

Java Basic for Tester

Java Introduction



- *History of Java*
- *Features of Java Programming*
- *Java Hello World*
- *Setting up the environment in Java*
- *Java JVM, JRE and JDK*
- *Java Data Types*
- *Java Operators*
- *Java Input and Output*
- *Java Expressions & Blocks*
- *Java Comment*

Java is a powerful general-purpose programming language. It is used to develop desktop and mobile applications, big data processing, embedded systems, and so on.



History of Java

The Very first version was released on January 23, 1996. The principal stable variant, JDK 1.0.2, is called Java 1.

1995

1998

2000

2002

2004

2006

2011

2014

2017

2018

2018

2019

J2SE 1.3
May 2000

J2SE 5.0
September 2004

JAVA SE 7
July 2011

JAVA SE 9
September 2017

JAVA SE 11
September 2018

JDK 1.1
February 1997

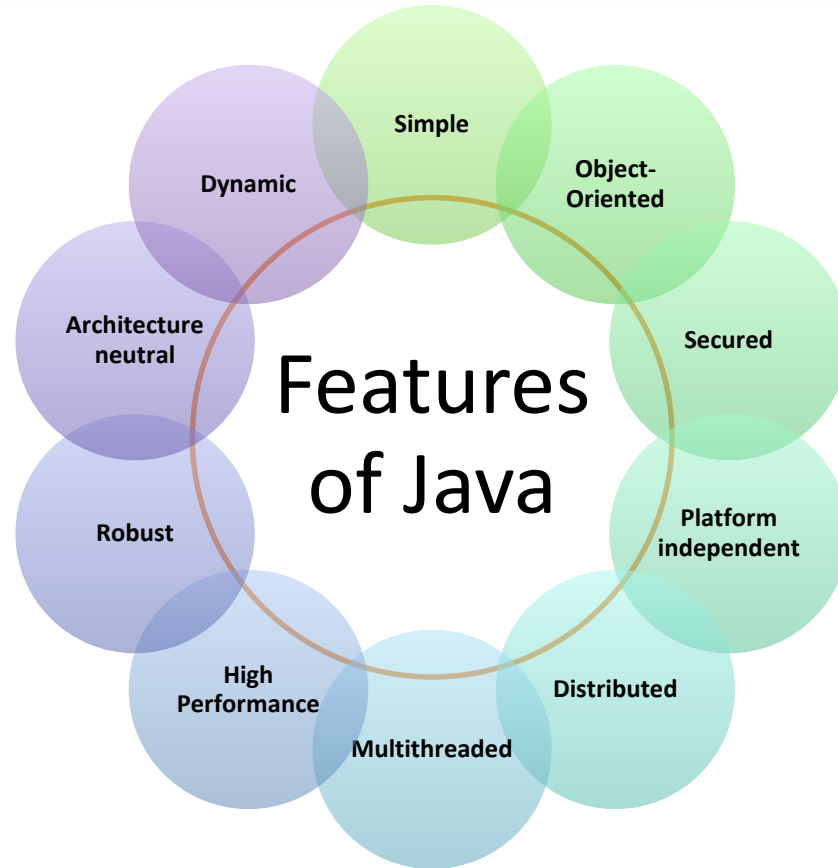
J2SE 1.4
February 2002

JAVA SE 6
December 2006

JAVA SE 8
March 2014

JAVA SE 10
March 2018

JAVA SE 12
March 2019



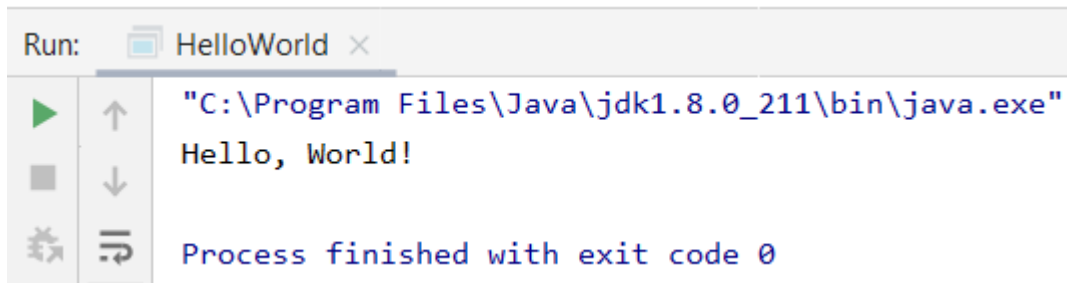
A "Hello, World!" is a simple program that outputs Hello, World! on the screen.

The requirement for Java Hello World Example:

1. Install the JDK if you don't have installed it, [download the JDK](#) and install it.
2. Set path of the jdk/bin directory.
3. Install IntelliJ IDEA Community Edition, [download and install it](#).
4. Create the java program
5. Compile and run the java program

Java "Hello, World!" Program

```
1 package JavaIntroduction;
2
3 // Your First Program
4 public class HelloWorld {
5     public static void main(String[] args) {
6         System.out.println("Hello, World!");
7     }
8 }
```



Run: HelloWorld x

"C:\Program Files\Java\jdk1.8.0_211\bin\java.exe"
Hello, World!

Process finished with exit code 0

How Java "Hello, World!" Program Works?

// Your First Program

In Java, any line starting with `//` is a comment.

Comments are intended for users reading the code to better understand the intent and functionality of the program.

It is completely ignored by the Java compiler (an application that translates Java program to Java bytecode that computer can execute).

```
1 package JavaIntroduction;
2
3 // Your First Program
4 public class HelloWorld {
5     public static void main(String[] args) {
6         System.out.println("Hello, World!");
7     }
8 }
```


How Java "Hello, World!" Program Works?

```
1 package JavaIntroduction;
2
3 // Your First Program
4 public class HelloWorld {
5     public static void main(String[] args) {
6         System.out.println("Hello, World!");
7     }
8 }
```

class HelloWorld { ... }

In Java, every application begins with a class definition.

In the program, HelloWorld is the name of the class, and the class definition is:

```
public class HelloWorld {

}
```

How Java "Hello, World!" Program Works?

```
1 package JavaIntroduction;  
2  
3 // Your First Program  
4 public class HelloWorld {  
5     public static void main(String[] args) {  
6         System.out.println("Hello, World!");  
7     }  
8 }
```

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

This is the main method. Every application in Java must contain the main method. The Java compiler starts executing the code from the main method.

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

How Java "Hello, World!" Program Works?

