Java Introduction:

When was Java Introduced?

- **AUTHOR** ----------------------- JAMES GOSLING
- **VENDOR** ---------------------- SUN MICRO SYSYTEM(ORACLE)
- **PROJECT NAME** ------------- GREEN TEAM
- **TYPE** --------------------------- OPEN SOURCE
- **INITIAL NAME** --------------- OAK
- **PRESENT NAME** ------------- JAVA
- **EXT** ----------------------------- .java,.class,.jar
- **PRESENT VERSION** ---------- JAVA18
- **OPERATING SYSTEM** ------- ANY OPERATING SYSTEM
- **BASIS** --------------------------- C++
- **PRINCIPLE** --------------------- WORA (write once run anywhere)

- In 1991 Sun microsystem had a requirement and they want to prepare new programming language.
- By using the new programming, they want to prepare software for cable tv, switch boxes, remote control.
- As per the requirement sun microsystem gather 30 members from their company as Team. This team led by **James Gosling** and name given to the project is **GREEN.**
- **Key persons of java programming language development:**
- James Gosling, Patrick Naughton and mike Sheridan were the Key Persons in the java programming language development.
- **Three feature that java development team added in programming before introducing this:**
  - Simple Programming language which means java is easy to learn, and its syntax is simple, clean and easy to understand.
  - Tightly coded language.
  - Architectural neutral programming language which means that gives less cost to the product.
  - For developing these features, they took almost 2 years.

Why is it called Java?

- Java is one of the famous places in Indonesia that place famous for Coffee. The name of coffee powder is java.
- This java coffee powder is used to make coffee for team at sun microsystem café.
- James Gosling and his team everyday used to take this coffee.
- From this they decided to give the name to new programming language as JAVA.

When was the first version of the Java programming language released?

- On dated 23rd January 1996 java introduced first version as **jdk 1.0**
- In 2010 java is managed by Oracle Corporation follows a pattern every even version is released in March month and every odd version is released in September month.
- **Latest Version of java is JAVA SE18.**

Why is Java referred to as WORA (Write Once, Run Anywhere)?

- In traditional programming language like C, C++ when program were compiled, they used to be convert into the code understood by particular underlying hardware.
- If we try the same code at another machine with different hardware will causes an error so need to recompile code in that case.
- But in JAVA, the program is not converted into code directly understood by hardware, rather it is converted into the BYTE code (.class file) So which is interpreted by JVM so once compiled it generates byte code file which can run on any operating system with any modification in the code.

Types of java classification:

- **Java SE (Java Standard Edition):**
  - It is mainly used for programming of standalone/desktop applications.
  - 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.
  - It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

- **Java EE (Java Enterprise Edition):**
  - It is an enterprise platform that is mainly used to develop web and enterprise applications.

- It is built on top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, JPA, etc.
- Mostly interact with server. Example, banking, e-commerce, telecom application.

- **Java ME (Java Micro Edition):**
  - It is a micro platform that is dedicated to embedded system development such as mobile applications.

Types of Java Applications:

- **Standalone Application:**
  - 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, Paint,
  - Microsoft office etc.
  - AWT and Swing are used in Java for creating standalone applications.
  - Standalone application do not required internet.

- **Web Application:**
  - An application that runs on the server side and creates a dynamic page is called a web application.
  - Examples of web application are WhatsApp, Gmail, Facebook, amazon etc.
  - Servlet, JSP, Struts, spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.
  - All web application require internet.

- **Enterprise Application:**
  - An application that is distributed in nature, such as banking applications, etc. is called an enterprise application.
  - It has advantages like high-level security, load balancing. In Java, EJB is used for creating enterprise applications.
  - Examples of Enterprice applications are Google Pay, PhonePe, and Billing System.

- **Mobile Application:**
  - An application which is created for mobile devices is called a mobile application.
  - Currently, Android and Java ME are used for creating mobile applications.
  - Examples of mobile application are Games like (PubG, Candy crush, Talking Tom etc.,), Share it, Instagram.

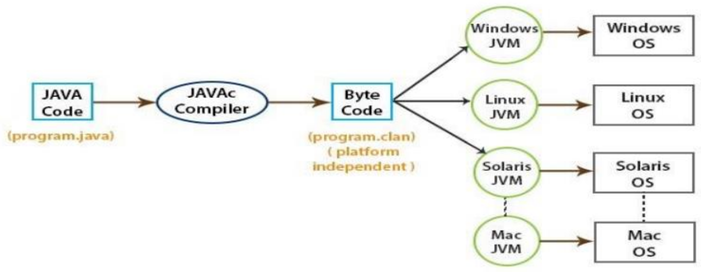
Different features of java:

- **Simple:**
  - Java is easy to learn, its syntax is simple and easy to understand.
  - In java there is no need to remove unreferenced object because it have automatic garbage collector.

- **Object-Oriented:**
  - Java is object-oriented programming language, everything in java is an object.
  - Object-oriented development includes concepts of objects and classes and many more.
  - This enables developers to have a wide variety of options for designing their software.

- **Portable:**
  - Java is portable because its facility to carry byte code to any platform.

- **Platform independent:**
  - Being platform-independent means, a program compiled on one machine can be executed on any machine in the world without any change.
  - Java achieves platform independence by using the concept of the BYTE code.
  - This is where the "Write once, run anywhere" (WORA) slogan for Java comes in, which means that we can develop applications on one environment (OS) and run on any other environment without doing any modification in the code.



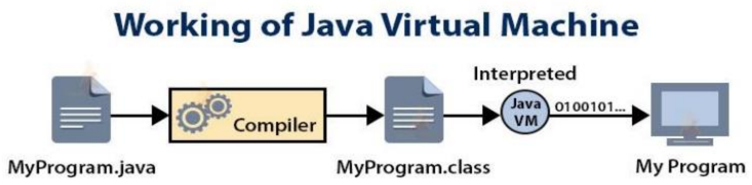
  -

- **Secured:**
  - Java is the best known for Security.
  - By using java, we can develop virus free system or software**.**

- **Architectural neutral:**
  - Architectural neutral means that the program written on one platform or OS is independent of other platforms or environments and can run on any other operating system without recompiling them.

- For Example, In C programming, the datatype occupies 2byte for 32-bit architecture and 4byte for 64-bit architecture but in java occupies 4byte for both 32- and 64-bit architecture.

- **Interpreted:**
  - Usually, a computer language can be either compiled or interpreted.
  - Java integrates the power of Compiled Languages with the flexibility of interpreted Languages.
  - Java compiler (javac) compiles the java source code into the bytecode.
  - Java Virtual Machine (JVM) then executes this bytecode which is executable on many operating systems and is portable.

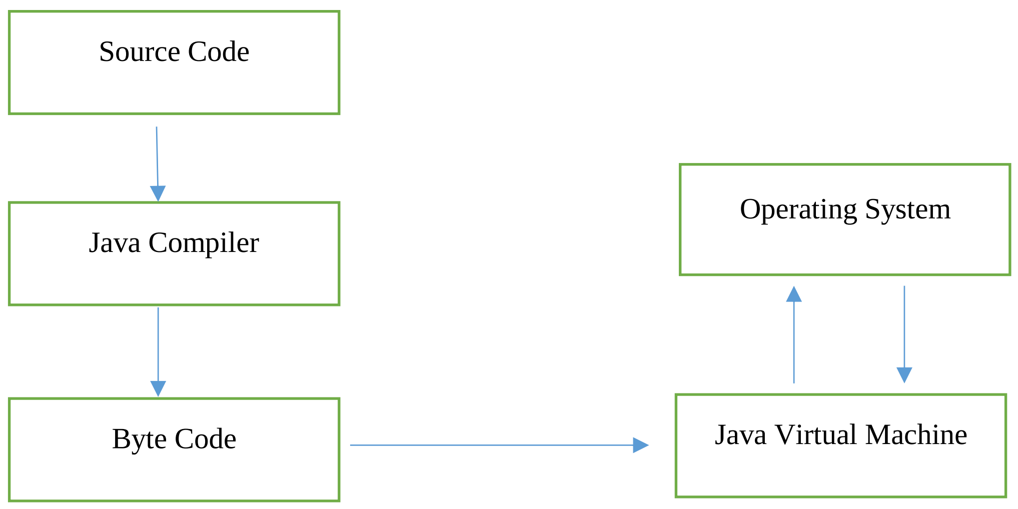


- **High Performance:**
  - Java provides high performance with the use of “JIT – Just in Time compiler”, in which the compiler compiles the code on-demand basis, that is, it compiles only that method which is being called.
  - This saves time and makes it more efficient.

- **Distributed:**
  - In Java, we can split a program into many parts and store these parts on different computers. A Java programmer sitting on a machine can access another program running on the other machine.
  - This feature in Java gives the advantage of distributed programming, which is very helpful when we develop large projects.

- **Dynamic:**
  - Java is dynamic language. It supports dynamic loading of classes.
  - It means classes are loaded on demand.

Architecture of Java:



- **Step 1:**
  - Whatever program we write it is called as **source code.**
  - Source code should always save as extension **".java".**

- **Step-2: (Compilation)**
  - The process of converting our program into system understandable form (byte code) is purpose of compiling a program.
  - to compile a program, go to cmd prompt and enter command as

javac filename **or className.java**

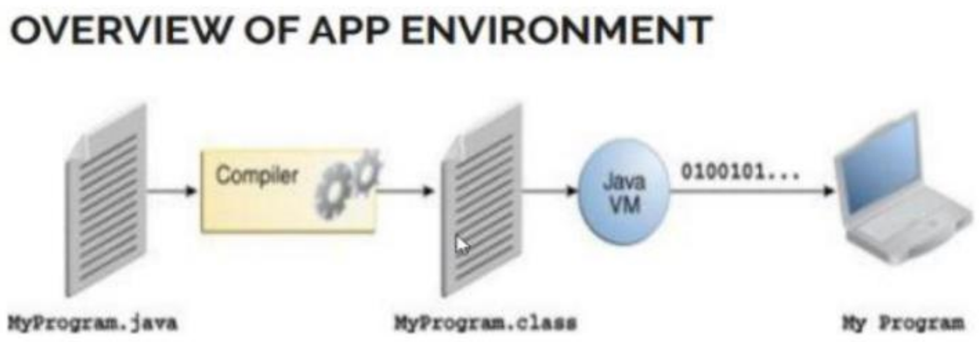
Ex: javac Hello.java

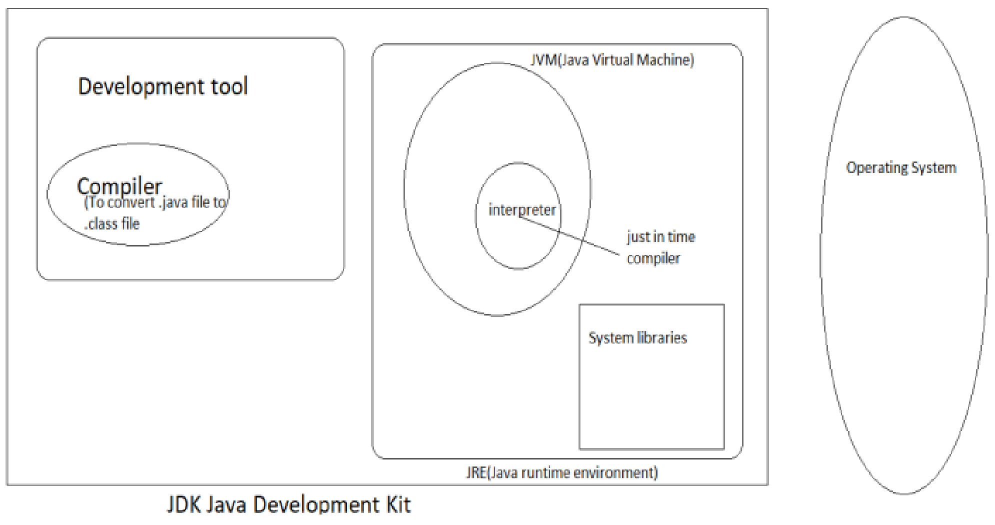
  - Compilation is done at once.
  - During compilation, compiler will check syntax errors like [ ], ;,(), { }, :, spellings and case sensitivity.
  - If anything is wrong, we will get compile time error.
  - If nothing is wrong there is one class file get generated (byte code file) with same as **.class extension**

- **Step-3: (Execution)**
  - JVM -java virtual machine is responsible for execution of every java program.
  - JVM's can identify by ------> Public static void main (String [ ] args)
  - It is like one software or one program.
  - Execution will happen in line-by-line manner.
  - During execution JVM will find logical error of program.
  - for executing program go to command prompt and enter command as

- java **filename or className.java**
- Ex: java Hello.java

- Once we enter this command JVM will go to class file and take first line and give to operating system for execution, once OS responds that I understood that line and it sends second line and it continues till last line like this whole code of class file gets executed.





JDK = JRE + Development tool

JRE = JVM + Library Classes.

JDK-Java Development Kit

JRE-Java Runtime Environment.

JVM-Java Virtual Machine.

- JDK-A kit which provides the environment to develop and execute (run) java program.
- JDK- JRE-provide runtime environment to run the program while development tool is used to develop java application.
- JRE-It provides environment to only RUN
- JVM-It is responsible for executing the java program line by line. It is also known as Interpreter.
- The main feature of java is WORA.WORA stands for write once run anywhere. The features states that we can write our code once and use it anywhere or any operating system.
- Our java program can run any of the platforms only because of the java virtual machine.
- It is a java platform component that gives us an environment to execute java programs.
- JVM’s main task is to convert byte code into machine code.
- JVM, first of all, loads the code into memory and verifies it. After that, it executes the code and provides a runtime environment.
- Java has two process, compilation and interpretation.
- The java compiler converts the code present in the java into byte codes.
- The java virtual machine converts the byte codes into machine code which the machines execute directly.
- The JRE build a runtime environment where we can execute the java programs. It takes the java code and combines the same with required libraries.
- Interpreter translates just one statement of program at a time into machine code.
- Compiler scans the entire program and translate the whole program of it into machine code at once.
- An interpreter takes very less time to analyze the source code. A compiler takes more time.

Difference between JDK , JRE and JVM :

JDK	**JRE**	**JVM**
JDK is a software development kit.	JRE is a software bundle that allows java program to run.	JVM is an environment for executing bytecode.
JDK-Java Development Kit.	JRE-Java Runtime Environment.	JVM-Java Virtual Machine.
JDK is a platform dependent.	JRE is a platform dependent.	JVM is a platform independent.
JDK contains tools for developing, debugging etc.	JRE contains class libraries and other supporting files.	Software development tools are not included in JVM.
JDK comes with the installer.	JRE only contains the environment to execute source code.	JVM bundled in both software JDK and JRE.

Every program has 3 main parts:

- **CLASS DECLARATION:**

  Ex: `public class Demo`

  It consists of 3 things:

  1. Access Modifier:
  - It indicates that program is accessible to other users or not
  - There are 4 access modifiers in java **public, private, protected and default**
  - In above section class is public so it is freely accessible
  - All access modifiers will be in lower case.

  2. `class`:
  - Class is a keyword (reserve word or predefine word) in java.
  - Every program must start with class keyword
  - All keyword must start with smaller case so class-c is small.

  3**. Class Name:**
  - Every class has some name i.e., class name or program name or file name.
  - Class name for standard should start with Capital letter.
  - Java file name and class name must be same for remembering purpose.
  - Class name can only be combination of A-Z,a-z,0-9,$,_ etc.

  Note: {//} --------->scope of class.

- **DEFINING MAIN METHOD:**

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

        //Business Logic
}
```

It consists of 5 parts:

1. **Access Modifier:**
2. **Non Access Modifier:**

   static--------------> can be used without object creation
   non static---------> needs to be used with object creation
3. **return type:** void indicated no specific return type
4. **method name:**

   -if any word contains ( ) ----> we can identified it as a method
   Ex: run ( ), display ( ) etc.

5. **command line arguments:**

`String[] args`
String - S is capital

-This statement can be written in 3 ways

String args [ ]
String [ ] args
String [ ] args

Note: In syntax of main method only String-S is capital remaining all words starting letters are small.

▪ PRINTING STATEMENT:

```
System.out.println("This is my sample program code");
```

System------> it is a pre define class [class System]
--------------> It is a dot operator, any word after it can be reference variable/object or method
out -----------> it is an object (predefine)
`println` ( ) ----> it is a method (predefine)

-In simple println ( ) is accessed through out object but out object is present in system class.
-System is a class which contains out object and out object is referring to println ( )
-whatever we gave in double quotes that message will be printed as it is.

Basis terms and Explanation:

If we want to write a program, we need to follow below steps:

- Create a project
- Create a package
- Create a class

▪ Project:

- Project is a collection of multiple source folder, multiple package, multiple classes, JRE system library, .jar files, predefined keywords, methods, syntax and many more.
- We can create multiple packages and classes inside the project.
- How to create a project
- Click on file menu>>New>>java project.
- The name of java project should start with upper case letter.

- **Src (Source Folder):**
  - It is a branch of project .It is collection of multiple packages and class.
  - We can create multiple packages and classes inside the src folder.

- **package:**
  - package is a collection of multiple classes.
  - **Two types of package:**
    - **default package:** if we are create number of classes without generating package. It shows under the default package.

    - **user define package:** If we create package then it will be user define package.

  - package name should start with lower case letter.
  - **Syntax: package name;**
  - **How to create Package:**
  - Right click on Src folder >> New>> java package>>Insert name>>Finish.

- **Class:**
- Collections of objects is called a class.
- A class is collection of member function (methods), variable, data types, printing statements, scanning statements, object, constructor etc.
- Class is the template or blueprint from which object is created and Object is the instance of class.
- Class can only be accessed from outside via its instance.
- A class name should start with Upper Case letters.
- **How to create Class:**
- Right click on package>>New>>New Class>Insert Name>>Finish.
- **A class in java can contains:**

  Variables
  Methods
  Constructors
  Blocks
  Nested Class: Class within other class called as Nested class in java.

- **JRE System Library:**
- It is a library which contains predefine classes, modules, keywords, syntax these are support to write a code.
- Any java program written in eclipse IDE can access predefined java function because of JRE system library. This JRE system library contains all files with **.jar**
- Extension.

Object in java:

- An object in Java is the real-world entity which has its own state and behavior.
- Combination of state i.e. variables and behavior (which can be achieved through methods) is called **an object.**
- A Java program can have as many objects as required.
- **An object in Java typically contain following things:**
- **State:** This is represented by the attributes and properties of an object.
- **Behavior:** This is defined by the methods of an object, In other words, its functionality.
- **For Example**, Mobile is an object. Its name is Samsung; color is black known as its state. It is used for calling, texting message so its behavior.
- When we create an object (instance) of class then space is reserved in heap memory.

Important note:

- **Project Name:** The project name should always start with uppercase letter.
- **Package name:** The package name should always start with lowercase letter.
- **Class Name:** The class name should always start with Uppercase Letter.
- **Method Name:** The method name should always start with Lowercase Letter.
- **Comments // :**These lines are used for comments. That are not considered during execution of program.
- **Signature Bracket ( ):** These bracket are used for argument.
- **+ Sign:** This + Sign is use for join string or statements.
- **Body or brake { } :** These bracket are called as body or brake. These are used to define boundary of particular class, methods.

Syntax Meaning: System.out.println( );

- **System:** It is name of java utility class.
- **out:** It is object which belongs to system class.
- **println:** It is used to call predefine method. Which is used to send any string to console on new line.
- **print:** It is used to call predefine method. Which is used to send any string to console on same line.
- **;** This symbol is used to terminate the particular line.

Methods (Member Function) in java

- Methods in java is collection of instructions that perform specific task.
- Without methods we are not able to write a program.
- It has memory location in JVM to store data and information. Methods is always declared inside class body.

- **Two types of methods:**
  - Business Method:
  - Regular Methods:
    - Static Method
    - Non-Static Method

- **Business Method:**
  - Java main () method
  - It is starting point for JVM to start execution of a java program.
  - Without main method JVM will not execute the program.
  - **Syntax of main () Method:**
  - public static void main (String [] args) { }
  - **public :** It is an access modifier .we should use public keyword before main () method so JVM can identify the execution point of the program.
  - **static:** we 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 a return type.
  - Void in main () method does not return any value.
  - **main():** It is default method name which is predefine in JVM.
  - **String [] args**: The main () method also accept some data from the user. It accepts the group of strings, which is called String Array.

- **Static Method:**
  - The method which contains static keyword and use to perform specific task.
  - It is static in nature.
  - Static method has fixed memory in JVM**.(Heap Memory)**
  - To call the static method there is no need to create an object.
  - **Syntax of Static method is:**
  - **public static void *staticMethodName*( ){ }**

- **Non-Static Method:**
  - The method which does not contains static keyword and use to perform specific task.
  - It is dynamic in nature.
  - Non-static method has dynamic memory in JVM.**(Stack Memory)**
  - To call the non-static method there is need to create an object.
  - Non-static method is more secure than static method.
  - **Syntax of Non-Static Method:**
  - **public void *nonStaticMethodName* ( ) { }**

Difference between Static and non-static method:

Static method	Non static method
Static methods are class level methods.	Non-Static methods are object level methods.
It can be accessible directly inside the main method (if it is defined in the same class) or by *classname.method* name or by *referencevariable.method* name.	It can be accessible only after creation of object.
Memory allocation is happening at the time of loading a class and de- allocation at the time of unload of class.	Memory allocation is happening once the method gets call and once the execution gets completed it gets de-allocate. Objection creation and destruction.
It is recommended only when we are sure of its usage.	It is recommended when we are not sure to execute all.
Syntax – *public static void nameofmethod.*	Syntax- *public void nameofmethod*

Process of executing a program:

- Test.java
- start jvm
- Jvm create the main thread.
- Locate Test.class file
- Load Test.class file --- static method and variable memory allocation
- execute main method --non static method loading by creating an object
- Unload Test.class file. --static method and variable memory deallocation
- JVM Shutdown.

Variables:

- Variables is piece of memory used to store an information.
- Variable are always assigned with datatypes. To represent type of data the variable holds.
- According to all programming language, we cannot declare information directly so variable are introduced.
- **int a=20;**
- **int** is a data type to represent type it holds integer value.
- **a** is variable that holds information 20.
- **Types of Variables:**
    - **Static Variable:**
    - **Non-Static Variable:**
    - **Local Variable:**

- **Static Variable:**
  - These are class level variable which is defined at class level.
  - Static variable are not declare and initialized inside the method.
  - They can be access throughout the class.
  - Syntax: static datatype variablename = value;
  - Static variable can be access through :
  - **Directly by the name (if it is in same class)**
  - Or by the **Classname.Variablename;**
  - By creating an object But it is not recommended
  - **Classname refvariable = new Classname ();**
  - **refvariable.staticvariablename;**
  - Static variable will be created at the time of class loading and destroy at the time of class unloading Hence the scope of static variable exactly the **.class file.**
  - If we don't assign the value to the variable then it attains default values.
  - **Those default values:**

    ```
    Int =0;
    byte =0;
    short =0;
    long =0;
    float = 0.0;
    double =0.0;
    boolean = false;
    char =' ';
    ```

  - Static variable share memory with every object .

- **Non-Static Variable or Instance Variable:**
  - These are class level variable which get define at class level.
  - They can access through only by object in case of static method.
  - **Syntax Datatype Variable Name = Value ;**   declaration initialization
  - Datatype **Variable Name;**   declaration of variable
  - If we don't assign the value to the variable then it attains default Values.
  - Non static Variable get differs from object to object.
  - Non-Static variable gets memory allocated at the time of object creation.
  - Non-Static variable gets memory de-allocated at the time of object destruction.
  - Scope of Non Static Variable is same as object scope.
  - If we wants to non-static variable in non-static method without creating an object.
  - Non Static variable doesn't share memory with every object .So its value differ from object to object.

- **Local Variable:**
  - The variables which are define within the curly braces or body of method.
  - The scope of that variable would be within the curly braces or body of that method.
  - If the local variable name is as same as non-static variable if we want to access non-static variable so in that case **so we use this keyword.**
  - If the local variable name is as same as static variable if we want to access static variable so in that case for accessing it we use **Classname.variable name;**
  - JVM does not provide any default value to the local variable. If the local variable is not initialized then the program will not be executed it break the code and gives an error.

Difference Between Static ,Non-static & Variables:

Static Variable	Non-Static Variable	Local Variable
They are define at class level.	They are define at class level.	They are define at method level or within the curly braces.
Scope is throughout the class.	Scope is throughout the object.	Scope is upto the boundary of curly braces.
Memory allocation is at time of class loading	Memory allocation is at time of object creation .	Memory allocation is at time of method calling
this keyword is not applicable in static method.For access it use Classname.variablename;	**this** keyword is applicable in non-static method and we are able to acess non static variable to non static area .	Not Applicable.
Value of static variable doesnot change object to object. Here we share the value of static variable with every object.	Value of Nonstatic variable change object to object . Here we share the value of static variable with every object.	Not Applicable.
JVM provide default value if the value has not been initialised	JVM provide default value if the value has not been initialised	JVM doesn’t provide any support for default values. Programmer has to initialise before using it .

Data Types in Java:

- Data types are used to represent type of data or information that we going to use in the programming.
- It is mandatory to declare the data type before variables.
- **Type of Data Types used in java:**
  - **Primitive Data Types:**
    - The Primitive Data types includes boolean, char, byte, short, int, long, float & double.
  - **Non-Primitive Data Types:**
    - The non-primitive data types are classes, interfaces, arrays, string etc.

- **Details of Primitive Data Types:**

Type	**Size**	**Range**	**Default**
boolean	1 bit	Store info. true or false	false
byte	1 byte =8 bits	Minimum value= -128 Maximum value= 127	0
short	2 byte=16 bits	Minimum value= -32768 (-2^15) Maximum value= 32767 (2^15-1)	0
char	2 byte=16 bits	Minimum value= -32768 (-2^15) Maximum value= 32767 (2^15-1)	0
int	4 byte=32 bits	Minimum value= -2147483648 (-2^31) Maximum value= 2147483647 (2^31-1)	0
float	4 byte=32 bits	Minimum value= -2147483648 (-2^31) Maximum value= 2147483647 (2^31-1)	0.0
long	8 byte =64 bits	Minimum value= **- 9,223,372,036,854,775,808** (-2^63) Maximum value= **9,223,372,036,854,775,807** (2^63-1)	0
double	8 byte =64 bits	Minimum value= **- 9,223,372,036,854,775,808** (-2^63) Maximum value= **9,223,372,036,854,775,807** (2^63-1)	0.0

Difference between primitive and non-primitive data type:

Primitive Data type	Non-primitive data type
Primitive data types are predefined in java.	Non-primitive data types are created by the programmer and is not defined by Java.
A primitive data type always has a value.	Whereas non-primitive data types can be null.
A primitive data type starts with a lowercase letter.	Non-primitive data types start with an uppercase letter.
The size of a primitive data type depends on the data type.	Non-primitive data types have all the same size.
Primitive data type are not used to perform certain operation.	Non Primitive types can be used to call methods to perform certain operations.
Primitive data type value range is limited.	Non-Primitive data type value range is unlimited.

Method Calling:

- **Calling of Method in either type of Methods:**
  - **Calling of Static Method in another static method:**
    - We can call direct (By the name of method) in another static method which is provided it in the same class.
    - **Second way** Classname.methodname (if this method is in another class) or by creating an object (Which is not recomended).
  - **Calling of Static Method in another Non-static method:**
    - We can call direct (By the name of method) in another Non-static method which is provided it in the same class.
    - Second way Classname.methodname (if this method is in another class) or by creating an object (Which is not recommended)
  - **Calling of Non-Static Method in another static method:**
    - We can call this method by creating an object of class which contain the particular Non-Static method.
  - **Calling of Non-Static Method in another Non-static method:**
    - By calling the direct name of method in NonStatic method provided it is in the same class. But another Non static method is available in outside of class then we have to create object for that.

- **We can call another class main method in different class inside the main method of particular class.**

- A method which gets exceute by JVM once programmer calls the entity in particular sequence.
- **Main method can be called from other class method like static method.**
- **For Example**

```
Public Class Test {

public static void main ( String [] args)    {

          Test2.main(args);

             }
        }


Public Class Test2 {

public static void main (String [] args)  {

System.out.println{" Test Class is running "}
         }
}
```

- Test Class Gives output As **Test Class is running. Because of Test2 Class Main Method is called inside the Test Class Main Method.**

METHODS WITH SOME INPUT:

- A method can have any number of inputs in the form of arguments.
- Ex: public static void add()//method with zero argument or 0 input.
- public static void add(int i)//method with integer argument.
- When we call any method with argument we have to pass that particular type of values.
- While passing arguments we have to make sure it will be as per sequence define in method declaration.

```
//static method without specific returntype and with arguments
 public static void registration(String name,String email,long contact)
 {
 System.out.println("Name :"+name);
 System.out.println("Email address:"+email);
 System.out.println("Contact details: "+contact);
 }
 }
```

METHOD WITH RETURN TYPE:

- If we want we can define a method with any specific return type.
- For Example:

```
public static int add()
    {
    //perform operation whatever we want//
    return integertypevale;
    }
```

- Whenever we define a method with return type, we are telling that my method is going to return that particular type ofdata (value).
- It is mandatory to write return statement if we have a method with specific return type.

- **return keyword:**
  - It is used inside a method whenever we define a method with specific return type.
  - return keywod is used to take data and exit from method.
  - return keyword must be last stattement in a method.
  - return keyword will not print the data.
  - we cannot write more than one retun statement in a method because after one return statement it exits from the method,
  - so that the next return statement cannot be executed.

- **void :**
  - no specific returntype or method is not writng any specific type of value.

```
//method with returntype
public class IntRegistration
{
 public static long phnofield()
 {
 return 9874561337l;
 }
 public static char genderfield()
 {
 return 'M';
 }
 public static void main(String args[])
 {
System.out.println(phnofield());
System.out.println(genderfield());
 }
}
```

Constructor in JAVA:

- Basically, a constructor is used to initialize data member/variables.To load Non-static method or member function into the object.
- A special method which gets execute at object creation.
- **Rules:**
  - The classname and constructor name must be same.
  - Constructor can accept the arguments.
  - Constructor can't have return type.
  - Class is having multiple constructor
  - Constructor can call another constructor by **using this keyword.**
- **Usage:**
  - A method which executes business logic at the time of object creation.
  - To initialise the data member.

- **Types of constructor:**
  - **Default constructor:**
  - **User defined Constructor:**

  - **Default constructor:**
    - If there is no constructor present in class. In that cases java compiler provides a default constructor that is known as ***default constructor.***
    - It does not have any parameter. Java automatically provides a default constructor that initialize all variables to zero.
    - ***Syntax: Classname ( ) { }***

  - **User defined Constructor:**
    - If the programmer defines the constructor inside the class body. Then it will be user define constructor.
    - **User define constructor having two types:**
    - **Zero argument Constructor:**
      - It is also known Non-argument/Non-parameterized constructor.
      - A constructor with no argument then it known as Zero argument constructor.

- **Parameterized Constructor :**
  - This is also known as argument constructor.
  - A constructor which has specific no. parameters or argument by which we called ***as parameterized constructor.***
  - It is used to initialize the data member of class with distinct value or we can provide the same values also.
  - This constructor is used to allow constructor by providing the different type of argument.
  - We can call another constructor in constructor class of same class by using **this keyword.**
  - We can call another constructor in constructor class of different class by using **super keyword.**

Control Statements:

- Control statements means which control the flow of program.
- A java program executes from top to bottom but if we want to control the order of execution of program based on the logic applied , we used control statements.
- **Various types of Control Statements:**
- **Conditional Statements:**
  - If Statement
  - If else
  - If else-if
  - Nested if
  - Nested if-else
  - Switch
- **Looping Statements:**
  - for loop
  - while loop
  - do while loop
- **Jump Statements:**
  - break
  - continue

Conditional Statements:

- **If Statement:**
  - if statement used to test condition .It will execute if block only if the condition is true.
  - **Syntax:**

```
if (Condition)  {

          Statements/action/logic

     }
```

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

                int a=40;
                int b=20;

                if (a>b) {

                        System.out.println("Condition is true
if block is running");
                }
        }
}
```

Output: Condition is true if block is running

- **If else:**
  - In if-else statement, if the specified condition in the if statement is true then it will execute if block only Otherwise else block will be executed.
  - **Syntax:**

if (Condition) {
Statements/action/logic
}
else {
Statements/action/logic
}

```
Public class IFelseCondition {

        public static void main(String[] args) {

                int a=20;

                if (a<=15)
                {
                        System.out.println("Condition true if block is
running");
                }

                else
                {
                        System.out.println("Condition false else block is
running");
                }
        }
}
```

Output: Condition false else block is running

- **If else-if:**
    - In if else if statement contains if statement is followed by multiple else if statements.
    - In if else if statement, suppose if condition is true, then if statement will be executed & the remaining code will be skipped
    - But no any condition is true then else block will be executed.
    - **Syntax:**

```
if (Condition1)  {
        Statements/action/logic
              }
        if (Condition2)  {
           Statements/action/logic
                         }
                  else   {
             Statements/action/logic
              }
```

```
public class IFelseIf {

	public static void main(String[] args) {

		int x=18;

		if (x<=20)
		{
			System.out.println("First if block is running");
		}

		else if (x>=20 && x<=40)
		{
			System.out.println("Condition true else if block is 
running");
		}
		else
		{
			System.out.println("Condition false else block is 
running");
		}
	}
}
```

Output: `First if block is running`

- **Nested if:**
  - In **nested if** statement, if statement can contain multiple if condition inside it.
  - Here check firstly check outer if block condition if it is true then it will enter the code. Otherwise it will run else block directly.
  - **Syntax:**

```
if (Condition) {
        If (condition) {
                Statements
                        }
                }
     else   {
                Statements.
                }
```

```
package conditionalStatements;

public class Nestedif {


	public static void main(String[] args) {

		System.out.println("Welcome to our blood donation camp");

		int age=28;
		int weight=50;

		if (age>20 && weight<=60)
		{
			System.out.println("Condition is eligible ..");

			if ( age>=18 && weight<=50)
			{
				System.out.println("Eligible to donate blood");
			}
		}

		else
		{
			System.out.println("You are not eligible");
		}
	}
    }
```

Output:

```
Welcome to our blood donation camp
Condition is eligible…..
Eligible to donate blood
```

- **Nested if-else:**
    - In nested if else statement contains if else statement inside it.
    - **Syntax:**

```
        if (Condition)  {
                If (condition) {
                           }
                 else {
                  }
               }
          else  {

        }
```

```
public class NestedIFelse {

	public static void main(String[] args) {

		int x=25;

		if (x<=40)
		{
			System.out.println("First if ");

			if (x<=30)
			{

				System.out.println("2nd if is true");
			}
			else
			{
				System.out.println("2nd else block is running");
			}
		}

		else
		{
			System.out.println("Outer else block is running");
		}
	}
			}
```

Output:

```
First if
2nd if is true
```

- **Switch Case:**
  - In java, switch statement contains multiple block of code called as cases.
  - In this, a single case is executed based on variable is being switched.
  - In switch statement we are going to use only one input.
  - Switch statement works with String, int, char etc.
  - The case must be unique, it should not be duplicate .
  - Each switch statement can have break statement to terminate the code once the condition is satisfied.
  - If the cases are not matched that we write then it will execute default block. So we have to provide default block for if the case is not matched.
  - **Syntax:**

```
switch (variable)
{
Case1: printing statement; break;
Case2: printing statement; break;
default :
        Printing statement   }
```

```
public class Colour12 {

	public static void main(String[] args) {

		String colour ="Grey";

		switch (colour) {

		case "Pink" : System.out.println("This is pink 
colour"); break;
		case "Blue" : System.out.println("This is Blue 
colour"); break;
		case "White" :System.out.println("This is White 
colour"); break;
		case "Black" :System.out.println("This is Black 
colour"); break;

		default :

			System.out.println("No one colour is 
matched");
		}
	}
}
```

Output: `No one colour is matched`

Looping Statements:

- Loop statement are used to execute the set of instructions in are 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 conditions evaluate to true.
  - **for loop :**
    - The java for loop is used to iterate a part of the programs.
    - If the number of iteration is fixed, it is recommended to use for loop.
    - **Syntax:**

for (initialization; condition; increment/decrement)
{
Statements;
}

- **Initialization:** It is the initial condition which is executed once when loop start. Here, we can initialize the variable or we can use an already initialized variable.
- **Condition:** It is the second condition which is executed each time to test the condition of the loop.It continues until the condition is false.
- **Increment/decrement:** It increments or decrements the variable value.
- **Statements:** The statement of the loop is executed each time until the second condition is false.
- **Program on For-loop:**

```
public class FactorialNumber {

    public static void main(String[] args) {

        int number=5;

        int fact=1;

        for (int i=1;i<=number;i++)   {

            fact =fact*i;
        }

        System.out.println(fact);
    }
  }

Output : 120
```

- **Java Nested for loop:**
    - If we have a for loop inside another loop, it 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;
             }  // end of inner for loop.

   }   // end of outer for loop.
```

- **Program on Nested for-loop:**

```
public class PrimeNumber2 {

        public static void main(String[] args) {

                System.out.print("Prime numbers are :");
                for (int i=100;i<=200;i++)
                {
                        int c=0;
                        for(int j=2;j<=i;j++) {
                                if(i%j==0) {
                                        c++;
                                }
                        }

                        if(c==1) {
                                System.out.print(" "+i);
                        }
                }

        }
}
```

Output:

```
Prime numbers are: 101 103 107 109 113 127 131 137 139 149

                   151 157 163 167 173 179 181 191 193 197 199
```

▫ While 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.
- This loop iterates continuously until the specified condition is gets true. It is also known as entry-controlled loop since the condition is checked at start of the loop.
- **Syntax:**

```
while (Condition)
        {
    Statements;
     Increments/decrements;
        }
```

- **Program on while loop:**

```
public class FibonnasiSeries {

    public static void main(String[] args) {

            int m=12;
            int a=0;
            int b=1;
            int c;

            int i=0;

            System.out.print("FibonnasiSeries is as  :");

            while (i<=m) {

                    c=a+b;

                    a=b;
                    b=c;
                    i++;

                    System.out.print(" " +c);
            }
    }
}
```

Output : Fibonacci Series is as : 1 2 3 5 8 13 21 34 55 89 144 233 377

- ### do while loop:
    - The java do-while loop is used to iterate a part of the program several times. It is used if the number of iteration is not fixed and we must have to execute the loop at least once.
    - Java do while loop is also called an exit control loop.
    - Because the condition is checked after the loop Body.
    - **Syntax:**

```
do
   {
      Statements.
    }
   while (Condition)
```

```
public class DoWhileLoop {

       public static void main(String[] args) {

              int i =1;

              do {

                     System.out.println("Do while loop is run upto 
5");
                     i++;
              }

              while (i<=5);
         }
       }
```

Output :

```
Do while loop is run upto 5
Do while loop is run upto 5
Do while loop is run upto 5
Do while loop is run upto 5
Do while loop is run upto 5
```

- ## Jump Statements:
    - ### break:
        - Java break keyword is used to break the loop or switch statement.
        - It breaks the current flow of the program at specified conditions.

```
public class Simple {

        public static void main(String[] args) {

     for (int i=1 ; i<=5 ; i++) {

                    if (i==3) {
                            break;
                             }
                    System.out.println("Loop breaks at : " + i);
                }
              }
           }
```

Output :

```
Loop breaks at : 1
Loop breaks at : 2
```

- **Continue :**
  - Java continue keyword is used to continue the loop.
  - It continues the current flow of the program and skips the remaining code at the specified condition.

