Contents

What is java?

Java is a class[1]-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is intended to let application developers write once, and run anywhere (WORA), meaning that compiled[2] Java code can run on all platforms that support Java without the need for recompilation. Java was first released in 1995 and is widely used for developing applications for desktop, web, and mobile devices. Java is known for its simplicity, robustness [3], and security features, making it a popular choice for enterprise-level applications.

Java was developed by James Gosling at Sun Microsystems Inc in May 1995 and later acquired by Oracle Corporation. It is a simple programming language. Java makes writing, compiling[4], and debugging[5] programming easy. It helps to create reusable code and modular programs[6]. Java is a class-based, object-oriented programming language and is designed to have as few implementation dependencies as possible. A general-purpose programming language made for developers to write once run anywhere that is compiled Java code can run on all platforms that support Java. Java applications are compiled to byte code that can run on any Java Virtual Machine. The syntax of Java is similar to C/C++.

[1] It is a user-defined blueprint/template or prototype from which objects are created. For Example: Dog, Cat, Monkey etc. are the object of "Animal" class

[2] Convert (a program) into a machine-code or lower-level form in which the program can be executed.

[3] Robustness is a measure of how well a software system can cope with invalid inputs or unexpected user interactions

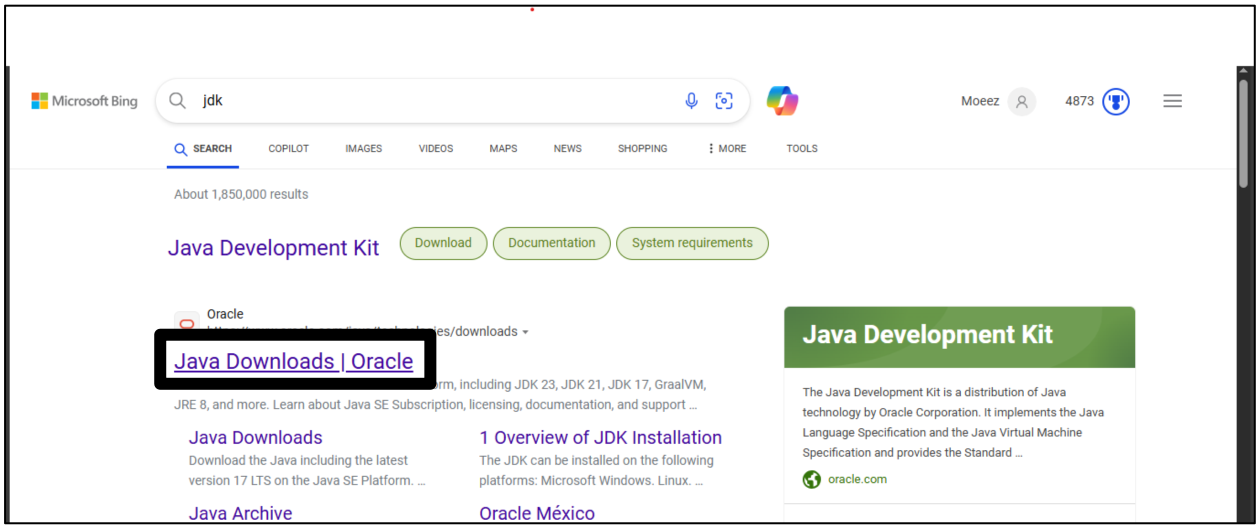
[4] Convert (a program) into a machine-code or lower-level form in which the program can be executed.

[5] the process of identifying, isolating, and resol ving errors or bugs in programs.

[6] the process of subdividing a computer program into separate sub-programs.

How to Install JAVA

1. First open your local Browser and type java Development Kit (JDK).
2. Click on the first link here.



About 1,850,000 results

Java Development Kit — Download | Documentation | System requirements

Oracle
…es/downloads
Java Downloads | Oracle
…rm, including JDK 23, JDK 21, JDK 17, GraalVM, JRE 8, and more. Learn about Java SE Subscription, licensing, documentation, and support ...

Java Downloads
Download the Java including the latest version 17 LTS on the Java SE Platform. ...

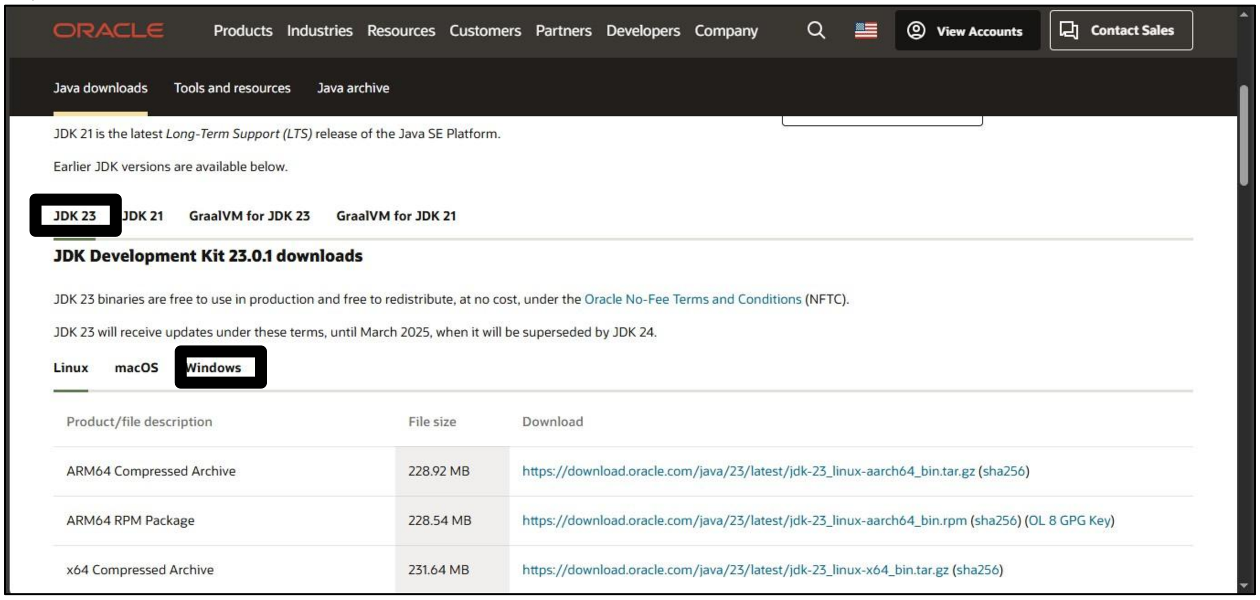
1 Overview of JDK Installation
The JDK can be installed on the following platforms: Microsoft Windows. Linux. ...

Java Archive

Oracle México

Java Development Kit
The Java Development Kit is a distribution of Java technology by Oracle Corporation. It implements the Java Language Specification and the Java Virtual Machine Specification and provides the Standard ...
oracle.com

3. Now, scroll to the version of the Java which you want to download and click on JDK Download option as shown below



JDK 21 is the latest *Long-Term Support (LTS)* release of the Java SE Platform.

Earlier JDK versions are available below.

JDK 23 | JDK 21 | GraalVM for JDK 23 | GraalVM for JDK 21

JDK Development Kit 23.0.1 downloads

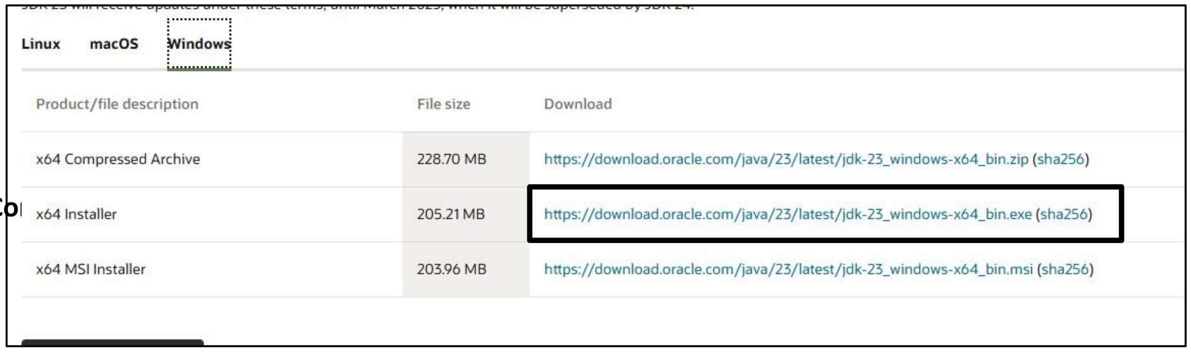
JDK 23 binaries are free to use in production and free to redistribute, at no cost, under the Oracle No-Fee Terms and Conditions (NFTC).