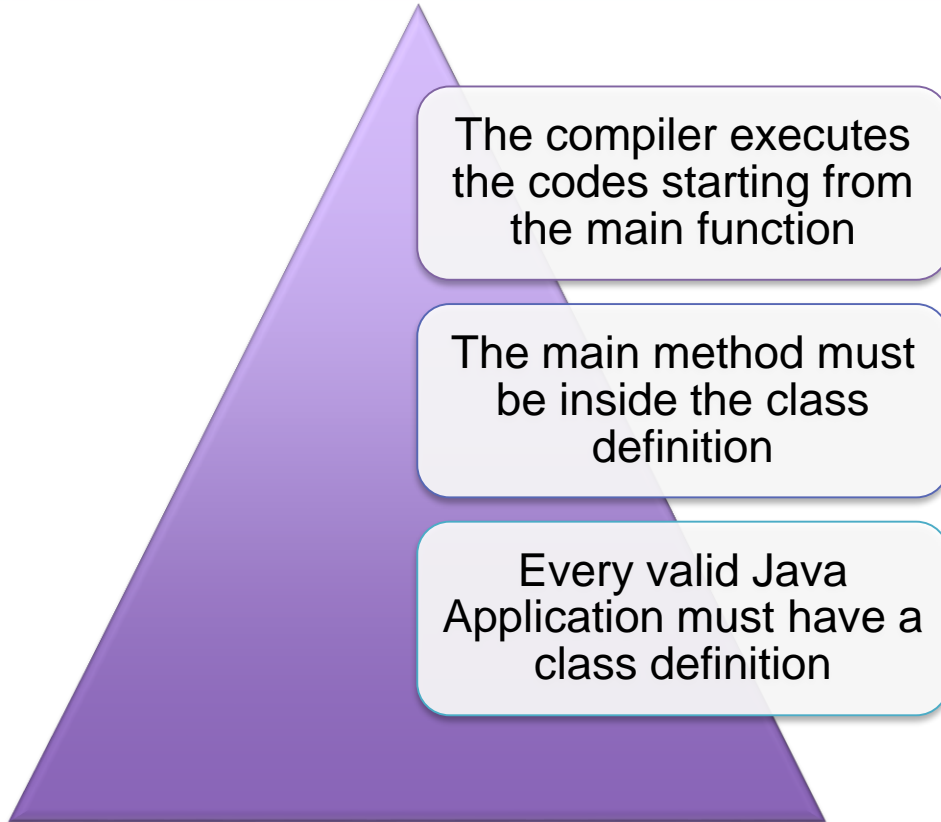
```
1 package JavaIntroduction;
2
3 // Your First Program
4 public class HelloWorld {
5     public static void main(String[] args) {
6         System.out.println("Hello, World!");
7     }
8 }
```

`System.out.println("Hello, World!");`

The following code prints the string inside quotation marks Hello, World! to standard output (your screen).

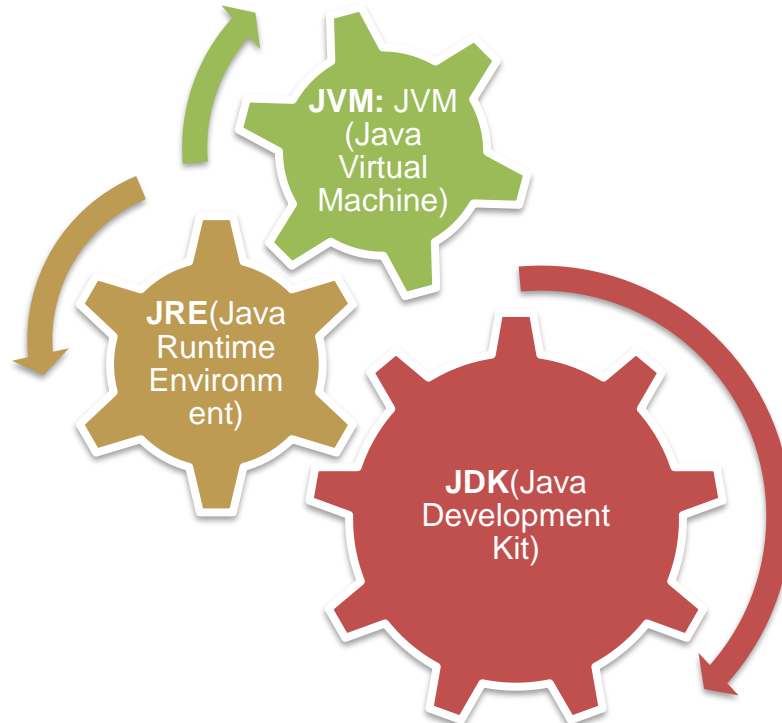
Notice, this statement is inside the main function, which is inside the class definition.

Notes





Setting up the environment in Java

There are few things which must be clear before setting up the environment



Step 1) Java JDK is available at [Download Java](#).

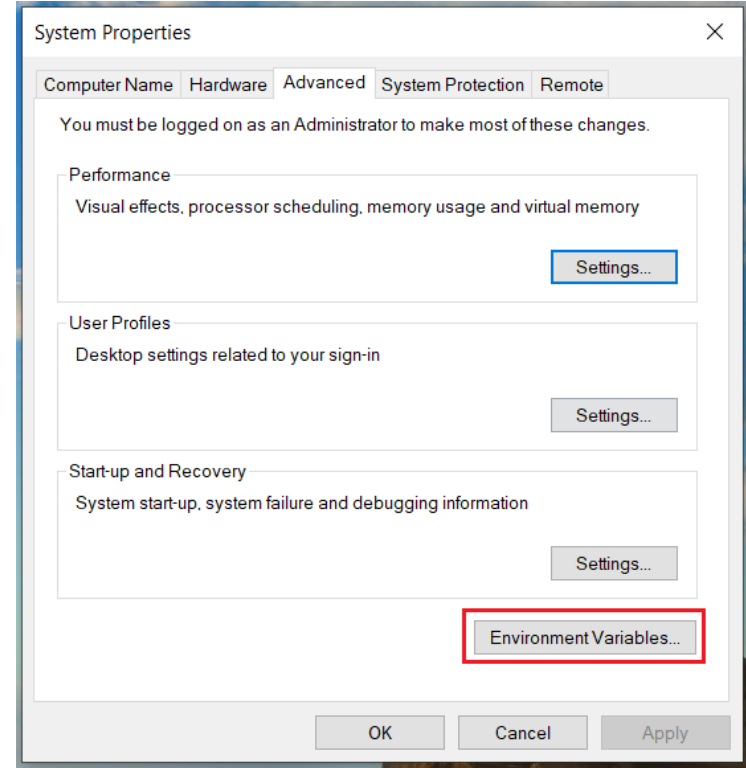
Click second last link for Windows(32 bit) and last link for Windows(64 bit) as highlighted below.

Windows x86	201.17 MB	 jdk-8u251-windows-i586.exe
Windows x64	211.54 MB	 jdk-8u251-windows-x64.exe

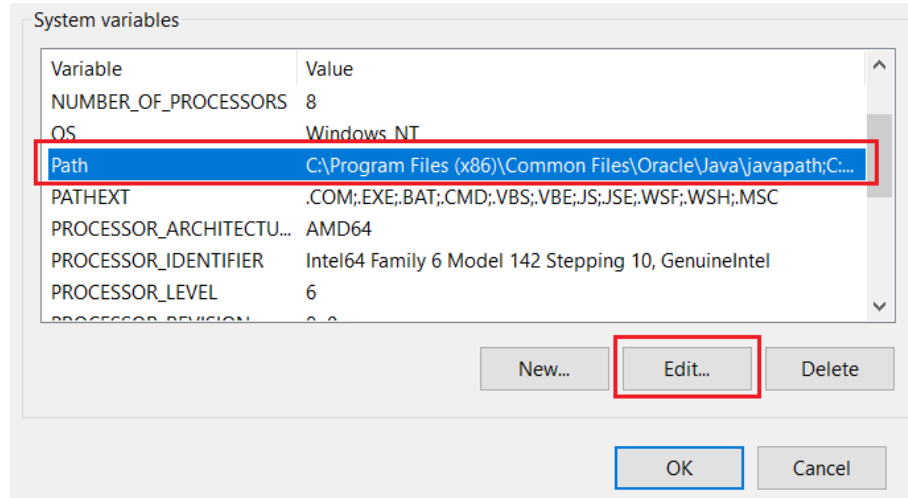
Setting up the environment in Java

Step 2) After download, run the .exe file and follow the instructions to install Java on your machine. Once you installed Java on your machine, you have to setup environment variable.

Step 3) Go to **Control Panel -> System and Security -> System**. Under **Advanced System Setting** option click on **Environment Variables** as highlighted below.