```
public class Simple {

        public static void main(String[] args) {

     for (int i=1 ; i<=5 ; i++) {

                    if (i==3) {
                            continue;
                             }
                    System.out.println("Loop continue at : " +
i);
                }
              }
            }
```

Output :

```
Loop continue at : 1
Loop continue at : 2
Loop continue at : 4
Loop continue at : 5
```

Java Keywords:

- Java keywords are also known as reserved words.
- Keywords are particular words that act as a key to a code. These are predefined words by Java so they cannot be used as a variable or object name or class name.
- **List of Java Keywords:**

  - **abstract:**
    - Java abstract keyword is used to declare an abstract class.
    - An abstract class can provide the implementation of the interface.
    - It can have abstract and non-abstract methods.

  - **boolean:**
    - Java boolean keyword is used to declare a variable as a boolean type.
    - It can hold True and False values only.

  - **break:**
    - Java break keyword is used to break the loop or switch statement.
    - It breaks the current flow of the program at specified conditions.

  - **byte:**
    - Java byte keyword is used to declare a variable that can hold 8-bit data values.

  - **case:**
    - Java case keyword is used with the switch statements to mark blocks of text.

  - **catch:**
    - Java catch keyword is used to catch the exceptions generated by try statements. It must be used after the try block only.

  - **char:**
    - Java char keyword is used to declare a variable that can hold unsigned 16-bit Unicode characters.

  - **class:**
    - Java class keyword is used to declare a class.

  - **continue:**
    - Java continue keyword is used to continue the loop.
    - It continues the current flow of the program and skips the remaining code at the specified condition.

- **default:**
  - Java default keyword is used to specify the default block of code in a switch statement.

- **do:**
  - Java do keyword is used in the control statement to declare a loop.
  - It can iterate a part of the program several times.

- **double:**
  - Java double keyword is used to declare a variable that can hold 64-bit floating-point number.

- **else:**
  - Java else keyword is used to indicate the alternative branches in an if statement.

- **enum:**
  - Java enum keyword is used to define a fixed set of constants.
  - Enum constructors are always private or default.

- **extends:**
  - Java extends keyword is used to indicate that a class is derived from another class or interface.

- **final:**
  - Java final keyword is used to indicate that a variable holds a constant value.
  - It is used with a variable.
  - It is used to restrict the user from updating the value of the variable.

- **finally:**
  - Java finally keyword indicates a block of code in a try-catch structure.
  - This block is always executed whether an exception is handled or not.

- **float:**
  - Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.

- **for:**
  - Java for keyword is used to start a for loop. It is used to execute a set of instructions/functions repeatedly when some condition becomes true.
  - If the number of iteration is fixed, it is recommended to use a for loop.

- **if:**
  - Java if keyword tests the condition. It executes the if block if the condition is true. ,
- **implements:**
  - Java implements keyword is used to implement an interface.
- **import:**
  - Java import keyword makes classes and interfaces available and accessible to the current source code.
- **instance of:**
  - Java instance of keyword is used to test whether the object is an instance of the specified class or implements an interface.
- **int:**
  - Java int keyword is used to declare a variable that can hold a 32-bit signed integer.
- **interface:**
  - Java interface keyword is used to declare an interface.
  - It can have only abstract methods.
- **long:**
  - Java long keyword is used to declare a variable that can hold a 64-bit integer.
- **native:**
  - Java native keyword is used to specify that a method is implemented in native code using JNI (Java Native Interface).
- **new:**
  - Java new keyword is used to create new objects.
- **null:**
  - Java null keyword is used to indicate that a reference does not refer to anything. It removes the garbage value.
- **package:**
  - Java package keyword is used to declare a Java package that includes the classes.

- **private:**
  - Java private keyword is an access modifier.
  - It is used to indicate that a method or variable may be accessed only in the class in which it is declared.

- **protected:**
  - Java protected keyword is an access modifier.
  - It can be accessible within the package and outside the package but through inheritance only. It can't be applied with the class.

- **public:**
  - Java public keyword is an access modifier.
  - It is used to indicate that an item is accessible anywhere. It has the widest scope among all other modifiers.

- **return:**
  - Java return keyword is used to return from a method when its execution is complete.

- **short:**
  - Java short keyword is used to declare a variable that can hold a 16-bit integer.

- **static:**
  - Java static keyword is used to indicate that a variable or method is a class method.
  - The static keyword in Java is mainly used for memory management.

- **super:**
  - Java super keyword is a reference variable that is used to refer to parent class objects.
  - It can be used to invoke the immediate parent class method.

- **switch:**
  - The Java switch keyword contains a switch statement that executes code based on test value.
  - The switch statement tests the equality of a variable against multiple values.

- **synchronized:**
  - Java synchronized keyword is used to specify the critical sections or methods in multithreaded code.

- **this:**
  - Java this keyword can be used to refer the current object in a method or constructor.

- **throw:**
  - The Java throw keyword is used to explicitly throw an exception.
  - The throw keyword is mainly used to throw custom exceptions. It is followed by an instance.

- **throws:**
  - The Java throws keyword is used to declare an exception.
  - Checked exceptions can be propagated with throws.

- **try:**
  - Java try keyword is used to start a block of code that will be tested for exceptions.
  - The try block must be followed by either catch or finally block.

- **void:**
  - Java void keyword is used to specify that a method does not have a return value.

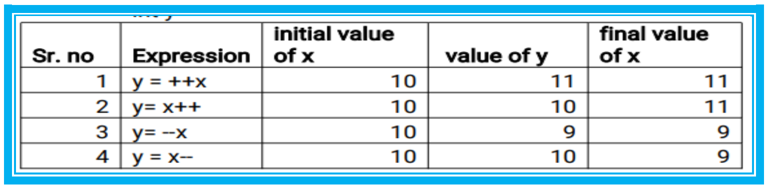
- **while:**
  - Java while keyword is used to start a while loop.
  - This loop iterates a part of the program several times.
  - If the number of iteration is not fixed, it is recommended to use the while loop.

Operators:

- Operators are nothing but to perform some operations on variables called as operators.
- There are many types of operators are given below:
  - Unary Operator
  - Bitwise Operator
  - Arithmetic Operator
  - Logical Operator
  - Shift Operator
  - Ternary Operator
  - Relational Operator
  - Assignment Operator

- **Unary Operator:**
  - In java unary operator require only one operand. Unary operators are used to perform various operation.
  - Incrementing/decrementing value by one
  - Negative an expression
  - Inverting the value of a Boolean.

Prefix	++expr, --expr
Postfix	expr++, expr--



Sr. no	Expression	initial value of x	value of y	final value of x
1	y = ++x	10	11	11
2	y= x++	10	10	11
3	y= --x	10	9	9
4	y = x--	10	10	9

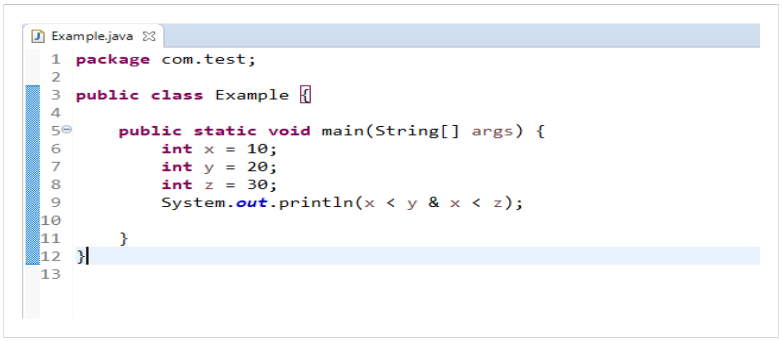
```
public class UnaryOperators {
	public static void main(String[] args) {

		int a = 10;
		a++; //it will increment value by 1 - 11
		System.out.println("value of a is=" + a);
//11
		a--; //it will decrement value by 1 -  10
		System.out.println("value of a is=" + a);
//10
	}
}
```

Output :

```
 value of a is=11
value of a is=10
```

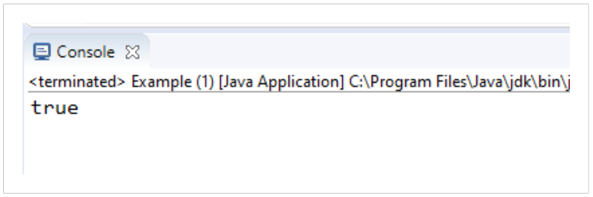
- **Bitwise operators:**
  - This operator is used to perform Bitwise AND & OR operation.
  - **Bitwise AND (&) operators:**
  - The bitwise & operator always checks both conditions whether first condition is true or false.

Expression 1	Expression 2	Results
T	T	T
T	F	F
F	T	F
F	F	F



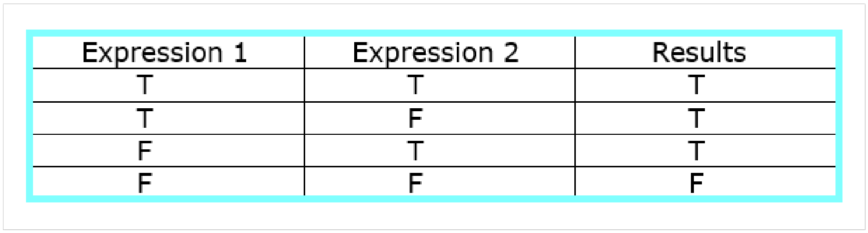
```
Example.java
package com.test;

public class Example {

    public static void main(String[] args) {
        int x = 10;
        int y = 20;
        int z = 30;
        System.out.println(x < y & x < z);

    }
}
```

- In this example, first condition 10<20 is become true and second condition 10<30 is becomes true, hence output is true.
- Output-



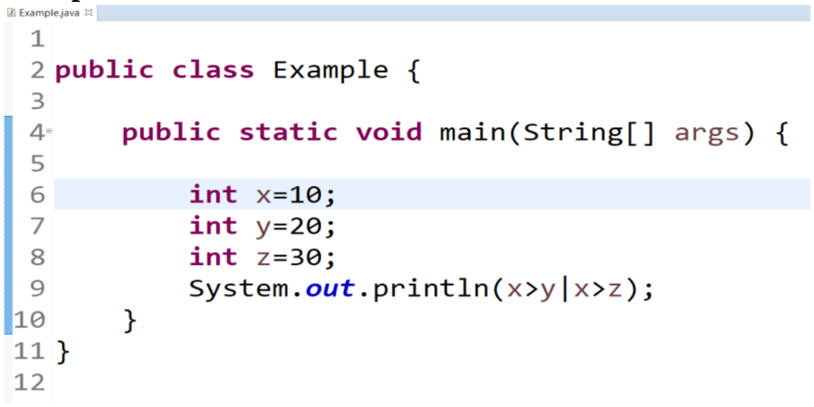
```
Console
<terminated> Example (1) [Java Application] C:\Program Files\Java\jdk\bin\j
true
```

- **Bitwise OR (|) operators-**
  - The bitwise (|) operator always checks both conditions whether first condition is true or false.



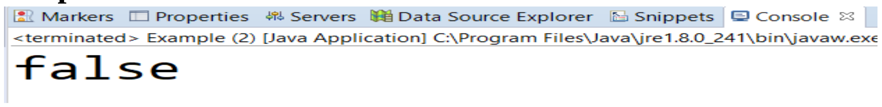
Expression 1	Expression 2	Results
T	T	T
T	F	T
F	T	T
F	F	F

- **Example-**



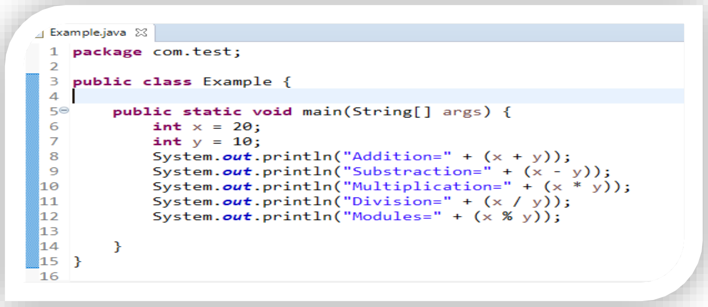
```
Example.java

public class Example {

    public static void main(String[] args) {

        int x=10;
        int y=20;
        int z=30;
        System.out.println(x>y|x>z);
    }
}
```

- In second example, first condition 10>20 is becomes false and second condition 10>30 is becomes false, hence output is false.
- **Output-**



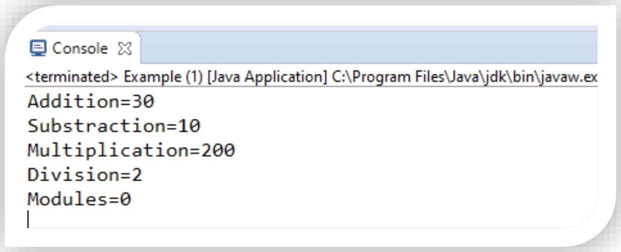
```
Markers  Properties  Servers  Data Source Explorer  Snippets  Console
<terminated> Example (2) [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe
false
```

- **Arithmetic operators:**
  - This operator is used to perform some mathematical operation such as addition (+), subtraction (-), Multiplication (*), Division (/) and modules (%), etc.
  - Example-



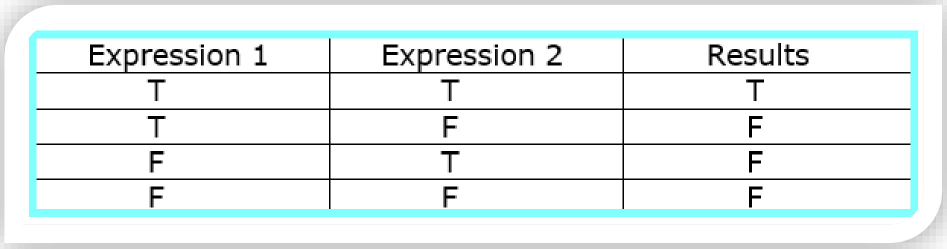
```
package com.test;

public class Example {

    public static void main(String[] args) {
        int x = 20;
        int y = 10;
        System.out.println("Addition=" + (x + y));
        System.out.println("Substraction=" + (x - y));
        System.out.println("Multiplication=" + (x * y));
        System.out.println("Division=" + (x / y));
        System.out.println("Modules=" + (x % y));

    }
}
```

  - **Output-**



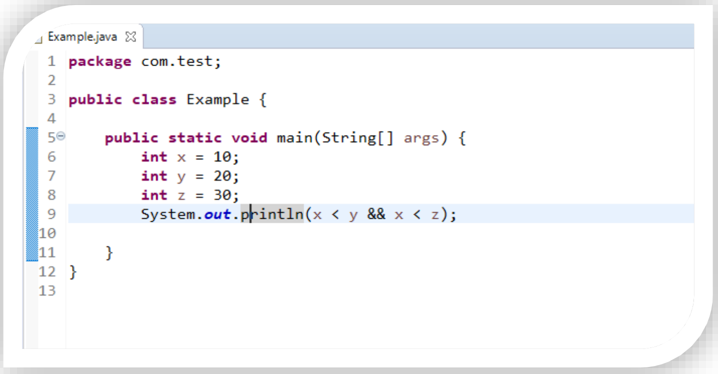
```
<terminated> Example (1) [Java Application] C:\Program Files\Java\jdk\bin\javaw.ex
Addition=30
Substraction=10
Multiplication=200
Division=2
Modules=0
```

- **Logical operators:**
  - This operators are used to perform logical AND & OR operation.
  - **Logical AND (&&) operators-**
  - Logical && operator doesn't check second condition if first condition is false. It checks second condition only if first one is true.



Expression 1	Expression 2	Results
T	T	T
T	F	F
F	T	F
F	F	F

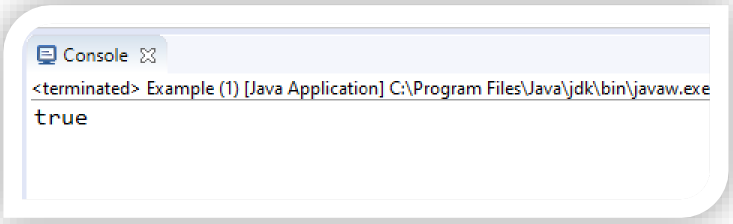
- **Example-**



```
package com.test;

public class Example {

    public static void main(String[] args) {
        int x = 10;
        int y = 20;
        int z = 30;
        System.out.println(x < y && x < z);

    }
}
```

- In this example, first condition 10<20 is becomes true and second condition 10<30 is becomes true, both conditions are true, hence output is true.

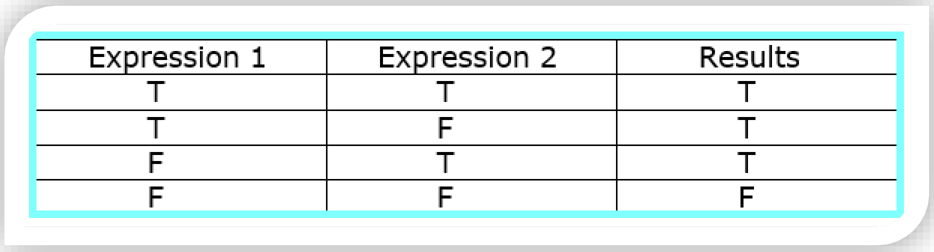
- **Output-**



```
<terminated> Example (1) [Java Application] C:\Program Files\Java\jdk\bin\javaw.exe
true
```

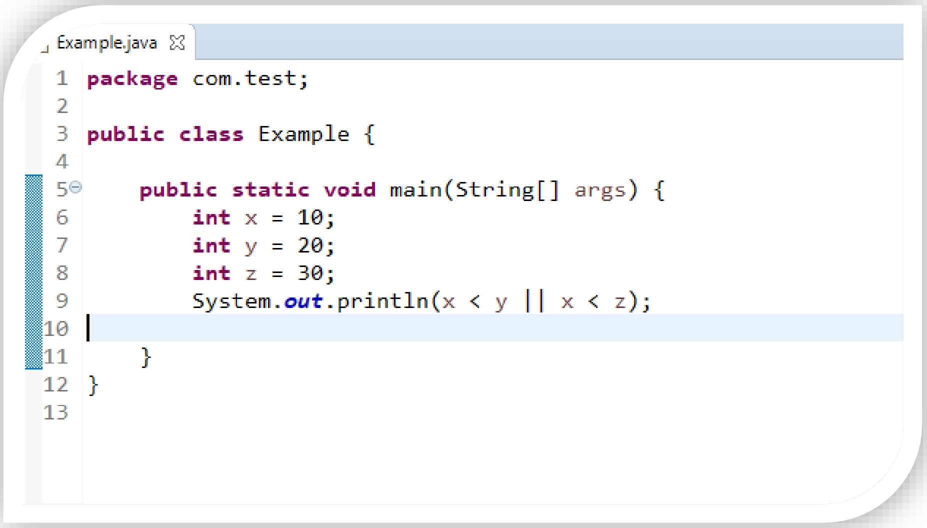
Logical OR (||) operators-

- Logical || operator doesn't check second condition if first condition is true. It checks second condition only if first one is false.



Expression 1	Expression 2	Results
T	T	T
T	F	T
F	T	T
F	F	F

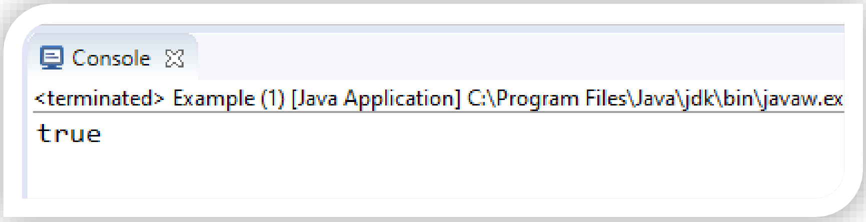
- **Example-**



```
package com.test;

public class Example {

    public static void main(String[] args) {
        int x = 10;
        int y = 20;
        int z = 30;
        System.out.println(x < y || x < z);

    }
}
```

- In this example, first condition 10<20 is becomes true and second condition 10<30 is becomes true, hence output is true.

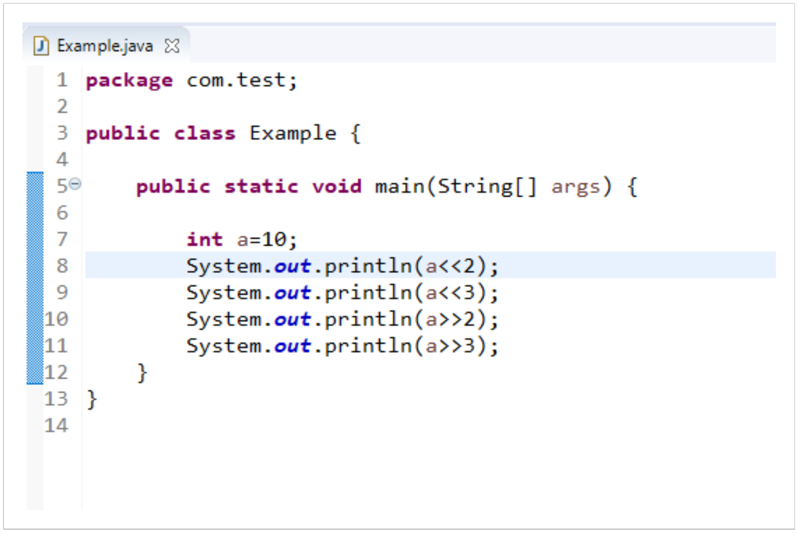
- **Output-**