JDK 23 will receive updates under these terms, until March 2025, when it will be superseded by JDK 24.

Linux | macOS | Windows

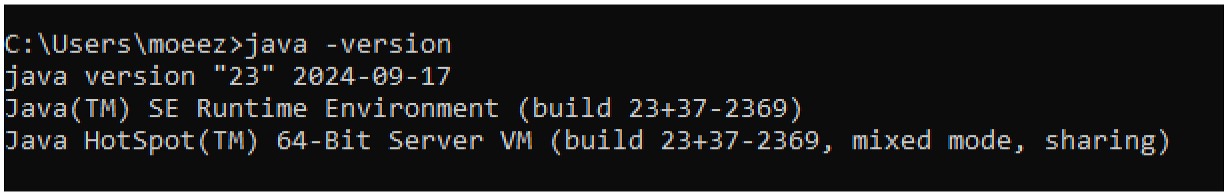
Product/file description	File size	Download
ARM64 Compressed Archive	228.92 MB	https://download.oracle.com/java/23/latest/jdk-23_linux-aarch64_bin.tar.gz (sha256)
ARM64 RPM Package	228.54 MB	https://download.oracle.com/java/23/latest/jdk-23_linux-aarch64_bin.rpm (sha256) (OL 8 GPG Key)
x64 Compressed Archive	231.64 MB	https://download.oracle.com/java/23/latest/jdk-23_linux-x64_bin.tar.gz (sha256)

4. Click on the download link to download link to download.



Linux | macOS | Windows

Product/file description	File size	Download
x64 Compressed Archive	228.70 MB	https://download.oracle.com/java/23/latest/jdk-23_windows-x64_bin.zip (sha256)
x64 Installer	205.21 MB	https://download.oracle.com/java/23/latest/jdk-23_windows-x64_bin.exe (sha256)
x64 MSI Installer	203.96 MB	https://download.oracle.com/java/23/latest/jdk-23_windows-x64_bin.msi (sha256)

5. After Downloading the file. Open it and install .
6. Run the installer once the download is complete.
7. Follow the installation wizard prompts, clicking “Next” as required.
8. Choose the installation directory and complete the installation process.
9. At last, To confirm if everything is set up properly, open cmd and type java –version
10. 

```
C:\Users\moeez>java -version
java version "23" 2024-09-17
Java(TM) SE Runtime Environment (build 23+37-2369)
Java HotSpot(TM) 64-Bit Server VM (build 23+37-2369, mixed mode, sharing)
```

JDK in Java

The Java Development Kit (JDK) is a cross-platformed software development environment that offers a collection of tools and libraries necessary for developing Java-based software applications and applets. It is a core package used in Java, along with the JVM [7](Java Virtual Machine) and the JRE (Java Runtime Environment). You can use Java Runtime Environment to run programs in your computer. JRE is available in JDK

The JDK has a private Java Virtual Machine (JVM) and a few other resources necessary for the development of a Java Application.

JDK comes with a collection of tools that are used for developing and running java programs. Which contains:

1. JRE (Java Runtime Environment)
1. JAVA (An interpreter/loader)
2. JAVAC (A compiler)
3. JAR(An archiver) and many more.

How Java Program is Executed

1) **Writing java program**

We need to have a java source code otherwise we won't be able to run the program you need to save it with the program.java extension. When we talk about creating a program or typing a code that solves any kind of problem that we may have, it means physically writing the program on any text editor like notepad, notepad++ or visual studios. You may or may not edit the program after you have written it once. When you create a program you don't just make it and let it be the way it is you can modify it and

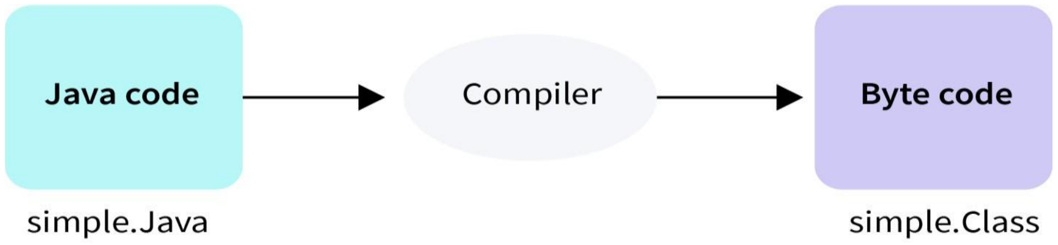
7 When you run a Java application, the JVM is responsible for converting the bytecode into machine-specific code.

save it on the device. But you need to save it with the java extension otherwise it would just appear as a normal text file.

2) Compiling a Java Program

Now once the program is created and does not have any errors or mistakes we can go ahead and compile the program when you compile a program it means the compiler compiles the program and if there are no errors after compiling the program we can further run it and get the desired output. We compile the Java program in a command prompt or another console.

First we need to set the path of the program we want to execute in CMD. Once your path is set you need to write the following command on CMD- **"javac filename.java".**



3) Loading the Program into the Memory by Java Virtual Machine

A lot of memory is required by JVMwhen you want to load the .class file extension before the execution. Loading is the process of placing a program in memory for it to run. The .class files are needed by the program to execute the file.

4) Java Virtual Machine verification for bytecode

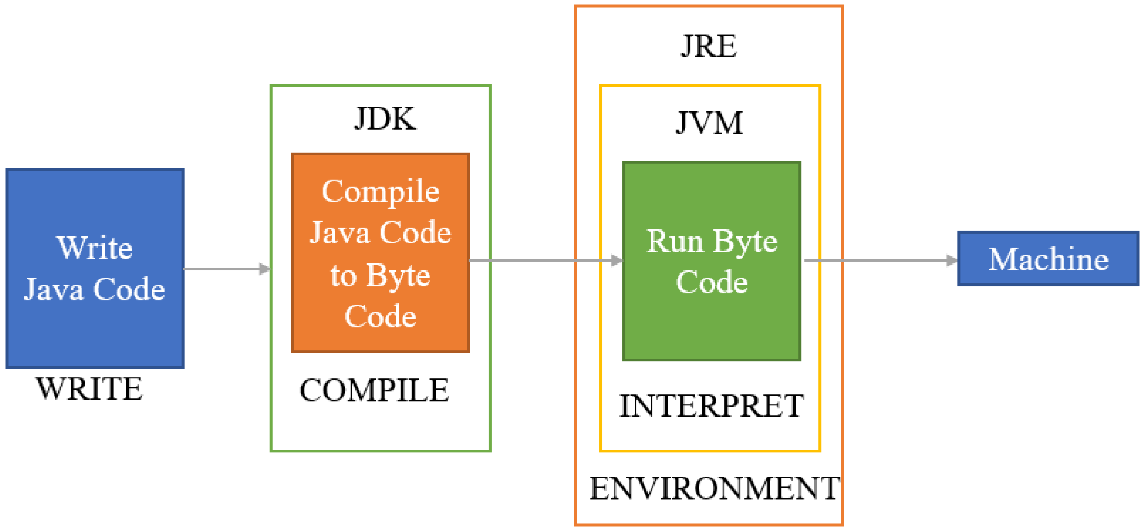
Jvm has a bytecode verifier that is due to maintenance of the security of the program. The bytecode verifies the code only and only after the classes have been loaded in the memory to maintain the security of the program. It makes sure that the bytecodes are valid and accessible. It also saves the computer from various viruses and unsecure websites

