2.1.1. What is a computer programming language?

Language is a means to communicate or interact.

For example, in this learning program, we are using English as the language to communicate or provide instructions to the learner.

Similarly, there are certain languages which can be used to provide instructions to computers. These are called computer programming languages.

These languages provide a way to represent data (like numbers, text or images, etc) and also provide a way to represent instructions which manipulate or work with that data.

There are different types of programming languages. In our current course we will be learning a computer programming language called **Java**.

Java is an Object-oriented programming language. An Object-oriented programming language models data into **Objects** and instructions to manipulate data as **Methods**. We will learn in detail about Objects, Methods and other aspects of Object-oriented programming language in later sections.

Other popular examples of Object-oriented programming languages are C++ and C#.

In the English language, if one wants to greet someone, one would say "Hello!".

Similarly in Spanish, one would say "Holla!".

If we want a computer to display a greeting message on the screen, we will have to provide instructions to the computer using a computer programming language.

For example, if we want to display "Hello!" using **Java** as the programming language, we will have to provide instructions as per Java programming language's grammar and vocabulary as given below:

```
public class Greetings {
```

- [ ] Computer Programming languages are very hard to learn.
- [ ] One needs to be a genius in Maths to become a computer programmer.
- [ ] Only those who study Computer Science Engineering or a related subject in college can learn write computer programs.
- [x] The sequence of instructions (in the form of source code) written in a computer programmin language is called a computer program.

## 1.2. A simple program in Java

Below is an example of a simple program written in **Java** programming language.

Change the text in the below code to make the program print **"Hello Java"** instead of **"Hello C"** and click on **Submit**.

We will learn more about the other aspects of the below code in the later sections.

**Note:** Please don't change the package name.

Explorer

FirstProgr...

```
public class FirstProgram {
    public static void main(String[] args) {
        System.out.println("Hello Java");
    }
}
```

Sample Test Cases

2.1.3. What is Java?

Java programming language was created by a company called Sun Mircosystems (which is now part of Oracle Inc) in the year 1995.

Click here to learn more about the history of Java.

Java can be used to develop virtually any kind of software applications like Games that can run on PCs and Mobiles, Desktop Applications, Web Applications, Embedded Applications (for example applications which can run in a watch or a pen) etc.

We need **JDK** (Java Development Kit) to develop applications using Java.

**JDK** contains a crucial tool known as the **compiler**. It first changes the source code in a [redacted] file into a special code called **bytecode**, by creating a [redacted] file. After that, the **Java Virtual Machine (JVM)** interprets this bytecode and generates the output. The process of converting a [redacted] file to a [redacted] file is known as **compilation**.

JRE (JRE) contains all the required tools and libraries to execute the compiled Java code.

Java became quickly popular because the code written in Java can be run on any operating system (Like, Windows, Linux or Mac) provided it has the JRE (Java Runtime Environment) installed on it.

**Note:** Code written in Java is compiled using the Java compiler which is bundled as part of JDK (Java Development Kit).

[Hint: You can click on the words in blue to know more about them.]

Select all correct statements given below:

- [ ] Java programming language was created in the year 2016.
- [ ] Java was created by Microsoft.
- [x] Compiled Java code can run on any Operating System provided that OS has JRE installed
- [x] JDK is used for developing applications using Java language. It contains tools like the com which is used during development.

PREV RESET SUBMIT NEXT

## 2.1.4. Understanding Structure of a Java Program

A **Java** program consists of one or more classes. A **class** is the top most building block which is given a name, we will learn more about it later.

A class usually consists of data (i.e encapsulates data) and methods which operate on that data.

Among the various classes, the **class** from where the execution starts contains a special method called main().

Below is the general structure followed while writing a **Java** class (i.e a source file with .txt extension):

```
License statement in comments detailing ownership
Package statement
Import statements
Documentation comment
Class or Interface definition
```

Click on the below **Live Demo** button to learn the structure of Java program.

LIVE DEMO

Though we will deal with the above sections in greater details later, below you will find brief notes on the same.

The license statement usually contains details about the copyright and other ownership information.

Java allows user to group classes into a folders called **packages**. A **package** statement includes a

Sample Test Cases +

Explorer

FirstJavaPr...

```
1   package ;
2   //Define a public class named "FirstJavaProgram"
3   public class FirstJavaProgram{
4       public static void main(String[]args){
5           System.out.println("India got its independence in 1947");
6       }
7   }
8
9   //Within the class, define the main method.
10
11
12  /*In the main method Write the print statement
13  in order to print the text "India got its independence in 1947"*/
```

Terminal Test cases

< PREV RESET SUBMIT NEXT >

3.1.1. Java Program Life Cycle

Click on the below **Live Demo** button to learn the steps involved in writing and executing a Java program.

LIVE DEMO

After learning from the Live Demo, select all correct statements given below:

- [ ] Java source code is written and saved in a file with **.class** extension.
- [x] The Java compiler uses the Java source file and generates file with **.class** extension.
- [x] The file with **.class** extension is called as Java class file.
- [x] The Java class file contains the Java bytecode, which is executed by the Java Virtual Mach (JVM).

< PREV RESET SUBMIT NEXT >

### 3.1.2. What is Java Runtime Environment?

JRE stands for Java Runtime Environment.

**Java Runtime Environment** which is commonly referred **JRE**, is the essential piece of software which is required to run any Java program on an operating system.

Note that **JDK** (which stands for Java Development Kit) also contains in it a **JRE** along with other tools like **compilers** which are required during the development phase.

While, **JRE** contains only those tools and libraries which are essential for running a Java program.

Click here to gain a better understanding of **JDK** and **JRE**.

JDK and JRE are available as separate downloads. Meaning, if one downloads and installs a JDK, JRE is automatically installed as it is bundled along with it.

However, when a JRE is separately downloaded an installed on a OS, only the JRE's content is installed, meaning the **compilers**, etc, which are part of JDK will not be present.

When one wants to write a Java program, since one also needs to compile it before running it, JDK is needed.

However, when one gets a compiled Java program in an executable form (from somewhere else on to his OS), a JRE is sufficient to run it.

JRE is a bundle of programs that mainly includes

- JVM (Java VM or Java Virtual Machine)
- Browser plugin for running Java Applets
- Java class libraries (these class libraries contain standard Java classes used during execution) and configuration tools

Select all correct statements given below:

- [ ] JRE is used for compiling Java source code to class files.
- [ ] Browser plugin contains the Java Compiler.
- [x] Java class libraries contain the class files of standard Java classes required for ru Applications or Applets.
- [ ] Java code can be run without a JVM.

Home Learn Anywhere

12209624.st@lpu.in

## 2.1. How to write an Empty Student Class?

This is how a simple class is written in Java (We will learn more about classes in detail in later sections).

Note the spellings of [redacted] and [redacted].
[redacted] and [redacted] are **keywords** in Java and hence are written in **lowercase**.

However the class name, which in this case is [redacted] should always **start** with a **capital letter**.

**Note: Keywords** are words which have special meaning in a programming language. In Java all keywords are in **lowercase**.

There is a small error in the given code. Correct the code in order to ex

Sample Test Cases

Explorer

Student.java

```
package q10737;
public class Student {
    public static void main(String args[]) {
        try {
            Class<?> clazz = Class.forName("q10737.Student"
            System.out.println("class Student found");
        } catch(ClassNotFoundException cnfe) {
            System.out.println("class Student not found
        }
    }
```

**You've been blocked**

You may have browser or remote desktop software running.
whatsapp.exe
Please retry after resolving the issue.

RETRY

Terminal Test cases

2.2. Error in Spelling

dentify the error and correct the code.

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

Student.java | StudentMa...

```
package q10751;
public class Student {
    public static void main(String[] args){
        System.out.println("public class Student found")
        }
}
```

Terminal | Test cases

3.2.3. Error in case

Identify the **error** in the case used in the **name of the class** and correct it.

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

Student.java | StudentMa...

```
package q10754;
public class Student  {
    public static void main(String[] args){
        System.out.println("class Student found");
    }
}
```

Terminal | Test cases

## 3.2.4. Error in the spelling of a keyword

Below code contains an empty class declaration with class name [redacted].

However there is an **error** in the code below, identify and correct the error.

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

Test.java | TestMain.ja...

```
1  package q11602;
2  public class Test {
3      public static void main(String[] args){
4          System.out.println("class Test found");
5      }
6  }
```

Terminal | Test cases

## 1.1. Java program structure

Java program is usually composed of multiple lines. Each line is composed of one or a combination of the below three input elements:

1. Comments
2. Whitespace characters
3. Tokens

A **comment** is a sequence of non-executable characters. There are three types of comments in Java which we will learn later.

In Java, the whitespace characters are:

1. Space – (ASCII SP) produced by pressing spacebar
2. Tab – (ASCII HT) produced by pressing the tab key
3. Form Feed character – (ASCII FF) usually used as the page separator char between lines/paragraphs
4. **Line Terminator chars** (used to separate two lines) – produced by pressing **Enter** key:
   5. Line Feed - (ASCII LF also called NL - New Line) - used in all Unix and Mac OS X systems
   6. Carriage Return - (ASCII CR) – used in MAC OS 9 and below
   7. Carriage Return followed by Line Feed- (ASCII CRLF) – used in Windows systems

All other input elements other than comments and whitespace are called as **tokens**. The tokens are further classified as:

1. Identifiers – Names used to refer or identify are called Identifiers. For example, names of variables, methods and classes are all called **Identifiers**.
2. Keywords – one of the 50 reserved words in Java language like , , , etc are called **Keywords**. These have a special meaning when used as part of the program.
3. Literals – these are the fixed values assigned in a source code. They can be of primitive, String or a null type.
4. Operators – in Java language we have 38 different operators like , , , , , etc.. (which we will learn later)
5. **Separators** – in Java language we have 12 Separators - , , , , , , , , , , , , ,

Select all the correct statements for the below code:

- [x] The token which is the name of the class is called an Identifier.
- [x] The token which is the name of the method is called an Identifier.
- [ ] The tokens and are called Keywords
- [ ] The open brace and the close brace used to open and close the class an respectively are called Operators.

## 4.1.2. Usage of Spaces and Tabs

**Spaces** and **Tabs** are very important for code formatting.

An **empty space** is used as a separator between two tokens. A **tab** is used only to **indent lines**.

Whenever a new block of code starts after the **opening brace** the lines inside that block should be **indented by one level** (one level means by one tab) compared to the line which contains the opening brace.

**Note:** Usually after a opening brace we press the **ENTER** key to go to next line, where a **tab** is automatically inserted for indentation.

Below example shows the usage of spaces and tabs:

```
public class SpacesAndTabsDemo {
	public static void main(String[] args) {
		for (int i = 0; i < 5; i++) {
			System.out.println("i = " + i);
		}
	}
}

 orange borders mark spaces
	 pink borders mark tabs
```

See and type the above example code in the provided Editor and observe the output.

**Note:** Please don't change the package name.

Sample Test Cases

SpacesAn...

```
package ;
public class SpacesAndTabsDemo {
	public static void main(String[] args) {
		for ( int i = 0; i < 5; i++) {
		System.out.println("i = " + i);
		}
	}
}
```

Terminal Test cases

## 1.3. Three types of comments in Java

In Java, there are three types of comments:

1. end-of-line comment : It starts with [redacted] and the content following [redacted] till the end of that line is a comment. It is also called as the single-line comment.
2. traditional comment : It starts with [redacted] and ends with [redacted], the content between [redacted] and [redacted] is the comment. It is also called as multi-line comment.
3. Java Doc comment : It starts with [redacted] and ends with [redacted], the content between [redacted] and [redacted] contains the java documentation information. Click here for more information on Java Doc comment.

Below code example shows all the three types of comments.

```
/**
 * This class has an example code for a sample Java Doc comment
 * @author James
 */
public class TestComments {

    public static void main(String[] args) {

        /* This is an example for traditional comment also called as star-comment or multi-line com

        System.out.println("This is a simple example on java comment lines");

    }//end of the main method

}//end of the class
```

Note that the extra space is marked as a pink block. Also note that the class declaration statement `public class TestComments` should not have space before it.

Sample Test Cases

Explorer

TestAdditi...

```
package q35952;
public class TestAddition {
    public static void main ( String[] args) {
        int num1 = 5;
        int num2 = 10;
        int result = num1 + num2;
        System.out.println("Sum: " + result);
    }
}
```

Terminal  Test cases

4.1.4. Correct the Error

In English language we normally use a Period ( also called Full Stop) to terminate a statement.

Similarly in Java, a semicolon is used to terminate a statement.

For example:

```
int x = 3
int y = x + 4
System.out.println("Holla!")
```

If we miss the semicolon at the end of a statement, it is a syntax error. And while compiling, Java compiler will flag an error.

For example, click **Submit** on the below code without fixing the error and notice the compilation error. And later you can fix the problem and re-submit the code.

**Note:** Please don't change the package name.

Sample Test Cases +

Explorer

PrintHello....

```
1  package q10800;
2  public class PrintHello {
3      public static void main(String[] args) {
4          System.out.println("Hello")
5      }
6  }
```

Terminal Test cases

4.1.5. Correct the Error in Syntax

Identify the error and correct it.

**Note:** Please don't change the package name.

PrintHello....

```java
package q10801;
public class PrintHello {
    public static void main(String[] args) {
        String text1 = "He";
        String text2 = "llo";
        String text3 = text1 + text2;
        String text4 = ", I am learning Java" ;
        String text5 = text3 + text4;
        System.out.println("text5 = " + text5);
    }
}
```

## 4.1.6. Missing Paranthesis

In Java, a method call is followed by an open and close parenthesis.

The arguments that the method accepts are passed in between these parenthesis.

For example in the below code:

```
System.out.println("Good Morning");
```

The argument, which in the above example is the String literal , is passed into the method .

Identify the error in the below code and correct it.

**Note:** Please don't change the package name.

Sample Test Cases +

Explorer

Test.java

```
package ql0802;
public class Test {
    public static void main(String[] args) {
        System.out.println("Hello");
    }
}
```

Terminal Test cases

4.1.7. Correct the Error

Identify the error in the given code and correct it.

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

PrintHello....

```
package q10803;
public class PrintHello {
    public static void main(String[] args) {
        System.out.println("Hello, I am learning Java!");
    }
}
```

Terminal Test cases

Home Learn Anywhere

1.8. Correct the Error

Find the error and correct the code.

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

PrintHello....

```
package q10804;
public class PrintHello {
    public static void main(String[] args) {
        System.out.println("Hello, I am learning Java!");
    }
}
```

Terminal Test cases

## 4.1.9. Understanding usage of braces

In Java, code statements are written in **code blocks**.

A **code block** starts with an open brace and end with a close brace.

For example:

```
public class A
    int x = 2;
    public static void main(String[] args)
        System.out.println("Holla!");
```

Identify and correct the error in the below code.

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

PrintHello....

```
package ql0805;
public class PrintHello {
    public static void main(String[] args) {
        System.out.println("Hello, I am learning Java!");
    }
}
```

Terminal  Test cases

< PREV  RESET  SUBMIT  NEXT >

4.1.10. Correct the Error

Find the error and correct it in the code.

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

PrintHello....

```
package ql0806;
public class PrintHello {
    public static void main(String[] args) {
        System.out.println("Hello, I am learning Java!");
    }
}
```

Terminal  Test cases

PREV  RESET  SUBMIT  NEXT

1.1.11. Correct the Error in Syntax

Identify the error and correct the code given below.

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

PrintHello....

```
package q10807;
public class PrintHello {
    public static void main(String[] args) {
        System.out.println("Hello, I am learning Java!");
    }
}
```

Home Learn Anywhere

4.1.12. Correct the Error in Syntax

Find the errors in the given code and correct it.

**Note:** Please don't change the package name.

Explorer

PrintHello....

```
package ql0808;
public class PrintHello {
    public static void main(String[] args) {
        String text1 = "He";
        String text2 = "llo";
        String text3 = text1 + text2;
        System.out.println("text3 = " + text3);
    }
}
```

Terminal Test cases

## 5.1.1. Introduction to main method

In Java programming language the code execution starts with a special method whose name is [redacted].

This method has a fixed signature (in Java, method signature comprises of method name and the parameter types).

The main method must be marked as [redacted], [redacted] and [redacted]. We will learn more about these in the ensuing sections.

This method also must take a String array as its argument as indicated in the below code:

```
public static void main(String[] args) {

}
```

Note that the method names in Java should always begin with lowercase.
Hence the method [redacted] is written with [redacted] in lowercase.
If the `main` method's signature does not match the syntax as shown above the code will not be executed even if there are no compilation errors.

Write the Java program to print the text "Hello, I am learning Java!" by defining the main class **PrintHello**

**Note:** Please don't change the package name.

**Sample Test Cases**

Explorer

PrintHello....

```
1   package [redacted];
2   public class PrintHello {
3       public static void main(String[] args) {
4           System.out.println("Hello, I am learning Java!");
5       }
6   }
```

Terminal  Test cases

< PREV  RESET  SUBMIT  NEXT >

## 5.1.2. Error in main method syntax

Identify the error in the code and correct it.

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

PrintHello....

```
package q10756;
public class PrintHello {
    public static void main(String[] args) {
        System.out.println("Hello, I am learning Java!");
    }
}
```

Terminal  Test cases

5.1.3. Error in main method syntax

Observe the below code and Identify the error and correct it.

**Note:** Please don't change the package name.

Sample Test Cases

PrintHello....

```
package q10757;
public class PrintHello {
    public static void main(String[] args) {
        System.out.println("Hello, I am learning Java!");
    }
}
```

Terminal Test cases

5.1.4. Error in main method syntax

Please see the below code and correct the errors if present.

**Note:** Please don't change the package name.

Sample Test Cases

PrintHello....

```
package q10758;
public class PrintHello {
    public static void main(String[] args) {
        System.out.println("Hello, I am learning Java!");
    }
}
```

Terminal  Test cases

< PREV  RESET  SUBMIT  NEXT >

5.1.5. Error in main method syntax

Find the errors in the below code and correct it.

