1) JDK, JRE, and JVM (Java Virtual Machine)

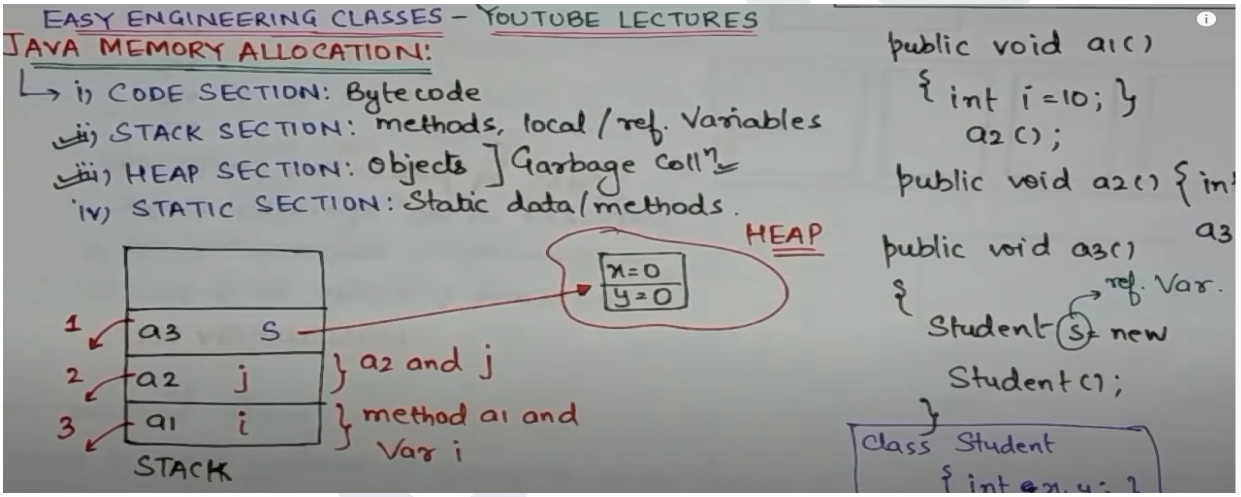
JDK - jdk stands for java development kit which internally contains JRE that is a java runtime environment and JVM that is a java virtual machine.
Jdk provides all the tools and libraries to work with the java language.

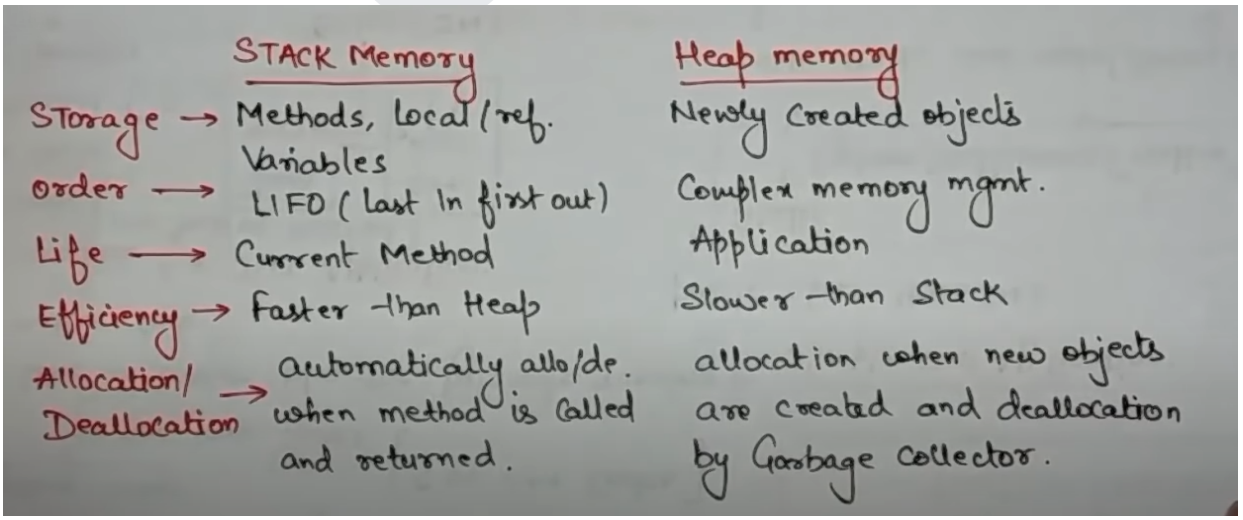
JRE- JRE is a java runtime environment which provides the environment to execute the byte code . it's internally provide us JVM which is responsible to execute a java program,

JVM - JVM stands for java virtual machine , it is software in the form of an interpreter written in c through which we can execute a java program.
It interprets the .class file and gives us an output.