5) Java Program Execution

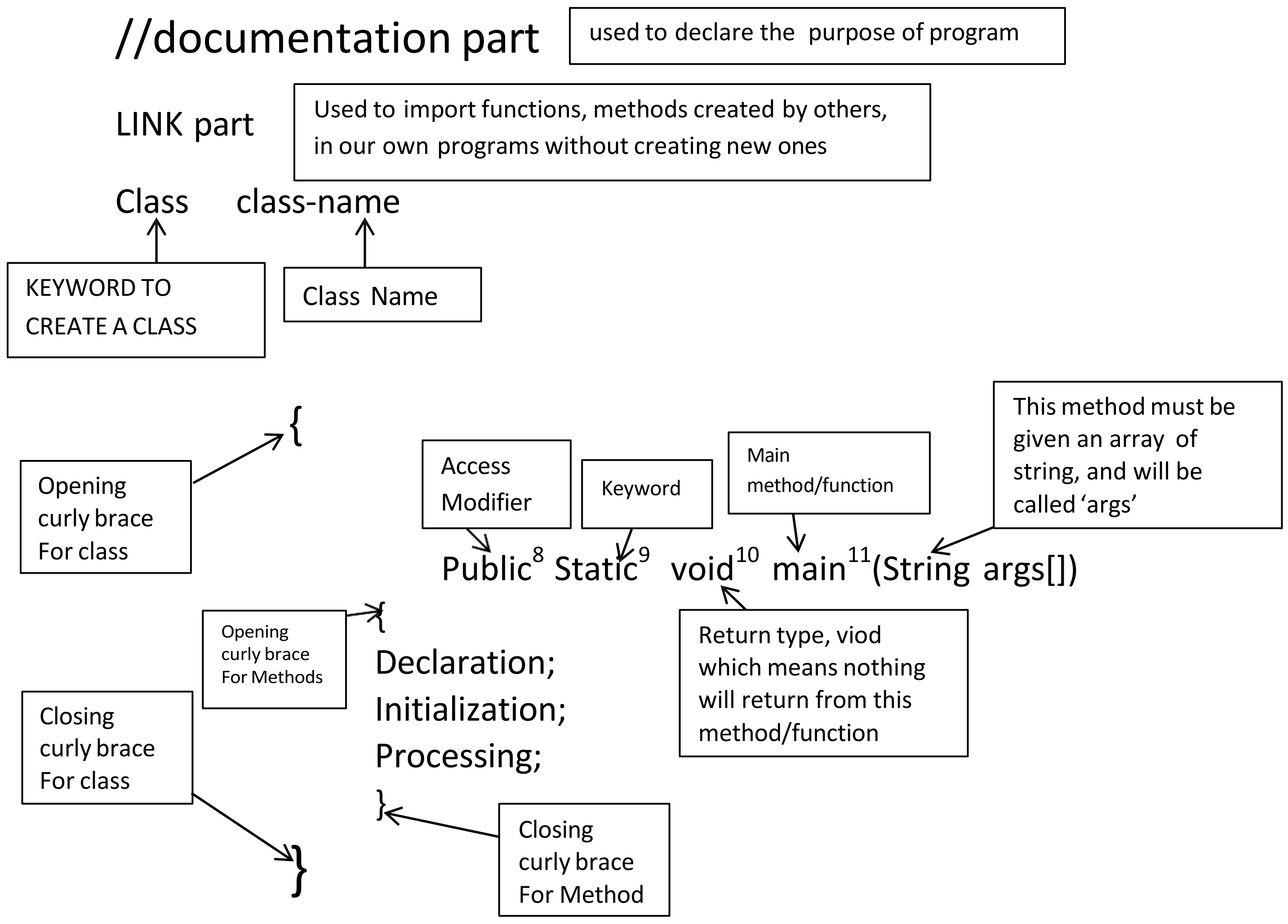
The above steps are executed by JVMwhen it interprets the bytecode. Earlier JVM's were slow and only interpreted one bytecode at a time. Nowadays the modern JVM's are much faster as they use

JIT(just-in-time) compilation units. These JVMs can execute various tasks at the same time. We also call them HotSpot compilers as they are used by JVM to find out the hot spots in our bytecode. Later on, the source code is converted bytecode into machine language.

The command to interpret/run the program : **“java filename.java”**



Simple Structure of Java program



//documentation part
used to declare the purpose of program
LINK part
Used to import functions, methods created by others, in our own programs without creating new ones
Class class-name
KEYWORD TO CREATE A CLASS
Class Name
Opening curly brace For class
{
Access Modifier
Keyword
Main method/function
This method must be given an array of string, and will be called 'args'
Public[8] Static[9] void[10] main[11](String args[])
Opening curly brace For Methods
{
Declaration;
Initialization;
Processing;
}
Return type, viod which means nothing will return from this method/function
Closing curly brace For class
Closing curly brace For Method
}

After writing this In any text editor. You need to save this program by pressing ctrl+s ,

You must give this program the same name as of your class name and then .java,

for instance, in this program , the name of our class is class-name, so we must name it class-name.java to make this program a proper java program.

[8] public keyword is an access modifier that represents visibility. It means it is visible to all.

[9] static is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method.

[10] Void is the return type of the method. It means it doesn't return any value.

[11] main represents the starting point of the program. We must put our code in main to execute it.

#1 Program

```
//program to print a simple text    (this is document part)

Class hello (we can name our class anything but also should follow the rules of an identifier, same for method/function name)

{
    Public static void main (String args[])
    {
        System.out.print("Welcome to java programming ");
    }
}
```

< System.out.println(); ---- this is used to print the output in nextline >

To do list

- Write a basic program to write your name, address and phone number.

Compiler VS Interpreter

Compiler and Interpreter are two different ways to translate a program from programming or scripting language to machine language. But Java is both interpreted as well as compiled.

What is a Compiler?

The Compiler is a translator that takes input i.e., High-Level Language, and produces an output of low-level language i.e. machine or assembly language. The work of a Compiler is to transform the codes written in the programming language into machine code (format of 0s and 1s) so that computers can understand.

Compiler runs its program run time is longer and occupies a larger part of memory. It has a slow speed because a compiler goes through the entire program and then translates the entire program into machine codes. Compiler checks the whole program before converting it into machine language. If found any error it doesn't proceed any further.

What is an Interpreter?

An Interpreter is a program that translates a programming language into a comprehensible language. The interpreter converts high-level language to an intermediate language. It contains pre-compiled code, source code, etc.

It translates only one statement of the program at a time. It translate program line by line. If found any error it doesn't proceed any further either.

Java has both compiler and interpreter

Java first compile then interprets.

Types of errors

Error is an illegal operation performed by the user which results in the abnormal working of the program. Programming errors often remain undetected until the program is compiled or executed. Some of the errors inhibit the program from getting compiled or executed. Thus errors should be removed before compiling and executing.

The most common errors can be broadly classified as follows:

1. Run Time Error:

Run Time errors occur or we can say, are detected during the execution of the program. Sometimes these are discovered when the user enters an invalid data or data which is not relevant. Runtime errors occur when a program does not contain any syntax errors but asks the computer to do something that the computer is unable to reliably do.

2. Syntax errors:

Syntax errors are those errors which prevent the code from running because of an incorrect syntax such as a missing semicolon at the end of a statement or a missing bracket, class not found, etc. These errors are detected by the java compiler and an error message is displayed on the screen while compiling. Compile Time Errors are sometimes also referred to as Compile Time Error.

3. Logical Error:

A logic error is when your program compiles and executes, but does the wrong thing or returns an incorrect result or no output when it should be returning an output. These errors are detected neither by the compiler nor by JVM. The Java system has no idea what your program is supposed to do, so it provides no additional information to help you find the error. Logical errors are also called Semantic Errors. These errors are caused due to an incorrect idea or concept used by a programmer while coding.

What is a Variable?

In Java, Variables are the data containers that save the data values during Java program execution. Every Variable in Java is assigned a data type that designates the type and quantity of value it can hold. A variable is a memory location name for the data.

Java variable is a name given to a memory location. It is the basic unit of storage in a program.

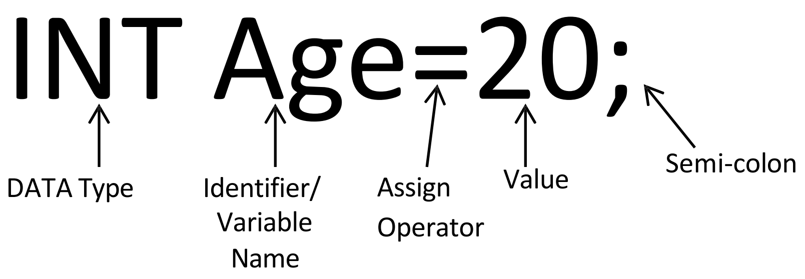
The value stored in a variable can be changed during program execution.

