Introduction To Computer

===

Computer - Electronic device which reduce human effort

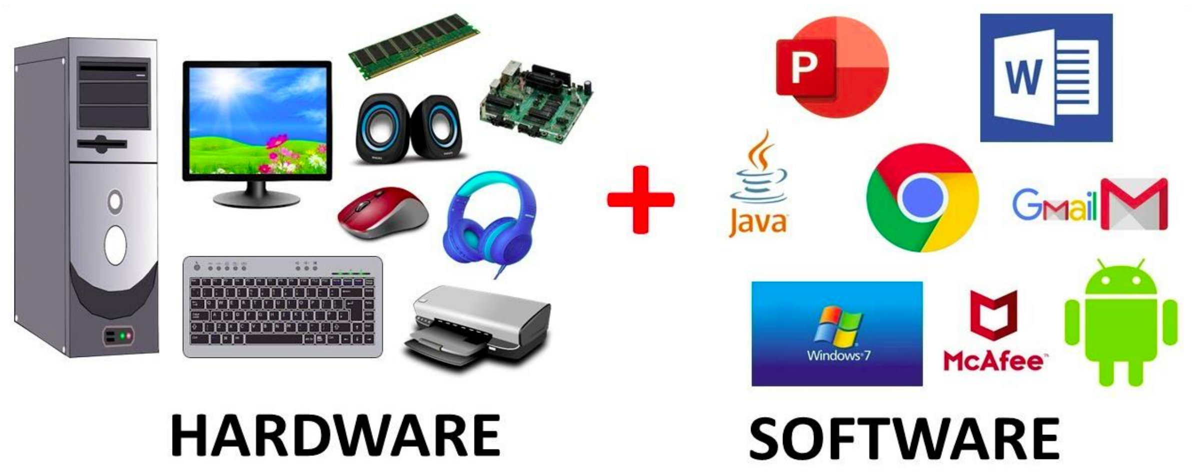
apple mac, lenovo, hp, dell etc...

Computer Hardware is defined as the physical part or component of a computer system which can be seen and touched.

Eg. Monitor, keyboard, mouse , CPU, printer etc

Computer software is defined as a set of instructions or collection of programs which are designed and developed to perform specific tasks.

Used to reduce human efforts



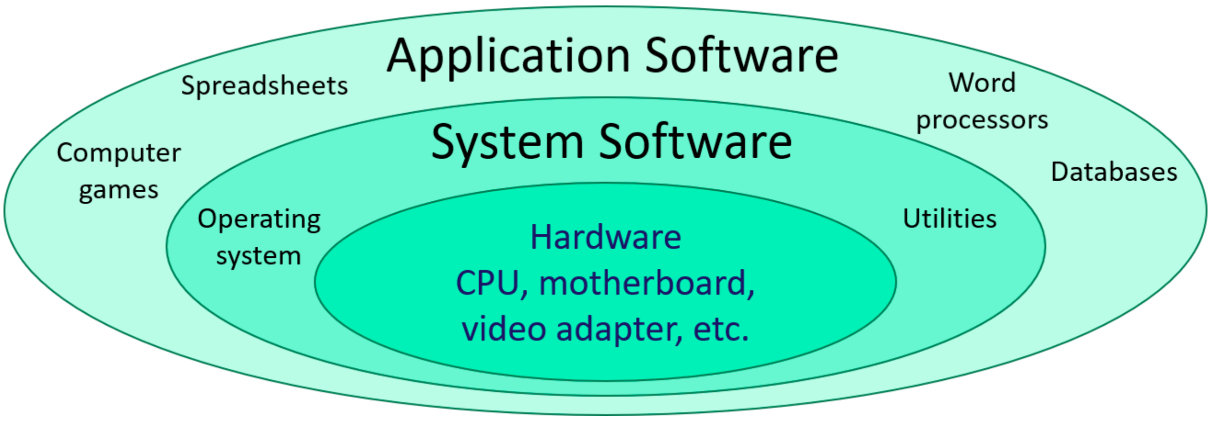
Types of Software

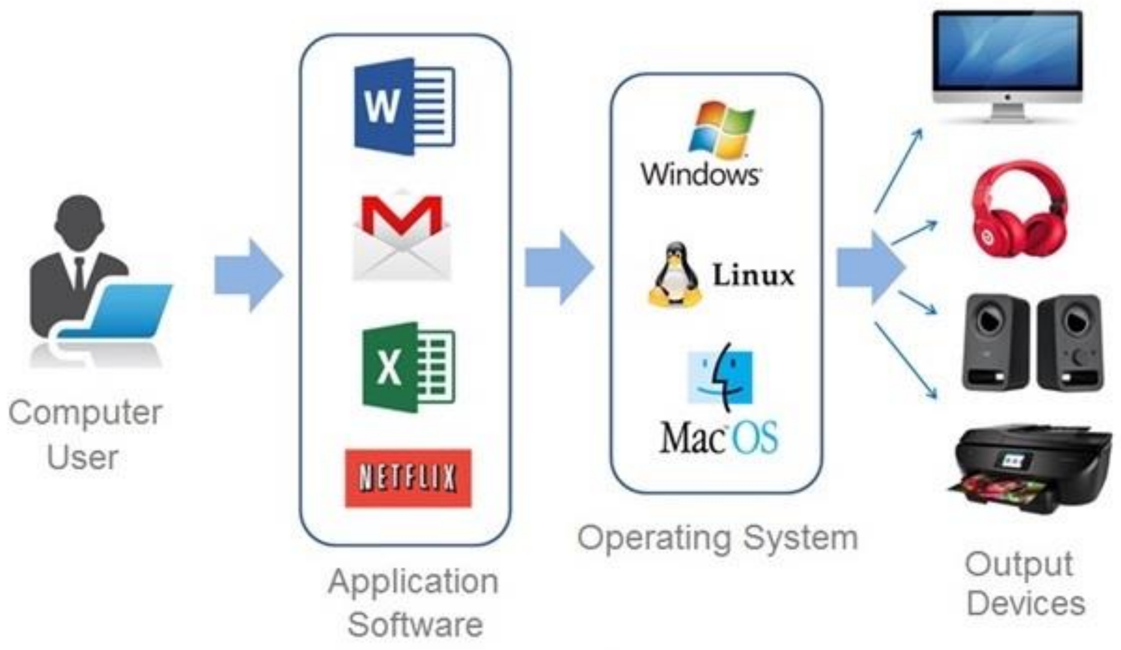
System Software -

- Also called as Platforms or Operating testing
- The ***system software*** is software which controls the hardware so that any application software can run and can be executed to perform various task mentioned by programmers.
- The primary examples of system software's are operating system such as Microsoft Windows, Linux, Mac, Unix, etc

Application Software -

- Application Software is a program that is designed and developed for specific purposes and used by user
- application software such as MS-OFFICE SUITE comes with MS Word, Excel, PowerPoint & Access and Adobe include Adobe Photoshop and Image ready together.





As we know, to communicate with a person, we need a specific language, similarly to communicate with computers, programmers also need a language is called Programming language.

What is Language?

Language is a mode of communication that is used to **share ideas, opinions with each other**.

What is a Programming Language?

A programming language is a **computer language** is used by **programmers (developers) to communicate with computers**.

What are different programming languages -

C, C++, Java, Python etc

So, it can also be defined. It is a set of instructions written in any specific language (C, C++, Java, Python) to perform a specific task.

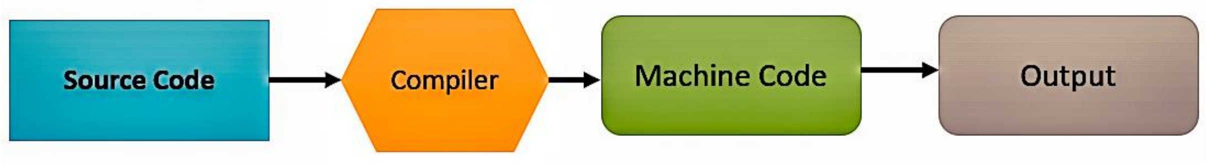
- A computer is a machine that can perform tasks according to the instructions provided by the user.
- Which is provided by user in the form of source code.

What is Source Code and machine code or byte code?

Source code is set of instructions those are human-readable that a programmer writes to perform specific task.

Machine code Byte code or Binary code - The codewhich is converted by compiler which will be udertstaabkle by computer

How Program get execute

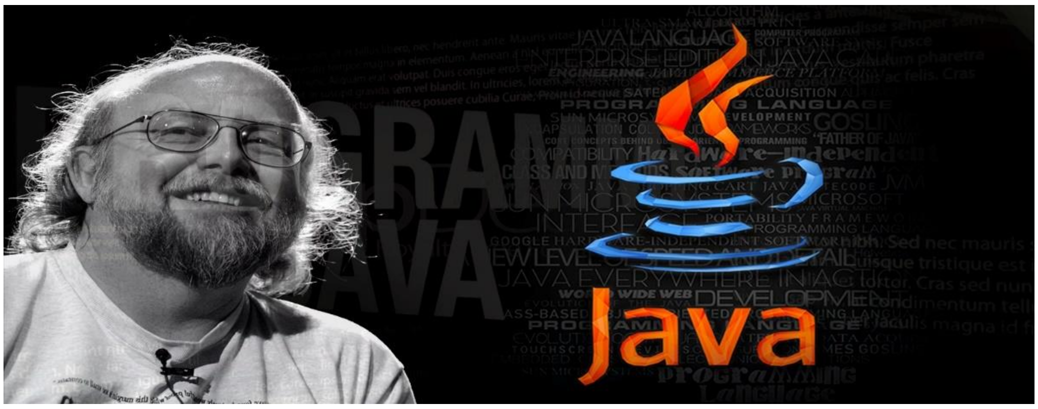


JAVA

==

What is Java?

- Java is general purpose programming language that is class based, object-oriented, platform independent and platform to develop an application.
- It provides software platform that runs on billions of devices, including notebook computers, mobile devices, gaming consoles, medical devices and many others.

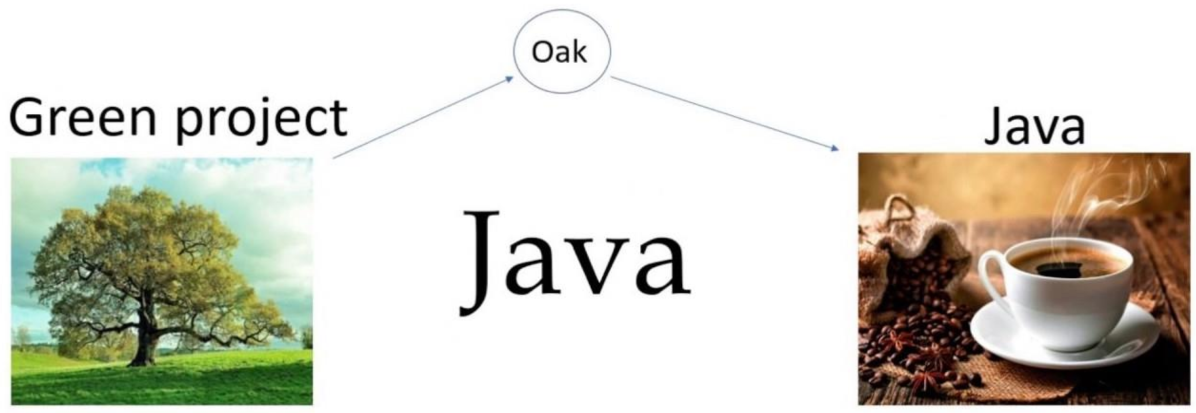


History of JAVA :-

1. Java was originally developed by James Gosling's at sun microsystem (which has been acquired by oracle in 2010) and released in 1995.
2. James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. The small team of sun engineers called Green Team.
3. Initially it was designed for small, embedded systems in electronic appliances like set-top boxes.
4. it was called Oak and was developed as a part of the Green project.

Why was Java named as "Oak"?

1. Why Oak? Oak is a symbol of strength and chosen as a national tree of many countries like the U.S.A., France, Germany, Romania, etc.
2. In 1995, Oak was renamed as "Java" because it was already a trademark by Oak Technologies.



Why is Java Programming named "Java"?

3. The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA", etc.
4. Java was so unique, most of the team members preferred Java over other names.
5. Java is an island in Indonesia where the first coffee was produced (called Java coffee). It is a kind of espresso bean. Java name was chosen by James Gosling while having a cup of coffee nearby his office.
6. In 1995, Time magazine called Java one of the Ten Best Products of 1995.

Java Version History

JDK 1.0 was released on January 23, 1996. After the first release of Java, there have been many additional features added to the language. Each new version adds new features in Java.

