OOP1 : Introduction

Example -> car class has wheels, seats, headlight, engine type
Template of car -> how it should be
Class is template of object and object is instance of class
BMW is instance of car template

Class is -> logical construct
Object is -> physical entity -> occupies space in memory

Properties of object -> state, identity, behaviour

Instance variables -> variables inside the object
Reference variables -> object
. Operator -> Linking reference variable with instance variable

Student student1; -> declaring reference to the object of type Student
By default content in the object are null
student1 = new Student(); -> instantiate the object
New keyword -> dynamically allots memory to runtime and returns reference to it
Objects are created in heap memory and reference variables are in stack and they will point to the object in heap
Student student1 [this happens at compile time] = new Student(); [this dynamic memory allocation happens at run time]

Constructor is a special type of function in class -> runs when you create the object -> can allocate some variables
It binds instance variables with objects
Default constructor is with default values for instance variables in the class
Default constructor and parametrised constructor -> constructor overloading -> Polymorphism
this keyword -> refer to the current object of class
We can call parametrised constructor from default constructor using this(parameter) inside default one

student1 = student2; -> both reference variable are pointing to the same object in the heap

Wrapper class example ->
We have int num = 10; and also we can Integer num = 10;
Here int is primitive data type but Integer is wrapper class and it has some methods.

Regular swap function will not work -> In java, everything is pass by value and not a pass by reference.

Final keyword -> restricts from modifying it
Final variables are always need to be initialised when you are declaring
Final is meaningful for primitive variables
Now if reference variable has final keyword, still we are able to change the instance variables of the object
Example :
final Student student1 = new Student();
student.name = “Sagar”;
student.name = “Harshal”; -> this is possible
Student student2 = new Student(“Sam”);
student1 = student2; -> this is not possible

When there is object in heap to whom none reference variable is pointing -> it will be removed by garbage collector by calling finalize() method
Now if you want to do something when java is destroying the object -> you can override the finalise() method in the class
We can not destroy the object manually -> it will be done automatically by java

OOP2 : Packages, Static, Singleton Class, In-built Methods

println() method internally calls toString() method
If toString is not present in the object class then it takes default implementation

package is just folder, it just opposite of company URL so that it will be unique

static variable is common to all objects. It will be same for all object of the class.
Access static variable by class name and not by reference variable
It can be accessed before object created of class. They are object independent
public static void main() -> main is first method to run in the project -> it should run without created object for it
So static variables or methods, they actually belong to the class and not the object
Inside static method, you can’t use that is non static method or variable directly as we know something non static belongs to the instance of the class that is object.
We can intantiate object and use that non static method inside static method.
We can’t use this keyword in static method of the class.

static{} block in the class will be run once when class is loaded for the first time

Inner class can be static but outer class can not be static. The static keyword in Java is used to define members (fields, methods, or nested classes) that belong to the class itself rather than to instances of the class. Since outer classes are already top-level entities, there's no need to make them static

As we know, static stuff is not dependent on the objects, they are class concepts and objects are created run time, classes are compiled at compile time. So static stuff is resolved at compile time.

System.out.println() :
out -> it is static variable of System class having PrintStream type
println() -> it is method of PrintStream class

Singletone class -> you can create object of it only once. This can be achieved by :
- make constructor private
- we will make one public method of same class that return instance of class
- for that we need to create private instance variable of same class type in same class

- if we need to create object of singletone class in another class, we need to make this public method static and hence instance variable static
- but this way we can create many objects, so add if clause in that method to check if instance == null; only then create object
- so when instance is not equal to null -> new reference variable will point to same object
- In this way, object is created only once

OOP3 : Inheritance, Polymorphism, Encapsulation, Abstraction

Inheritance :

Child class is inherting the properties from parent class
Child class = properties of parent class + properties of its own
When we call constructor of child class, we need to initialize variables of parent class also
super() -> this is used to initialize variables of parent class

anything that is private can be used in the same file only and not in the outside file

It is the type of reference variable and not type of object created recognises what is accessible by that reference variable
If reference variable of parent class but it is pointing to child class object -> it will not able to access child class instance variables
You are given access to variables those are in the ref type

Why child class reference variable can't point to parent class object ?
access is there -> it should be initialized -> but parent doesn't know about child class variables
that is why it thorws error

Every class you create is inherited from Object class. So we can use super() into our class constructor
If we want to access something from parent class in child class, we can use super keyword. e.g. this.weight = super.weight
super() should be called on first line of child class constructor
If super() is not provided in child class constructor, then it picks default constructor of parent class

Single inheritance -> one class extends another class
Multilevel inheritance -> chain of classes extending
Multiple inheritance -> not allowed in java, we can achieve this but by using interfaces
Hierechiel inheritance -> one parent class is inherited by many child class
Hybrid ineritance -> combination of single + multilevel inheritance