Variables in Java are only a name given to a memory location. All the operations done on the variable affect that memory location.

In Java, all variables must be declared before use.

To declare a variable you need to first declare its data type(which tell the program what type of data will be stored in variable).Then give it a name(the name of the variable is known as identifier) after that use an assign operator (=) then the value you want to store ending with a semi-colon(;)

e.g:



```
INT Age=20;
```

Java-Token

In Java, Tokens are the smallest elements of a program that is meaningful to the compiler. They are also known as the fundamental building blocks of the program. Tokens can be classified as follows:

1. Keywords
2. Identifiers
3. Constants
4. Special Symbols
5. Operators
6. Comments
7. Separators

Keywords: Keywords are pre-defined or reserved words in a programming language. Each keyword is meant to perform a specific function in a program. Since keywords are referred names for a compiler, they can't be used as variable names because by doing so, we are trying to assign a new meaning to the keyword which is not allowed .Java language supports the following 60 keywords:

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

Identifiers:

Identifiers are used as the general terminology for naming of variables, functions and arrays. These are in other words name for our variables, functions and arrays which a user can use to call the variables, functions and arrays created. These are user-defined names consisting of a long sequence of letters and digits with either a letter or the underscore (_) as a first character. Identifier names must differ in spelling and case from any keywords. You cannot use keywords as identifiers; they are reserved for special use. Once declared, you can use the identifier in later program statements to refer to the associated value.
The rules to declare an Identifier are as below:

1. A valid identifier must have characters [A-Z] or [a-z] or numbers [0-9], and underscore (_) or a dollar sign ($). For example, @java1 is not a valid identifier because it contains a special character which is @.
2. There should not be any space in an identifier. For example, java 1 is an invalid identifier.
3. An identifier should not contain a number at the starting. For example, 123javatpoint is an invalid identifier.
4. An identifier should be of length 4-15 letters only. However, there is no limit on its length. But, it is good to follow the standard conventions.
5. We can't use the Java reserved keywords as an identifier such as int, float, double, char, etc. For example, int double is an invalid identifier in Java.

Java is a Case-Sensitive language which means upper case and lower case matters. “a” and “A” are two different things . another thing to keep in mind while naming an Identifier.
Examples of valid identifiers:
MyVariable
MYVARIABLE
myvariable
x
i
x1
i1
_myvariable
$myvariable
sum_of_array
java123

Examples of invalid identifiers:
My Variable // contains a space
123java // Begins with a digit
a+c // plus sign is not an alphanumeric character
variable-2 // hyphen is not an alphanumeric character

```
sum_&_difference // ampersand is not an alphanumeric character
```

Constants/Literals:

Constants are also like normal variables. But the only difference is, their values cannot be modified by the program once they are defined. Constants refer to fixed values. They are also called as literals. Constants may belong to any of the data type.
Syntax:
final data_type variable_name;
e.g
final int a=10; // no modification of variables is now possible, value cannot change now.

Special Symbols:

The following special symbols are used in Java having some special meaning and thus, cannot be used for some other purpose.

[] () {}, ; * =

Brackets[]: Opening and closing brackets are used as array element reference. These indicate single and multidimensional subscripts.
Parentheses(): These special symbols are used to indicate function calls and function parameters.
Braces{}: These opening and ending curly braces marks the start and end of a block of code containing more than one executable statement.
comma (,): It is used to separate more than one statements like for separating parameters in function calls.
semi colon : It is an operator that essentially invokes something called an initialization list.
asterick (*): It is used to create pointer variable.
assignment operator: It is used to assign values.

Operators:

Java provides many types of operators which can be used according to the nee d. They are classified based on the functionality they provide. Some of the types are-

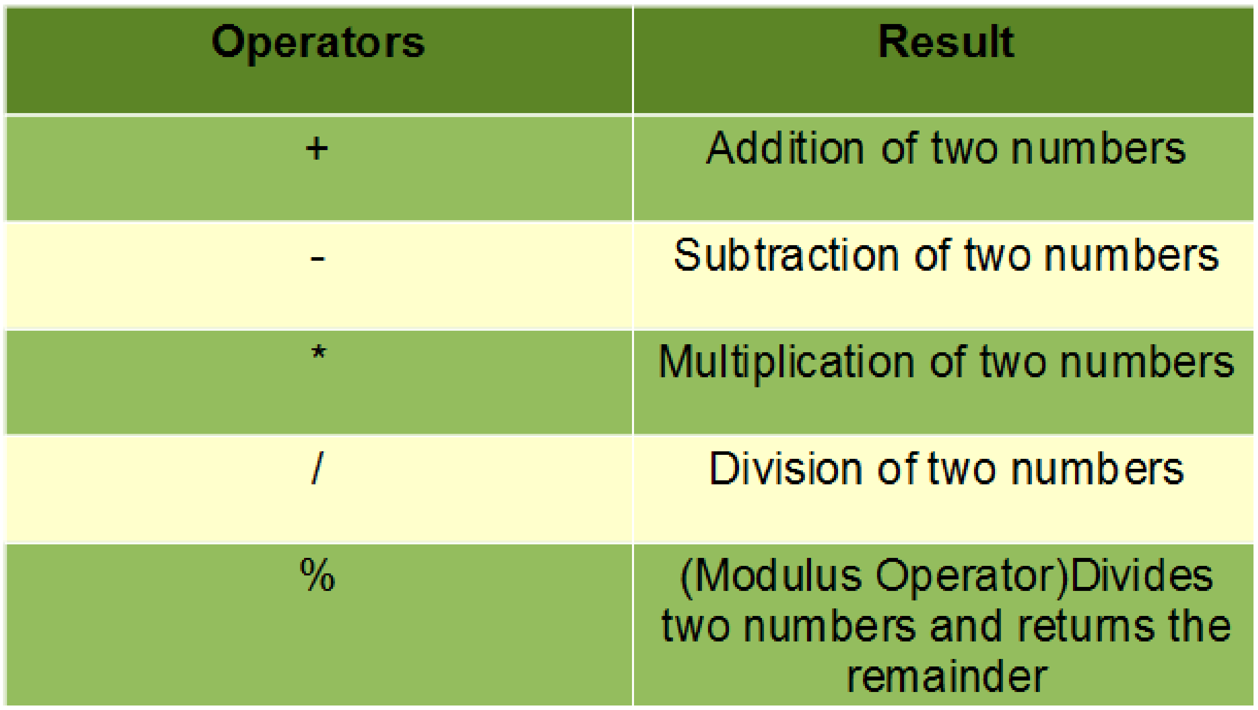
1. Arithmetic Operators
2. Unary Operators
3. Assignment Operator
4. Relational Operators
5. Logical Operators

Syntax of an operator:
Operand operator operand;
e.g.: a+b;

1. **Arithmetic Operators**

These operators involve the mathematical operators that can be used to perform various simple or advanced arithmetic operations on the primitive data types referred to as the operands. These operators consist of various unary and binary operators that can be applied on a single or two operands. The different operators that Java has to provide under the arithmetic operators are as below:

Arithmetic Operators in Java



Operators	Result
+	Addition of two numbers
-	Subtraction of two numbers
*	Multiplication of two numbers
/	Division of two numbers
%	(Modulus Operator)Divides two numbers and returns the remainder

2. **Unary Operators**

Java unary operators are the types that need only one operand to perform any operation like increment, decrement, negation, etc. It consists of various arithmetic, logical and other operators that operate on a single operand.

