Hyperiondev

TASK

Java - Basics

Visit our website

Introduction

WELCOME TO THE JAVA - BASICS TASK!

Java is a high-level programming language developed in 1995 by Sun Microsystems. This general-purpose language is simple to learn, highly portable and can run on a variety of platforms. This task provides a brief overview of both software engineering itself and the fundamental concepts of Java such as variables, data types and control structures.

WHAT IS SOFTWARE ENGINEERING?

To understand the concept of software engineering, you first need to understand what software is. You might think that a software engineer is simply concerned with computer programs that provide desired features, functions, and performance when executed. However, this is not entirely the case. A software engineer is not just responsible for computer programs, but also all associated documentation and configuration data, which ensures the correct operation of the program. Software that can be sold to a customer is known as a **software product**.

Now, what is software engineering? The IEEE (1993) defines software engineering as follows:

> *Software engineering: (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1).*

In simpler terms, software engineering is an engineering discipline, which is concerned with the development of a software product using well-defined scientific principles, methods and procedures.

SOFTWARE ENGINEERING VS PROGRAMMING

You might be tempted to use the terms *programmer* and *software engineer* interchangeably since they both develop software applications. However, this would be inaccurate. There are some distinct differences between the responsibilities and approaches to the job for these two roles.

Much like other engineering disciplines, software engineering approaches developing software as a formal process of understanding requirements, designing software, deploying, testing and maintaining it. Computer programmers, on the other hand, write code to the specifications given to them by software engineers. Software engineering is also a team activity, while programming is a solitary one.

WHAT MAKES A GOOD SOFTWARE ENGINEER?

You might assume that the most important quality that a software engineer can possess is technical experience and knowledge, however, this is not entirely true. A good software engineer is more than the sum of the technologies with which they have worked. Below is a list of some important traits that set successful software engineers apart:

- **Passion for Code** — This will come as no surprise, but a good software engineer should have a passion for programming. A software engineer should have an inherent interest in programming and enjoy working with code, as it is something that you will be working with almost every day.

- **Team Player** — Very few projects are small enough to be undertaken by a single person. Therefore, software engineers often find themselves working in a team. Software engineers must have excellent teamwork skills.

- **Excellent Communication Skills** — Software engineers should be able to communicate successfully with team members as well as clients. They should have both great written and verbal communication skills and be able to write reports as well as give instructions.

- **Curious** — Good software engineers are curious people. They want to know why and how things happen, like how the code and conditions produce certain software behaviour. This curiosity enables these engineers to think outside the box and come up with creative solutions to problems.

- **Wide-Ranging Technical Experience** — You can't escape the fact that a successful software engineer should have broad technical experience. Knowing a single programming language is not sufficient. A good software engineer should be well-versed in best practices like agile, task management software, version control and working in different development environments. A good software engineer should also always be willing to learn new technologies.

This is by no means an exhaustive list of the qualities of a good software engineer. However, it gives you a sense of what it takes to become successful in this field.

THE ECLIPSE IDE

An IDE, or Integrated Development Environment, is an essential tool used by software engineers to develop code. It consists of a source code editor, a compiler and a debugger. It increases a software engineer's productivity by condensing the entire process of code creation, compilation and testing. The Eclipse IDE is the most popular IDE for Java, with a market share of 65% in 2014. Eclipse is primarily used for developing Java applications, however, it can also be used for developing applications in other programming languages via plug-ins. These languages include JavaScript, PHP, Python and Ruby to name a few.

CREATING A SIMPLE JAVA PROGRAM

The following Java program will display the message "Hello World!" on the console (the computer's display device).

```
public class Welcome {
    public static void main(String args[]) {
        System.out.println("Hello World!");
        //prints out Hello World
    }
}
```

The first line in the above program defines a class. Every Java program must have at least one class and every class has a name (you will learn more about classes in the upcoming tasks). The class defined above is named *Welcome*. The class must be the same name as the file it's in.