**Note:** Please don't change the package name.

Sample Test Cases +

Explorer

PrintHello....

```
package q10759;
public class PrintHello {
    public static void main(String[] args) {
        System.out.println("Hello, I am learning Java!");
    }
}
```

Terminal Test cases

< PREV RESET SUBMIT NEXT >

Error in main method syntax

tify the errors in the code and correct it.

e: Please don't change the package name.

Sample Test Cases

Explorer

PrintHello....

```
package q10760;
public class PrintHello {
    public static void main(String[] args) {
        System.out.println("Hello, I am learning Java!");
    }
}
```

Terminal Test cases

or in main method syntax

t the error in the given code.

Please don't change the package name.

Sample Test Cases

Explorer

PrintHello....

```
package q10785;
public class PrintHello {
    public static void main(String[] args) {
        System.out.println("Hello, I am learning Java!");
    }
}
```

Terminal Test cases

5.1.8. Error in main method syntax

Observe the below code and correct the errors.

**Note :** Please don't change the package name.

Sample Test Cases

Explorer

PrintHello....

```
package q10790;
public class PrintHello {
    public static void main(String[] args) {
        System.out.println("Hello, I am learning Java!");
    }
}
```

Terminal

Test cases

< PREV RESET SUBMIT NEXT >

Home Learn Anywhere

in the missing code

he missing code for the main method.

Please don't change the package name.

mple Test Cases

Explorer

PrintHello....

```
package q10791;
public class PrintHello {
    public static void main(String[] args){
        System.out.println("Hello, I am learning Java!");
    }
}
```

Terminal Test cases

PREV RESET SUBMIT NEXT

## 5.1.10. Introduction to main method

The Java programs can be passed arguments on the command line while executing. These arguments are made available in a String array, which is passed into the [redacted] method.

Click on [redacted] to understand how the arguments passed on command line are available to the [redacted] method.

Consider the following code snippet that demonstrates the usage of command-line arguments:

```
public class CommandLineArgumentsDemo {
    public static void main(String[] args) {
        System.out.println("args.length : " + args.length);
        System.out.println("args[0] : " + args[0]);
        System.out.println("args[1] : " + args[1]);
        System.out.println("args[2] : " + args[2]);
        System.out.println("args[3] : " + args[3]);
    }
}
```

Assume that the program is executed with four command-line arguments.

What is the purpose of the main method in the given code?

Choose the right one from the available options.

- It displays the number of command-line arguments passed to the program.
- It reads input from the user during program execution.
- It performs calculations and manipulations on the command-line arguments.
- It serves as the entry point of the program and executes the code.

< PREV RESET SUBMIT NEXT >

## 5.1.11. Understanding Arguments

04:56 aA Switch to dark theme

The below code demonstrates how command line arguments are passed to a class called `CommandLineArgumentDemo`

```
>> java CommandLineArgumentDemo Red Blue Green
```

In the above code :

1. the **first** argument **Red** can be accessed as `args[0]`
2. the **second** argument **Blue** can be accessed as `args[1]`
3. the **third** argument **Green** can be accessed as `args[2]`

Click on Live Demo to understand how the arguments passed on command line are available to the `main` method.

In the above example, if we want to print the **first argument** the code will be as given below:

```
System.out.println(args[0]);
```

Similarly, if we want to print the **third argument** the code will be as given below:

```
System.out.println(args[2]);
```

Note that the index values passed in `args[index]` always start with `0` (zero) for **first** argument, `1` for **second** argument and so on.

Assume that **three** command line arguments will be passed to the below class with name `CommandLineArgumentDemo`.

Write code in the `main(String[] args)` method to print only the **second** argument.

Sample Test Cases +

Command...

```
package q10793;
public class CommandLineArgumentDemo {
    public static void main(String[] args) {
        //Write the code fragment in the below println(
method to print only the second argument
        System.out.println(args[1]);
    }
}
```

Terminal Test cases

## 5.1.12. Understanding Command line arguments

Assume that **five** command line arguments will be passed to the below class with name `CommandLineArgumentDemo`.

Write code in the `main(String[] args)` method to print only the **fourth** argument.

**Note:** Please don't change the package name.

Sample Test Cases

Command...

```
package q10794;
public class CommandLineArgumentDemo {
    public static void main(String[] args) {
        //Write the code fragment in the below println(
method to print only the fourth argument
        System.out.println(args[3]);
    }
}
```

Terminal  Test cases

## 5.1.13. Understanding Command line arguments

Assume that **five** command line arguments will be passed to the below class with name `CommandLineArgumentDemo`.

Write code in the `main(String[] args)` method to print only the **first** argument.

**Note:** Please don't change the package name.

Sample Test Cases +

Command...

```
package q10795;
public class CommandLineArgumentDemo {
    public static void main(String[] args) {
        //Write the code fragment in the below println( ) method to print only the first argument
        System.out.println(args[0]);
    }
}
```

Terminal Test cases

## 5.1.14. Understanding Command line arguments

Assume that **five** command line arguments will be passed to the below class with name `CommandLineArgumentDemo`.

Write code in the `main(String[] args)` method to print only the **fifth** argument.

**Note:** Please don't change the package name.

Sample Test Cases

Command...

```
package q10796;
public class CommandLineArgumentDemo {
    public static void main(String[] args) {
        //Write the code fragment in the below println( ) method to print only the fifth argument
        System.out.println(args[4]);
    }
}
```

Terminal  Test cases

## 5.2.1. Example for printing Hello

The PrintHello class demonstrates printing text to the console.

You will notice that the line, `System.out.println("Hello, World!");` is responsible for printing the text.

This line is written inside the `main` method.
Please note that in Java, **String literals** (i.e. text) must be enclosed in **double quotes**.
In our case you will notice that the String literal `Hello, World!` is enclosed in double quotes.

Click on Live Demo to understand basics of printing.

See and modify the code in the provided Editor to print the statement as per the test case.

**Note:** Please don't change the package name.

Sample Test Cases

PrintHello....

```
package q35959;
public class PrintHello {
    public static void main(String[] arg) {
        System.out.println("Java is high-level programming language..!");
    }
}
```

Terminal  Test cases

Correct the Error

00:16

tify the error in the below code and correct it.

e: Please don't change the package name.

Explorer

PrintHello....

```
1  package q10762;
2  public class PrintHello {
3      public static void main(String[] args) {
4          System.out.println("Hello");
5      }
6  }
```

5.2.3. Correct the Error

00:07

Find the error in the given code and correct it.

**Note:** Please don't change the package name.

Explorer

PrintHello....

```
package q10763;
public class PrintHello {
    public static void main(String[] args) {
        System.out.println("Hello");
    }
}
```

5.2.4. Correct the Error

00:13

Identify the error in the given code and correct it.

**Note:** Please don't change the package name.

Sample Test Cases

+

Explorer

PrintHello....

```
1  package q10764;
2  public class PrintHello {
3      public static void main(String[] args) {
4          System.out.println("Hello");
5      }
6  }
```

## 5.2.5. Understanding the difference between println and print

03:00

Among the various print methods in Java (about which we will learn later),
System.out.`println` and
System.out.`print` are the most commonly used ones.

As the name suggests `println` method prints the text passed to it and also prints a new line char (`\n`) at the end.
However, the `print` method prints only the text passed, without appending the new line char as in the case of `println`.

As programmers we mostly use the `println` method when we want to see the output on separate lines.

However, there may be situations when we want to produce output in chunks and also want all the output to be on the same line. In such situations we have to use `print`.

Click on Live Demo to understand the difference between **print** and **println** methods.

In the code given, use the appropriate print methods such that the below output is produced.

```
Ganga Nile
Amazon
END
```

Note that we are printing a space `" "` between `text1` and `text2` so that in the output also there is a space between **Ganga** and **Nile**.

**Note:** Please don't change the package name.

Sample Test Cases

+

Explorer

PrintMe.java

```java
package q10765;
public class PrintMe {
    public static void main(String[] args) {
        String text1 = "Ganga";
        String text2 = "Nile";
        String text3 = "Amazon";
        System.out.print (text1);
        System.out.print (" ");
        System.out.println  (text2);
        System.out.println (text3);
        System.out.println("END");

    }
}
```

Terminal  Test cases

## 5.2.6. Correct the Error

In Java, a String is enclosed in **double quotes** and not in single quotes. We will learn more about Strings in the ensuing sections. Identify the error and correct the code.

**Note:** Please don't change the package name.

PrintHello....

```
package q10766;
public class PrintHello {
    public static void main(String[] args) {
        System.out.println("Hello, I am learning Java!");
    }
}
```

## 6.1.1. Understanding Number System

02:37

In computers we normally use **four** different **numbering systems** - **Decimal**, **Binary**, **Octal** and **Hexadecimal**.

**Decimal** is a number system which we humans normally use in our day-to-day transactions with currency, with counting etc. In this system we use the symbols 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 to denote various numbers.

In a **Binary** number system we use `0's` (zeros) and `1's` (ones) as the only symbols to represent numbers of all magnitudes (sizes). For example, a normal decimal number `3` (three), will be represented in a binary as `11`. We will learn more about it in the later sections.

Binary system is mostly using in computers and other computing devices.

A number in a particular base is written as $(\text{Number})_{\text{base of number}}$, for example $(34)_{10}$ is a **decimal** number (Thirty Four) and $(11)_2$ is a **binary** number `11` (we will read it as One One and not Eleven) which actual represents a decimal number whose value is **3**.

Since we normally use the decimal number system the decimal number $(124)_{10}$ is simply written as **124**. However, if we want to represent a binary **One Zero One**, we will write it as $101_2$.

Similarly we have **octal** number system which uses **8** as the base. It is usually used in digital displays and in representing file permissions under UNIX/Linux operating systems.

**Hexadecimal** or **Hex** is a number system that uses **16** as the base to represent numbers. We will learn more about octal and hexadecimal in the later sections.

Select all the correct statements given below.

- [x] A number system that uses only two digits, 0 and 1 is called the **Binary** number system.
- [ ] The ten decimal digits are from **1** to **10**.
- [x] The two symbols **0** and **1** are known as bits in a **binary** system.
- [x] An **octal** system base is **8**.

## 6.1.2. What is Decimal System?

The numbers that we use regularly in our daily life, like: 0, 1, 2, 3,...2333, 99999 etc belong to Decimal System. It is also called a `base-10` system.

It is called `base-10` because it uses 10 unique digits from **0** to **9** to represent any number in this number system.

A base (also called radix) is the number of unique digits or symbols (including `0`) used to represent a given number.

In **Decimal System** (which is base-10) a total of 10 digits (0, 1, 2, 3, 4, 5, 6, 7, 8 and 9) are used to represent a number of any size (magnitude).

For example, **Three Hundred and Twenty One** is represented as 321, where

```
321 = (3 * 10²) + (2 * 10¹) + (1 * 10⁰)
321 = (3 * 100) + (2 * 10) + (1 * 1)
      [100's]     [10's]     [units]
```

Similarly fractions are represented with the base (10) being raised to a negative power.

Select all the correct statements given below.

- [ ] In Decimal System, $102 = (1 * 1^{100}) + (0 * 1^{10}) + (2 * 1^{1})$
- [x] In Decimal System, $102 = (1 * 10^{2}) + (0 * 10^{1}) + (2 * 10^{0})$
- [ ] In Decimal System, $0 = (1 * 1^{0})$
- [x] In Decimal System, $0 = (0 * 10^{0})$

## 6.2.1. What is Binary System?

The numbering system which uses `base-2` is called Binary System. In Binary System (base-2) a total of 2 digits (0 and 1) are used to represent a number of any size (magnitude).

For example, **Zero** is represented as `0`, where
$0 = (0 * 2^0) = (0 * 1)$

Lets try representing **One** (1):
$1 = (1 * 2^0) = (1 * 1)$

Lets try representing **Two** (2), since 0 or 1 are the only digits we can use to represent 2, let us divide 2 by 2 and write down [quotient][reminder], i.e.: `[1][0]`
$2 = (1 * 2^1) + (0 * 2^0) = (2) + (0)$

Lets try representing **Three** (3), since 0 or 1 are the only digits we can use to represent 3, let us divide 3 by 2 and write down [quotient][reminder], i.e.: `[1][1]`
$3 = (1 * 2^1) + (1 * 2^0) = (2) + (1)$

Lets try representing **Four** (4), since only 0 and 1 can be used, let us divide 4 by 2 and write down [quotient][reminder], i.e.: [2] `[0]`, by repeating the above logic for 2 (whose value we already know as `[1][0]`) we get `[1][0][0]`
$4 = (1 * 2^2) + (0 * 2^1) + (0 * 2^0)$
$4 = (4) + (0) + (0)$

Lets try representing **Fourteen** (14), since only 0 and 1 should be used, let us divide 14 by 2 and write down [quotient][reminder], i.e.: [7] `[0]`, by repeating the above logic for 7 (7 = [3] `[1]`, and 3 = `[1][1]`) we finally get `[1][1][1][0]`
$14 = (1 * 2^3) + (1 * 2^2) + (1 * 2^1) + (0 * 2^0)$
$14 = (8) + (4) + (2) + (0)$

Lets try representing **Hundred and Fourteen** (114), let us divide 114 by 2 and write down [quotient]

- [ ] In Binary System, decimal 10 is represented as $(1 * 10^1)$
- [ ] In Binary System, decimal 10 = binary `10` [i.e. one and zero in binary]
- [x] In Binary System, decimal 10 = binary `1010`
- [x] In Binary System, decimal 100 = binary `1100100`
- [ ] In Binary System, decimal 100 = binary `100100`

## 6.2.2. Usage of Binary System

01:31

The **binary number system** is used both in mathematics and digital electronics.

The **binary numeral system** or **base-2 numeral system** represents numeric values using only two symbols - `0` (zero) and `1` (one).

Computers have circuits (logic gates) which can either be in `off` or `on` state. These two states are represented by `0` (zero) and `1` (one).

It is for this reason computation in computers is performed using a binary number system (base-2) where all numbers are represented using `0's` and `1's`.

Each **binary digit** i.e., a single **0** (zero) or **1** (one) is called a **bit**. A collection of **8** such bits is called a **Byte**.

The way we use grams and kilograms to measure, in computer terminology we have different names given to multiples of $2^{10}$ (i.e., **1024** times existing value) .

For example:

```
1 byte     = 8 bits
1 kilobyte = 1024 bytes
1 megabyte = 1024 kilobytes
1 gigabyte = 1024 megabytes
1 terabyte = 1024 gigabytes
1 petabyte = 1024 terabytes
```

In a computer when we type some text, store images, music, videos or any type of data, we should remember that they all are eventually stored in binary format on the disk.

Select all the correct statements given below.

- [x] In **binary system** the base is **2**.
- [ ] A **byte** is composed of **10** bits.
- [x] 1MB (megabyte) = 8388608 bits.
- [ ] A **decimal** number cannot be represented as a **binary** number.

6.2.3. Which statements are correct when using Binary System

00:40

Select all the correct answers

- [ ] In Binary System, decimal 3 is represented as `0b111`
- [x] In Binary System, decimal 3 is represented as `0b11`
- [ ] In Binary System, decimal 3 is represented as `0b3`
- [ ] In Binary System we cannot represent 3, as 3 is greater than 2

## 6.3.1. What is Octal System?

02:11

The numbering system which uses `base-8` is called Octal System. A base (also called radix) is the number of unique digits or symbols (including `0`) used to represent a given number. In Octal System (base-8) a total of 8 digits (0, 1, 2, 3, 4, 5, 6 and 7) are used to represent a number of any size (magnitude).

For example, **Zero** is represented as `0`, where

0 = ( `0` * $8^0$) = ( `0` * 1)

Similarly 1, 2 ...up to 7 **One** (1):

1 = ( `1` * $8^0$) = ( `1` * 1)

2 = ( `2` * $8^0$) = ( `2` * 1)

...

7 = ( `7` * $8^0$) = ( `7` * 1)

Now, lets try representing **Twenty One** (21), since 0 to 7 are the only digits we can use to represent 21, let us divide 21 by 8 and write down [quotient][reminder], i.e.: `[2][5]`

21 = ( `2` * $8^1$) + ( `5` * $8^0$) = (16) + (5)

Similarly, lets try representing **Four Hundred and Twenty One** (421), let us divide 421 by 8 and write down [quotient][reminder], i.e.: [52] `[5]` (further dividing 52 by 8 we get `[6][4]`), which is `[6][4][5]`

421 = ( `6` * $8^2$) + ( `4` * $8^1$ + ( `5` * $8^0$) = (384) + (32) + (5)

Click on Live Demo to understand **Decimal** to **Octal** conversion.

**In Java, Octal numerals are prefixed with a leading `0` (zero).** For example, to store an octal value of seven into a variable numberSeven, we write

```
int numberSeven = 07;
```

Similarly, if we want to store a octal representation of decimal number 9 into a variable numberNine, we write

```
int numberNine = 011;
```

- [ ] In Octal System the base is 10
- [ ] In Octal System, Decimal value of `10` is represented as `21`
- [x] In Octal System, Decimal `8` is represented as `10`
- [x] In Octal System, Decimal of 10 is written as `012`

## 6.3.2. Conversion of an Octal number into Binary and vice versa

05:00

The numbering system which uses `base-2` is called binary, while the numbering system which uses `base-8` is called an octal.

In **binary system** (base-2) a total of 2 digits (0 and 1) are used to represent a number of any size (magnitude), where as in **octal system** (base-8) a total of 8 digits (0, 1, 2, 3, 4, 5, 6 and 7) are used to represent a number of any size (magnitude).

The highest digit in the **octal system** is $(7)_8$. The number $(7)_8$ in binary is represented as $(111)_2$. You will notice that we are using three binary digits (bits) to represent the highest octal digit.

In **octal** to **binary** conversion, we will have to use **three bits** to represent each **octal digit**.

The following table shows the conversion of each **octal digit** into their corresponding **binary digits**.

| **Octal** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Binary** | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |

For example, an octal number `0246` is converted into binary as

```
Octal Number  ->   2   4   6
Binary Number ->  010 100 110
```

Hence, `0246` is $(010100110)_2$.

Click on Live Demo to understand **octal** to **binary** conversion.

Similarly while converting a **binary** number into an **octal**, we first divide the **binary** number into

- [ ] $(369)_8$ is an octal number.
- [x] Each octal digit is represented using three bits.
- [x] Binary number $(10101010)_2$ is equivalent to an octal number $(252)_8$.
- [x] An octal number $(364)_8$ is equivalent of binary number $(011110100)_2$.