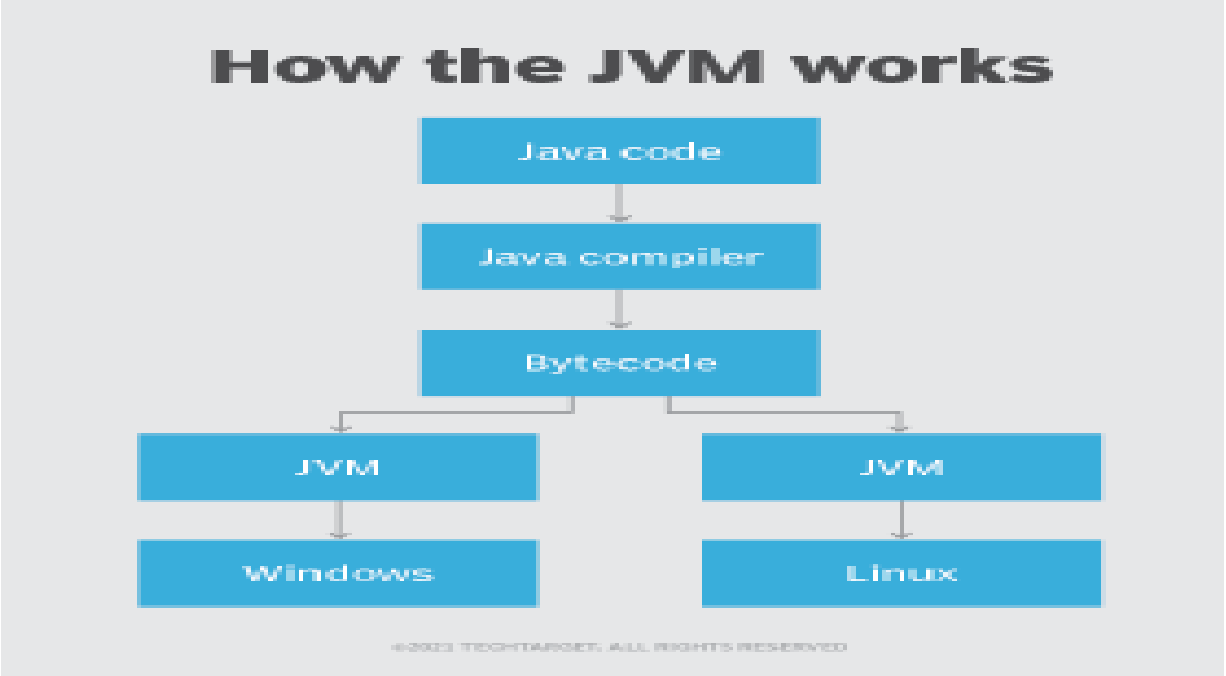
JDK = JRE+JVM
JRE=JVM+Library

2) Java Memory Allocation





3) Jvm Working - Machine Independent



4) Unicode System

In every language, different letters are present and the code assigned to every letter is also different which means multiple languages have multiple codes for various letters.Certain languages have many character sets, the code assigned to each character may vary in terms of length. For example, some characters can be encoded with a single byte, others might require two or more bytes.

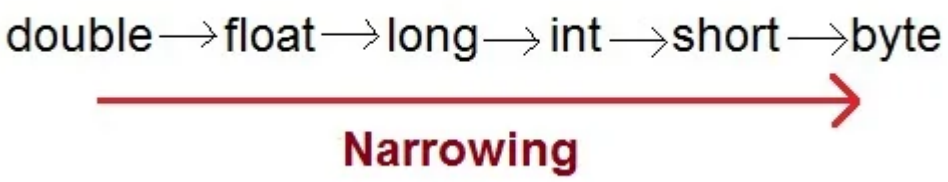
These problems led to finding a better solution for character encoding that is the Unicode System.

- The Unicode system is an international character encoding technique that can represent most of the languages around the world.
- Hexadecimal values are used to represent Unicode characters.
- There are multiple Unicode Transformation Formats:
    1. UTF-8: It represents 8-bits (1 byte) long character encoding.
    2. UTF-16: It represents 16-bits (2 bytes) long character encoding
    3. UTF-32: It represents 32-bits (4 bytes) long character encoding.

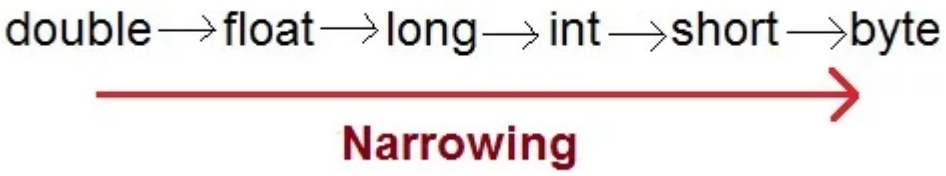
5) **Class , Object -** class of student

6) **Type Casting -**

Widening Casting(Implicit)

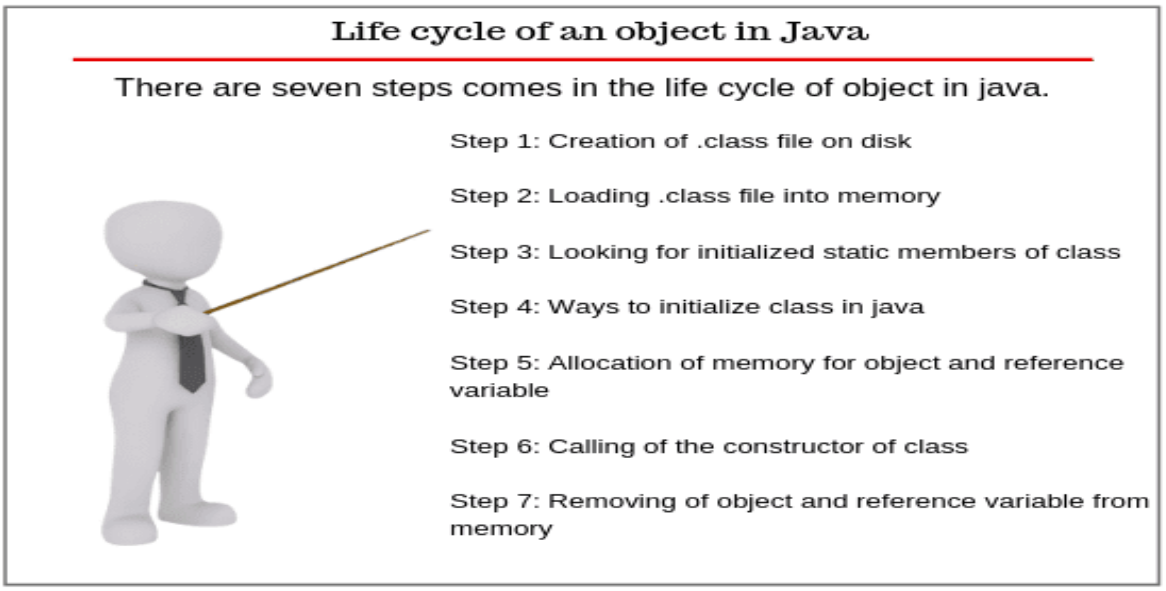


Narrowing Casting(Explicitly done)



7) change one data to another
Automatically conversion by jvm - implicit type casting
Int i = 50
Double d = i ;
o/p - 50 , 50.0
***
Double i = 50.0;
Int j = i;
Compile time error
Cant converter automatic
Forcefully by Programmer - explicit type casting - data loss
Double i = 50.0
Int j = (**int**) i;
o/p- 50

8) **Why java is partial object oriented -** 7 primitive data type
9) **Life cycle of an object**



10) Anonymous object in java

Anonymous object in Java means creating an object without any reference variable. Generally, when creating an object in Java, you need to assign a name to the object. But the anonymous object in Java allows you to create an object without any name assigned to that object.

So, if you want to create only one object in a class, then the anonymous object would be a good approach.

