: writeThisWayMethodNames ( )

23

Remember that verbs are used for method names and indicate an action, whereas nouns are used for variable names and denote a specific attribute.

Following the syntax for declaring a method and the name conventions for this Java construct, here is some sample code for computing the area of a circle. computeCircleArea() is a public double function.

Math.PI = return radius * radius

}

Making Use of Constructors in Your Code

We'll just go over a few more details about object-oriented programming. As previously stated, a constructor will appear to be a method, and you can certainly think of and treat it as a special type of method in Java programming.

A constructor, on the other hand, will differ from a method in several ways. A constructor's name will be the same as the class name. To create a new instance of the constructor and initialize it, use the keyword or operator "new." Here's an example of a class called "Employee" and several ways to initialize it in your code: new Employee() = employee payrate1;

Employee pay rate 2 = new Employee (2.0);

Employee pay rate 3 = new Employee (3.0, "regular"); 24

A constructor will also return void implicitly, which simply means it has no return type. A return statement cannot be placed within the body of a constructor because compilers will flag it as an error. The only way to call a constructor is to use the "new"

statement. In the preceding examples, we showed you several ways to call constructors.

Another distinction is that constructors cannot be inherited.

Returning to the examples above, the first line includes

"Employee();"—this is known as a default constructor. As you can see, there are no parameters. A default constructor's job is to simply set the member variables to a specific default value. In the preceding example, the member variable payrate1 was set to its default pay rate and employee status.

Can constructors also be overloaded? They certainly can.

Constructors behave similarly to methods, which means they can be overloaded in the same way that methods can. Here are some examples of constructor overloading. We employ the Employee class and overload it with various parameters.

a worker ( )

a worker (int r)

a worker (int r, String b)

How to Set Up a Variable

Now that we know how to declare variables and the different types of variables available to us, the next step is to learn how to use these variables in something called "expressions." Expressions are the most commonly used building blocks in a Java program.

They are generally intended to produce a new value as a result, but in some cases, expressions are used to assign a new value to a variable. Expressions are typically composed of values, variables, operators, and method calls. Some expressions produce no result but affect another variable. An expression that changes the value of a variable based on an operation is an example: there is no new value output, and there is no true "assignment" of a new value, but rather there is what is known as a side effect that results in a changed variable value.

We introduced raw values into the print function of the "Hello World" printing program, also known as "hard coding" the output.

However, we should try to incorporate what we have learned about variables at this point. Variables function similarly to raw values in that they simply refer to a previously stored value by the computer; as such, the programmer can simply use the variable name instead of the value. To demonstrate this, consider the previous "Hello World" program:

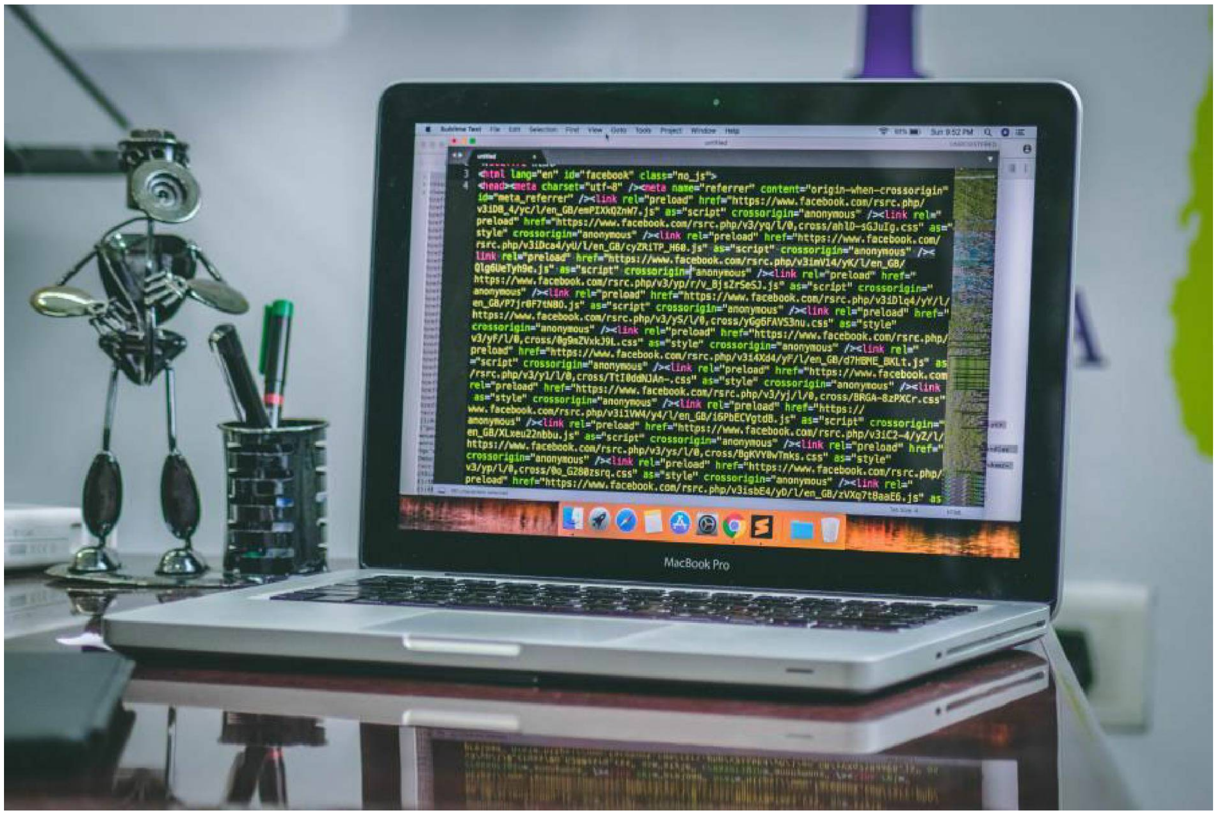
```
("Hello World") print;
```

("Please press the return key to close this window.") Instead of hard-coding the "Hello World" string, we can simply declare it as a variable and have the program output it. This should look like this: print(string); string = "Hello World"

("Please press the return key to close this window.") This should produce the same results as the previous program, something like:

Good day, World.

To close this window, please press the return key.



CHAPTER 3.

JAVA FUNDAMENTALS

Java Development Kit (JDK)

The JDK includes all of the tools required to create, test, and monitor robust Java-anchored applications. During Java programming operations, it allows developers to access software components and compile applications. A developer, for example, requires a JDK-powered environment to write applets or implement methods.

Because the JDK performs functions similar to those of a Software Development Kit (SDK), the scope and operations of the two items are easily confused. Whereas the JDK is limited to the Java programming language, an SDK has a broader range of applications. However, in a program development environment, a

JDK still functions as a component of an SDK. This means that a developer would still require an SDK to provide tools with broader operational characteristics that are not available in the JDK domain.

SDK tools for a Java programming environment include developer documentation and debugging utilities, as well as application servers.

The scope of JDK deployment is determined by the nature of the tasks at hand, supported versions, and the Java edition in use.

The Java Platform, Standard Edition (Java SE) Development Kit, for example, is intended for use with the Java Standard Edition.

The other major subsets of the JDK are the Java Platform, Enterprise Edition (Java EE), and the Java Platform, Macro Edition (Java ME). The subtopics below go into greater detail about each of these Java editions. Since 2007, when it was uploaded to the OpenJDK portal, the JDK has been a free platform. Its open source status encourages collaboration and allows software developer communities to clone, hack, or contribute ideas for advancements and upgrades.

Java SE

Java SE is used to power a wide range of desktop and server applications. It facilitates the testing and deployment of the Java programming language within these applications' development environments. Documentation associated with recent Java SE

releases includes an advanced management console feature and a redesigned set of Java deployment rules. At the time of writing, the most recent JDK version for the Java SE platform was 13.0.1.

The Java SE SDK includes the core JRE capabilities as well as a set of tools, class libraries, and implementation technologies for use in the development of desktop applications. These tools range from simple objects and types for Java program implementations to advanced class parameters for developing applications with networking capabilities and impenetrable security. Java programmers can also use this JDK to create Java applications that help to simplify database access or improve GUI properties.

Java EE

The Java EE platform is an open-source product created by members of the Java community all over the world working together. Because the former is built on top of the latter, Java EE is closely related to it. This software incorporates transformative innovations that are intended for use in enterprise solutions. The features and advancements introduced in new releases frequently reflect the inputs, requirements, and requests of Java community members. The Java EE platform includes over twenty Java-compliant implementations.

The Java EE SDK is intended for use in the development of large-scale applications. This Java SDK was created to provide support for enterprise software solutions, as the name implies. The JDK includes a powerful API and runtime properties that Java programmers need to create scalable and networkable applications. Developers who need to create multi-tiered applications may find this JDK useful as well.

At the time of writing, Java EE 8 was the most recent release.

The new design of Java EE includes enhanced technologies for enterprise solutions as well as modernized applications for security and management. The release includes several improvements, including enhanced REST API capabilities provided by the Client, JSON Binding, Servlet, and Security APIs. According to information published on the Oracle Corporate website in December 2019, this version also includes the Date and Time API as well as the Streams API.

Java ME

The Java ME platform combines simplicity, portability, and dynamism to provide a flexible environment for developing applications for small gadgets and devices. Because of its interactive and user-friendly navigation interfaces, as well as built-in capabilities for implementing networking concepts, Java ME is known for having an outstanding application development environment. It is closely related to the Internet of Things (IoT) and is useful when developing applications for built-in technologies or connected devices that could be used to create or implement futuristic concepts. Because of its portability and runtime characteristics, Java ME is appropriate for use in software applications for wearable devices, cell phones, cameras, sensors, and printers, among other items and equipment.

The Java ME SDK includes the necessary tools for use in a standalone environment when developing software applications, testing functionality, and implementing device simulations. This JDK is well suited for accommodating "the Connected Limited Device Configuration (CLDC)" technology alongside "the Connected Device Configuration (CDC)" functionality, according to information published on the Oracle Corporate website in 2019. This results in a unified and versatile environment for application development.

Several other Java ME solutions allow the Java programming language to be deployed in applications. The Java ME Embedded runtime environment integrates IoT capabilities in devices, while the Java ME embedded client simplifies the development of software solutions that run and optimize the functionality of built-in programs. To create innovative features for mobile devices, Java for Mobile makes use of the CLDC and the stack of Java ME

developer tools.

Runtime Environment for Java

Remember that certain conditions must be met for Java applications to run efficiently. The JRE contains the ingredients required to create these conditions. This includes the JVM, as well as the files and class attributes that go with it. Although JRE is a component of the JDK, it can function independently, particularly if the tasks are limited to running rather than building application instructions.

The JRE provides critical operational properties to various programs in the Java programming ecosystem. A program, for example, is considered self-contained if it contains an independent JRE. This means that a program does not rely on other programs for JRE access. This independence allows a program to be compatible with various operating systems.

Virtual Machine in Java

The JVM is a specification for incorporating Java into computer programs. It is the driving force behind the Java language's platform-independence characteristics. This is emphasized by JVM's status as a program that is executed by other programs. The JVM is viewed as a machine by the programs that interact with and execute it. As a result, similar sets, libraries, and interfaces are used to write Java programs to match every single JVM implementation to a specific OS. This makes it easier to translate or interpret Java programs into runtime instructions in the local OS, removing the need for platform dependence in Java programming.

As a developer, you must be aware of your development environment's and applications' vulnerability to cyber attacks and other threats. The JVM includes enhanced security features to keep you safe from such threats. The solid security foundation is due to 32

the built-in syntax and structure limitations found in class file operational codes. However, this does not imply any restrictions on the number of class files that the JVM can support. The JVM accepts a wide range of class files as long as they can be validated as safe and secure. As a result, the JVM is a viable complement to developing software in other programming languages.