Sr.No	**Version**	**Release date**
1	JDK Alpha and Beta	1995
2	JDK 1.0	23rd Jan 1996
3	JDK 1.1	19th Feb 1997
4	J2SE 1.2	8th Dec 1998
5	J2SE 1.3	8th May 2000
6	J2SE 1.4	6th Feb 2002

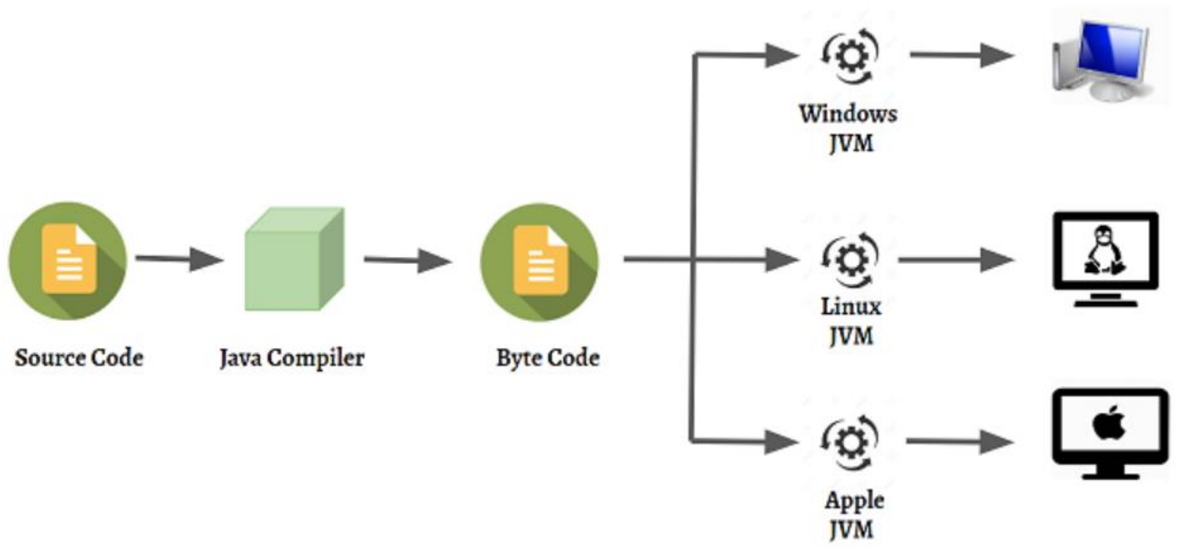
7	J2SE 5.0	30th Sep 2004
8	Java SE 6	11th Dec 2006
9	Java SE 7	28th July 2011
10	Java SE 8	18th Mar 2014
11	Java SE 9	21st Sep 2017
12	Java SE 10	20th Mar 2018
13	Java SE 11	September 2018
14	Java SE 12	March 2019
15	Java SE 13	September 2019
16	Java SE 14	Mar 2020
17	Java SE 15	September 2020
18	Java SE 16	Mar 2021
19	Java SE 17	September 2021
20	Java SE 18	to be released by March 2022

a. It promises WORA = "***Write One Run Anywhere***"
b. Since Java SE 8 release, the Oracle corporation follows a pattern in which every even version is release in March month and an odd version released in September month.

How Java is WORA?

- **WORA,** which is abbreviated as **Write Once Run Anywhere,** is the feature applicable to those programs which will be run on any operating systems or any machine. Sun Microsystem gave this terminology for their programming language - Java.
- According to this concept, the same code must run on any machine, and hence the source code needs to be portable.
- So Java allows run Java bytecode on any computer irrespective of the machine or the hardware, using JVM (Java Virtual Machine).

- The bytecode generated by the compiler is not platform-specific and hence takes the help of JVM to run on a wide range of machines. So we can call Java programs as a write once and run on any computer residing anywhere.



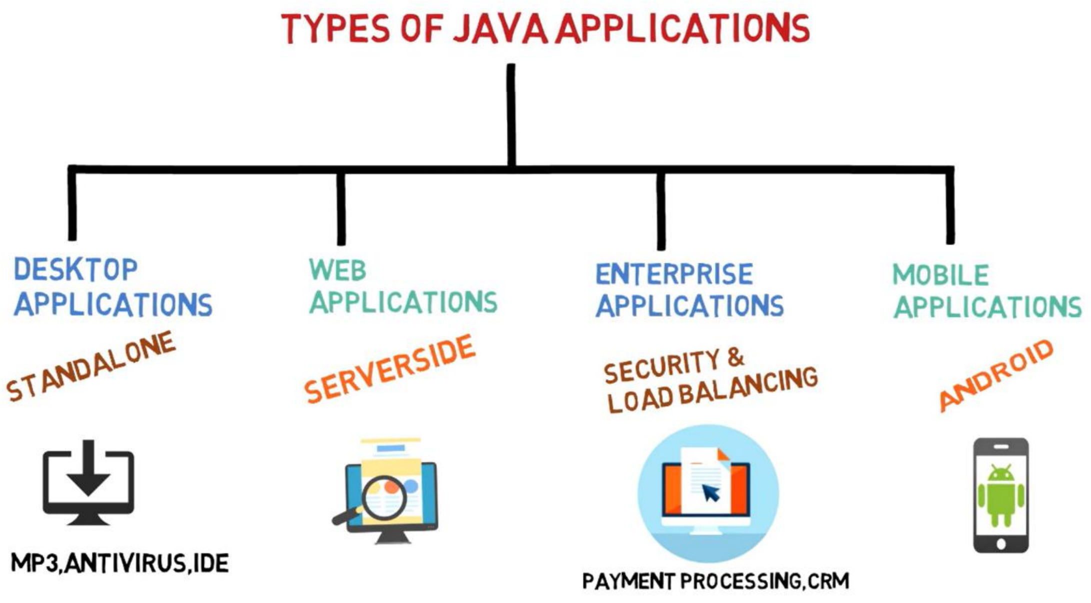
Application

According to Sun, 3 billion devices run Java. There are many devices where Java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus, etc.
2. Web Applications such as irctc.co.in, javatpoint.com, etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games, etc.

Types of Java Applications

There are mainly 4 types of applications that can be created using Java programming:



1. Standalone Application

a. Also called as Desktop based or windows-based application. These are software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc.

b. AWT and Swing are used in Java for creating standalone applications.

2. Web Application

c. An application that runs on the server side and creates a dynamic page is called a web application. Examples Facebook, amazon etc.

d. Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

3. Enterprise Application

e. An application that is distributed in nature, such as banking applications, etc. is called an enterprise application.

f. It has advantages like high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.

4. **Mobile Application**

   g. An application which is created for mobile devices is called a mobile application.

   h. Currently, Android and Java ME are used for creating mobile applications.

Java Platforms / Editions

1. **Java SE (Java Standard Edition)**

   a. It is a Java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc.

   b. It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

2. **Java EE (Java Enterprise Edition)**

   c. It is an enterprise platform that is mainly used to develop web and enterprise applications.

   d. It is built on top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, JPA, etc.

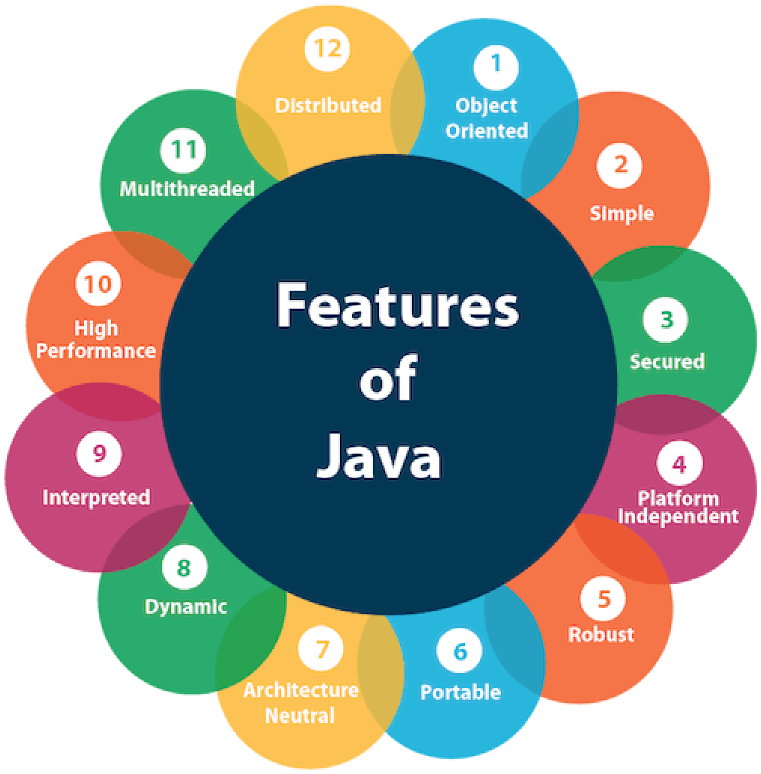
3. **Java ME (Java Micro Edition)**

   e. It is a micro platform that is dedicated to mobile applications.

Features of Java

A list of the most important features of the Java language is given below.

1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Interpreted
9. High Performance
10. Multithreaded
11. Distributed
12. Dynamic



- **Object Oriented** – In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- **Simple** – Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
- **Secure** – With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

- **Portable** – Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.
- **Platform Independent** – Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
- **Architecture-neutral** – Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.
- **Robust** – Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.
- **Interpreted** – Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
- **High Performance** – With the use of Just-In-Time compilers, Java enables high performance.
- **Multithreaded** – With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.
- **Distributed** – Java is designed for the distributed environment of the internet.
- **Dynamic** – Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

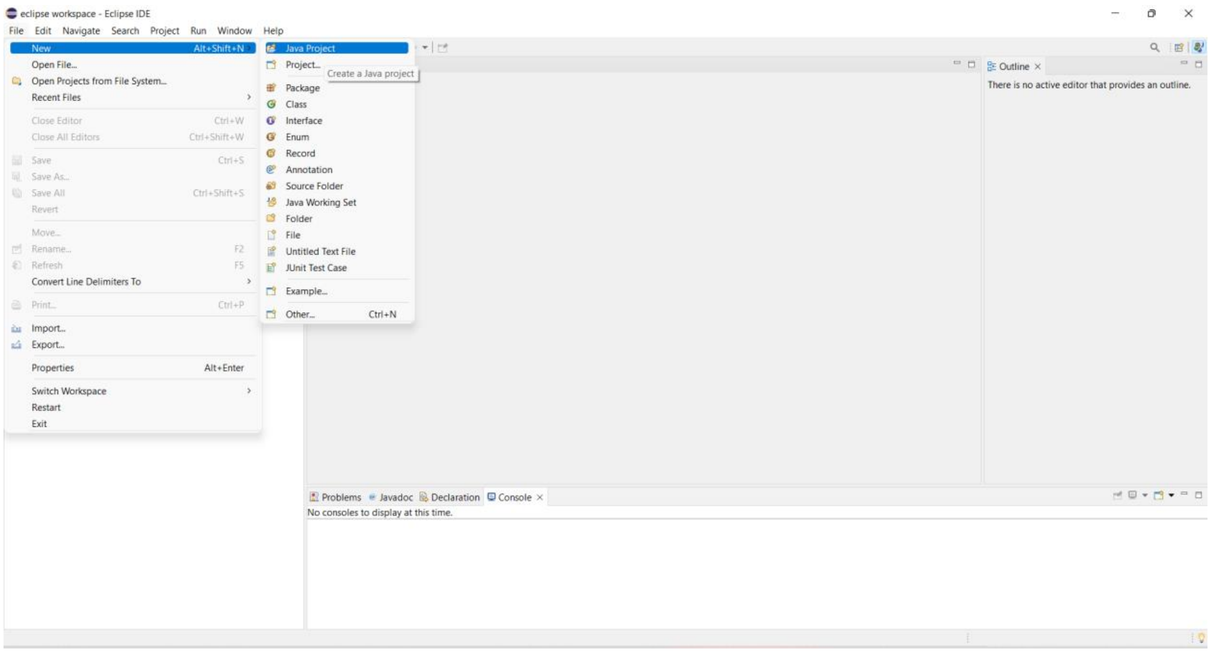
==

Description about java program structure

1. Project –

a. Project is a collection of multiple packages, multiple classes, user defined folder (for test data), supporting libraries, jar files and so on.

b. We can create many packages and classes inside a project.



There is no active editor that provides an outline.

No consoles to display at this time.

2. Package –

c. Package is a collection of multiple classes

d. There are two types of packages

   i. User defined package – If we create package

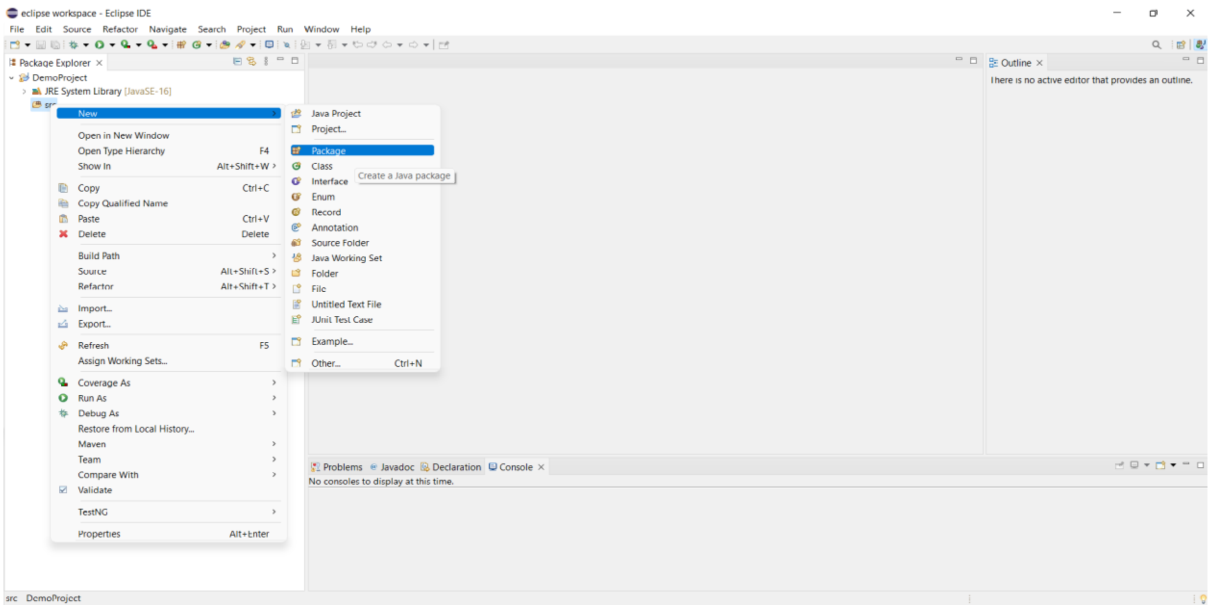
   ii. Default package – If we are created number of classes without generating package and shows under default package.

e. Syntax –

package package_name ;

f. If package does not contain class then symbol should be colourless

g. If class created inside package the symbol should be colourful



There is no active editor that provides an outline.