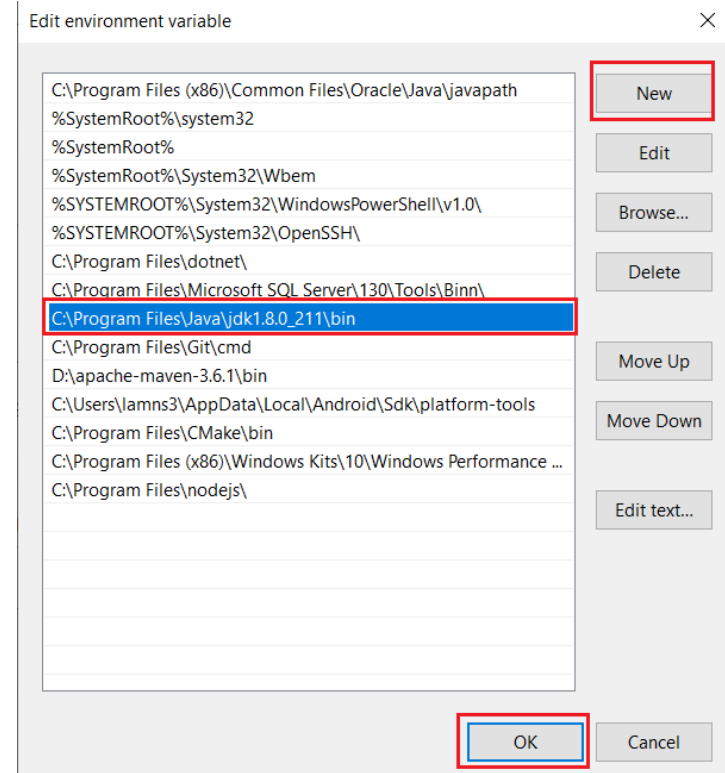
Step 4) Now, you have to alter the “Path” variable under **System variables** so that it also contains the path to the Java environment. Select the “**Path**” variable and click on Edit button as highlighted below.



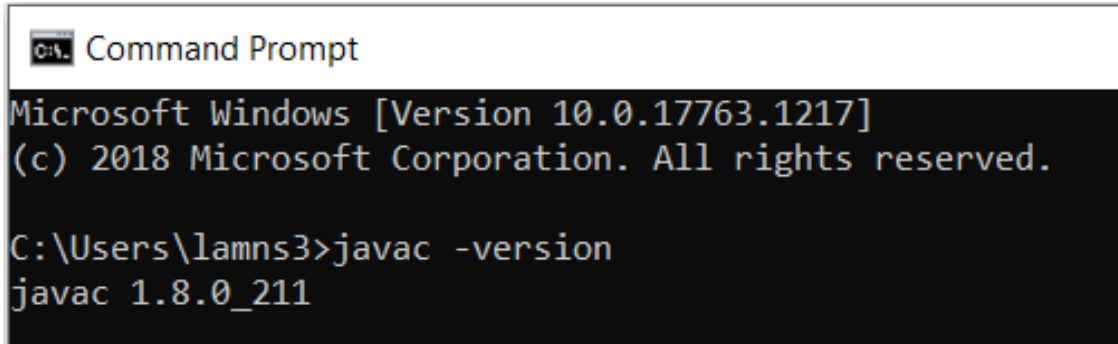
Setting up the environment in Java

Step 5) You will see list of different paths, click on New button and then add path where java is installed.

By default, java is installed in
“C:\Program Files\Java\jdk\bin” folder
OR “C:\Program Files(x86)\Java\jdk\bin”.
In case, you have installed java at any
other location, then add that path.



Step 6) Click on OK, Save the settings and you are done !! Now to check whether installation is done correctly, open command prompt and type ***javac -version***. You will see that java is running on your machine.



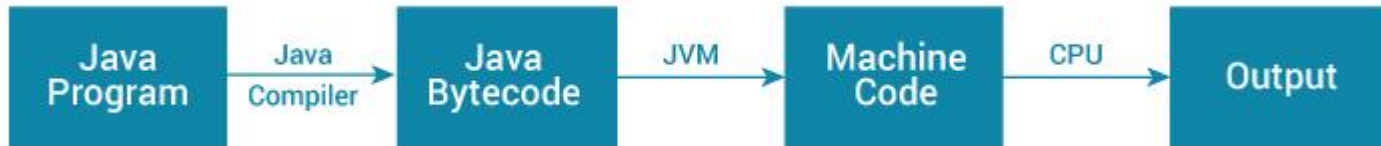
```
Command Prompt
Microsoft Windows [Version 10.0.17763.1217]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\lamns3>javac -version
javac 1.8.0_211
```

JVM (Java Virtual Machine) is an abstract machine that enables your computer to run a Java program.

When you run the Java program, Java compiler first compiles your Java code to bytecode. Then, the JVM translates bytecode into native machine code (set of instructions that a computer's CPU executes directly).

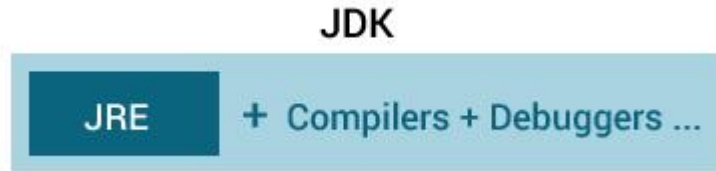
Java is a platform-independent language. It's because when you write Java code, it's ultimately written for JVM but not your physical machine (computer). Since JVM executes the Java bytecode which is platform-independent, Java is platform-independent.



JRE (Java Runtime Environment) is a software package that provides Java class libraries, Java Virtual Machine (JVM), and other components that are required to run Java applications.



JDK (Java Development Kit) is a software development kit required to develop applications in Java. When you download JDK, JRE is also downloaded with it.



Java Variables

A variable is a location in memory (storage area) to hold data. To indicate the storage area, each variable should be given a unique name (identifier).

How to declare variables in Java?

```
1 package JavaIntroduction;
2 // Your First Program
3 public class HelloWorld {
4     public static void main(String[] args) {
5         int year = 2020;
6         System.out.println("Hello, World!");
7     }
8 }
```

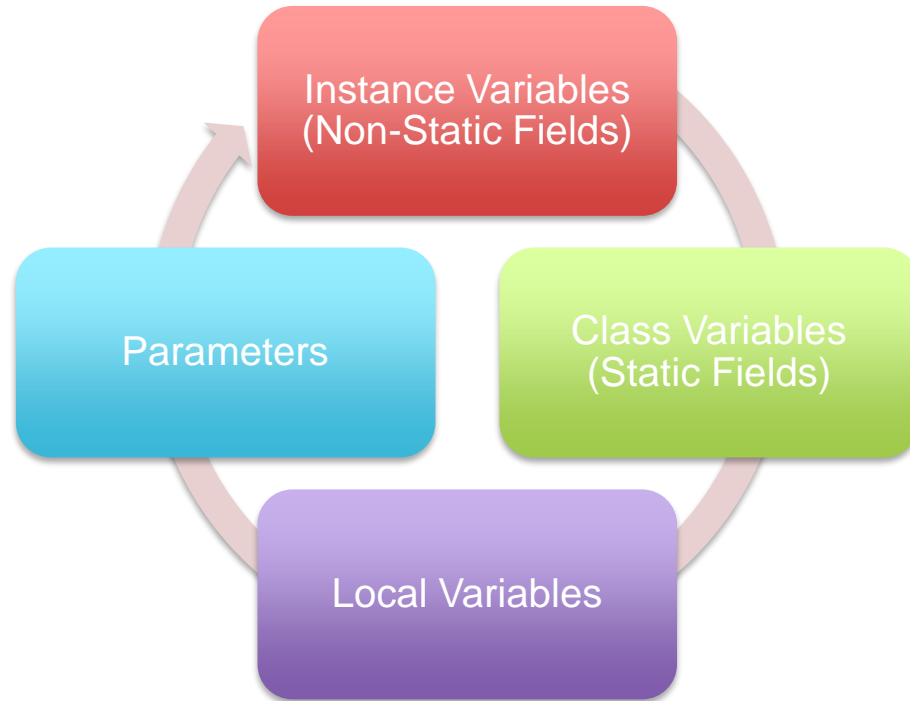
```
1 package JavaIntroduction;
2 // Your First Program
3 public class HelloWorld {
4     public static void main(String[] args) {
5         int year;
6         year = 2020;
7         System.out.println("Hello, World!");
8     }
9 }
```