Polymorphism :

meaning -> many ways to represent
types of polymorphism :
- Compile time/static -> method overloading

  Same name for method, but types, arguments, return types, ordering can be different
  e.g. constructor overloading
- Runtime/dynamic -> method overriding
  @Override is simple annotation to check if the method is overridden
  What is able to access is identified by type of reference variable and which is able to access isnidentified by type of object
  Parent object = new Child(); -> upcasting -> Priority is given to the child class method

How java determines which method to run if there is method overriding ?
- Using dynamic method dispatch
  When you have a method in a **parent class** that is **overridden** in a **child class**, and you call that method using a **reference of the parent class**, the JVM determines which version of the method (parent or child) to execute based on the **actual object** being referred to at runtime.

We can not override method that is final
Early binding / static binding :
- Early binding happens at **compile time**. The method to be called is determined by the compiler based on the **reference type** (not the actual object).
- For **static methods**, **private methods**, and **final methods** (since they cannot be overridden).
  Also for **variables** (fields) in Java, as they are resolved at compile time.

Late binding/ dynamic binding :
- Late binding happens at **runtime**. The method to be called is determined by the JVM based on the **actual object** (not the reference type)
- For **overridden methods** (non-static, non-final, non-private methods).This is the basis of **polymorphism** in Java.

Static methods can be inherited but can’t be overridden
Overriding depends on objects -> static doesn’t depend on object -> static can’t be oevridden

<u>Encapsulation :</u>

Wrapping up implementation of the data memebers and methods in a class

<u>Abstraction :</u>

hiding unnessesary details and showing valuable information

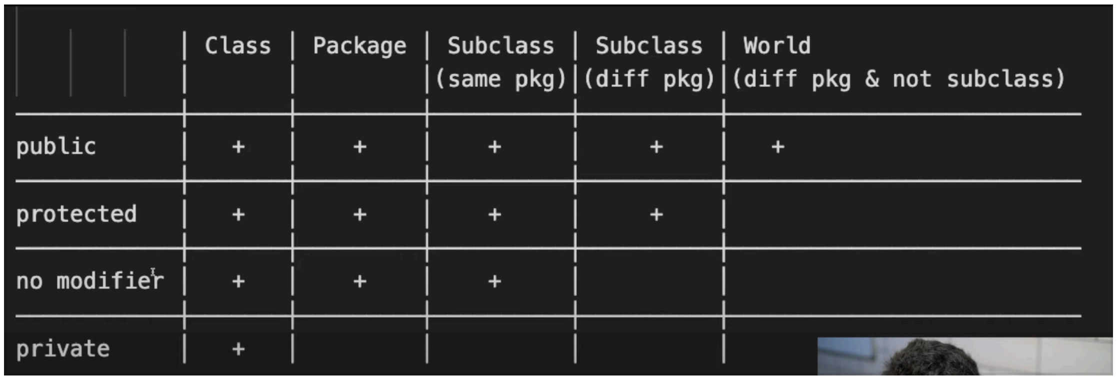
Example : You have car and a key to start. You just need a key to start and not the actual implementation of how clutch are being used with engine stuff.
You just need to use println method and need not to know the actual implementation of it

What is difference between this two ?
Abstraction is solving design level issue and encapsulation is solving engineer level implementation issue.
Encapsulation is all about containing information and abstraction is all about gaining information

What is difference between data hiding and encapsulation ?
data hiding -> security
encapsulation -> hiding complexity of system

OOP4 : Access control, in built packages and Object class

Private -> Only accessible in the same file
Public -> can access anywhere
Default -> in the same package it is allowed
Protected -> it is used in terms of inheritance



	Class	Package	Subclass (same pkg)	Subclass (diff pkg)	World (diff pkg & not subclass)
public	+	+	+	+	+
protected	+	+	+	+	
no modifier	+	+	+		
private	+				

private will be used for sensitive data
no modifier is used when we want to not use data outside of the package
protected -> in different package it is only allowed to access using subclass

Packages -> 1. User defines 2. In built packages
- Lang package : automatically imported, basic operations for calculations
- io : file related things
- util : data structures, collections
- applet : for development
- awt : GUI
- net : for networking

Any class in java extends Object class
It has methods :
- hasCode() -> unique repesentation of the object via number
- toString() -> gives object in string format
- finalize() -> for garbage collection
- equals() :
  obj1 == obj2 -> this checks if both reference variables are pointing to the same object
  obj1.equals(obj2) -> this checks content

instanceof check if that class extends parent class or not getClass().getName() -> name of class

OOP5 : Abstract classes, interfaces, annotations

Java doesn’t support multiple inheritance

abstract classes -> template
Child classes must implement all the methods of parent abstract class