3. **Assignment Operators**

These operators are used to assign values to a variable. The left side operand of the assignment operator is a variable, and the right side operand of the assignment operator is a value. The value on the right side must be of the same data type of the operand on the left side. Otherwise, the compiler will raise an error. This means that the assignment operators have right to left associativity, i.e., the value given on the right-hand side of the operator is assigned to the variable on the left. Therefore, the right-hand side value must be declared before using it or should be a constant. The general format of the assignment operator is,

variable operator value;

4. **Relational Operators**
These are a bunch of binary operators used to check for relations between two operands, including equality, greater than, less than, etc. They return a boolean result after the comparison and are extensively used in looping statements as well as conditional if-else statements and so on. The general format of representing relational operator is:

variable1 relation_operator variable2

5. **Logical operators**
These are used to perform logical "AND", "OR" and "NOT" operations, i.e. the function similar to AND gate and OR gate in digital electronics. They are used to combine two or more conditions/constraints or to complement the evaluation of the original condition under particular consideration. One thing to keep in mind is, while using AND operator, the second condition is not evaluated if the first one is false. Whereas while using OR operator, the second condition is not evaluated if the first one is true, i.e. the AND and OR operators have a short-circuiting effect. Used extensively to test for several conditions for making a decision.

- AND Operator (&&) – if(a && b) *if true execute else don't+
- OR Operator (||) – if(a || b) [if one of them is true to execute else don't+
- NOT Operator (!) – !(a<b) [returns false if a is smaller than b]

Comments:

In Java, Comments are the part of the program which are ignored by the compiler while compiling the Program. They are useful as they can be used to describe the operation or methods in the program. The Comments are classified as follows:

- Single Line Comments
- Multiline Comments

1. // This is a Single Line Comment

2. /*
   This is a Multiline Comment
   */

Separators:

Separators are used to separate different parts of the codes. It tells the compiler about completion of a statement or line in the program. The most commonly and frequentlyused separator in java is semicolon (;).

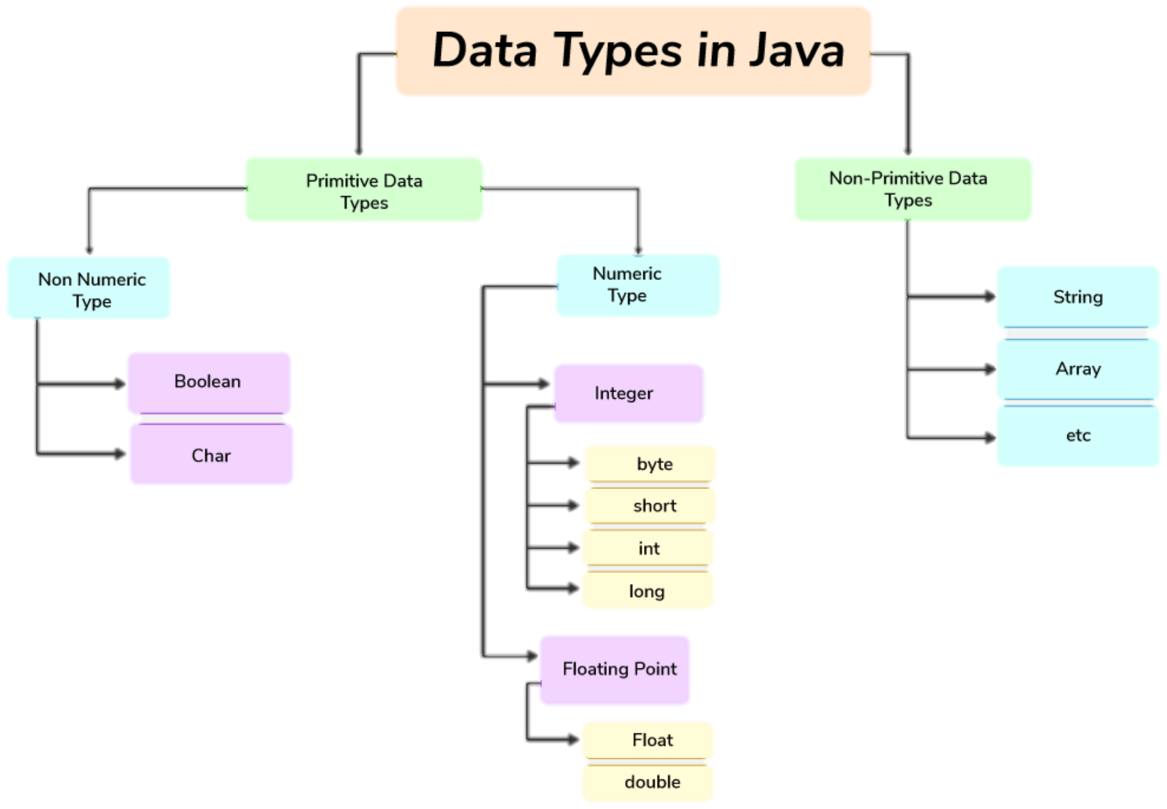
int variable;

Data Types in Java

Data types in Java are of different sizes and values that can be stored in the variable that is made as per convenience and circumstances to cover up all test cases. Java has two categories in which data types are segregated

Primitive Data Type: such as boolean, char, int, short, byte, long, float, and double. The Boolean with uppercase B is a wrapper class for the primitive data type boolean in Java.

Non-Primitive Data Type or Object Data type: such as String, Array, etc.



1. Primitive data type:

Primitive data are only single values and have no special capabilities. There are 8 primitive data types. They are depicted belowin tabular format below as follows:

Type	Default	Size	Range of values they can store
boolean	false	8 bits	true, false
byte	0	8 bits	-128 to 127
char	\u0000	16 bits	characters representation of ASCII values 0 to 255
short	0	16 bits	-32,768 to 32,767
int	0	32 bits	-2,147,483,648 to 2,147,483,647
long	0	64 bits	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	0.0	32 bits	upto 7 decimal digits
double	0.0	64 bits	upto 16 decimal digits

You can you any of these to declare a variable before assigning any value.

2. Non-Primitive Data Types:

The Non-Primitive Data Types will contain a memory address of variable values because the reference types won’t store the variable value directly in memory. They are strings, arrays, etc.

Type	Default	Size
String	varies	sequence of characters
Arrays	varies	collection of elements of the same type

How to Initialize variables

To create a variable, you must specify the type and assign it a value:

Syntax:

```
datatype variableName = value;
```

```
e.g. int a=5;
```

#2 Program

```
//Program to initialize and use different variables

class first

{

            public static void main(String args[])

            {

            int myNum = 5;              // Integer (whole number)

            float myFloatNum = 5.99f;    // Floating point number

            char myLetter = 'D';        // Character

            boolean myBool = true;       // Boolean

            String myText = "Hello";  //Multiple characters

            //printing the variable, write println to print in nextline

            System.out.println(myNum); //printing the interger variable

            System.out.println(myFloatNum); // printing thefloat  variable

            System.out.println(myLetter);// printing thecharacter  variable

            System.out.println(myText);// printing the string  variable

                                                    Or

                        System.out.println("The integer variable = " + myNum);

                        System.out.println("The float variable = " + myFloatNum);

                        System.out.println("The boolean variable = " + myBool);

                        System.out.println("The character variable = " + myLetter);

                        System.out.println("The string variable = " + myText);

            }

}
```

#3 Program

```
//program to perform different operations on variables
class Add
{
        publicstatic  void main(String args[])
                {
int myNum=5;
int z=myNum+myNum;   //addition of same two variables
System.out.println(z);
                                        //or
System.out.println(myNum+myNum); //you can  directly add both while printing
float myFloatNum = 5.99f;
float x=myFloatNum+myFloatNum;             //same for float
System.out.println(x);
                                                //or
System.out.println(myFloatNum+myFloatNum);
//if you try to add two character or string they would concatenate/merge with each other.
// + works as a concatenater/merger for strings and characters
                }
}
```

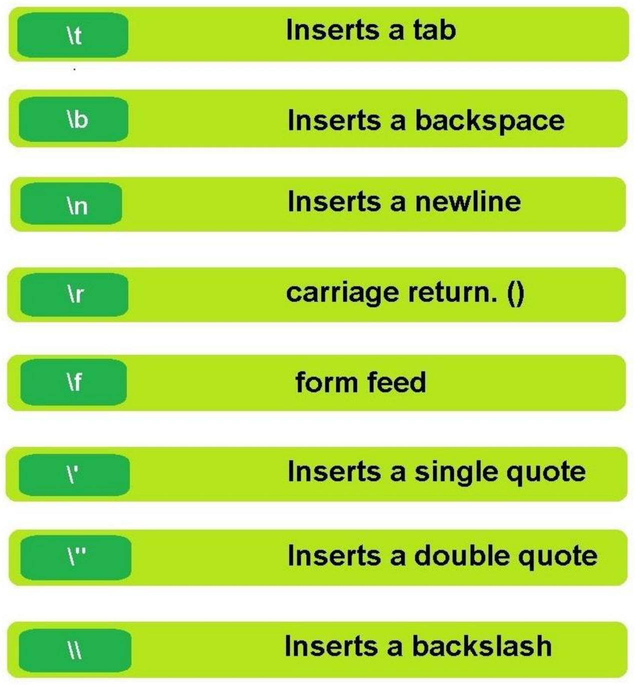
To do list

- Write a basic program for addition,substraction,multiplication and division for two variable with/without third variable.
- Write a program for concatenation of 5 strings into a single string.
- Write a basic program for addition,substraction,multiplication and division for more than 2 variable with/without third variable.

