Date
Page No.

OOPS in JAVA

- class
- object
- Method & method Passing
- 4 Pillars of OOPS
  - ↳ Abstraction
  - ↳ Encapsulation
  - ↳ Inheritance
  - ↳ Polymorphism

Class

↳ user defined blue print from which objects are created.
↳ Create multiple objects of the same behaviour.
↳ Class contains fields & methods to describe the behaviour of an object

Java Constructors

↳ code that initialises a newly created object.

① Default constructor (No argument constructor)
Constructor created at the time of class creation by compiler if no other constructor is declared.

② Parameterized constructor
constructor is called parameterized as it contains one or more parameters. Its used to provide different values to the distinct objects at the time of their creation.

Modifiers in Java

- Access Modifiers

Scope	Private	Default	Protected	Public
Same class	✓	✓	✓	✓
" Package subclass	×	✓	✓	✓
" non subclass	×	✓	✓	✓
Diff. package subc.	×	×	✓	✓
Diff. P. non subc.	×	×	×	✓

- Non Access Modifiers
  - Static: Makes attribute dependent on the class
  - Final: Once defined, no change
  - Abstract*: make classes & func abstract
  - Synchronized: Used to sync the thread.

Date
Page No.

Inheritance (Pillar 1)

↳ Property of a subclass to inherit properties & functionalities from the parent class

↳ All objects have "Object class" as their parent class.

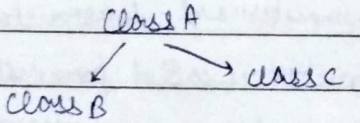
↳ Methods can be overridden but attributes cannot.

↳ To call a parent class constructor, super() is used.

① Single Inheritance
Sub class inherits properties of a single parent
class A ⟶ class B

② Multi Level Inheritance
More than 1 parent but at different levels of inheritance
class A → class B → class C

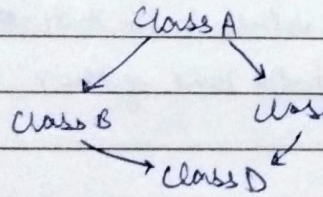
③ Hierarchical Inheritance
1 parent can have more than 1 child



④ Hybrid Inheritance
combination of more than 1 type of inheritance

⑤ Multiple Inheritance
⇒ Not supported in java as it leads to "diamond Problem".
⇒ can be achieved using interfaces
⇒ when 1 class has multiple parents

* Diamond problem:



here the constructors & distructors of class A is called twice when class D is constructed or destroyed.

Association: • Relation between 2 different classes, established via objects.

<1> One to One
(2) One to Many
(3) Many to one
(4) Many to Many

Date
Page No.

Polymorphism: (Pillar 2)

↳ Ability of a (variable) function to take multiple forms.
↳ Allows one method to have multiple implementations.

* Compile time Polymorphism (Static Binding)

[Example: Method overloading]
Type of object is determined at the compile time.
Eg: add (int A, int B)
add (int A)
add (int A, int B, Int C)

* Runtime Polymorphism. (Dynamic Binding)

[Eg: Method Overriding]
↳ overriden method is resolved at runtime.
↳ Reference variable is used to call an overridden method of a superclass at runtime.
Eg: class mobile
↳ void sms() ①
class oneplus
↳ void sms() ②
main
↳ call sms from Object of oneplus, then method ② is called.

Pillar 3: Abstraction

↳ Hiding details & showing only necessary things to the user.

<1> Abstract class (0-100%)

↳ class defined with abstract keyword.
↳ can't be instantiated.
↳ Can contain abstract & non abstract methods.
↳ Can contain constructors & static methods
↳ can contain final methods which force the subclass not to change the body of the method.

Eg:
```
public abstract class MyAbstract_class{
    public abstract void abstractMethod();
    public void display(){System.out.print("concrete method");
}
```

Date
Page No.

<2> Interface : ↳ An interface in java is a blueprint of a class that contains static constants & abstract methods.

↳ Represents an "IS-A" Relationship.

↳ You need to implement an interface to use it's methods & constants.

Relation b/w 2 or more classes:

① IS-A : Eg: Kiwi (is a) fruit
Kiwi → fruit

② HAS-A : Eg: Boy (Has A) NAME.

Eg:
```
public interface Bike {
    public void start();}
class Honda implements Bike {
    public void start(){ print("HBike");
}
Class Rider{
  void main(){
    Bike b1 = new Honda();
    b1.start();
  }
}
```

Encapsulation (Pillar 4)

↳ Binding data using getter & setter methods.

• Steps to Achieve that:

↳ Declare the variables of class as private.

↳ Provide public setter & getter methods to modify & view the variables value.

Eg: pvt string name;
public string getName(){return name;} // getter
public void setName(string Name){this.name = name;} //setter

Aggregation: → special form of association representing "HASA" relationship.

Composition:

↳ More restrictive form of aggregation that makes two entities highly dependent on each other.

↳ It represents the part-of relationship where the composed object cannot exist without the other entity.