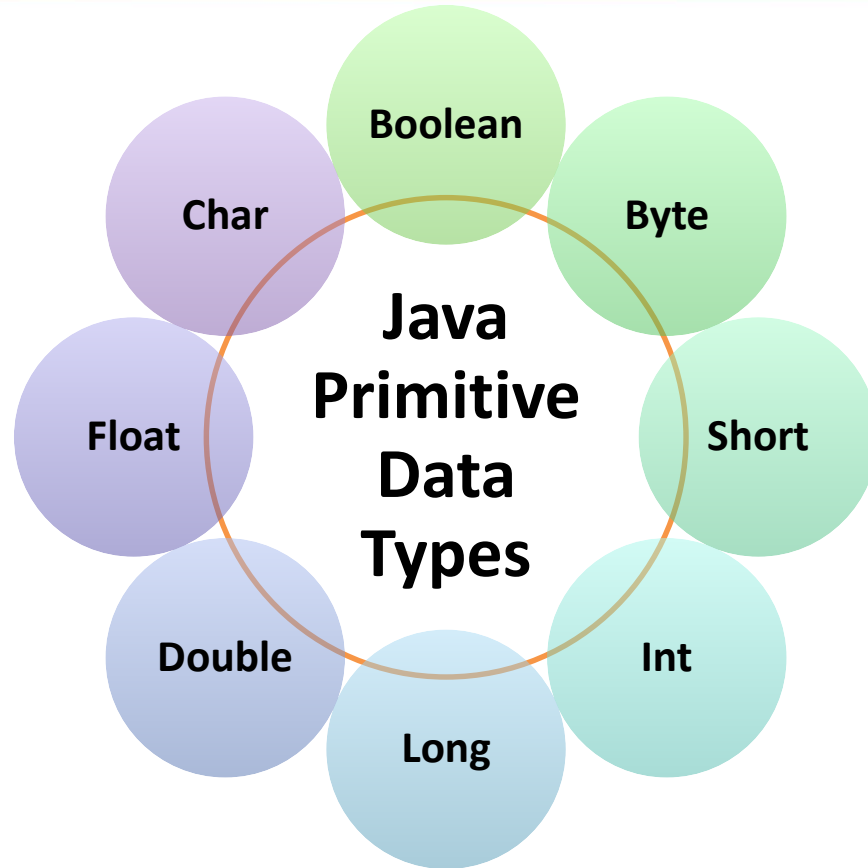
Rules for Naming Variables in Java

Java programming language has its own set of rules and conventions for naming variables. Here's what you need to know:

- Variables in Java are case-sensitive.
- A variable's name is a sequence of Unicode letters and digits. It can begin with a letter, **\$** or **_**. However, it's a convention to begin a variable name with a letter. Also, variable names cannot use whitespace in Java.
- When creating variables, choose a name that makes sense. For example: **score**, **number**, **level** makes more sense than variable names such as **s**, **n**, and **l**.
- If you choose one-word variable names, use all lowercase letters. For example, it's better to use **year** rather than **YEAR**, or **yEAR**.
- If you choose variable names having more than one word, use all lowercase letters for the first word and capitalize the first letter of each subsequent word. For example, **nextYear**.

There are 4 types of variables in Java programming language:

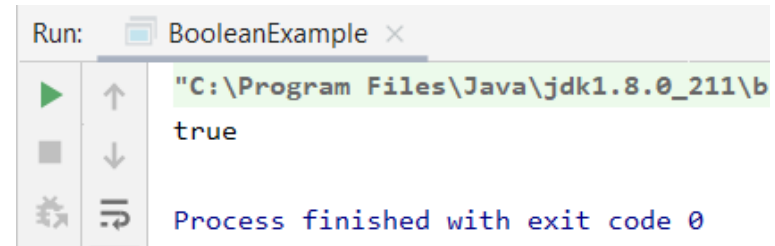




boolean

- The **boolean** data type has two possible values, either **true** or **false**.
- Default value: **false**.
- They are usually used for true/false conditions.

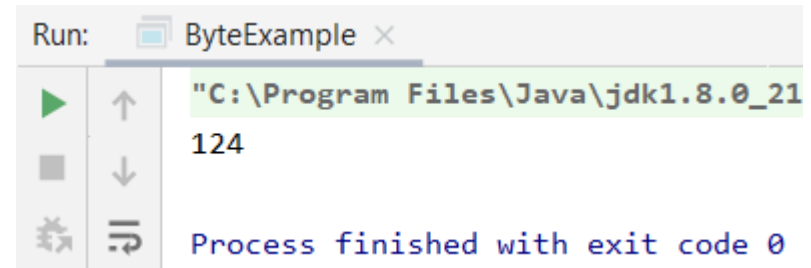
```
1 package JavaIntroduction;  
2  
3 public class BooleanExample {  
4     public static void main(String[] args) {  
5         boolean flag = true;  
6         System.out.println(flag);  
7     }  
8 }
```



byte

- The byte data type can have values from **-128** to **127** (8-bit signed two's complement integer).
- It's used instead of **int** or other integer data types to save memory if it's certain that the value of a variable will be within [-128, 127].
- Default value: **0**

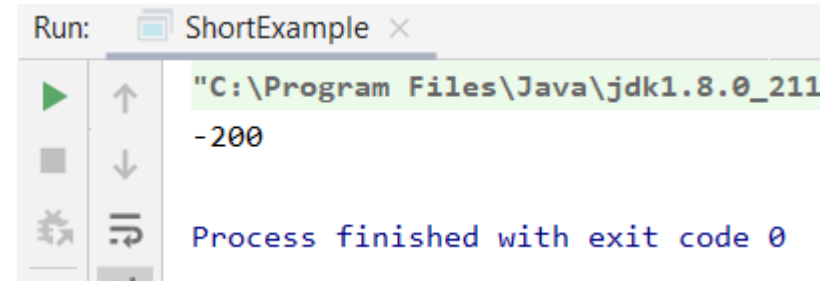
```
1 package JavaIntroduction;
2
3 public class ByteExample {
4     public static void main(String[] args) {
5         byte range;
6         range = 124;
7         System.out.println(range);
8     }
9 }
```



short

- The short data type can have values from **-32768** to **32767** (16-bit signed two's complement integer).
- It's used instead of other integer data types to save memory if it's certain that the value of the variable will be within [-32768, 32767].
- Default value: **0**

```
1  package JavaIntroduction;
2
3  public class ShortExample {
4      public static void main(String[] args) {
5
6          short temperature;
7          temperature = -200;
8          System.out.println(temperature);
9      }
10 }
```