The second line of the program defines the **`main()`** method. A method is a construct that contains statements, like a function. A class may contain many methods. However, the *main()* method is the starting point where the program begins execution.

The *main()* method above contains the **`System.out.println()`** statement, which outputs the message "Hello World" to the console. Note that all statements in Java end with a semicolon (**`;`**), which is known as the statement terminator.

Line four of the program is a comment. Single-line comments are preceded by double slashes (`//`), while multiline comments are enclosed between `/*` and `*/`.

All of the program's components are grouped by using a pair of curly braces (`{ }`). Every class groups the data and methods of the class and every method groups the statements in the method.

A note from the

HyperionDev Team

The History of Software Engineering

In 1968 a conference was organised by NATO to discuss the "software crisis". The software crisis was the name given to the difficulties encountered in the 1960s with developing large, complex systems. This was when the term "software engineering" was first introduced. It was suggested that the use of the engineering approach to develop software would reduce development costs and lead to more reliable software.

In the 1970s, the concepts of structured programming were developed. The concepts of object-oriented development were also introduced with the development of Smalltalk languages. During the late 1970s, the first programming environments were created.

During the 1980s, the Ada programming language was developed. This language included concepts of structured programming and information hiding. CASE (Computer-Aided Software Engineering) tools were also introduced during this time to support design methods. The 1980s also saw increased use of object-oriented programming with the use of languages such as C++ and Objective-C as well as object-oriented design methods.

In the 1990s, object-oriented programming became mainstream and Java was developed and released. During this time the concept of software architecture received a great deal of attention. There was also increased use of client-server distributed architectures. The concept of component-based software engineering as well as the UML was also proposed.

During the early 2000s, the use of IDEs (Integrated development environments) became more common. The use of UML also became widespread. Use of scripting languages, like Python, for software development also increased. C# was also developed in the early 2000s and was seen as a competitor to Java.

READING INPUT FROM THE CONSOLE

Java uses `System.out` to refer to the standard output device, which is the computer's display monitor. It uses `System.in` to refer to the standard input device, which is the keyboard.

In the program above, we use the `System.out.println()` method to print a message to the console, but what if we would like to read input from the console? To do this, create a *Scanner* object to read input from `System.in`. You do this by using the following syntax:

```
Scanner input = new Scanner(System.in);
```

`new Scanner(System.in)` creates a Scanner object (you will learn more about objects later), while the code `Scanner input` declares a variable of type `Scanner`. However, before creating a new `Scanner` object, we need to import the `java.util` package. In order to do this we use the following code:

```
import java.util.Scanner;
```

VARIABLES

Variables are used to store values that you might wish to use later in the program. They are called variables because the values they store can be changed.

A variable needs to be declared before it can be used. Declaring a variable informs the compiler how much memory to allocate to the variable based on its data type. Unlike in JavaScript, to declare a variable, you need to specify the data type followed by the variable name.

Examples of variable declarations can be found below:

```
int number;
double interestRate;
```

After the variable is declared, it can be assigned a value by using the assignment operator (`=`).

```
number = 234;
interestRate = 4.56;
```

NAMED CONSTANTS

A named constant holds a permanent value that cannot be changed. To declare a constant, use the following syntax:

```
final double PI = 3.14159;
final int MAX_VALUE = 100;
```

It is good practice to use capital letters to name constants.

NUMERIC DATA TYPES

There are six numeric data types for integers and floating-point numbers in Java.

- Integer types: byte, short, int and long.
- Floating-point number types: float and double.

Name	Range	Storage Size (bits)
byte	-128 to 127	8
short	-2^{15} to $2^{15}-1$	16
int	-2^{31} to $2^{31}-1$	32
long	-2^{63} to $2^{63}-1$	64
float	Negative range: - 3.4028235E+38 to - 1.4E - 45 Positive range: 1.4E - 45 to 3.4028235E + 38	32
double	Negative range: - 1.7976931348623157E + 308 to - 4.9E - 324 Positive range: 4.9E - 324 to 1.7976931348623157E + 308	64

CASTING

You can convert one data type to another using casting. There are two types of casting in Java, implicit and explicit.