## 6.4.1. What is Hexadecimal System?

The numbering system which uses `base-16` is called Hexadecimal System or simply Hex. A base (also called radix) is the number of unique digits or symbols (including `0`) used to represent a given number. In Hexadecimal System (base-16) a total of 16 symbols are used.

Digits `0` (zero) to `9` (Nine) are used to represent values from `0` to `9` respectively and alphabets `A`, `B`, `C`, `D`, `E` and `F` (or `a`, `b`, `c`, `d`, `e` and `f`) are used to represent values from `10` to `15` respectively.

**In Java and many other programming languages `0x` is used as a prefix to denote a hexadecimal representation.**

For example, in hexadecimal the value of **Zero** is represented as `0x0`, where

0 = ( `0` * $16^0$) = ( `0` * 1)

Similarly `1`, `2` ...up to `9`:

1 = ( `1` * $16^0$) = ( `1` * 1)

2 = ( `2` * $16^0$) = ( `2` * 1)

...

9 = ( `9` * $16^0$) = ( `9` * 1)

10 = `A` = ( `10` * $16^0$) = ( `10` * 1)

15 = `F` = ( `15` * $16^0$) = ( `15` * 1)

Now, lets try representing **Twenty One** (21), remember we can use 0 to 9 and the alphabets A to F to represent 21, let us divide 21 by 16 and write down [quotient][reminder], i.e.: `[1][5]`

21 = `0x15` = ( `1` * $16^1$) + ( `5` * $16^0$) = (16) + (5)

Similarly, lets try representing **One Hundred and Sixty** (160), let us divide 160 by 16 and write down [quotient][reminder], i.e.: [10][0], [A][0] (since 10 is represented by A)

160 = `0xA0` = ( `10` * $16^1$ + ( `0` * $16^0$) = (160) + (0)

Similarly, lets try representing **Three Thousand Four Hundred and Sixty** (3460). Let us divide 3460 by 16 and write down [quotient][reminder], i.e.: [216] `[4]`, (216 divided by 16 is [13] `[8]`, which is represented as `[D][8]`). So 3460 is represented as `0xD84`

3460 = `0xD84` = ( `13` * $16^2$) + ( `8` * $16^1$) + ( `4` * $16^0$) = ( `13` * 256) + ( `8` * 16) + ( `4` * 1) = (3328) + (128) + (4)

- [ ] In Hexadecimal System, a base of `10` is used
- [x] Decimal `100` is equal to `0x64` in Hex
- [x] Hex value of 0xCAFE is equal to Decimal 51966
- [x] `0x0001` in Hex is equal to `0X1` in Hex and is equal to 1 in Decimal

## 6.4.2. Conversion of Hexadecimal number into Octal and vice versa

00:46

To convert a **hex** number to an **octal**, we will first convert the **hex** number to **binary** and then to **octal**.

Below are the detailed steps followed for the above mentioned process:

1. Convert each **hex digit** to its corresponding **four binary digits** (bits) from right to left.
2. Combine all the **binary digits**.
3. Separate the **binary digits** into groups of **3 bits** each from right to left.
4. Ensure the left most group has **3 bits** by prefixing necessary zeros.
5. Find the **octal** equivalent of each group and combine the resultant digits together.

For example, hexadecimal number `0x5AF6` is converted into binary as

```
Hex Number    ->    5    A    F    6
Binary Number ->  0101 1010 1111 0110
Binary Number ->  0101101011110110
Binary Number ->   0 101 101 011 110 110
Binary Number ->  000 101 101 011 110 110 // After prefixing zeros in the left most group
Octal Number  ->   0   5   5   3   6   6
Octal Number  ->   055366
```

Hence, the **octal** equivalent of the given hexadecimal number is $(55366)_8$

Click on Live Demo to understand **hexadecimal** to **octal** conversion.

Similarly an **octal** can be converted into a **hex** following the below steps:

1. Convert each **octal digit** to its corresponding **three binary digits** (bits) from right to left.
2. Combine all the **binary digits**.
3. Separate the **binary digits** into groups of **4 bits** each from right to left.
4. Ensure the left most group has **4 bits** by prefixing necessary zeros.
5. Find the **hex** equivalent of each group and combine the resultant digits together.

For example, an octal number `0246` is converted into hexadecimal as

- [x] $(ABCD)_{16}$ is a hexadecimal number.
- [x] Each hexadecimal digit is represented as `four bits` binary number.
- [x] Octal number $(157)_8$ is equivalent to the hexadecimal number $(6F)_{16}$.
- [ ] Hexadecimal number $(369)_{16}$ is equivalent of octal number $(1561)_8$.

< Prev | Reset | Submit | Next >

## 6.4.3. Conversion of Hexadecimal number into Binary and vice versa

05:16

The numbering system which uses `base-16` is called **hexadecimal system** (or simply hex).

In **binary** (base-2) a total of 2 digits (0 and 1) are used to represent a number of any size (magnitude) and in **octal** (base-8) a total of 8 digits (0 to 7) are used.

Similarly in **hexadecimal system** (base-16), a total of 16 digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F) are used to represent a number of any size (magnitude).

The highest digit in hex is $(F)_{16}$. The number $(F)_{16}$ in binary is represented as $(1111)_2$. You will notice that we are using **four binary digits** (bits) to represent the highest hexadecimal digit.

In **hex** to **binary** conversion, we will have to use **four bits** to represent each **hex digit**.

The following table shows the conversion of each **hex digit** into their corresponding **binary digits**.

| Hexadecimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bina | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

- [ ] $(F3G9)_{16}$ is a hexadecimal number.
- [ ] Each hexadecimal digit is represented as `three bit` binary number.
- [x] Binary number $(10101110)_2$ is equivalent to the hexadecimal number $(AE)_{16}$.
- [x] Hexadecimal number $(369)_{16}$ is equivalent of binary number $(1101101001)_2$.

## 7.1.1. What is a Data Type?

Computer programming is all about processing information. This information could be of different data formats(like Image, Audio, Video, Text etc). When it comes to computer programming, irrespective of the format, the data contained in those formats is represented in binary form, i.e. groups of Bits called as Bytes. (.i.e which essentially contain a combination of `0's` and `1's`).

For easier and efficient processing, data is classified into different types (**Data Types**) such as **byte**, **char**, **boolean**, **int**, **float**, **String**, **Array**, **Enum** etc. These data types can be grouped as **Primitive** and **Composite (or Reference Type)**.

As their names suggest, primitives form the primary building blocks(these usually have built-in support in the language) and composites are usually made up of one or more primitive type.

Each **Data Type** determines

1. the meaning of the data that it represents
2. the possible values it can take
3. the way or format in which the value of this type is stored
4. the possible operations that can be performed with it

Java is a statically typed language, which means that the type imposes some constraints on the value it can hold and the operations that can be performed, and this is verified by the compiler during the compilation.

In Java, we need to declare the type of a variable before being able to use it. For example if we want to store a value `25` in a variable with name `age` we cannot directly say

```
age = 25;
```

We have to first declare `age` to be of type `int` like:

`int age;` and then we can say

`age = 25;` or we can also declare and initialize in a single step, like below:

```
int age = 25;
```

We will learn more about various data types in the ensuing exercises.

- [ ] The contents of PDF and Text files are not stored as bits in a Hard Disk.
- [x] If a text file contains only a decimal value 12, then the value 12 is stored as 1100 in its binary form.

## 7.2.1. Understanding Numeric Data Types in Java

00:19

In Java, there are different numeric data types to represent whole numbers(integral) and fractions(floating-point) of different sizes (magnitudes).

The numeric types are the integral types and the floating-point types.

The integral types are byte, short, int, and long, whose values are 8-bit, 16-bit, 32-bit and 64-bit signed two's-complement integers, respectively, and char, whose values are 16-bit unsigned integers representing UTF-16 code units (#3.1).

The floating-point types are float, whose values include the 32-bit IEEE 754 floating-point numbers, and double, whose values include the 64-bit IEEE 754 floating-point numbers.

*[Source: Java Language Specification, Primitive Types and Values]*

Select all the correct statements given below.

- [ ] `int`, `short`, `byte` and `long` are floating-point types
- [x] `char` values are 16-bit integers
- [x] `double` can hold a value larger than the maximum `float` value
- [x] `long` can hold a value greater than the maximum `int` value

## 7.3.1. What is an Integer Data Type?

01:32

In Java, the `int` data type denotes a 32-bit signed (two's-complement) integer, which can hold any value between - 2,147,483,648 ($-2^{31}$) and 2,147,483,647 ($2^{31}-1$).

The primitive type `int` has a corresponding wrapper class called `Integer`. Both int and Integer can be used interchangeably. Which means, we can say:

```
int num1 = 3;
Integer num2 = 4;
Integer total = num1 + num2;
int valueOfNum2 = total - num1;
```

The fixed values `3`, `4` assigned to `num1` and `num2` respectively are called integer literals.
The default value of a primitive `int` is `0`, when not initialized. However, the default value of a reference of type Integer is `null`, when not initialized.
In Java 8 and later, int can also represent unsigned 32-bit integers, with a min value of 0 and a max value of $2^{32}-1$. To use unsigned integers, use `Integer` class.
We will learn more about the wrapper class `Integer` later.

Select all the correct statements given below.

- [ ] We can use an `int` data type to store a String
- [ ] We can use an `int` data type to store a Boolean value of true or false.
- [ ] A `int` literal value can be enclosed in single or double quotes.
- [x] An `int` literal value can be a negative or a positive number with the range of $-2^{31}$ and $2^{31}-1$.

7.3.2. Working with int and char

Observe the code in the given editor

Note how an `int` data type can be assigned values from **binary**, **octal** and **hex** and **character literals**.

In case the value happens to be a **char** literal, their **Unicode** value is assigned.
Note: The initial 256 characters of the ASCII character set are the same in **Unicode** character set.
Correct the given code in order to print the Unicode value of the given characters(**A** and **Z**).

**Note:** Please don't change the package name.

Sample Test Cases

IntegerDe...

```java
package q10770;
public class IntegerDemo {
    public static void main(String args[]) {
        int decimalOne = 1;
        System.out.println("decimalOne = " + decimalOne);
        int binaryThree = 0b11;
        System.out.println("binaryThree = " + binaryThree);
        int octalEight = 010;
        System.out.println("octalEight = " + octalEight);
        int hexTen = 0xA;
        System.out.println("hexTen = " + hexTen);
        int unicodeValueOfA = 65;
        System.out.println("unicodeValue Of A = " + unicodeValueOfA);
        int unicodeValueOfZ = 90;
        System.out.println("unicodeValue Of Z = " + unicodeValueOfZ);
    }
}
```

Terminal Test cases

## 7.3.3. Usages of the Integer wrapper class

The Integer class wraps a primitive **int** value.

This class provides constants and methods that are useful while working with int(s).

For example it has a method Integer.parseInt(String s, int radix) which can be used to convert a string value representing a binary, octal, hex or a decimal integer within the valid range of a Integer

```
int x = Integer.parseInt("B", 16);
//parses the string literal "B" using base-16 and returns decimal 11 as a int

int y = Integer.parseInt("747");
//parses the string literal "747" using base-10 and returns decimal 747 as an int
//Note that we need not pass the radix when we want to convert to decimal (or base-10)
```

In the below code, the main method of **CalculateDifference** will be passed **two arguments**. Both the values can be any number between Integer.MIN_VALUE to Integer.MAX_VALUE.

Complete the below code so that it produces the correct output.

**Note:** You can assume that the first string value passed in `args[0]` and the second string value passed in `args[1]` will always contain a decimal number (number represented in base-10, like 2, 3, 4 etc.). It means that you can directly use the Integer.parseInt(text) method without passing any radix for the second parameter.

[Hint: You can convert **arg[0]** to `int` by the statement `int value = Integer.parseInt(args[0]);`]

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

CalculateD...

```
package q10771;
public class CalculateDifference {
    public static void main(String[] args) {
        // In the below two lines write code to convert
        // the string value passed in args[0] and args[1] into int values using Integer.parseInt method
        // convert the value in args[0] to int and store in firstValue
        // convert the value in args[1] to int and store in secondValue
        int firstValue = Integer.parseInt(args[0]);
        int secondValue = Integer.parseInt(args[1]);
        System.out.println("The difference of args[0] and args[1] is : " + (firstValue - secondValue));
    }
}
```

Terminal Test cases

## 7.4.1. What is a Long Data Type?

In Java, the `long` data type denotes a 64-bit signed (two's-complement) integer, which can hold any value between -9,223,372,036,854,775,808 ($-2^{63}$) and 9,223,372,036,854,775,807 ($2^{63}-1$).

**`long` is used whenever an `int` data type is not sufficient to represent a number.**
The primitive type `long` has a corresponding wrapper class called `Long`. Both long and Long can be used interchangeably. Which means, we can say:

```
long num1 = 22222222223L;
Long num2 = 23L;
Long total = num1 + num2;
long valueOfNum2 = total - num1;
```

The fixed values `22222222223L`, `23L` assigned to `num1` and `num2` respectively are called long literals.
As you can notice, the long literal values are suffixed with char 'L' (lowercase of 'L' can also be used, however the uppercase is more readable)
The default value of a primitive `long` is `0L`, when not initialized. However, the default value of a reference of type Long is `null`, when not initialized.
In Java 8 and later, **long** can also represent **unsigned 64-bit integers**, with a min value of 0 and a max value of $2^{64}-1$.
To use unsigned long integers, use the `Long` class.
We will learn more about the wrapper class `Long` later.

Select all the correct statements given below.

- [ ] The max value of a `long` is twice the max value of an `int`
- [ ] A `long` can also represent decimal (fractional) numbers.
- [ ] A `long` literal value can be enclosed in single or double quotes.
- [x] A `long` literal value can be a negative or a positive number within the range of $-2^{63}$ and $2^{63}-1$

## 7.4.2. Usages of the Long wrapper class

The Long class wraps a primitive long value.
This class provides constants and methods that are useful while working with large numbers. For example it has a method Long.parseLong(String s, int radix) which can be used to convert a string value representing a binary, octal, hex or a decimal integer within the valid range of a long.

```
long x = Long.parseLong("C", 16);
//parses the string literal "C" using base-16 and returns decimal 12 as a long

long y = Long.parseLong("380");
//parses the string literal "380" using base-10 and returns decimal 380 as a long
```

In the below code, the main method of CalculateDifference will be passed two arguments. Both the values can be any number between Long.MIN_VALUE to Long.MAX_VALUE .
You can assume that the string values passed in args[0] and args[1] will always contain numbers represented in base-10.
Complete the below code so that it produces the correct output.

**Note:** Please don't change the package name.

Sample Test Cases

CalculateD...

```
package q10772;
public class CalculateDifference {
    public static void main(String[] args) {
        long firstValue = Long.parseLong(args[0]);
        long secondValue = Long.parseLong(args[1]);

        System.out.println("The difference of args[0] and args[1] is : " + (firstValue - secondValue));
    }
}
```

Terminal  Test cases

## 7.5.1. What is a Short Data Type?

In Java, the `short` data type denotes a 16-bit signed (two's-complement) integer, which can hold any value between - 32,768 ($-2^{15}$) and 32,767 ($2^{15}-1$).

The primitive type `short` has a corresponding wrapper class called `Short`. Both short and Short can be used interchangeably. Which means, we can say:

```
short num1 = 3;
Short num2 = 4;
Short total = (short)(num1 + num2);
short valueOfNum2 = (short)(total - num1);
```

The default value of a primitive `short` is `0`, when not initialized. However, the default value of a reference of type Short is `null`, when not initialized.
We normally use `int` instead of `short` to hold integer values. However, when using large arrays if we know that the values stored will be within the range of a `short`, using an array of type `short` may help saving memory.

We will learn more about the wrapper class `Short` later.

Select all the correct statements given below.

- [ ] We can use an `short` data type to store a String
- [ ] We can use an `short` data type to store a Boolean value of true or false.
- [ ] A `short` literal value can be enclosed in single or double quotes.
- [x] A `short` literal value can be a negative or a positive number with the range of $-2^{15}$ and $2^{15}-1$.

## 7.5.2. Learn how type widening works when using short

The below statement declares two `short` variables x and y initialized to 3 and 4 respectively.

```
short x = 3, y = 4;
```

However, when we try to perform an addition operation, such as given below, we get a compilation error : `Type mismatch: cannot convert from int to short`.

```
short sum = x + y;
```

In Java, when a binary operator (like, *, / ,%, +, etc. ) is being used, if at least one operand is of type `double`, `float` or `long` then the operation is carried out by automatically widening the other operand also to that former's type (if it is not already of the former's type) and the result is also of the former's type.
And for all other numeric types (like `byte`, `short`, `char`) the operands are automatically widened to `int` and the result is also an `int`.

Since x and y are of type short, they both are automatically widened to `int`, and the resultant int is being assigned to sum which is of type short.
The compile-time error occurs while assigning the result which happens to be an int. To fix it, we will have to explicitly type cast the result to a short, as shown below.

```
short sum = (short)(x + y);
```

Applying the above understanding identify and correct the error so that the correct output is produced.

**Note:** Please don't change the package name.

Sample Test Cases

ShortDem...

```
package q10774;
public class ShortDemo {
    public static void main(String[] args) {
        short mangos = 3;
        short bananas = 4;
        short fruits = (short)(mangos + bananas);
        System.out.println("fruits = " + fruits);// should print 7
    }
}
```

Terminal  Test cases

## 7.5.3. Usages of the Short wrapper class

The Short class wraps a primitive short value.
This class provides constants and methods that are useful while working with shorts. For example it has a method Short.parseShort(String s, int radix) which can be used to convert a string value representing a binary, octal, hex or a decimal integer within the valid range of a short

```
short x = Short.parseShort("A", 16);
//parses the string literal "A" using base-16 and returns decimal 10 as a short

short y = Short.parseShort("3203");
//parses the string literal "3203" using base-10 (by default) and returns decimal 3203 as a short
```

In the below code, the `main` method in the class `GuessShort` will be passed one argument in `args`.
The argument can be any number between -32,768 to 32,767.
You can assume that the string value passed in `args[0]` will always contain a number represented in base-10.
Here the program will check the given number is in the range of byte or not.
Complete the below code so that it produces the correct output.