No consoles to display at this time.

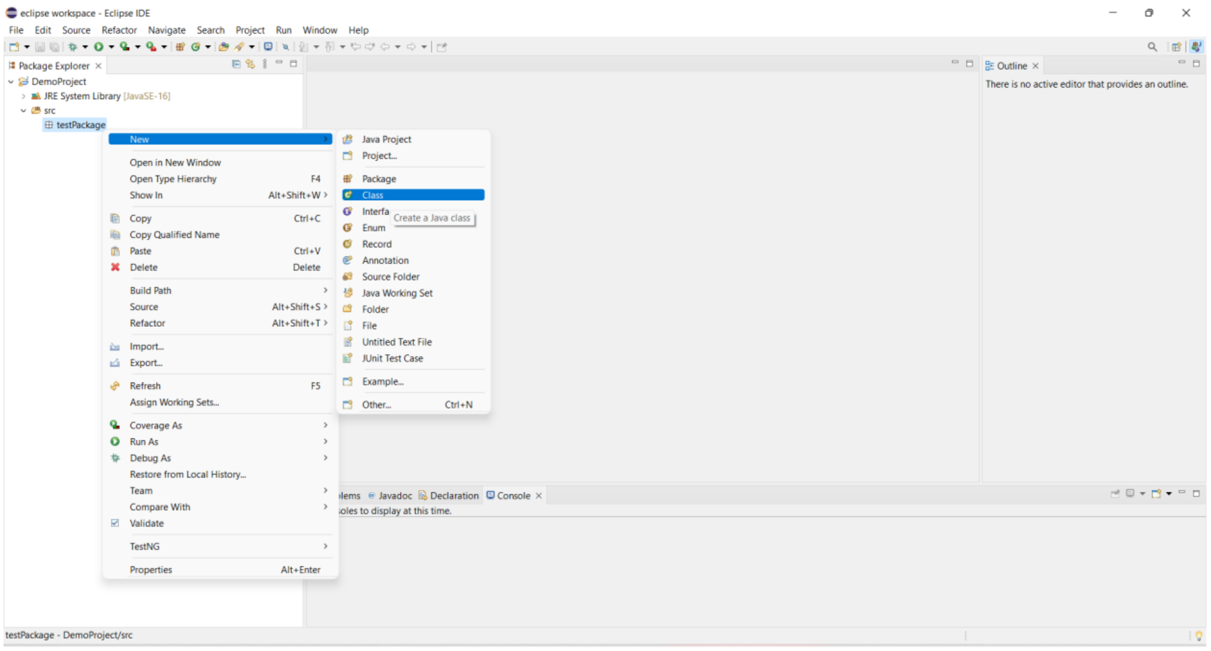
3. Class

a. A class is collection of member functions(methods), variable, datatypes, printing statements, scanning statement, object, constructor etc.

b. For all class name the first letter should be uppercase

c. If several words are used to form a name of the class each inner words first letter should be in upper case

Eg. public class MyFirstJavaProgram



There is no active editor that provides an outline.

oles to display at this time.

==

Creating a Java Project in Eclipse IDE

- How to create a project
  - Open Eclipse IDE Click on file >> New >> Java Project >> { Project name } >> Click on finish >> Don't Create
- How to create package
  - click on src >> right click >> new >> select package >> { package name } >> finish
- How to create class
  - Click on specific package>> right click >> new >> select class >> provide name >> finish

===

Case sensitivity/Name Convention -

Java is a case sensitive which means identifier "Hello" and ""hello" which have different meaning in java

1. **ProjectName**
   - All project name start with uppercase letter
   - If have multi name project, then uppercase of each first letter of word

   Eg. DemoProject

2. **PackageName**
   - all method starts with lower case letter
   - If have multi name package, then lowercase of first word and upper case of next words

   Eg. testPackage

3. **ClassName**
   - All classname start with uppercase letter
   - If have multi name class, then uppercase of each first letter of word

   Eg. public class MyFirstJavaProgram

4. **methodName**
   - All method starts with lower case letter

- If have multi name method, then lowercase of first word and upper case of next words

  Eg. myMethodName()

Basic Terms and Rules

5. Comments(//)

- These lines are used for comments or writing some statement that will be not considered during execution of programs

6. Signature Bracket ()

- This bracket we called as signature bracket or argument bracket

7. + Sign

- This + sign is ued for concatinating or we can say that join the string or statements

8. JRE library

- In JRE library there are supporting jar files to write jave programs

9. { }

- It is called as body or curly brakes

  eg. class body, method body etc.

10.src

- src is a source folder where the project source file content programs

11.System.out.println()

- System - it is name of jave utility class
- out - it is object which belongs to system class to call predefined method
- println - it is predefined method or utility method which is used to send any string to console

==

Method/Member Function in Java

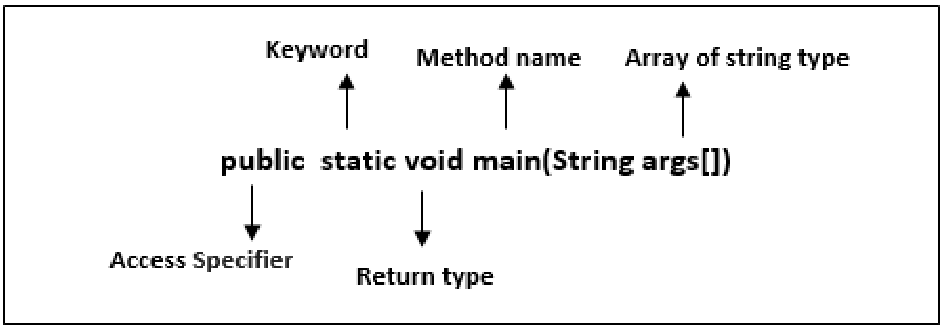
- Method in java is a collection of instructions that perform specific task. Without method we are not able to write program.
- It is also called member function
- It is memory location in JVM to store the data and information
- Why use methods? To reuse code: define the code once, and use it many times.
- A method must be declared within a class.

There are two types of methods

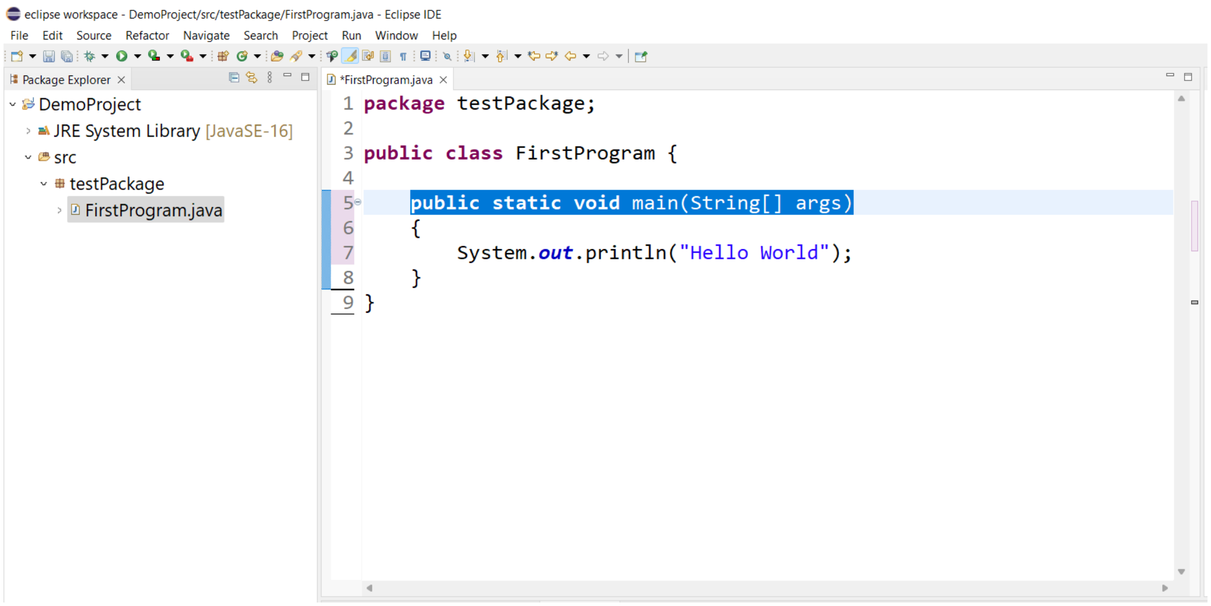
1. Business method – main method
2. Regular method

Java main() method

- The main() is the starting point for JVM to start execution of a Java program. Without the main() method, JVM will not execute the program.
- The syntax of the main() method is:



- **public:** It is an access specifier. We should use a public keyword before the main() method so that JVM can identify the execution point of the program.
- **static:** You can make a method static by using the keyword static. We should call the main() method without creating an object.
- **void:** In Java, every method has the return type. Void keyword in main() method does not return any value.
- **main():** It is a default method name which is predefined in the JVM.
- **String args[]:** The main() method also accepts some data from the user. It accepts a group of strings, which is called a string array.


eclipse workspace - DemoProject/src/testPackage/FirstProgram.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
DemoProject
JRE System Library [JavaSE-16]
src
testPackage
FirstProgram.java
*FirstProgram.java

```java
package testPackage;

public class FirstProgram {

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

Regular method

Two types of methods

1. Static method
2. Non-Static method

1. Static Method

- ➢ The method which contains static keyword and used to perform particular task
- ➢ A static method in Java is a method that is part of a class rather than an instance of that class. (To call static method there is no need to create object.)
- ➢ It is static in nature
- ➢ Main method is the best example of static method
- ➢ It can be represented as

public static void methodName()

{

Body of the program for execution.

}

- ➢ It can call as ***ClassName.methodName();*** or only ***methodName();***
    - ❖ Class variables and methods can be accessed using the class name followed by a dot and the name of the variable or method.

2. Non-Static Method

- The method which don't have static keyword
- It is dynamic in nature
- To call non-static method we need to create object

```
public void methodName()
{
        Body of the program for execution.
}
```

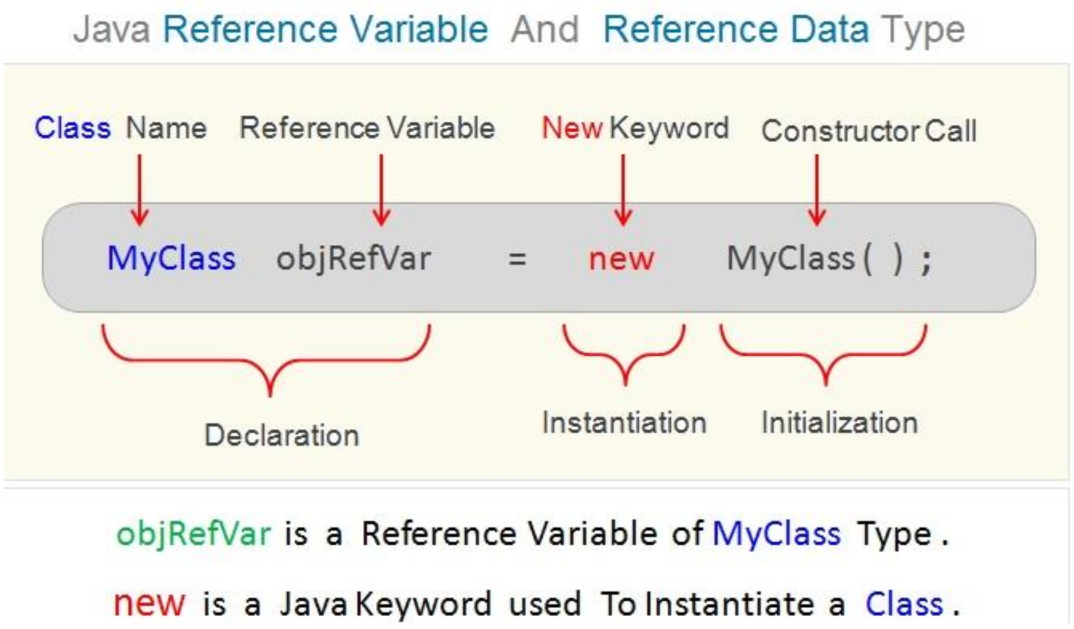
- It can call as object.methodName();

Object

- It is instance of class/example of class and used to call non-static method.
- An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical.

How to Create Object in Java using new keyword