Implicit casting occurs automatically when you assign a value to a variable whose type supports a larger range of values. For example, you can assign an `int` value to a `float`. This is known as widening a type.

An example of implicit casting is as follows:

```
int number1 = 5;
float number2 = number1;
```

Casting a type with a large range to a type with a smaller range is known as narrowing a type. Use **explicit casting** to do this. To explicitly cast a type, specify the target type in parentheses, followed by either the variable's name or value you wish to cast.

An example of explicit casting is as follows:

```
int number1 = (int)23.34;
```

```
long longNum = 656666L;
int intNum = (int) longNum;
```

THE CHARACTER AND STRING DATA TYPES

In Java, you can represent a single character using the character data type, `char`. All character values must be enclosed between single quotation marks. For example:

```
char letter = 'N';
char number = '5';
```

You can also represent a string of characters using the `String` data type. Like many other programming languages, all strings must be enclosed between double quotation marks. For example:

```
String newString = "Java is fun!";
```

THE BOOLEAN DATA TYPE

As in JavaScript, the boolean data type can only hold one of two possible values: true or false. Boolean values are the result of comparison operations. For example, the following statement will display true:

```
int number = 10;
System.out.println(number < 100);
```

IF-ELSE STATEMENTS

As you know, selection statements play a pivotal role in programming. Selection statements allow the program to decide which statements to execute based on a condition. An *if statement* will execute a block of code only if the condition is true. The syntax for an if statement in Java is as follows:

```
if (boolean expression) {
    statement(s);
}
```

What happens if the condition is false? If you had an *if statement*, nothing would happen. If you want to take alternative actions when the condition is false, add an *else statement* to create an *if-else statement*. If the condition turns out to be false, the statements indicated by the *else statement* will be executed. The syntax for an *if-else statement* is as follows:

```
if (boolean expression) {
    statement(s);  // executed if boolean expression evaluates to true
}
else {
    statement(s);  // executed if boolean expression evaluates to false
}
```

An *if* or *if-else statement* can also be placed within another *if* or *if-else statement* to form a *nested statement*.

For example, the following code tests what symbol a student should be awarded based on their class mark. If the student achieves a mark of 80 or higher, they receive an A, else if they achieve a mark of 70 or greater, they receive a B, and so on.

```
if (mark >= 80)
    mark = 'A';
else if (mark >= 70)
    mark = 'B';
else if (mark >= 60)
    mark = 'C';
else if (mark >= 50)
    mark = 'D';
else
    mark = 'F';
```

THE WHILE LOOP

Loops are used to control how many times a statement or sequence of statements are executed. The *while loop* is a simple loop that executes statements repeatedly while a condition is true.

The syntax for the while loop is:

```
while (continuation condition) {
    statement(s);
}
```

To avoid an infinite loop (loop that never terminates), you must make sure that the continuation condition will eventually become false and the loop stops running.

THE FOR LOOP

The *for loop* provides a concise, simple way to iterate over a range of values. The syntax for the for loop is:

```
for (initialisation; termination; increment) {
    statement(s);
}
```

The initialisation expression is executed once when the loop begins. It is normally used to initialise a control variable. Next, the termination expression tests whether the control variable has reached its termination value. If not, the statements in the body of the loop are executed. The increment expression then increments or decrements the control variable and the termination value is tested once again. The process is repeated until the variable has finally reached its termination value.

DEALING WITH ERRORS

Everyone makes mistakes. Maybe you've made some already. It's important to be able to *debug* your code. As you learned with JavaScript, as you code, you will encounter the following types of errors:

- **Compilation errors**: When you try to compile your code, you get an error in the bottom JGrasp panel. This is due to missing semicolons, syntax errors, etc. Here's an example of such an error that you may see when you try to compile your code:

```
example.java:93: error: ';' expected
```

This means that on line 93 of the file example.java, you forgot to insert a semicolon. To 'debug' this problem, while in JGrasp, hold down the CTRL key and press L. You should be able to see line numbers on the left of the program pane. Go to the line indicated by the error message and correct the error, then try to compile your code again.

