OOP (Object Oriented Programming)

Dadta Hiding :

Outside person can’t access our internal data directly or our internal data should not go out direclty, this OOP feature is called as data hiding. And, after the validataion and identification, outside person can access our data.

By declaring the data member as **private**, we can achieve the data hiding.

Abstraction :

Hiding internal implementation and just highlighting the set of services what are offering is a concept of abstraction. By using interface and abstract classes we can achieve abstraction.

Encapsulation :

The process of binding the data and corresponding methods into a single unit is nothing but the encapsulation. If any component follows the data hiding and abstraction such type of component is called encapsulated component.
Ex. *Encapsulation => data hiding + abstraction*

Tightly Encapsulated class:

A class is said to be tightly encapsulated if and only if each and every variable declared as private.

IS-A relationship :

It is also known as inheritance. The main advantage is code reusability. By using the “extends” keyword we can implement the is-a relationship.

HAS-A Relationship :

It is also known as the composition and aggregation. There is no specific keyword to implement the has-a relationshiop. Most of the time, we are dependent on the new keyword.

```
class Engine {
    // engine functions
}
```

```
class Car {
    Engine e = new Engine();
}
```

Difference between the composition and aggregation :

=> Without existing the container object, if there is **no chance** of existing the contained object then container object is strongly assosiated with the contained object. This is nothing but the composition. (Objects are strongly associated !)

Ex. Without existing the university, there is no chance of existing the Departements.

=> Without existing the container object, if there is **a chance** of existing the contained object then container object is strongly assosiated with the contained object. This is nothing but the composition. (Objects are weakly associated !)

Ex. Without existing the departments, there is a chance of existing the Professors.

=> In composition, container object holdes directly the contained object but in the aggregation, container object holds just the reference of contained objects.

IS-A Relationship	HAS-A Relationship
If we want the total functionality of a class autimatically, the we should go for the is-a.	If we want the part of functionality of a class, then we should go for the has-a.
Ex. Inheritance.	Ex. “new” keyword.

Method Signature :

In java, method signature consists of method names, followed by argument types.

Ex. public static int m1(int x, float y) {}
Method signature => m1(int, float)

Overloading :

Two methods are said to be overloaded, iff both mehtods having the same name but different arguments types.
Ex.

```
class Test {

    public static int m1(int x) {return x;}
    public static int m1(double y) {return y;}

}
```

In overloading, method resolution is always takes care by compiler, based on the reference type. Hence overloading is compile time polymorphism or static polymorphism or early binding. Runtime object won't play any role.

Overriding :

What methods are available for parent class, are by default available to the child class. If child class is not satisfied with the parent class implementation then child is allowed to redefine that methods based on its requirement. This process is called as overriding.

In overriding, return types must be same. But this rule was applicable since java1.4, but onwards we can table co-varient return types. According to this, child class return type need not same as parent type, its child class type is also allowed. This is applicable only for objects and not for the primitives.

Private methods can't be overriden. If you are doing so, then you will endup by creating two independent private methods with same name

We can't override the parent class final method, in child class. Otherwise we will get the compile time error. But parent's class abstract method, we should override in the child class.

In overriding, the following modefiers won't keep any restriction.

- synchronized
- native
- strictfp
- abstract

final to non-final overriding is not possible, but non-final to final is possible. abstract to non-abstract and non-abstract to abstract is possible.

While overriding, we can't reduce the scope of the modefier but we can increase the scope of the modefier.
Ex.

default=> default, protected, public.
protected => protected, public.
public => public only.

If a child class throws any checked exception, compulsory parent class method should throw the same checked exception or its parent checked exception. Otherwise we will get the C.T. error.

With respect to static methdos, there is no concept of overriding instead it is method hiding.

Method Hiding	Overrding
Both static.	Non-static.
Compiler reference	JVM runtime.
Also known as compile time polymorphism.	Also known as runtime polymorphism.

With respect to var-arg methods, we can override the one var-arg method with another var-arg only. If we are trying to override with the normal method then it will become overloading and not overrding.