```
ClassName object = new ClassName();
```


Java Reference Variable And Reference Data Type
Class Name
Reference Variable
New Keyword
Constructor Call
MyClass objRefVar = new MyClass ( ) ;
Declaration
Instantiation
Initialization
objRefVar is a Reference Variable of MyClass Type .
new is a Java Keyword used To Instantiate a Class .

ClassName

- ❖ It is name of class at the time of object creation.
- ❖ Class is used as datatype to declare reference variable.

ObjectReferenceVariable

- ❖ Used to provide the reference of object.

New

- ❖ It is keyword used to create object of class.

Constructor();

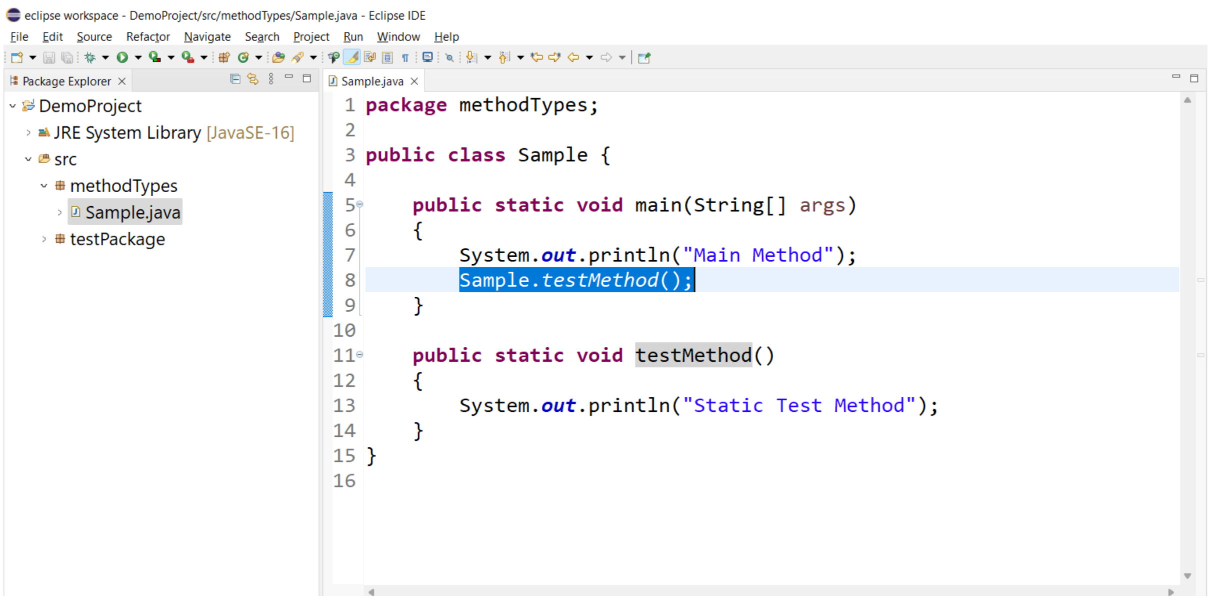
- ❖ Constructor name should be same as class name.
- ❖ Used to initialize the information of the particular class.

==

Why is the main method in Java static?

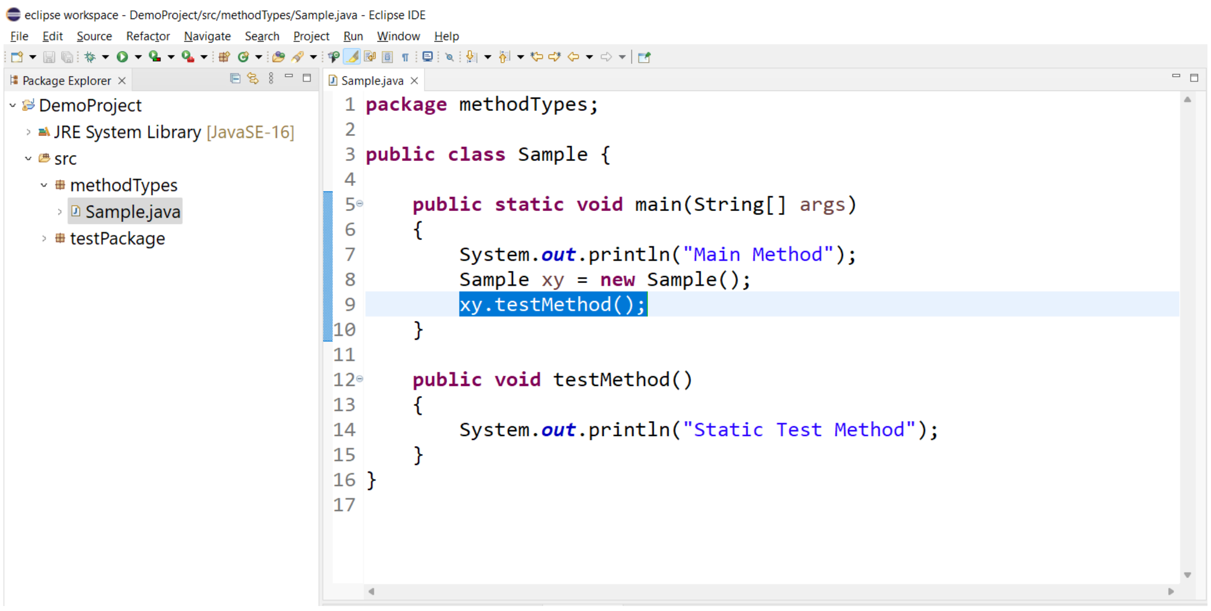
It's because calling a static method isn't needed of the object. If it were a non-static function, JVM would first build an object before calling the main() method, resulting in an extra memory allocation difficulty.

Static Method Example -


eclipse workspace - DemoProject/src/methodTypes/Sample.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
DemoProject
JRE System Library [JavaSE-16]
src
methodTypes
Sample.java
testPackage
Sample.java

```java
package methodTypes;

public class Sample {

    public static void main(String[] args)
    {
        System.out.println("Main Method");
        Sample.testMethod();
    }

    public static void testMethod()
    {
        System.out.println("Static Test Method");
    }
}
```

Non Static Method Example -


eclipse workspace - DemoProject/src/methodTypes/Sample.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
DemoProject
JRE System Library [JavaSE-16]
src
methodTypes
Sample.java
testPackage
Sample.java

```java
package methodTypes;

public class Sample {

    public static void main(String[] args)
    {
        System.out.println("Main Method");
        Sample xy = new Sample();
        xy.testMethod();
    }