Syntax- new class_name();

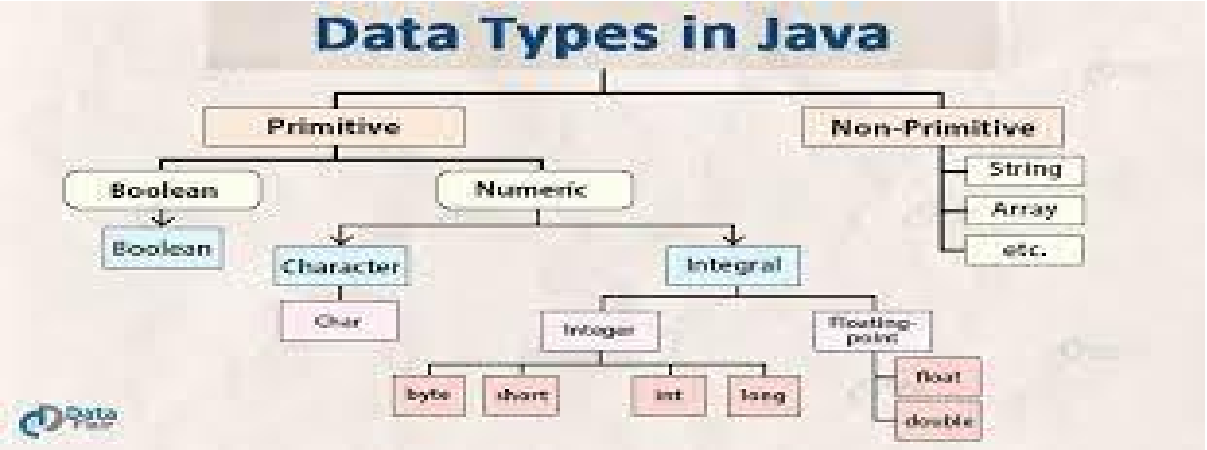
Eg. new Student("chail");

11) Packages in java

A package in Java is used to group related classes. Think of it as a folder in a file directory. We use packages to avoid name conflicts, and to write a better maintainable code. Packages are divided into two categories:

- Built-in Packages (packages from the Java API)
- User-defined Packages (create your own packages)
-

12) Data Type in java



13) Variable in java

A variable is the name of a reserved area allocated in memory.

1) Local Variable - A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.A local variable cannot be defined with "static" keyword.

2) Instance Variable- A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as static . 1) Local VariableIt is called an instance variable because its value is instance-specific and is not shared among instances.

3) Static variable

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

```
1.  public class A
2.  {
3.    static int m=100;//static variable
4.    void method()
5.    {
6.      int n=90;//local variable
7.    }
8.    public static void main(String args[])
9.    {
10.     int data=50;//instance variable
11.   }
12. }//end of class
```

14) Method vs function in java

Function set of instruction to perform specific task

Method is set of instruction that are associated with object

Function does not need any object while method linked with object

We can call method with its name but method called by object name

Function used to pass or return data whole method operated the data in class

Function is independent functionality while method lies under oops

Function can only work with provided data while method can access all data provided in the given class

15) Static and final keyword in java

Final keyword → you can't change the value of a variable once defined.
constructor called once then initialise the object and final also initialise once so we can directly give final value once

Static keyword - we make different objects in these objects. I want to count how many students have made it till now .
I can take variable num student variable and every time when we make an object there should be num student variable
It is needed that every object have this variable e
Num student is not property of object we are just counting the object
In this scenario we take static
We just take one copy of numstudent variable
Static keyword now take space once and every object take this copy and i will increase value by 1

16) Constructor in java

- A constructor in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes.

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

Java constructor overloading -The process of defining multiple constructors of the same class is referred to as Constructor overloading. However, each constructor should have a different signature or input parameters. In other words, constructor overloading in Java is a technique that enables a single class to have more than one constructor that varies by the list of arguments passed. Each overloaded constructor is used to perform different task in the class**.**

17) Access modifier in java -

Access modifier -

1) Default - if we write nothing considered as default , access within the package
2) Public - access everywhere
3) Private - only can be access within the class only
4) Protected -

Getter and setter →

I have made the property private so i can't access outside the class but i want to access this value and want to change here comes the getter and setter

Why make private
If someone makes a object of class and set the enrol number -ve , this should not be allowed that's why we make private and i can give the value using a function

Java has member variable and member methods

```
Public void setEnroll(int enrol){
This.enroll = enrol
}
```

18) This keyword in java -

we have enrol class property and we are passing same name in set enrol
Now I want to change the local var. From set enrol value but java compile confused about the same name and set value change it's value itself so we use this keyword
This keyword point the class variable

What if we print object

In array we have stack and heap memory
Address stored in stack and whole object stored in heap memory
When we try to print object we only get object
Packagename.classname@address
If we print this then we get object
Because this keyword is local pointer which pointing to the object

Using this we can call function and also constructor as well using inheritance
Inner class can use private variable of outer class

19) Garbage in java - we don;t have garbage in java

There are some fix value
For int =0
String = null
boolean = false
double=0.0

20) Static keyword

In total there are 57 keywords in Java. One among them is "Static".In Java, the static keyword is mainly used for memory management. It can be used with variables, methods, blocks and nested classes. It is a keyword which is used to share the same variable or method of a given class. Basically, static is used for a constant variable or a method that is the same for every instance of a class. The main method of a class is generally labelled static.

When a member of the class is declared as static, it can be accessed before the objects of its class are created, and without any object reference.

static keyword is a non-access modifier and can be used for the following:

Static block , variable , method , class

Static Block

If you need to do the computation in order to initialise your static variables, you can declare a static block that gets executed exactly once, when the class is first loaded.

Static Variable

When you declare a variable as static, then a single copy of the variable is created and divided among all objects at the class level. Static variables are, essentially, global variables. Basically, all the instances of the class share the same static variable. Static variables can be created at class-level only.