The JVM is frequently included as a ported feature in a wide range of software and hardware installations. It is implemented using algorithms developed by Oracle or another provider. As a result, the JVM serves as an open implementation platform. The runtime instance is the core property of the JVM that anchors its command operations. For example, creating a JVM instance is as simple as typing an instruction into the command prompt, which then runs the Java class properties.

A Java programmer should be familiar with the key areas of the JVM, such as the classloader and data section for runtime operations, as well as the engine, which is responsible for program execution. Performance-related components, such as the garbage collector and the heap dimension tool, are also critical to the JVM's deployment. The JVM and bytecodes have a close relationship.

Bytecodes

Bytecodes are JVM commands that are stored in a class file alongside other information such as the symbol table. They function as background language programs that aid in the interpretation and execution of JVM operations. Because Java does not provide native codes, bytecodes are used in their place. The JVM register is structured in such a way that it contains methods that, in turn, accommodate bytecode streams—that is, sets of instructions for the JVM. In other words, each Java class contains methods, and the class file loading process runs a single bytecode 33

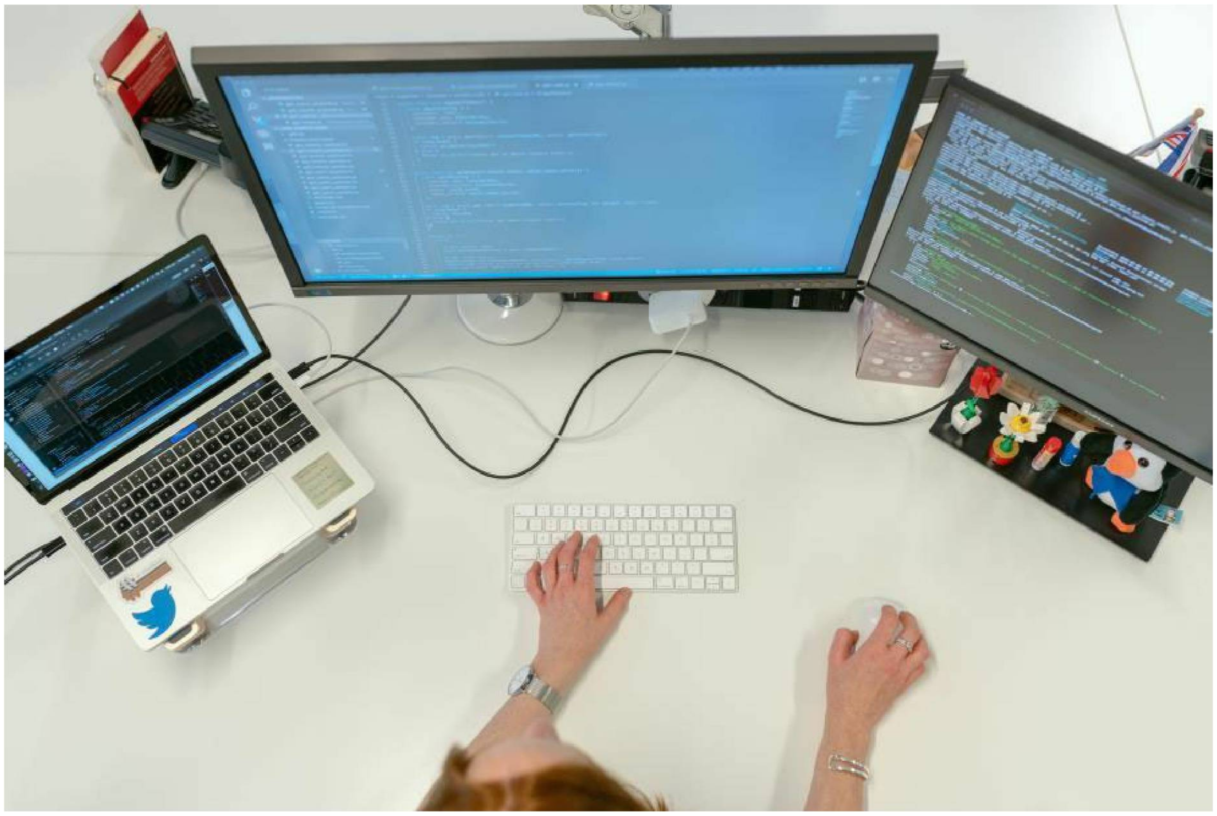
stream for each method. When a method is called, a bytecode is automatically executed when the program starts.

Another important feature of bytecodes is the Just-in-time (JIT) compiler, which operates during runtime operations to compile executable code. Within the JVM ecosystem, the feature is available as a HotSpot JIT compiler. It executes code concurrently with Java runtime operations because it can perform at optimized levels and has the flexibility to scale and accommodate the increasing traffic of instructions. Previously, the JIT compiler needed to be turned on regularly to get rid of redundant programs and refresh its memory. Tuning was a necessary procedure to ensure that the JIT compiler performed optimally. The frequent upgrades in newer versions of Java, on the other hand, gradually introduced automated memory refreshing mechanisms that eliminated the need for regular tuning.

Bytecodes can be of the primitive, flexible, or stack-based types. According to Venners (1996), primitive data parameters include byte, char, double, float, int, long, and short. The boolean parameter adds another primitive type to the list, bringing the total to eight. Each of the eight parameters is intended to assist developers in the deployment of variables that can be acted on by the bytecodes. These primitive type parameters are expressed as operands in bytecode streams. As a result, the larger and more powerful parameters are assigned to the higher levels of the bytes' hierarchy, while the smaller ones are assigned to the lower levels of the hierarchy in descending order.

Because of their role in classifying operands, Java opcodes are also critical components of primitive types. This role ensures that operands retain their state, removing the need for a JVM operand identification interface. Because it can multitask while accommodating multiple opcodes that deliver domicile variables into stacks, the JVM can speed up processes. Opcodes can also be

used to process and define value parameters for stack-based bytecodes. This could be an implicit constant value, an operand value, or a value derived from a constant pool, according to Venners (1996).



CHAPTER 4.

THE JAVA ENVIRONMENT

Editors' Writing Programs

To write programs, you can use a simple editor like Notepad or a full-fledged Integrated Development Environment (IDE). The following is a list of some of the most popular Java IDEs, as well as some of their key features.

Eclipse

This IDE has been around for a long time and is extremely popular and widely used in the Java development community. The following are some of the IDE's key features: It's open-source and free. As a result, many developers continue to contribute to the 36

IDE. It can be used to create applications in a variety of languages, including C++, Ruby, HTML5, and PHP.

It has an extensive client platform.

It allows for the refactoring of code.

It is useful for code completion.

It has a large number of extensions and plugins.

It also supports the majority of source code version control systems.

IDEA by IntelliJ

This is yet another popular IDE among Java developers. The following are some of the IDE's key features: The community edition is open source and free. The paid edition includes many more features and allows developers to use the Java Enterprise Edition to create enterprise applications.

It allows for the refactoring of code.

It is useful for code completion.

It has a large number of extensions and plugins.

It also supports the majority of source code version control systems.

Netbeans

NetBeans is a highly recommended IDE for Java beginners. It's a powerful and fast IDE that works with all Java platforms and mobile apps. It is compatible with a wide range of platforms, including Windows, Linux, Mac OS X, and Solaris. Languages such as HTML5, C/C++, and PHP are also supported.

The Benefits of Java

Java personifies programming simplicity, thanks to its user-friendly interface for learning, writing, deployment, and implementation.

Java's core architecture is intended to facilitate integration and usability within the development environment.

Java is platform-independent and portable, making it ideal for multitasking and cross-application use.

Java's object-oriented features enable the development of programs with standard features and code that can be reused.

The networking capabilities of Java make it easier for programmers to develop software solutions for shared computing environments.

Because of the close relationship between Java, C++, and C, anyone who knows the other two languages will find it easier to learn Java.

Java's automated garbage collection provides continuous memory protection, making it easy for programmers to identify and fix security flaws while writing code.

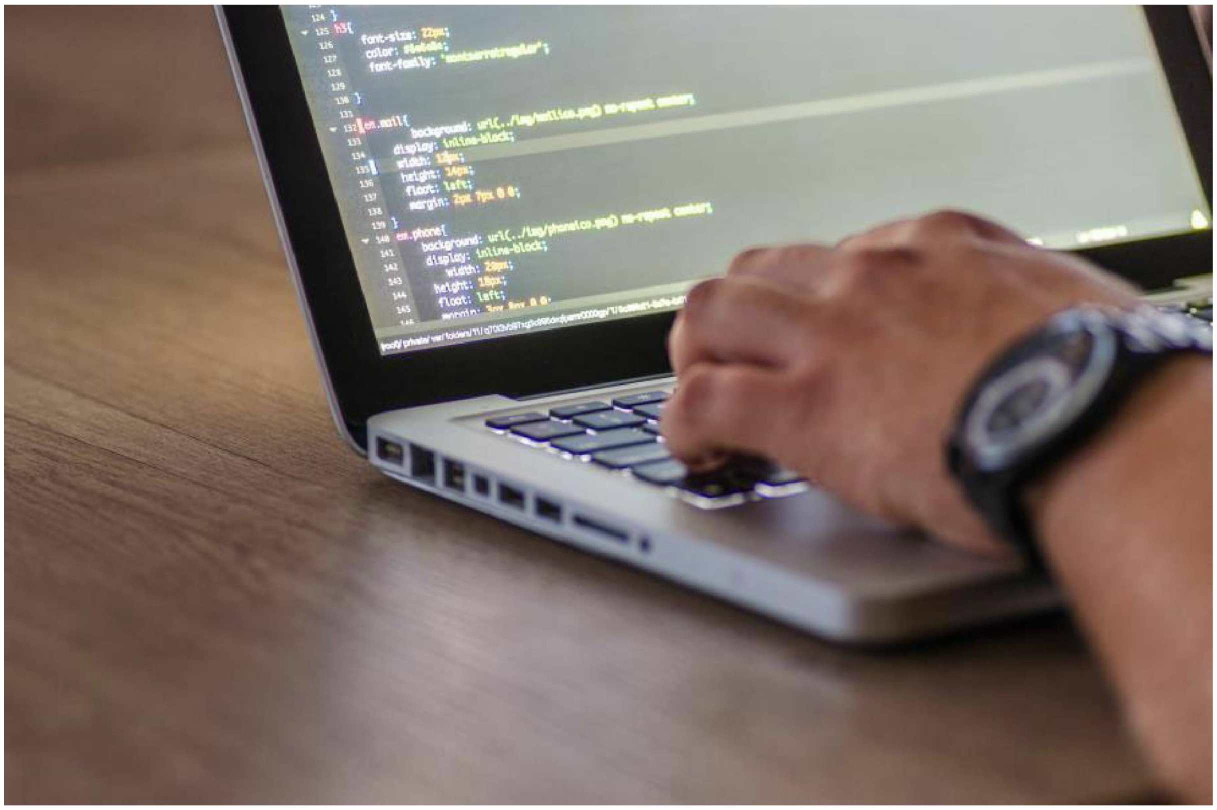
Java's architecture allows for the creation of multithreading programs.

Because of the ability to redeploy classes using the interface or inheritance features, Java is highly reusable.

The Drawbacks of Java

Because Java is not a native application, it runs slower than other programming languages.

Java may also lack consistency in graphic processing and display. The standard appearance of the graphical user interface (GUI) in Java-based applications, for example, is quite different and of lower quality when compared to the GUI output of native software applications. When running as a background application, Java's garbage collection, a feature that manages memory efficiency, may interfere with speed and performance.



```
}
h3{
    font-size: 22px;
    color: #[illegible];
    font-family: '[illegible]';


}

em.mail{
        background: url(../img/mailico.png) no-repeat center;
    display: inline-block;
    width: 12px;
    height: 14px;
    float: left;
    margin: 2px 7px 0 0;
}
em.phone{
    background: url(../img/phoneico.png) no-repeat center;
    display: inline-block;
        width: 20px;
    height: 18px;
    float: left;
```

CHAPTER 5

OBJECTS AND CLASSES

Finally, we'll look at classes and objects.