    public void testMethod()
    {
        System.out.println("Static Test Method");
    }
}
```

==

Data Types in Java

==

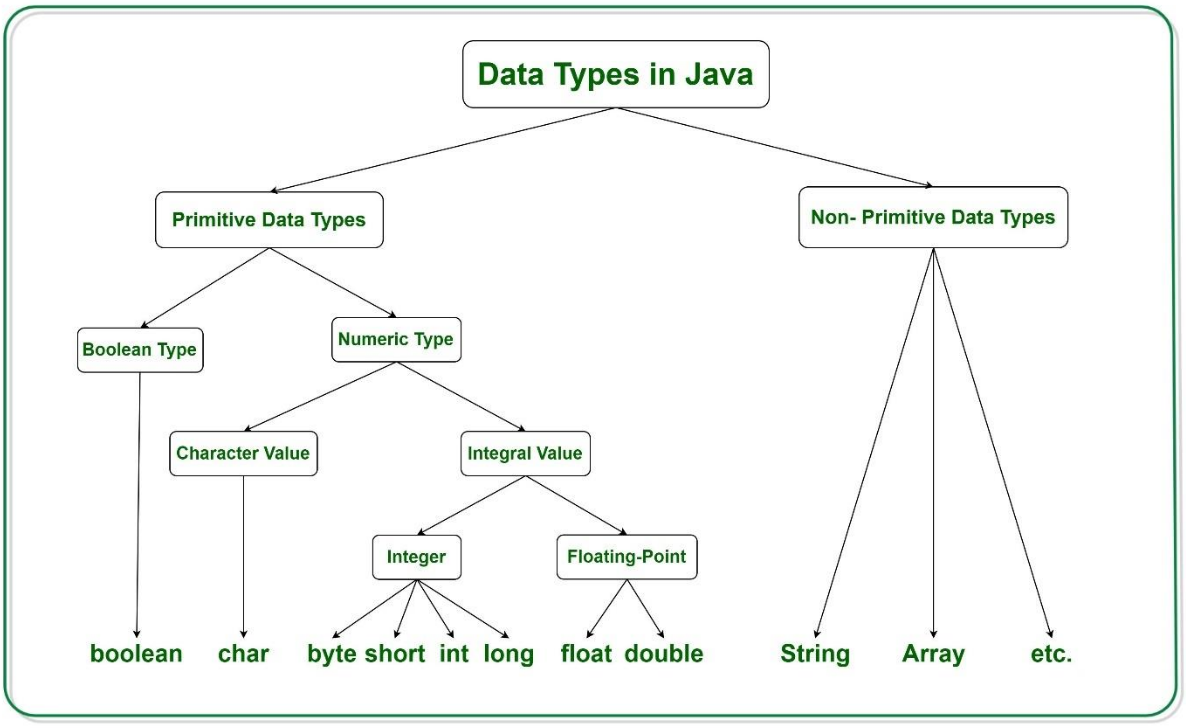
- ❖ Datatypes are used to represent types of data or information that we going to used in the programming.
- ❖ It is mandatory / compulsory to declare datatype before variable.

Types of data types in Java:

1. **Primitive data types:**
   - ➢ The primitive data types include boolean, char, byte, short, int, long, float and double.
   - ➢ Have fixed size
2. **Non-primitive data types:**
   - ➢ The non-primitive data types include Classes, Interfaces, and Arrays.
   - ➢ Not have fixed size

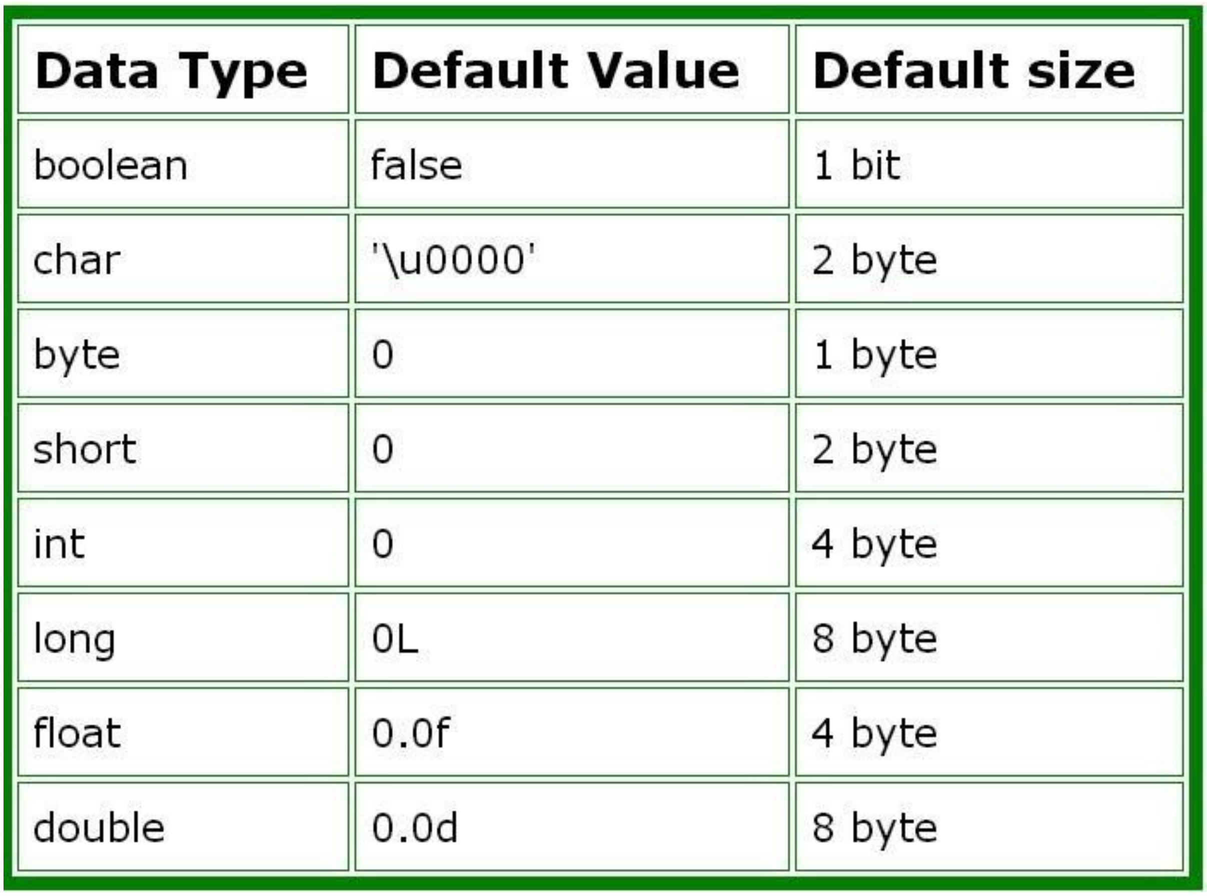
Java Primitive Data Types

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.



There are 8 types of primitive data types:

1. boolean data type
2. byte data type
3. char data type
4. short data type
5. int data type
6. long data type
7. float data type
8. double data type



Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

1. Boolean Data Type

- The Boolean data type is used to store only two possible values: true and false.
- This data type is used for simple flags that track true/false conditions.

- The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

  Example: Boolean one = **false**

2. Byte Data Type

- The byte data type is an example of primitive data type.
- Its value-range lies between -128 to 127
- Its minimum value is -128 and maximum value is 127. Its default value is 0.
- The byte data type is used to save memory in large arrays where the memory savings is most required.

  Example: **byte** a = 10, **byte** b = -20

3. Short Data Type

- Its value-range lies between -32,768 to 32,767
- Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.
- The short data type can also be used to save memory just like byte data type.

  Example: **short** s = 10000, **short** r = -5000

4. Int Data Type

- Its value-range lies between - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1)
- Its minimum value is - 2,147,483,648 and maximum value is 2,147,483,647. Its default value is 0.
- The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

  Example: **int** a = 100000, **int** b = -200000

5. Long Data Type

- Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1).
- Its minimum value is - 9,223,372,036,854,775,808 and maximum value is 9,223,372,036,854,775,807. Its default value is 0.
- The long data type is used when you need a range of values more than those provided by int.

  Example: long a = 100000L, long b = -200000L

6. Float Data Type

- The float data type is a single-precision 32-bit IEEE 754 floating point.
- Its value range is unlimited.
- It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers.
- The float data type should never be used for precise values, such as currency.
- Its default value is 0.0F.

  Example: float f1 = 234.5f

7. Double Data Type

- The double data type is a double-precision 64-bit IEEE 754 floating point.
- Its value range is unlimited.
- The double data type is generally used for decimal values just like float.
- The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

  Example: double d1 = 12.3

8. Char Data Type

- The char data type is a single 16-bit Unicode character.
- Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535).

- The char data type is used to store characters.

  Example: `char letterA = 'A'`

Why char uses 2 byte in java and what is \u0000 ?

It is because java uses Unicode system not ASCII code system. The \u0000 is the lowest range of Unicode system. To get detail explanation about Unicode visit next page.

9. String

- String is a sequence of characters. But in Java, string is an object that represents a sequence of characters.
- A String variable contains a collection of characters surrounded by double quotes:

  Eg. `String s="Welcome To Velocity";`

===

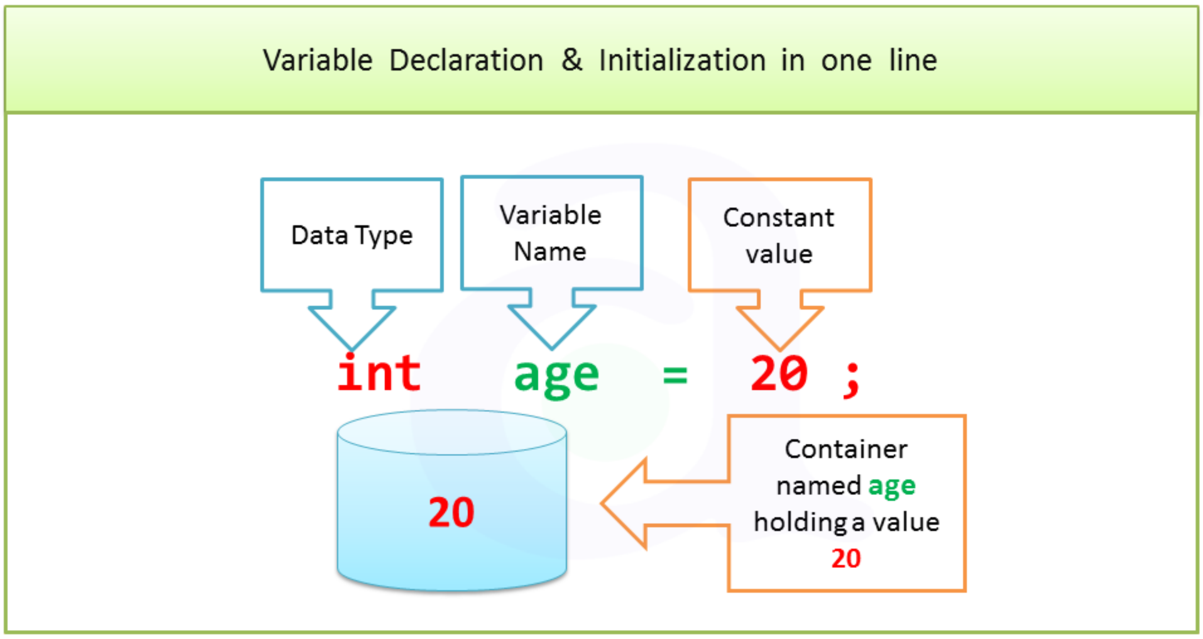
Variables

==

- Variable is nothing but piece of memory used to store information.
- A variable is a name given to a memory location. It is the basic unit of storage in a program.
- A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.

1. According to all programming language we cannot declare information directly, so variables are introduced to declare information and to store it.
2. In java programming variable can't be declare directly to store info so we have declared datatype before it.
3. **Reusable** – It help us the info will use again and again

How to declare variables?

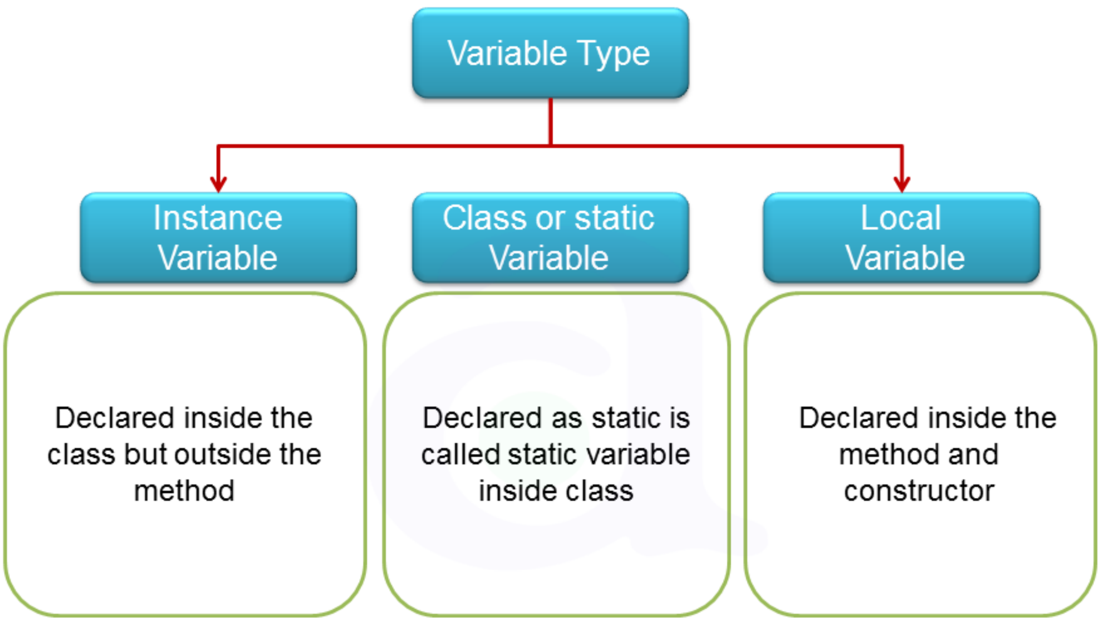

Variable Declaration & Initialization in one line
Data Type
Variable Name
Constant value
int age = 20 ;
20
Container named age holding a value 20

datatype variable = information;

- **datatype**: Type of data that can be stored in this variable.
- **Variable name**: Name given to the variable.
- **value**: It is the initial value stored in the variable.

Types of Variable

1. Local Variable
2. Static Variable/Class Variable
3. Global Variable/Instance Variable


Variable Type
Instance Variable
Class or static Variable
Local Variable
Declared inside the class but outside the method
Declared as static is called static variable inside class
Declared inside the method and constructor

1. Local Variable
   - ➢ The variables which are declared inside method body, method block or constructor called as local variables
   - ➢ We can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.
   - ➢ These variables are temporary variable
   - ➢ It cannot be defined by static keyword
   - ➢ Initialization of the local variable is mandatory before using it
2. Static Variable
   - ➢ Static variables also known as class variable
   - ➢ Static variables are declared in class body but outside of method/block with static keyword before it.
   - ➢ We have only one copy of it.

- Initialization of static variable is not necessary
- Static variables are directly accessible to any method
- We have default values

int, byte, short, long	0
float double	0.0
String	null
boolean	false

3. Global Variable
   - Also called as Instance/non-static variable
   - Global variables are declared in class body but outside of method/block
   - Initialization of global variable is not necessary
   - Global variables are only accessible in non-static method, not able to access in static method. But if we want to call in main method we can call it with object.
   - We have default values as

int, byte, short, long	0
float double	0.0
String	null
boolean	false

==

Constructor

===

- ➢ Constructor are used to initialize of data members(variables) of class and to load non-static member/methods into object
- ➢ Constructor is a block of code similar to method
- ➢ Constructors are special member of class
- ➢ All classes have constructors, whether you define one or not, because Java automatically provides a default constructor that initializes all member variables to default values. However, once you define your own constructor, the default constructor is no longer used.
- ➢ At the time of constructor declaration below are points need to followed
  - o Constructor name should be same as class name
  - o We should not declare any return type for Constructor
  - o Any number of Constructor can be declared in class but name should same as class name but having different argument.

Ways to call a constructor

1. By creating object of a class
   - ➢ Constructor executes automatically when we create an object.
2. By using 'new' keyword with constructor name with method signature followed by semicolon

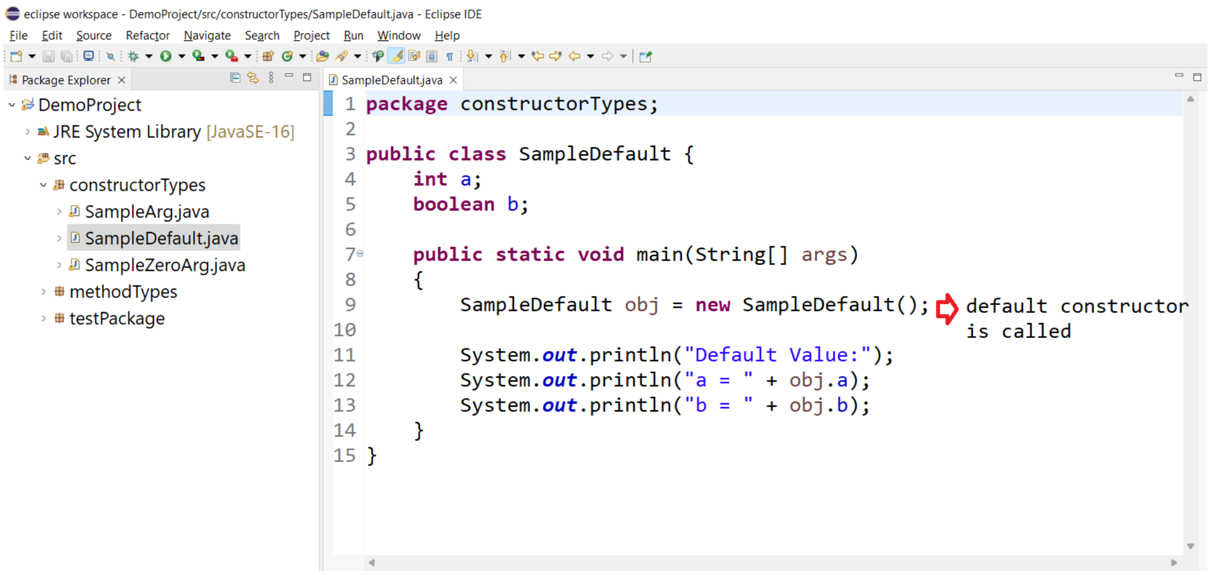
   new ConstructorName();

There are 2 types of constructor

1. Default Constructor
2. User defined Constructor
   a. Zero Argument/ Non Argument Constructor
   b. Parameterized/ Argument Constructor

1. Default Constructor

- If Constructor is not declared in class, then at the time compilation compiler will provide/consider the constructor.
- If we do not create any constructor, the Java compiler automatically create a no-arg constructor during the execution of the program. This constructor is called default constructor.
- It initializes all member variables to their default values
- The best example of default constructor is main method


eclipse workspace - DemoProject/src/constructorTypes/SampleDefault.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
SampleDefault.java
DemoProject
JRE System Library [JavaSE-16]
src
constructorTypes
SampleArg.java
SampleDefault.java
SampleZeroArg.java
methodTypes
testPackage

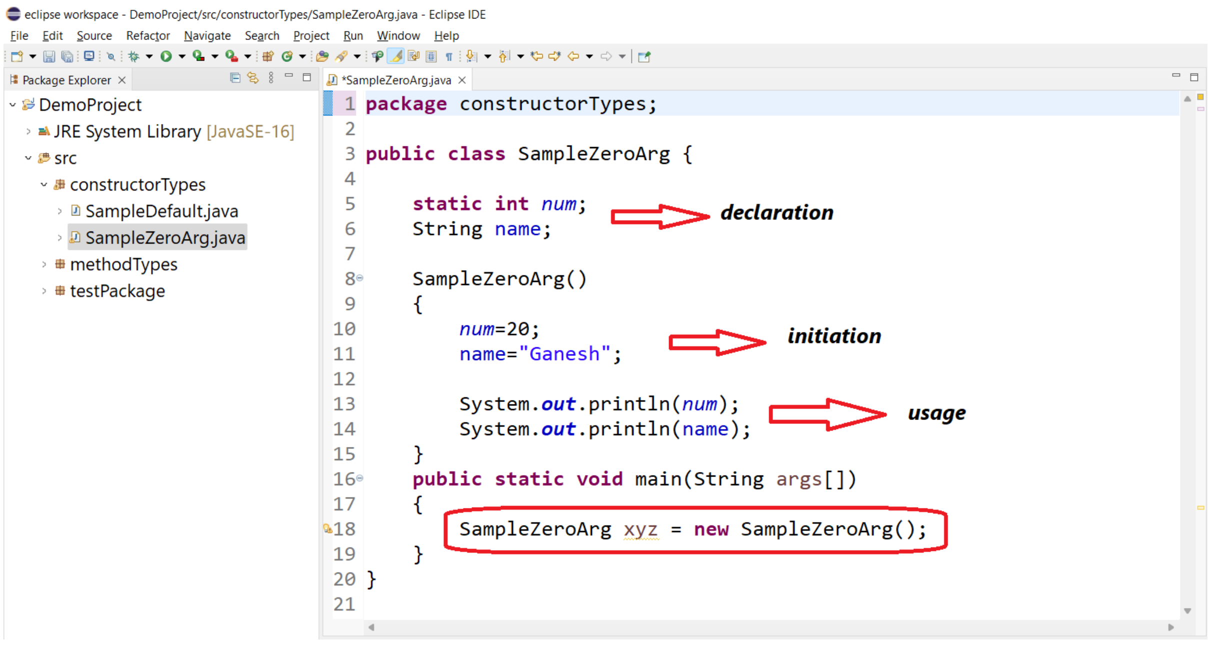
```
package constructorTypes;

public class SampleDefault {
    int a;
    boolean b;

    public static void main(String[] args)
    {
        SampleDefault obj = new SampleDefault();   default constructor
                                                   is called