Static Method

When a method is declared with the *static* keyword, it is known as a static method. The most common example of a static method is the *main( )* method. Methods declared as static can have the following restrictions:

- **They can directly call other static methods only.**
- **They can access static data directly.**

Static Class

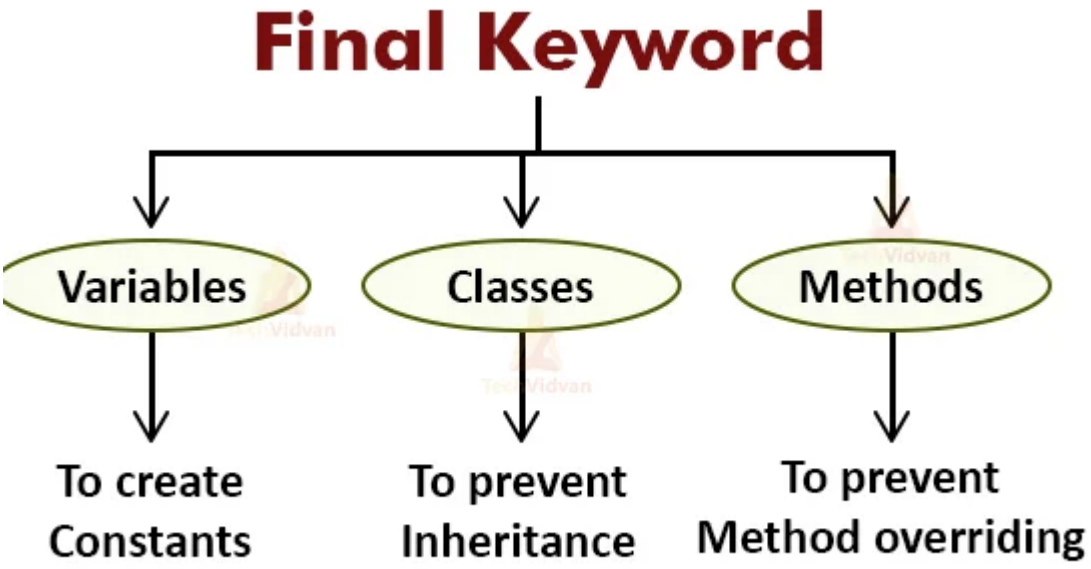
A class can be made static only if it is a nested class. Nested static class doesn't need a reference of Outer class. In this case, a static class cannot access non-static members of the Outer class.

21) final keyword -

Java final keyword is a non-access specifier that is used to restrict a class, variable, and method. If we initialise a variable with the final keyword, then we cannot modify its value.

If we declare a method as final, then it cannot be overridden by any subclasses. And, if we declare a class as final, we restrict the other classes to inherit or extend it.

In other words, the final classes can not be inherited by other classes.



final variable

Syntax of defining a final variable:

final int number = 10; //final variable

final float value; //blank final variable

static final double rate = 2.5; //final and static variable

we must use the final variables only for the assigning values that we want to remain constant throughout the execution of the program.

final method

The Method with Final Keyword cannot be overridden in the subclasses. The purpose of the Final Method is to declare methods of how's definition can not be changed by a child or subclass that extends it.

Suppose we make a class Animal and declare a non-final method sound() in it. Someone creates a Dog class and overrides the sound() method in it and prints the wrong sound of the Dog such as Meow or Roar, etc.

Final class

Java final class can't be extended by other classes in the inheritance. If another class attempts to extend the final class, then there will be a compilation error.

22) static keyword -

Java Static Variable

The purpose of the static method is when we need some piece of code that has to execute every time then we declare it as Static.
the static variables are also called the class variables
The variables declared with the static keyword refer to the common property for all the objects of the class.

For example, the university name is the same for its students, employees, faculties, etc. In other words, we can say that only a single copy of the static variable is there which is shared among all the instances of the class.

We can access the static variables directly from static as well as non-static methods.

Java Static Block

We know that a block is nothing but the lines of code enclosed within curly braces. If we declare any block as static, then such blocks are called static blocks in Java.

We use the static block when we need to initialize the static variables. The static blocks load or execute at the time of loading a class. There can be multiple static blocks inside a Java program.

Java Static Methods

Static methods are the methods declared with the static keyword. When we declare a method as static, we can call this method or access this method without creating an object or instance of the class.

As we know that for calling non-static methods, we need to first create the object of the class and then call the method through the object, but unlike the non-static methods, we can call the static methods directly with the class name.

Java Static Classes

We can also declare a class as static; the condition being that the class should be a nested class, i.e, it should be present within the class, then only we can declare it as static.

The nested static class does not require an object of its Outer class. One more restriction with the nested static class is that it cannot access the non-static data members of the Outer class.

Have you ever wondered why this main() method is always static?

It is because there is no need to create an object of the class to call this method. We have already learned this fact in the section of the Static method of this article.

We know that during the execution of the java program, the JVM always starts with the main() method of the class, and suppose this main() method was non-static then JVM had to create the object of the class and then call this method which would raise the problem of memory allocation.

Therefore the main() method of Java is always declared as static.

23) Java inner class -

A class within another class is called a nested class or an inner class. In other words, the inner class is a member of a class just as a class has a member as variables and methods; it can also have another class as its member.

Such class, which has other classes as its members is called a top-level class or outer class. A top-level class may contain any number of inner classes.

Need for Inner Class in Java

- **It helps in the logical grouping of classes that belong together:**

Suppose there is a class that is useful only to a single class, then we can logically embed it in that class and keep the two classes together. It will help their package to be more streamlined.

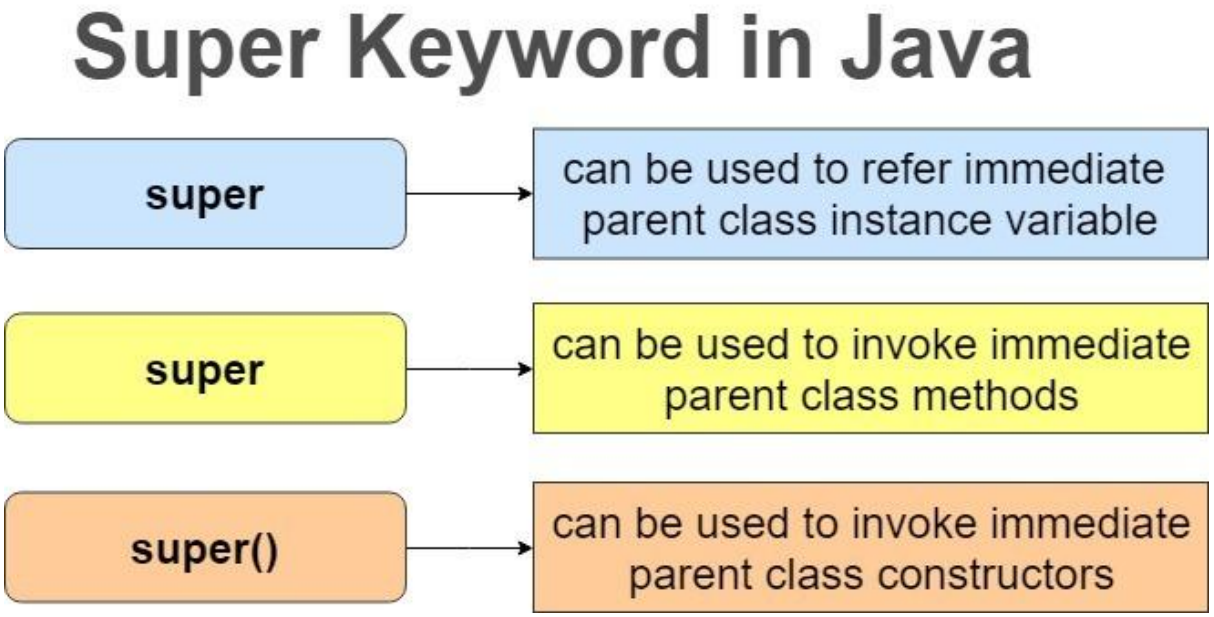
- **It helps to increase the encapsulation:**