Difference between overloading and overriding :

Property	**Overloading**	**Overriding**
Method Name	same	same
Arg Type	different	same
Method signatures	different	same
Return types	no restrictions	covarient-type
static,final methods	can be	can’t be
Method resolution	by compiler	by JVM
also known as	C.T. polymorphism	R.T. polymorphism

Polymorphism :

One name but multiple forms, is a concept of polymorphism.

Ex.1. Method name is same but we can pass different types of arguments is a concept of overloading.

Ex.2 Method signature is same but in parent class there is one type of implementation and in child class there is another type of implementation is nothgin but the overriding.

Parent class reference can be used t hold the child object but by using that reference we can call only methods availabel in parent class and we can’t call child specific methods. But by using child object reference we can call both paren and child class methods.

Three pillors of OOPs :

- **Encapsulation**
- **Inheritance**
- **Polymorphism**

Coupling :

The degree of dependency between the components is called as coupling. If dependency is more, then we can say it is tightly coupled otherwise it is loosly coupled.

- Worst kind of programming.

Cohesion :

For every component, a clear well defined functionality is required / defined then that component is said to be in high cohession.

- Best kind of programming.

Static control flow :

- Identification of static members from top to bottom
- execution of static variables assignments and static blocks from top to bottom.
- Execution of main method.

Static block will be executed at the time of class loading, hence at the time of class loading, if we want to perform any activity we have to define that inside static block.

Static control flow (in parent to child) :

- Identification of static members from parent to child
- Execution of static variables assignments and static blocks from parent to child
- Execution of strictly child class main method.

Instance control flow :

Whenever we are executing a java class, first static control flow will be executed (IMP). In static, if we are creating the object, the following sequence of event will be executed as a part of instance control flow.

- Identification of instance members from top to bottom.
- Execution of instance variables assignments and instance blocks from top to bottom.
- Execution of constructor.

Note :

- Static control flow is a 1 time activity which will be performed at time of loading.
- But the instance control flow is not one time activity and it will be performed for every object creation.
- Object creation is most costly operation if there is no specific requirement then it is not recomended to create and object.
- **From static area**, we can't access the instance members directly because while executing static area JVM may not identify the instance members.

Instance control flow (Parent and Child) :

- Identification is instance members from parent to child.
- Execution of instance variable assignment and instance block only in parent class.
- Execution of parent constructor
- Execution of instance variable assignements block inchild class
- Execution of child constructor.

Constructors :

Whenever we are creating the object, then some code will be executed automatically to perform the initialization of the object, that is the concept of Constructor.
The main purpose of constructor is initialization of an object but not to create the object.

Rules to create the constructors :

- Name of the class and name of constructor must be same.
- Return type for constructors is not applicable.
- The only applicable modefiers for consturctors are :
  - public
  - private
  - protected
  - default

Compiler is responsible to generate the default constructor, but not JVM. If we are not writing any constructor then and then only compiler will create the default consturtor otherwise it won't.

Prototype of default constructor :

It is always no-arg constructor. The access modefier of default constructor is exactly same as of clas. It consists only one line i.e. **super()**.

Notes :

- The first line inside every constructor should either “super()” or “this()” and if we are not writing anything then compiler will always generate the “super()”.
- We can take super(), or this() only in the first line of constructor.If we are trying to take it anywhere else then C.T.
- Within the constructor we can take either ‘super()’ or ‘this()’ but not both simulteniously.
- We can use super() and this() only in constructor, if we are trying to use it outside of the class then C.T. error.
- For constructors inheritance and overriding concepts are not applicable but overloading concept is applicable.

super() and this()	super and this
These are constructor calls to calll super class and current class constructors resp.	These are the keywords to refer super and current class instance members.
We can use only in constructor as a first line.	We can use anywhere except the static area.
We can use these only once in constructor.	We can use these any number of times.

- ➢ If parent class contains any argument constructor then while writing child class we have to take special care w.r.t. constructors.
- ➢ Whenever we are writing any arg-constructor, it is highly recomended to write no-arg constructor also.
- ➢ If parent class throws any checked exception, compulsory child class constructor should throw the smae checked exception or it parent’s otherwise C.T error.

Singletone Classes :

For any java class, if we are allowed to create only 1 object, such type of class is called as singletone class. We can create the singletone class by creating the private constructor, private static variable and public factory method.