- **Runtime errors:** Your program compiles fine, but when it is actually run, an error occurs and the program stops. The error is also printed in the JGrasp panel. Say we had this line in our program:
  `int a = Integer.parseInt("18.2");`

  Your code would compile properly, but when running you'd get the error:

```
 Exception in thread "main" java.lang.NumberFormatException: For
input string: "18.2"
      at
java.lang.NumberFormatException.forInputString(NumberFormatExcepti
on.java:65)
      at java.lang.Integer.parseInt(Integer.java:492)
      at java.lang.Integer.parseInt(Integer.java:527)
      at example.main(example.java:107)
```

  Look carefully. The type of exception is “java.lang.NumberFormatException”. “NumberFormatException” must have something to do with the format of the number you gave `Integer.parseInt`. You can't cast 18.2 to an Integer because integers don't accept decimal points. It's important to think in a deductive way to solve runtime errors.

- **Logical errors**: Your program compiles and runs, but the output isn't what you expected. This means your logic applied to the problem as a solution is wrong. When we introduce loops and control statements, these will become more frequent. Even after years of programming, it takes a long time to sit down and design an algorithm and find out why it isn't working.

A note from the

HyperionDev Team

Some Java History:

Java was created primarily by a man named James Gosling, working at Sun Microsystems — now owned by the Oracle Corporation. The so-called “Green Team” he led had planned for the language to be used to unite different consumer devices. It was designed to be lightweight so it could run using the devices’ limited hardware while still being able to allow communication between devices. However, once the internet began to gain traction, the team realised their new language would be ideal for it, and shifted their focus.

Java’s popularity is due to its small footprint on electronic devices. Its early boom into the technological world means it was given a chance to grow and flourish enormously.

So take a note from the Green Team, and keep in mind that even if you don’t end up going where you had planned, it isn’t always a bad thing!

Compulsory Task 1

This task will walk you through how to install Eclipse.

- To use Eclipse for Java, you will first need to install the Java Development Kit (JDK). If you do not have it already installed you can find a tutorial on how to install it **here**.
- Once the JDK is installed, you can download Eclipse from **https://www.eclipse.org/downloads/**
    - Under “Get Eclipse Neon” select “Download Packages.”
    - Then choose “Eclipse IDE for Java Developers.”
    - Under “Download Links” select your operating system.
    - You then click on the “Download” button to begin your download.
    - Once the file has downloaded, unzip it into a directory of your choice and click on the eclipse.exe file (Windows) to start the IDE.
- When you have successfully installed and downloaded Eclipse, you will be required to select a workspace directory. Once you have chosen a directory, select OK.
- You are then greeted by the welcome screen. Take a screenshot of this screen and send it to us to show that you have successfully installed Eclipse.

If you are having any difficulties installing or downloading Eclipse or the JDK, please feel free to contact an expert code reviewer.

Compulsory Task 2

Follow these steps:

- Create a new file called **highRoller.java**
- You are going to create a game where you play against the program. You will simulate you and the program each rolling a 6-side dice and see who rolls the highest number. The best of three rounds wins.
- The program should randomly generate a number between 1 and 6 to represent the roles of the dice *(Hint: look up Math.random)*
- The program should then prompt the user to 'roll their dice'.
- Once the user has 'rolled their dice', the program should print both numbers and inform the user whether they win, lose or draw.
- The best of three rounds wins.

Compulsory Task 3

Follow these steps:

- Create a new file called **prime.java**
- Write a program that determines if a number entered by a user is a prime number.
- Prompt the user to enter a positive integer.
- Then calculate if the number is prime (i.e. it only has factors of 1 and itself).
- Finally, print out the number and state whether or not it is a prime number.

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.