Suppose there are two top-level or outer classes, named C1 and C2, where class C2 needs to access the private members of the class C1. By nesting class C2 within the class C1, members of C1 can be declared as private, and C2 can access them.

Also, we can protect C2 from the outside world. Eventually, this will lead to strong encapsulation and security.

- **It helps to increase the readability and maintainability of the code:**
-

24) Java inner class -



25) Oops

Everything depends on object
Whenever we started development in any field , first we have to make low level design
Means we have a design system in such a way that every scenario and requirement meets.
Nowadays in development we cover the real life problem so to deal with that real world object we need object oriented programming .

We try to solve problems like real life problems.

Company has different department
In every department there might be different entities. These are objects that mean everything in the company revolves around them.

Class and Object -

Let's have a batch of student with 10 students
All they considered as object
Every student has some identity (name , email , address , phone etc)
These are properties of each student
For all those 10 student we don’t want to write their properties again and again
So what i will do is
I will wrap every properties in the class
That means i am making an template for each object
If i come to your organisation you will give me a form to fill my details right

Every time the candidate will come to you and you will not write the properties of every student
You have just declared one template for them and whenever any object needs that info you will simply give them that template and based on that they will fill their info.

Class is a template to make an object .

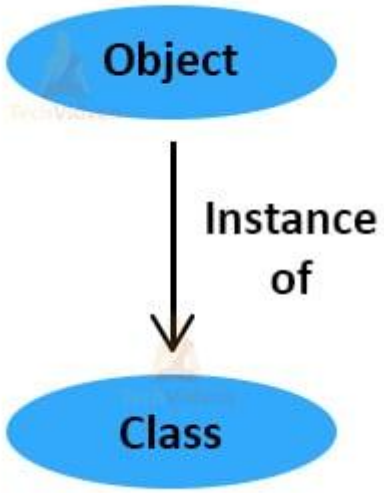
Packages are the top on the project
Every class must have in the main package
Package may have class and package

Only in a package the class is exist

Java.util
Scanner - import java.util.scanner

Without main class the class will not run

What is an Object?



The object is a bundle of data and its behavior or methods.
Objects have two characteristics: states and behaviors.

Examples of states and behaviors of an object:

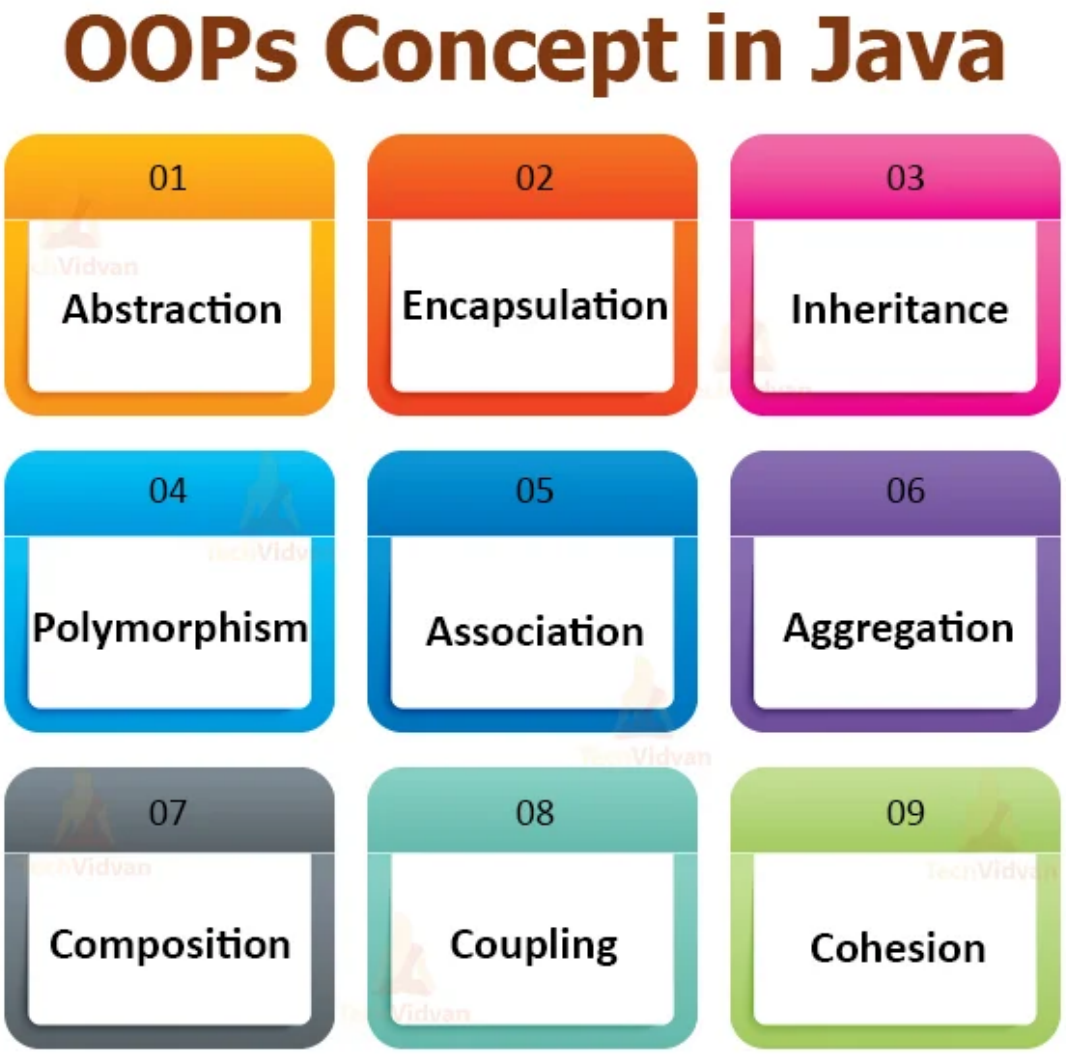
Object: Student

State: name, age, gender

Behavior: study, play, run

What is Class in OOPs Concepts?