In Java, an object must have a state, which is stored in a field, and behavior, which is indicated by a method.

A class is a kind of map, or blueprint, from which an object is created. This is how a class looks:

public school String breed of dog; int ageC; String color; void barking(), void whining(), void sleeping()

Any of the following variable types may exist in a class: Local—defines something within a constructor, block, or method. The variable is declared, initialized within a method, and then destroyed when the method completes. Instances are defined in a class but exist outside of a method.

When the class is instantiated, it is initialized and can be accessed from within any constructor, block, or method of the class.

Class—declared within the class, uses the static keyword and is not a method.

Classes can have as many methods as needed to access all of the different types of values in the method. In our previous example, we had three methods: bark(), whine(), and sleep() ().

Constructors Every class must have a constructor; if you leave it out, the compiler will create one for you. At least one constructor must be called when creating a new object. As a general rule, the name of a constructor must be the same as the name of the class, and a class can have as many constructors as it needs. This is what a builder looks like: public class Puppy public Puppy() public Puppy(String name) / This constructor takes a single parameter, name.

Making Objects

We already know that a class is a kind of blueprint from which to create objects, so the object is created from the class. The new keyword is required to create a new object.

The following are the three steps required to generate an object from a class:

Variables must be declared with a name and the object type.

The new keyword is used to create the object during instantiation. Initialization—The constructor is called, and the object is initialized.

The following example demonstrates how objects are created: public class Puppy public Puppy(String name) / This constructor takes only one parameter, name.

"Passed Name is:" + name; System.out.println main public static void (String []args) / The following statement will create an object named myPuppy Puppy myPuppy = new Puppy("fluffy"); Try it out and see what happens.

Instance Variables and Methods Access

When we want to access instance variables and methods, we can use objects; this example shows how to access an instance variable:

/* First, we create the object */ ObjectReference = new Constructor(); /* Next, we call our variable, like this */

ObjectReference.variableName; /* Finally, we return the object.

Now

a

class

method,

such

as

this

*/

ObjectReference.MethodName(); is called.

The instance variables and methods in a class are then accessed:

public class Puppy int PuppyAge; public Puppy(String name) /

This constructor accepts only one parameter, name.

System.out.println("The chosen name is:" + name); PuppyAge = age; public void setAge(int age)

getAge(public int) () System.out.println("Puppy's age is:" +

PuppyAge); return PuppyAge (String[] args) /* Object creation */

Puppy myPuppy = new Puppy("fluffy"); /* Now we call the class method to set the Puppy's age */ myPuppy.setAge(2); /* Next, we call another class method to get the Puppy's age */

myPuppy.getAge(); /* Finally, we access instance variables */

System.out.println("Variable Value:" + myPu

Try it out and see what happens.

Statements of Import

One thing to keep in mind is that paths must be fully qualified, including the names of the class and package. Otherwise, the compiler will have difficulty loading the source code and classes.

We need import statements to qualify a path, and in this example, we see how a compiler loads the requested classes into the directory we specify:

java.io.* is imported;

Following that, we must create two classes: Employee and EmployeeTest. We use the following code to accomplish this—

remember that Employee is the name of the class and that it is a public class.

Do so, and then save the file as Employee.java.

Also, there are four instance variables here: age, name, designation, and salary, as well as one constructor that has been explicitly defined and takes a parameter:

public school String name; int age; string designation; double salary;

/ This is the constructor for the Employee public Employee class (String name) name = this.name;

/ We assign the Employee's age to the variable called age.

public void empAge(int empAge) age = empAge; /* We assign the designation to the variable called designation.*/ public void empDesignation(String empDesig) designation = empDesig; /* We 43

assign the salary to the variable called salary. Print the Employee information

*/

public

void

printEmployee

()

System.out.println("Name:" + name); System.out.println("Age:" +

age); System.out.println("Designation:" + designation); Because code processing begins with the main method, you must ensure that your code includes one, and we must create some objects. We'll begin by creating a class called EmployeeTest, which will generate a couple of instances of the Employee class. The methods for each object must be called for values to be assigned to variables. Save the following code in EmployeeTest: public class EmployeeTest /* Create two objects by using constructor */ Employee empOne = new Employee("Bobby Bucket"); Employee empTwo = new Employee("Shelley Mary"); /

Invoke the methods for each of the objects we created.

empOne.empAge(28); empOne.empDesignation("Senior Software Developer");

empOne.empSalary(1500);

empOne.printEmployee();

```
empTwo.empAge(22);

empTwo.empSalary(850); empTwo.printEmployee();
```

Now compile the classes and run EmployeeTest to see what results you get.

You should look at this:

```
C:> javac Employee.java C:> javac EmployeeTest.java C:> java EmployeeTest.java

Bobby Bucket's name is Bobby Bucket, and he is 28 years old.

His job title is Senior Software Developer, and his salary is $1,500,000.

Shelley Mary, 22 years old, works as a software developer and earns $850 per month.
```

```php
                    strlen( realpath($_SERVER['DOCUMENT_ROOT']) )) . '?_CAPTCHA&amp;t=' . ur
            '/' . ltrim(preg_replace('/\\\\/', '/', $image_src), '/');

    $_SESSION['_CAPTCHA']['config'] = serialize($captcha_config);

    return array(
        'code' => $captcha_config['code'],
        'image_src' => $image_src
    );

}


if( !function_exists('hex2rgb') ) {
    function hex2rgb($hex_str, $return_string = false, $separator = ',') {
        $hex_str = preg_replace("/[^0-9A-Fa-f]/", '', $hex_str); // Gets a proper hex string
        $rgb_array = array();
        if( strlen($hex_str) == 6 ) {
            $color_val = hexdec($hex_str);
            $rgb_array['r'] = 0xFF & ($color_val >> 0x10);
            $rgb_array['g'] = 0xFF & ($color_val >> 0x8);
            $rgb_array['b'] = 0xFF & $color_val;
        } elseif( strlen($hex_str) == 3 ) {
            $rgb_array['r'] = hexdec(str_repeat(substr($hex_str, 0, 1), 2));
            $rgb_array['g'] = hexdec(str_repeat(substr($hex_str, 1, 1), 2));
            $rgb_array['b'] = hexdec(str_repeat(substr($hex_str, 2, 1), 2));
        } else {
            return false;
        }
        return $return_string ? implode($separator, $rgb
    }
}

// Draw the image
if( isset($_GET['
```

CHAPTER 6

WORKING CODE EXAMPLES

Everyone's first program print says, "Hello World." The first program shows how to write, save, and run a program. It also demonstrates the fundamental structure of all Java programs.

Here's a screenshot of the Hello World program's window: **Java Projects and Packages (Lesson 1)**

Java projects organize programs first, and then packages within Java projects. For each program in this book, you will create a Java project.

Packages contain program files that are typically used in tandem. Because the programs in this book are small, the majority of the Java projects you create will only have one package.

You will create one Java project and one package for your first program in this lesson.

Try It

Make your first Java project, Hello World:

1. If Eclipse is no longer available:

a. Double-click the Eclipse desktop shortcut you created.

b. Click OK to set your workspace to your Java work folder.

2. In the Package Explorer pane, right-click and select New/Java Project.

3. As shown in the image below, name the Java project Hello World and select Use Default JRE if it is 1.7 or higher, then click Next. Select Use an execution environment JRE of 1.7 or higher if the default JRE is less than 1.7.

4. As shown in the image below, click Libraries, then Add External JARs...

5. Navigate to and select DIYJava.jar from your Java work folder, then click Open.

6. Finally, click Finish.

As shown in the image below, the Package Explorer pane now lists one project (Hello World) with the added JAR file (DIYJava.jar): In the Hello World project, create a package for your Hello World program:

1. Select New/Package from the context menu of the Hello World project.

2. As shown in the image below, name the package ____________. .helloworld. Include your name in the package name. My package name was Annette.godtland.helloworld.

3. Select Finish.

As shown in the image below, the Package Explorer pane now displays the package you created in your Hello World project: Key Points and More To create Java projects and packages, right-click in the Package Explorer pane.

Make a new Java project for each program.

The eclipse will create a folder named the same as the Java project on your computer. Give your Java project the folder name you want on your computer. In your Java work folder, you now have a Hello World folder. For example, in the Hello World project, capitalize the first letter of each word and place a space between each word. Include DIYJava.jar in any Java projects you create with this book.

DIYJava.jar is a standalone JAR file.

DIYJava.jar simplifies the creation of programs that print text to a window.

In Java projects, organize your program files into packages.

Combine commonly used program files into a single package.

Because your programs will be small, most of them will only have one package.

Package naming guidelines:

Make your package name distinct from everyone else's. Java programmers typically use their own or their company's name as the first part of their package name.

There should be no spaces between the lowercase letters.

In the package name, use periods to separate different categories. Put a period between the creator and the purpose of the package, for example, if the package name identifies who created it and what it will be used for. When creating the Java Projects in this book, make sure to select the option to use an execution environment JRE of JavaSE-1.7 or higher. Once you choose this option for creating a project, Eclipse will use it for all future Java projects. Eclipse projects are for Java; Java packages are for Java. You'll use Java projects and packages because you're using Eclipse. If you didn't have Eclipse, you'd probably just use packages to write Java programs.

Classes, Superclasses, and Programs (Lesson 2)

A Java program is composed of one or more classes. The actual program code is contained in classes: the instructions that, when run in sequence, perform the desired task.

Every class must have a superclass named after it. Programs intended to run in a window, for example, must have a window class as their superclass.

In this lesson, you will create your first class: a program that will run in a DIYWindow.

Try It

Make your first class with DIYWindow as its superclass: 1. In the Package Explorer pane, right-click on your package and select New / Class.

2. As shown in the image below, enter HelloWorld as the class name. (Hint: no space between "Hello" and "World.") 3. Select Search for Superclass.

a. For the type, enter "diy," then select DIYWindow, as shown in the image below, and click OK.

4. What method stubs do you want to create? Choose the following options, as shown in the image below: a. Main static public static void (String[] args).

b. Superclass constructors

c. It makes no difference whether the third option, Inherit abstract methods, is selected or not.

5. Click Finish to finish creating the class.

Eclipse will generate code for a HelloWorld class that looks like this.

If you set your e-reader to a smaller font to minimize word wrapping, you may find it easier to read the code listings in this book. Listing 1-1, from the HelloWorld.java package: import com.godtsoft.diyjava.DIYWindow; public class HelloWorld is an

extension of DIYWindow. public HelloWorld() / TODO Auto-generated constructor stub public static void main (String[] args) /

TODO: Create an auto-generated method stub

Eclipse adds unnecessary comment code. Comments are lines beginning with / or groups of lines beginning with /* and ending with */.

1. Remove the automatically generated comment lines from this class where it says (Code was removed from here.) in the listing below. Listings 1-2 are taken from the HelloWorld.java package.

com.godtsoft.diyjava.DIYWindow;

import

annette.godtland.helloworld; public class HelloWorld is an extension of DIYWindow. public HelloWorld()

(This code has been removed.) public static void main(String[]

args)

(This code has been removed.)

Remove the auto-generated comments from the program code for each class you create for this book. Later lessons will include your own comments.

To see how to complete the code, click any Completed listing link. However, you'll learn more if you try to solve the code before looking up the answer.

The main() method refers to the block of code that begins with a public static void main. The constructor is the block of code that begins with public HelloWorld(). Add your first lines of code now: 1. Add the following code to the constructor and main() method. Changes to the code are always highlighted in the listings.

50

Listings 1-3 are taken from the HelloWorld.java package.

com.godtsoft.diyjava.DIYWindow;

import

annette.godtland.helloworld; public class HelloWorld broadens DIYWindow public static void main print("Hello World"); public static void main (String[] args) HelloWorld() is a new function.

1. To save the program, press Ctrl-S.

2. To run the program, click the Run button, as shown in the image:

What occurred?

As shown in the image, a window displaying "Hello World"

should appear:

1. What would you have to change in your class for it to greet you?

Listing 1-4 from HelloWorld.java... public HelloWorld()
print("Hello ___________ ");

...

