CLASS -12 (2025-26)

# ***JAVA RIVISION TOUR***

## CHAPTER 2

## Assignments:-

### 1. Definitions:

(i) **Class**: A class in Java is a blueprint or template for creating objects. It defines properties (variables) and behaviors (methods) that the objects created from it will have.
(ii) **Object**: An object is an instance of a class. It represents a real-world entity and contains both data (attributes) and methods (functions) defined by its class.

### 2. Difference between Object and Class:

| Class | Object |
|---|---|
| Blueprint or template | Instance of a class |
| Defines structure and behavior | Holds actual data and can perform actions |
| No memory is allocated | Memory is allocated when created using new keyword |

### 3. Abstraction and Encapsulation:

- **Abstraction** is the process of hiding complex internal implementation details and showing only essential features to the user.
- **Encapsulation** is the technique of bundling the data (variables) and methods that operate on the data into a single unit, i.e., class, and restricting direct access to some of the object's components.

**Interrelation**: Encapsulation helps achieve abstraction by hiding the internal data using access modifiers and exposing only necessary information via public methods.

**Example of Abstraction:**

```
abstract class Shape {
    abstract void draw();
}

class Circle extends Shape {
    void draw() {
        System.out.println("Drawing Circle");
    }
}
```

### 4. Key features of objects:

- **State**: Represented by attributes or fields.
- **Behavior**: Represented by methods.
- **Identity**: Each object has a unique identity (memory location).
- **Encapsulation**: Combines data and behavior.
- **Reusability**: Can be reused through inheritance.

### 5. Constructor and its Role:

- A **constructor** is a special method that is automatically called when an object is created.
- Its role is to initialize the object’s data members.

### 6. Two basic types of constructors in Java:

- **Default constructor** (no parameters)
- **Parameterized constructor** (with parameters)

### 7. Difference between class members and instance members:

| Class Members | Instance Members |
|---|---|
| Declared with static keyword | No static keyword |

| | |
|---|---|
| Belong to the class | Belong to the instance (object) |
| Accessed using class name | Accessed using object reference |

### 8. Keyword to create a class member:

- **static**

### 9. Can static methods access instance members?

- **No**, static methods cannot directly access instance members because they do not belong to any object.

### 10. Keyword to protect a class from outside the package (by default):

- **(d) Don't use any keyword at all**
  (default access modifier restricts access to within the same package)

### 11. Keyword to make a member visible in all subclasses across packages:

- **(c) public**

12. The use of `protected` keyword to a member in a class will restrict its visibility as follows:
**(c) visible in all classes in the same package and subclasses in other packages**

13. Keywords used to control access to a class member:
**(a) default, (c) protected, (e) public**

**14.**

- **Private members**: Accessible only within the class itself.
- **Public members**: Accessible from any other class.

**15.**

- **Protected members**: Accessible within the same package and by subclasses in other packages.
- **Public members**: Accessible from anywhere.
- **Private members**: Accessible only within the class.

**16.** A class enforces **information hiding** using **access modifiers** like `private` to restrict access to its internal data and expose only what's necessary using `public` methods.

**17.**

- `**static**` keyword makes a member belong to the class rather than to any object.
- **With static**: Shared across all instances.
- **Without static**: Separate copy for each object.

**Example:**

```
class Demo {
    static int count = 0;
    int id;

    Demo(int id) {
        this.id = id;
        count++;
    }
}
```

**18.**

```
class Student {
    private int rollno;
    private char grade;
    public Student(int r, char g) {
        rollno = r;
        grade = g;
    }

    public void init() { } // just declaration
    public void display() { } // just declaration
}
```

---

**19.**

```
class Sample {
    int i;
    char c;
    float f;

    public Sample(int i, char c, float f) {
        this.i = i;
        this.c = c;
        this.f = f;
    }
}
```

---

**20.**
**Constructor functions obey access rules** means their visibility depends on access modifiers like `public`, `private`, etc., just like other methods or fields.

---

**21.**

- **Parameterized constructor**: Takes arguments to initialize members.
- **Non-parameterized constructor**: Takes no arguments and often assigns default values.

---

**22.**
An **object maintains its state** using **instance variables**. Each object has its own copy of these variables.

---

**23.**
**Constructor vs Method:**

| Constructor | Method |
|---|---|
| Same name as class | Can have any name |
| No return type | Has return type |
| Automatically called during object creation | Called manually on an object |

---

**24.**
If a method or field is **static**, it belongs to the class, not to any instance. Shared by all objects.

---

**25.**

```
class Point {
    double x, y;

    Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    double distance(Point p) {
        return Math.sqrt(Math.pow(x - p.x, 2) + Math.pow(y - p.y, 2));
    }
}
```

**26.**

```
Animal Lion = new Animal(240, 3.6);
```

**27.**

```
Lion.weight = 250;
Lion.length = 3.8;
```

**28.**

```
class BankAccount {
    private String name;
    private String type;
    private double balance;

    BankAccount(String name, String type, double balance) {
        this.name = name;
        this.type = type;
        this.balance = balance;
    }

    void deposit(double amount) {
        balance += amount;
    }

    void withdraw(double amount) {
        if (balance >= amount) balance -= amount;
        else System.out.println("Insufficient Balance");
    }

    void display() {
        System.out.println("Name: " + name + ", Balance: " + balance);
    }
}
```

**29.**

```
class Simple {
    int x = 10;
    static int y = 5;

    public static void main(String[] args) {
        Simple obj = new Simple();
        int input = Integer.parseInt(args[0]);
        int result = (obj.x * input) / y;
        System.out.println("Result: " + result);
    }
}
```

**30.**

```
class DistanceConverter {
    public static void main(String[] args) {
        double feet = Double.parseDouble(args[0]);
        double inches = feet * 12;
        System.out.println(feet + " feet = " + inches + " inches");
    }
}
```

## 28. Design a class to represent a bank account:

```
class BankAccount {
    private String depositorName;
    private String accountType;
    private double balance;

    // Constructor to initialize values
    public BankAccount(String name, String type, double amount) {
        depositorName = name;
        accountType = type;
        balance = amount;
    }

    // Method to deposit an amount
    public void deposit(double amount) {
        balance += amount;
    }

    // Method to withdraw an amount after checking balance
    public void withdraw(double amount) {
        if (balance >= amount) {
            balance -= amount;
        } else {
            System.out.println("Insufficient balance.");
        }
    }

    // Method to display name and balance
    public void display() {
        System.out.println("Name: " + depositorName);
        System.out.println("Balance: " + balance);
    }
}
```

## 29. Program using command-line argument, instance and class variables:

```
class Simple {
    int x = 10;                // instance variable
    static int y = 2;          // class variable
    public static void main(String[] args) {
        if (args.length > 0) {
            int input = Integer.parseInt(args[0]);
            Simple obj = new Simple();
            int result = (obj.x * input) / y;
            System.out.println("Result: " + result);
        } else {
            System.out.println("Please enter a number as command line argument.");
        }
    }
}
```

---

## 30. Program to convert feet to inches using command-line arguments:

```
class DistanceConverter {
    public static void main(String[] args) {
        if (args.length > 0) {
            double feet = Double.parseDouble(args[0]);
            double inches = feet * 12;
            System.out.println(feet + " feet = " + inches + " inches");
        } else {
            System.out.println("Please enter distance in feet as command line argument.");
        }
    }
}
```

**************************************