A class is a blueprint that creates as many objects as we need. For example, we have a class Website that has two data members or fields or instance variables. This class is just a blueprint or a template. It does not represent any real website.



1. Abstraction in Java

Abstraction is a process to represent only "relevant" or essential data and "hide" the unnecessary or background details of an object from the user.

Suppose, You are driving a car. While driving, you only know the essential features of a car, such as gear handling, steering handling, use of the clutch, accelerator, brakes, etc. But while driving, do you get into the car's internal details like wiring, motor working, etc.?

You just change the gears or apply the brakes, etc. What is happening inside the car is hidden from you. This is an abstraction where you only know the essential things to drive a car without including the background details or explanations.

Take another example of 'switchboard'. You only press individual switches according to your requirement. What is happening inside, how it is happening, etc. You need not know. Again this is an abstraction; you know only the essential things to operate on the switchboard.

We can achieve abstraction in two ways:

a) Abstract Class

b) Interface

a. Abstract class

An Abstract class in Java uses the 'abstract' keyword. If we declare a class as abstract, we cannot instantiate it, which means we cannot create an abstract class object. Also, In an abstract class, there can be abstract as well as concrete methods.

We can achieve 0-100% abstraction using abstract class.

b. Interface

Interface is a blueprint of a class. An interface is a collection of abstract methods and static constants. Each method in an interface is public and abstract, but there is no constructor. Interfaces also help to achieve multiple inheritance in Java.

We can achieve 100% abstraction using interfaces.

2. Encapsulation in Java

Encapsulation is a way of combining both data members and functions/methods into a single unit. In Encapsulation, we keep the fields within a class as private, and then we provide access to them using public getter and setter methods.

Encapsulation is a kind of protective barrier that keeps the data and methods safe within the class itself. Using Encapsulation, we can reuse the code components or variables without allowing open access to the data.

We can implement Encapsulation in two ways:

1. Declare the instance variables as private. We make them private, so no one from outside the class can access them directly. We can only set and get the values of these variables using the methods of the class.

2. Provide the getter and setter methods in the class. These methods set and get the values of the instance variables.

3. Inheritance in Java

Inheritance is a feature of Object-Oriented Programming in Java that allows programmers to create new(child) classes that share some of the attributes of existing(parent) classes. It is an object-oriented process by which one class acquires or inherits the properties and functionalities of another class.

Inheritance provides the reusability of code. Each child class defines only those features that are unique to it, and the child class inherits the rest of the features from the parent class.

The most significant advantage of Inheritance is that we need not rewrite the base class's code in the child class. We can use the variables and methods of the base class in the child class as well.

Types of Inheritance in Java

1. Single Inheritance: Single Inheritance is a child and parent class relationship where one class extends another class.

2. Multilevel Inheritance: Multilevel Inheritance is a child-parent relationship when a class extends the child class, and that child class becomes a parent class for another class, and so on. For example, class A extends class B, and class C extends class B.

3. Hierarchical Inheritance: Hierarchical Inheritance refers to a child-parent class relationship where more than one class can extend the same parent class. For example, class B extends class A, and class C extends class A.

4. Multiple Inheritance: Multiple Inheritance refers to a parent-child class relationship when one child class extends more than one parent class. This means, a child class can have more than one parent class. Java does not support multiple inheritance using classes, but with interfaces.

4. Polymorphism in Java

This Java OOPs concept lets programmers use the same word to mean different things in different contexts. One form of Polymorphism in Java is method overloading. That's when the code itself implies different meanings. The other form is method overriding.

Polymorphism is an object-oriented programming feature that allows us to perform a single action in different ways.

Types of Polymorphism

a. Static Polymorphism

b. Dynamic Polymorphism

a. Static Polymorphism

Polymorphism that the compiler resolves during the compile-time is called the static polymorphism. We can consider Method overloading as a static polymorphism example in Java.

Method Overloading allows us to have more than one method with the same name in a class with a different signature. The above example of polymorphism is the example of method overloading or static polymorphism in Java.

b. Dynamic Polymorphism

The other name for Dynamic Polymorphism is Dynamic Method Dispatch. Dynamic or runtime polymorphism is a technique in which the overridden method is resolved at runtime rather than the compile-time. That's why it is called runtime polymorphism.

Since both child class and parent class have the same method , JVM determines which methods to call at runtime.

Cohesion

Cohesion refers to the level of performing a single well-defined task by a component. A highly cohesive method performs a single well-defined task. While, the weakly cohesive method will split the task into different parts.

The java.io package is a highly cohesive package in Java because this package contains the classes and interfaces related to I/O(Input/Output). The java.util package is considered as a

weakly cohesive package because there are unrelated classes and interfaces in it.

A relationship in Java means different relations between two or more classes. For example, if a class Bulb inherits another class Device, then we can say that Bulb is having is-a relationship with Device, which implies Bulb is a device.

In Java, we have two types of relationship:

1. **Is-A relationship: Whenever one class inherits another class, it is called an IS-A relationship.**
2. **Has-A relationship: Whenever an instance of one class is used in another class,**
3. **it is called HAS-A relationship.**

Is-A relationship

IS-A Relationship is wholly related to Inheritance. For example – a kiwi is a fruit; a bulb is a device.