If any method in class is abstract then it should declare as a abstract
We can’t create object of abstract class because in the abstract classes methods are declared only and not initialized

Static methods can’t be overriden so we can’t create abstract static method, but we can define static method in abstract class and call them with class name

Also, you can’t have final abstract class, as final will restrict from inheriting the class

Interfaces are abstract classes in whch functions are abstract and public
Variables are static and final by default

Abstract class can provide the implementation of interface but vice versa is not true

implements keyword

You should not use interface for performance critical applications as classes look for which method to implement at runtime

Generally we create seperate class implementing seperate interfaces

One interface is extending another then you need to implement all methods of those interfaces in a class

Annotations are also interfaces itself

default method in a interface -> default implementation of a method in a interface

Static interface methods will always have a body and they can’t be overidden

OOP6 : Generics, Custom ArrayList, Lambda Expression, Exception handling, Object Cloning

When we write code for custom arraylist -> it can work with only one data type like ArrayList<Integer>
That is why we need Generics -> it allows to provide parametrized type
Generics also helps in type safetly

Generics in Java are a way to create classes, interfaces, and methods that can work with any data type. They allow you to write more flexible and reusable code by using placeholders for types.

Wildcards in Java generics provide flexibility when defining methods that work with various types. They are represented by the question mark (?) and can be used in three main ways: unbounded wildcards, upper-bounded wildcards, and lower-bounded wildcards.

- An unbounded wildcard can represent any type. It is useful when you want to define a method that can work with any type of object.
- An upper-bounded wildcard specifies that the type can be a specific type or any subtype of that type. This is useful when you want to read from a data structure but not write to it.
- A lower-bounded wildcard specifies that the type can be a specific type or any supertype of that type. This is useful when you want to write to a data structure but not read from it.

While using comparable -> compareTo method is required and important while comparing based on instance variables

Lambda expressions in Java are a way to write **shorter and cleaner code** for implementing **functional interfaces** (interfaces with only one abstract method). They allow you to treat functionality as a method argument or code as data. Think of them as a quick way to write a method without needing to define a full method name or return type.

Exceptions are for problems that can be handled and recovered from.
Errors are for serious, unrecoverable problems that usually indicate system or JVM failures.

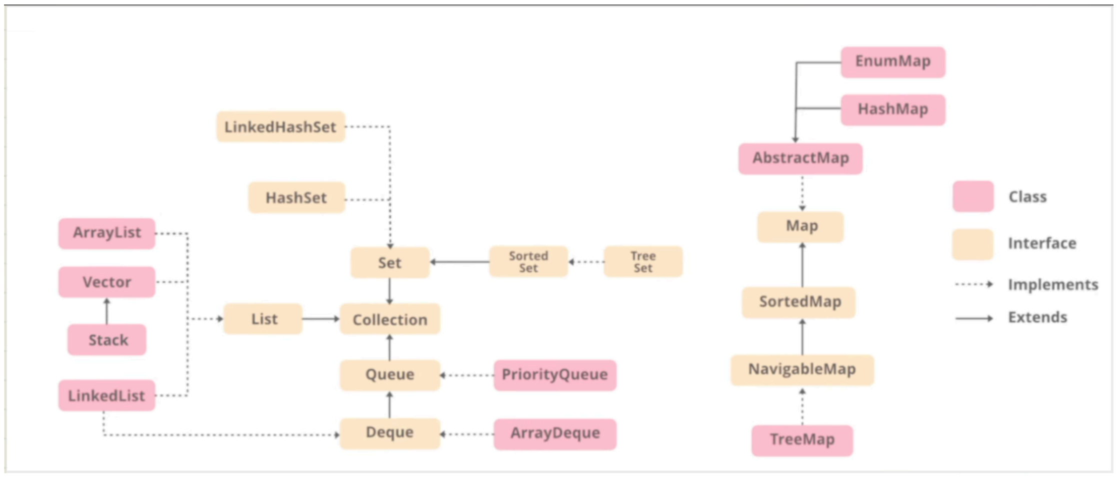
Exception and error both are subclass of Throwable class
- Checked exception -> compile time
- Unchecked exception -> run time

```
try{
}
```

```
catch(Exception e){
}
finally{
This block will always execute
}
```

For cloning object, we have clonable interface and it has clone method, so override in you class
With this it creates, shallow copy :
for primitives making new fields but for reference variables, it is pointing to previous one

OOP7 : Collections framework, Vector class, Enums



Collection and Map are interfaces

Multiple threads can access arraylist but for vector, one thread can access at a time.

Enum is group of variables, constants. They are public, static, final by nature
Enum doesn’t have constructor as public or protected as we do not want to create new object.