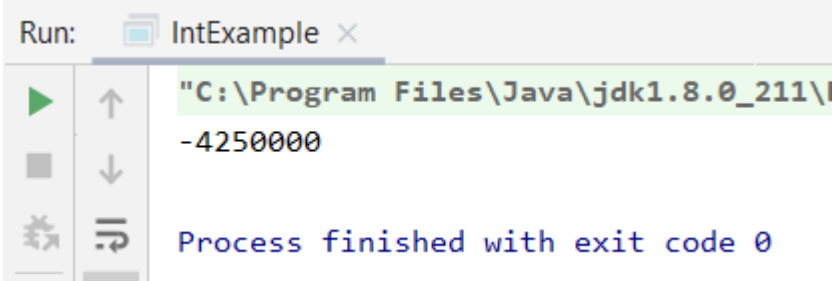


```
Run: ShortExample x
"C:\Program Files\Java\jdk1.8.0_211"
-200
Process finished with exit code 0
```

int

- The int data type can have values from **-231** to **231-1** (32-bit signed two's complement integer).
- If you are using Java 8 or later, you can use unsigned 32-bit integer with a minimum value of 0 and a maximum value of 232-1.
- Default value: **0**

```
1 package JavaIntroduction;  
2  
3 public class IntExample {  
4     public static void main(String[] args) {  
5  
6         int range = -4250000;  
7         System.out.println(range);  
8     }  
9 }
```

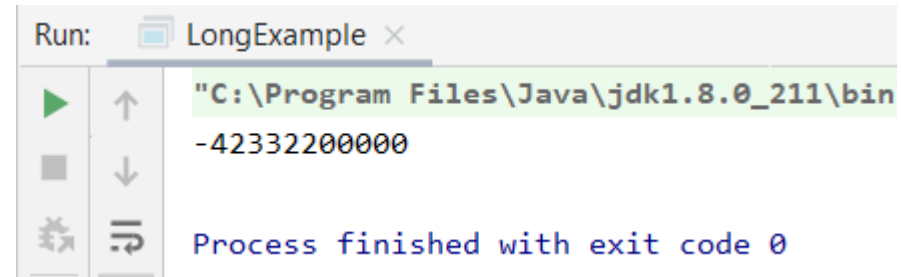


```
Run: IntExample x  
"C:\Program Files\Java\jdk1.8.0_211\  
-4250000  
Process finished with exit code 0
```

long

- The long data type can have values from **-2⁶³** to **2⁶³-1** (64-bit signed two's complement integer).
- If you are using Java 8 or later, you can use unsigned 64-bit integer with a minimum value of 0 and a maximum value of 2⁶⁴-1.
- Default value: **0**

```
1 package JavaIntroduction;
2
3 public class LongExample {
4     public static void main(String[] args) {
5
6         long range = -42332200000L;
7         System.out.println(range);
8     }
9 }
```



Run: LongExample x

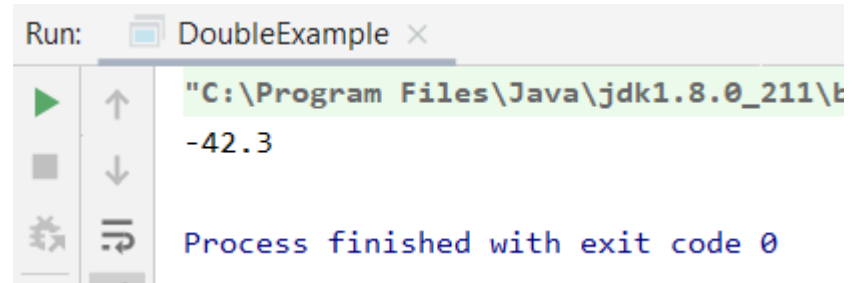
"C:\Program Files\Java\jdk1.8.0_211\bin
-42332200000

Process finished with exit code 0

double

- The double data type is a double-precision **64-bit** floating-point.
- It should never be used for precise values such as currency.
- Default value: **0.0 (0.0d)**

```
1 package JavaIntroduction;
2
3 public class DoubleExample {
4     public static void main(String[] args) {
5
6         double number = -42.3;
7         System.out.println(number);
8     }
9 }
```

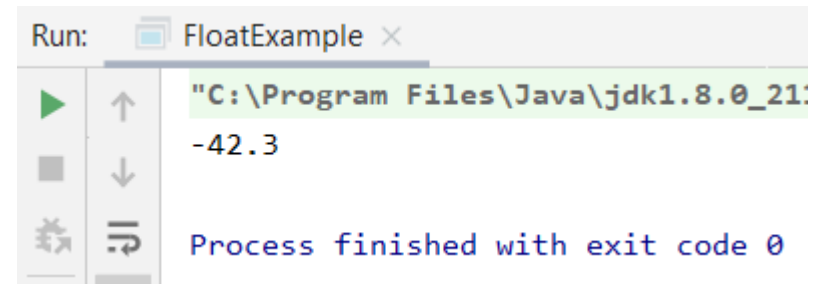


```
Run: DoubleExample x
"C:\Program Files\Java\jdk1.8.0_211\bin\java.exe" -Djava.class.path=. DoubleExample
-42.3
Process finished with exit code 0
```

float

- The float data type is a single-precision **32-bit** floating-point.
- It should never be used for precise values such as currency.
- Default value: **0.0 (0.0f)**

```
1 package JavaIntroduction;
2
3 public class FloatExample {
4     public static void main(String[] args) {
5
6         float number = -42.3f;
7         System.out.println(number);
8     }
9 }
```

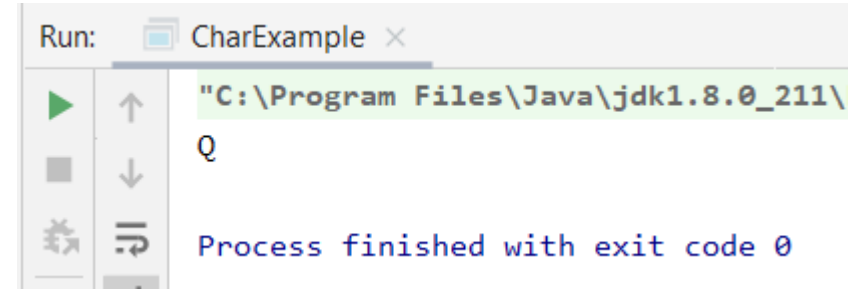


```
Run: FloatExample x
"C:\Program Files\Java\jdk1.8.0_21:
-42.3
Process finished with exit code 0
```


char

- It's a **16-bit** Unicode character.
- The minimum value of the char data type is **'\u0000'** (0). The maximum value of the char data type is **'\uffff'**.
- Default value: **'\u0000'**

```
1 package JavaIntroduction;  
2  
3 public class CharExample {  
4     public static void main(String[] args) {  
5  
6         char letter = '\u0051';  
7         System.out.println(letter);  
8     }  
9 }
```



Operators are special symbols (characters) that carry out operations on operands (variables and values).

For example, **+** is an operator that performs addition.

Assignment Operator

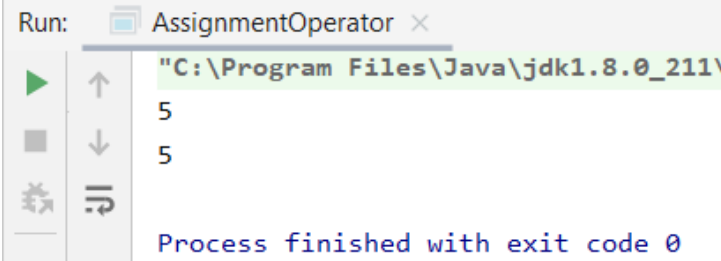
Assignment operators are used in Java to assign values to variables. For example,

```
int age;  
age = 5;
```

The assignment operator assigns the value on its right to the variable on its left. Here, **5** is assigned to the variable **age** using **= operator**.