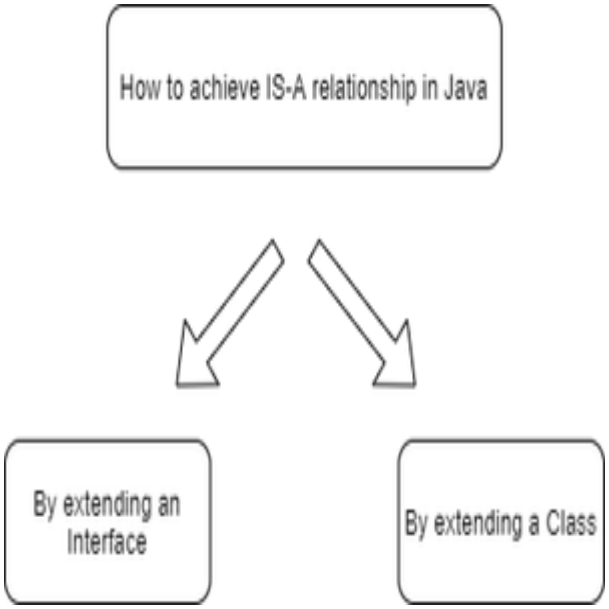
- **IS-A relationship can simply be achieved by using extends Keyword.**
- **IS-A relationship is additionally used for code reusability in Java and to avoid code redundancy.**
- **IS-A relationship is unidirectional, which means we can say that a bulb is a device, but vice versa; a device is a bulb is not possible since all the devices are not bulbs.**
- **IS-A relationship is tightly coupled, which means changing one entity will affect another entity.**

Advantage of IS-A relationship

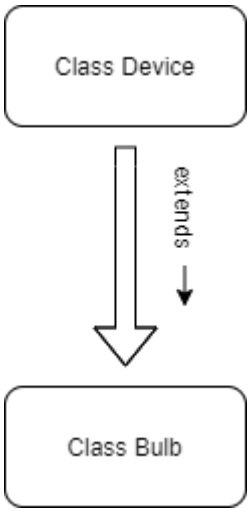
- **Code Reusability.**
- **Reduce redundancy.**

How to achieve IS-A relationship

IS-A relationship can simply be achieved by extending an interface or class by using extend keyword.



Let's understand the IS-A relationship with the help of a flowchart –



In the above flowchart, the class Bulb extends class Device, which implies that Device is the parent class of Bulb, and Class Bulb is said to have an Is-A relationship. Therefore we can say Bulb Is-A device.

Has-A Relationship Examples

---

1. A most common example of Has-A relationship in Java is “A person has an address”.

the person represents the whole and the address represents the part. A part cannot exist by itself.

There are two types of Has-A relationship that are as follows:

a. Aggregation

b. Composition

Aggregation:It focuses on establishing a Has-A relationship between two classes.

In other words, two aggregated objects have their own life cycle but one of the objects has an owner of Has-A relationship and child object cannot belong to another parent object.

For example, a library has students. If the library is destroyed, students will exist without library.

Composition: It focuses on establishing a strong Has-A relationship between the two classes.

In other words, two composited objects cannot have their own life cycle. That is, a composite object cannot exist on its own. If one composite object is destroyed, all its parts are also be deleted.

For example, a house can have multiple rooms. A room cannot exist independently and any room cannot belong to two different houses. If the house is destroyed, all its rooms will be automatically destroyed.

26) Garbage collection in java

The heap is divided into three sections:

- **Young Generation: Newly created objects start in the Young Generation. The Young Generation is further subdivided into an Eden space, where all new objects start, and two Survivor spaces, where objects are moved from Eden after surviving one garbage collection cycle. When objects are garbage collected from the Young Generation, it is a minor garbage collection event.**
- **Old Generation: Objects that are long-lived are eventually moved from the Young Generation to the Old Generation. When objects are garbage collected from the Old Generation, it is a major garbage collection event.**
- **Permanent Generation: Metadata such as classes and methods are stored in the Permanent Generation. Classes that are no longer in use may be garbage collected from the Permanent Generation.**

27) Method overloading and overriding in java

Java Method Overloading example

```
1. class OverloadingExample{
2. static int add(int a,int b){return a+b;}
3. static int add(int a,int b,int c){return a+b+c;}
4. }
```

Java Method Overriding example

```
1. class Animal{
2. void eat(){System.out.println("eating...");}
3. }
4. class Dog extends Animal{
5. void eat(){System.out.println("eating bread...");}
6. }
```

28) Annotation in java

- Annotations start with '@'.
- Annotations do not change the action of a compiled program.
- Annotations help to associate *metadata* (information) to the program elements i.e. instance variables, constructors, methods, classes, etc.

- **Annotations are not pure comments as they can change the way a program is treated by the compiler. See below code for example.**
- **Annotations basically are used to provide additional information, so could be an alternative to XML and Java marker interfaces.**

29) Why multiple inheritance not support in java

However, a class can implement one or more interfaces, which has helped Java get rid of the impossibility of multiple inheritances.

The reason behind this is to prevent ambiguity.

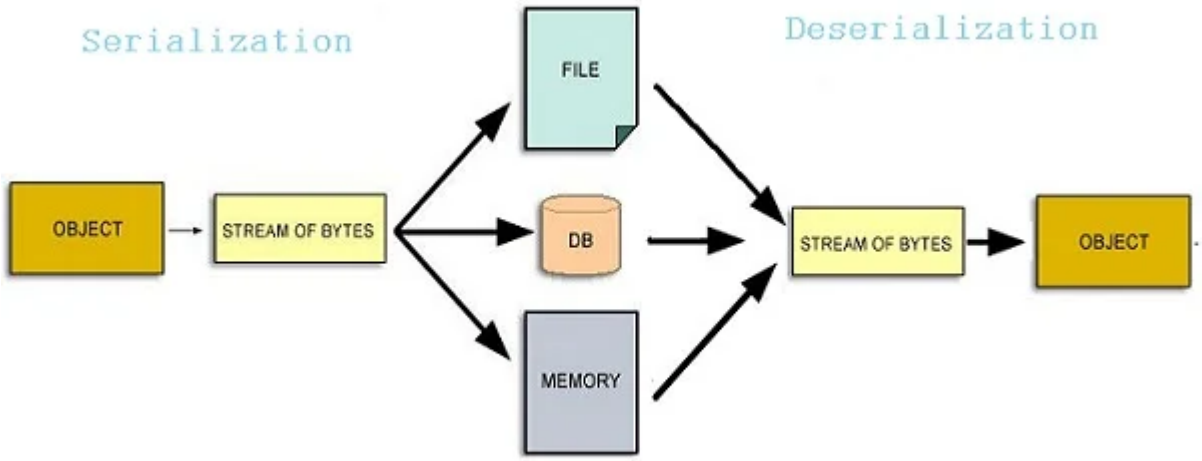
Consider a case where class B extends class A and Class C and both class A and C have the same method display().