**Note:** Please don't change the package name.

Sample Test Cases +

GuessShor...

```
package q10775;
public class GuessShort {
    public static void main(String[] args) {
        //in the below line write code to convert
        //the string value passed in args[0] into a short
        //Hint: Since args[0] is going to be a String
        // you can use Short.parseShort(args[0]); to convert
        // the value in args[0] into a short
        short shortValue = Short.parseShort(args[0]) ;

        if (Byte.MIN_VALUE <= shortValue && shortValue <= Byte.MAX_VALUE) {
            System.out.println("The short value " + shortValue + " is in the range of a byte ");
        } else {
            System.out.println("The short value in args[0] is : " + shortValue);
        }
    }
}
```

Terminal Test cases

## 7.5.4. Identify and correct the error

02:01

The class `HolidaysCalculator` is passed two string arguments.
The first argument ( `args[0]` ) contains the count of **national holidays** declared in that country.

The second argument ( `args[1]` ) contains the count of **company specific holidays**.

The below code is supposed to print the **total number** of holidays as the **sum** of both **national** and **company** holidays.

Fill in the missing code to :

1. convert the string passed in `args[0]` to short and assign to nationalHolidays
2. convert the string passed in `args[1]` to short and assign to companyHolidays
   [**Hint:** Remember to use type-casting where required.]

**Note:** Please don't change the package name.

Sample Test Cases +

Explorer

HolidaysC...

Submit

```
package q10776;
public class HolidaysCalculator {
    public static void main(String[] args) {
        short nationalHolidays = Short.parseShort (args[0]);
        short companyHolidays = Short.parseShort (args[1]);
        short totalHolidays = (short) (nationalHolidays + companyHolidays);
        System.out.println("totalHolidays = " + totalHolidays);
    }
}
```

Terminal Test cases

## 8.1.1. What is a Byte Data Type?

15:04

In Java, the `byte` data type denotes a 8-bit signed (two's-complement) integer, which can hold any value between - 128 ($-2^7$) and 127 ($2^7-1$).

The primitive type `byte` has a corresponding wrapper class called `Byte`. Both byte and Byte can be used interchangeably. Which means, we can say:

```
byte num1 = 3;
Byte num2 = 4;
Byte total = (byte)(num1 + num2);
byte valueOfNum2 = (byte)(total - num1);
```

The default value of a primitive `byte` is `0`, when not initialized. However, the default value of a reference of type Byte is `null`, when not initialized.
We normally use `int` instead of `byte` to hold integer values. However (like in the case of short), when using large arrays if we know that the values stored will be within the range of a `byte`, using a `byte` array may help saving memory.
A `byte` data type can also be used to indicate to a programmer that the value it will hold is small. More commonly they are used while manipulating raw data using byte arrays.

We will learn more about the wrapper class `Byte` later.

Select all the correct statements given below.

- [ ] We can directly assign bits to a byte like **byte myByte3 = "11";**
- [ ] A `byte` data type can store only positive integer values.
- [x] The max value a byte can be assigned is `0b1111111`
- [ ] A byte occupies $2^8$ bits in memory

## 8.1.2. Learn how type widening works when using byte

00:29

The below statement declares two `byte` variables x and y initialized to 3 and 4 respectively.

```
byte x = 3, y = 4;
```

However, when we try to perform an addition operation, such as given below, we get a compilation error: `Type mismatch: cannot convert from int to byte`.

```
byte sum = x + y;
```

In Java, when a binary operator (like, *,/, %, +, etc. ) is being used, if at least one operand is of type `double` , or `float` or `long` then the operation is carried out by automatically widening the other operand also to that former's type (if it is not already of the former's type) and the result is also of the former's type.
And for all other numeric types, the operands are automatically widened to `int` and the result is also an int.

Since x and y are of type byte, they both are automatically widened to int, and the resultant int is being assigned to sum which is of type byte.
The compile-time error occurs while assigning the result which happens to be an int. To fix it, we will have to explicitly typecast the result to a byte, as shown below.

```
byte sum = (byte)(x + y);
```

Applying the above understanding identify and correct the error so that the correct output is produced.

**Note:** Please don't change the package name.

Sample Test Cases

+

Explorer

ByteDemo...

Submit

```
package q10778;
public class ByteDemo {
    public static void main(String[] args) {
        byte b1 = 4;
        byte b2 = 20;
        byte sum = (byte) (b1 + b2);
        System.out.println("sum = " + sum);// should print 24
    }
}
```

Terminal

Test cases

Plots

## 8.1.3. Usages of the Byte wrapper class

00:30

The Byte class wraps a primitive byte value.
This class provides constants and methods that are useful while working with bytes. For example it has a method Byte.parseByte(String s, int radix) which can be used to convert a string value representing a binary, octal, hex or a decimal integer within the valid range of a byte

```
byte x = Byte.parseByte("111", 2);
//parses the string literal "111" using base-2 and returns decimal 7

byte y = Byte.parseByte("23", 10);
//parses the string literal "23" using base-10 (by default) and returns decimal 23
```

In the below code, the main method of GuessDataType will be passed an argument. The argument can be any number between 0 to 127.
You can assume that the string value passed in args[0] will always contain a number represented in base-10.
Complete the below code so that it produces the correct output.

**Note:** Please don't change the package name.

Sample Test Cases +

Explorer

GuessData...

Submit

```
package ql0779;
public class GuessDataType {
    public static void main(String[] args) {
        //write code to convert the value pased in args[0] into a byte
        byte numberOfBits = Byte.parseByte(args[0]);

        switch(numberOfBits) {
            case 8 :
                System.out.println("byte");
                break;
            case 16 :
                System.out.println("short or a char");
                break;
            case 32 :
                System.out.println("int or a float");
                break;
            case 64 :
                System.out.println("long or a double");
                break;
            default :
                System.out.println("Could not guess the data type of size : " + numberOfBits + " bits");
                break;
        }

    }
}
```

Terminal Test cases

## 8.1.4. Identify and correct the error

00:45

Create a class `HolidaysCalculator` with a main method. The method receives two command line arguments the **first argument** contains the count of **national holidays** declared in that country and the **second argument** contains a count of **company specific holidays**. Write a code to print the total number of holidays as the sum of both national and company holidays.

For example:

```
Cmd Args : 3 4
totalHolidays = 7
```

**Note:** Please don't change the package name.

Sample Test Cases +

Explorer

HolidaysC...

Submit

```java
package q10780;
public class HolidaysCalculator {
    public static void main(String[] args) {
        Byte nationalHolidays = Byte.parseByte(args[0]);
        Byte companyHolidays = Byte.parseByte(args[1]);
        Byte totalHolidays = (byte)(nationalHolidays + companyHolidays);
        System.out.println("totalHolidays = " + totalHolidays);
    }
}
```

Terminal Test cases

Plots

## 8.2.1. What is a Float Data Type?

07:35

In Java, the `float` data type denotes a 32-bit single-precision 32-bit IEEE 754 floating point number.
In Java, a float can represent

1. both positive and negative numbers
2. positive infinity (Float.POSITIVE_INFINITY) and negative infinity (Float.NEGATIVE_INFINITY)
3. NaN (a special Not-a-Number value, i.e. mathematically undefinable number).
   A NaN value is used to represent the result of invalid operations such as dividing zero by zero, or square root of a negative number. Its value is represented by a constant in `Float` class (Float.NaN) etc.

The **32 bits** of a **float** are divided into three parts.

```
[S]   [E][E][E][E][E][E][E][E]   [M][M][M][M][M][M][M][M][M][M][M][M][M][M][M][M][M][M][M][M][M][M][M]
      - The [S] bit is used to mark the sign of the number.
      - The 8 bits (marked [E]) determines the value of the exponent.
      - The remaining 23 bits (marked [M]) determines the mantissa (significand).

value = mantissa X base^exponent   (our base is 2)
value = M X 2^E

                                                    For more info Floating Point Numbers
```

The primitive type `float` has a corresponding wrapper class called `Float`. Both float and Float can be used interchangeably. Which means, we can say:

```
float num1 = 3.0f;
float num2 = 4.2f;
float total = num1 + num2;
float valueOfNum2 = total - num1;
```

The fixed values `3.0f`, `4.2f` assigned to `num1` and `num2` respectively are called float literals.
**These are suffixed with** `f` to indicate that they are floating point values.
The default value of a primitive `float` is `0.0f`, when not initialized. However, the default value of a variable of type Float is `null`, when not initialized.

- [ ] Since a `float` data type can represent fractions, it can be used for currencies.
  No, float should not be used for currencies where precise values are required. Instead we should use java.math.BigDecimal
- [ ] A `float` uses 4 bits of memory
  Read the first line again.
- [x] A `float` uses 4 bytes of memory
  Correct!
- [ ] A `float` uses more space than an `int`
  No, both use 4 bytes in java

## 8.2.2. Usages of the Float wrapper class

05:50

The Float class wraps a primitive **float** value.

This class provides constants and methods that are useful while working with **float**(s).

For example, it has a method Float.parseFloat(String s) which can be used to convert a string value to its corresponding **float** value.

```
float x = Float.parseFloat("4.5");
//parses the string literal "4.5" and returns a float value of 4.5f
```

In the below code, the main method of **CalculateDifference** will be passed **two arguments**.
Both the values can be any number between Float.MIN_VALUE to Float.MAX_VALUE.

Complete the below code so that it produces the correct output.

**Note:** You can assume that the first string value passed in `args[0]` and the second string value passed in `args[1]` will always contain a decimal number with fractional part (like 2.99, 3.782, 4.7, etc.). It means that you can directly use the **Float.parseFloat**(text) method.

[Hint: You can convert **arg[0]** to `float` by the statement `float value = Float.parseFloat(args[0]);`]

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

CalculateD...

Submit

```
package q10782;
public class CalculateDifference {
    public static void main(String[] args) {
        // In the below two lines write code to convert
        // the string value passed in args[0] and args[1] into int values using Float.parseFloat method
        // convert the value in args[0] to int and store in firstValue
        // convert the value in args[1] to int and store in secondValue
        float firstValue = Float.parseFloat(args[0]);
        float secondValue = Float.parseFloat(args[1]);
        System.out.println("The difference of args[0] and args[1] is : " + (firstValue - secondValue));
    }
}
```

Terminal

Test cases

## 8.3.1. What is a Double Data Type?

01:07

In Java, the `double` data type denotes a 64-bit double-precision 64-bit IEEE 754 floating point number. In Java, a `double` is more commonly used than a `float` in calculations like sine, cos, tan etc... Like in float a double can represent

1. both positive and negative numbers
2. positive infinity (Double.POSITIVE_INFINITY) and negative infinity (Double.NEGATIVE_INFINITY)
3. NaN (a special Not-a-Number value, i.e. mathematically undefinable number).
   A NaN value is used to represent the result of invalid operations such as dividing zero by zero, or square root of a negative number. Its value is represented by a constant in `Float` class (Double.NaN) etc.

The **64 bits** of a **double** are divided into three parts.

```
- 1 Sign bit is used to mark the sign of the number.
- 11 bits are used to determine the value of the exponent.
- remaining 52 bits are used to determine the mantissa (significand).
                                    For more info Double Precision Floating Point Numbers
```

The primitive type `double` has a corresponding wrapper class called `Double`. Both double and Double can be used interchangeably. Which means, we can say:

```
double num1 = 3.4d;
Double num2 = 1.2d;
Double total = num1 + num2;
double valueOfNum2 = total - num1;
```

The fixed values `3.4d`, `1.2d` assigned to `num1` and `num2` respectively are called double literals. **These are suffixed with** `d` to indicate that they are double precision floating point values.
The default value of a primitive `double` is `0.0d`, when not initialized. However, the default value of a reference of type Double is `null`, when not initialized.
We will learn more about the wrapper class `Double` later.

- [ ] Since a `double` data type can represent double precision fractions, it can be used for currencies.
- [ ] A `double` uses 8 bits of memory
- [x] A `double` uses 8 bytes of memory
- [ ] A `double` uses same space as an `int`

## 8.3.2. Usages of the Double wrapper class

00:47

The Double class wraps a primitive double value.
This class provides constants and methods that are useful while working with large precision fractional numbers. For example it has a method Double.parseDouble(String s) which can be used to convert a string value to a double

```
Double y = Double.parseDouble("380.234d");
//parses the string literal "380.234d"
```

In the below code, the main method of CalculateDifference will be passed two arguments. Both the values can be any number between Double.MIN_VALUE to Double.MAX_VALUE.

Complete the below code so that it produces the correct output.

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

CalculateD...

Submit

```
package q10857;
public class CalculateDifference {
    public static void main(String[] args) {
        double firstValue = Double.parseDouble(args[0]);
        double secondValue = Double.parseDouble(args[1]);

        System.out.println("The difference of args[0] and args[1] is : " + (firstValue - secondValue));
    }
}
```

Terminal

Test cases

Plots

8.3.3. Choose the correct statements

02:23

Select all the correct statements.

- [x] `double d = 7.;` is a valid statement
- [x] `double d = .7;` is a valid statement
- [x] `double d = 7;` is a valid statement
- [x] `double d = 1e1;` is a valid statement
- [ ] The suffix `d` or `D` is a must while writing a double literal

## 8.3.4. Learn how type widening works when using float

01:50

The below statement declares a `float` and a `double` as x and y initialized to 3.4f and 5.1d respectively.

```
float x = 3.4f;
double y = 5.1d;
```

However, when we try to perform an addition operation, such as given below, we get a compilation error : `Type mismatch: cannot convert from double to float`.

```
float sum = x + y;
```

When a binary operator (like, *, / ,%, +, etc. ) is being used, if at least one operand is of type `double` and the other `float` then the operation is carried out by automatically widening the `float` operand also to `double`.
And for all other numeric types (like `byte`, `short`, `char`, `int` and `long`) the operands are automatically widened to `float` and the result is also a `float`.

In our case, since y is of type `double` and x is of type `float`, x is widened to `double`. And the resultant which is also a `double` is assigned to sum (which is of type `float`).
A compile-time error occurs while assigning the result (of type `double`) to sum. To fix it, we will have to explicitly type cast the result to a `float`, as shown below.

```
float sum = (float)(x + y);
```

Applying the above understanding identify and correct the error so that the correct output is produced.

**Note:** Please don't change the package name.

Sample Test Cases +

Explorer

FloatDem...

Submi

```
package q10858;
public class FloatDemo {
    public static void main(String[] args) {
        float x = 1.5f;
        double y = 1.1d;
        float sum = (float)(x + y);
        System.out.println("sum = " + sum);// should print 2.6
    }
}
```

Terminal Test cases

## 8.4.1. Is a float or double good for currency calculations?

01:46

**Observe the code in the editor**

If you expect it to print `0.3`, there is a nasty surprise for you!! It actually evaluates to `0.30000000000000004`

To understand what is wrong, you should know more about how floating point numbers are represented in binary format(base-2). The number of bits used for precision limits the set of rational numbers that can be represented exactly using a float or a double. This is why we get results like these when we use float or double in arithmetic operations.

> **Note:** Never use float or double in currency calculations which require a high degree of precision. `BigDecimal` is the right choice in Java for such calculations.
> We will learn about `BigDecimal` in the next sections.

Now, declare the x and y variables as **double** data types. Execute the code and observe the result

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

FloatingPo...

Submit

```java
package q10859;
public class FloatingPointArithmetic {
    public static void main(String[] args) {
        Double x = 0.2;
        Double y = 0.1;
        System.out.print(x + y);
    }
}
```

Plots

Terminal

Test cases

## 8.4.2. BigDecimal introduction

01:38

Java provides a class called BigDecimal for representing and computing decimal numbers with any required precision.

`BigDecimal` is not a wrapper class of any primitive data type. It is a convenience class for handling large decimal numbers with arbitrary precision. The value to be represented by BigDecimal should be passed to one of the BigDecimal constructors.

The provided code has omitted the inclusion of the declaration of the data type. Declare the variable **x** with the suitable data type and execute the code.

You will notice an import statement at the beginning of the program to use `BigDecimal` class. We will learn about packages and import statements in other sections.

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

BigDecima...

Submit

```
package q10860;
import java.math.BigDecimal;
public class BigDecimalUsage {
    public static void main(String[] args) {
        BigDecimal x = new BigDecimal("3.14159265358");
        System.out.print(x);
    }
}
```

Terminal

Test cases

< Prev

Reset

Submit

Next >

## 8.4.3. Understanding scale and precision in BigDecimal

18:29

`Precision` **is the total number of digits in a number.** For example,

| Number | Precision |
|---|---|
| 246 | 3 |
| 56.256 | 5 |
| 1.0074256 | 8 |

When we apply a `precision` of 3 to the value 3.14159265358, using `new MathContext(3)`, we get a result of 3.14.

When precision is applied to a **BigDecimal**, the number may undergo rounding (default being RoundingMode.HALF_UP) and the value may change. Below table shows how **BigDecimal** rounds-off numbers to given precisions:

| Number | Precision | Rounded Value |
|---|---|---|
| 2.34 | 2 | 2.3 |
| 2.36 | 2 | 2.4 |
| 24.6 | 2 | 25 |
| 56.246 | 4 | 56.25 |

Sample Test Cases

Explorer

BigDecima...

```java
package q10861;
import java.math.BigDecimal;
import java.math.MathContext;
public class BigDecimalUsage {
    public static void main(String[] args) {

        BigDecimal x = new BigDecimal("3.14159265358", new MathContext(5));
        System.out.print(x);
    }
}
```

Terminal  Test cases

## 8.4.4. BigDecimal introduction

13:01

Read and understand how the below methods work by clicking on each of the links below. Later use them to complete the partial code given below.

- add(BigDecimal augend, MathContext mc)
- subtract(BigDecimal subtrahend, MathContext mc)
- multiply(BigDecimal multiplicand, MathContext mc)
- divide(BigDecimal divisor, MathContext mc)

**Note:** Retain the precision as 5. While calculating sum and difference, let it be between **x** and **y**. Similarly consider the product of **x** and **y** and while calculating the quotient consider **x** as the dividend and **y** as the divisor.