Assignment Operator

```
1 package JavaIntroduction;
2
3 public class AssignmentOperator {
4     public static void main(String[] args) {
5
6         int number1, number2;
7
8         // Assigning 5 to number1
9         number1 = 5;
10        System.out.println(number1);
11
12        // Assigning value of variable number2 to number1
13        number2 = number1;
14        System.out.println(number2);
15    }
16 }
```



Run: AssignmentOperator ×

"C:\Program Files\Java\jdk1.8.0_211\
5
5

Process finished with exit code 0

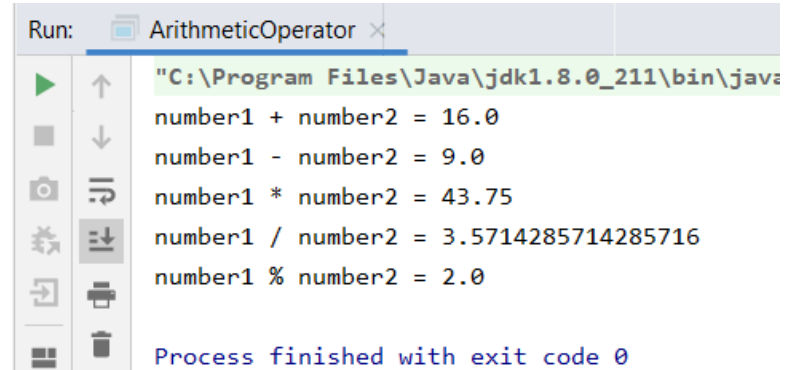
Arithmetic Operators

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication, etc.

Operator	Meaning
+	Addition (also used for string concatenation)
-	Subtraction Operator
*	Multiplication Operator
/	Division Operator
%	Remainder Operator

Arithmetic Operators

```
1 package JavaIntroduction;
2
3 public class ArithmeticOperator {
4     public static void main(String[] args) {
5
6         double number1 = 12.5, number2 = 3.5, result;
7
8         // Using addition operator
9         result = number1 + number2;
10        System.out.println("number1 + number2 = " + result);
11
12        // Using subtraction operator
13        result = number1 - number2;
14        System.out.println("number1 - number2 = " + result);
15
16        // Using multiplication operator
17        result = number1 * number2;
18        System.out.println("number1 * number2 = " + result);
19
20        // Using division operator
21        result = number1 / number2;
22        System.out.println("number1 / number2 = " + result);
23
24        // Using remainder operator
25        result = number1 % number2;
26        System.out.println("number1 % number2 = " + result);
27    }
28 }
```



Run: ArithmeticOperator x

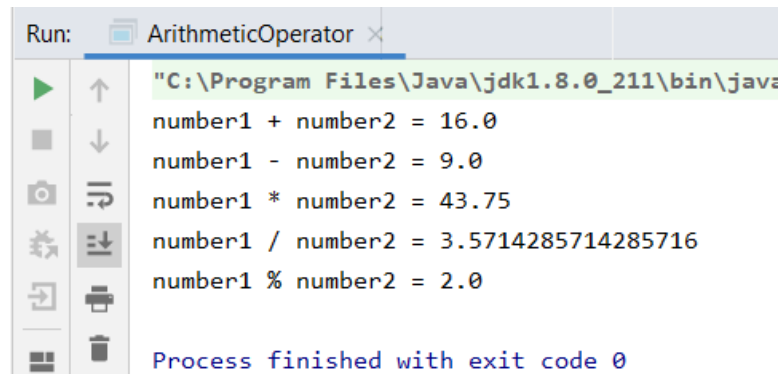
"C:\Program Files\Java\jdk1.8.0_211\bin\java

number1 + number2 = 16.0
number1 - number2 = 9.0
number1 * number2 = 43.75
number1 / number2 = 3.5714285714285716
number1 % number2 = 2.0

Process finished with exit code 0

Arithmetic Operators

```
1 package JavaIntroduction;
2
3 public class ArithmeticOperator {
4     public static void main(String[] args) {
5
6         double number1 = 12.5, number2 = 3.5, result;
7
8         // Using addition operator
9         result = number1 + number2;
10        System.out.println("number1 + number2 = " + result);
11
12        // Using subtraction operator
13        result = number1 - number2;
14        System.out.println("number1 - number2 = " + result);
15
16        // Using multiplication operator
17        result = number1 * number2;
18        System.out.println("number1 * number2 = " + result);
19
20        // Using division operator
21        result = number1 / number2;
22        System.out.println("number1 / number2 = " + result);
23
24        // Using remainder operator
25        result = number1 % number2;
26        System.out.println("number1 % number2 = " + result);
27    }
28 }
```



Run: ArithmeticOperator

"C:\Program Files\Java\jdk1.8.0_211\bin\java

number1 + number2 = 16.0
number1 - number2 = 9.0
number1 * number2 = 43.75
number1 / number2 = 3.5714285714285716
number1 % number2 = 2.0

Process finished with exit code 0

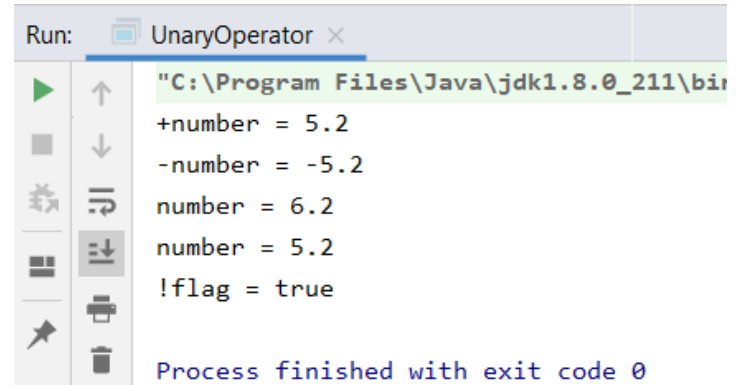
Unary Operators

The unary operator performs operations on only one operand.

Operator	Meaning
+	Unary plus (not necessary to use since numbers are positive without using it)
-	Unary minus: inverts the sign of an expression
++	Increment operator: increments value by 1
--	decrement operator: decrements value by 1
!	Logical complement operator: inverts the value of a boolean

Unary Operators

```
1 package JavaIntroduction;
2
3 public class UnaryOperator {
4     public static void main(String[] args) {
5
6         double number = 5.2, resultNumber;
7         boolean flag = false;
8
9         System.out.println("+number = " + +number);
10        // number is equal to 5.2 here.
11
12        System.out.println("-number = " + -number);
13        // number is equal to 5.2 here.
14
15        // ++number is equivalent to number = number + 1
16        System.out.println("number = " + ++number);
17        // number is equal to 6.2 here.
18
19        // -- number is equivalent to number = number - 1
20        System.out.println("number = " + --number);
21        // number is equal to 5.2 here.
22
23        System.out.println("!flag = " + !flag);
24        // flag is still false.
25    }
26 }
```



Run: UnaryOperator x

"C:\Program Files\Java\jdk1.8.0_211\bin

+number = 5.2
-number = -5.2
number = 6.2
number = 5.2
!flag = true

Process finished with exit code 0

Unary Operators

You can also use **++** and **--** operator as both **prefix** and **postfix** in Java. The **++** operator **increases** value by **1** and **--** operator **decreases** the value by **1**.

```
int myInt = 5;  
++myInt // myInt becomes 6  
myInt++ // myInt becomes 7  
--myInt // myInt becomes 6  
myInt-- // myInt becomes 5
```

Equality and Relational Operators

The equality and relational operators determine the relationship between the two operands. It checks if an operand is greater than, less than, equal to, not equal to and so on. Depending on the relationship, it is evaluated to either true or false.