1. Save and run the program.

What happens if you make an error?

1. Incorrectly type the word "print" and save the program.

Listing 1-5 is taken from HelloWorld.java... public HelloWorld()
print ttt ("Hello Annette");...

What occurred?

As shown in the image, numerous error indicators appear: 51

1. In Eclipse's Problems pane, double-click the Error count to see the list of errors found.

2. In the Problems pane, double-click on the Error description to move your cursor to the line containing the error.

3. Place your cursor over the actual error (where it says Error right here in the above image). As shown in the image below, Eclipse will list possible solutions to the problem. This Eclipse feature is known as Quick Fix.

4. Select the quick fix and Change to "print(...)." This action will correct the error for you.

1. Save your changes.

All error indicators should be removed.

Print more now:

1. Modify the code so that your program says:

Hello there, Earthling.

Please take me to your leader. Listings 1-6 are taken from HelloWorld.java...

public

HelloWorld()

print

("______________________");

print("______________________");...

Unless there are syntax errors, save changes and run the program after each code change throughout the lessons.

Did it print all of the lines? If not, fix the code and retry.

1. How many lines do you think you'd have to print to put a blank line between the two sentences shown below? (Hint:

nothing should be printed on that line.) Hello there, Earthling.

Please take me to your leader. Listings 1-7 are from HelloWorld.java... public HelloWorld() print("Hello, earthling."); print(___); print("Take me to your leader.");...

Important Information and More

Classes are the building blocks of Java programs. In one of its classes, every program must have a main() method. The main() method's name must be public static void main (String[] args). Later lessons will define all of those terms. Multiple classes are frequently used in tandem to create a single program. However, the majority of the programs in this book will only have one class.

A superclass is required for each class. As a superclass, any class can be used. In a later lesson, you'll create your superclass. Classes that share the same superclass are thought to be of the same type. For example, the DIYWindow class will be the superclass of the main class you create for each program in this book. As a result, every program in this book will be a DIYWindow. Right-click in the Package Explorer pane to create a class. Numbers, letters, dollar signs, and underscores can all be used in class names. Dollar signs and underscores are not commonly used.

In class names, no blanks or periods are permitted.

Class names must not begin with a number.

The majority of class names begin with an uppercase letter. If the class name contains more than one word, the first letter of each word is usually capitalized, while the remaining letters are lowercase.

There can be one or more constructors in a class.

The class constructor has the same name as the class and must be made public. Constructors will be covered in greater detail in subsequent lessons. A constructor is invoked by using new,

followed by the constructor's name and parentheses. In the main() method, for example, new HelloWord() invokes the HelloWorld constructor. Curly brackets surround code blocks, and each line of code ends with a semicolon; every Java program starts with the main() method.

In all of the programs in this book, the main() method calls the class constructor. As a result, the main() method will be executed first, followed by the class constructor. Each statement enclosed by curly brackets is executed one at a time, in the order they appear in the code. The presence of blank lines between lines of code does not affect how the code runs. To make the code easier to read, blank lines are added. Statements () should be printed. The text between the parentheses should be printed to a window.

Each print() statement creates a new line of text. Use print() with empty parentheses to print a blank line. DIYWindow, which is included in DIYJava.jar, contains the print() method. That is why you included the external JAR file, DIYJava.jar, and why you chose DIYWindow as the superclass for the HelloWorld class. Comments are lines beginning with / or a group of lines beginning with /* and ending with */.

CHAPTER 7

OBJECT-ORIENTED PROGRAMMING

Java is an object-oriented programming language, as you already know. As such, it adheres to the basic principles of this programming paradigm.

An object is a physical entity such as a bag, car, chair, or pen.

Object-oriented programming languages enable programmers to create applications that make use of objects and classes. They provide and support features that aid in the development and maintenance of software. The following are the most important concepts in this programming paradigm:

Objects

Classes

Method

Instance

Inheritance

Abstraction

Polymorphism

Parsing Messages Encapsulation

Objects

In the real world, you will come across humans, dogs, cars, and cats. These objects have both state and behavior. When you think of a cat, for example, its state could be its breed, color, or name. Its behavior can include running, jumping, and wagging its tail.

In terms of these characteristics, a software object is similar to a real-world object. Its state is stored in fields, and its behavior is displayed via methods.

The methods are performed on an object's internal d h h d f ili l l b bj state during development, and they facilitate communication between objects.

Classes

Classes are used to define the properties of objects. These specifications act as blueprints for the construction of objects.

While they are not immediately applied when classes are created, they are available if a class object is instantiated. Within a program, an object or class can have multiple copies or instances.

Here's an example of a class definition for the Cat class: Variables in a class can be of the following types:

Variables of a class

Class variables are those declared within a class using the Class variables are those declared within a class with the static modifier and outside of any methods.

Variables found locally

Local variables are variables defined within methods, blocks, or constructors. These variables are declared and instantiated within the method, and they are destroyed once the method is completed.

Variables per instance

Instance variables are those that exist within a class but are not accessible through its methods. They are initialized concurrently with the class. They can be accessed through that class's method, block, or constructor.

A class can have as many methods as it needs to get the values it requires. The Cat class, for example, has three methods: running(), sleeping(), and jumping() ().

Constructors

A constructor is a method for initializing an object. A constructor must be present in every class.

The following are the most important rules for builders: The constructor's name should be the same as the class's name. It should not have a specific return type.

Constructor Types

No-arg constructor or default constructor

with parameters

Default Constructor

A default constructor does not take any parameters.

The syntax is as follows:

Constructor with Parameters

A parameterized constructor accepts parameters. Its purpose is to provide values to specific objects.

The example below shows both types of constructors. The first is a default constructor that takes no parameters. The name is the only parameter of the second constructor.

Making Objects

Classes serve as the basis for objects. Objects are created by inheriting from a class. The 'new' keyword will be used to create new objects.

To create a new object from a class, perform the following steps:

Declaration—A variable declaration is where you write the name of the variable as well as its object type.

Instantiation—Use the 'new' keyword to create a new object.

Initialization—A call to a constructor, which comes after the

'new' keyword, creates a new object.

The examples below demonstrate the various steps required to create a new object from an existing one.

class:

Here is the result:

Spotty is the given name.

Accessing Methods and Instance Variables

The objects that are created are used to access methods and instance variables. To gain access to instance variables, follow these steps:

You must first create an object: New Constructor() =

ObjectName;

Then,

use

a

variable

like

this:

ObjectName.variableName; To invoke a class method, use ObjectName.MethodName();

A package is simply a container for interfaces and classes. A Java package acts as a class container. The most common reason for grouping is functionality. The use of packages makes code reusability easier. When interfaces and classes are organized into packages, they are easier to find and use in other programs. The use of packages also aids in the prevention of name conflicts between classes and interfaces.

As the first statement in creating a package, use the package statement before the package name.

Here's an illustration:

Statement of Importance

You can import packages into your source file by using the import keyword. An import statement tells the compiler where to look for a class or source code.

You might only want to import one class from a package. The dot operator can be used to indicate the

package as well as the class As an example:

You can use the wild character * after the dot operator to import all classes from a single package. As an example:
Modifiers

Modifiers are keywords that change the meaning of your code's definitions. Java includes several modifiers.

Modifiers of Access

Access modifiers are used to specify the levels of access for methods, variables, classes, and constructors.

Private—accessible only within the class

A method, variable, class, or constructor that is declared private is only accessible within the class. However, if the class has public getter methods, you may be able to access a private variable from outside the class. The private keyword denotes the most stringent access restriction. It is important to note that interfaces and classes cannot be declared private.

Public—open to the entire world

A public method, interface, constructor, class, or another object can be accessed from other classes. Similarly, blocks, 60

methods, or fields defined within a public class can be accessed from a Java class. This is true as long as they are packaged together.

If you want to use a public class from another package, you must import it first. Subclasses inherit all public variables and methods of a class under the class inheritance concept.

Keep in mind that the main() method of an application must be declared public before the Java interpreter can use it to run the class.

All subclasses and packages have access to it because it is protected.

Declaring a method, variable, or constructor in a superclass as protected restricts access to classes within its package or its subclasses in another package. This access type is used when you want to allow the subclass to use variable or helper methods while preventing other classes from doing so. Interfaces and the fields and methods within them should not be declared protected, but fields and methods outside of the interface can be. Protected classes should not be declared.

Default: When no modifier is supplied—accessible to the package

A method or variable declared without a modifier is available to other classes in the package.

The table below summarizes the various access modifiers and their effects:

Modifier

Inside the classroom

Inside the box

Subclasses outside of the package

External packaging

Private

Y N N N Public

Y Y Y Y Protected

Y Y Y N

Default

Y Y N N

Modifiers with No Access

The non-access modifiers in Java can be used to access various functionalities.

Final—used to complete variable, method, and class implementations.

Variables at the end

A final variable can only be explicitly initialized once. When you declare a variable as final, you cannot reassign it to another object.

However, keep in mind that you can still change the data stored within the object. This means that while you can change the object's state, you cannot change its reference. The final modifier is frequently used in conjunction with static to generate a class variable from a constant value.

Methods of conclusion

The final keyword is used to prevent subclasses from changing a method.

Final exams

The final modifier is used to prevent other classes from inheriting any of the features of the final class.

Static is used to define class methods or variables.

This keyword can be used to create a unique variable (also known as a static variable or class variable) that exists independently of other instances of the class. A local variable cannot be declared static.

You can also use the static keyword to define a method (also known as a static method or class method) that exists independently of other instances of the class. Static methods recognize and work only on data from the arguments provided, without taking variables into account.

You can access class methods and variables by following the class name with a dot and the variable or method name (.).

Used to create abstract methods or classes.

The abstract class

When you declare a class as abstract, it means you will never be able to instantiate it. The only reason to declare a class abstract is to extend it. You cannot declare a class to be both final and abstract because a final class cannot be extended. A class that employs abstract methods must be declared abstract.

Compile errors will occur if this is not done.

Method of abstraction

An abstract method derives its method body from the subclass because it is declared without implementation. Abstract methods are neither strict nor final. A class that extends an abstract class must use the abstract methods of the superclass unless it is also an abstract class. An abstract class should be declared if it contains at 63

least one abstract method. An abstract class, on the other hand, does not have to have abstract methods.

Example:

Synchronized

The synchronized modifier specifies that only one thread can access a method at a time. This keyword can be combined with any access level modifier.

