Morning Session:

1. Inheritance: Building Class Hierarchies and Code Reusability

Concept

- Inheritance allows a class (**subclass/child**) to inherit fields and methods from another class (**superclass/parent**).
- `extends` keyword is used.
- **Types of Inheritance**:
  - Single (`A → B`)
  - Multilevel (`A → B → C`)
  - Hierarchical (`A → B`, `A → C`)
  - **Java does not support Multiple Inheritance** (but interfaces allow it).

Example

```
class Animal {
    void eat() {
        System.out.println("Animal is eating");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("Dog is barking");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.eat();  // Inherited from Animal
        d.bark(); // Dog's own method
    }
}
```

Output:

```
Animal is eating
Dog is barking
```

2. Polymorphism: Dynamic Method Dispatch & Runtime Binding

Concept

- **Polymorphism** = "Many forms" (Same method behaves differently).
- **Types**:
  - **Compile-time (Method Overloading)**
  - **Runtime (Method Overriding + Inheritance)**

Example (Method Overriding)

```
class Vehicle {
   void run() {
      System.out.println("Vehicle is running");
   }
}

class Bike extends Vehicle {
   @Override
   void run() {
      System.out.println("Bike is running");
   }
}

public class Main {
   public static void main(String[] args) {
      Vehicle v = new Bike(); // Upcasting
      v.run(); // Calls Bike's run() (Runtime Polymorphism)
   }
}
```

Output:

```
Bike is running
```

3. Exception Handling (try-catch-finally)

Concept

- **Exception**: An unexpected event disrupting normal flow.
- **Keywords**:
  - `try` → Risky code
  - `catch` → Handles exception
  - `finally` → Always executes
  - `throw` → Manually throw exception
  - `throws` → Declares possible exceptions

Example

```
public class Main {
    public static void main(String[] args) {
        try {
            int a = 10 / 0; // ArithmeticException
        } catch (ArithmeticException e) {
            System.out.println("Cannot divide by zero!");
        } finally {
            System.out.println("This always runs");
        }
    }
}
```

Output:

```
Cannot divide by zero!
This always runs
```

4. Abstraction & Encapsulation

Abstraction (Hiding Implementation)

- **Abstract Classes**:

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

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

- **Interfaces (Pure Abstraction):**

```
interface Drawable {
    void draw();
}

class Circle implements Drawable {
    @Override
    public void draw() {
        System.out.println("Drawing Circle");
    }
}
```

Encapsulation (Data Hiding)

```
class Student {
    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Afternoon Session:

1. String Manipulation (String Class Methods)

Key Methods

Method	Example	Output
`length()`	`"Hello".length()`	`5`

Method	Example	Output
charAt(int)	"Java".charAt(2)	'v'
substring(int)	"Hello".substring(1)	"ello"
equals()	"Hi".equals("hi")	false
toUpperCase()	"abc".toUpperCase()	"ABC"

Example

```
String s = "Hello World";
System.out.println(s.length());      // 11
System.out.println(s.substring(6));  // "World"
System.out.println(s.toUpperCase()); // "HELLO WORLD"
```

2. Access Modifiers & Package Visibility

Modifier	Class	Package	Subclass	World
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
default	✓	✓	✗	✗
private	✓	✗	✗	✗

Example

```
class Test {
   public int a = 10;
   private int b = 20;
   protected int c = 30;
}
```

3. File I/O (Reading & Writing Text Files)

Reading a File

```
import java.io.*;
```

```
public class Main {
    public static void main(String[] args) {
        try (BufferedReader br = new BufferedReader(new FileReader("input.txt"))) {
            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Writing to a File

```
try (BufferedWriter bw = new BufferedWriter(new FileWriter("output.txt"))) {
    bw.write("Hello, File I/O!");
} catch (IOException e) {
    e.printStackTrace();
}
```

4. Multithreading (Concurrent Programming)

Creating Threads

1. **Extending `Thread` Class**

```
class MyThread extends Thread {
    public void run() {
        System.out.println("Thread is running");
    }
}

public class Main {
    public static void main(String[] args) {
        MyThread t = new MyThread();
        t.start();
    }
}
```

2. **Implementing `Runnable` Interface**

```
class MyRunnable implements Runnable {
    public void run() {
        System.out.println("Thread is running");
    }
}

public class Main {
    public static void main(String[] args) {
        Thread t = new Thread(new MyRunnable());
        t.start();
    }
}
```

5. SOLID Principles (Best Practices)

Principle	Description	Example
Single Responsibility	A class should have one job.	`User` class handles only user data, not logging.
Open-Closed	Open for extension, closed for modification.	Use interfaces for new features.
Liskov Substitution	Subclasses should extend without breaking.	`Square` should not inherit `Rectangle` if it changes behavior.
Interface Segregation	Avoid fat interfaces.	Split `Machine` into `Printer` and `Scanner`.
Dependency Inversion	Depend on abstractions, not concretions.	Use `Database` interface instead of `MySQLDatabase`.