        System.out.println("Default Value:");
        System.out.println("a = " + obj.a);
        System.out.println("b = " + obj.b);
    }
}
```

2. User defined constructor

- If programmer declared constructor in Java class, then it is considered to be user defined constructor
- User defined constructor is divided into two types
  1. Zero argument constructor and
  2. parameterised constructor

1. Zero argument constructor

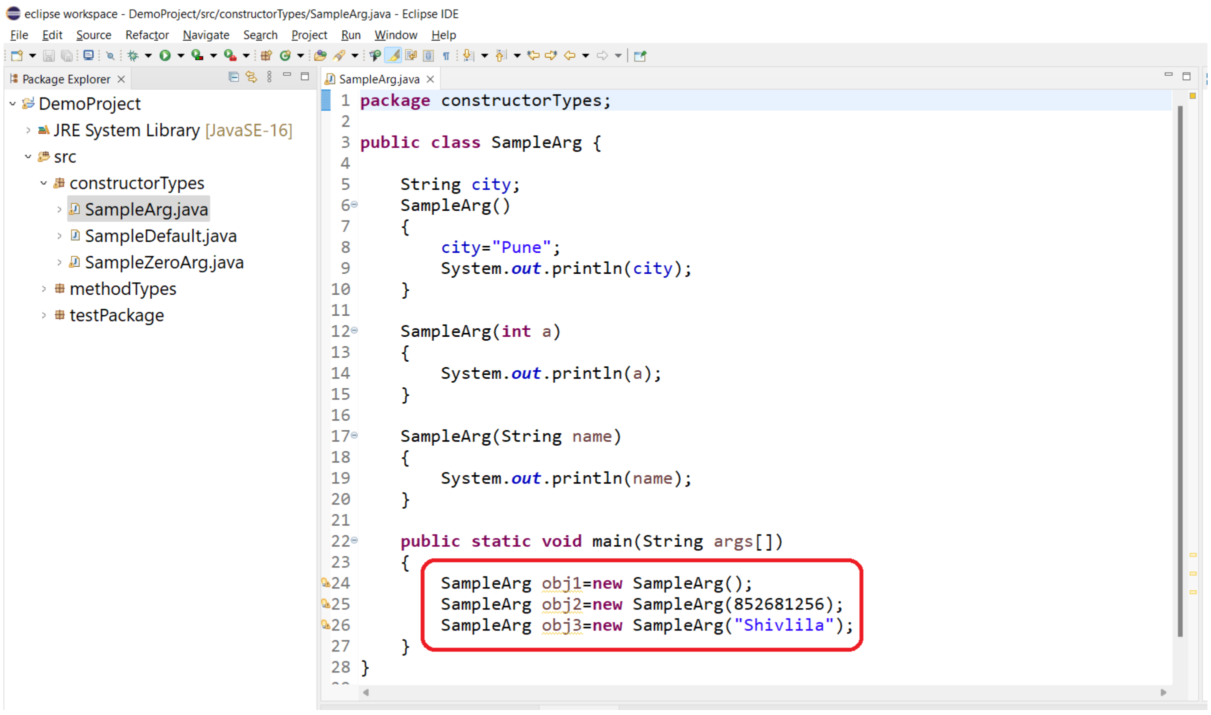
- It can be used to initialise data member of a class
- Constructor with no argument is known as zero argument constructor or non-argument constructor


eclipse workspace - DemoProject/src/constructorTypes/SampleZeroArg.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
DemoProject
JRE System Library [JavaSE-16]
src
constructorTypes
SampleDefault.java
SampleZeroArg.java
methodTypes
testPackage
*SampleZeroArg.java
declaration
initiation
usage

```java
package constructorTypes;

public class SampleZeroArg {

    static int num;
    String name;

    SampleZeroArg()
    {
        num=20;
        name="Ganesh";

        System.out.println(num);
        System.out.println(name);
    }
    public static void main(String args[])
    {
        SampleZeroArg xyz = new SampleZeroArg();
    }
}
```

2. **Parametrize constructor**

- Constructor with arguments in signature bracket is called as parameterised constructor
- We can declare multiple constructors having different arguments (distinct parameters) in class


eclipse workspace - DemoProject/src/constructorTypes/SampleArg.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
DemoProject
JRE System Library [JavaSE-16]
src
constructorTypes
SampleArg.java
SampleDefault.java
SampleZeroArg.java
methodTypes
testPackage
SampleArg.java

```java
package constructorTypes;

public class SampleArg {

    String city;
    SampleArg()
    {
        city="Pune";
        System.out.println(city);
    }

    SampleArg(int a)
    {
        System.out.println(a);
    }

    SampleArg(String name)
    {
        System.out.println(name);
    }

    public static void main(String args[])
    {
        SampleArg obj1=new SampleArg();
        SampleArg obj2=new SampleArg(852681256);
        SampleArg obj3=new SampleArg("Shivlila");
    }
}
```

Important Notes –

- ❖ Access scope of constructors will be same as a class access, it means if class is a public then constructor is a public, if class is private then constructor is also private or vice versa
- ❖ Constructors are going to invoke/use at the time of object creation at the time of object creation
- ❖ A constructor cannot be abstract or static or final.
- ❖ A constructor can be overloaded but cannot be overridden.

How Constructors are different from Methods in Java?

- ❖ Constructors must have the same name as the class within which it is defined while it is not necessary for the method in Java.
- ❖ Constructor does not have any return type while method have the return type or void which does not return any value.
- ❖ Constructors are called only once at the time of Object creation while method(s) can be called any number of times.

Constructor Overloading -

- ❖ The class allows to define multiple constructors by providing different types of arguments in simple words constructors with argument is known as parameters constructors where constructor overloading is placed
- ❖ **Def –** In a class if we have multiple constructors with different argument but having same name is known as constructor overloading.

==

Operators in Java

==

- Operators in Java are symbols that are used to perform operations on variables and values.

Types:-

1. Unary Operator,
2. Arithmetic Operator,
3. Relational Operator,
4. Shift Operator,
5. Bitwise Operator,
6. Logical Operator,
7. Ternary Operator and
8. Assignment Operator

Java Operator Precedence

Operator	**Category**	**Precedence**
Unary	postfix	*expr*++ *expr*--
	prefix	++*expr* --*expr*
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >=
	equality	== !=
Bitwise	AND	&
	OR	|
	XOR	^
Logical	AND	&&
	OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>=

1. Java Unary Operator

- The Java unary operators require only one operand.
- Unary operators are used to perform operations like
  - incrementing/decrementing a value by one
  - negating an expression

Operator	Meaning	Work
a++ a--	postfix	Print + operation
++a --a	prefix	Operation + print

2. Java Arithmetic Operators

- Java arithmetic operators are used to perform addition, subtraction, multiplication, and division.
- They act as basic mathematical operations.

Operator	Meaning	Work
+	Addition	To add two operands.
-	Subtraction	To subtract two operands.
*	Multiplication	To multiply two operands.
/	Division	To divide two operands.
%	Modulus	To get the area of the division of two operands.

3. Relational operators

- The **Java Relational operators** compare between operands and determine the relationship between them.
- The output of the relational operator is (true/false) boolean value

Operator	Meaning
==	Is equal to
!=	Is not equal to
>	Greater than
<	Less than
>=	Greater than or equal to

<=	Less than or equal to

4. Shift Operator

- It is used to shift all bits in value to left/right side of specified number in times
- **Left Shift Operator**
  - The Java left shift operator **<<** is used to shift all of the bits in a value to the left side of a specified number of times.
- **Java Right Shift Operator**
  - The Java right shift operator **>>** is used to move the value of the left operand to right by the number of bits specified by the right operand.

5. Logical Operators

- These operators are used to perform logical "AND", "OR" and "NOT" operation.
- They are used to combine two or more conditions.

Operator	**Meaning**	**Working**
&&	AND	True --- All conditions need true False --- any one condition will be false
||	OR	True --- any one condition is True False -- all condition false
!	Not	Invert conditions

6. Bitwise Operators

Operator	**Meaning**	**Work**
&	AND Operator	True - All condition true False - Any one condition false
|	OR Operator	True - any one condition is true False - all condition false
^	XOR Operator	False - Both condition same / T or F True - Have different values of conditions

7. Assignment Operators

- The **Java Assignment Operators** are used when you want to assign a value to the expression. The assignment operator denoted by the single equal sign =.
- In a Java assignment statement, any expression can be on the right side and the left side must be a variable name.

Operator	Example
=	C = A + B will assign value of A + B into C
+=	C += A is equivalent to C = C + A
-=	C -= A is equivalent to C = C – A
*=	C *= A is equivalent to C = C * A
/=	C /= A is equivalent to C = C / A
%=	C %= A is equivalent to C = C % A
<<=	C <<= 2 is same as C = C << 2
>>=	C >>= 2 is same as C = C >> 2
&=	C &= 2 is same as C = C & 2
^=	C ^= 2 is same as C = C ^ 2
|=	C |= 2 is same as C = C | 2

8. Java Ternary Operator

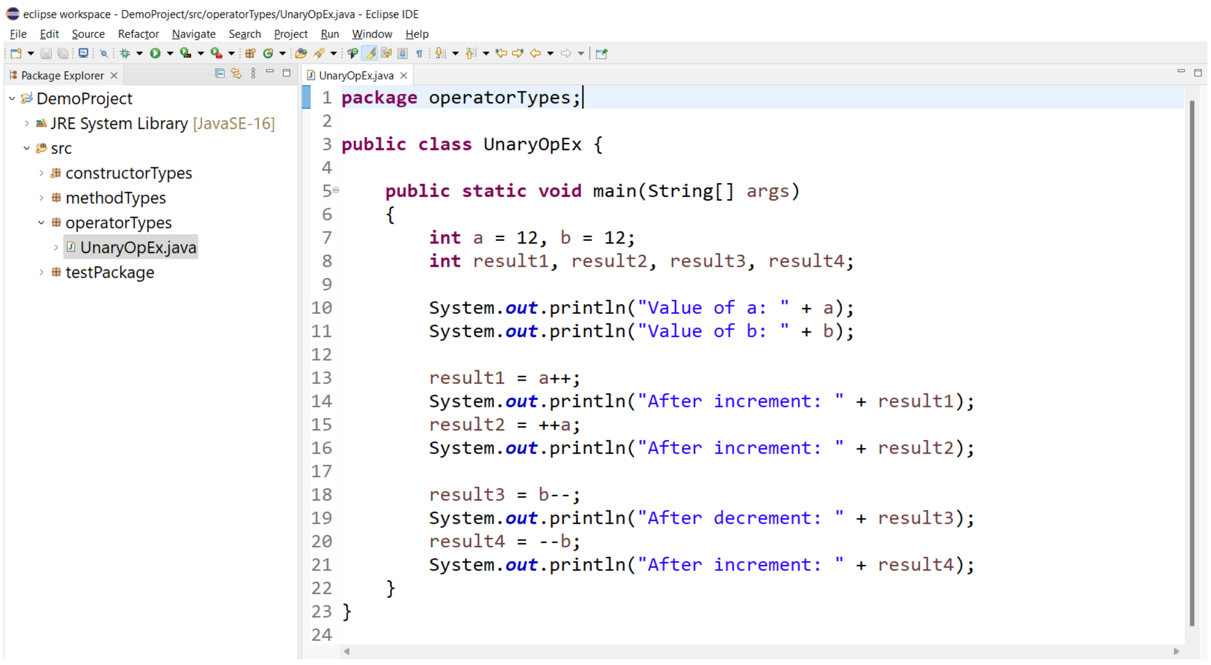
- The ternary operator (conditional operator) is shorthand for the if-then-else statement.

  variable = Expression ? expression1 : expression2

Here's how it works.

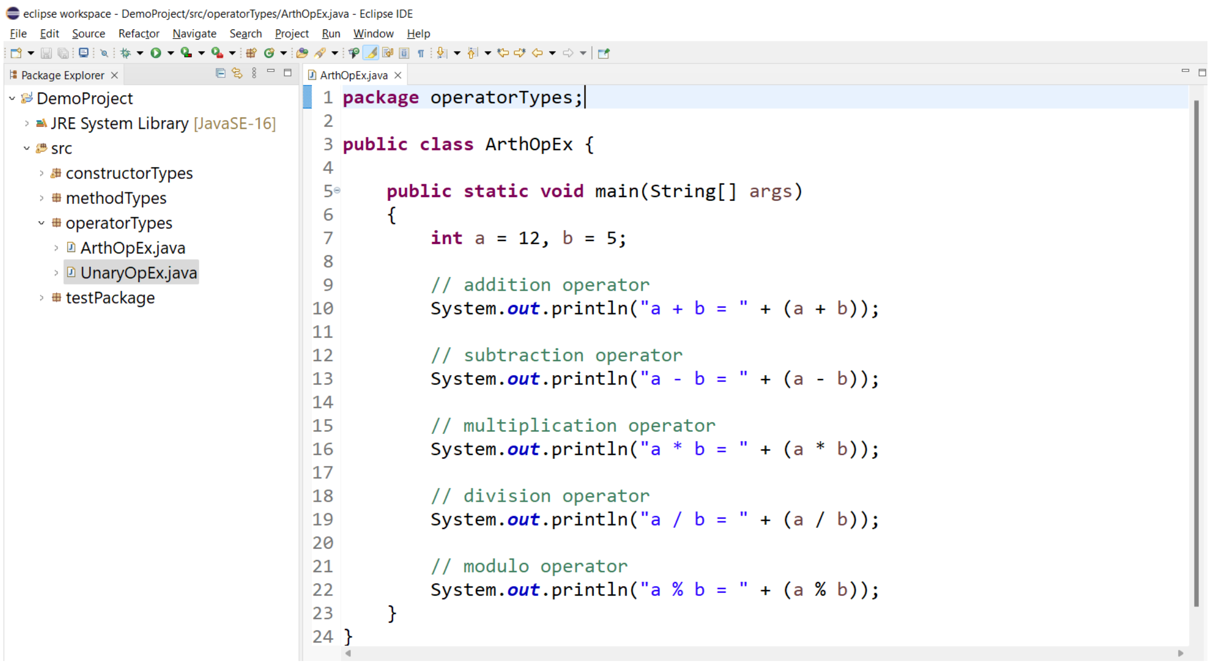
- If the Expression is true, expression1 is assigned to the variable.
- If the Expression is false, expression2 is assigned to the variable

Unary Operator


eclipse workspace - DemoProject/src/operatorTypes/UnaryOpEx.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
UnaryOpEx.java
DemoProject
JRE System Library [JavaSE-16]
src
constructorTypes
methodTypes
operatorTypes
UnaryOpEx.java
testPackage

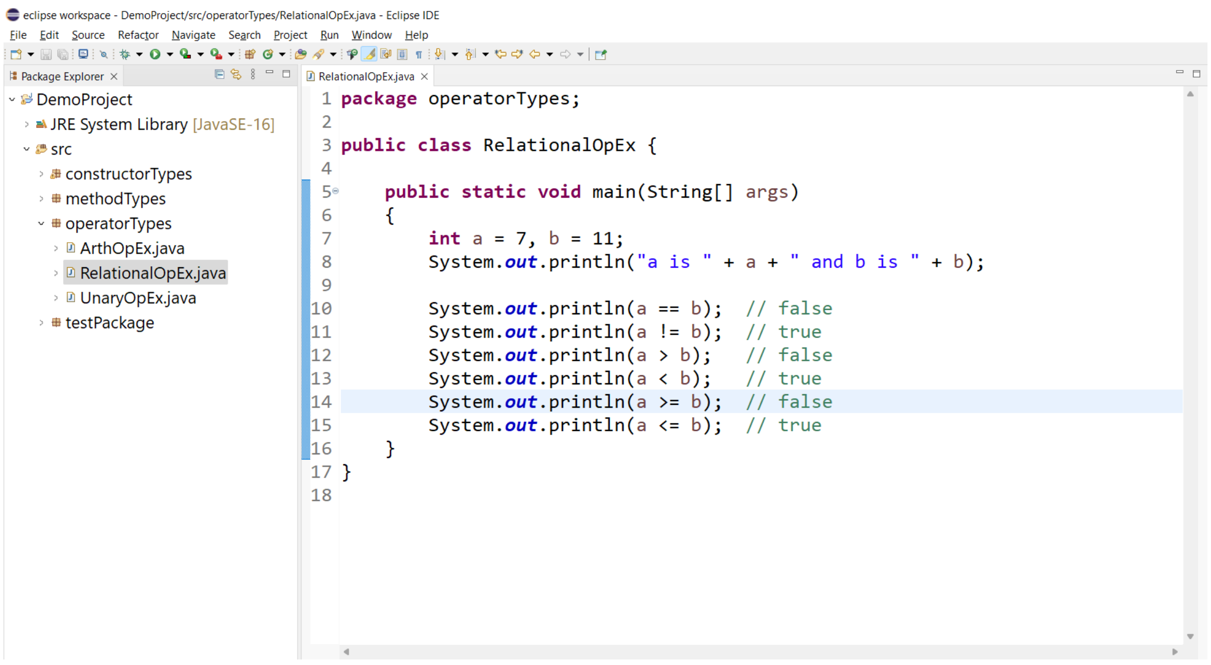
```java
package operatorTypes;