```
<terminated> Example (1) [Java Application] C:\Program Files\Java\jdk\bin\javaw.ex
true
```

▪ Shift operators (Right/Left)-
  - **Right shift operator** >> is used to move left operands value to right by the number of bits specified by the right operand.
  - For ex. 20>>2   20/2^2=5
  - **Left shift operator** << is used to shift all of the bits in a value to the left side of a specified number of times.
  - For ex. 10<<2   10*2^2=40

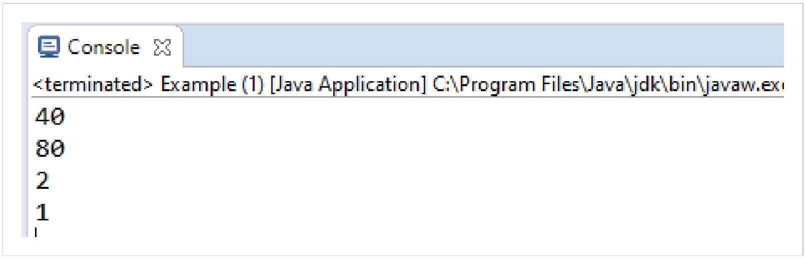
- Example-



Example.java

```
package com.test;

public class Example {

    public static void main(String[] args) {

        int a=10;
        System.out.println(a<<2);
        System.out.println(a<<3);
        System.out.println(a>>2);
        System.out.println(a>>3);
    }
}
```

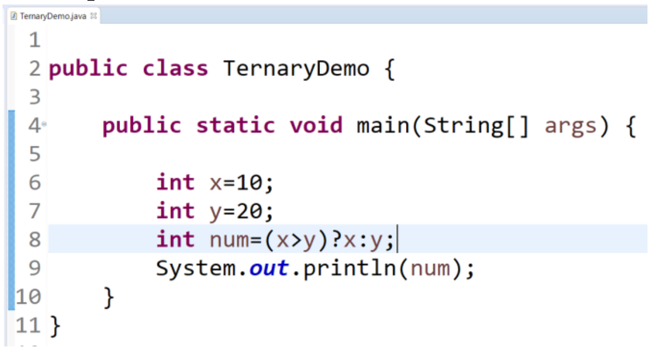
- **Output-**



Console

```
<terminated> Example (1) [Java Application] C:\Program Files\Java\jdk\bin\javaw.ex
40
80
2
1
```

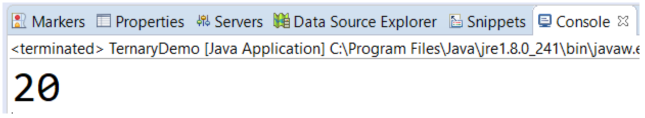
- **Ternary operators-**
  - It includes three operands.
  - **Why?**
  - If else statement requires group of line code to execute the statement but by using this, we can write the code into one line only.
  - Example



TernaryDemo.java

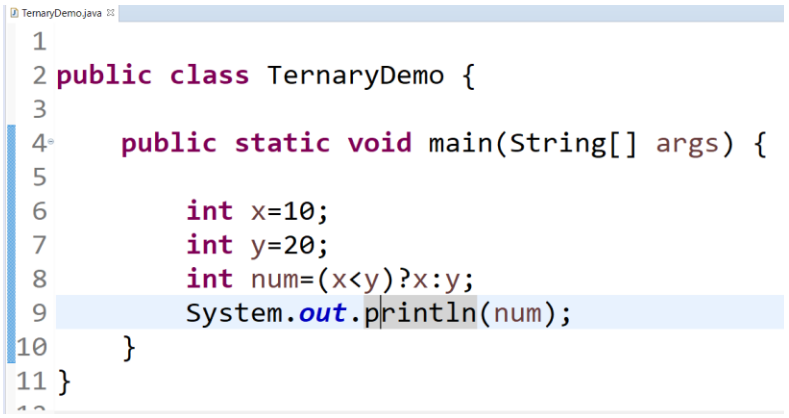
```

public class TernaryDemo {

    public static void main(String[] args) {

        int x=10;
        int y=20;
        int num=(x>y)?x:y;
        System.out.println(num);
    }
}
```

- In this example, condition 10>20 becomes false, so output is 20.
- Output:



Markers Properties Servers Data Source Explorer Snippets Console

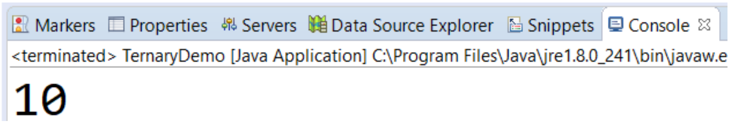
```
<terminated> TernaryDemo [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.e
20
```



TernaryDemo.java

```

public class TernaryDemo {

    public static void main(String[] args) {

        int x=10;
        int y=20;
        int num=(x<y)?x:y;
        System.out.println(num);
    }
}
```

- **Output-**

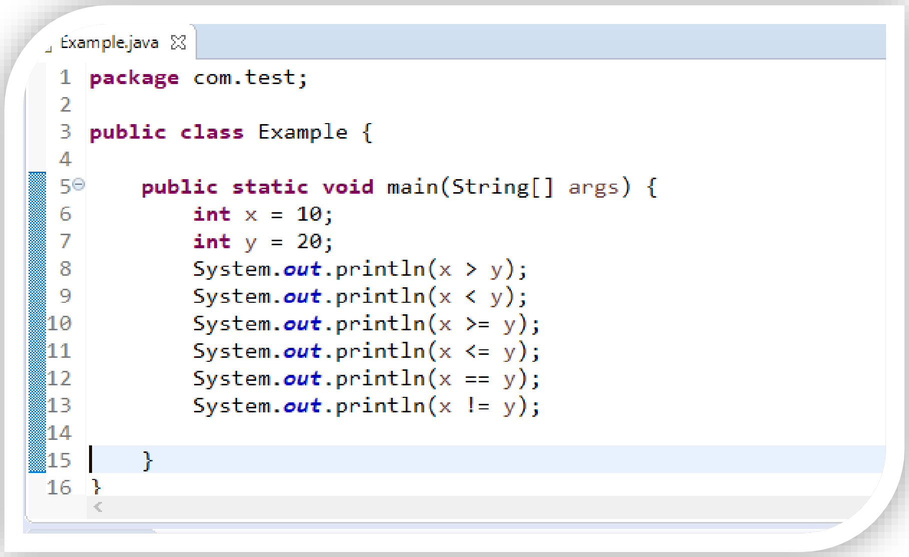


Markers Properties Servers Data Source Explorer Snippets Console

```
<terminated> TernaryDemo [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.e
10
```

Relational Operators:

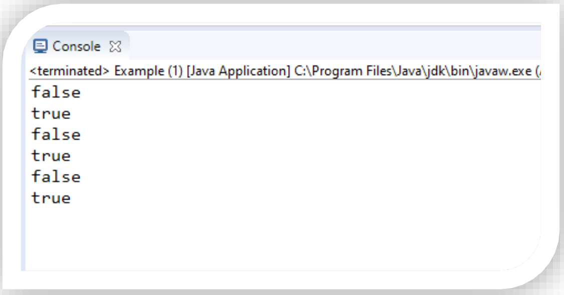
- This operators are used to perform greater than (>), less than (<), greater than or equal to (>=), less than or equal to (<=), equal to (==), not equal to (!=), etc.
- **Example-**



Example.java

```
package com.test;

public class Example {

    public static void main(String[] args) {
        int x = 10;
        int y = 20;
        System.out.println(x > y);
        System.out.println(x < y);
        System.out.println(x >= y);
        System.out.println(x <= y);
        System.out.println(x == y);
        System.out.println(x != y);

    }
}
```

- **Output-**



```
<terminated> Example (1) [Java Application] C:\Program Files\Java\jdk\bin\javaw.exe (
false
true
false
true
false
true
```

- **Assignment operators-**
  - This operator is used to assign the values to variable.
  - Syntax- Variable =value;

```
public class AssignmentOperators {

	public static void main(String[] args) {

		int a=10;
		System.out.println("value of a 
is>>"+a);
	}
}
```

Output : `value of a is>>10`

Java User Input (Scanner Class) :

- **Scanner** is a class in **java.util** package used for obtaining the input of the primitive types like int, double, etc. and strings.
- It is the easiest way to read input in a Java program.
- To create an object of Scanner class, we usually pass the predefined object System.in.
- To read numerical values of a certain data type, the method to use is ***nextXYZ()***.
- For example, to read a value of type short, we can use ***nextShort()*** and so on.
- To read strings, we use ***nextLine()* or *next()*.**
- 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)

- **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.

```
public class Example {

        public static void main(String args[]) {

                Scanner scanner = new Scanner(System.in);
                System.out.println("Enter the number>>");
                int no = scanner.nextInt();
                System.out.println("Enter the name>>");
                String name = scanner.next();
                getStudentInformation(no, name);
        }
```

```
	static void getStudentInformation(int no, String 
name) {
		System.out.println("Student no is>>" + no);
		System.out.println("Student name is>>" + 
name);
	}

		}
```

Output :

```
Enter the number>>

29
Enter the name>>

Tanvir Shinde
Student no is>>29
```

```
public class Factorial {

	public static void main(String[] args) {

		Scanner jk = new Scanner (System.in);

		System.out.println("Enter your number?");

		int number = jk.nextInt();

		int i;
		int fact=1;

		for ( i=1; i<=number ;i++) {

			fact = fact*i;
		}
		System.out.println("Your value is:"+fact);
		jk.close();
	}
}
```

Output :

```
Enter your number?

5
Your value is:120
```

Differences between C and JAVA :

C	JAVA
C is a Procedural Programming Language.	Java is an Object-Oriented Language.
C was developed by Dennis M.Ritchie in 1972.	Java language was developed by James Gosling in 1995.
In the C declaration variable are declared at the beginning of the block.	In Java, you can declare a variable anywhere.
C does not support multithreading.	Java has a feature of threading.
C support pointers.	Java does not support pointers.
Memory allocation can be done by malloc.	Memory allocation can be done by new keyword.
Garbage collector needs to manage manually.	Garbage collector is managed automatically.
C does not have a feature of overloading functionality.	Java supports method overloading.
C is platform dependent language.	Java is platform independent language.
C is library Oriented language. #include	Java is package and class based language. package java.util;
Operating system is responsible to call main ( ).	JVM is responsible to call main( ).

OOPs (Object-Oriented Programming System):

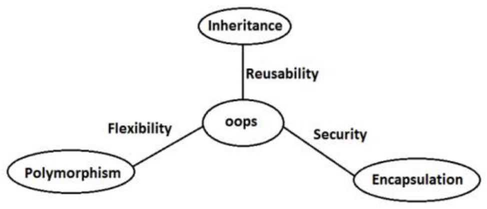
- Object means a real-world entity such as a pen, chair, table, computer, watch, etc.
- Object-Oriented Programming is a methodology to design a program using classes and objects.
- ***Pillars/feature of OOPs:-***
    - *Object*
    - *Class*
    - *Inheritance*
    - *Polymorphism*
    - *Abstraction*
    - *Encapsulation*

Object: -

- **Any entity that has state and behavior is known as an object.**
- For example, a chair, pen, table, keyboard, bike, etc.
- It can be physical or logical.
- An Object can be defined as an instance of a class.
- An object contains an address and takes up some space in memory.
- Objects can communicate without knowing the details of each other's data.

Class:

- Collection of objects is called class. It is a logical entity.
- A class can also be defined as a blueprint from which you can create an individual object.
- Class doesn't consume any space.

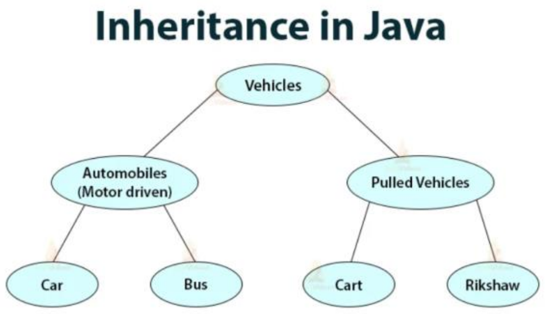


- Inheritance talks about reusability.
- Polymorphism talks about flexibility.
- Encapsulation talks about security.

Inheritance:

- This represents a relationship through which every property (methods, variable) of super class will be by default available to the sub class.
- Inheritance in Java is a mechanism in which one class acquires all the properties and behaviours of another class with the help of extends keyword.
- Inheritance is one of the most important object-oriented programming language concept.

- Also, we can say that derived class can acquire the properties of baseclass with **extends keyword.**
- By using "extends" keywords we can implement IS-A relationship.
- The main advantage of IS-A relationship is reusability.
- **Real time Example of Inheritance:**



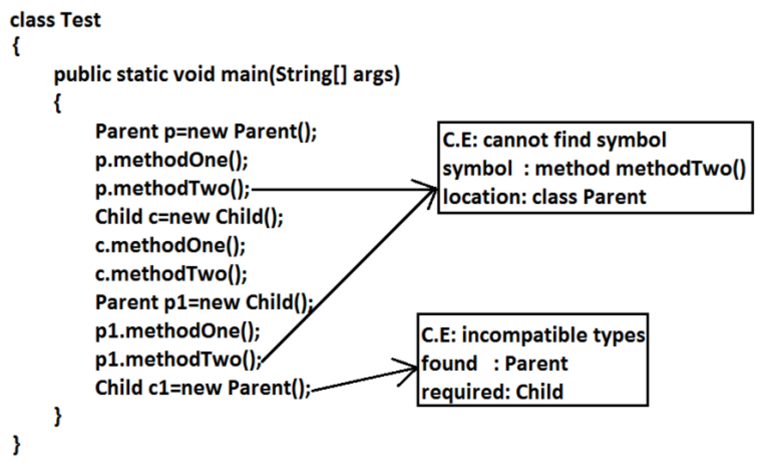
- **Automobiles** and **Pulled Vehicles** are subclasses of **Vehicles**.
- **Vehicles** are the base class or superclass of **Automobiles** and **pulled Vehicles**.
- **Car** and **Bus** are sub-classes or derived classes of **Automobiles**.
- **Automobiles** are the base class or superclass of **Car** and **Bus.**

- **Advantages of Inheritance:**
- Why use inheritance in java
- For code reusability.
- For code optimization.
- The main advantage of inheritance is reusability that means we have to define a code a place (we specified some logic in a class) which can be use by multiple classes.
- Super class can have multiple child classes but converse is not possible.
- A class can extend only one class at a time that is multilevel inheritance is possible but multiple inheritance is not possible.
- Cyclic inheritance is not possible.
- Same class cannot inherit itself.

- **Types of Inheritance:**
  - Single level inheritance
  - Multi-level inheritance
  - Multiple inheritance
  - Hierarchical inheritance

Example:

```
class Parent {
public void methodOne(){ }
}

class Child extends Parent {
public void methodTwo() { }
}
```



```
class Test
{
    public static void main(String[] args)
    {
        Parent p=new Parent();
        p.methodOne();
        p.methodTwo();
        Child c=new Child();
        c.methodOne();
        c.methodTwo();
        Parent p1=new Child();
        p1.methodOne();
        p1.methodTwo();
        Child c1=new Parent();
    }
}
```

```
C.E: cannot find symbol
symbol  : method methodTwo()
location: class Parent
```

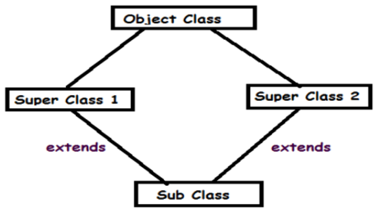
```
C.E: incompatible types
found   : Parent
required: Child
```

- **Conclusion :**
  - Whatever the super class has by default available to the Sub class but whatever the Sub class has by default not available to the super class.
  - Hence on the child reference we can call both super and Sub class methods. But on the super reference we can call only methods available in the super class and we can't call child specific methods.
  - Super class reference can be used to hold sub class object but by using that reference we can call only methods available in parent class and child specific methods we can't call.
  - Sub class reference cannot be used to hold Super class object.

♦ **Single level inheritance:**
- For single level inheritance two classes are mandatory.
- It is an operation where inheritance takes place between two classes to perform single level inheritance.
- In short we can say that when a class inherits another class it is known as single level Inheritance.

- **Multi-level inheritance:**
  - Multilevel inheritance takes place between three or more classes.
  - In multilevel inheritance one subclass (derived class) can acquire the property of another superclass (base class) and this phenomenon continues so that we **know as "Multi level Inheritance"**
  - In short when there is a chain of inheritance it is also known as multilevel inheritance.

- **Multiple inheritance:**
  - In multiple inheritance one subclass (derived class) acquiring properties of two super classes (two base classes) at a same time so this known as multiple inheritance.
  - **Java doesn't support multiple inheritance because it results diamond ambiguity problem.**
  - It means the diamond like structure get created in JVM and object variable get confused for whom need to inherited the property of superclass (base class)into subclass (derived class).
  - Superclass (base class) of all the classes is object class so there diamond ambiguity forms.
  - To reduce the complexity and simplify the language multiple inheritance not supported in java.



- **Hierarchical inheritance:**
  - Hierarchical Inheritance takes place between one superclass (base class) and multiple subclass (derived class).
  - So, the property of superclass (base class) can be acquired by multiple subclass (derived class) this is known as **Hierarchical Inheritance.**
  - The two or more classes inherits a single class it is known as Hierarchical Inheritance.
  - This inheritance takes place between one superclass (base class) and multiple (derived class).

- **Inheritance with respect to the variables:**
  - All the global or class level variable will be available to the Sub class by default like methods.
  - If the same name of non-static variable in Super class as well as sub class. If we want to access the non-static variable from Super class then in that non-static method (of child class).We can access it **by using "super" keyword.**
  - If the same name of static variable is available is sub class as well as in super class. Then we can access the particular **variable by Classname.variable name**
  - If we accessing it through only name then it points towards the nearest static variable.
  - All the global variables defines in the parent class can be accessible to the child class.

- **Inheritance with respect to the Constructor:**
  - Super class constructor will call automatically Sub class constructor whether the constructor is present in parent class or not.
  - If constructor is not present in the Super class then it will call default constructor of parent class. Then it call the constructor of sub class.
  - If sub class constructor does not match with the super class (which means type of constructor of super class and sub Class). Then specifically we have to call super class constructor in the first line of sub class by using super keyword like super (inside argument);
  - Constructor does not follow inheritance principle as all the constructors available in parent class are not by default available to the child class.
  - **Hence constructor does not follow inheritance principle.**

This keyword in Java :

- **this keyword:**
- It is a reserved keyword in Java that is used to refer to the current class object.
- It is a reference variable through which the method is called.
- **Other uses of this keyword are:**

  We can use it to refer current class instance **variable.**
  We can use it to invoke the current class method (implicitly).
  We can pass it as an argument in the **method and constructor calls.**
  We can also use it for returning the current class instance from the method.

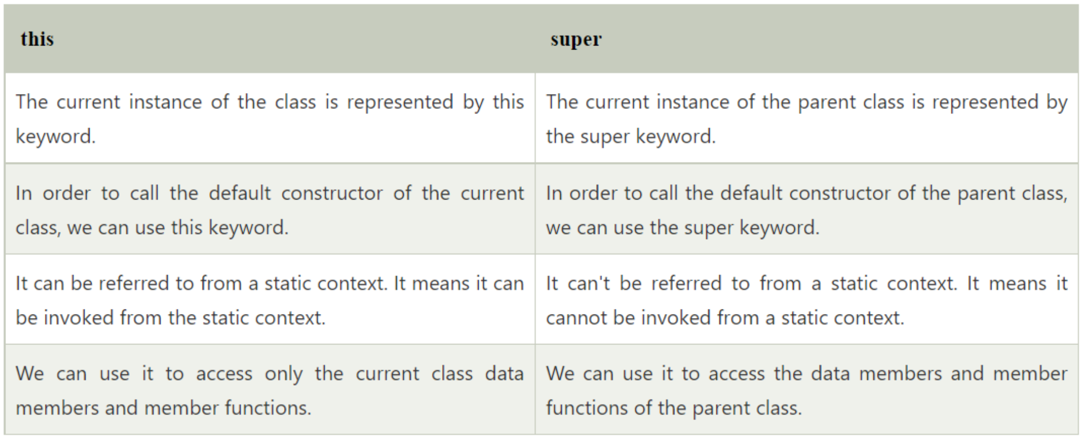
- **this() Constructor:**
  - The constructor is used to call one constructor from the other of the same class. Let's take an example of both **this** keyword and **this ()** to understand how they work.

Super keyword in Java :

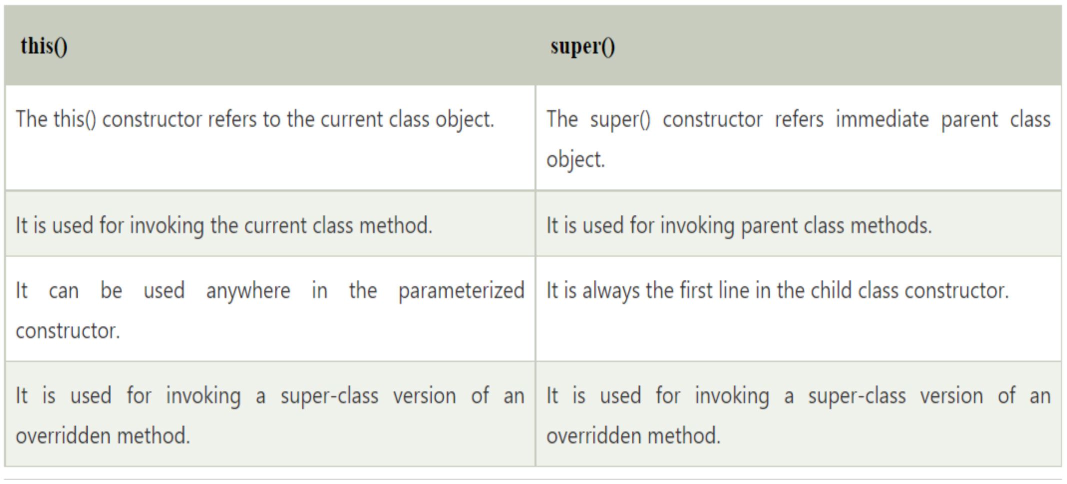
- The super keyword in Java is a reference variable which is used to refer immediate Base or super class object.
- Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

- **Usage of Java super Keyword**
- super can be used to refer immediate parent class instance variable.
- super can be used to invoke immediate parent class method.
- super() can be used to invoke immediate parent class constructor.

Difference between this and Super Keyword:

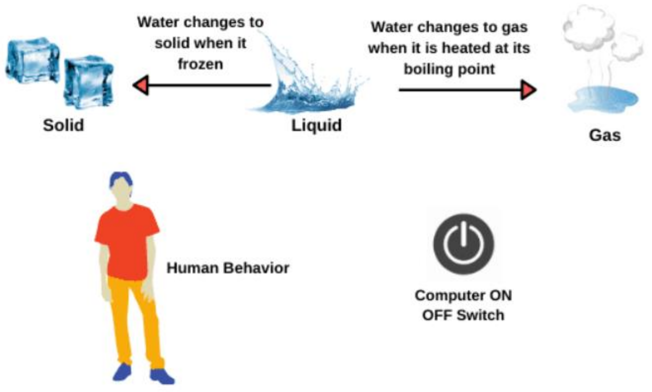


this	super
The current instance of the class is represented by this keyword.	The current instance of the parent class is represented by the super keyword.
In order to call the default constructor of the current class, we can use this keyword.	In order to call the default constructor of the parent class, we can use the super keyword.
It can be referred to from a static context. It means it can be invoked from the static context.	It can't be referred to from a static context. It means it cannot be invoked from a static context.
We can use it to access only the current class data members and member functions.	We can use it to access the data members and member functions of the parent class.



this()	super()
The this() constructor refers to the current class object.	The super() constructor refers immediate parent class object.
It is used for invoking the current class method.	It is used for invoking parent class methods.
It can be used anywhere in the parameterized constructor.	It is always the first line in the child class constructor.
It is used for invoking a super-class version of an overridden method.	It is used for invoking a super-class version of an overridden method.

- **Polymorphism:**
  - Polymorphism is derived form 2 Greek words: poly and morphs so poly means many and morphs mean forms
  - **Definition:** Method having different forms with the same name is called **polymorphism.**
  - **Types of Polymorphism:**
  - **Overloading and Overriding.**
  - **Real Time example of Polymorphism:**
  - In real-time, polymorphism can be explained as the different roles played by a single person. For instance, a man can be a son, husband, father, etc.
  - We all know that water is a liquid, but it changes to solid when it frozen, and it changes to a gas when it is heated at its boiling point.
  - We all use a single button to switch ON and OFF the computer.


Water changes to solid when it frozen
Water changes to gas when it is heated at its boiling point
Solid
Liquid
Gas
Human Behavior
Computer ON OFF Switch

- **Overloading:**
  - A method can be called as overloaded if and only if the method name is same but the argument is different or we can say at least order of the argument passes or data type of the arguments are different
  - Overloading is also known as **compile time polymorphism** and **early binding.**
  - Method overloading is performed within the class.
  - **Reason:**
  - At the time of compilation of program for over loaded method we get to know which method will get execute.
  - In Overloading method resolution is based on reference variable.
  - **Note:** Overloading is applicable for inheritance as well.

