Date 21 June (2025)
Page No

1. Introduction & Concepts

• Object - oriented Programming (OOP) is a programming paradigm based on the concepts of objects, which contain data (fields) and methods (functions).

Main OOP principles

(i) Encapsulation → Bind data and code together (e.g., classes).

(ii) Abstraction → Hide complex implementation.

(iii) Inheritance → Acquire properties from another class.

(iv) polymorphism → Many forms (same method behaves differentely).

(1) Classes

A class is a bueprint for objects. It defines properties (variables) and behaviours (methods).

Example →

```
public class Car {
  //properties (fields)
  String color ;
  int speed;

  //Method (behavior)
   void drive () {
  Sout ("Car is driving ~~");
   }
}
```

2. Objects

→ An object is an instance of a class. It has real memory allocated.

object creation → Example →

```
                           object     keyword   class Name
Class Name ← Car myCar = new Car(); //creating an object
             myCar.Color = "Red";
             mycar.drive();
```

(i) Properties of Object

→ An object has:

(a) State (data/fields)
(b) Behavior (methods)
(c) Identity (unique memory reference).

(ii) Access Instance Variables

→ Use dot(.) notation with the object reference.

Example →

```
    Car1.Color = "Blue"; //setting variable
Sout(Car1.Color); //Accessing variable
```

(iii) Dynamic Allocation

→ Dynamic allocation means that memory for an object is allocated during runtime, not a compile time.

In Java, all objects are created dynamically using the new keyword. Java handles memory management automatically using the heap memory.

- Syntax of Dynamic Allocation

⇒ Classname refference. = new classNName(arguments);
- new ClassName(~~) → allocates memory in heap for the object.
- reference → stores the memory address/reference in the stack.

(3) Constructors

A constructor is a special method in a class that is used to initialize objects when they are created.
Its job is to initialize the object (set its values)

- Key points about Constructors:

Feature	Description
a) Same name as class	Constructor must have Same name as the class
b) No return type	No return type, not even void
c) Called automatically	When you use new, Java automatically calls the constructor
d) Used to set values	You can use constructors to set intial values of variables.

Page No.

3 (i) Constructor Types

(a) Default Constructor
→ A default Constructor is a constructor that takes no parameters.
It sets default values when you Create an object.
Java gives you one automatically if you don't make any Constructor yourself.

Example →
```
class Student {
    Student () {    //default Constructor
Sout("Default Constructor Called");
    }
}
Student s = new Student ();
```

(b). Parameterized Constructor
→ A parameterized Constructor is a Constructor that takes inputs (parameters). You give values while creating the object, and those values are used to initialize the object.
Example →
```
                      class Student {
                      String name ;
                      int age;
Parameterized
constructor ← //  Student(String n, int a){
                         name = n;
                         age = a;
                      }
void show(){
  Sout(name, age);
  }
}
Student s = new Student("Ravi", 20); //gives values while
                                       Creating object
 s.show(); // output → Ravi20
```

(C) Constructor Overloading
⇒ Constructor overloading means creating more than one constructor in the same class, but with different number or types of parameters.
It gives us flexibility to create objects in different ways.

Example →

```
                                  Class Book{
                                  String title;
Constructor 1, Takes only one     int pages;
                     parameter
//part of constructor    ←         Book(String t){
      overloading                   title = t;
                                    pages = 100 //
Constructor 2 - Takes two         }
                  parameter
//part of constructor     ←       Book(String t, int p){
      overloading                   title = t;
                                    pages = p;
                                  }
                                // - - Method to print output here -
                                }
     (Cons 1)                      //main class & method here -
   (Cons 2)                        Book b1 = new Book("Java");
                                   Book b2 = new Book("Python", "1300");
       }}
```

(II) Using this Keyword in Constructor
-> this refers to the current object (the one that is being created).
We use this when the parameter name is the same as the instance variable name to avoid confusion and make it clear we are talking about the object's variable.

Example ->

```
Class Person {
  String name;
 Person(String name) {   // 'this.name' means
 this.name = name;          the object's name
 }

  void show() {
  Sout(name);
 }
}
```

(III) Calling one Constructor from Another Using this(--)
-> we use this(--) to call one constructor from another in the Same Class. This help us reuse Code and avoid repating the Same logic in every Constructor.

Example =>

```
Class Laptop{
String brand;
  int price;
 ~~[illegible]~~ Laptop() {
this("HP", 5000); // calls the Constructor below
 }
Laptop(String b, int p){
    brand = b ;
    price = p ;
  }
   void show (){
   sout("brand, price);
    }
 }
```

(4) Wrapper Class

A wrapper class turns a primitive type (like int, char, double) into an object.

⇒ Imp. Points

(i) Needed when using primitives in Collections like ArrayList.

(ii) Java does autoboxing (int → Integer) and unboxing (Integer → int)

Example →

```
int a = 10;
Integer obj = a;    // autoboxing
 int b = obj;   // ouboxing
  sout (b);   // output 10 .
```

(5) final keyword

⇒ final means the value or thing cannot be changed.

⇒ Imp points.

(i) final variable = value can't change

(ii) final method = can't override

(iii) final class = can't inherit

<u>Example</u>

```
        final int x = 5;
    // x = 10;    // Error, cannot change the
                  // final variable
    final class Animal {}
  // class Dog extends Animal    // Error, can't ~~ex~~
                                 // extend final class
```