public class UnaryOpEx {

    public static void main(String[] args)
    {
        int a = 12, b = 12;
        int result1, result2, result3, result4;

        System.out.println("Value of a: " + a);
        System.out.println("Value of b: " + b);

        result1 = a++;
        System.out.println("After increment: " + result1);
        result2 = ++a;
        System.out.println("After increment: " + result2);

        result3 = b--;
        System.out.println("After decrement: " + result3);
        result4 = --b;
        System.out.println("After increment: " + result4);
    }
}
```

Arithmetic Operator


eclipse workspace - DemoProject/src/operatorTypes/ArthOpEx.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
ArthOpEx.java
DemoProject
JRE System Library [JavaSE-16]
src
constructorTypes
methodTypes
operatorTypes
ArthOpEx.java
UnaryOpEx.java
testPackage

```java
package operatorTypes;

public class ArthOpEx {

    public static void main(String[] args)
    {
        int a = 12, b = 5;

        // addition operator
        System.out.println("a + b = " + (a + b));

        // subtraction operator
        System.out.println("a - b = " + (a - b));

        // multiplication operator
        System.out.println("a * b = " + (a * b));

        // division operator
        System.out.println("a / b = " + (a / b));

        // modulo operator
        System.out.println("a % b = " + (a % b));
    }
}
```

Relational Operator


eclipse workspace - DemoProject/src/operatorTypes/RelationalOpEx.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
DemoProject
JRE System Library [JavaSE-16]
src
constructorTypes
methodTypes
operatorTypes
ArthOpEx.java
RelationalOpEx.java
UnaryOpEx.java
testPackage
RelationalOpEx.java

```java
package operatorTypes;

public class RelationalOpEx {

    public static void main(String[] args)
    {
        int a = 7, b = 11;
        System.out.println("a is " + a + " and b is " + b);

        System.out.println(a == b);  // false
        System.out.println(a != b);  // true
        System.out.println(a > b);   // false
        System.out.println(a < b);   // true
        System.out.println(a >= b);  // false
        System.out.println(a <= b);  // true
    }
}
```

Logical Operator

Bitwise Operator

Shift Operator

Ternary Operator

Assignment Operator

Java Control Statements | Control Flow in Java

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements.

Java provides three types of control flow statements.

1. Decision Making statements
    - if statements
    - switch statement
2. Loop statements
    - for loop
    - while loop
    - do while loop
    - for-each loop
3. Jump statements
    - break statement
    - continue statement

Decision-Making statements:

As the name suggests, decision-making statements decide which statement to execute and when. Decision-making statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided. There are two types of decision-making statements in Java, i.e., If statement and switch statement.

===

1) If Statement:

The "if" statement is used to evaluate a condition. The control of the program is depending upon the specific condition. The condition of If statement gives a Boolean value, either true or false.

There are four types of if-statements given below.

1. Simple if statement
2. if-else statement

3. if-else-if ladder
4. Nested if-statement

==

1) Simple if statement:

It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.

Syntax:-

```
if(condition)
{
    statement 1; //executes when condition is true
}
```

Example.

```
public class Student{
     public static void main(String[] args) {
          int x = 10;
          int y = 12;
          if(x+y > 20)
          {
               System.out.println("x + y is greater than 20");
          }
 }
}
```

Output:

```
x + y is greater than 20
```

==

2) if-else statement

The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.

Syntax:-

```
        if(condition)
        {
                statement 1; //executes when condition is true
        }
        else
        {
                statement 2; //executes when condition is false
        }
```

Example.

```
    public class Student {
        public static void main(String[] args) {
            int x = 10;
            int y = 12;
            if(x+y < 10)
            {
                    System.out.println("x + y is less than10");
             }
            else
            {
                    System.out.println("x + y is greater than 20");
            }
        }
    }
```

Output:

```
        x + y is greater than 20
```

==

3) if-else-if ladder:

The if-else-if statement contains the if-statement followed by multiple else-if statements. In other words, we can say that it is the chain of if-else statements that create a decision tree where the program may enter in the block of code where the condition is true. We can also define an else statement at the end of the chain.

Syntax:-

Syntax:

```
if(condition 1) {
statement 1; //executes when condition 1 is true
}
else if(condition 2) {
statement 2; //executes when condition 2 is true
}
else {
statement 2; //executes when all the conditions are false
}
```

Example

```
public class Student {
    public static void main(String[] args) {
        String city = "Delhi";
        if(city == "Meerut")
        {
            System.out.println("city is meerut");
        }
        else if (city == "Noida")
        {
            System.out.println("city is noida");
```

```
                }
                else if(city == "Agra")
                {
                        System.out.println("city is agra");
                }
                else
               {
                       System.out.println(city);
                 }
              }
       }

Output:  Delhi
```

==

4. Nested if-statement

In nested if-statements, the if statement can contain a **if** or **if-else** statement inside another if or else-if statement.

Syntax:

```
          if(condition 1)
          {
                 statement 1; //executes when condition 1 is true
              if(condition 2) {
                       statement 2; //executes when condition 2 is true
          }
           Else
          {
                 statement 2; //executes when condition 2 is false
          }
          }
```

Example

```
       public class Student {
             public static void main(String[] args) {
                    String address = "Delhi, India";
                if(address.endsWith("India"))
               {
```

```
 if(address.contains("Meerut"))
 {
        System.out.println("Your city is Meerut");
 }
else if(address.contains("Noida"))
{
        System.out.println("Your city is Noida");
}
else
{
       System.out.println(address.split(",")[0]);
}
}
else
{
       System.out.println("You are not living in India");
}
}
}
```

Output: Delhi

===

Switch Statement:

In Java, Switch Statements are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched.

The switch statement is easier to use instead of if-else-if statements. It also enhances the readability of the program.

Points to be noted about switch statement:

- The case variables can be int, short, byte, char, or enumeration. String type is also supported since version 7 of Java
- Cases cannot be duplicate
- Default statement is executed when any of the case doesn't match the value of expression. It is optional.

- Break statement terminates the switch block when the condition is satisfied. It is optional, if not used, next case is executed.
- While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value.

Syntax:

```
switch (expression)
{
    case value1:
        statement1;
        break;
        .
        .
        .
    case valueN:
        statementN;
        break;
    default:
        default statement;
}
```

Example

```
public class Student implements Cloneable {
        public static void main(String[] args) {
            int num = 2;
            switch (num)
            {
                case 0:System.out.println("number is 0");
                break;
                case 1:System.out.println("number is 1");
                break;
               default:System.out.println(num);
            }
        }
    }
```

Output: 2

While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value. The switch permits only int, string, and Enum type variables to be used.

Program Conducted in Class

==

1. If Statement

```
package ControlStatement;

public class IfStatement {

	public static void main(String args [])
	{
		int a=100;
		int b=50;
		//if statement
		if(a>b)    //false   //true
		{
			System.out.println("a is greater than b");
		}

		if (a<b)
		{
			System.out.println("b is greater than a");
		}
	}
}
```

Output :- a is greater than b

==

2. Nested If Statement

```
package ControlStatement;

public class NestedIfEx {

	public static void main(String[] args)
	{
		int a=10;
		int b=20;

		if(a>b)    //true //false
		{
			if(a==0) //false
			{
				System.out.println("a is less than b but equal to 0");
			}

			System.out.println("a is less than b but not equal to 0");
```

```
				}

				System.out.println("b is greater than a ");
		}

}
```
Output :- b is greater than a

==

3. **If Else Statement**

```
package ControlStatement;

public class IfElseStatement {

		public static void main(String args [])
		{
				int x=10;
				int y=8;

				//if else statment
				if(x>y)
				{
						System.out.println("x is greater than y");
				}
				else
				{
						System.out.println("x is less than y");
				}
		}

}
```
Output :- x is greater than y

==

4. **Nested If Else Statement**

```
package ControlStatement;

public class IfElseIfNested {

		public static void main(String[] args) {


				int x=100;
				int y=100;  //20
```

```
			if(x>y)   //true  //
			{
				if(x==y)  //false
				{
					System.out.println("x and y are same");
				}
				else
				{
					System.out.println("x is greater than y ");
				}
			}
			else
			{
				System.out.println("y is greater than x ");
			}