Operator	Description	Example
==	equal to	5 == 3 is evaluated to false
!=	not equal to	5 != 3 is evaluated to true
>	greater than	5 > 3 is evaluated to true
<	less than	5 < 3 is evaluated to false
>=	greater than or equal to	5 >= 5 is evaluated to true
<=	less than or equal to	5 <= 5 is evaluated to true

Equality and Relational Operators

```
1  package JavaIntroduction;
2
3  ▶ public class RelationalOperator {
4  ▶     public static void main(String[] args) {
5
6         int number1 = 5, number2 = 6;
7
8         if (number1 > number2) {
9             System.out.println("number1 is greater than number2.");
10        }
11        else {
12            System.out.println("number2 is greater than number1.");
13        }
14    }
15 }
16
```

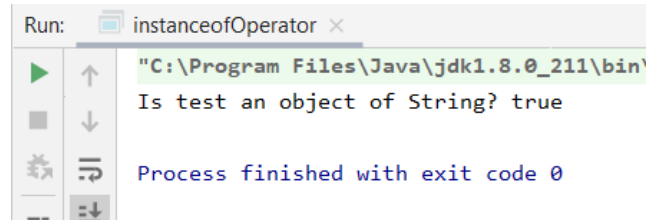
Run: RelationalOperator ×

```
▶  "C:\Program Files\Java\jdk1.8.0_211\bin\java
  number2 is greater than number1.
  Process finished with exit code 0
```

instanceof Operator

In addition to relational operators, there is also a type comparison operator **instanceof** which compares an object to a specified type.

```
1 package JavaIntroduction;
2
3 public class instanceofOperator {
4     public static void main(String[] args) {
5
6         String test = "lamns3";
7         boolean result;
8
9         result = test instanceof String;
10        System.out.println("Is test an object of String? " + result);
11    }
12 }
```



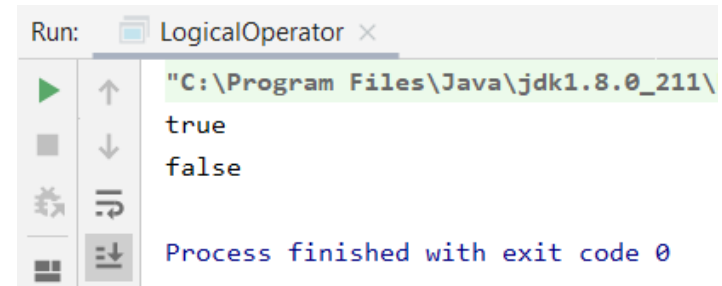
Logical Operators

The logical operators `||` (conditional-OR) and `&&` (conditional-AND) operate on boolean expressions.

Operator	Description	Example
<code> </code>	conditional-OR: true if either of the boolean expression is true	false true is evaluated to true
<code>&&</code>	conditional-AND: true if all boolean expressions are true	false && true is evaluated to false

Logical Operators

```
1 package JavaIntroduction;
2
3 public class LogicalOperator {
4     public static void main(String[] args) {
5
6         int number1 = 1, number2 = 2, number3 = 9;
7         boolean result;
8
9         // At least one expression needs to be true for the result to be true
10        result = (number1 > number2) || (number3 > number1);
11
12        // result will be true because (number1 > number2) is true
13        System.out.println(result);
14
15        // All expression must be true from result to be true
16        result = (number1 > number2) && (number3 > number1);
17
18        // result will be false because (number3 > number1) is false
19        System.out.println(result);
20    }
21 }
```



The screenshot shows the 'Run' console of an IDE. The title bar says 'Run: LogicalOperator x'. The console output shows the execution path: 'C:\Program Files\Java\jdk1.8.0_211\' followed by 'true' and 'false' on separate lines. At the bottom, it states 'Process finished with exit code 0'. On the left side of the console, there are standard IDE icons for running, stepping through, and debugging the code.

Ternary Operator

The conditional operator or ternary operator **?:** is shorthand for the **if-then-else** statement. The syntax of the conditional operator is:

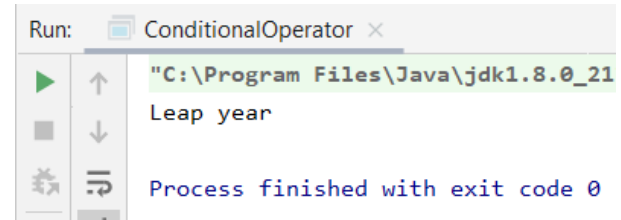
variable = Expression ? expression1 : expression2

Here's how it works.

- If the Expression is true, expression1 is assigned to the variable.
- If the Expression is false, expression2 is assigned to the variable.

Ternary Operator

```
1 package JavaIntroduction;
2
3 public class ConditionalOperator {
4     public static void main(String[] args) {
5
6         int februaryDays = 29;
7         String result;
8
9         result = (februaryDays == 28) ? "Not a leap year" : "Leap year";
10        System.out.println(result);
11    }
12 }
```



Java Output

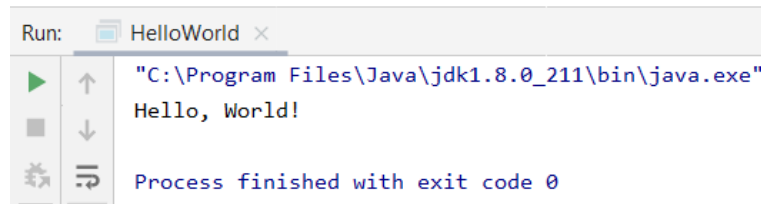
System.out.println(); or

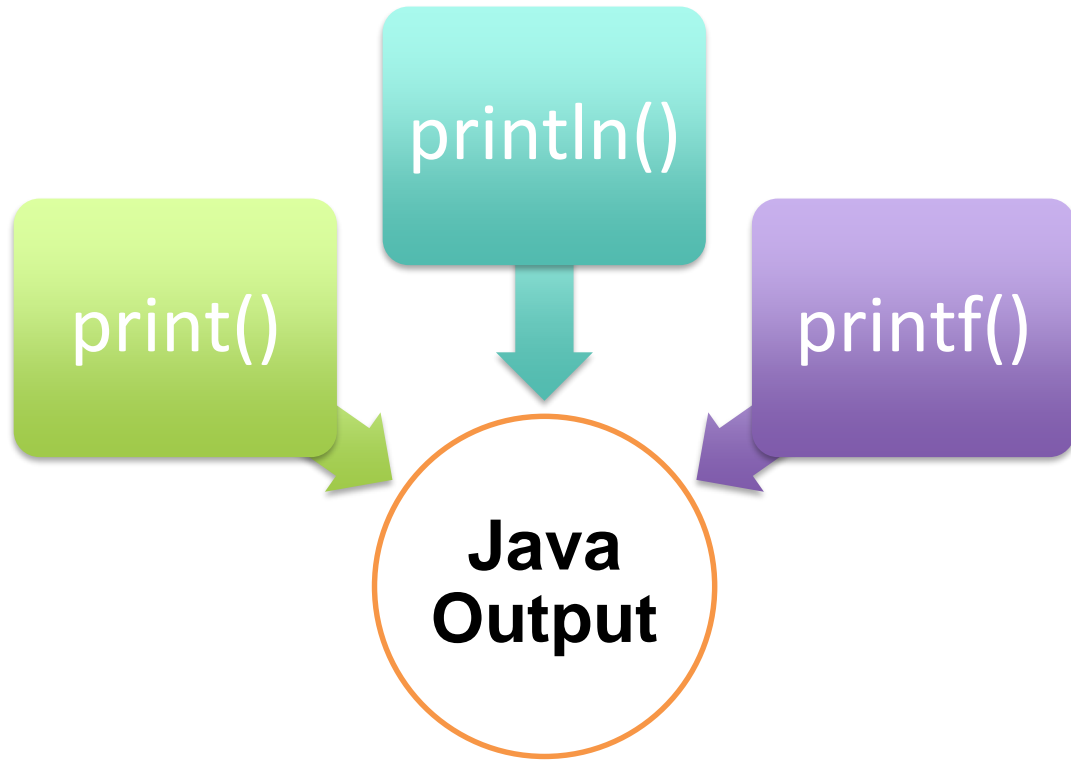
System.out.print(); or

System.out.printf();

```
1 package JavaIntroduction;
2
3 // Your First Program
4 public class HelloWorld {
5     public static void main(String[] args) {
6         System.out.println("Hello, World!");
7     }
8 }
```

to send output to standard output (screen).