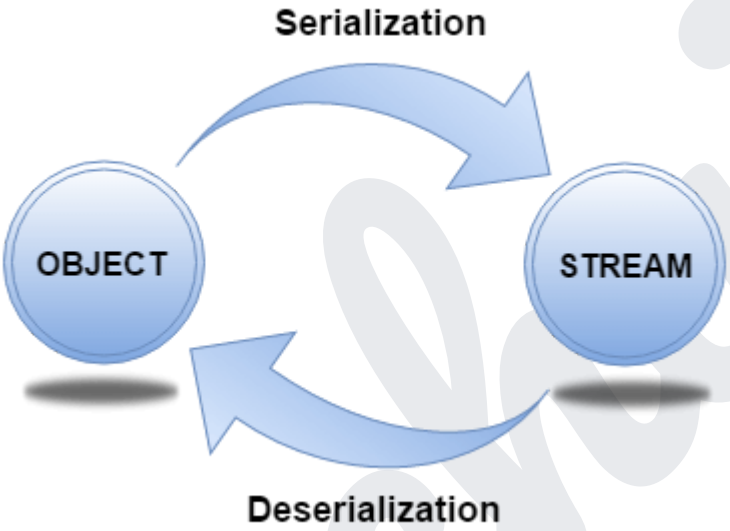
Now java compiler cannot decide, which display method it should inherit. To prevent such situation, multiple inheritances is not allowed in java.

30) Serialization and deserialization in java

Serialization is a process of converting an object into a sequence of bytes which can be persisted to a disk or database or can be sent through streams. The reverse process of creating object from sequence of bytes is called deserialization.

Static members are never serialized because they are connected to class not object of class.





Serializable is a marker interface (has no data member and method). It is used to "mark" Java classes so that the objects of these classes may get a certain capability. The Cloneable and Remote are also marker interfaces.

The Serializable interface must be implemented by the class whose object needs to be persisted.

```
1. import java.io.Serializable;
2. public class Student implements Serializable{
3.  int id;
```

```
4.  String name;
5.  public Student(int id, String name) {
6.   this.id = id;
7.   this.name = name;
8.  }
```

In the above example, *Student* class implements Serializable interface. Now its objects can be converted into stream. v

transient Keyword

While serializing an object, if we don't want certain data member of the object to be serialized we can mention it transient. transient keyword will prevent that data member from being serialized.

```
class studentinfo implements Serializable
{
   String name;
   transient int rid;
   static String contact;

}
```

- **Making a data member transient will prevent its serialization.**
- **In this example rid will not be serialized because it is transient, and contact will also remain unserialized because it is static.**

31) Array in java

1991-sunmcrosystem , later acquired by oracle
Start name - oak'
James gosling and patrick - green team

Java - object oriented general purpose robust platform independent to create application software .

Jvm (java virtual machine) - specification , instruction , how bytecode exc.

Jre - java runtime env. – implementation of jvm

Jre has jvm in it.

Jdk - java development kit - contain devv took , jre

Jre can only run program so jdk privive compoile debugger

jdk - jre + compiler +ndebugger + java doc
Jre- jvm + class library java.util

8888888

Java setup in system -

Primitive data type - inbuilt data type - 8 type (byte short char int long float double boolean),
Non primitive / reference data type / user defined data type - string , array , list

Boolean - 1 bit
Byte - 1 byte
Short - 2
Char - 2
Int - 4
Float - 4
Long - 8
Double - 4

Operator - we perform any operation using operator
Operators are symbol to perform operation
3+5
3,5→ operand
+ → operator

Bitwise - &(AND) , ^(exclusive or) , |(inclusive or)
Logical - && ||

?: → ternary

For each -

```
Int nums[]={10,20,30,40};
Int s=0;
For ( int x : nums)
{

s=s+x;
}
```

Break - terminate loop
Continue - skip current iterator and continue with next iterator

Array -

OOPs -

Everything depends on object
Whenever we started development in any field , first we have to make low level design
Means we have a design system in such a way that every scenario and requirement meets.
Nowadays in development we cover the real life problem so to deal with that real world object we need object oriented programming .

We try to solve problems like real life problems.

Company has different department
In every department there might be different entities. These are objects that mean everything in the company revolves around them.

Class and Object -

Let's have a batch of student with 10 students
All they considered as object
Every student has some identity (name , email , address , phone etc)
These are properties of each student
For all those 10 student we don't want to write their properties again and again
So what i will do is
I will wrap every properties in the class
That means i am making an template for each object
If i come to your organisation you will give me a form to fill my details right
Every time the candidate will come to you and you will not write the properties of every student
You have just declared one template for them and whenever any object needs that info you will simply give them that template and based on that they will fill their info.

Class is a template to make an object .

Packages are the top on the project
Every class must have in the main package
Package may have class and package

Only in a package the class is exist

Java.util
Scanner - import java.util.scanner

Without main class the class will not run

Access modifier -

5) Default - if we write nothing considered as default , access within the package
6) Public - access everywhere

7) Private - only can be access within the class only
8) Protected -

Getter and setter →

I have made the property private so i can't access outside the class but i want to access this value and want to change here comes the getter and setter

Why make private
If someone makes a object of class and set the enrol number -ve , this should not be allowed that's why we make private and i can give the value using a function

Java has member variable and member methods

```
Public void setEnroll(int enrol){
This.enroll = enrol
}
```

This keyword → we have enrol class property and we are passing same name in set enrol
Now I want to change the local var. From set enrol value but java compile confused about the same name and set value change it's value itself so we use this keyword
This keyword point the class variable

What if we print object

In array we have stack and heap memory
Address stored in stack and whole object stored in heap memory
When we try to print object we only get object
Packagename.classname@address
If we print this then we get object
Because this keyword is local pointer which pointing to the object

Using this we can call function and also constructor as well using inheritance

Inner class can use private variable of outer class

constructor - we can have a lot of entity so every time we have to call the set method and set it's value rather than we can create a constructor and set value at a time of making an object constructor construct an object

There is a default constructor

We don;t have garbage in java
There are some fix value

For int =0
String = null
boolean = false
double=0.0