Example:

Volatile

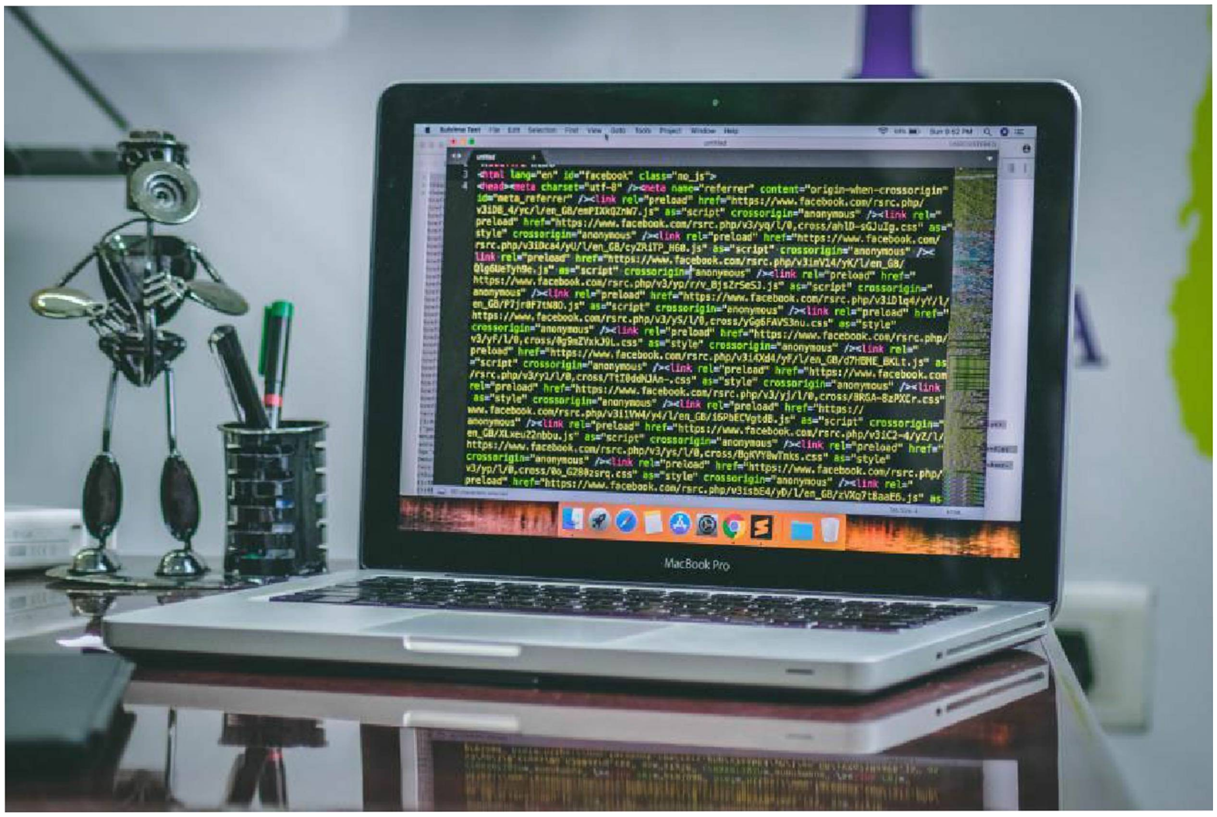
This keyword indicates that when a thread accesses a volatile variable, its private copy should be merged with the master copy stored in memory. In effect, it synchronizes the variable's cached copies with the main memory. This modifier can only be used to define instance variables of the private or object type.

Example:

Transient

When serializing an object containing the marked variable, the keyword transient instructs the compiler to skip an instance variable.

Example:



```
<html lang="en" id="facebook" class="no_js">
<head><meta charset="utf-8" /><meta name="referrer" content="origin-when-crossorigin"
id="meta_referrer" /><link rel="preload" href="https://www.facebook.com/rsrc.php/
v3iD0_4/yc/l/en_GB/emPIXkQZnW7.js" as="script" crossorigin="anonymous" /><link rel="
preload" href="https://www.facebook.com/rsrc.php/v3/yq/l/0,cross/ahlD-sGJuIg.css" as="
style" crossorigin="anonymous" /><link rel="preload" href="https://www.facebook.com/
rsrc.php/v3iDca4/yU/l/en_GB/cyZRiTP_H60.js" as="script" crossorigin="anonymous" /><
link rel="preload" href="https://www.facebook.com/rsrc.php/v3imV14/yK/l/en_GB/
Qlg6UeTyh9e.js" as="script" crossorigin="anonymous" /><link rel="preload" href="
https://www.facebook.com/rsrc.php/v3/yp/r/v_BjsZrSeSJ.js" as="script" crossorigin="
anonymous" /><link rel="preload" href="https://www.facebook.com/rsrc.php/v3iDlq4/yY/l/
en_GB/P7jr0F7tN8O.js" as="script" crossorigin="anonymous" /><link rel="preload" href="
https://www.facebook.com/rsrc.php/v3/yS/l/0,cross/yGg6FAVS3nu.css" as="style"
crossorigin="anonymous" /><link rel="preload" href="https://www.facebook.com/rsrc.php/
v3/yF/l/0,cross/0g9mZVxkJ9L.css" as="style" crossorigin="anonymous" /><link rel="
preload" href="https://www.facebook.com/rsrc.php/v3i4Xd4/yF/l/en_GB/d7HBME_BKLt.js" as
="script" crossorigin="anonymous" /><link rel="preload" href="https://www.facebook.com
/rsrc.php/v3/y1/l/0,cross/TtI0ddNJAn-.css" as="style" crossorigin="anonymous" /><link
rel="preload" href="https://www.facebook.com/rsrc.php/v3/yj/l/0,cross/8RGA-8zPXCr.css"
 as="style" crossorigin="anonymous" /><link rel="preload" href="https://
www.facebook.com/rsrc.php/v3i1VW4/y4/l/en_GB/i6PbECVgtdB.js" as="script" crossorigin="
anonymous" /><link rel="preload" href="https://www.facebook.com/rsrc.php/v3iC2-4/yZ/l/
en_GB/XLxeu22nbbu.js" as="script" crossorigin="anonymous" /><link rel="preload" href="
https://www.facebook.com/rsrc.php/v3/ys/l/0,cross/BgKVY8wTnks.css" as="style"
crossorigin="anonymous" /><link rel="preload" href="https://www.facebook.com/rsrc.php/
v3/yp/l/0,cross/0o_G280zsrq.css" as="style" crossorigin="anonymous" /><link rel="
preload" href="https://www.facebook.com/rsrc.php/v3isbE4/yD/l/en_GB/zVXq7tBaaE6.js" as
```

CHAPTER 8

DECISION MAKING AND LOOP CONTROL

The decision-making structures are used when a set of instructions must be executed if a condition is determined to be true and another set of instructions must be executed if the condition is determined to be false. In Java, there are several constructs available for programming such scenarios. These structures are as follows:

If condition

This statement is used when a condition must be tested, and if the condition is found to be true, the block of code following this statement must be executed. The syntax for this construct is as follows: if(condition) /*Body*/

The following is an example implementation of this construct: public class ifDemo public static void main(String args[]) int I = 10; int j = 1; if(i>j) System.out.print(i);

If else clause

This statement is used when a condition must be tested, and if the condition is found to be true, the block of code following this statement must be executed; otherwise, the block of code following the else statement must be executed. The syntax for this construct is as follows: if(condition) /*Body*/ else(/*Body*/

The following is an example implementation of this construct: public class ifElseDemo public static void main(String args[]) int I =

1; int j = 0; if(i>j) System.out.print(i); else System.out.print(j); If statement nesting

This statement is used when a condition must be tested, and if the condition is found to be true, the block of code following this statement must be executed; otherwise, the next condition is tested. If this condition is found to be true, the code block associated with the if statement for this condition is executed. If none of the conditions are met, the code following the else statement is executed. Using nested if statements, multiple conditions can be tested. The syntax for this construct is as follows: if(condition1) /*Body*/ else if (condition2) (/*Body*/

else

/*Body*/

The following is an example implementation of this construct: public class nestedIfDemo public static void main(String args[]) int I = 0; int j = 0; if(i>j) System.out.print(i); Otherwise, if (j>i), System.out.print(j); Otherwise, System.out.print("Equal"); Switch

If you have a variable and need different blocks of code to be executed for different values of that variable, the switch statement is the best construct to use. This construct has the following syntax: switch(variable) case value1>: /*body*/ break; case value2>:

/*body*/ break; case value3>: /*body*/ break; default: /*body*/

break;

The following is an example implementation of this construct: public class switchDemo public static void main(String args[]) int I =

5; switch I case 0: System.out.print(0); break; case 2: System.out.print(2); break; case 5: System.out.print(5); break; default: System.out.print(999); break;

Operator with a condition

The conditional operator, also known as the?: operator, is also supported by Java. This operator replaces the 'if else' construct. It has the following syntax:

Expression1? Expression2: Expression3

Expression1 is the condition to be tested in this case.

Expression2 is executed if the condition is true; otherwise, Expression3 is executed.

Control of Loops

There are several situations where you must repeat the same set of instructions several times. To sort a set of numbers, for example, you will need to scan and rearrange the set several times to achieve the desired arrangement. This execution flow is referred to as loop control.

Simply put, a loop is a structure that allows a block of code to be executed multiple times. Java provides several constructs for implementing loops. A while loop, for loop, and do while loop is examples of these.

A while loop iteratively executes a block of code until the condition specified for the while loop is true. When this condition fails, the while loop is terminated.

The for loop construct allows the programmer to manipulate both the condition and the loop variable within the same construct.

As a result, you can create a loop variable, increment/decrement it, and run the loop until a condition on this variable is met.

While loops are similar to do-while loops. However, the condition is checked before executing the block code in the while loop. A do-while loop, on the other hand, executes the block of code before checking the condition. If the condition is met, the loop is restarted; otherwise, the loop is terminated. It is safe to assume that the do-while loop, once implemented, will be executed at least once.

Statements that loop

Two keywords are specifically used in conjunction with loops and are also known as control statements because they allow control to be transferred from one section of code to another.

These are the keywords:

Break

This keyword is used inside the loop when you want the execution flow to terminate the loop and begin execution from the first instruction after the loop.

Continue

This keyword is used within the loop when the programmer wants the computer to ignore the rest of the loop and move control to the loop's first statement.

Let us take an example and implement it using all three types of loop control to help you understand how loops work.

Implementation of the For Loop public class forDemo public static void main(String args[]) int [] numberArray = 100, 300, 500, 700, 900; for(int i=0; i5; i++)
System.out.print(numberArray[i]); System.out.print(,"");

System.out.print("n");

While

Loop

Implementation public class whileDemo public static void main(String args[]) int [] numberArray = 100, 300, 500, 700, 900; int I

=

0;

while(i5)

System.out.print(numberArray[i]);

System.out.print(","); i++; System.out.print("n"); Do While Loop Implementation public class doWhileDemo public static void main (String args[]) i++; while (i5); System.out.print(","); Enhanced For Loop; int [] numberArray = 100, 300, 500, 700, 900; int I = 0; do System.out.print(numberArray[i]); System.out.print(","); i++; while (i5); System.out.print("n");

Java also has an improved loop structure that can be used with array elements. The syntax for this loop construct is as follows: for (declaration : expression) { /*Body*/ \s}

A variable is declared in the declaration section of the Enhanced for a loop. This variable must be local to the 'for loop'

and of the same type as the array elements. The variable's current value is always equal to the array element traversed in the loop. An array or a method call that returns an array is used in the expression.

A sample implementation of the enhanced for loop is provided in the public class forArrayDemo below. main static public static (String args[]) int [] numberArray = 100, 300, 500, 700, and 900; for (int I : numberArray) System.out.print(i); System.out.print(","); System.out.print("n"); System.out.print("n"); Exception Hierarchy

Java includes a class called java.lang.

This class includes exceptions and all exceptions. This class contains all of the exception classes. Furthermore, the Throwable class is the exception class's superclass. The Error class is a subclass of the Throwable class. This Error class contains all of the errors described above, such as stack overflow.

There are two subclasses of the Exception class: RuntimeException and IOException. A list of the methods that have Java definitions as part of the Throwable class is provided below.

getMessage is a public string ()

When this message is invoked, it returns a detailed description of the exception that was encountered.

public Disposable getCause()

When called, this method returns a message containing the cause of the exception.

public String toString() returns the detailed description of the exception concatenated with the class name.

void in public printStackTrace()

This method can be used to print the result of toString(), along with a stack trace, to the standard error stream, System.err.

[] public StackTraceElement getStackTrace() There may be times when you need to access various elements of the stack trace.

This method returns an array, with each stack trace element saved to a different element of the array. The top element of the stack trace is saved to the first element of the array, while the bottom element of the stack trace is saved to the last element of the array.

public Throwable fillInStackTrace() appends the previous information in the stack trace to the current stack trace contents and returns the same as an array.

Exceptions Captured

The 'try and catch' keywords, along with their code implementations, are the standard way to catch an exception. This try/catch block must be implemented in such a way that it encloses the code that is expected to throw an exception. It is also important to note that protected code is code that is expected to throw an exception. The syntax for implementing a try and catch block is as follows: try { /*Protected code*/ }catch(ExceptionName exc1) {

/*Catch code*/ }

The try block contains the code that is expected to throw an exception. If an exception is raised, the catch block implements the corresponding action to be taken for exception handling. Every try block must include either a catch block or a final block.