```
public class Overrloading {

        public void method1 (int a) {

                System.out.println(" one argument Output :"+ a);
}
```

```
                public void method1 (int b, int c) {

                        int d=b+c;
        System.out.println("two  argument Output :"+ d);
        }

        public static void main(String[] args)
        {
        Overrloading ol = new Overrloading () ;
        ol.method1(10);
        ol.method1(12, 13);
        }
        }
```

Output:

```
    one argument Output :10
  two argument Output :25
```

- **Overriding:**
  - A method can be called as overridden method if the name and signature in the sub class and superclass class is same.
  - Method overriding is happen between two class that **have IS-A (Inheritance Relationship)**
  - Note: method resolution is completely based on object.
  - Overriding is also known as **run time polymorphism** and **late binding.**
  - **Reason:**
  - At run time of program for over ridden method we get to know which method will get execute.
  - In Overriding method resolution is based on run time object.
  - If we declare method as final then we cannot override that method
  - **Note:** Parent reference can be used to hold child object but converse is not possible
  - Which means child reference variable cannot be used to hold parent Object.
    - *Parent pp = new Child ();* this is valid in java
    - *Child cc =new Parent ();* this is not valid in java

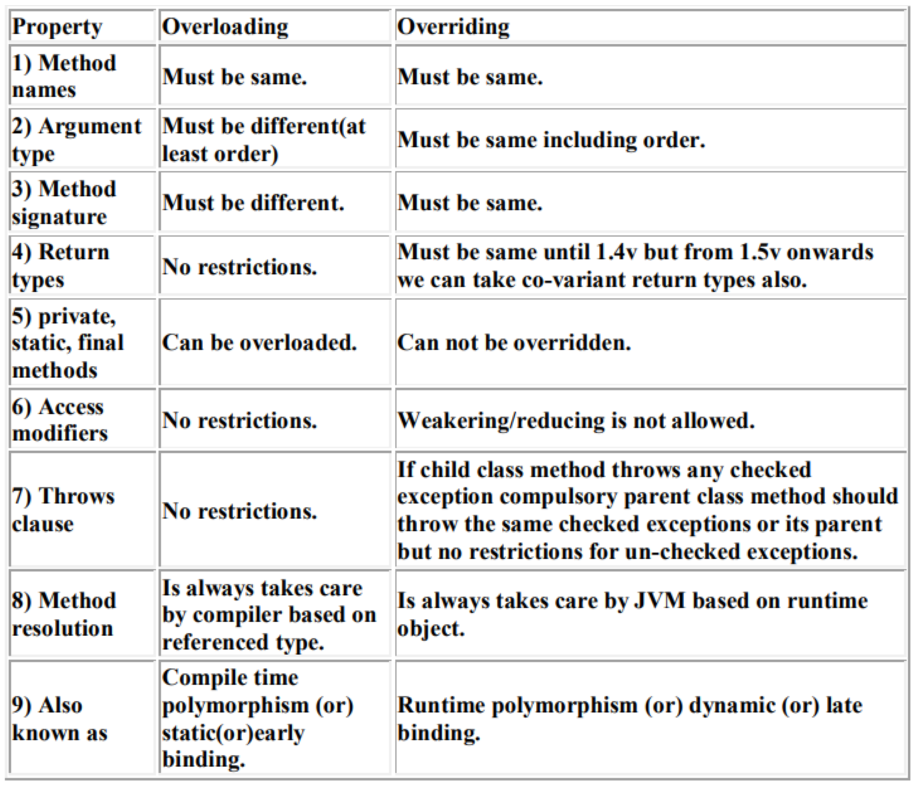
  - ***Overriding with respect to Static methods:***
  - We can't override a static method as non-static or either way after doing the same we will get compile time error.

  - **Reason:**
  - To access static, we don't need an object and for non-static we always need to have an object.

- ***Method hiding:***
  - If two static methods try to get override then it is not overriding but it is method hiding.
  - That means method resolution is based on reference type but not run time object.

- **Polymorphism with respect to variable:**
  - Variable resolution always take care by compiler-based reference type whether the type of variable is static or non-static.
  - In short overriding concept is not applicable to variables but only for methods.

Differences between overloading and overriding?

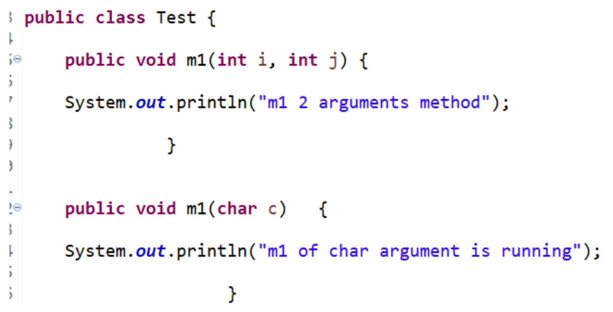


Property	Overloading	Overriding
1) Method names	Must be same.	Must be same.
2) Argument type	Must be different(at least order)	Must be same including order.
3) Method signature	Must be different.	Must be same.
4) Return types	No restrictions.	Must be same until 1.4v but from 1.5v onwards we can take co-variant return types also.
5) private, static, final methods	Can be overloaded.	Can not be overridden.
6) Access modifiers	No restrictions.	Weakering/reducing is not allowed.
7) Throws clause	No restrictions.	If child class method throws any checked exception compulsory parent class method should throw the same checked exceptions or its parent but no restrictions for un-checked exceptions.
8) Method resolution	Is always takes care by compiler based on referenced type.	Is always takes care by JVM based on runtime object.
9) Also known as	Compile time polymorphism (or) static(or)early binding.	Runtime polymorphism (or) dynamic (or) late binding.

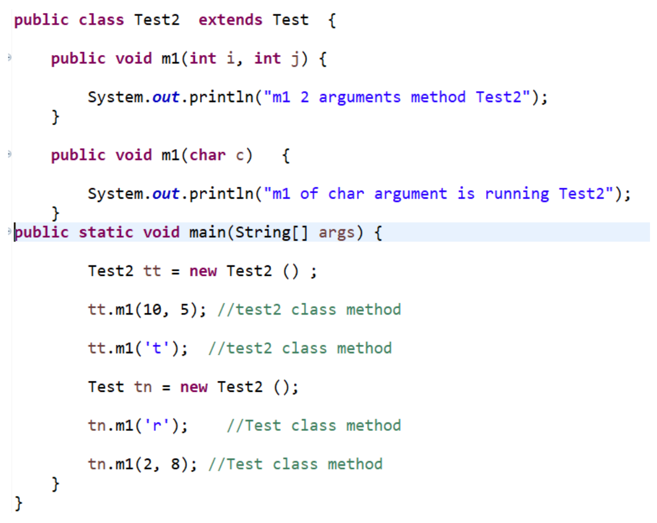
Difference between overriding and method hiding:

overriding	method hiding
Both Super and sub class methods should be non-static.	Both Super and Sub class methods should be static.
Method resolution is always takes care by JVM based on runtime object.	Method resolution is always takes care by compiler based on reference type.
Overriding is also considered as runtime polymorphism (or) dynamic polymorphism (or) late binding.	Method hiding is also considered as compile time polymorphism (or) static polymorphism (or) early biding.

- **Example of Overriding**

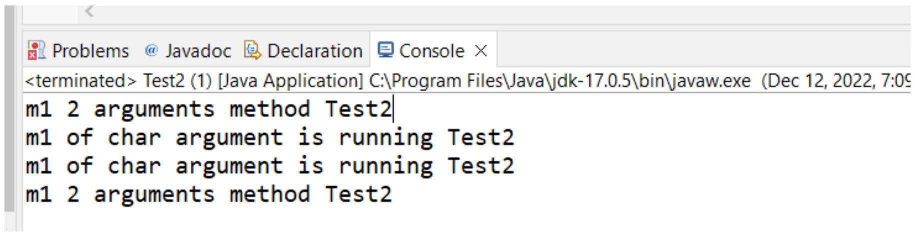


```
public class Test {

    public void m1(int i, int j) {

    System.out.println("m1 2 arguments method");

              }


    public void m1(char c)   {

    System.out.println("m1 of char argument is running");

                     }
```



```
public class Test2  extends Test  {

    public void m1(int i, int j) {

        System.out.println("m1 2 arguments method Test2");
    }

    public void m1(char c)   {

        System.out.println("m1 of char argument is running Test2");
    }
public static void main(String[] args) {

        Test2 tt = new Test2 () ;

        tt.m1(10, 5); //test2 class method

        tt.m1('t');  //test2 class method

        Test tn = new Test2 ();

        tn.m1('r');    //Test class method

        tn.m1(2, 8); //Test class method
    }
}
```

Output:



```
<terminated> Test2 (1) [Java Application] C:\Program Files\Java\jdk-17.0.5\bin\javaw.exe  (Dec 12, 2022, 7:09
m1 2 arguments method Test2
m1 of char argument is running Test2
m1 of char argument is running Test2
m1 2 arguments method Test2
```

Encapsulation:

- Binding of data and corresponding methods into a single unit is called Encapsulation.
- If any java class follows data hiding and abstraction such type of class is said to be encapsulated class.
- For example, a capsule which is mixed of several medicines.
- We can create a fully encapsulated class in java by making all the data members of the class as private.
- And we can use setter and getter methods to set and get the data in it.
- The java Bean class is the example of a fully encapsulated class.
- Encapsulation is just to the data and make it available through hiding the internal functionality to the end user.
- ***Encapsulation = Data Hiding + Abstraction***

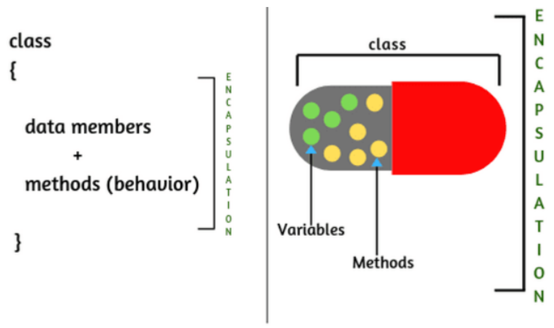
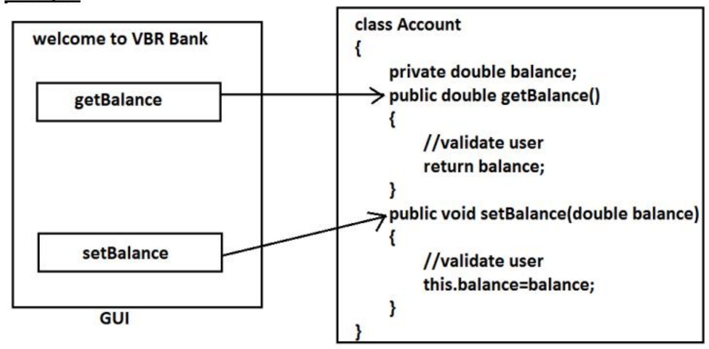


Fig: Encapsulation

- ### Data Hiding:
  - Our internal data should not go out directly that is outside person can't access our internal data directly.
  - By using private modifier, we can implement data hiding.
  - **For example:**

```
class Account {
private double balance;
......................;
......................;
}
```