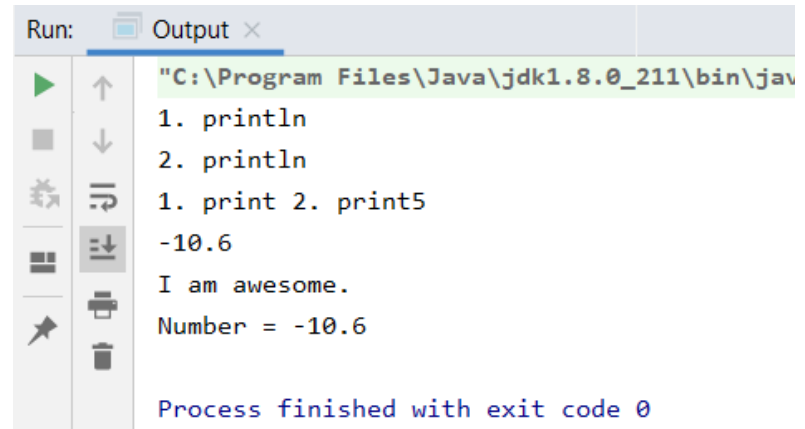
print() - It prints string inside the quotes.

println() - It prints string inside the quotes similar like `print()` method. Then the cursor moves to the beginning of the next line.

printf() - Tt provides string formatting

Java Output

```
1 package JavaIntroduction;
2
3 public class Output {
4     public static void main(String[] args) {
5
6         System.out.println("1. println ");
7         System.out.println("2. println ");
8
9         System.out.print("1. print ");
10        System.out.print("2. print");
11
12        Double number = -10.6;
13
14        System.out.println(5);
15        System.out.println(number);
16
17        Double number1 = -10.6;
18
19        System.out.println("I am " + "awesome.");
20        System.out.println("Number = " + number1);
21
22    }
23 }
```



Run: Output x

"C:\Program Files\Java\jdk1.8.0_211\bin\jav

1. println
2. println
1. print 2. print5
-10.6
I am awesome.
Number = -10.6

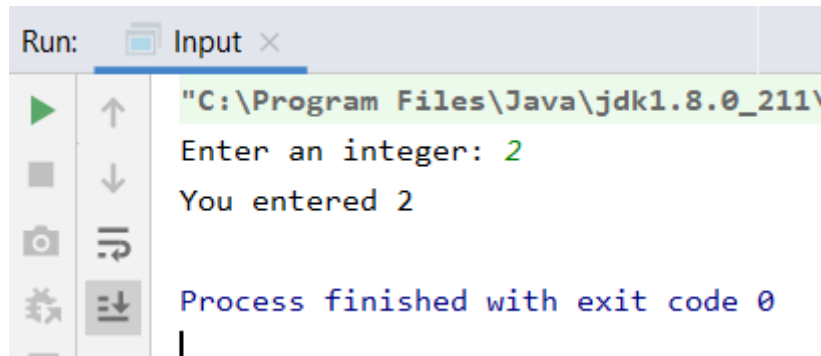
Process finished with exit code 0

Java Input

```
// create an object of Scanner  
Scanner input = new Scanner(System.in);  
  
// take input from the user  
int number = input.nextInt();
```

Java Input

```
1 package JavaIntroduction;
2
3 import java.util.Scanner;
4
5 public class Input {
6     public static void main(String[] args) {
7
8         Scanner input = new Scanner(System.in);
9
10        System.out.print("Enter an integer: ");
11        int number = input.nextInt();
12        System.out.println("You entered " + number);
13
14        // closing the scanner object
15        input.close();
16    }
17 }
```



Run: Input x

"C:\Program Files\Java\jdk1.8.0_211\
Enter an integer: 2
You entered 2

Process finished with exit code 0

Java Input

```
1 package JavaIntroduction;
2
3 import java.util.Scanner;
4
5 public class Input {
6     public static void main(String[] args) {
7
8         Scanner input = new Scanner(System.in);
9
10        System.out.print("Enter an integer: ");
11        int number = input.nextInt();
12        System.out.println("You entered " + number);
13
14        // closing the scanner object
15        input.close();
16    }
17 }
```

Similarly, we can use **nextLong()**, **nextFloat()**, **nextDouble()**, and **next()** methods to get **long**, **float**, **double**, and **string** input respectively from the user.

Java Expressions

A Java expression consists of variables, operators, literals, and method calls.

```
int score;
```

```
score = 90;
```

```
Double a = 2.2, b = 3.4, result;
```

```
result = a + b - 3.4;
```

```
if (number1 == number2)
```

```
    System.out.println("Number 1 is larger than number 2");
```


Java Statements

In Java, each statement is a complete unit of execution.

```
int score = 9*5;
```

Here, we have a statement. The complete execution of this statement involves **multiplying** integers **9** and **5** and then assigning the result to the variable **score**.

Java Statements

In Java, each statement is a complete unit of execution.

```
int score = 9*5;
```

Here, we have a statement. The complete execution of this statement involves **multiplying** integers **9** and **5** and then assigning the result to the variable **score**.

Expression statements

We can convert an expression into a statement by terminating the expression with a ;

```
// expression  
number = 10  
// statement  
number = 10;
```

Java Blocks

A block is a group of statements (zero or more) that is enclosed in curly braces **{ }**

```
1  package JavaIntroduction;
2
3  // Your First Program
4  ▶ public class HelloWorld {
5  ▶     public static void main(String[] args) {
6      System.out.println("Hello, World!");
7      }
8  }
```

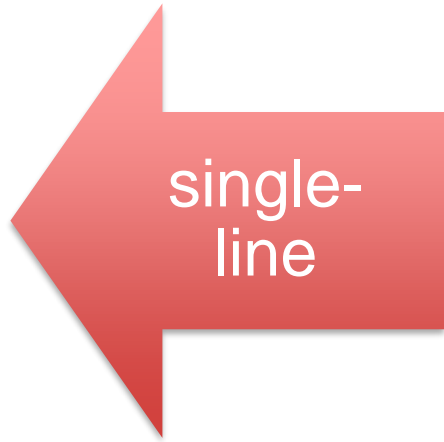
Java Blocks

However, a block may not have any statements.

```
class Main {  
    public static void main(String[] args) {  
  
        if (10 > 5) { // start of block  
  
        } // end of block  
    }  
}
```

In computer programming, comments are a portion of the program that are completely ignored by Java compilers. They are mainly used to help programmers to understand the code.

```
// declare and initialize two variables  
int a =1;  
int b = 3;  
  
// print the output  
System.out.println("This is output");
```



A **single-line** comment starts and ends in the same line. To write a single-line comment, we can use the **// symbol**.

When we want to write comments in **multiple lines**, we can use the multi-line comment. To write multi-line comments, we can use the **/*....*/ symbol**.

```
1  package JavaIntroduction;
2  /* This is an example of multi-line comment.
3     * The program prints "Hello, World!" to the standard output.
4  */
5
6  // Your First Program
7  public class HelloWorld {
8      public static void main(String[] args) {
9          System.out.println("Hello, World!");
10     }
11 }
```


Thank you