The exception that is expected to be raised must be declared as part of the catch statement. When an exception occurs, execution is transferred to the catch block. The catch block is executed if the raised exception matches the exception defined in the catch block.

A sample try and catch block implementation is provided below. The code creates a two-element array. The code, however, attempts to access the third element, which does not exist. As a result, an exception has been thrown.

```
import java.io.*; public demoException class { public static void
main(String args[]) { try { int arr[] = new int[2];
System.out.println("Accesing the 3rd element of the array:" +

arr[3]);

}catch(ArrayIndexOutOfBoundsException

exp)

{

System.out.println("Catching

Exception:"

+

exp);

}
System.out.println("Reached

outside

catch

block");

}

}
```

Implementing Multiple Catch Blocks

A line of code may result in multiple exceptions.

Implementation of multiple catch blocks is also permitted to meet this requirement. Below is the syntax for such an implementation.

```
try catch /*Protected Code*/ (ExpType1 exp1) *Catch block 1*/
```

catch (ExpType2 exp2) /*Catch block #2*/ catch (ExpType3 exp3)

/*Catch block #3*/

The syntax shown above demonstrates the use of three catch blocks. You can, however, use as many catch blocks as you want.

When this code is run, the protection is run. If an exception occurs, the type of exception is compared to the first catch block's exception. If the exception type does not match, the first catch block is skipped, and the exception type for the second catch block is checked for matching. The corresponding catch block is executed if it contains the same exception type as the raised exception.

CHAPTER 9

ADT, DATA STRUCTURES, AND JAVA

COLLECTIONS ARE COVERED

Type of abstract data (ADT)

A logical description of the data and the operations that can be performed on it is an abstract data type (ADT). ADT is defined as a data user's point of view. ADT is concerned with the data's possible values and the interface exposed by it. The actual implementation of the data structure is unimportant to ADT.

For example, suppose a user wants to save some integers and calculate their mean value. This data structure's ADT will include two functions: one for adding integers and another for calculating the mean value. This data structure's ADT does not specify how it will be implemented.

Data-Structure

Data structures are concrete representations of data that are defined from the perspective of a programmer. The data structure describes how data will be stored in memory. Every data structure has advantages and disadvantages. Depending on the type of problem, we select the best data structure for it.

Data can be stored in an array, a linked list, a stack, a queue, a tree, and so on.

Please keep in mind that we will be looking at various data structures and their API in this chapter. So that the user can use them without knowing how they are implemented internally.

Collection Framework in Java

The JAVA programming language includes the JAVA Collection Framework, which is a collection of high-quality, high-performance, and reusable data structures and algorithms.

The following are some of the benefits of using a JAVA collection framework:

1. Programmers are not required to repeatedly implement basic data structures and algorithms. As a result, it prevents the wheel

from being reinvented. As a result, the programmer can focus more on business logic.

2. The JAVA Collection Framework code is well-tested, high-quality, and fast. Using them improves the overall quality of the programs.

3. Development costs are reduced because basic data structures and algorithms are reused in the Collections framework.

4. Because most Java developers use the Collection framework, it is simple to review and understand programs written by other developers. The collection framework is also well documented.

Array An array is a collection of elements of the same datatype.

ADT Array Operations

The array's API is as follows:

1. Inserts an element in the kth position. In O(1) constant time, a value can be stored in an array at the Kth position. We simply need to save value at arr[k].

2. Reading the value in the kth position. It also takes O(1) constant time to access the value stored at some index in the array.

All that remains is to read the value stored at arr[k].

3. Replace the value stored in the kth position with a new value. O(1) constant time complexity.

For instance, public class ArrayDemo public static void main (String[] args) for (int I = 0; I 10; i++) arr[i] = I

The length of a JAVA standard array is fixed. When we don't know how much memory we need, we create a larger array. Thus wasting space. If an array is already full and we want to add more values to it, we must first create a new array with enough space and then copy the old array over to the new array. To avoid manual reallocation and copying, we can store data in ArrayLists from the JAVA Collection framework or Linked Lists.

Implementation of an ArrayList in JAVA Collections

In JAVA Collections, an ArrayListE> is a data structure that implements the ListE> interface, which means it can contain 75

duplicate elements. ArrayList is a dynamic array implementation that can grow and shrink as needed. (When the internal array is full, a larger array is allocated and the old array values are copied to it.) Import java.util.ArrayList; public class ArrayListDemo; public static void main (String[] args) Integer>ArrayList new ArrayListInteger> al (); l.add(1); / add 1 to the list's end al.add(2); / append number 2 to the end of the list

System.out.println("Array

Contents:

"

+

al);

System.out.println("Array

Size:

"

+

al.size());

System.out.println("Array

IsEmpty:

"

+

al.isEmpty());

al.remove(al.size() -1); / the last element of the array is removed.

al.removeAll(al); / The array's elements are all removed.

"Array IsEmpty: " + al.isEmpty()); System.out.println Output:

```
Array contents: [1, 2]

Size of the array: 2

IsArrayEmpty: false

IsArrayEmpty: true
```

List of Links

Linked lists are a dynamic data structure that allocates memory at run time. The linked list is not designed to store data linearly. A linked list node contains a link that points to the next elements in the list.

Because there is no direct access to linked list elements, linked lists are slower than arrays in terms of performance. When we do not know the number of elements to be stored ahead of time, a 76

linked list is a useful data structure. There are many different kinds of linked lists: linear, circular, doubly, doubly circular, and so on.

The list that is linked to Operations of ADT

The API for the Linked list is provided below.

Insert(k): inserts k at the beginning of the list.

Add an element to the beginning of the list. Simply add a new element and move the pointers. As a result, this new element becomes the list's first element. This operation will take O(1) time to complete.

Delete(): Removes the first element from the list.

Remove an element from the list's beginning. All we need to do is move one pointer. This operation will also take O(1) time to complete.

PrintList(): Displays all of the list's elements.

Begin with the first element and work your way through the pointers. This operation will take O(N) time to complete.

Find(k): Determine the position of the element with the value k.

Begin with the first element and follow the pointer until we find the desired value or reach the end of the list. This operation will take O(N) time to complete.

It should be noted that binary search does not work on linked lists.

FindKth(k): Locate the element at position k.

Begin with the first element and work your way through the links until you reach the kth element. This operation will take O(N) time to complete.

IsEmpty(): Check to see if the list has zero elements.

Simply check the list's head pointer; if it is Null, the list is empty; otherwise, the list is not empty. This operation will take O(1) time to complete.

Implementation of LinkedList in JAVA Collections

LinkedListE> is a data structure in JAVA Collections that also implements the ListE> interface.

Example:

java.util.LinkedList import; public class LinkedListDemo LinkedListDemo public static void main (String[] args) Integer>LinkedList ll = new LinkedListInteger>(); ll.addFirst(2); / The number 8 is added to the list. ll.addLast(10); / The number 9 is added to the end of the list. ll.addFirst(1); / The number 7 is added to the top of the list. ll.addLast(11); / The number 20 is added to the end of the list. System.out.println("Contents of Linked List: " +

ll); ll.removeFirst(); ll.removeLast(); System.out.println("Contents of Linked List: " + ll); ll.removeLast();
System.out.println("Contents of Linked List: " + ll);

Output: Linked List Contents: [1, 2, 10, 11]

[2, 10] Linked List Contents

Stack

A stack is a type of data structure that uses the Last-In-First-Out (LIFO) strategy. This means that the last element added will be the first to be removed.

The stack's various applications include:

Recursion: System stack is used to implement recursive calls.

1. Expression postfix evaluation.

2. Stack-based backtracking implementation

3. Tree and graph depth-first search

4. Converting a decimal number to a binary number, and so on.

ADT Stack Operations

Push(k): Pushes k to the top of the stack.

Pop(): Remove an element from the stack's top and return its value.

Top(): Returns the value of the stack element at the top.

Size() returns the number of stack elements.

IsEmpty() checks to see if the stack is empty. If the stack is empty, it returns true; otherwise, it returns false.

All of the preceding stack operations are implemented in O(1) time complexity.

JAVA Collection stack implementation

The stack is created by invoking the push and pop methods of the Stack T> class.

Example:

79

```
public school public static void main StackDemo (String[] args)
Integer>Stack stack = new IntegerStack (); int temp;
stack.push(1); stack.push(2); stack.push(3);
System.out.println("Stack: "+stack); System.out.println("Stack

size:

"+stack.size());
```

System.out.println("Stack pop: "+stack.pop()); Stack output: [1, 2, 3]

Stack height: 3

3 stack pop

Stack isEmpty: false Stack top: 2

The push and pop methods of the ArrayDequeT> class are also used to implement the stack.

ArrayDequeT> and StackT> are both provided by JDK. Both of these classes are applicable. However, ArrayDequeT> has some advantages.

1. The first reason is that StackT> is not driven by the Collection interface.

2. Second StackT> drives from VectorT>, allowing for random access and breaking stack abstraction.

3. Third ArrayDeque is more efficient than StackT>.

Queue

A queue is a data structure that operates on the First-In-First-Out (FIFO) principle. The first element added to the queue will be the first to be removed, and so on.

The following applications make use of queue:

1. Use of shared resources (e.g., printer)

2. Multiprogramming

3. The message queue

4. Queue is used to implement BFS or breadth-first traversal of a graph or tree.

ADT Queue Operations:

Add(K): Inserts a new element k at the end of the queue.

Remove() returns the value of an element that was removed from the front of the queue.

Front() returns the value of the queue element at the front.

Size(): Returns the number of queue elements.

IsEmpty() returns 1 if the queue is empty and 0 otherwise.

All of the preceding queue operations are implemented in O(1) time complexity.

Implementation of a queue in the JAVA Collection

The class implementation of a doubly ended queue is ArrayDequeT>. It will behave like a queue if we use add(), remove(), and peek (). (It also behaves like a stack when we use push(), pop(), and peekLast().) Import java.util.ArrayDeque; public class QueueDemo;

public

static

void

main;

(String[]

args)

Integer>ArrayDeque new ArrayDequeInteger> que (); que.add(1); que.add(2); que.add(3); System.out.println("Queue: "+que); System.out.println("Queue

size:

"+que.size());

System.out.println("Queue peek: "+que.peek()); Output:

Queue: [1, 2, 3]

Size of the queue: 3

81

Peek at the queue: 1 queuing remove: 1 queuing isEmpty: false
82

CHAPTER 10

FILE MANAGEMENT

This chapter goes over how to read, write, create, and open files in detail. There is a large selection of file I/O classes and methods to choose from.

Text File Reading

Reading a text file is an important skill in Java that has many practical applications. In Java, the FileReader, BufferedReader, and Scanner classes are useful for reading plain text files. Each of them possesses unique characteristics that make them uniquely qualified to handle specific situations.

BufferedReader

This method reads text from a continuous stream of character input. It buffers characters, arrays, and rows to make them easier to read. You have the option of specifying the buffer type or using the standard type. For the most part, the default is sufficient. Each read application generated by a Reader, in particular, directs the fundamental personality or byte stream to generate a specific read application. As a result, it is recommended to wrap a BufferedReader around any Reader whose read) (transactions, such as file readers and InputStreamReaders, can be costly. As an example:

New BufferedReader = BufferedReader in (Reader in, int size)
FileReader

Character documents of convenience to read. The

constructors of this class assume that the default character format and byte-buffer size are appropriate.

Scanner A simple text scanner that parses primitive types and strings using regular expressions. A Scanner uses a delimiter model to divide its input into tokens that, by definition, fit white space.

The resulting tokens can then be transformed into values of various types using various next techniques.

Import java.io.File, java.util.

Scanner; public class ReadFromFileUsingScanner throws Exception public static void main(String[] args)

/ The filepath is now set as a parameter so that it can be scanned.