```
class Account
{
        private double balance;
        public double getBalance()
        {
                //validate user
                return balance;
        }
        public void setBalance(double balance)
        {
                //validate user
                this.balance=balance;
        }
}
```

- After providing proper username and password only , we can access our Account information.
- The main advantage of **data hiding is security.**
- **Note:** recommended modifier for data members is private.

Abstraction:

- Hide internal implementation and just highlight the set of services, is called abstraction.
- By using abstract classes and interfaces we can implement abstraction.
- **For example:**
- *By using ATM GUI screen bank people are highlighting the set of services what they are offering without highlighting internal implementation.*

- **The main advantages of Abstraction are:**
  - We can achieve security as we are not highlighting our internal implementation.(i.e., outside person doesn't aware our internal implementation.)
  - Enhancement will become very easy because without effecting end user we can able to perform any type of changes in our internal system.
  - It provides more flexibility to the end user to use system very easily.
  - It improves maintainability of the application.
  - It improves modularity of the application.
  - It improves easyness to use our system.

- **The main advantages of encapsulation are :**
  - We can achieve security.
  - Enhancement will become very easy.
  - It improves maintainability and modularity of the application.
  - It provides flexibility to the user to use system very easily.

- **The main disadvantage of encapsulation:**
  - it increases length of the code and slows down execution.

Access Modifier:

- Access Modifier: It is defines as the keyword which tells/shows the accessibility of a particular method, variable or class.
- **Access Modifier applicable to class.**
    - **public**
    - **<default>**
    - **final**
    - **abstract**

- **Note: Final cannot be used together with abstract.**
    - **public:** It can accessible throughout the project.

    - **default:** The class which declare as <default> can be accessible within the package. We cannot acess it outside of package.

    - **final:** Whichever class declared as final it cannot follow inheritance. We can use this properties by only creating the object of that particular class.
        - The class which is declared as final cannot be extended by another class
        - **For Ex.** If we declare like this public final Class A which means we can acess the properties Class A to another class by creating an object throughout the project
        - If we declare like this final Class A which means we can acess the properties Class A to another class by creating an object within that particular package only.

    - **Abstract:**
        - A modifier which applicable to class and methods. If we don't have an idea about complete functionality which means we know about the major part but we are not sure about the implementation then we should declare or use abstract modifier.
        - If we declare a class as abstract then it is not necessary to declare an abstract method
        - But if a class contains abstract method then we have to declare that class as abstract.
        - Every child or implementation class of abstract class has to implement all the abstract method otherwise we have to declare that class as abstract and we have provide the implementation in subsequent child or implementation classes.
        - We don't instantiate or make an object of abstract class but we can access the complete methods from abstract class through child object by making IS-A (Inheritance) relationship.
        - Final cannot be used together with abstract.
        - Abstract class can have constructor.
        - Abstract class contains 0-100% abstract methods.

Access Modifier applicable to member functions(methods or variable):

- **Public:** if we declared a method or a variable as public then the visibility of that element throughout the project.
- **Private:** if a particular method or variable is declared as private then we access the method only where it was defined.
- **<Default>:** if a particular method or variable is declare as <default> then it would be accessible throughout the package only.
- **Protected =** <default>+child classes.
  - We can access the protected members with in the package as we are doing in <default> but when we wants to access them outside the package then we should use child class reference only.
- **Final:** if we declared a variable as final then we cannot change it or reassign it later.

Scenario	private	default	protected	public
Within the same class	Yes	Yes	Yes	Yes
From the child class of same package	No	Yes	yes	Yes
From the non-child class of same package	No	yes	yes	Yes
from the child class of outside package	No	No	Yes (We should use child reference only)	yes
From the non-child class of outside the package	No	No	No	yes

Abstraction in Java:

- Abstraction is a process of hiding the implementation *details / code / information* and showing only functionality to the end user.
- Hiding internal functionality and showing external functionality to users.
- Another way, it shows only essential things to the user and hides the internal details
- Example: Sending SMS where you type the text and send the message.
- You don't know the internal processing about the message delivery.
- There are two ways to achieve abstraction in java.
  - ***Abstract Class (0-100%)***
  - ***Interface (100%)***

- **Abstract class in Java:**
  - Abstract is access modifier which is applicable to class and methods.
  - A class which is declared as abstract is known as abstract Class.
  - If we don’t have an idea of complete functionality which means we know about the major part but we are not sure about the implementation then we should declare or use abstract modifier.
  - If we declare a class as abstract method but if a class contains abstract method, then we have to declare that class as abstract.
  - Every child class of abstract class has to implement all the abstract methods otherwise we have to declare that class as abstract and have to provide the implementation in subsequent child classes.
  - We cannot create an object of abstract class but we can access the complete methods from abstract class through child object.
  - Final cannot be used together with abstract.
  - Abstract class can have constructor and static methods also.

- **Abstract Method in Java:**
  - A method which is declared with an abstract keyword and does not have implementation is known as an abstract method.
  - A method does not have a method body.
  - **Example of Abstract Method.**
  - public abstract void methodName () ;

- **Concrete Class:**
  - The class which is responsible to implement all the abstract methods from the abstract class.
  - We can’t create object of abstract class to create object of abstract class there is approach called as **“concrete class”.**

Interface in Java:

- The interface in Java is a mechanism to achieve abstraction.
- It is used to achieve abstraction and multiple inheritance in Java.
- Interface contains 100% abstract methods if we try to define a method with body then it will be an error.
- Whenever we are not sure about complete logic for any of the functionality then we should declare the particular class as Interface.
- To implement an interface, we have to use ***implements keyword.***
- If any class doesn’t want to implement all of the methods, then we should declare that class as abstract and provide the implementation in the child classes.

- If don’t make an object of abstract class. Also, Interface can’t have constructor.
- When an interface inherits another **interface extends keyword** is used whereas **class use implements** keyword to inherit an interface.
- **Example:**

  Interface1 extends Interface2,
  Public Class implements Interface1
  Public Class implements Interface1, Interface2

- All the methods present inside an interface are by default public and abstract whether we are declaring or not.

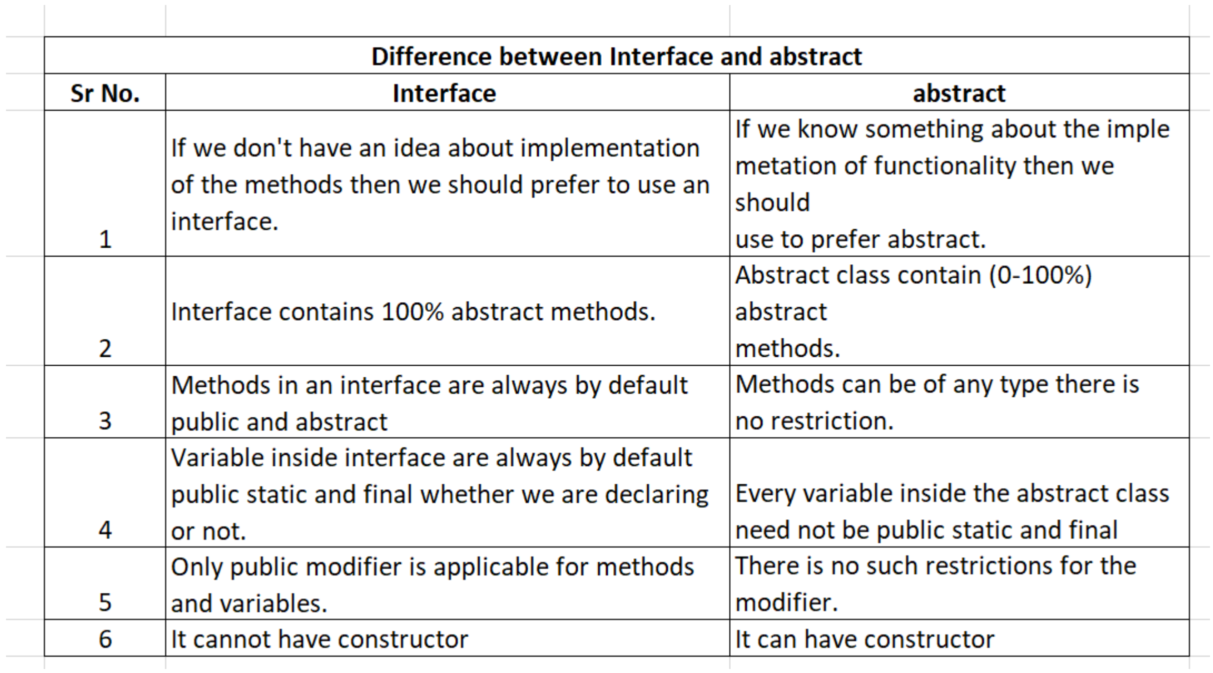
- **Implementation Class:**
  - A class which provides implementation for all the incomplete methods of interface with the help of implement keyword is known **as *Implementation class***
  - At the time of Implementation declared method with public access modifiers compulsory.

- **Static Method with respect to interface:**
  - We can have static method inside an interface provided we have the body of the method as well.
  - *Static method can’t define without body inside an interface.*

- **Variable Resolution with respect to the interface:**
  - Variable inside an interface are by default public static and final whether we are declaring or not.
  - **Reason for public:** *To access them from anywhere.*
  - **Reason for static:** *User should be able access it without object creation.*
  - **Reason for final:** *As static variable share its memory with all objects so implemented class is not allowed to change its value.*

- **Key Points in interface:**
  - If both interfaces have same name of methods in it then the class which is providing the implementation have to define those methods only once.
  - If both the interfaces method having the same signature but return type are different so the class which implements those interface will not able to implement that method.

Difference between Interface and abstract in Java?



Difference between Interface and abstract		
Sr No.	**Interface**	**abstract**
1	If we don't have an idea about implementation of the methods then we should prefer to use an interface.	If we know something about the imple metation of functionality then we should use to prefer abstract.
2	Interface contains 100% abstract methods.	Abstract class contain (0-100%) abstract methods.
3	Methods in an interface are always by default public and abstract	Methods can be of any type there is no restriction.
4	Variable inside interface are always by default public static and final whether we are declaring or not.	Every variable inside the abstract class need not be public static and final
5	Only public modifier is applicable for methods and variables.	There is no such restrictions for the modifier.
6	It cannot have constructor	It can have constructor

Conversion/ Casting in JAVA:

- **Casting:** Converting one type of information into another type of information is called conversion or casting.
- **Casting Types:**
    - Primitive Casting.
    - Non-primitive Casting.

    - **Primitive Casting:**
        - **Implicit type casting: Type(datatype) Casting**
            - It is type of casting where lower data type is converted into higher data type.
            - It is also called ***widening or broadening*** of variable. While performing this implicit type casting there is no *loss of information.*
            - It is also known as ***Generalization.***

```
public class ImplicitTypeCast {

        public static void main(String[] args) {
                byte b=100;
                short c= (short)b;
                System.out.println(c);
        }
}

Output: 100
```

- **Explicit type Casting:**
    - It is type of casting where higher order data type is converted into lower order data type is known as ***down casting.***
    - It is also called narrowing of variable while performing explicit type casting there can be loss of information.
    - It is also known as ***specialization.***
    - Rules applicable for different data type's manipulation.

        Byte + byte = Int
        Short + short = int
        Byte + short = int
        Int + int = long
        Long + double = double

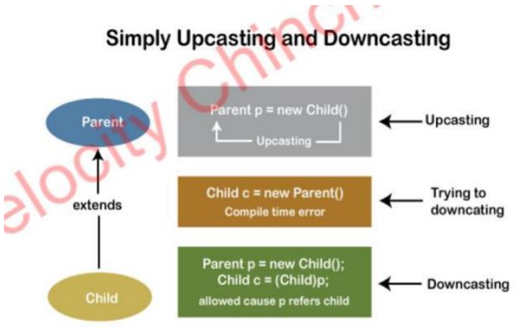
```
public class ExplicitTypeCast {
	public static void main(String[] args) {

		short s= 31767;
		byte b=(byte)s;

		System.out.println(b);

	}
}
```

Output : 23

- **Non-primitive Casting:**
    - Non primitive casting means converting one type of class into another type of ***class is known as non-primitive casting.***
    - **Types of Non-primitive Casting:**
    - **Upcasting: (lower to upper)**
        - Whenever child class is type cast to parent then it is called ***Upcasting.***
        - Assigning subclass property into superclass is known *as Upcasting.*
        - Before performing upcasting inheritance operation takes place.
        - Or the classes should have IS-A relationships.
        - Where properties of superclass are inherited into subclass.
        - Programmers can declare new properties in subclass.
        - At the time of upcasting the properties which are inherited from superclass are only eligible for upcasting. New properties declare are not eligible.
        - **Syntax:** *Superclass refvar = new Subclass();*

- ♦ **DownCasting: (higher to lower):**

  - Assigning super class properties to sub class after upcasting is known as ***Downcasting.***
  - In java downcasting is rarely used because java doesn't support downcasting directly.
  - Before performing *downcasting* compulsory *upcasting* should be performed.
  - We can perform downcasting in *explicit way i.e*. forcefully.
  - So there may be loss of data or information.
  - *In downcasting, he we are trying to hold parent object by child reference variable hence this will gives an exception Called as **ClassCastException.***



Constructor chaining-

- *A constructor is called from another constructor in the same class this process is known as **constructor chaining.***
- It occurs through inheritance.
- When we create an instance of a sub class, all the constructors of the inherited class (super class) are first invoked, after that the constructor of the calling class (sub class) is invoked.
- **Rules for constructor chaining-**
  - An expression that uses this keyword must be the first line of the constructor.
  - Order does not matter in constructor chaining.
  - There must exist at least one constructor that does not use this keyword.
- **How to achieve constructor chaining in two ways:**
- ***Within the same class-***
  - If the constructors belong to the same class, we use this keyword.
- ***From the base class and derived class-***
  - If the constructor belongs to different classes (parent and child classes), we use the super keyword to call the constructor from the base class.

- Program for constructor chaining within same class:

```
public class Chaining {

	Chaining() {
		this(5);
		System.out.println("This is Default constructor");
	}

	Chaining(int x) { //this is parameterized constructor with int
		this("Java");
		System.out.println(x);
	}

	Chaining(String str) { //this is parameterized constructor with
string
		System.out.println(str);
	}

	public static void main(String args[]) {
		Chaining chaining=new Chaining(); //calling default
constructor here
	}

}
```

Output : Java

```
5
This is Default constructor
```

- **Program for constructor chaining within super and sub class:**

```
		public class Base {

		    String name;

		    Base() {
			this("");
			System.out.println("No-argument constructor of base
		class....");
	}

	Base(String name) {
		this.name = name;
		System.out.println("Calling parameterized constructor of base
    class....");
	}
    }
```

```
public class Derived extends Base {

	Derived() {
		System.out.println("No-argument constructor of derived
class");
	}

	Derived(String name) {
		super(name); //calling base class constructor
		System.out.println("Calling parameterized constructor of
derived class");
	}

	public static void main(String args[]) {
		Derived derived = new Derived("test"); //calling Derived
class parameterized constructor
	}
}
```

Output :

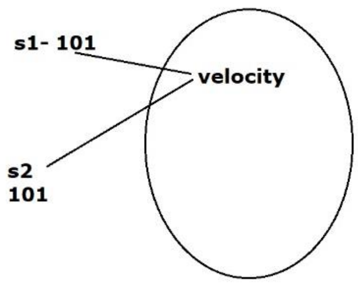
```
	Calling parameterized constructor of base class....
	Calling parameterized constructor of derived class
```

String in Java:

- Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters.
- The **java.lang.String** class is used to create a string object.
- **STRING AS NON-PRIMITIVE DATATYPE-** String data type always starts with upper case letter, and it does not have fixed memory type.
- **STRING AS CLASS-** When we use String as a class then we can create the object of String class.
- String objects are going to be stored inside the String pool area which is present inside the Heap area.
- All String literals in java programs such as "ABC" are implemented as instances (object) of this class.
- There are two ways to create String object:
    - By string literal
    - By new keyword

- **By string literal: (String constant pool area)**

  - Java String literal is created by using double quotes.
  - *String s="velocity";*
  - Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool.
  - *String s1=" velocity ";*
  - *String s2=" velocity ";* //It doesn't create a new instance

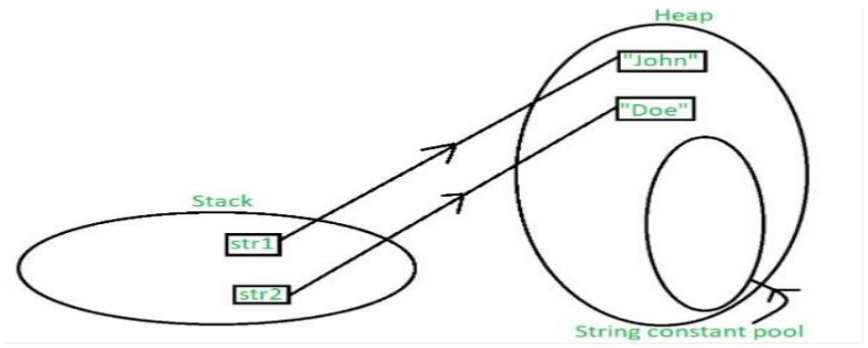


String constant pool

  - In the above example, only one object will be created. Firstly, JVM will not find any string object with the value "Welcome" in string constant pool that is why it will create a new object.
  - “velocity” store in heap memory and reference variable s1 and s2 are in stack memory.
  - After that it will find the string with the value "velocity" in the pool, it will not create a new object but will return the reference to the same instance.
  - **Note:** String objects are stored in a special memory area known **as the "string constant pool".**

  - **Why Java uses the concept of String literal?**
    - To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool)

- **By new keyword: (String non-constant pool area)**
    - If we declare the String with new keyword then the object creation takes place in non-constant pool area/heap area.
    - *String str1= new String ("John");*
    - *String str2 = new String("Doe")*
    - In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "john" and "Doe" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).



- STRING CLASS METHODS USAGE:

1. .isEmpty()

- It gives output as "true" only if the String has no value in it nor even white space such as String s= "";

2. .isBlank()

- It gives output as "true" when the string only has white space or no value in it such as String s= "";Strings= " ";

3. .length();

- To check the length of String.

4. .toUpperCase();

- To convert lower case letter string to upper case letter string.

5. ***.toLowerCase();***
    - To convert UPPER case letter string to LOWER case letter string.

6. ***.startsWith("Hello***");
    - To check whether String starts with provided word.

7. ***.endsWith("Hello");***
    - To check whether String ends with provided word.

8. ***.concat(Hello);***
    - To concat the two strings

9. ***.indexOf('e')***
    - To find the index of provided Character

10. ***.lastIndexOf('h')***
    - To find the last index of provided Character.

11. ***.charAt(2)***
    - To check which character is present on the provided number.

12. ***.substring(5)***
    - To check the String present after provided length number.

13. ***.substring(2,5);***
    - To check the String bet char present at 2 and char present at 5(for example 2& 5);

14. ***s1.equals(s3)***
    - To check whether both the strings are same.

15. ***S2.equalsIgnoreCase(s8)***
    - When two Strings are same but they are different to each other in case sensitivity and ignore that case sensitivity we use this command.

16. ***S8.replace(oldvalue, new value)***
   - To replace the old value with new value.

17. ***.trim();***
   - To remove the initial space of in string.

18. ***.contains();***
   - This method is used to match particular sequence of string into available string and return true if it got matches and false if it does not matches.

19. ***.toCharArray();***
   - This method is used to convert the string into character array.

20. ***.isDigit(char ch);***
   - This method is used to check whether the given character is string is digit or not and accordingly returns true or false.

21. To convert primitive to String:

```
  Int z=12345;
String convertedint =  String.valueOf(z);
System.out.println(convertedint);
System.out.println(convertedint + 4);
```

Output:

```
12345
123454
```

22. To convert String to primitive:

```
String bkn = “true”;
Boolean convertedboolean = Boolean.parseBoolean(bkn);
System.out.println(convertedboolean);

If (convertedboolean)
{
System.out.println(“Boolean value has been converted”);
}

Output:
Boolean value has been converted.
```

23. .split(String regex);

- This method is used to split the string

```
String tobesplit = “Mohan Jagtap”;
  String [] splittedstring = tobesplit.split(“\\S”);
  For (String sp : splittedstring)
{
System.out.println(sp);
}
              Output : Mohan
                       Jagtap

String tosplit =”MaheshShinde”;
String [] splittedString = tosplit.split(“S”);
 For (sk: splittedString)  {
System.out.println(splittedString);
}
              Output: Mahesh
                      Shinde.
```

```
public class Smethods {
public static void main(String[] args) {
        String s1= "";
        String s2= "Bharat Mata Ki jay";
        String s3= "Jai Hind";
        String s4= "Ye dil mange more";
        String s5= "Har Har Mahadev";
        String s7= " ";
        String s8= " BHARAT MATA KI JAY";
        String s6=new String ("Vande mataram");
System.out.println(s1.isEmpty());
System.out.println(s7.isBlank());
System.out.println(s3.length());
System.out.println(s3.toUpperCase());
System.out.println(s3.toLowerCase());
System.out.println(s1.startsWith("Vande"));
System.out.println(s1.endsWith("mataram"));
System.out.println(s1.concat(" Jai Hind"));
System.out.println(s1.indexOf('e'));
System.out.println(s2.lastIndexOf('y'));
System.out.println(s2.charAt(7));
System.out.println(s2.substring(5));
System.out.println(s2.substring(2, 5));
System.out.println(s2.equals(s3));
System.out.println(s2.equalsIgnoreCase(s8));
System.out.println(s8.replace("JAY", "JAYHO"));
System.out.println(s8.trim());
}
```

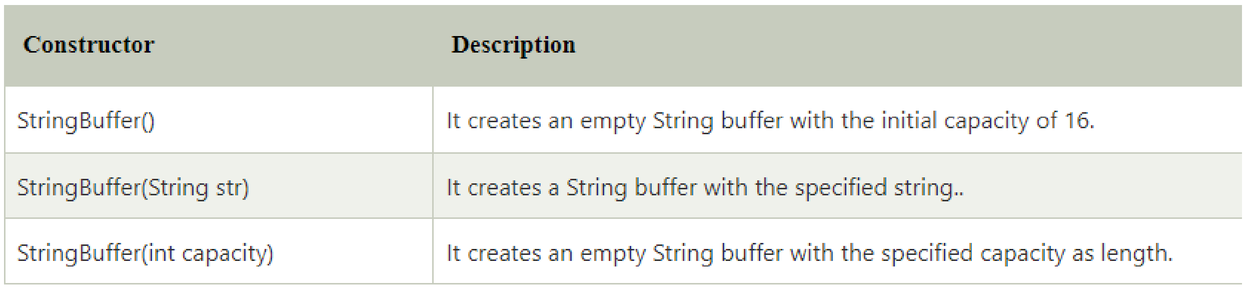
Output :