Contents

Escape Sequences in Java

A character with a backslash (\) just before it is an escape sequence or escape character. We use escape characters to perform some specific task. The total number of escape sequences or escape characters in Java is 8 but you need only a few.



\t	Inserts a tab
\b	Inserts a backspace
\n	Inserts a newline
\r	carriage return. ()
\f	form feed
\'	Inserts a single quote
\"	Inserts a double quote
\\	Inserts a backslash

List of Escape Sequences in Java

Why will we need Escape sequence?

Suppose we would like to run this java code:

```
public class Test {

        public static void main(String[] args)

        {

                System.out.println("Hello world, welcome to "Java programming ".");

        }

}
```

We want this output: Hello world, Welcome to "Java Programming"

But this won’t work as the code will give you a runtime error as the compiler expects nothing but only strings inside the quotation mark but when the compiler found a quotation mark, it expects another quotation mark in the near future (the closing one) and between them, the string of text should be created. During this case, the quotation marks of the word “java programming” gets nested(inside another quotation marks). Once the compiler reaches here, the compiler gets confused. According to the rule, the quotation mark suggests the compiler for creating a string but the compiler was busy with doing that thing previously and the code gives us a compile -time error.

To fix this we will use an escape sequence, in escape sequence the backslash (\) glued with a character (the character which has to be escaped) is called a control sequence.

For this particular program we can use \” escape sequence ,

```
public class Test {

        public static void main(String[] args)

        {

                System.out.println("Hello world, welcome to \"Java programming \".");

        }

}
```

Output: Hello world, welcome to "Java programming".

Some more Examples of Java Escape Characters

- Java code for the escape sequence \t:

```
// \t -> It gives a tab between two words.
publicclass Test {
    public static void main(String[] args)
    {
        System.out.println("Welcom to \t Java Programming! ");
    }
}
```

Output: Welcome to   Java Programming!

- Java code for the escape sequence \n :

```
// This \n escape sequence is for a new line.

public class Test {
        public static void main(String[] args)
        {
                System.out.println("Welcome to \nJava Programming! ");
        }
}
```

Output: Welcome to
Java Programming!

- Java code for the escape sequence \f:

```
// This \f escape sequence is a form feed character
// It is an old technique and used to indicate a page break.

public class Test {
        public static void main(String[] args)
        {
                System.out.println("Welcome to\f Java Programming ");
        }
}
```

```
Output: Good Morning Geeks!
                          How are you all?
```

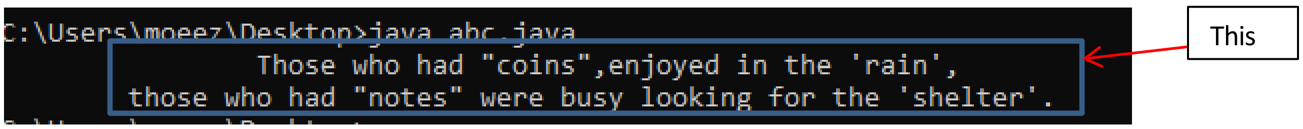
- Java code for the escape sequence \’

```
// This \' escape sequence is for printing a single quotation mark on the text string

public class Test {
        public static void main(String[] args)
        {
                System.out.println("Welcome to \'Java\'   Programming");
        }
}
```

Output: Welcome to 'Java' Programming

This is **Escape Sequence**

Write a program in java to get this output:



```
C:\Users\moeez\Desktop>java abc.java
                Those who had "coins",enjoyed in the 'rain',
  those who had "notes" were busy looking for the 'shelter'.
```

Same as shown here.

Java User Defined Variable

The Scanner class is used to get user input, and it is found in the java.util package (link part of document).

To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation. In our example, we will use the nextLine() method, which is used to read Strings:

```
import java.util.Scanner;  // Import the Scanner class

class Main {

  public static void main(String[] args) {

    Scanner myObj[1] = new Scanner(System.in);  // Create a Scanner object,

    System.out.println("Enter username");

    String userName = myObj.nextLine();  // Read user input

    System.out.println("Username is: " + userName);  // Output user input

  }

}
```

Method	Description
nextBoolean()	Reads a boolean value from the user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads a int value from the user
nextLine()	Reads a String value from the user
nextLong()	Reads a long value from the user
nextShort()	Reads a short value from the user

[1] you can name it anything but it must follow the rules of an identifier

#1 Program

```
//program for getting name ,age, salary and rank of an employee

import java.util.Scanner;

class abc {

  public static void main(String[] args) {

    Scanner myObj = new Scanner(System.in);


    System.out.println("Enter name, age and salary:");


    // String input

    String name = myObj.nextLine();


    // Numerical input

    int age = myObj.nextInt();

    double salary = myObj.nextDouble();


//Character input

char Rank = myObj.next().charAt(0);

    // Output input by user

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

    System.out.println("Age: " + age);

    System.out.println("Salary: " + salary);

    System.out.println("Rank: " + Rank);

  }
}
```

To do list

- Write a program to prepare the marks card for a student having 5 subjects,also calculate percentage.
- Write a program in which you calculate the total amount for buying from a grocery store.
- Write a program to add,sub,multiple and divide 2 or more variables given by the user with and without third variable.

#2 Program

```
//program for swapping of two variable with and without third variable

class abc {

  public static void main(String[] args) {

        System.out.println("enter the values for your variables");

        int x=10;

        int y=20;

        System.out.println("Values Before Swapping ");

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

        System.out.println("Y = "+y);



        //Swapping using third variables

        int z=x;

        x=y;

        y=z;

/*In this program, z temporarily holds the value of x, allowing x to take on the value of y, and then y
to take on the value stored in z. swapping completed */

                                    //or

//Swapping without using the third variables

                x=x+y;
                y=x-y;
                x=x-y;
```

```
//This program first adds the values of x and y and stores the sum in x. Then, it subtracts y from the
new value of x to get the original value of x and assigns it to y. Finally, it subtracts the new value of y
from x to get the original value of y and assigns it to x.

        //Now printing the variables after swapping

        System.out.println("Values After Swapping ");

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

        System.out.println("Y = "+y);

  }

}
```

To do list

- Write a program of swapping with user defined variables.
- Write a program of swapping for swapping of characters (not numbers)

Java Format Specifiers

Format specifiers in Java are special sequences of characters used to format output data. They are primarily used with methods like System.out.printf() and String.format() to create formatted strings. Each specifier begins with a % and is followed by a character that represents the data type. Here are some common format specifiers:

%d: for integers

%f: for floating-point numbers

%c: for characters

%s: for strings

%x: for hexadecimal integers

Format specifiers begin with a percent character (%) and terminate with a "type character, " which indicates the type of data (int, float, etc.) that will be converted the basic manner in which the data will be represented (decimal, hexadecimal, etc.)

```
//program to use format specifiers

Class forspe{

            Public static void main(String arg[])

{

            int number = 42;

            double pi = 3.14159;

            char letter = 'A';

            String word = "Hello";

System.out.printf("Integer: %d\n", number);

System.out.printf("Floating-point: %.2f\n", pi);

System.out.printf("Character: %c\n", letter);

System.out.printf("String: %s\n", word);

}

}
```

#3 Program

```
import java.util.Scanner;

class pro3 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the first number: ");
        int a = sc.nextInt();

        System.out.print("Enter the second number: ");
        int b = sc.nextInt();

        System.out.printf("The sum of %d + %d = %d\n", a, b, a + b);
    }
}
```

To do list

- **Write a program to add 5 variable using format specifiers**
- **Write a program to perform arithmetic operations on variables using format specifiers.**

Strings in Java

A string is a sequence of characters, Syntax :

String name;

Name=new String("Java");

String is a class but can be used like a data type.