		}

}
```

Output :- y is greater than x

===

5. Even and odd number

```
package ControlStatement;

public class EvenOddEx {

	public static void main(String args [])
	{
		int a=1567;   //Even

		if (a%2==0)    //0- even   1-Odd
		{
			System.out.println("a is my even number");
		}
		else
		{
			System.out.println("a is my odd number");
		}

	}

}
```

Output :- a is my odd number

===

6. If Else Ladder

```
package ControlStatement;

public class IfElseLadder {

	public static void main(String args [])
	{
		int seema=20;

		if(seema>80)
		{
			System.out.println("Grade is A");
		}
		else if((65<=seema)&&(seema<79))
		{
			System.out.println("Grade is B");
		}
		else if((50<=seema)&&(seema<64))
		{
			System.out.println("Grade is C");
		}
		else
		{
			System.out.println("Grade is D");
		}
	}
}
```

Output:- Grade is D

===

7. Switch Statement

```
package ControlStatement;

public class SwitchStatement {

	public static void main(String[] args) {

		int number=42;

		switch(number)  //20 29
		{
			case 29:System.out.println("My number is 29");
			break;
```

```
                              case 42:System.out.println("My number is 42");
                              break;
                              case 44:System.out.println("My number is 44");
                              break;

                              default:System.out.println("My number is not from 29,42,44");
                              break;
                    }
          }
}
```
Output :- My number is 42

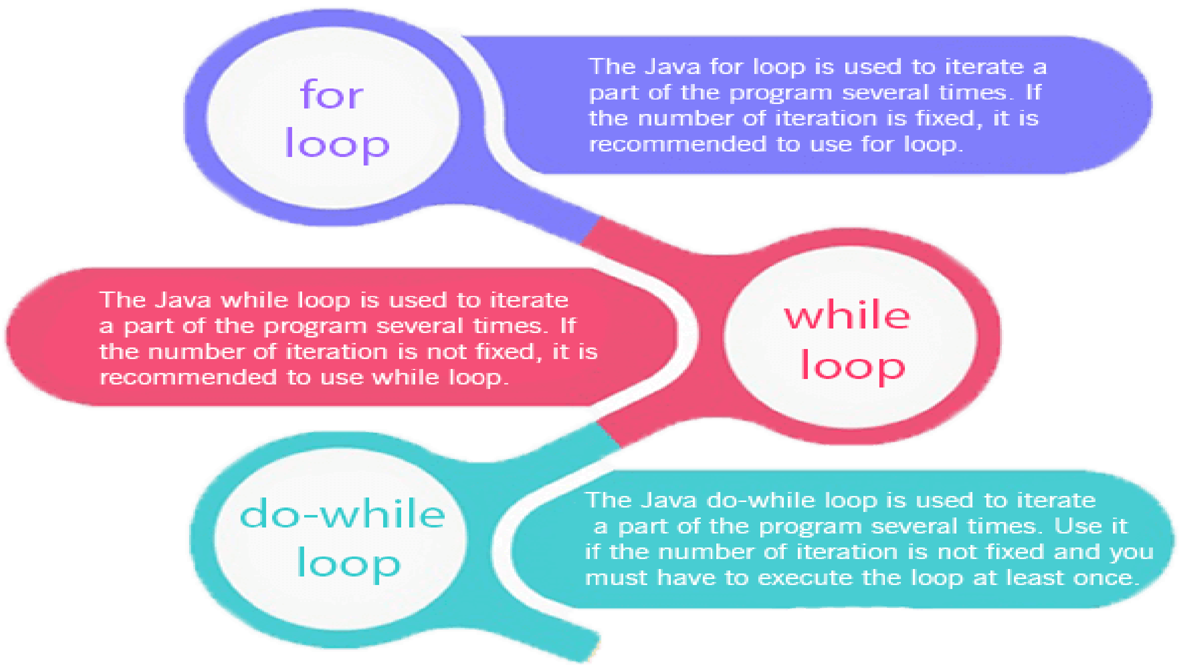
===

Loops in Java

Loop statements are used to execute the set of instructions in a repeated order. The set of instructions execute depends upon a particular condition and In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true.

In Java, we have three types of loops that execute similarly.

1. for loop
2. while loop
3. do-while loop


for loop
The Java for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.
The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.
while loop
do-while loop
The Java do-while loop is used to iterate a part of the program several times. Use it if the number of iteration is not fixed and you must have to execute the loop at least once.

Java for loop

The Java *for loop* is used to iterate a part of the program several times. If the number of iterations is **fixed**, it is recommended to use for loop.

It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code. We use the for loop only when we exactly know the number of times, we want to execute the block of code.

Syntax:

```
for (initialization, condition, increment/decrement)
{
        statements;
}
```

1. **Initialization**: It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable.
2. **Condition**: It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false.
3. **Increment/Decrement**: It increments or decrements the variable value.
4. **Statements**: The statement of the loop is executed each time until the second condition is false.

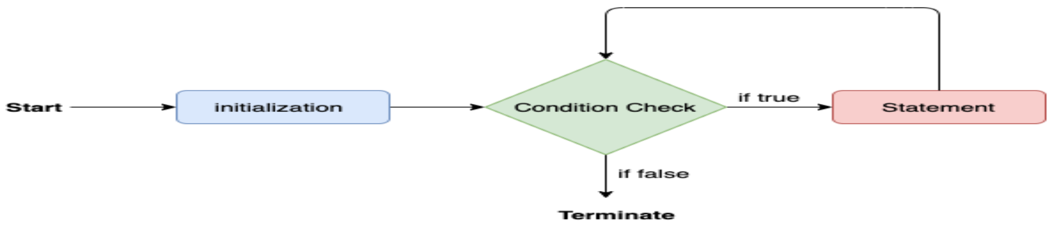
Q. Print 1 to 10

```
public class ForExample {
    public static void main(String[] args) {
        //Code of Java for loop
        for(int i=1;i<=10;i++)
        {
                System.out.println(i);
        }
    }
}
```

Output – 1

```
....
10
```

Flowchart:



Java Nested for Loop

If we have a for loop inside the another loop, it is known as nested for loop. The inner loop executes completely whenever outer loop executes.

Syntax:

```
for (initialization, condition, increment/decrement)
{
        for (initialization, condition, increment/decrement)
        {
                statements;
        }

}
```

Example: NestedForExample.java

```java
public class NestedForExample {
    public static void main(String[] args) {
        //loop of i
        for(int i=1;i<=3;i++)
        {
              //loop of j
              for(int j=1;j<=3;j++)
              {
                   System.out.println(i+" "+j);
               }//end of i
         }//end of j
     }
}
```

Output:

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

Java while loop

The while loop is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop.

1. The Java while loop is used to iterate a part of the programs repeatedly until the specified Boolean condition is true. As soon as the Boolean condition becomes false, the loop automatically stops.
2. The initialization and increment/decrement doesn't take place inside the loop statement in while loop.
3. It is also known as the entry-controlled loop since the condition is checked at the start of the loop.

Syntax:

```
1.          while(condition)
            {
                statements;
                Increment / decrement statement;
            }
```

Example:-

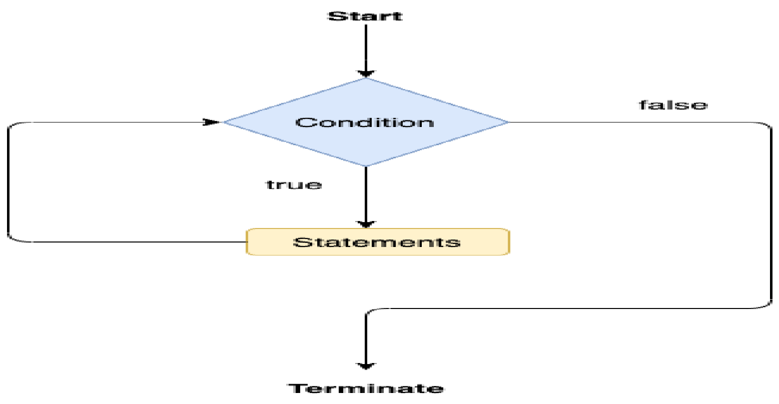
we print integer values from 1 to 10.

```
public class WhileExample {
    public static void main(String[] args) {
        int i=1;
        while(i<=10)
        {
            System.out.println(i);
            i++;
        }
    }
}
```

Output – 1

```
.
.
10
```

Flowchart



Java do-while loop

The Java *do-while loop* is used to iterate a part of the program repeatedly, until the specified condition is true. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use a do-while loop.

1. Java do-while loop is called an **exit control loop**.
2. Therefore, unlike while loop and for loop, the do-while check the condition at the end of loop body. The Java *do-while loop* is executed at least once because condition is checked after loop body.

Syntax:

```
do
{
        //statements;
} while (condition);
```

Example:-

we print integer values from 1 to 10.

```
public class DoWhileExample {
        public static void main(String[] args) {
                int i=1;
                do
                {
                        System.out.println(i);
```

```
                    i++;
               }while(i<=10);
          }
     }
```

Output – 1

```
.
.
10
```

Java for Loop vs while Loop vs do-while Loop

Comparison	for loop	while loop	do-while loop
Introduction	for loop is a control flow statement that iterates a part of the programs multiple times.	while loop is a control flow statement that executes a part of the programs repeatedly on the basis of given boolean condition.	do while loop is a control flow statement that executes a part of the programs at least once and the further execution depends upon the given boolean condition.
When to use	If the number of iteration is fixed, it is recommended to use for loop.	If the number of iteration is not fixed, it is recommended to use while loop.	If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use the do-while loop.

Syntax	for(init;condition;incr/decr){
	// code to be executed
	}
		while(condition){
		//code to be executed
		}
			do{
			//code to be executed
			}while(condition);

Syntax for infinitive loop	for(;;){
	//code to be executed
	}
		while(true){
		//code to be executed
		}
			do{
			//code to be executed
			}while(true);

Java User Input (Scanner)

==

Scanner Class

- The Scanner class is mainly used to get the user input, and it belongs to the *java.util* package.
- In order to use the Scanner class, you can create an object of the class and use any of the Scanner class methods.

Input Types

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
nextLine().chatAt(i)	Reads a char value from the user

Object creation

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

Examples:-

```
import java.util.Scanner;  // Import the Scanner class
 public class Example1 {
                public static void main(String[] args) {
                          // Create a Scanner object
                          Scanner obj = new Scanner(System.in);
                          System.out.println("Enter username");
                          // Read user input
                          String userName =obj.nextLine();
                          // Output user input
                          System.out.println("Username is: " + userName);
                  }
  }
```

Output :-

```
Enter username
My name is Khan
Username is: My name is Khan
```

```
import java.util.Scanner;
public class Example2 {
                   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();
                            // Output input by user
                            System.out.println("Name: " + name);
                            System.out.println("Age: " + age);
                            System.out.println("Salary: " + salary);
                       }
 }
```

Output :-

```
Enter name, age and salary:
Mayur
25
100000
Name: Mayur
Age: 25
Salary: 100000
```

Close Scanner

- Java Scanner class uses the “Close ()” method to close the Scanner.
- Input.close();

Do you need to close a Scanner class?

- It is better but not mandatory to close the Scanner class as if it is not closed, the underlying Readable interface of the Scanner class does the job for you. The compiler might flash some warning though if it is not closed.
- It is a good programming practice to explicitly close the Scanner using the Close () method once you are done using it.

===

Java Math class

==

Math class

Java Math class provides several methods to work on math calculations like min(), max(), avg(), sin(), cos(), tan(), round(), ceil(), floor(), abs() etc.

Method	**Description**
Math.abs()	It will return the Absolute value of the given value.
Math.max()	It returns the Largest of two values.
Math.min()	It is used to return the Smallest of two values.
Math.round()	It is used to round of the decimal numbers to the nearest value.
Math.sqrt()	It is used to return the square root of a number.
Math.cbrt()	It is used to return the cube root of a number.
Math.pow()	It returns the value of first argument raised to the power to second argument.

Other than this the **java.lang.Math** class contains various such as the logarithm, cube root, and trigonometric functions etc.

```
public class JavaMathExample1
  {
      public static void main(String[] args)
      {
          double x = 28;
          double y = 4;

          // return the maximum of two numbers
          System.out.println("Maximum number of x and y is: " +Math.max(x, y));

          // return the square root of y
          System.out.println("Square root of y is: " + Math.sqrt(y));
```

```
        //returns 28 power of 4 i.e. 28*28*28*28
        System.out.println("Power of x and y is: " + Math.pow(x, y));

        // return the logarithm of given value
        System.out.println("Logarithm of x is: " + Math.log(x));
        System.out.println("Logarithm of y is: " + Math.log(y));

        // return the logarithm of given value when base is 10
        System.out.println("log10 of x is: " + Math.log10(x));
        System.out.println("log10 of y is: " + Math.log10(y));

        // return the log of x + 1
        System.out.println("log1p of x is: " +Math.log1p(x));

        // return a power of 2
        System.out.println("exp of a is: " +Math.exp(x));

        // return (a power of 2)-1
        System.out.println("expm1 of a is: " +Math.expm1(x));
    }
  }
```

Output:-

```
Maximum number of x and y is: 28.0
Square root of y is: 2.0
Power of x and y is: 614656.0
Logarithm of x is: 3.332204510175204
Logarithm of y is: 1.3862943611198906
log10 of x is: 1.4471580313422192
log10 of y is: 0.6020599913279624
log1p of x is: 3.367295829986474
exp of a is: 1.446257064291475E12
expm1 of a is: 1.446257064290475E12
```