Example

File

=

new

File("C:UsersuserNameDesktopexample.txt"); Example Scanner =

new Scanner(file); while (example1.hasNextLine()) Using the 84

Scanner

class

but

without

loops:

System.out.println(example1.nextLine());

Import

java.io.File,

java.io.FileNotFoundException,

java.util.Scanner; public class example

FileNotFoundException is thrown by public static void main(String[]

args).

File

example

=

new

File("C:UsersuserNameDesktopexample.txt"); Scanner example1 =

new Scanner(file); / we will use Z as a delimiter sc.useDelimiter("Z"); System.out.println(example1.next()); Read a text file as String in Java package io; import java.nio.file.*; Exception is thrown by public static example(String fileName).

String

example1

=

"";

string

example1

=

new

String(Files.readAllBytes(Paths.get(fileName)));

return

string

example1;

The exception is thrown by public static void main(String[]

args).

String

example1

=

example("C:UsersuserNameDesktopexample.java");
System.out.println(example1); Text file writing

To write bytes or lines to a file, use one of the write methods.

Methods of writing include: (Path, byte[], OpenOption...)
Write(Path, Iterable, Charset, OpenOption...)

File Renaming and Deletion

Renaming

To rename a file in Java, we can use the renameTo(fileName) method within the File class.

Deleted Files

Files saved with the Java program will be permanently deleted without being moved to the trash/recycle bin. Using java.io.File.delete, this abstract pathname is removed from the file or folder. If a folder exists, java.nio.file.files.deleteifexists will delete it. If the folder is not open, it deletes a folder from the route as well.

Java Advanced Topics

Generics

Bugs are an unavoidable part of any non-trivial software project. Although careful planning, programming, and testing can help reduce their omnipresence, they will always find a way to enter your system somewhere. This becomes especially apparent as new features are added and the size and complexity of your codebase grow.

Some bugs, fortunately, are easier to detect than others.

Compile-time bugs, for example, can be identified quickly; you can use the compiler's error codes to determine what the problem is and solve it right away. Runtime bugs, on the other hand, can be much more difficult to detect; they do not always occur instantly, and when they do, they may occur at a point in the program that is far from the true source of the problem. Generics help to stabilize your software by making it possible to identify more bugs at compile time.

Generics, in a nutshell, allow types (classes and objects) to have parameters when defining courses, interfaces, and techniques. Type parameters, like the more familiar formal 86 parameters used in method declarations, provide distinct outputs that allow you to reuse the same code. The distinction is that values are formal parameter inputs, whereas kinds are type parameter responses. Generic code has many advantages over non-generic code. Programmers can use generics to introduce generic algorithms that operate on different types of collections, can be tailored, and are secure and easier to read.

Types of Generics

A generic form is a generic category or interface that has been type-parameterized. To demonstrate the concept, a simple box class can be modified. Consider a non-generic box category that can be applied to any type of object. It only needs two techniques: set), (adding an item to the cabinet), and get), (retrieving it. Because their methods accept or return an object, you can pass in whatever you want as long as it is not a primitive type. At compile-time, there is no way to check how the class is used. One part of the software may put an integer in the cabinet and expect to get integers out of it, while another part of the software may move through a string incorrectly, resulting in a runtime error.

The title of the category is displayed in the segment of the type parameter delegated by angle brackets (>). It denotes the form parameters (also known as sort factors) T1, T2, and T. To update the box category to use generics, change the file "government class box" to "government class box T >" to generate a generic type statement. This introduces the type variable, T, which can be used anywhere within the class. This replaces all Object events with the string T. Any non-primitive type that you specify can be a type variable: any type of class, interface, array, or even another type variable. This method can also be applied to generic interfaces.

By convention, type designations are single upper case letters.

This stands in stark contrast to the variable naming conventions you are already familiar with, and for good reason: without this convention, it would be difficult to distinguish between a type variable and a normal class or object name.

The following are the most commonly used type parameter names:

E - Element (commonly used by the Java Collections Framework)

N - Number K - Key

T - Type

V - Value S,U,V, and so on—2nd, 3rd, and 4th types

These names will appear throughout the JDK and API.

Generic Type Invocation and Instantiation

To refer to the generic box category within your system, perform a generic type invocation that returns T with a specific concrete value, such as Integer:

Box integerBox;

You might think that invoking a generic sort is similar to invoking a normal process, but instead of adding an assertion to a procedure, you transfer a type argument—Integer in this case —to the box category itself.

Many designers use the terms "type parameter" and "type statement" interchangeably, but they do not have the same definitions. When coding, sort arguments are used to generate a parameterized type. As a result, the T in Foo T > is a type parameter, and the Foo String > f string is a type contention. This class adheres to this concept by using these words.

This software, like any other variable statement, does not generate a new item in the box. It simply states that integerBox will have a reference to an "Integer Box," which is how Box Integer> is read. A parameterized type is the name given to a generic type invocation.

To instantiate this class, use the new keyword as usual, but replace the class name with Integer>:

Box integerBox = new BoxInteger>();

In the most recent versions of Java, you can substitute an empty set of type arguments (>) for the type arguments required to invoke a generic class constructor as long as the compiler can determine or infer the type arguments from the context. This angle bracket couple, >, is known colloquially as "The Diamond." For example, you can use the preceding declaration to create a Box Integer> instance: Box new Box = integerBox> () ;

Generic methods

Generic techniques are methods for implementing parameters of their type. This is similar to a generic type declaration, except that the type parameter's scope is restricted to the method in which it is declared. Static and nonstatic generic techniques are permitted in addition to generic class constructors.

A generic technique's syntax consists of a list of type parameters enclosed in angle brackets that appear before the procedure's return type. For static generic techniques, the type parameter segment must come before the method's return type.

The Util class contains compare, which is a generic technique for comparing two Pair items.

ex1 = new Pair> PairInteger, String>

PairInteger, String> ex2 = new Pair> (49, "string1"); boolean comparison = Util.Integer, String>compare(ex1, ex2); Type Parameters with Bounds

Sometimes you want to limit the types that can be used as form statements in a parameterized type. A technique that works on figures, for example, could only recognize the Number of cases or their subclasses. Limit parameters are used for exactly this purpose.

To indicate a defined type parameter, list the name of the type parameter, followed by the extension's keyword, followed by the upper bound, which is Number in this case. It is worth noting that the word expands is commonly used in this context to mean either

"extends" (as in courses) or "implements" (as in interfaces).

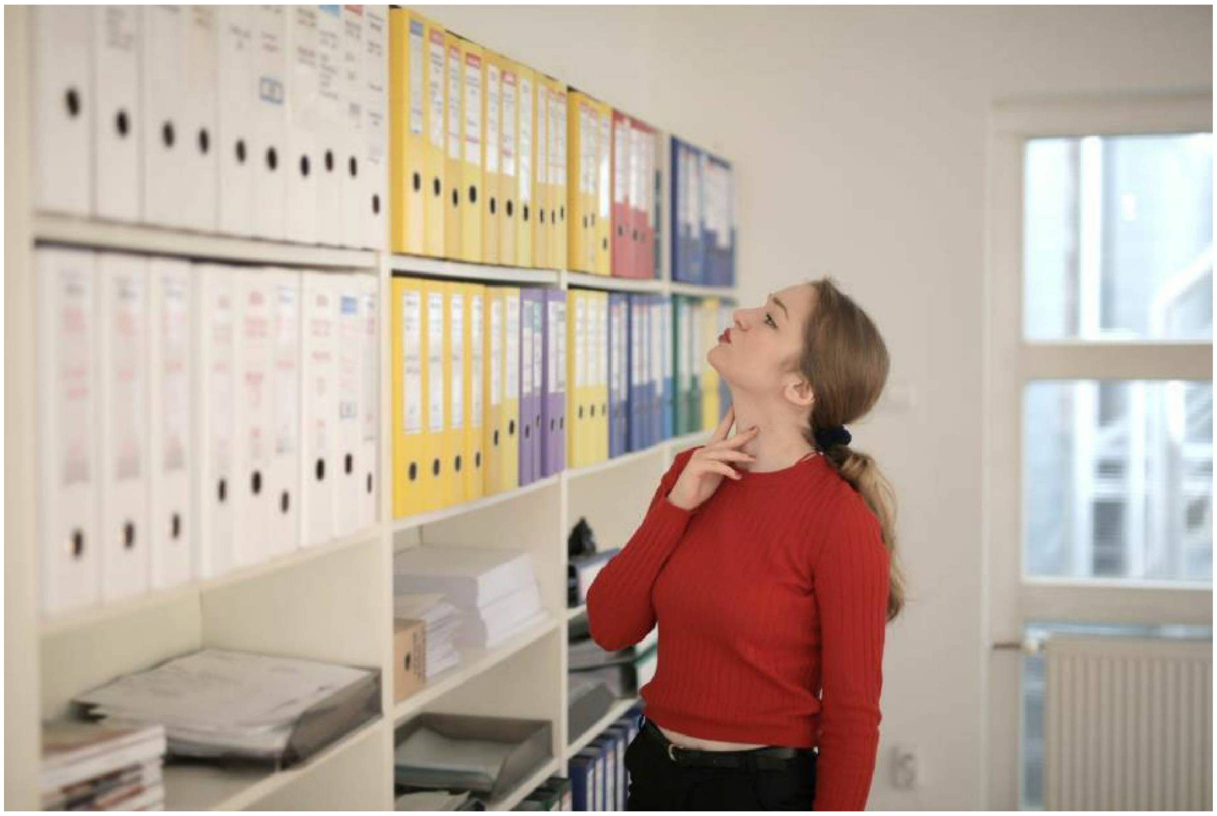
Multiple Boundaries

Type parameters can have multiple bounds. A type variable with multiple bounds is a subtype of every type in the bound. A class must be specified first if one of the bounds is a class. As an example:

Class X /*... */ interface Z /*... */ interface Y { /* ... */ }

class R T extends X, Y, and Z> { /* ... */ }

If bound X is not specified first, the following compile-time error occurs: class R T extends Y, X, and Z> /*... */ / compilation error



CHAPTER 11

COLLECTIONS

Functional collections

We expose every step when we externalize code. This is a defining feature of imperative programming. However, one of the recurring themes of functional programming is the concept of doing the inverse: internalizing code. Internalizing code is the process of hiding details within a function. However, functional programming takes this concept a step further by transforming everyday programming micro-patterns into functions. One of those micro-patterns is iterating over data, whether with iterators, for-loops, or while-loops. Functional programming alters our perception of them.

Java's functionalization effort would be incomplete unless the Collections library, specifically the Collection, Map, List, and Set interfaces, were redesigned. This is because iteration is frequently performed over collections, which are the foundation of Java programs. It is a natural location for the functionalization effort to take place. However, any significant changes, such as adding new methods to the Collection interface, would break backward compatibility with all pre-Java 8 programs. This includes not only the JDK's hierarchy that extends the Collection interface but also any open or closed source third-party library and in-house classes.

Java's designers have never used, and will never use such a strategy. Nonetheless, changes were required; the library had been introduced in 1998, eons in the software industry years ago, and was showing its age.

The need to modernize and functionalize the Collections library prompted the introduction of default methods. Default methods are ideal because they allow behavior to be added at the root of the hierarchy without disrupting dependent classes.

Subclasses have the option of inheriting or overriding the behavior.

Subclasses, unlike interface methods, can automatically accept new default methods without recompilation. There is instant compatibility. Default methods have proven useful in and of themselves, but their existence is due to the need to functionalize the Collections library.

Parallelization is a recurring theme in functional programming.

Simply put, functional programming improves the mousetrap for parallel processing. As a result, the Collections library has been enhanced to take advantage of multi-core CPUs when processing collections. The Collections library, along with Streams, is at the forefront of bringing functional-style parallel processing to Java.

We can now apply what we've learned about Java's standard functional interfaces to Collections. Let's take a look at what happened to the Collections library in Java 8.

Interface for collection

We'll start with a brand new default method in the Collection interface that's available to lists and sets. The forEach() method is as follows:

This method loops through the entire collection, allowing the Consumer to determine what to do with each element. This is an example of declarative programming and the concept of internal iteration. The method for iterating is not specified. This is in contrast to external iteration, which specifies how to iterate as well as what to do with each element in code.