Different ways to print in java:

1. System.out.print() //print normally no new line at the end
2. System.out.println() //print a new line at the end
3. System.out.format() //for format specifiers
4. System.out.printf() //another format specifier
    a. System.out.printf("%c",char);
    b. System.out.printf("%d",int);
    c. System.out.printf("%f",float);
    d. System.out.printf("%s",Strings);

String Method

String methods operate on java strings. They can be used to find length of the string,convert to lowercase or uppercase ETC.

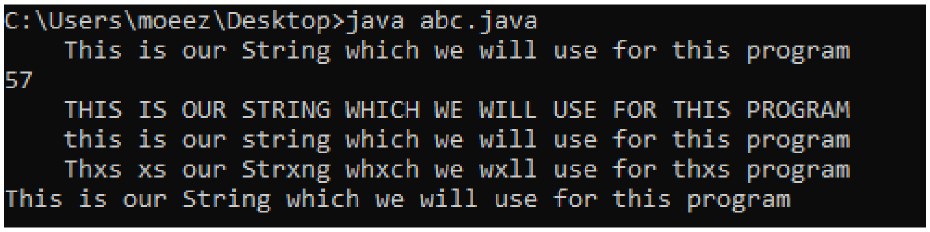
Some of the commonly used String methods are:

Method	Description	Return Type
length()	Returns the length of a specified string	int
concat()	Appends a string to the end of another string	String
charAt()	Returns the character at the specified index (position)	Char
replace()	Searches a string for a specified value, and returns a new string where the specified values are replaced	String
substring()	Returns a new string which is the substring of a specified string	String
toUpperCase()	Converts a string to upper case letters	String
toLowerCase()	Converts a string to lower case letters	String

trim()	Removes whitespace from both ends of a string	String

#4 Program

```
//program for using string methods

class pro4{

   public static void main(String[] args) {

            String str="    This is our String which we will use for this program";

            System.out.println(str);

            //string methods start from here

            int a=str.length();   //this will put the length of the string in variable a;

            System.out.println(a); //printing length of the string



String cap=str.toUpperCase();   //this will convert the string in capital case  and store in  variable cap

            System.out.println(cap);



            String low=str.toLowerCase();  //this will convert the string in lower case

            System.out.println(low);



            String rep=str.replace('i','x'); //this will replace every I in this string  with x
            System.out.println(rep);



            String tr=str.trim(); //this will remove all the whitespaces before and after string

            System.out.println(tr);
            }
}
```

Output: 

```
C:\Users\moeez\Desktop>java abc.java
    This is our String which we will use for this program
57
    THIS IS OUR STRING WHICH WE WILL USE FOR THIS PROGRAM
    this is our string which we will use for this program
    Thxs xs our Strxng whxch we wxll use for thxs program
This is our String which we will use for this program
```

To do list

- Write a program to get the length of a userdefined string.
- Write a program to convert user defined string into capital and lower case
- Write a program to replace the character of a string with user defined values.

Contents

Object Oriented Programming

In programming, we can do Procedural programming or Object oriented programming.

Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods. Till now we used procedural programming to make our programs but now we will make programs by using Object oriented programming (oops)

Object-oriented programming has several advantages over procedural programming:

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time

Before using object oriented programming we will learn about objects and classes first.

What are Classes and Objects?

Classes and objects are the two main aspects of object-oriented programming.

Java is an object-oriented programming language.

Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has attributes, such as weight and color, and methods, such as drive and brake.

A Class is like an object constructor, or a "blueprint" for creating objects.

Create a Class

To create a class, use the keyword class:

```
public class Main {
int x = 5;
}
```

Create an Object

In Java, an object is created from a class. We have already created the class named Main, so now we can use this to create objects.

To create an object of Main, specify the class name, followed by the object name, and use the keyword new:
Create an object called "myObj" and print the value of x:

```
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```

Multiple Objects

You can create multiple objects of one class:

Example

Create two objects of Main:

```
class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj1 = new Main();  // Object 1
    Main myObj2 = new Main();  // Object 2
    System.out.println(myObj1.x);
    System.out.println(myObj2.x);
  }
}
```

Using Multiple Classes

You can also create an object of a class and access it in another class. This is often used for better organization of classes (one class has all the attributes and methods, while the other class holds the main() method (code to be executed)).

Remember that the name of the java file should match the class name which holds the main method. In this example, now we will create a java program in which we will use 2 classes in the same program.

```
public class First {
  int x = 5;
}
class Second {
  public static void main(String[] args) {
    First myObj = new First();
    System.out.println(myObj.x);
  }
}
```

I need to name this program "Second.java" as my main method is in that class.

Java Class Attributes

Previously, we used the term "variable" for x in the example (as shown below). It is actually an attribute of the class. Or you could say that class attributes are variables within a class:

E.g,

```
public class Main {
  int x = 5;
  int y = 3;
}
```

Another term for class attributes is fields.

Accessing Attributes

You can access attributes by creating an object of the class, and by using the dot syntax (.):

The following example will create an object of the Main class, with the name myObj. We use the x attribute on the object to print its value:

Example

Create an object called "myObj" and print the value of x:

```
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```

Modify Attributes

You can also modify attribute values:

Example

Set the value of x to 40:

```
public class Main {
  int x;

  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 40;
    System.out.println(myObj.x);
  }
}
```

Or override existing values:

Example

Change the value of x to 25:

```
public class Main {
  int x = 10;

  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 25;  // x is now 25
    System.out.println(myObj.x);
  }
}
```

If you don't want the ability to override existing values, declare the attribute as final:

Example

```
public class Main {
  final int x = 10;

  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 25; // will generate an error: cannot assign a value to a final variable
    System.out.println(myObj.x);
  }
}
```

Multiple Objects

If you create multiple objects of one class, you can change the attribute values in one object, without affecting the attribute values in the other:

Example

Change the value of x to 25 in myObj2, and leave x in myObj1 unchanged:

```
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj1 = new Main();  // Object 1
    Main myObj2 = new Main();  // Object 2
    myObj2.x = 25;
    System.out.println(myObj1.x);  // Outputs 5
    System.out.println(myObj2.x);  // Outputs 25
  }
}
```

Multiple Attributes

You can specify as many attributes as you want:

Example

```
public class Main {
  String fname = "Shin";
  String lname = "Chan";
  int age = 10;
  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println("Name: " + myObj.fname + " " + myObj.lname);
    System.out.println("Age: " + myObj.age);
  }
}
```

Java Class Methods

You learned from the Java Methods chapter that methods are declared within a class, and that they are used to perform certain actions:

Create a method named myMethod() in Main:

```
public class Main {
  static void myMethod() {
    System.out.println("Hello World!");
  }
}
```

myMethod() prints a text (the action), when it is called. To call a method, write the method's name followed by two parentheses () and a semicolon;

Example

Inside main, call myMethod():

```
public class Main {
  static void myMethod() {
    System.out.println("Hello World!");
  }

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

// Outputs "Hello World!"
```

Static vs. Public

You will often see Java programs that have either static or public attributes and methods.