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

BigDecima...

Submit

Plots

```java
package q10862;
import java.math.BigDecimal;
import java.math.MathContext;
public class BigDecimalUsage {
    public static void main(String[] args) {
        MathContext mathContext = new MathContext(5);
        BigDecimal x = new BigDecimal("3.145", mathContext);
        BigDecimal y = new BigDecimal("1.792", mathContext);
        BigDecimal sum = x.add(y, new MathContext(4)); //fill
        BigDecimal difference = x.subtract(y, new MathContext(4)); //fill
        BigDecimal product = x.multiply(y, new MathContext(5)); //fill
        BigDecimal quotient = x.divide(y, new MathContext(5)); //fill
        System.out.println("sum = " + sum);
        System.out.println("difference = " + difference);
        System.out.println("product = " + product);
        System.out.println("quotient = " + quotient);
    }
}
```

Terminal

Test cases

## 9.1.1. Using underscore in Numeric Literals

07:30

Java version 7 has added a new feature to improve the readability of Numeric Literals.
We can include an underscore ('_') in numeric literals to show grouping of digits.
For example:

```
long mySalary = 900_000L;// Which is equal to 900000 (Nine Hundred Thousand)
```

```
long magicNumberInHex = 0xCAFE_BABE; // Want to know more about Magic Number?
```

**Note: Underscore is only for programmer's readability, it is completely ignored during computations.**

Usage rules for underscore:

1. Underscore can only be used between two digits.
2. Underscore cannot be used at the beginning or ending of a number.
3. Underscore cannot be used before a suffix like F (used in float), L (used in Long) or D (used in Double).
4. Underscore cannot be used before or between a radix prefix used for binary or octal or hex numbers

Select all the correct answers given bellow.

- [ ] `int x = _34;` is a valid usage of Underscore
- [ ] `int x = 45_;` is a valid usage of Underscore
- [ ] `int x = 0_xAB;`
- [x] `int mySelf = 0___0___7;`

## 9.1.2. Usage of Underscore in Numeric Literals

01:11

Select all the correct statements from the given options.

- [ ] `int x = _07_08_09_;` is a valid usage of Underscore
- [x] `int x = 07_08_09;` is a valid usage of Underscore
- [x] `int x = 0xA_B;`
- [x] `int x = 1_____0;`

## 9.2.1. Understanding Boolean data type

05:32

In Java, the `boolean` data type denotes a logical quantity with two possible values, indicated by the literals `true` or `false`.

The primitive type `boolean` has a corresponding wrapper class called `Boolean`. Both `boolean` Boolean can be used interchangeably. Which means, we can say:

```
Boolean isHot = true;
boolean isSnowing = false;
boolean isSummer = isHot && (!isSnowing);
```

The fixed values `true` and `false` used above are called boolean literals.
The default value of a primitive `boolean` is `false`, when not initialized. However, the default value of a reference of type Boolean is `null`, when not initialized.
We will learn more about the wrapper class `Boolean` later.

Select all the correct statements given below.

- [ ] In Java, `true` is equal to `1` and `false` is equal to `0`
- [ ] `boolean isSuccess = "true";` is a valid statement.
- [ ] `boolean isSuccess = 'true';` is a valid statement.
- [x] `boolean isResponse = false;` is a valid statement.

## 9.2.2. Usages of the Byte wrapper class

02:28

The Boolean class wraps a primitive boolean value.
This class provides constants and methods that are useful while working with booleans. For example it has a method Boolean.parseBoolean(String s) which can be used to convert a string value representing one of the two logical states of being true or false into their corresponding boolean value

```
boolean x = Boolean.parseBoolean("true");
boolean y = Boolean.parseBoolean("false");
```

In the below code, the main method of GuessHoliday will be passed an argument. The argument can be either "true" or "false".
You can assume that the string value passed in **args[0]** will always contain a boolean value of either true or false.
Complete the below code so that it produces the correct output.

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

GuessHoli...

Subm

```
package q10810;
public class GuessHoliday {
    public static void main(String[] args) {
        //write code to convert the value pased in args[0] into a boolean
        boolean isHoliday = Boolean.parseBoolean(args[0]);
        if (isHoliday) {
            System.out.println("I am sleeping!");
        } else {
            System.out.println("I am working hard!");
        }
    }
}
```

Terminal  Test cases

## 9.3.1. What is a Character Data Type?

24:27

In Java, the `char` data type denotes a 16-bit unsigned integer (between 0 and 65535), which represent the Unicode values between '**\u0000**' and '**\uffff**'.

The primitive type `char` has a corresponding wrapper class called `Character`. Both char and Character can be used interchangeably. Which means, we can say:

```
char gender1 = 'M';
Character gender2 = gender1;
```

**A character literal must be wrapped in single quotes and it cannot span multiple lines.**

The fixed value `'M'` assigned to `gender1` is called character literal.
The default value of a primitive `char` is `0`, when not initialized. However, the default value of a reference of type Character is `null`, when not initialized.

**Since `char` is a 16-bit integer type, it can be interchangeably used with int.**
We will learn more about the wrapper class `Character` later.

Select all the correct statements given below.

- [ ] We can use an `char` data type to store a String
- [ ] We can use a `char` data type to store a Boolean value.
- [ ] A `char` literal value can be enclosed in single or double quotes.
- [x] A `char` literal value can be a number between `0` and $2^{16}$-1.

## 9.3.2. Understanding special characters

03:03

Have a look at the following code:

```
public class SpecialCharacters {
    public static void main(String[] args) {
        char aChar = 'A';
        char tabChar = '\t';
        char digitChar = '7';
        char newLineChar = '\n';
        char spaceChar = ' ';
        char hyphenChar = '-';
        char percentageChar = '%';
        System.out.print(aChar);
        System.out.print(tabChar);
        System.out.print(digitChar);
        System.out.print(newLineChar);
        System.out.print(spaceChar);
        System.out.print(hyphenChar);
        System.out.print(percentageChar);
    }
}
```

This code assigns different special characters to variables and then prints them using `System.out.print()`.

Among the options provided for you, which represents the exact sequence of **special characters** that will appear in the console output?

- A 7-% - The output will be the character 'A', followed by a tab character, the character '7', the new line character, a space character, a hyphen '-', and the percentage symbol '%'.
- A 7-% - The output will be the character 'A', followed by a space, the character '7', the new line character, a space character, a hyphen '-', and the percentage symbol '%'.
- A 7-% - The output will be the character 'A', followed by a tab character, the character '7', the new line character, a hyphen '-', and the percentage symbol '%'.
- A7\n-% - The output will be the character 'A', followed by the character '7', a newline character '\n', a hyphen '-', and the percentage symbol '%'.

## 9.3.3. Working with Character wrapper class

07:57

In Java, the primitive `char` type has a wrapper class called Character, which provide useful methods to work with characters.

Understand the usage of most commonly used methods in Character class:

1. isLetter()
2. isDigit()
3. isLetterOrDigit()

Observe the code that is provided for you in the code skeleton.

Use the appropriate methods and perform respective conversions to finish the incomplete code by following the instructions provided in the relevant comment lines.

**Note:** Please don't change the package name.

Sample Test Cases +

Explorer

Character...

Submit

```java
package q35961;
public class CharacterDemo {
    public static void main(String[] args) {
        char ch1 = 'A';
        char ch2 = '7';
        char ch3 = '*';
        // Checking if ch1 is a character
        System.out.println("ch1 is a character: " + Character.isLetter(ch1));
        // now its your return
        System.out.println("ch2 is a character: " + Character.isLetter(ch2));
        // Check if ch1 is a character
        System.out.println("ch2 is a digit: " + Character.isDigit(ch2));
        // Checking if ch1 is a letter or digit
        System.out.println("ch1 is a letter or digit: " + Character.isLetterOrDigit(ch1));
        // Checking if ch3 is a letter or digit
        System.out.println("ch3 is a letter or digit: " + Character.isLetterOrDigit(ch3));
    }
}
```

Terminal Test cases

## 9.3.4. Working with Character wrapper class

09:34

The wrapper class Character, wraps a single `char` value and provides useful methods to work with its content.

Understand the usage of most commonly used methods in Character class:

1. isUpperCase()
2. isLowerCase()
3. toUpperCase()
4. toLowerCase()
5. isWhitespace()

Observe the code that is provided in the code skeleton.

By following the guidelines provided for you in the relevant comment lines, complete the incomplete code by using certain properties of characters and perform respective conversions.

**Note:** Please don't change the package name.

Sample Test Cases +

Explorer

Character...

```java
public class CharacterDemo2 {
    public static void main(String[] args) {
        char ch1 = 'A';
        char ch2 = 'd';
        char ch3 = '-';
        // Checking if ch1 is an uppercase character
        System.out.println("ch1 is an uppercase char: " + Character.isUpperCase(ch1));
        // now its your return

        // Check if ch1 is a lowercase character
        System.out.println("ch1 is a lowercase char: " + Character.isLowerCase(ch1));
        // Check if ch2 is a lowercase character
        System.out.println("ch2 is a lowercase char: " + Character.isLowerCase(ch2));
        // Convert ch2 to uppercase
        System.out.println("ch2 to uppercase: " + Character.toUpperCase(ch2));
        // Convert ch1 to lowercase
        System.out.println("ch1 to lowercase: " + Character.toLowerCase(ch1));
        // Check if ch3 is a letter or digit
        System.out.println("ch3 is a letter or digit: " + Character.isLetterOrDigit(ch3));
        // Check if ch3 is a whitespace character
        System.out.println("ch3 is a whitespace char: " + Character.isWhitespace(ch3));
    }

}
```

Terminal Test cases

## 9.3.5. Write a Java program to Display Default values of all Primitive data types

07:53

Write a **java** program to display the default values of all primitive data types.

Write a class `PrimitiveTypes` with `main(String[ ] args)` method.

Write code to produce the below output:

```
byte default value = 0
short default value = 0
int default value = 0
long default value = 0
boolean default value = false
double default value = 0.0
float default value = 0.0
```

**Note:** Please don't change the package name.

Sample Test Cases +

Explorer

PrimitiveT...

```java
package q10815;
public class PrimitiveTypes {
    public static void main(String[] args) {
        System.out.println("byte default value = 0");
        System.out.println("short default value = 0");
        System.out.println("int default value = 0");
        System.out.println("long default value = 0");
        System.out.println("boolean default value = false");
        System.out.println("double default value = 0.0");
        System.out.println("float default value = 0.0");
    }
}
```

Terminal Test cases

## 9.4.1. Understanding String class

02:23

A String is a sequence of zero or more characters enclosed in double-quotes. For example,

```
String name = "Ganga";
```

The value `"Ganga"` is called the string literal. We cannot open a double-quote on one line and close the double-quote in another line, they both should be in the same line.

We can embed escape characters (like `\t, \n`) also in a string literal. For example,

```
String greetingText = "Hello Thames\nWelcome to London!";
```

The above line when printed would have `"Hello Thames"` in one line and `"Welcome to London!"` in the next line.

We will learn about the String class in detail later.

Select all correct statements.

- [ ] In Java, a String literal can only contain ASCII characters.
- [ ] `String text = Hello;` is a valid statement.
- [ ] `String text = 'Hello';` is a valid statement.
- [ ]
```
String text = "Hello
                World";
```
is a valid statement.
- [x]
```
String text = "Hello"
              + "World";
```
is a valid statement.

## 9.4.2. Program to Understand Command line arguments

16:57

Write the code in the below `main()` method to calculate and print the `Total` and `Average` marks scored by a student from the input given through the **command line arguments**.

Assume that **four** command line arguments `name`, `marks1`, `marks2`, `marks3` will be passed to the `main()` method in the below class with name `TotalAndAvgMarks`.

```
Sample Input and Output:
For example, if the command line arguments to the main() method are : Narmada 75.50 67.75 78.25.
The program should print the result as:
    Name = Narmada
    Marks1 = 75.5
    Marks2 = 67.75
    Marks3 = 78.25
    Total Marks = 221.5
    Average Marks = 73.833336
```

**Note:** Consider the three marks passed in the command line arguments as `floats`.

**Note:** Please don't change the package name.

Sample Test Cases +

Explorer

TotalAndA...

```java
package q10817;
public class TotalAndAvgMarks {
    public static void main(String[] args) {
        String name = (args[0]);
        float marks1 = Float.parseFloat(args[1]);
        float marks2 = Float.parseFloat(args[2]);
        float marks3 = Float.parseFloat(args[3]);
        float total = marks1 + marks2 + marks3;
        float avg = total/3;
        System.out.println("Name = " + name);
        System.out.println("Marks1 = " + marks1);
        System.out.println("Marks2 = " + marks2);
        System.out.println("Marks3 = " + marks3);
        System.out.println("Total Marks = " + total);
        System.out.println("Average Marks = " + avg);
    }
}
```

Terminal Test cases

## 9.5.1. Understanding Escape Sequences in char and String literals

05:15

In Java, backslash character ( `\` ) is used to mark an escape sequence. An Escape Sequence is an escape character `\` followed by a character, like `\n` or `\t`. The presence of the escape character changes the meaning of the character which follows it. For example when we print the String literal `"Hello\tWorld"` we would see the result as

```
Hello	World
```

In the String literal `"Hello\tWorld"`, `\t` represents the TAB character.

Similarly if we want to print a double quote inside a double quoted String literal, we need to escape the double quote by the escape character `\`. For example :

```
String escapedDoubleQuote = "Hello \" (Quote)";
System.out.println(escapedDoubleQuote);
```

Would produce the below output:

```
Hello " (Quote)
```

Similarly if we want to print a single quote inside a char literal, we need to escape the single quote by the escape character. For example :

```
char singleQuoteChar = '\'';
```

Place appropriate escape characters in the given code to get the output as shown in the test cases. **Note** the effects of `\t` and `\n` in the resulting output when executed successfully.

**Note:** Please don't change the package name.

Sample Test Cases +

Explorer

EscapeSeq...

```
package q35995;
public class EscapeSequenceDemo {
	public static void main(String[] args) {
		System.out.print("One\tTwo Three");
		System.out.print("\n");
		System.out.print("Four");
	}
}
```

Terminal Test cases

Section - 1 [Section 1 of 1] | Q1 of 2 | Marks 5

Jump to

100%

Test time left: 56:4

## Spacing between integers

Write a Java program that takes two integers from the user and prints them on a single line, separated by a tab space.

**Input format:**
The two lines of the input are the integers.

**Output format:**
The output is the integers separated by a tab space.

**Note:**
The code for handling inputs is already been given, your task is to fill in the required code.

## Sample Test Cases

**Test Case 1:**

**Expected Output:**

```
21
23
21→23
```

**Test Case 2:**

**Expected Output:**

```
123
456
123→456
```

q22550/IntegerConcat

Submit

```java
package q22550;

import java.util.Scanner;

public class IntegerConcatenator {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int num1 = scanner.nextInt();

        int num2 = scanner.nextInt();

    // write your code here...
        System.out.println(num1 + "\t" + num2);
        scanner.close();
    }
}
```

## Preceding letter

Write a Java program that prompts the user to enter a single letter from the alphabet series and displays the preceding letter in the alphabet series.

**Input format:**
The input is a single letter from the alphabet series.

**Output format:**
The output is the preceding letter to the user-given letter.

**Note:**
- Input character must be between **B** to **Z** or **b** to **z**.
- The code for handling inputs is already been given, your task is to fill in the required code.

## Sample Test Cases

**Test Case 1:**

**Expected Output:**

```
I
H
```

**Test Case 2:**

**Expected Output:**

```
f
e
```

q22548/PreviousLetter

Submit

```java
package q22548;

import java.util.Scanner;

public class PreviousLetter {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        char inputLetter = scanner.next().charAt(0);

        // write your code here..
System.out.println((char)(inputLetter - 1));
    }
}
```

## Execution Results

2 out of 2 shown cases successful

2 out of 2 hidden cases successful

✔ Test Case - 1 (Execution Time: 205 ms)

| Expected Output | User Output |
|---|---|
| I<br>H | I<br>H |

✔ Test Case - 2 (Execution Time: 211 ms)

| Expected Output | User Output |
|---|---|
| f<br>e | f<br>e |

9.3. Character Data Type

9.4. String

9.5. Escape Characters

9.5.1. Understanding Escape Sequences in...

9.6. Java - Data types [Unscheduled]

10. Unit 1 - Lesson 9

10.1. Variables

10.1.1. Understanding Variables and its Types

10.1.2. Difference between Variable and Re...

10.2. Keywords

10.2.1. Understanding Java keywords

10.3. NaN and null

10.3.1. Correct usage of NaN

10.3.2. Understanding null literal

10.4. valueOf() method

## 10.1.1. Understanding Variables and its T...

01:22

In any programming language, a variable is a named storage location. That name is also called as `identifier`. For example:

```
int age = 25;
String name = "Bhargo";
```

Java variables names must start with a character and can include digits. We will learn more about Java Naming and Coding conventions later.

Java being a strongly typed language, every variable has a type associated.

In Java, there are two types of variables

1. **Primitive Type** ( henceforth we will refer to them simply as **variables**)
2. **Reference Type** ( henceforth we will refer to them as **references**)

The primitive types are `boolean` and the remaining numeric types ( `byte`, `short`, `int`, `long`, `float`, `double` and `char` )

The reference types can be of any `class` or of type `array`.

For example, in the earlier code snippet, we will refer to `age` as a **variable** since it is of a primitive type ( `int` )
Similarly we refer to `name` as a **reference**, since it is of type `String` class.

Select all the correct statements that apply to the below code.

- [ ] `gender` is a reference
- [ ] `today` is a variable
- [x] `buffer` is a reference
- [x] `namesArr` is a reference

## 1.2. Difference between Variable and Reference

01:19

the below code

```
double x = 2.3d;
String name = "Bhargo";
byte[] buffer = new byte[1024];
```

- `x` is called a variable
- `name` is called a reference
- `buffer` is called a reference

A variable can be compared to a cup, which holds the value of the primitive date type.
A reference can be compared to a cup, which holds the address of the object it points to.