```
true
true
8
JAI HIND
```

```
jai hind
false
false
Jai Hind
-1
17
M
t Mata Ki jay
ara
false
false
BHARAT MATA KI JAYHO
BHARAT MATA KI JAY
```

Java StringBuffer Class:

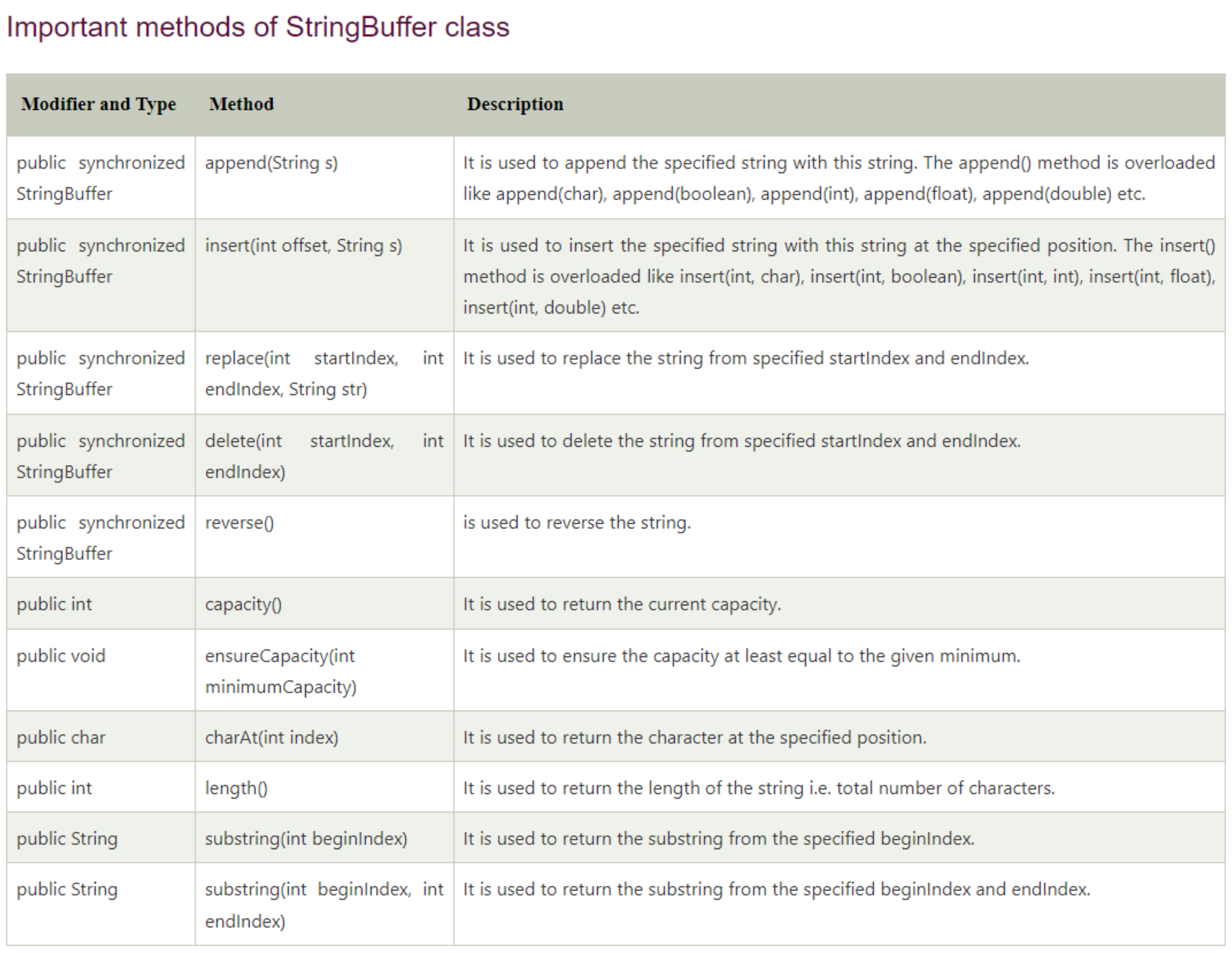
- Java StringBuffer class is used to create mutable (modifiable) String objects.
- The StringBuffer class in Java is the same as String class except it is mutable i.e. it can be changed.
- Note: Java StringBuffer class is thread-safe i.e. multiple threads cannot access it simultaneously. So it is safe and will result in an order.



Constructor	Description
StringBuffer()	It creates an empty String buffer with the initial capacity of 16.
StringBuffer(String str)	It creates a String buffer with the specified string..
StringBuffer(int capacity)	It creates an empty String buffer with the specified capacity as length.

What is a mutable String?

- A String that can be modified or changed is known as mutable String. StringBuffer and StringBuilder classes are used for creating mutable strings.

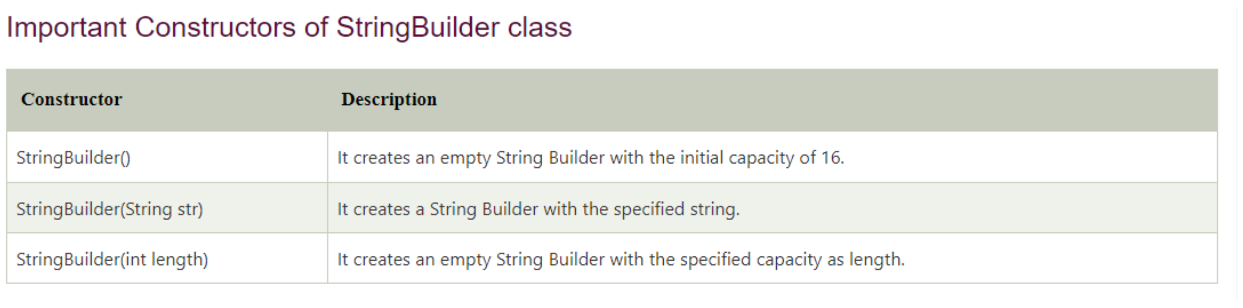


Important methods of StringBuffer class

Modifier and Type	Method	Description
public synchronized StringBuffer	append(String s)	It is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
public synchronized StringBuffer	insert(int offset, String s)	It is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
public synchronized StringBuffer	replace(int startIndex, int endIndex, String str)	It is used to replace the string from specified startIndex and endIndex.
public synchronized StringBuffer	delete(int startIndex, int endIndex)	It is used to delete the string from specified startIndex and endIndex.
public synchronized StringBuffer	reverse()	is used to reverse the string.
public int	capacity()	It is used to return the current capacity.
public void	ensureCapacity(int minimumCapacity)	It is used to ensure the capacity at least equal to the given minimum.
public char	charAt(int index)	It is used to return the character at the specified position.
public int	length()	It is used to return the length of the string i.e. total number of characters.
public String	substring(int beginIndex)	It is used to return the substring from the specified beginIndex.
public String	substring(int beginIndex, int endIndex)	It is used to return the substring from the specified beginIndex and endIndex.

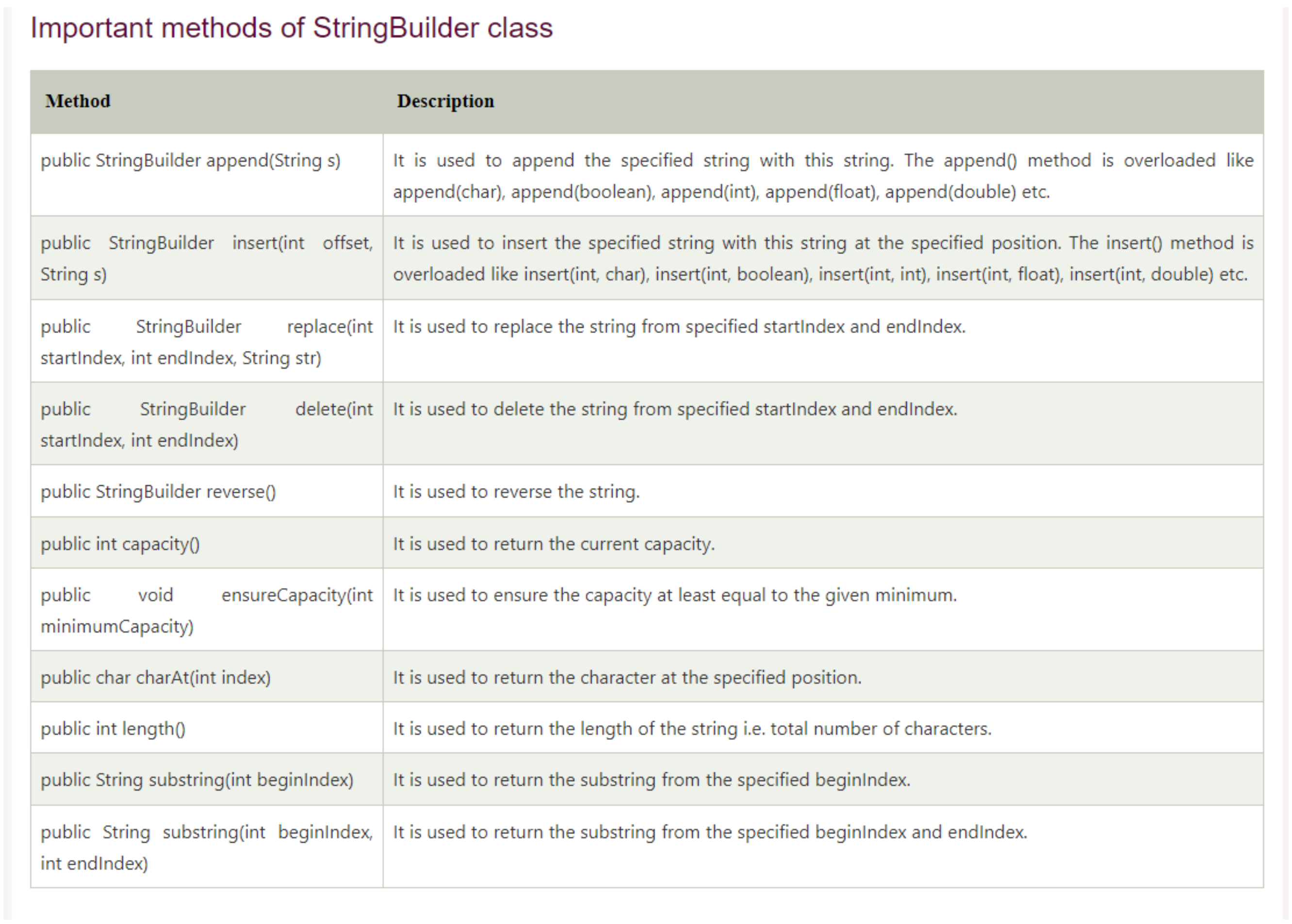
Java StringBuilder Class:

- Java StringBuilder class is used to create mutable (modifiable) String.
- The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK 1.5.



Important Constructors of StringBuilder class

Constructor	Description
StringBuilder()	It creates an empty String Builder with the initial capacity of 16.
StringBuilder(String str)	It creates a String Builder with the specified string.
StringBuilder(int length)	It creates an empty String Builder with the specified capacity as length.



Important methods of StringBuilder class

Method	Description
public StringBuilder append(String s)	It is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
public StringBuilder insert(int offset, String s)	It is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
public StringBuilder replace(int startIndex, int endIndex, String str)	It is used to replace the string from specified startIndex and endIndex.
public StringBuilder delete(int startIndex, int endIndex)	It is used to delete the string from specified startIndex and endIndex.
public StringBuilder reverse()	It is used to reverse the string.
public int capacity()	It is used to return the current capacity.
public void ensureCapacity(int minimumCapacity)	It is used to ensure the capacity at least equal to the given minimum.
public char charAt(int index)	It is used to return the character at the specified position.
public int length()	It is used to return the length of the string i.e. total number of characters.
public String substring(int beginIndex)	It is used to return the substring from the specified beginIndex.
public String substring(int beginIndex, int endIndex)	It is used to return the substring from the specified beginIndex and endIndex.

- **StringBuilder append() method:**
  - The StringBuilder append() method concatenates the given argument with this String.

```
public class Demo {

	public static void main(String[] args) {

		String str = "Demo";
		StringBuilder test = new StringBuilder("Demo");
		System.out.println(test.append("Added"));
	}
}
```

Output: DemoAdded

- **StringBuilder replace() method :**
  - The StringBuilder replace() method replaces the given string from the specified beginIndex and endIndex.

```
public class Demo {

	public static void main(String[] args) {
		StringBuilder sb=new StringBuilder("Hello");
		sb.replace(1,3,"Java");
		System.out.println(sb);
	}
}
```

Output : `HJavalo`

- **StringBuilder insert() method**
  - The StringBuilder insert() method inserts the given string with this string at the given position.

```
public class Demo {

	public static void main(String[] args) {
		StringBuilder sb=new StringBuilder("Hello");
		sb.insert(1,"J");
		System.out.println(sb);
	}
}
```

Output: `HJello`

- **StringBuilder delete() method**
  - The delete() method of StringBuilder class deletes the string from the specified beginIndex to endIndex:

```
public class Demo {
	public static void main(String[] args) {
		StringBuilder sb=new StringBuilder("Hello");
		sb.delete(1, 2);
		System.out.println(sb);
	}
}
```

Output: `Hllo`

- **StringBuilder reverse() method**
  - The reverse() method of StringBuilder class reverses the current string.

```
public class Demo {
	public static void main(String[] args) {
		StringBuilder sb=new StringBuilder("Hello");
		sb.reverse();
		System.out.println(sb);
	}
}
```

Output : olleH

- **StringBuilder capacity() method**
  - The capacity() method of StringBuilder class returns the current capacity of the Builder.
  - The default capacity of the Builder is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2.
  - For example if your current capacity is 16, it will be (16*2)+2=34.

```
public class Demo {
	public static void main(String[] args) {
		StringBuilder sb=new StringBuilder();
		System.out.println("String Builder capacity 
is : " + sb.capacity());
		sb.append("Hello");
		System.out.println("String Builder capacity 
is : " + sb.capacity());
		sb.append(" Java is my favourite language");
		System.out.println("String Builder capacity 
is : " + sb.capacity());
	}
}
```

Output :

```
String Builder capacity is : 16
String Builder capacity is : 16
String Builder capacity is : 35
```

- StringBuilder ensureCapacity() method

  - The ensureCapacity() method of StringBuilder class ensures that the given capacity is the minimum to the current capacity.
  - If it is greater than the current capacity, it increases the capacity by (oldcapacity*2)+2.
  - For example if your current capacity is 16, it will be (16*2)+2=34.

```
public class Demo {
    public static void main(String[] args) {
            StringBuilder sb=new StringBuilder ();
            System.out.println (sb.capacity());       //default 16
            sb.append("Hello");
            System.out.println(sb.capacity());       //now 16
            sb.append ("Java is my favourite language");
            System.out.println(sb.capacity());       //now (16*2)+2=34
i.e (oldcapacity*2)+2
            sb.ensureCapacity(10);                     //now no change
            System.out.println(sb.capacity());    //now 34
            sb.ensureCapacity(50);                   //now (34*2)+2
            System.out.println(sb.capacity());   //now 70
    }
}
```

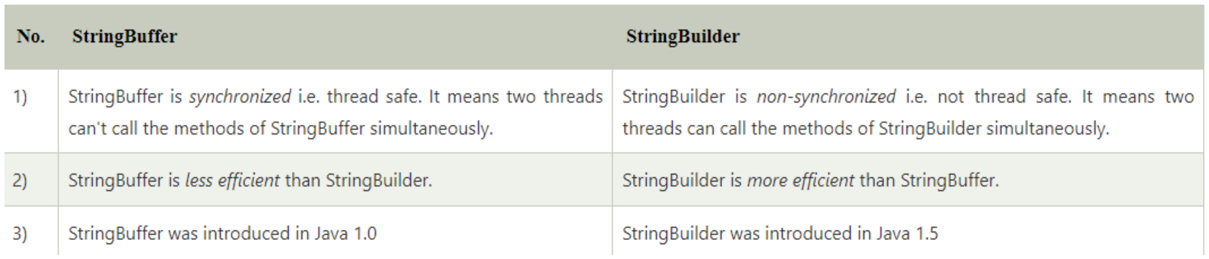
Output:

```
16
16
34
34
70
```

Difference between String and String Buffer :

No.	String	StringBuffer
1)	The String class is immutable.	The StringBuffer class is mutable.
2)	String is slow and consumes more memory when we concatenate too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when we concatenate t strings.
3)	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.
4)	String class is slower while performing concatenation operation.	StringBuffer class is faster while performing concatenation operation.
5)	String class uses String constant pool.	StringBuffer uses Heap memory

Difference between StringBuffer and StringBuilder :



No.	StringBuffer	StringBuilder
1)	StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
2)	StringBuffer is *less efficient* than StringBuilder.	StringBuilder is *more efficient* than StringBuffer.
3)	StringBuffer was introduced in Java 1.0	StringBuilder was introduced in Java 1.5

Java String compare:

There are three ways to compare String in Java:

By Using equals() Method

By Using == Operator

By compareTo() Method

By Using equals() Method:

- The String class equals() method compares the original content of the string.
- It compares values of string for equality. String class provides the following two methods:
- **public boolean equals(Object another)** compares this string to the specified object.
- **public boolean equalsIgnoreCase(String another)** compares this string to another string, ignoring case.

```
public class CompareObject {

   public static void main(String[] args) {
            String s1="Sachin";
            String s2="Sachin";
            String s3=new String("Sachin");
            String s4="Saurav";
            System.out.println(s1.equals(s2));    //true
            System.out.println(s1.equals(s3));    //true
            System.out.println(s1.equals(s4));    //false
   }
}
```

Output :

```
true
true
false
```

- **By Using == operator:**
  - **The == operator compares references not values.**

```
public class CompareObject {

   public static void main(String[] args) {
          String s1="Sachin";
          String s2="Sachin";
          String s3=new String("Sachin");
          System.out.println(s1==s2);     //true (because both
refer to same instance)
          System.out.println(s1==s3);   //false (because s3
refers to instance created in non pool)
   }
}
```

Output :

```
true
false
```

- **By Using compareTo() method:**
  - The String class compareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.
  - Suppose s1 and s2 are two String objects. If:

    s1 == s2 : The method returns 0.

    s1 > s2 : The method returns a positive value.

    s1 < s2 : The method returns a negative value.
  - **It compares value of ASCII code for small letter (a-z 97-122 ,A-Z 65-90)**

```
public class CompareObject {

       public static void main(String[] args) {
                 String s1="Sachin";
                 String s2="Sachin";
                 String s3="Ratan";
                 System.out.println(s1.compareTo(s2));     //0
                 System.out.println(s1.compareTo(s3));    //1(because s1>s3)
                 System.out.println(s3.compareTo(s1));   //-1(because s3 < s1 )
       }
}
```

Output:

```
0
1
-1
```

Questions on Java :

1. **Can we keep main method as private?**
   - No, we cannot keep main method as private because private fields cannot be accessed from outside of class.
   - And JVM needs to access main method from outside of class.
   - If we keep main method as private, program will compile but it will not be executed.
   - We will get Error as Main Method not found in given class.

2. **Can we keep main method as non-static?**
   - No, we can't write main method as non-static because if method is non static, we have to create an object and JVM can't create object on its own to access main method.
   - If we keep main method as non-static program will compile but it will not be executed.

3. **What if local variable and global variable names are same?**
   - If local variable and global variable names are same first priority is always given to local variables because they are inside method and nearer for execution to JVM.

4. **Is initialization of global variable mandatory?**
   - They are not mandatory to initialize, if we did not initialize, JVM will take default values based on data types.

     byte,short,int and long --------> 0
     float and double ------------> 0.0
     String ---------------------> null
     boolean ---------------------> false
     char ---------------------> empty space

   - Global variables can be public, default, private, protected and final
   - Global variables are also called as Data members.

5. **Can we change the syntax of main method?**
   - JVM is responsible to execute every java program.
   - JVM // the identity of jvm is public static...., if we change that then jvm will tell identity is not found

     {
     public static void main (command line args)
     }

   - Execution of every java program will always start from main method
   - Because JVM only knows main method
   - If main method is not there or main method syntax is changed our program will not be executed.
   - So, the answer is No we cannot change syntax of main method.

6. **What if we skip break keyword in case?**
   - If we skip break keyword, we will not get compile error.
   - If there is no break keyword in case statement JVM will execute all the cases irrespective of condition until it finds next break statement.

7. **What if we skip break keyword in default?**
   - It does not matter because after default nothing is there to execute. So, writing of break keyword is not mandatory in default statement.

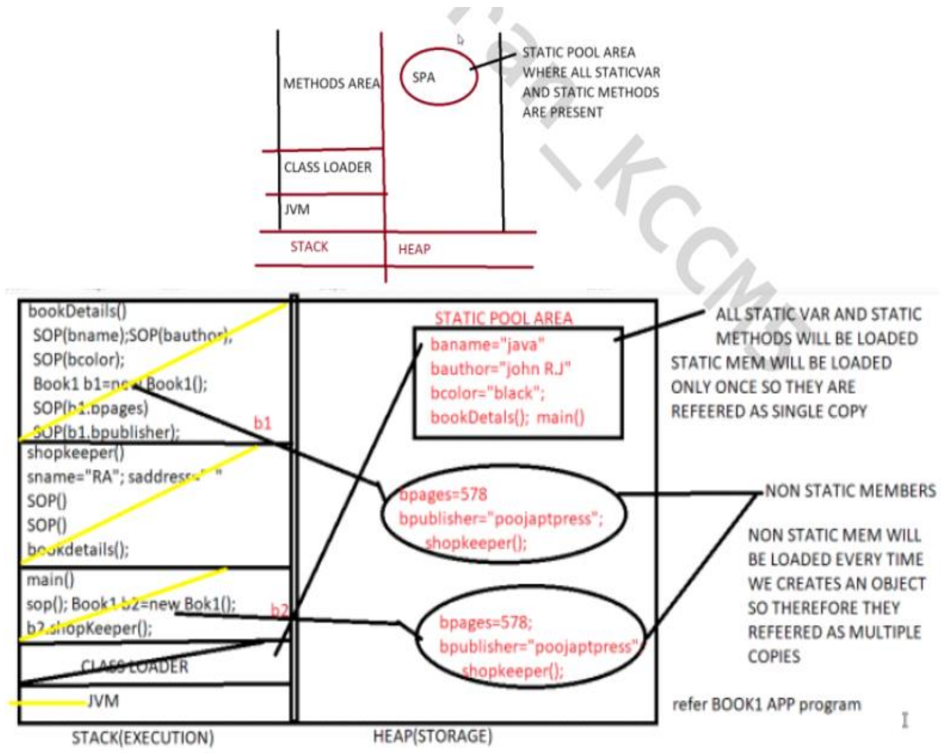
8. **Can we overload final methods?**
   - Yes, we can overload final methods because final keyword says do not change implementation and in overloading, we are not changing implementation rather we are changing arguments.

9. **Can we overload private methods or not?**
   - Yes, we can overload private methods because private methods are accessible everywhere in same class and overloading also happens within class.

10. **Can we overload non-static methods or not?**
    - Yes, we can overload non-static methods but to call them we have to create an Object.

11. **Explain the executions process or memory management in detail?**
    - Whenever we execute any program there will be two memory blocks that gets created.
      - stack area also called as Execution Area
      - Heap area also called as Storage Area

    - **Important points:**
      - First JVM will enter into stack area by making a call to class Loader.
      - Class loader is like a function (program) whose job is to load all **static** members into **static pool area (SPA).**
      - SPA is a part of heap area.
      - Once class loader loads all static members into SPA its job is done, so it will come out of **stack area.**
      - Next JVM will make a call to main method.
      - Upon call to any method, method will get loaded in stack area under method area.
      - Next Execution of main method will start.
      - Once entire execution is completed main () also come out of stack area.
      - Next JVM before coming out of stack area it will make a call to garbage collector whose job is to cleanup entire memory for next execution.



- **Conclusion**
  - Static methods and static variables are available (SPA) before execution starts, that’s why they are accessible without object creation.
  - Non static members are not available before execution so to access them we have to create an object. With new keyword object will get created and loads all non-static members inside it.
  - Non static variables and non-static methods, they are loaded together in object so that’s why since they belongs to same area non-static variable can be accessed directly in non-static method.
  - Static members will get loaded only once in SPA that’s why they are referred as single copy.
  - Since Non static members will get loaded whenever we created an object that’s why they referred as multiple copies.
  - LOCAL variables are present in stack area and GLOBAL variables are present in heap area.
  - Static variables are present in SPA (Heap).
  - Non static variables are present in object (Heap).
  - All reference variables are local variables.
  - Since once execution completes, method will come out of stack area that is the reason local variable scope is within method because they are present inside method.

Difference Between Static and Non-Static:

Static	Non static
1.static means single copy.	1.Non static means multiple copies.
2.static members are present in static pool area.	2.Non static members are present in object.
3.static members can be accessed in 3 ways 1.Directly 2.Through class name 3.Through object reference	3.Non static members can be accessed through object.
4.static members will get loaded only once in SPA.	4.Non static members will get loaded whenever we created an object.
5.class loader is responsible to load static members.	5.new keyword is responsible to load all non static members.
6.Any changes made to static variables will effect entire class that is why static variables are also called as class variables.	6.Any changes made to non static variable will effect only a particular object that why they are called as instance variables.
7.We go for static, if the data is fixed. Static String clg="Qspiders";	7.We go for non static, if the data is changing. String courses="CSE";
8.static methods cannot be inherited.	8.Non static methods can be inherited.
9.static methods cannot be overridden.	9.Non static methods can be overridden.

12. Can I call more than one constructor from constructor?

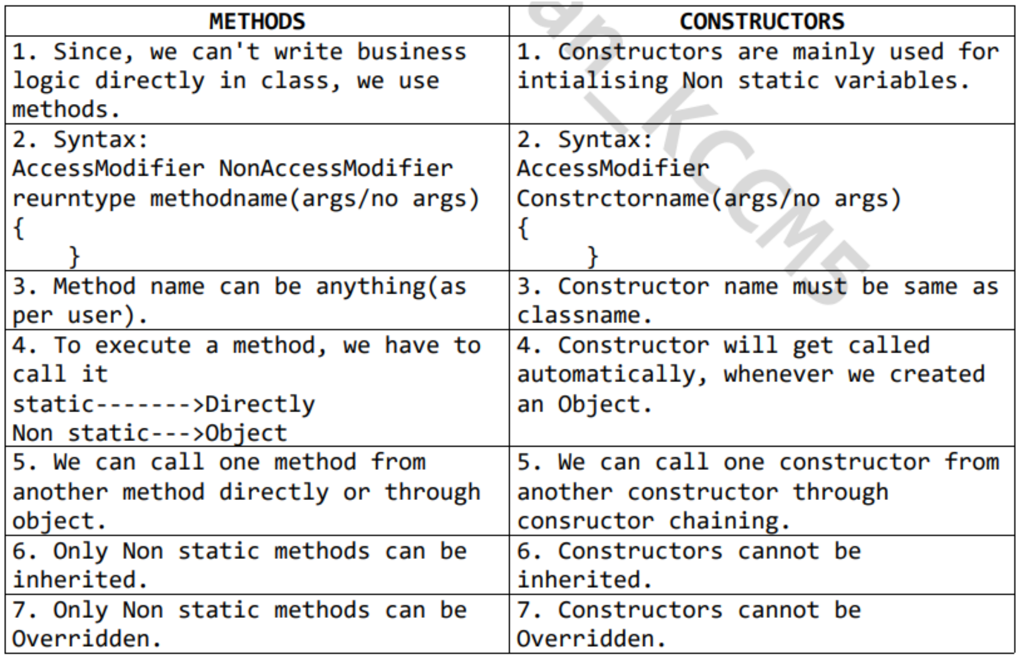
- No, we cannot call more than one constructor from constructor because.
- If we call we have to write call to this in 2nd statement which is violation of rule.

```
  public Add()
  {
this(100,200);       //call another const of same class which has 2
integer args
this(223.4,50);      //CTE because call to this must be first statement
System.out.println ("Default add constructor");
  }
```

Difference Between class and Object:

class	Object
A class is logical entity.	An object is physical entity.
class represents logic.	Object represents state and behaviour.
class can be created using class keyword.	Object can be created using new keyword.
When class is created, memory will not be allocated.	When object gets created memory will be allocated(Heap).
For each module, we can create one class.	For one class, we can create multiple Objects.

Difference between Methods and Constructors:



METHODS	CONSTRUCTORS
1. Since, we can't write business logic directly in class, we use methods.	1. Constructors are mainly used for intialising Non static variables.

2. Syntax:
AccessModifier NonAccessModifier reurntype methodname(args/no args)
{
}
	2. Syntax:
	AccessModifier
	Constrctorname(args/no args)
	{
	}

3. Method name can be anything(as per user).	3. Constructor name must be same as classname.
4. To execute a method, we have to call it static------->Directly Non static--->Object	4. Constructor will get called automatically, whenever we created an Object.
5. We can call one method from another method directly or through object.	5. We can call one constructor from another constructor through consructor chaining.
6. Only Non static methods can be inherited.	6. Constructors cannot be inherited.
7. Only Non static methods can be Overridden.	7. Constructors cannot be Overridden.

13. Can we inherit Constructors or not?

- No we cannot inherit constructors because they are not member of a class (members of class are methods and variables) and constructors are mainly used for initialization of Non-static variable.
- Constructors cannot be inherited but they can be invoked by using call to super.

14. Why do we go for Overriding??

- When subclass want the properties of super class but does not want the implementation of it. In such case, sub class can change implementation by means of Overriding.
- **Ex: Caller's tune**

  Normaldays---->tone()---->NormalRingtone

  DuringMarch--->tone()---->CoronaRingtone
- **Rules For Overriding**
  - Inheritance is compulsory.
  - method signature must be same (methodname and arguments) as super class in sub class.
  - method header must be same (access modifier, non-accessmodifier, return type)
  - Overridden method should not be final.

15. Can we create object of abstract class?

- No, we cannot create an object of abstract class because it contains abstract methods and abstract method does not have any body to execute.

16. Can we Inherit abstract classes or not?

- Yes, if any child classes are there for abstract class that child class has to undergo with any of the one rule.
- Complete all incomplete methods of abstract class by means of overriding.
- Declare your class also as abstract class.

17. Can we define Constructor in abstract class?

- Yes, we can define a constructor in abstract class.

18. Can we declare abstract class as final?

- No, we cannot declare an abstract class as final because if it is final it cannot be extended or inherited.

19. Can we declare abstract method as final?

- No, we cannot declare an abstract method as final because if it is final, it cannot be overridden and we must have to override abstract method to complete it.

20. Can we declare abstract method as static?

- No, we cannot declare an abstract method as static because if it is static, it cannot be overridden and we must have to override abstract method to complete it.

21. Can we achieve Multiple Inheritance through abstract class?

- No, we cannot achieve Multiple Inheritance through abstract class because one class cannot extend more than one abstract class.

22. Can we create an object of interface?

- No, we cannot create an object of interface because all methods are by default abstract. But we can create a reference of interface.

23. Multiple Inheritance through Interface?

- Through classes it is not possible because of:-
    - Ambiguity problem
    - Constructor chaining problem
    - diamond problem

24. Can a constructor be final?

- No, a constructor cannot be final because final keyword is applicable only for class, methods and final not with constructors.

25. Will final methods get inherited?

- Yes, final methods will get inherited but only thing is they cannot be overridden.

26. Why multiple inheritance is possible through interface?

- A class cannot extend more than one class but it can implements multiple interfaces.
- In interface there is no possibility of Ambiguity problem because even though multiple interfaces contains same method name but implementation will be given only once because in interface class gives implementation and it won't implement multiple methods with same name and same argument in single class

27. Explain about final keyword?

- Final is a keyword which basically indicates that there is no more changes are allowed.
- **Example :**
  - final match
  - final result
  - final destination
- **final keyword is applicable with:**
  - methods
  - variables(Local and Global)
  - Class

- **final with variable:**
  - if a variable is declared as "final", we cannot reinitialize the value.
  - Ex: final int score=100;
  - score=300;//Re-initialization-->not allowed bcoz score is final possible combination of final variable
  - final static int i=100;//final with static variable
  - final String m="fail";//final with non-static variable
  - final char gender='M';//final with local variable

- **final with class**
  - If a class is declared as final, we cannot inherit or extends that class. So, therefore a super class can never be final.
  - But a final class can extends Non final class, i.e, subclass can be final but super class cannot be final.

- **final with methods**
  - If a method is declared as final we cannot override it.
  - final keyword says that do not make further changes and in overriding we are changing the implementation. Therefore, final methods cannot be overridden.
  - It is mandatory to initialize final variable while declaration, if we did not initialize we will get compile time error.

28.Difference Between static and final :

static	final
static means single copy.	final means fixed copy.
it is not mandatory to intialise static while declaration.	it is mandatory to intialise final while declaration.
static variable can be reintialised.	final variable cannot be reintialised.
A class cannot be static.	A class can be final.
static keyword is applicable with method, variables, blocks and nested class.	final is applicable with methods, variables and classes.
Ex:static String clg="Qspiders";	Ex:final float pie=3.414f;

29. What is mean by protected modifier?

- Protected fields can be accessible anywhere within class.
- Protected fields can be accessible within another class but that class should belongs to same package.
- Protected fields can be accessible within another class of another package but only after inheritance.
- A class can never be protected.

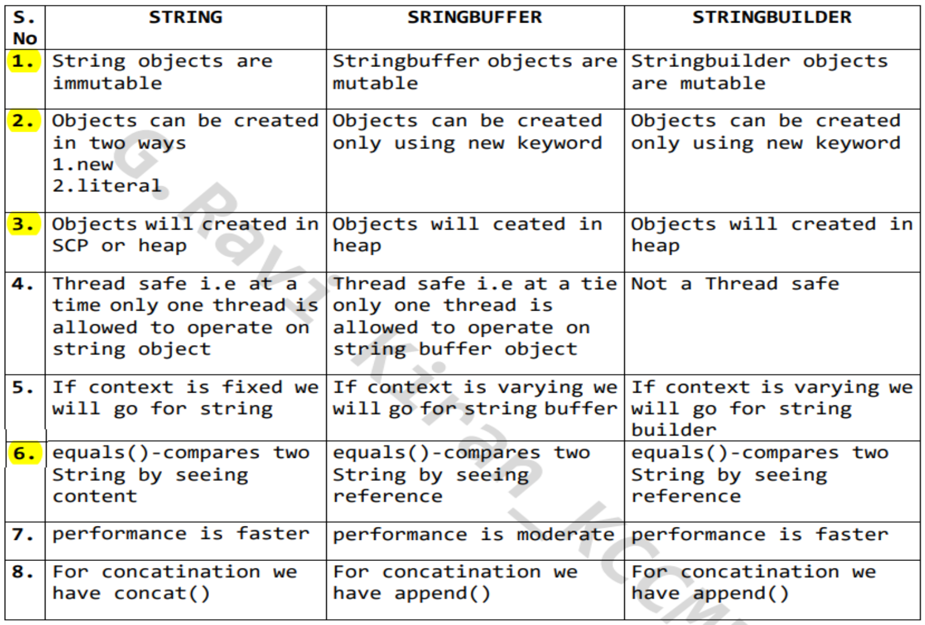
30.Difference Between Error and Exception :

Error	Exception
1. An error is caused due to lack of system resources.	1. An exception is caused because of some problem in code.
2. An error is irrecoverable i.e, an error is a critical condition cannot be handled by code of program.	2. An Exception is recoverable i.e, we can have some alternate code to handle exception.
3. There is no ways to handle error.	3. We can handle exception by means of try and catch block.
4. As error is detected program is terminated abnormally.	4. As Exception is occurred it can be thrown and caught by catch block.
5. There is no classification for Error.	5. Exceptions are classified as checked and unchecked.
6. Errors are define in java.lang.Error package	6. Exceptions are define in java.lang.Exception package

31. Does constructor return any value?

- No, constructor does not return any value. While declaring a constructor you will not have anything like return type.
- In general, Constructor is implicitly called at the time of instantiation. And it is not a method, its sole purpose is to initialize the instance variables

32.Difference Between String ,StringBuffer and StringBuilder:



S. No	STRING	SRINGBUFFER	STRINGBUILDER
1.	String objects are immutable	Stringbuffer objects are mutable	Stringbuilder objects are mutable
2.	Objects can be created in two ways 1.new 2.literal	Objects can be created only using new keyword	Objects can be created only using new keyword
3.	Objects will created in SCP or heap	Objects will ceated in heap	Objects will created in heap
4.	Thread safe i.e at a time only one thread is allowed to operate on string object	Thread safe i.e at a tie only one thread is allowed to operate on string buffer object	Not a Thread safe
5.	If context is fixed we will go for string	If context is varying we will go for string buffer	If context is varying we will go for string builder
6.	equals()-compares two String by seeing content	equals()-compares two String by seeing reference	equals()-compares two String by seeing reference
7.	performance is faster	performance is moderate	performance is faster
8.	For concatination we have concat()	For concatination we have append()	For concatination we have append()

33. Can we declare a constructor as final?

- No, a constructor can't be made final. A final method cannot be overridden by any subclasses.
- The final modifier prevents a method from being modified in a subclass.
- The main intention of making a method final would be that the content of the method should not be changed by any outsider.

34. Can we declare an interface as final?

- Interface method cannot be final. Cannot be declared final.

ABSTRACT CLASS	FINAL CLASS
Uses the “abstract” key word.	Uses the “final” key word.
This helps to achieve abstraction.	This helps to restrict other classes from accessing its properties and methods.
Can be inherited	Cannot be inherited
Cannot be instantiated	Can be instantiated
A few methods can be implemented and a few cannot	All methods should have implementation
Abstract class methods functionality can be altered in subclass	Final class methods should be used as it is by other classes
For later use, all the abstract methods should be overridden	Overriding concept does not arise as final class cannot be inherited

35. Why is Java not a pure object-oriented language?

- Java supports primitive data types - byte, Boolean, char, short, int, float, long, and double and hence it is not a pure object-oriented language.

36. Difference between Heap and Stack Memory in java. And how java utilizes this?

- Stack memory is the portion of memory that was assigned to every individual program. And it was fixed.
- On the other hand, Heap memory is the portion that was not allocated to the java program but it will be available for use by the java program when it is required, mostly during the runtime of the program.
- **Java Utilizes this memory as -**
- When we write a java program then all the variables, methods, etc. are stored in the stack memory.
- And when we create any object in the java program then that object was created in the heap memory. And it was referenced from the stack memory.

37. What is a ClassLoader?

- Java Classloader is the program that belongs to JRE (Java Runtime Environment).
- The task of ClassLoader is to load the required classes and interfaces to the JVM when required.
- Example- To get input from the console, we require the scanner class. And the Scanner class is loaded by the ClassLoader.

38. Can we override the static methods?

- No, we cannot override static methods because method overriding is based on dynamic binding at runtime and the static methods are bonded using static binding at compile time. So, we cannot override static methods.
- We can’t override a static method as non-static or either way after doing the same we will get compile time error.
- **Reason:** To access static we don’t need an object and for non-static we always need to have an object.
- **Method hiding:** If two static methods try to get override then it is not overriding but it is method hiding. That means method resolution is based on reference type but not run time object.

39. Can you use abstract and final both with a method?

- Similarly, you cannot override final methods in Java. But, in-case of abstract, you must override an abstract method to use it.
- Therefore, you cannot use abstract and final together before a method.

40. Can there be any abstract method without an abstract class?

- Yes, you can declare abstract class without defining an abstract method in it.
- Once you declare a class abstract it indicates that the class is incomplete and, you cannot instantiate it.

41. **Difference between Static Method and Non-Static methods:**
- Static method is contains static keyword and it is static in nature.
- Static method has fixed memory in JVM.
- We can call static method in main method by directly the name of static method with method signature followed by semicolon.
- *Non-static method does not static keyword and it is dynamic in nature.*
- *We cannot call non-static method directly in main method. We have create an object. To call non-static method .Then* ***object dot with non-static method name with method signature followed by semicolon.***
- *This method does not have any fixed memory in JVM.*
- As we know about the Static method as well as non-static method we can declare multiple times. As compared to static method, non-static method is having lengthy process. Still java recommended to use Non-static method why?
- Because if I talk about security, security is less for static method as compared to non-static method. It easily accessible in main method. & for non-static method we cannot easily access it. For accessing the non-static method we must have the knowledge about the object.
- If we want our code is more secure so we need to use non-static method to write a program.
- We can customize all code into non-static method but we cannot customize all code into static because it is static in nature.
- That’s all about the static and non-static methods.

42. Is it possible that the ‘finally’ block will not be executed? If yes then list the case.
- Yes. It is possible that the ‘finally’ block will not be executed. The cases are
- Suppose we use ***System.exit()*** in the above statement.
- If there are fatal errors like Stack overflow, Memory access error, etc.

43. What is the main objective of garbage collection..?
- The main objective of this process is to free up the memory space occupied by the unnecessary and unreachable objects during the Java program execution by deleting those unreachable objects.
- This ensures that the memory resource is used efficiently, but it provides no guarantee that there would be sufficient memory for the program execution.

44. Can you call a constructor of a class inside another constructor?
- Yes, the concept can be termed as constructor chaining and can be achieved using this() .

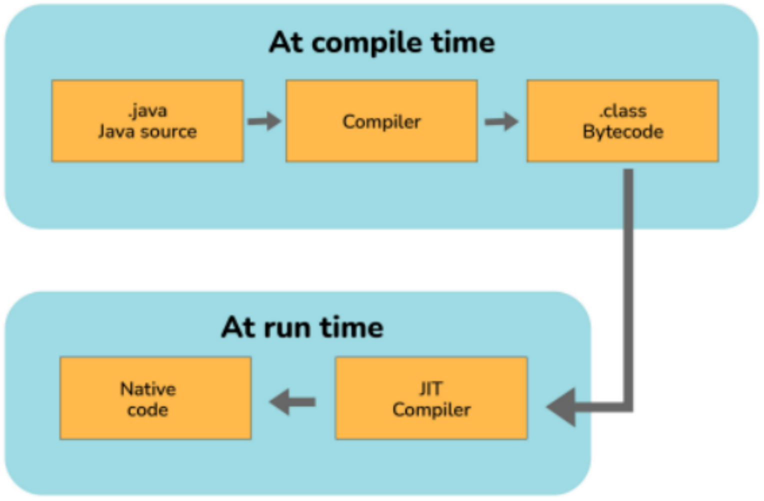
45. How is an infinite loop declared in Java?

- Infinite loops are those loops that run infinitely without any breaking conditions. Some examples of consciously declaring infinite loop is:

  Using For Loop:
  Using while loop:
  Using do while loop:

46. Tell us something about JIT compiler?

- JIT stands for Just-In-Time and it is used for improving the performance during run time. It does the task of compiling parts of byte code having similar functionality at the same time thereby reducing the amount of compilation time for the code to run.
- The compiler is nothing but a translator of source code to machine-executable code.
- But what is special about the JIT compiler? Let us see how it works:
  - First, the Java source code (.java) conversion to byte code (.class) occurs with the help of the **javac** compiler.
  - Then, the .class files are loaded at run time by JVM and with the help of an interpreter, these are converted to machine understandable code.
  - JIT compiler is a part of JVM. When the JIT compiler is enabled, the JVM analyzes the method calls in the .class files and compiles them to get more efficient and native code. It also ensures that the prioritized method calls are optimized.
  - Once the above step is done, the JVM executes the optimized code directly instead of interpreting the code again. This increases the performance and speed of the execution.


At compile time
.java
Java source
Compiler
.class
Bytecode
At run time
Native
code
JIT
Compiler