We will demonstrate these functionalized collection methods using the slapstick comedy trio of The Three Stooges from the golden age of Hollywood films: Larry, Moe, and Curly. To begin, print the contents of a collection:

The forEach() method can be used in a variety of situations. It is a far more convenient method of iterating through collections and should be your de-facto standard. However, because it is a declarative construct, there are some limitations. Most notably, unlike in a while loop, you cannot change the state of local variables. Algorithms must be functionally rethought. For the time being, just know that forEach() is ideal for iterations that do not change state.

We will now look at another new method in Collection:

The method removeIf() internalizes the entire iteration, testing, and removing process. It only needs to be informed of the condition for removal.

We can easily determine what type of lambda to use by using the now-familiar predicate functional interface.

ReplaceAll() can be used to replace all of the contents of a List: ReplaceAll(functional)'s interface is an UnaryOperator, which is a Function sub interface.

All subclasses can use replaceIf() and replaceAll(), with one caveat: the underlying class must support removal or an exception will be thrown.

These examples demonstrate the brevity of functional programming. The function did the majority of the work, with the lambda providing the details.

The map interface

The constant need to check for the presence of the map before adding, updating, or removing an element is one of the most vexing aspects of using lists as values in maps. First, you must attempt to extract the list and, if it cannot be found, create it. Assume we have a method that updates a movie database that is implemented as a Map. The key of the map is the year of the film, and the value is a list of films from that year. Before Java 8, the code would have looked like this:

With these default methods, Java 8 provides a better alternative:

Let's begin with the computation methods. Each variant allows the mapping function to generate the map's value. Mapping happens

conditionally

in

computeIfPresent()

and

computeIfAbsent(). So, using these methods, we can refactor the following code example:

It's worth noting that the computeIfAbsent() lambda handles the list's creation. When it comes time to add the movie to the list via the compute() method, the add() method will never throw a NullPointerException because the list is guaranteed to exist.

In this case, using the computeIfAbsent() method is overkill; instead, we should use putIfAbsent():

This method has no lambda and expects a value to be given rather than calculated. Even though no lambda was used, this is still a functional style method. It demonstrates that functional code can be expressed without the use of lambdas.

If you still need to extract the data, you can use the getOrDefault() method for a more functional approach:

As an alternative, you can use the merge() method. It makes checking the existence of a list easier. If the key (year) does not exist on the map, it places the value (titles) on the map. If it exists, a BiFunction can decide what to do with the two lists: 95

It can also be used as follows:

The BiFunction can also return null, indicating to the merge that the key should be deleted.

The takeaway is that we have eliminated the overhead code of checking for the existence of an element and can now concentrate on what is important: defining how to create the list and how to add an element to the list.

The map interface has been enhanced with additional functional methods such as forEach(), replace(), and replaceAll(), and it employs the same code internalization principle. The complete list can be found in the appendix.

Spliterator

The Collections library is still subject to the same concurrent access restrictions. As always, select the Collections library class that corresponds to your thread safety needs. This is because the new methods shown above are simply functional abstractions riding on top of the same underlying data structures. Because they are still based on multiple threads mutating the collection and synchronizing access to the underlying data, these methods are not particularly amenable to functional programming's take on parallel processing. There is, however, a new Java 8 abstraction that is compatible. It is intended to perform parallel data iteration. The Spliterator interface embodies the concept. This interface's premise is to partition data and pass chunks to different threads.

Spliterators can be obtained via the Collection interface, which includes the subinterfaces List and Set.

These three methods are central to the Spliterator interface: 96

The method trySplit() divides the underlying data into two parts. It generates a new Spliterator with half of the data and retains the other half in the original instance. Each can be assigned to a thread, which iterates through the partitioned data using forEachRemaining (). The tryAdvance() method is a one-at-a-time variant that returns the next element or null if the list is empty.

Spliterators do not perform parallel processing themselves, but rather provide the abstraction for it. Here's an example of the concept in action:

The method isMovieInList() parallelizes the search to determine if it is contained in the list given a title and a list of movies. If found, it sets the flag in booleanHolder to true. It selects a Spliterator instance from the list, divides it in half, and distributes one half to each thread. If more threads become available, the splitting process can be repeated.

Spliterators can be obtained from other Collection types as well as from other JDK libraries. Numerous implementations deal with various properties such as finite/infinite, ordered/non-ordered, sorted/non-sorted, and mutable/immutable.

They take on the characteristics of their underlying data structure.

Spliterators are low-level abstraction that provides finer-grained control over parallelized iteration. The API, on the other hand, lacks some of the refinements required to implement functionally friendly algorithms. For the search in the preceding example, we needed to store the state in the BooleanHolder.

Wrap up

This wraps up our look at the new and improved Java 8

Collections library, as well as the standard functional interfaces.

Expect major changes from third-party APIs, just as there have been many changes in the standard JDK libraries to support functional concepts. But the most significant change is yet to come.

Important points

A set of functional interfaces is included in the new package java.util.function. These are divided into four families, each of which is represented by an archetype: Consumer, Function, Predicate, and Supplier.

Each functional interface family defines variants that specialize in type and arity.

Methods that enable functional composition are also defined by functional interfaces. Multiple lambdas can be combined to form super functions that appear to be one.

The Collections library has been updated and made more functional. This was accomplished by using default methods at the top levels of the hierarchy, ensuring backward compatibility.

Internal iteration has been considered when designing the new functional methods in Collections, Lists, Sets, and Maps. Methods that iterate over collections and act on each element are given behavioral parameters in the form of lambdas and method references.

Internal iteration is a type of declarative programming that is essential in functional programming. It relieves the developer of the need to describe the "how" and instead focuses on the "what"

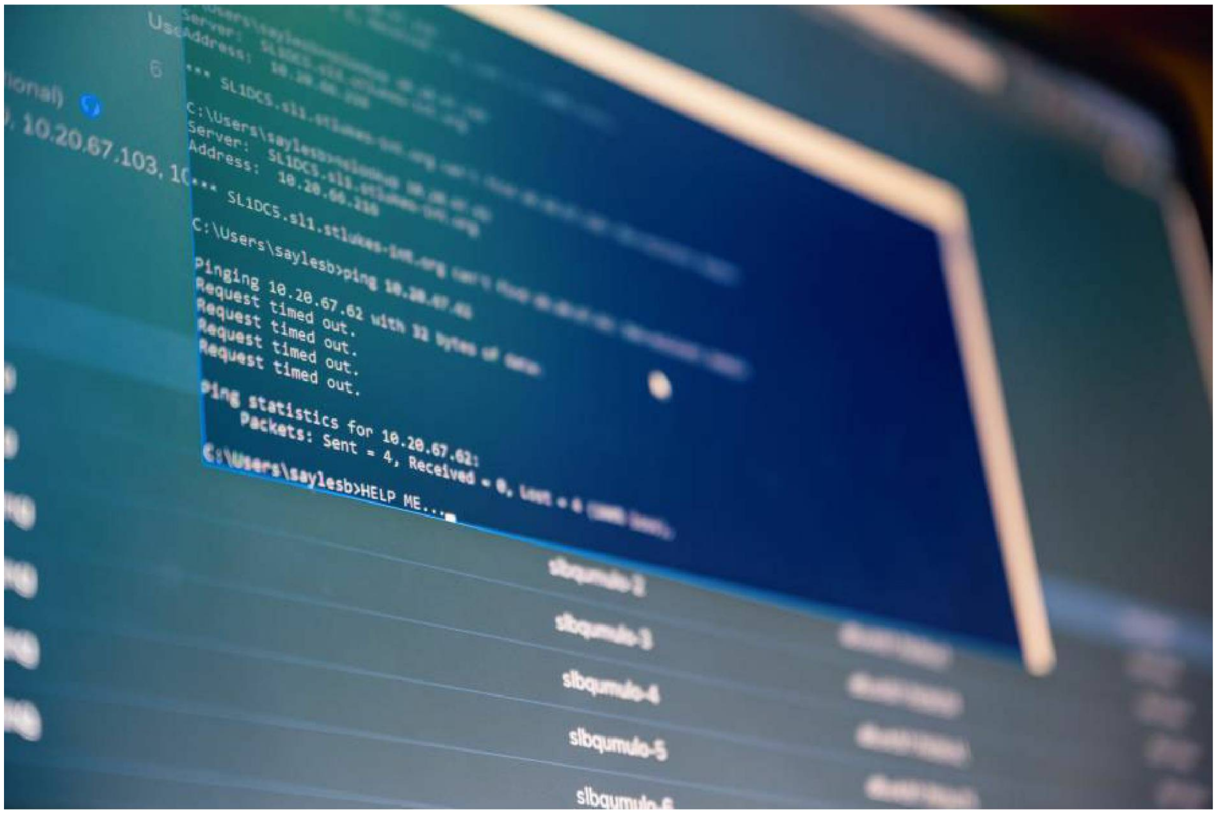
to do.

98

Spliterators are intended for iterating over collections in parallel. Data is partitioned, and each chunk is distributed to various threads for parallel processing.

A non-prime number is greater than one.

99



```
*** SL1DC5.sl1.stlukes-int.org
C:\Users\saylesb>
Server:  SL1DC5.sl1.stlukes-int.org
Address:  10.20.66.210
*** SL1DC5.sl1.stlukes-int.org
C:\Users\saylesb>ping 10.20.67.62
Pinging 10.20.67.62 with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Ping statistics for 10.20.67.62:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
C:\Users\saylesb>HELP ME...
```

CONCLUSION

The Java Development Kit (JDK) is a necessary tool for compiling, documenting, and packaging Java programs. Along with JRE, the JDK includes an interpreter or loader, a compiler called javac, an archiver (JAR), the Javadoc document generator, and many other tools required for successful Java development.

The Java Runtime Environment is abbreviated as JRE. It is the execution environment for Java bytecode and is used to implement the Java Virtual Machine. The JRE also includes all of the class libraries and other support files that the JVM requires at runtime.

It is, in essence, a software package that contains everything we need to run Java programs, as well as a physical implementation of the Java Virtual Machine.

JVM is an abbreviation for Java Virtual Machine. The JVM is an abstract machine, a specification that provides us with the JRE that executes our bytecode. The JVM must adhere to the following 100

notations: Specification, which is a document that describes how the JVM is implemented, Implementation, which is a program that meets the JVM specification requirements, and Runtime Instance, which is the JVM instance that is created whenever a Java command is typed into the command prompt and a class is run.

All three are inextricably linked and rely on one another to function.

With this, I'd like to thank you for choosing my Java programming guide. As you can see, it is a simple yet complex language with a wide range of topics to learn. You should now have a solid understanding of the fundamental concepts of Java programming and how to apply them.

Simply put, your next step is to practice. And don't stop practicing. You can't possibly read this guide once and think you know everything. I strongly advise you to take your time with this; carefully follow the tutorials and don't move on from any section until you fully understand it and what it all means.

There are several useful Java forums available online, full of people ready and willing to assist you and point you in the right direction. There are also a plethora of online courses available, some of which are free and some of which require payment, but all of them are beneficial and can assist you in taking your learning to the next level.

Did you find this guide useful? I hope it was everything you hoped for and more, and that it has put you on the right track to landing your dream job!

I hope you found my overview of computer programming useful. It's a very basic start, but it should have given you an idea of where to start. It should also have demonstrated that computer programming isn't all that difficult and can be quite exciting,

especially once you start seeing your results appear on the screen and your computer, in short, doing what you tell it to do!

If this has given you a good idea of what to expect, you may want to progress to more advanced programming in your chosen language. A word of caution: don't try to learn more than one language at a time; otherwise, you'll get lost. At this point, the only other piece of advice I can give you is to practice...and practice some more. The more you do, the more you will learn and desire to learn.

Thank you for downloading my book; if you found it useful, please leave a review on Amazon.com.