In the example above, we created a static method, which means that it can be accessed without creating an object of the class, unlike public, which can only be accessed by objects:

Example
An example to demonstrate the differences between static and public methods:

```
public class Main {
  // Static method
  static void myStaticMethod() {
   System.out.println("Static methods can be called without creating objects");
  }

  // Public method
  public void myPublicMethod() {
   System.out.println("Public methods must be called by creating objects");
  }

  // Main method
  public static void main(String[] args) {
   myStaticMethod(); // Call the static method
   // myPublicMethod(); This would compile an error

   Main myObj = new Main(); // Create an object of Main
   myObj.myPublicMethod(); // Call the public method on the object
  }
}
```

To do list

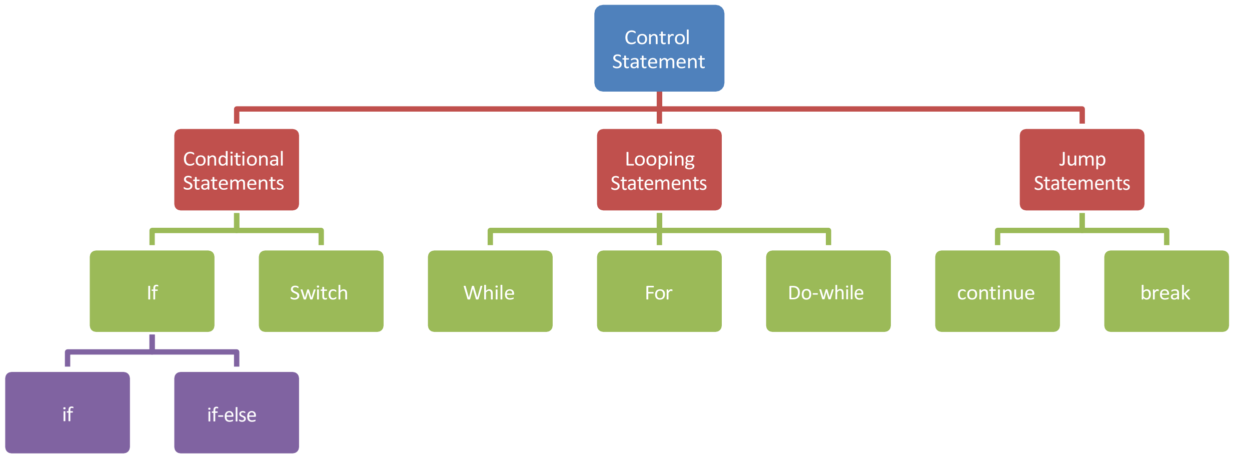
- Write a program of addition,subtraction and multiplication in java using classes and objects
- Write a program of addition with and without creating an object .
- Write a program to get data from different employes but using same variables.

Contents

Java Control flow Statements

Control flow statements are fundamental components of programming languages that allow developers to control the order in which instructions are executed in a program. They enable execution of a block of code multiple times, execute a block of code based on conditions, terminate or skip the execution of certain lines of code, etc. They are used to control the flow of our block of code. They are of 3 types.

Types of Control Statements



The if Statement

Use the if statement to specify a block of Java code to be executed if a condition is true.

Syntax

```
if (condition) {
  // block of code to be executed if the condition is true
}
```

Note that if is in lowercase letters. Uppercase letters (If or IF) will generate an error.

```
if (20 > 18) {
  System.out.println("20 is greater than 18");
}
```

example

```
int x = 20;
int y = 18;
if (x > y) {
  System.out.println("x is greater than y");
}
```

The else Statement

Use the else statement to specify a block of code to be executed if the condition is false.

Syntax

```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}
```

Example

```
int time = 20;
if (time < 18) {
  System.out.println("Good day.");
} else {
  System.out.println("Good evening.");
}
```

// Outputs "Good evening."

The else if Statement

Use the else if statement to specify a new condition if the first condition is false.

Syntax

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and condition2 is true
} else {
  // block of code to be executed if the condition1 is false and condition2 is false
}
```

Example

```
int time = 22;
if (time < 10) {
  System.out.println("Good morning.");
} else if (time < 18) {
  System.out.println("Good day.");
} else {
  System.out.println("Good evening.");
}
// Outputs "Good evening."
```

Java Switch Statements

Instead of writing many if..else statements, you can use the switch statement.

The switch statement selects one of many code blocks to be executed:

Syntax

```
switch(expression) {
case x:
   // code block
   break;
  case y:
   // code block
  break;
  default:
    // code block
}
```

This is how it works:

The switch expression is evaluated once.
The value of the expression is compared with the values of each case.
If there is a match, the associated block of code is executed.
The break and default keywords are optional, and will be described later in this chapter
The example below uses the weekday number to calculate the weekday name:

Example

```
int day = 4;
switch (day) {
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;
}
// Outputs "Thursday" (day 4)
```

The break Keyword

When Java reaches a break keyword, it breaks out of the switch block.

This will stop the execution of more code and case testing inside the block.

When a match is found, and the job is done, it's time for a break. There is no need for more testing.

A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block.

The default Keyword

The default keyword specifies some code to run if there is no case match:

Example

```
int day = 4;
switch (day) {
case 6:
   System.out.println("Today is Saturday");
   break;
  case 7:
   System.out.println("Today is Sunday");
   break;
  default:
   System.out.println("Looking forward to the Weekend");
}
// Outputs "Looking forward to the Weekend"
```

Loops

Loops can execute a block of code as long as a specified condition is reached.Loops are handy because they save time, reduce errors, and they make code more readable.

Java While Loop

The while loop loops through a block of code as long as a specified condi tion is true:

Syntax

```
while (condition) {
  // code block to be executed
}
```

In the example below, the code in the loop will run, over and over again, as long as a variable (i) is less than 5:

Example

```
int i = 0;

while (i < 5) {

  System.out.println(i);

  i++;

}
```

The Do/While Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

```
do {

  // code block to be executed

}

while (condition);
```

The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

Example

```
int i = 0;
do {
  System.out.println(i);
  i++;
}
while (i < 5);
```

Java For Loop

When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop:

Syntax

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

Statement 1 is executed (one time) before the execution of the code block.

Statement 2 defines the condition for executing the code block.

Statement 3 is executed (every time) after the code block has been executed.

The example below will print the numbers 0 to 4:

Example

```
for (int i = 0; i < 5; i++) {
  System.out.println(i);
}
```

Java Break

You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch statement.

The break statement can also be used to jump out of a loop.

This example stops the loop when i is equal to 4:

Example

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
   break;
  }
  System.out.println(i);
}
```

Java Continue

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 4:

Example

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  System.out.println(i);
}
```

Break and Continue in While Loop

You can also use break and continue in while loops:

Break Example

```
int i = 0;
while (i < 10)  {
  System.out.println(i);
  i++;
  if (i == 4) {
    break;
  }
}
```

Continue Example

```
int i = 0;
while (i < 10) {
  if (i == 4) {
  i++;
    continue;
  }
  System.out.println(i);
  i++;
}
```

Java Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with square brackets:

```
String[] cars;
```

We have now declared a variable that holds an array of strings. To insert values to it, you can place the values in a comma-separated list, inside curly braces:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

To create an array of integers, you could write:

```
int[] myNum = {10, 20, 30, 40};
```

Access the Elements of an Array

You can access an array element by referring to the index number.

This statement accesses the value of the first element in cars:

Example

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars[0]);
// Outputs Volvo
```

Note: Array indexes start with 0: [0] is the first element. [1] is the second element, etc.

Change an Array Element

To change the value of a specific element, refer to the index number:

Example

```
cars[0] = "Opel";
```

Example

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
cars[0] = "Opel";
System.out.println(cars[0]);
// Now outputs Opel instead of Volvo
```

Array Length

To find out how many elements an array has, use the length property:

Example

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars.length);
// Outputs 4
```

Nested Loops in Programming

In programming, Nested Loops occur when one loop is placed inside another. These loops are quite useful in day-to-day programming to iterate over complex data structures with more than one dimension, such as a list of lists or a grid. In this article, we will learn about the basics of nested loops and how they are used in different programming languages.

Nested loops are programming structures where one or more loops are placed inside another loop. This allows for more complex control flow and repetitive execution in programs. Nested loops are commonly used in various programming languages to iterate over multidimensional arrays, perform matrix operations, and implement nested structures.

Syntax of Nested Loops:

The basic syntax for nested loops involves placing one loop inside another, creating a hierarchical structure. There are two main types of nested loops: inner loop and outer loop.

```
// Outer Loop
```

```
for (outer_initialization; outer_condition; outer_update) {

   // Inner Loop

   for (inner_initialization; inner_condition; inner_update) {

      // Loop body

   }

}
```

You can use any loop for nested loop

Example

```
class me
{
        public static void main(String args[])
        {
                int r,c;
                r=1;

    while(r<=5)// Outer Loop
                 {
                        c=1;
                        while(c<=r)// Inner Loop
                        {
                                System.out.print("*");
                                c++;
                        }
     System.out.print("\n");
                r++;
  }
        }

}
```

The end