- When not initialized, the default value of a variable is `0` for numeric primitive types and `false` for boolean types.
- When not initialized, the default value of a reference is `null`. If a reference has a `null` value, it means that the reference does not refers (points) to any object.

Select all the correct statements that apply to the below code.

```
String name = null;
int age = null;
byte[] buffer = null;
boolean status = null;
```

- [x] Statement `String name = null;` is correct
- [ ] Statement `int age = null;` is correct
- [x] Statement `byte[] buffer = null;` is correct
- [ ] Statement `boolean status = null;` is correct

## 10.2.1. Understanding Java keywords

00:36

In every programming language there are certain keywords. Below are the keywords in Java:

| | | | | |
|---|---|---|---|---|
| abstract | continue | for | new | switch |
| assert | default | if | package | synchronized |
| boolean | do | goto | private | this |
| break | double | implements | protected | throw |
| byte | else | import | public | throws |
| case | enum | instanceof | return | transient |
| catch | extends | int | short | try |
| char | final | interface | static | void |
| class | finally | long | strictfp | volatile |
| const | float | native | super | while |

These words have special meaning and usage. They cannot be used for naming variables, references or methods or classes.

- [x] **null** is not a keyword in Java.
- [x] **if** is a keyword.
- [ ] **main** is a keyword.
- [ ] **Class** is a keyword.

< Prev | Reset | Submit | Next >

## 3.1. Correct usage of NaN

03:41

ne Float.NaN and Double.NaN refer to a result that is not a number.
or example we get a NaN (not a number result) when dividing zero by zero or when we try to alculate the square root of a negative number.

While coding in Java, to verify if the result of an expression is **not a number**, we should never compare the result (using == operator) to either Float.NaN or Double.NaN.

The correct way of verifying if the value is not a number is to use the method Float.isNaN() or Double.isNaN().

Observe the provided code in the editor, Use the **NaN** in an appropriate manner so that the code print the result as **true.**

Sample Test Cases

Explorer

CorrectU...

Submit

```
package q10823;
public class CorrectUsageOfNaN {
    public static void main(String[] args) {
        double x = 0d / 0d;
        System.out.println("x == Double.NaN : " + (x == Double.NaN));
        System.out.println("Double.isNaN(x) : " + (Double.isNaN(x)));
    }
}
```

Terminal  Test cases

‹ Prev  Reset  Submit  Next ›

## 3.2. Understanding null literal

01:30

Java, `null` is a special type. It has only one value and that is `null`.

he `null` literal is written as plain `null`, without any double quotes.

reference having a `null` value indicates that it is not pointing to a valid object.

Java code, when an attempt is made to access a member (like field or method using the dot ) f a null reference, `NullPointerException` is thrown. We will learn more about Exceptions ater.

```
Float a;  // a is initalized to null, it is equal to saying Float a = null;
String text;  // text is initalized to null, it is equal to saying String text = nu
int x; // x is initialized to 0
```

There is an incomplete code provided in the editor. Your task is to complete the code by assigning the values of all declared variables to a **null** value and observe the result.

Note: Please don't change the package name.

Sample Test Cases

Explorer

NullTest.j...

Submit

```
package q10824;
public class NullTest {
    public static void main(String[] args) {
        String text = null;
        System.out.println("text : " + text);
        byte[] buffer = new byte[1024];
        buffer = null;
        System.out.println("buffer : " + buffer);
        String name = null;
        System.out.println("text == name : " + (text ==
name));
    }
}
```

Terminal  Test cases

Plots

## 0.4.1. Understanding valueOf() method

03:42

The below table shows the **primitive types** and their corresponding **wrapper classes** in Java:

| char | Character | Number Types |
|---|---|---|
| byte | Byte | Byte.valueOf(String s) |
| short | Short | Short.valueOf(String s) |
| int | Integer | Integer.valueOf(String s) |
| float | Float | Float.valueOf(String s) |
| double | Double | Double.valueOf(String s) |
| long | Long | Long.valueOf(String s) |

All the above wrapper classes present for the **number types** contain a useful method called `valueOf()`, which converts a string to an instance of that number type.

Fill the missing code in the given code to understand the usage of the `valueOf()` method in the wrapper classes.

Sample Test Cases

Explorer

Wrapper...

Submit

```java
package g36054;
public class WrapperClassDemo {
    public static void main(String[] args) {
        String text1 = "101";
        String text2 = "34";
        String text3 = "5.5";
        String text4 = "-35.593933";
        String text5 = "93593933";


        // 1. print the Byte value of text1.
        System.out.println(Byte.valueOf(text1));
        // 2. print the Short value of text2.
        System.out.println(Short.valueOf(text2));
        // 3. print the Integer value of text2.
        System.out.println(Integer.valueOf(text2));
        // 4. print the Float value of text3.
        System.out.println(Float.valueOf(text3));
        // 5. print the Double value of text4.
        System.out.println(Double.valueOf(text4));
        // 6. print the Long value of text5.
        System.out.println(Long.valueOf(text5));


    }
}
```

Terminal  Test cases

< Prev  Reset  Submit  Next >

## 11.1.1. Understanding Arithmetic Operators

00:33

In Java, we have 5 arithmetic operators:

| Operator | Description |
|---|---|
| + | Used for addition and string concatenation |
| - | Used for subtraction |
| * | Used for multiplication |
| / | Used for division |
| % | Remainder/Modulus operator for finding remainder |

The usage is

```
operand1 operator operand2
```

**According to Java coding conventions, a single space should be provided to the left and right of an operator.**

**Note:** When any of the above arithmetic operators is used in an expression, if at least one operand is of type `double`, `float` or `long` then the operation is carried out by automatically widening the other operand also to that former's type (if it is not already of the former's type) and the result is also of the former's type. And for all other numeric types (like `byte`, `short`,

x + y = 19
x - y = 13
x * y = 48
x / y = 5
x % y = 1

x + y = 19
x - y = 13
x * y = 48
x / y = 1
x % y = 5

x + y = 19
x - y = 13
x * y = 39
x / y = 5
x % y = 1

x + y = 18
x - y = 13
x * y = 39
x / y = 5
x % y = 1

Next >

## 11.2.1. Understanding Comparison and Relational Operators

00:55

In Java, below are the comparison and relational operators:

| Operator | Description |
|---|---|
| == | Checks if two values are equal |
| != | Checks if two values are not equal |
| > | Checks for greater-than condition |
| >= | Checks for greater-than-or-equals condition |
| < | Checks for less-than condition |
| <= | Checks for less-than-or-equals condition |

The usage is

```
operand1 operator operand2
```

**According to Java coding conventions, a single space should be provided to the left and right of the operator.**

The result of the expression involving any of the above mentioned comparison or relational operators is always a `boolean` value (true or false).

- [x] `x` and `z` are equal
- [ ] `x` is greater than `z`
- [x] `x` and `y` are not equal
- [ ] `x` is not less than or equal to `z`

< Prev | Reset | Submit | Next >

## 12.1.1. Understanding Unary Operators

02:06

In Java, we have 5 Unary operators:

| Operator | Description |
|---|---|
| + | Unary Plus, used for indicating a positive value |
| - | Unary Minus, subtracts from zero |
| ++ | Increment operator. Increments the value by one. |
| -- | Decrement operator. Decrements the value by one. |
| ! | Negation operator. Negates a boolean value. |

The usage is

```
operator operand
```

**According to Java coding conventions, no space should be provided between a unary operator and operand.**

**Note:** When Increment or Decrement operator is applied before an operand (e.g. ++i or --i), the value of the operand is changed first and the changed value is substituted in the expression.

- [x] Value of y is -10
- [ ] Value of z is 11
- [ ] Value of k is 10
- [ ] Value of k is 11
- [x] Value of k is 12

< Prev Reset Submit Next >

## 12.1.2. Understanding Unary Operators

08:10

Write a Java program that demonstrates the use of **unary operators**.

Implement a class called `UnaryOperatorsDemo` with a `main` method. Inside the `main` method, perform the following operations:

- Print the value of **x** after applying the unary `+` operator to it.
- Print the value of **x** after applying the unary `-` operator to it.
- Print the original value of **x**.
- Increment the value of **x** by **1** using the pre-increment operator.
- Print the updated value of **x**.
- Print the value of **x** using the post-increment operator.
- Print the updated value of **x**.
- Decrement the value of **x** by **1** using the pre-decrement operator.
- Print the updated value of **x**.
- Print the value of **x** using the post-decrement operator.
- Print the final value of **x**.
- Declare a boolean variable `flag` and initialize it with the value `false`.
- Print the value of `flag`.
- Print the negation of `flag` using the logical NOT operator ('!'). Your task is to write the complete missing code for the `UnaryOperatorsDemo` class, ensuring that it produces the expected output when executed.

Sample Test Cases

Explorer

UnaryOp...

Submit

```java
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a number: ");
        int x = scanner.nextInt();

        System.out.println("+x = " + +x);
        System.out.println("-x = "+ -x);
        System.out.println("x = " + x);
        System.out.println("++x = " + ++x);
        System.out.println("x = " + x);
        System.out.println("x++ = " + x++);
        System.out.println("x = " + x);
        System.out.println("--x = " + --x);
        System.out.println("x = " + x);
        System.out.println("x-- = " + x--);
        System.out.println("x = " + x);

        System.out.print("Enter a boolean value (true/false): ");
        boolean flag = scanner.nextBoolean();

        System.out.println("flag = " + flag);
        System.out.println("!flag = " + !flag);


        scanner.close();
    }
```

Terminal  Test cases

Plots

## 12.2.1. Understanding Assignment Operator

01:29

In Java, `=` is the assignment operator. It is the most common and important operator in any programming language.
It is used to assign a value to a variable. Its usage is

```
LHS = RHS;
```

LHS (left-hand side to the `=` operator) can be a field (variable), expression or an array access.
RHS (right-hand side) can be a value, variable or an expression.
For example

```
int x = 10;                  // variable x is assigned value 10
String[] namesArr = new String[10]; // an empty String array of size 10 is assigned to
namesArr[0] = "Ganga";          // namesArr's 0th index is assigned a value "Ganga"
car.speed = getCurrentSpeed();      // the field speed in car reference is assigned a
```

Assignment operator verifies the compatibility of types of LHS and RHS.
Automatic type widening is performed when possible, else a compilation error is reported.

**According to Java coding conventions, a single space should be provided to the left and right of the assignment operator.**

Select all valid assignment statements.

- [ ] int x = true;
- [x] String text = null;
- [ ] byte[] buffer = "Ganga";
- [x] int myHeight = Integer.MAX_VALUE;
- [x] int x = 3, y = 5, z = 5;

## 12.3.1. Understanding Conditional Operators

08:15

In Java, `&&` represents conditional-and operator (also called `logical-and`) and `||` represents conditional-or operator (also called `logical-or`).
Their usage is

```
(expression1) operator (expression2)
```

For example, the expression `(isWeekend && isRaining)` is read in English as "isWeekend and isRaining"

Similarly, the expression `(isTasty || isSweet)` is read in English as "isTasty or isSweet"

Both conditional-and operator( `&&` ) and conditional-or operator( `||` ) operators evaluate from left to right.

In case of a `&&`, if the left expression evaluates to `false`, it will skip evaluating the expression on the right side of the operator.
In case of a `||`, if the left expression evaluates to `true`, it will skip evaluating the expression on the right side of the operator.

The result of the expression involving any of the above mentioned conditional operators is always a `boolean` value (true or false).

**According to Java coding conventions, a single space should be provided to the left and right of the operator.**

Select all valid statements for the below code:

```
int x = 3;
int y = 4;
```

- [ ] `x > y && y > z is true` is printed
- [x] `z == x && y > z is true` is printed
- [ ] `x == y || x > z is true` is printed
- [x] `x == y || x > z is false` is printed

Home Learn Anywhere

## 3.2. Understanding the usage of Ternary Operator

01:47

here is yet another conditional operator called the ternary operator `?:`. Its usage is

```
condition ? expression1 : expression2
```

he condition should always evaluate to a `boolean`. If the condition evaluates to `true`, xpression1 is evaluated and its value returned, else expression2 is evaluated and its value eturned. For example,

```
int javaCertificationMarks = 75;
int passMarks = 65;
String result = (javaCertificationMarks > passMarks) ? "Passed Java Certification" : "F
```

n the above code, since the condition `javaCertificationMarks > passMarks` evaluates to `true`, `result` is assigned the value `"Passed Java Certification"`

There is an incomplete code provided in the editor. Your task is to use the ternary operator in an appropriate position so that it displays the result without any errors.

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

TernaryO...

Submit

```
package q10834;
public class TernaryOperatorsDemo {
    public static void main(String[] args) {
        int javaCertificationMarks = 75;
        int passMarks = 65;
        String result = (javaCertificationMarks > passMarks) ?
 "Passed Java Certification" : "Failed Java Certification";
        System.out.println("result : " + result);
    }
}
```

Plots

Terminal Test cases

Prev Reset Submit Next

## .1.1. Understanding Bitwise Operators

04:26

Java, below are the Bitwise and Bit Shift operators:

| Operator | Description |
|---|---|
| ~ | is the **unary bitwise complement** operator. It flips the bits represented by a number. Its usage is `~operand`, the operand should be of primitive integral type. For example if a binary number `110` is flipped we get `001` |
| & | is the **bitwise AND** operator. Its usage is `operand1 & operand2`, where operand1 and operand2 should be of primitive integral type. It sets a bit to 1 if and only if both of the corresponding bits in its operands are 1, and to 0 if the bits differ or both are 0. E.g. `1010 & 1100` will result in `1000`. |
| \| | is the **bitwise OR** operator. Its usage is `operand1 \| operand2`, where operand1 and operand2 should be of primitive integral type. It sets a bit to 1 if one or both of the corresponding bits in its operands are 1, and to 0 if both of the corresponding bits are 0. E.g. `1010 \| 1100` will result in `1110`. |

- [ ] Statement -1 results in a value of `"Ga"` being assigned to `x`
- [ ] Statement -2 results in a value of `23.4` being assigned to `y`
- [x] Value of `k % 11`
- [x] Value of `z % 44`

< Prev | Reset | Submit | Next >

## 13.2.1. Understanding instance of Operator

33:52

In Java, there is a keyword `instanceof`, which is used to verify if an object is an instance of a particular class.
Its usage is

```
reference instanceof ClassName
```

For example, the expression `(args instanceof String[])` evaluates to `true`.

The result of an expression using `instanceof` is always a `boolean` value (true or false).

**According to Java coding conventions, a single space should be provided to the left and right of this operator.**

Select all the valid statements in the below code:

```
String name1 = "Ganga";
String name2 = "Amazon";

boolean statement1 = name1 instanceof String;

boolean statement2 = name2 instanceof name1;

boolean statement3 = name1 instanceof Object;

boolean statement4 = args instanceof Object;

boolean statement5 = args instanceof String[];
```

- [x] `statement1` evaluates to `true`
- [ ] `statement2` evaluates to `true`
- [x] `statement2` will not compile.
- [ ] `statement3` will not compile
- [ ] `statement3` evaluates to `false`
- [ ] `statement4` evaluates to `false`
- [x] `statement5` evaluates to `true`

## 6.2.2. Understanding the usage of instanceof Operator

01:08

There is a code provided in the editor. Your task is to use the **instanceof** operator appropriately in order to acquire the output as shown in the displayed test case.

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

Instance...

Submit

Plots

```
package q10836;
public class InstanceOfOperatorDemo {
    public static void main(String[] args) {
        if (args instanceof String[]) {
            System.out.println("args instanceof String[] is true");
        }
        if (args instanceof Object) {
            System.out.println("args instanceof Object is true");
        }
    }
}
```

Terminal

Test cases

## 13.3.1. Understanding String Concatenation Operator

02:37

In Java, `+` operator is also used to concatenate strings. For example:

| Expression | Result |
|---|---|
| "Ganga" + "River" | "GangaRiver" |
| "Ganga" + (2 + 3) | "Ganga5" |
| "Ganga" + 2 + 3 | "Ganga23" |
| 2 + 3 + "Ganga" | "5Ganga" |
| 2.3f + "Ganga" | "2.3Ganga" |
| "Ganga" + 'R' + 3 + true | "GangaR3true" |

Below are the rules involved in String concatenation:

1. The `+` operator is syntactically left-associative, when used in string concatenation or numeric addition
2. If `one operand` in the concatenation expression is of type String, the other operand is also `converted to String`.
3. The `result of a String concatenation` expression is a reference to a `String` object containing characters in the left-side operand followed by the characters in the right-side operand.

According to Java coding conventions, a single space should be provided to the left and

Value of `text3 = "GangaRiver"`

`String text4 = text1 + x + text2 + y;` is a compilation error. Since we have mixed up `String` and `int` data types while using `+` operator.

Value of `text5 = "34River"`

Value of `text6 = "GangaRiver34"`

`String text7 = text1 + '-' + today;` is a compilation error.

< Prev Reset Submit Next >

## 3.1. Understanding String Concatenation Operator

02:37

Java, `+` operator is also used to concatenate strings. For example:

| Expression | Result |
|---|---|
| "Ganga" + "River" | "GangaRiver" |
| "Ganga" + (2 + 3) | "Ganga5" |
| "Ganga" + 2 + 3 | "Ganga23" |
| 2 + 3 + "Ganga" | "5Ganga" |
| 2.3f + "Ganga" | "2.3Ganga" |
| "Ganga" + 'R' + 3 + true | "GangaR3true" |

Below are the rules involved in String concatenation:

1. The `+` operator is syntactically left-associative, when used in string concatenation or numeric addition
2. If `one operand` in the concatenation expression is of type String, the other operand is also `converted to String`.
3. The `result of a String concatenation` expression is a reference to a `String` object containing characters in the left-side operand followed by the characters in the right-side operand.

According to Java coding conventions, a single space should be provided to the left and

Value of `text3 = "GangaRiver"`

`String text4 = text1 + x + text2 + y;` is a compilation error. Since we have mixed up `String` and `int` data types while using `+` operator.

Value of `text5 = "34River"`

Value of `text6 = "GangaRiver34"`

`String text7 = text1 + ' ' + today;` is a compilation error.

< Prev | Reset | Submit | Next >

## Understanding usage of String concatenation operator

01:35

ass `PrinHello` is created with a **main** method. The method receives river names as ents.

given code, include a print statement in order to display the concatenation of the two s (river name and suffix).

:

efer to the displayed test case for better understanding

lease don't change the package name.

Sample Test Cases

Explorer

PrintHell...

Submit

```
package q10839;
public class PrintHello {
    public static void main(String[] args) {
        String suffix = "River";
        for (String riverName : args) {
            System.out.println(riverName + suffix);
        }
    }
}
```

Terminal

Test cases

Prev

Reset

Submit

Next

## 3.1. Understanding Compound Assignment Operators

02:25

va supports the below compound assignment operators

`= -= *= /= %= &= ^= |= <<= >>= >>>=`

Usage:

```
int x = 3, y = 4, z = 0;

z += x; // is same as writing z = z + x;
z -= y; // is same as writing z = z - y;
```

Select all the valid statements from the below code.

```
int x = 2, y = 4, z = 5, value1 = 0, value2 = 0, value3 = 0;

value1 *= x;

value2 = x += 2 * y;

y *= z;

value3 -= 1;
```

- [x] `value1 = 0`
- [ ] `int value2 = x += 2 * y;` results in a compilation error
- [x] `y *= z;` results in `y`'s value being changed to `20`
- [x] `value3 = -1`

< Prev | Reset | Submit | Next >

## 1.1. Understanding Operator Precedence

03:09

an expression when two or more operators are used there are certain rules followed to aluate the expression. For example:

```
int result1 = 3 + 4 * 3; // evaluates to 15 and not 21
```

he below table shows the precedence applied to the operators during their evaluation.

| Precedence | Operators |
|---|---|
| `expr++ expr--` | postfix |
| `++expr --expr +expr -expr ~ !` | unary |
| `* / %` | multiplicative |
| `+ -` | additive |
| `<< >> >>>` | shift |
| `< > <= >= instanceof` | relational |
| `== !=` | equality |
| `&` | bitwise AND |
| `^` | bitwise exclusive OR |

- [ ] `value1 = 18`
- [x] `value2 = 12`
- [ ] `int value3 = ++z + ++k;` results in compilation error.
- [ ] `value4 = 12`

## Understanding Implicit Type Conversion

01:51

a, an expression always has a type, which is deduced from the types of the variables, ds or literals involved in that expression.

**conversion** is performed to convert one or both the operands to an appropriate data type e evaluation.

`e conversion` means converting one data type value into another data type value.

e are two types of **type conversions**: implicit conversion (also called **type coercion**) and icit conversion (also called **type casting**).

ase of `implicit type conversion`, compiler automatically converts one data type value another data type value.

licit type conversions can occur during assignment or while using any other operators. During signment the R-value is converted to the type of L-value.

hen values of different data types are used in arithmetic, relational and logical operators, the lue of the lower size data type size is converted automatically into the data type of higher size fore the evaluation.

or example in the statement `float a = 5 + 5.6;` **integer** type value `5` is automatically onverted into a **float** type as `5.0` before the addition is performed.

Sample Test Cases

Explorer

ImplicitC...

Subm

```
package q10842;
public class ImplicitConversion {
    public static void main(String[] args) {
        int i = 20, intsum;
        int ch = 'a';
        float floatsum;
        intsum = i + ch;
        floatsum = i + ch;
        System.out.println("Integer result = " + intsum + " Float result = " + floatsum);
    }
}
```

Terminal  Test cases

## Understanding Explicit Type Conversion

03:14

grammer can instruct the compiler to explicitly convert a value of one type to another using **ecast** operator.

n a `typecast` operator is explicitly used, the type conversion process is called `licit type conversion` or `type casting`.

syntax for using a **typecast operator** is: `(data_type) expression`, Where the **ression** is converted to the target **data_type** enclosed within the parentheses.

re the expression may contains constants or variables and the **data_type** must be a primitive a type or void.

r example, the expression `(float)1 / 3` is evaluated as `1.0 / 3` yielding **0.333333** here as `1 / 3` yields **0**.

the expression `((int)num)%2`, if `num` is a **float** variable with value `5.5`, then the xpression evaluates to `1`.

Below is an example which demonstrates **type casting**:

```
public class ExplicitConversion {
    public static void main(String[] args) {
        float x, y;
        x = 7 / 3;
```

Sample Test Cases

Explorer

ExplicitC...

```
package q10843;
public class ExplicitConversion {
    public static void main(String[] args) {
        int i = (int)18.99f;
        System.out.println("int value = " + i);
        float f = i;
        System.out.println("after float widening : " + f);
        int big = 1234567890;
        float approx = big;
        System.out.println("The lose value = " + (big - (int)approx));
    }
}
```

Terminal  Test cases

< Prev  Reset  Submit  Next >

## 15.1.1. Understanding Control Flow Statements and if-else

02:10

The statements inside a Java program are generally executed from top to bottom, in the order they appear. `Control flow statements` break up the flow of execution by employing decision making, looping, and branching, enabling the program to conditionally execute particular blocks of code.

The `if-then` statement tells a program to execute a certain section of code only if a particular test evaluates to **true**. `if (Expression) {Statement}`.

```
int SENIOR_CITIZEN_AGE = 60;
if (age > SENIOR_CITIZEN_AGE) {
    System.out.println("Person is a senior citizen");
}
```

The `if-then-else` statement provides a secondary path of execution when an `if` clause evaluates to **false**. `if (Expression) {Statement} else {Statement}`.

```
int SENIOR_CITIZEN_AGE = 60;
if (age > SENIOR_CITIZEN_AGE) {
    System.out.println("Person is a senior citizen");
} else {
    System.out.println("Person is not a senior citizen.");
}
```

Sample Test Cases

Explorer

VotingTes... VotingTest...

Submit

```
package q10846;
public class VotingTest {
    int VOTING_AGE = 18;
    public void canVote(int age) {
        //Write the code with if-else statement here to produce the correct output
        System.out.println(((age >= 18) ? "Can Vote" : "Cannot Vote"));
        // System.out.println(answer);
        // if(age >= 18){
        //  System.out.println("Can Vote");
        // }
        // else{
        //  System.out.println("Cannot Vote");
        // }
    }
}
```

Terminal Test cases

< Prev Reset Submit Next >

## 5.1.2. Understanding if-then-else statement

06:08

The `if-else-if` construct extends the `if-else` construct by allowing to chain multiple `if` constructs as shown below:

```
if (expression_1) {
    statement_1;
} else if (expression_2) {
    statement_2;
} else if (expression_3) {
    statement_3;
} else if (expression_4) {
    statement_4;
} else {
    statement_5;
}
```

As shown in the above syntax, multiple `if` constructs can be chained to any length. The `else` construct which appears at the end is optional, and if it is to be included it has to be only at the end.

The `if-else-if` construct is used whenever we have multiple mutually exclusive `if` conditions which work on the same input.

In a `if-else-if` construct the conditions are evaluated from top to bottom. Whenever a condition evaluates to **true** (1), the control enters into that `if-block` and after that the control comes out of the complete `if-else-if` construct ignoring all the remaining `if` and `else`

Sample Test Cases

Explorer

TicketPr...

```
1  package q10847;
2  public class TicketPrice {
3      int INFANT_AGE = 3;
4      int CHILD_AGE = 13;
5      int SENIOR_CITIZEN_AGE = 60;
6      int CENTENARIAN_AGE = 100;
7
8      public void printTicketPrice(int age) {
9
10         String answer = (age <= INFANT_AGE || age >=100) ?
   "Ticket Price: 0":(age <=13 || age >= 60) ? "Ticket Price: 
   5" : "Ticket Price: 10";
11         // System.out.println("Ticket Price: 0");
12         // //fill the appropriate else if condition
13         // System.out.println("Ticket Price: 5");
14         // //fill the appropriate else condition
15         // System.out.println("Ticket Price: 10");
16         System.out.println(answer);
17
18     }
19 }
```

Terminal  Test cases

## 15.1.3. Understanding if-then-else statement

02:57

See the below code which uses a nested `if-then-else` statement for calculating grade for a given GPA.

The class `GradeCalculator` has a method `calculateGrade(double gpa)`. It receives a double argument `gpa`, a decimal number between 0.0 and 5.0. The method uses a `if-then-else` statement to display the appropriate Grade for the GPA.

Add another condition that will print `Grade: D` when the given GPA is between 2.0 and 2.49 (both inclusive).

Do use a the System.out.**println**() instead of System.out.**print**().

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

GradeCal... GradeMai...

Submit

```
package q10848;
public class GradeCalculator {
    public void calculateGrade(double gpa) {
        if (gpa >= 4.5) {
            System.out.println("Grade: A");
        } else if (gpa >= 3.5 && gpa <= 4.49) {
            System.out.println("Grade: B");
        } else if (gpa >= 2.5 && gpa <= 3.49) {
            System.out.println("Grade: C");
        } else if (gpa >= 2.0 && gpa <= 2.49){
            System.out.println("Grade: D");
        }
        else {
            System.out.println("Grade: F");
        }
    }
}
```

Terminal Test cases

< Prev Reset Submit Next >

## 15.1.4. Understanding if-then-else statement

01:44

See the below code which uses a `if-then-else` statement for calculating AM or PM for a given hour.

The class `AmPmCalculator` has a method `checkAmPm` it takes one parameter `hour` of type `int`, with value between 0 and 23. The method should use `if-then-else` statement to display AM or PM.

Fill in the `if` condition to check if hour is between 0 and 11 (both inclusive) for **AM**. Fill in the `else if` condition to check if hour is between 12 and 23 (both inclusive) for **PM**.

**Note**: Use System.out.**println**() instead of System.out.**print**().

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

AmPmCa... AmPmMai...

Submit

```
package q10849;
public class AmPmCalculator {
    public void checkAmPm(int hour) {
        if ( hour >=0 && hour <=11) { //fill the condition for AM here
            System.out.println("AM");
        } else if (hour >=12 && hour <= 23) { //fill the condition for PM here
            System.out.println("PM");
        } else {
            System.out.println("Invalid Hour!!");
        }
    }
}
```

Terminal Test cases

< Prev Reset Submit Next >

## 15.1.5. Understanding if-then-else statement

04:05

Write code which uses `if-then-else` statement to check if a given account balance is greater or lesser than the minimum balance.

Write a class `BalanceCheck` with `public` method `checkBalance` that takes one parameter balance of type `double`.

Use `if-then-else` statement and print **Balance is low** if balance is less than **1000**. Otherwise, print **Sufficient balance**.

**Note:** You need not write the **main** method. You can directly write the `checkBalance(double balance)` method in the `BalanceCheck` class.

Use System.out.**println()** instead of System.out.**print()**.

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

BalanceC... BalanceM...

Submit

```
package q10850;
class BalanceCheck
    {
        public void checkBalance(double balance)
        {
            double minimun = 1000;
            String answer = (balance <1000)?"Balance is low":"Sufficient balance";
            System.out.println(answer);
        }
    }
```

Terminal Test cases

< Prev Reset Submit Next >

## 5.1.6. Write a Java code to calculate the Roots of a Quadratic equation

38:27

Write code to calculate **roots** of a **quadratic equation**.

Write a class `QuadraticRoots` with `main` method. The method receives three arguments, write code to parse them into `double` type.

For example:

```
if the values 2, 5, 3 are passed as arguments, then the output should be First root is
If the values 3, 2, 1 are passed then the output should be Roots are imaginary.
Similarly, if the values 2, 4, 2 are passed then the output should be Roots are equal a
```

**Note:** Make sure to use the **print()** and not the **println()** method.

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

Quadratic...

Submit

```java
package q10851;
import java.lang.Math;
class QuadraticRoots
{
    public static void main(String[] args)
    {
        double a = Double.parseDouble(args[0]),b = Double.parseDouble(args[1]),c = Double.parseDouble(args[2]);
        double Dis = ((b*b)-(4*a*c)), sqrt = Math.sqrt(Dis);

        if (Dis < 0 ){
            System.out.println("Roots are imaginary");
        }
        else if(Dis == 0 ) {
            double root = (-b + sqrt)/(2*a);

            System.out.println("Roots are equal and value is : " + root);
        } else{
            double FirstRoot = ((-b + sqrt)/(2*a)), SecondRoot = ((-b - sqrt)/(2*a));
            System.out.println("First root is : " + FirstRoot + " Second root is : " + SecondRoot);
        }
    }
)
```

Terminal Test cases

The program should take a single input, which is the year (an integer).

**Output format:**

The program should output whether the given year is a valid Olympic year or not.

Execution Results

Show Results

## Submitted Source Code :

```
//File Name: q22407/OlympicYearValidator.java
//=====================================
package q22407;
import java.util.Scanner;
public class OlympicYearValidator {
   public static boolean isValidOlympicYear(int year) {

            // write your code here
            return (year%4==0) ? true : false;


   }
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter the year: ");
      int year = scanner.nextInt();
      boolean isValidOlympic = isValidOlympicYear(year);
      System.out.println(year + " is" + (isValidOlympic ? " " : " not ") + "a valid Olympic year.");
   }
}
```

2 Compilation Errors ✓ 5.0 View Answer

**Output format:**

The program should output the total cost based on the criteria mentioned above.

Execution Results

Show Results

## Submitted Source Code :

```
    if (numberOfBooks <= 5)
    {
        return (price * numberOfBooks);
    }
    else if (numberOfBooks < 11 && numberOfBooks > 5)
    {
        return (numberOfBooks * (price - (price * 0.10)));
    }
    else if (numberOfBooks > 10 && numberOfBooks < 21)
    {
        return (numberOfBooks * (price -(price * 0.15)));
    }
    else
    {
        return (numberOfBooks * (price - (price * 0.20)));
    }

  }
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter the number of books purchased: ");
    int numberOfBooks = scanner.nextInt();
    double totalCost = BookDiscountCalculator.calculateTotalCost(numberOfBooks);
    System.out.println("Total Cost: $" + totalCost);
  }
}
```

```java
public class OutdoorActivityRecommender {
  // Method to recommend outdoor activity based on weather and user preferences
  public static void recommendOutdoorActivity(String weather, String activityPreference) {
        // System.out.print(weather);
        boolean w1 = weather.equals("sunny"), w2 = weather.equals("rainy"), w3 = weather.equals("snowy");
        boolean ap1 = activityPreference.equals("active"), ap2 = activityPreference.equals("relaxed");

        if(w1 && ap1)
        {
              System.out.println("Hiking");
        }
        else if ( w1 && ap2)
        {
              System.out.println("Picnic");
        }
        else if (w2 && ap1)
        {
              System.out.println("Play board games indoors");
        }
        else if ( w2 && ap2)
        {
              System.out.println("Read a book indoors");
        }
        else if (w3 && ap1)
        {
              System.out.println("Skiing");
```

```
        {
            System.out.println("Build a snowman");
        }
        else{
            System.out.println("Invalid weather conditions.");
        }


    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the current weather conditions (sunny/rainy/snowy): ");
        String weather = scanner.nextLine();
        System.out.print("Enter your activity preference (active/relaxed): ");
        String activityPreference = scanner.nextLine();
        recommendOutdoorActivity(weather, activityPreference);
    }
}
```

## Submitted Source Code :

```
//File Name: q22469/BudgetManager.java
//==================================
package q22469;
import java.util.Scanner;
public class BudgetManager {
   // Method to calculate savings based on monthly income and expenses
   public static double calculateSavings(double monthlyIncome, double fixedExpenses, double variableExpenses) {

         return monthlyIncome - fixedExpenses - variableExpenses;




 }
 public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter your monthly income: $");
    double monthlyIncome = scanner.nextDouble();
    System.out.print("Enter your fixed expenses: $");
    double fixedExpenses = scanner.nextDouble();
    System.out.print("Enter your variable expenses: $");
    double variableExpenses = scanner.nextDouble();
    double savings = calculateSavings(monthlyIncome, fixedExpenses, variableExpenses);
    System.out.println("Your savings for the month: $" + savings);
 }
```

| | | | |
|---|---|---|---|
| 6 | Compilation Errors | ✓ | 5.0 |

Home Learn Anywhere

## Submitted Source Code :

```
public class SimpleCalculator {
    public static void performOperation(double num1, double num2, char operator) {
        double result;
        switch (operator) {
            case '+':
                result = num1 + num2;
                break;
            case '-':
                result = num1 - num2;
                break;
            case '*':
                result = num1 * num2;
                break;
            case '/':
                if (num2 == 0){
                    System.out.println("Error: Division by zero");
                    return;
                }
                else{
                    result = num1 / num2;
                }
            default:
                return;
```

Compilation Errors

## Submitted Source Code :

```
switch (operator) {
    case '+':
        result = num1 + num2;
        break;
    case '-':
        result = num1 - num2;
        break;
    case '*':
        result = num1 * num2;
        break;
    case '/':
        if (num2 == 0){
            System.out.println("Error: Division by zero");
            return;
        }
        else{
            result = num1 / num2;
        }
    default:
        return;



}

System.out.println("Result of " + num1 + " " + operator + " " + num2 + ": " + result);
```

6 Compilation Errors ✓ 5.0

**Output format:**
Print the corresponding day of the week based on the day number.

Execution Results

## Submitted Source Code :

```
public static void checkDayOfWeek(int dayNumber) {
    System.out.print("The corresponding day of the week is: ");
    switch (dayNumber){
        case 1:
            System.out.println("Monday");
            break;
        case 2:
            System.out.println("Tuesday");
            break;
        case 3:
            System.out.println("Wednesday");
            break;
        case 4:
            System.out.println("Thursday");
            break;
        case 5:
            System.out.println("Friday");
            break;
        case 6:
            System.out.println("Saturday");
            break;
        case 7:
            System.out.println("Sunday");
            break;
        default:
            System.out.println("Invalid Day Number");
```

- If the time is between 12:00 AM and 6:00 AM (night), the user has **Very Low Energy**
- Other wise print **Invalid time input. Please enter a valid time (0-23)**

**Input format:**

The program should take a single input, which is the user's input for the current time (in 24-hour format).

**Output format:**

The program should output the user's energy level based on the provided time.

Execution Results

# Submitted Source Code :

```
// Method to analyze the user's energy level based on the time
public static void analyzeEnergyLevel(int time) {
        if (time > 5 && time < 12){
                System.out.println("High Energy");
        } else if ( time > 12 && time < 18)
        {
                System.out.println("Medium Energy");
        } else if (time > 18 &&  time < 24){
                System.out.println("Low Energy");
        } else if ( time > 0 && time < 6)
        {
                System.out.println("Very Low Energy");
        } else {
                System.out.println("Invalid time input. Please enter a valid time (0-23)");
        }

}
public static void main(String[] args) {
```

Submitted Source Code :

```
//File Name: q22424/DiscountEligibilityChecker.java
//==================================
package q22424;
import java.util.Scanner;
public class DiscountEligibilityChecker {
  // Method to check discount eligibility and calculate the discount percentage
  public static void checkDiscountEligibility(double purchaseAmount) {
        System.out.print("Discount Eligibility: ");
        if (purchaseAmount >= 100){
              System.out.println("Yes");
              System.out.println("Discount Percentage: 10%");
        }
        else if (purchaseAmount >= 50 && purchaseAmount < 100)
        {
              System.out.println("Yes");
              System.out.println("Discount Percentage: 5%");
        }
        else{
              System.out.println("No");
        }



  }
  public static void main(String[] args) {
```

Compilation Errors ✕ 3.5

## Submitted Source Code :

```
public class ZodiacSignDeterminer {
  public static void determineZodiacSign(int month, int day) {
    String zodiacSign;
    zodiacSign = "";
    if ((month == 1 && day >= 20) || (month == 2 && day <=18 ) ){
        zodiacSign = "Aquarius";
    }
    else if ((month == 2  && day > 18) || (month == 3 && day <=20)){
        zodiacSign = "Pisces";
    }
    else if ((month == 3 && day > 20) || (month == 4 && day <=19)){
        zodiacSign = "Aries";
    }
    else if ((month == 4 && day > 19) || (month == 5 && day <= 20)){
        zodiacSign = "Taurus";
    }
    else if ((month == 5 && day > 20) || (month == 6 && day <= 20)){
        zodiacSign = "Gemini";
    }
    else if ((month == 6 && day > 20 )|| (month == 7 && day <= 22)){
        zodiacSign = "Cancer";
    }
    else if ((month == 7 && day > 22) || (month == 8 && day <= 22)){
        zodiacSign = "Leo";
    }
    else if ((month == 8 && day > 22) || (month == 9 && day <= 22)){
```

None of the participants of this test answered this question correctly. For the solution, please consult your faculty.

0 Compilation Errors ✓ 5.0

## Submitted Source Code :

```
else if ((month == 6 && day > 20 )|| (month == 7 && day <= 22)){
        zodiacSign = "Cancer";
    }
    else if ((month == 7 && day > 22) || (month == 8 && day <= 22)){
        zodiacSign = "Leo";
    }
    else if ((month == 8 && day > 22) || (month == 9 && day <= 22)){
        zodiacSign = "Virgo";
    }
    else if((month == 9 && day > 22) || (month == 10 && day <= 22)){
        zodiacSign = "Libra";
    }
    else if((month == 10 && day > 22) || (month == 11 && day <= 21)){
        zodiacSign = "Scorpio";
    }
    else if((month == 11 && day > 21) || (month == 12 && day <= 21)){
        zodiacSign = "Sagittarius";
    }
    else if((month == 12 && day > 21 ) || (month == 1 && day < 20)){
        zodiacSign = "Capricorn";
    }
    System.out.println("Your zodiac sign is " + zodiacSign );

}
```

None of the participants of this test answered this question correctly. For the solution, please consult your faculty.

| | | | |
|---|---|---|---|
| 1.0 | Compilation Errors | ✓ | 5.0 |

Input format:
The program should take an integer representing the traffic light state.

**Output format:**
Print the corresponding action based on the traffic light state.

Execution Results

## Submitted Source Code :

```
//File Name: q22540/TrafficLightController.java
//==================================
package q22540;
import java.util.Scanner;
public class TrafficLightController {
    public static void controlTrafficLight(int lightState) {
        System.out.print("Traffic light is ");
        switch (lightState){
            case 1:
                System.out.println("Stop");
                break;
            case 2:
                System.out.println("Slow down");
                break;
            case 3:
                System.out.println("Go");
                break;
            default:
                System.out.println("Invalid Light State");
        }
```

## 1. Problem solving with if-else

01:28

te a class `IsWorkingDay` with a **main** method. The method eives one command line argument which is the name of the ekday.

te the missing code in the **If-else** statement such that if the ument passed to the **main** method is equal to **Sunday** the gram should print **Holiday**, otherwise it should print **Working** y.

Example:

```
md Args : Sunday
oliday
```

**te:** We use public boolean equals(Object anObject) method to ck if two strings have the same value. This method will be lained in more detail in later sections.

**te:** Please don't change the package name.

Sample Test Cases

Explorer

IsWorkin...

Submit

```
package q10844;
public class IsWorkingDay {
    public static void main(String[] args) {
        //Fill in the missing code using if-else statement
        if (args[0].equals("Sunday")){
            System.out.println("Holiday");
        }else{
            System.out.println("Working Day");
        }


    }
}
```

Plots

## 16.1.2. Problem solving with if-else

03:58

Create a class `IsWorkingDay` with a **main** method. The method receives one command line argument which is the name of the weekday. If the argument is equal to **Sunday** or **Saturday** the program should print **Holiday**, otherwise it should print **Working Day**.

For Example:

```
Cmd Args : Monday
Working Day
```

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

IsWorkin...

Submit

```
package q10845;
class IsWorkingDay
    {
        public static void main(String[] args)
        {
            if((args[0].equals("Sunday")) || (args[0].equals("Saturday"))){
                System.out.println("Holiday");
            }
            else{
                System.out.println("Working Day");
            }
        }
    }
```

Plots

< Prev | Reset | Submit | Next >

## 16.1.3. Problem solving with if-else

16:26

Create a class `SameNumbers` with a **public** method `checkNumbers` that takes two parameters `firstNo` and `secondNo` are of type `int` and returns true if `firstNo` and `secondNo` are equal. The return type of `checkNumbers` should be `boolean`.

For example:

```
Cmd Args : 10 10
true
```

```
Cmd Args : 2 3
false
```

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

SameNu... | SameNu... | Submit

```java
package q10864;

public class SameNumbers {

    public boolean checkNumbers(int firstNo, int secondNo) {
    {
        if(firstNo == secondNo){
            return true;
        }
        return false;

    }
    public void main(String[] args){
        int num1 = Integer.parseInt(args[0]);
        int num2 = Integer.parseInt(args[1]);
        boolean result = checkNumbers(num1, num2);
        System.out.println(result);
    }
}
```

< Prev | Reset | Submit | Next >

## 4. Problem solving with if-else

01:39

ite a class `PrintBiggerNumber` with a **public** method `ckNumbers` that takes two parameters `firstNo` and `ondNo` are of type `int` and returns true if `firstNo` is greater `secondNo`. The return type of `checkNumbers` should be `lean`.

se are examples for understanding:

```
md Args : 65 32
rue
```

```
md Args : 4 100
alse
```

**te:** Please don't change the package name.

Sample Test Cases

Explorer

PrintBigg... | PrintBigge...

Submit

Plots

```
package q10865;

public class PrintBiggerNumber {
    /**
     * Compare if firstNo is greater than secondNo are not
     *
     *
     * @return result
     *
     */

    public boolean checkNumbers(int firstNo, int secondNo) {

        //Write your code here
        return firstNo > secondNo;

    }
}
```

## 5. Problem solving with if-else

03:47

ate a class `PrintSum` with a **public** method `computeSum` that
es two parameters `firstNo` and `secondNo` are of type `int`.
program should print the sum of `firstNo` and `secondNo`. If
`rstNo` and `secondNo` are equal it should double the sum and
t the output.

ese are examples for understanding:

```
md Args : 23 5
8
```

```
md Args : 6 6
24
```

te: Please don't change the package name.

Sample Test Cases

Explorer

PrintSum... | PrintSum...

Submit

```
package q10920;
class PrintSum
    {
        public static void computeSum(int first, int second)
        {
            if (first == second){
                System.out.println(2*(first+second));
            } else{
                System.out.println(first+second);
            }
        }
    }
```

Plots

## 1.6. Problem solving with if-else

05:25

eate a class `PrintDiff` with a **public** method `computeDiff` at takes two parameters `firstNo` and `secondNo` are of type `nt`.

alculate the difference between `firstNo` and `secondNo`. If the lue of the difference is in between `-25` and `25`, the program ould **double** the value and print the output, otherwise it should int the difference as it is.

or example:

```
Cmd Args : 15 7
16
```

**ote:** Please don't change the package name.

Sample Test Cases

Explorer

PrintDiff.j... | PrintDiffM...

Submit

```java
package q10921;
public class PrintDiff {
    public void computeDiff(int firstNo, int secondNo) {
        int diff = firstNo - secondNo;
        if((diff > -25) && (diff < 25)){
            System.out.println(2*diff);
        }else{
            System.out.println(diff);
        }
    }
}
```

Plots

< Prev | Reset | Submit | Next >

## 16.1.7. Problem solving with if-else

01:55

Create a class `PrintWorkingHour` with a **public** method `checkWorkingHour` that takes one parameter `hour` of type `int`.

If the `hour` is in between `9` and `17` (both inclusive) the program should print **Working hour**, otherwise it should print **Non-working hour**.

For Example:

```
Cmd Args : 14
Working hour
```

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

PrintWor... PrintWorki...

Submit

```
package q10922;
public class PrintWorkingHour {
    public void checkWorkingHour(int hour) {
        //Write your code here
        if ((hour >=9)&&(hour <=17)){
            System.out.println("Working hour");
        }else{
            System.out.println("Non-working hour");
        }
    }
}
```

## 16.1.8. Problem solving with if-else

06:33

Create a class `CheckSum100` with a **public** method `checkSum` that takes two parameters `first` and `second` are of type `int`. If one of the numbers is **100** or the sum of the numbers is **100** the program should print **true**, otherwise it should print **false**.

For example:

```
Cmd Args : 15 100
true
```

**Note:** Make sure to use `println` and not `print` method.

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

CheckSu... CheckSu...

Submit

```
package q10923;
class CheckSum100
    {
        static public void checkSum(int num1, int num2)
            {
                if((num1 == 100)|| (num2 == 100) || ((num1 +num2)==100)){
                System.out.println("true");
                }else{
                System.out.println("false");
                }
            }
    }
```

## 16.1.9. Problem solving with if-else

06:13

Create a class `CheckPositiveNegative` with a **public** method `checkPositiveNegative` that takes two parameters `first` and `second` are of type `int`.

Follow the given instructions while writing the code

1. If both numbers are **negative**, the program should print **negative**
2. If both numbers are **positive**, the program should print **positive**
3. If one of the numbers is **positive** and the other is **negative**, the program should print **mixed**

For example:

```
Cmd Args : 1 2
positive
```

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

CheckPo... CheckPos... Submit

```java
package q10924;
class CheckPositiveNegative
    {
        public static void checkPositiveNegative(int first, int second)
        {
            if ((first < 0) && (second < 0)){
                System.out.println("negative");
            }else if ((first >0) && (second > 0)){
                System.out.println("positive");
            }else{
                System.out.println("mixed");
            }
        }
    }
```

< Prev Reset Submit Next >

## 16.1.10. Problem solving with if-else

01:56

Create a class `CheckMultipleOf` with a **public** method `checkMultipleOf` that takes two parameters `first` and `second` are of type `int` and returns true if `first` is multiple of `second`. The return type of `checkMultipleOf` is `boolean`.

Here is an example:

```
Cmd Args : 18 9
true
```

**Hint:** Use the % (modulus) operator. It can be used to find the remainder after division by a number.
For example, to check if a given number is even we write

```
if (number % 2 == 0) {// when divided by 2, if the reminder
    System.out.println("number is even");
}
```

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

CheckMu... CheckMul... Submit

```
package q10925;
class CheckMultipleOf
    {
        public static boolean
        checkMultipleOf(int f, int s)
        {
            return (f % s == 0);
        }
    }
```

## 16.1.11. Problem solving with if-else

02:17

Create a class `AgeCheck` with a **public** method `checkAge` that takes two parameters `age1` and `age2` are of type `int`. The program should print **Discount Price** if the `age1` is less than **10** and `age2` is greater than **60**, otherwise it should print **Full Price**.

For Example:

```
Cmd Args : 3 62
Discount Price
```

**Note:** Make sure to use `println` and not `print` method.

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

AgeChec... AgeCheck...

Submit

```
package q10926;
class AgeCheck{
    public static void checkAge(int a1, int a2)
    {
        if((a1 < 10) && (a2 > 60)){
            System.out.println("Discount Price");
        }else{
            System.out.println("Full Price");
        }
    }
}
```

## 16.1.12. Problem solving with if-else

06:03

Create a class `RangeCheck` with a **public** method `checkRange` that takes two parameters `first` and `second` are of type `int`. If one of the numbers is within the range of 25 to 35 (both inclusive) the program should print **Within range**, otherwise it should print **Out of range**.

For example:

```
Cmd Args : 25 79
Within range
```

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

RangeCh... | RangeCh...

Submit

```
package q10927;
class RangeCheck{
    public static void checkRange( int f, int s){
        if(((f>24) && (f<36)) || ((s > 24) && (s < 36))){
            System.out.println("Within range");
        }else{
            System.out.println("Out of range");
        }
    }
}
```

Plots

## 16.1.13. Understanding Counting

13:44

Click on Live Demo to understand how to write code which can perform basic counting.

Below program has a class `CountDemo` with a **main** method. The method receives **four** integer values as command line arguments. These **four** integers represent the marks in Maths, Science, Social and English.

Fill the missing code inside the **main** method such that the program should print the **passCount**. The **passCount** should reflect the count of subjects in which the marks scored is **greater than or equal to** `50`.

For example:

```
Cmd Args : 75 55 65 23
passCount = 3
```

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

CountDe...

Submit

```
package q10928;
public class CountDemo {
    static final int PASS_MARK = 50;
    public static void main(String[] args) {
        int passCount = 0;
        for( String i : args){
            if (Integer.parseInt(i) >= 50){
                passCount++;
            }
        }
        // Fill in the missing code here to count
        System.out.println("passCount = " + passCount);
    }
}
```

Plots

< Prev | Reset | Submit | Next >

## 16.1.14. Problem solving with if-else

03:48

Create a class `CountSeniorCitizens` with a **public** method `countSeniorCitizens` that takes four parameters `age1`, `age2`, `age3`, and `age4` are of type `int`. The program should print the count of numbers which are **greater** than **60**.

For Example:

```
Cmd Args : 60 61 59 58
1
```

[Hint: You can use multiple **if conditions** to keep track of the count.]

**Note:** Make sure to use `println` and not `print` method.

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

CountSe... CountMai...

Submit

```
package q10929;
class CountSeniorCitizens{
    public static void countSeniorCitizens(int a, int b, int c, int d){
        int count = 0;
        if(a>60){
            count++;
        }if(b>60){
            count++;
        }if(c> 60){
            count++;
        }if(d>60){
            count++;
        }
        System.out.println(count);
    }
}
```

Plots

< Prev Reset Submit Next >

## 16.1.15. Problem solving with if-else

05:12

Create a class `FindMax` with a **public** method `findMax` that takes three parameters `arg1`, `arg2`, and `arg3` are of type `int`. The program should print the largest of the given numbers.

For example:

```
Cmd Args : 10 20 5
20
```

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

FindMax.j... | FindMaxM...

Submit

Plots

```
package q10930;
class FindMax{
    public static void findMax(int a, int b, int c){
        if((a>b) && (a>c)){
            System.out.println(a);
        }else if ((b>a) && (b>c)){
            System.out.println(b);
        }else{
            System.out.println(c);
        }
    }
}
```

## 16.1.16. Problem solving with if-else

04:01

Create a class `FindNearestNumber` with a **public** method `findNearest` that takes two parameters `first` and `second` are of type `int`. Find the number that is closer to **100**, among **first** and **second**.

For example:

```
Cmd Args : 99 102
99
```

**Hint:** There is a useful method called Math.abs(int number) which returns an absolute value for the number passed.
For example:

```
Math.abs(-3) will return 3
Math.abs(4) will return 4
```

You may want to use it.

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

FindNear... FindNeare...

Submit

```
package q10931;
import java.util.*;
class FindNearestNumber{
    public static void findNearest(int n1, int n2){
        if(Math.abs(100-n1) > Math.abs(100-n2)){
            System.out.println(n2);
        }else{
            System.out.println(n1);
        }
    }
}
```

Plots

< Prev Reset Submit Next >

## 16.1.17. Problem solving with if-else

01:36

Create a class `NumberEndingCheck` with a **public** method `checkNumberEnding` that takes two parameters `first` and `second` are of type `int`. The program should print **true** if both the `first` and `second` are ends with the same digit otherwise, it should print **false**.

For Example:

```
Cmd Args : 55 75
true
```

**Hint:** The **%** operator can be used to find the remainder after division by a number.

**Note:** Please don't change the package name.

Sample Test Cases

Explorer

NumberE... NumberE...

Submit

```
package q10932;
class NumberEndingCheck{
    public static void checkNumberEnding(int f, int s){
        System.out.println((f%10)==(s%10));
    }
}
```

Plots

< Prev Reset Submit Next >

## 16.1.18. Problem solving with if-else

00:47

Given:

```
String[] elements = { "abc", "xyz", "pqr" };
String first = (elements.length > 0) ? elements[0] : null;
```

What is the result ?

- [ ] A compilation error occurs
- [ ] An exception is thrown at runtime
- [ ] The variable 'first' is set to null
- [x] The variable 'first' is set to "abc"

## 16.1.19. Choose correct answer

01:10

Given:

```
public class Test {
    public static void main(String[] args) {
        Boolean test = new Boolean(true);
        Integer x = 343;
        Integer y = new Test().go(test, x);
        System.out.println(y);
    }

    int go(Boolean b, int i) {
        if (b)
            return (i / 7);
        return (i / 49);
    }
}
```

What is the result ?

- 7
- ✓ 49
- 343
- Compilation fails.
- An exception is thrown at runtime.

< Prev | Reset | Submit | Next >