🔹 OOPs in Java – Definition

OOPs is a programming paradigm based on the concept of **objects**, which contain **fields (attributes/properties)** and **methods (behaviors)**.
It helps in **reusability, modularity, scalability, and security**.

The **four main pillars of OOPs in Java** are:

1. **Encapsulation** – Wrapping data (variables) and code (methods) together in a single unit (class).
2. **Abstraction** – Hiding internal implementation details and showing only essential features.
3. **Inheritance** – One class acquiring properties and behaviors from another class (code reusability).
4. **Polymorphism** – Performing the same action in different ways (method overloading/overriding).

🔹 Real-Time Example – Banking System

1. Encapsulation Example

We hide account details and provide only getters/setters.

```
class BankAccount {
    private String accountNumber;
    private double balance;

    // Constructor
    public BankAccount(String accountNumber, double balance) {
        this.accountNumber = accountNumber;
        this.balance = balance;
    }

    // Getter
    public double getBalance() {
        return balance;
    }

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

    public void withdraw(double amount) {
        if(amount <= balance) {
            balance -= amount;
        } else {
            System.out.println("Insufficient balance!");
        }
    }
}
```

---

2. Abstraction Example

We use an abstract class Payment and let subclasses decide the implementation.

```
abstract class Payment {
    abstract void makePayment(double amount);
}

class UpiPayment extends Payment {
    void makePayment(double amount) {
        System.out.println("Paid " + amount + " via UPI");
    }
}

class CardPayment extends Payment {
    void makePayment(double amount) {
        System.out.println("Paid " + amount + " via Card");
    }
}
```

---

3. Inheritance Example

SavingsAccount inherits from BankAccount.

```
class SavingsAccount extends BankAccount {
   private double interestRate;

   public SavingsAccount(String accountNumber, double balance, double interestRate) {
      super(accountNumber, balance);  // call parent constructor
      this.interestRate = interestRate;
   }

   public void addInterest() {
      double interest = getBalance() * interestRate / 100;
      deposit(interest);
   }
}
```

---

4. Polymorphism Example

Method **overloading** and **overriding** in action.

```
class Calculator {
   // Compile-time polymorphism (overloading)
   int add(int a, int b) {
      return a + b;
   }
   double add(double a, double b) {
      return a + b;
   }
}

class Loan {
   // Runtime polymorphism (overriding)
   void calculateInterest() {
      System.out.println("Generic Loan interest");
   }
}

class HomeLoan extends Loan {
    @Override
    void calculateInterest() {
        System.out.println("Home Loan interest rate = 7%");
    }
}

class CarLoan extends Loan {
    @Override
    void calculateInterest() {
        System.out.println("Car Loan interest rate = 9%");
    }
}
```

✅ **Summary**

- **Encapsulation** → Data hiding (BankAccount).
- **Abstraction** → Hiding implementation (Payment).
- **Inheritance** → Reusability (SavingsAccount).
- **Polymorphism** → Multiple forms (Calculator and Loan).

🔹 Types of Inheritance in Java

1. Single Inheritance

One class inherits from another.

```
// Parent class
class Employee {
    String name = "Ravi";
    void display() {
        System.out.println("Employee Name: " + name);
    }
}

// Child class
class Developer extends Employee {
    void work() {
        System.out.println(name + " is coding...");
    }
}

public class SingleInheritanceExample {
    public static void main(String[] args) {
        Developer d = new Developer();
        d.display();   // from parent
        d.work();      // from child
    }
}
```

---

2. Multilevel Inheritance

A class inherits from a derived class (grandparent → parent → child).

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

class Employee extends Person {
    void work() {
        System.out.println("Employee is working...");
    }
}

class Manager extends Employee {
    void manage() {
        System.out.println("Manager is managing team...");
    }
}

public class MultilevelInheritanceExample {
    public static void main(String[] args) {
        Manager m = new Manager();
        m.eat();   // from Person
        m.work();  // from Employee
        m.manage();// from Manager
    }
}
```

---

3. Hierarchical Inheritance

Multiple classes inherit from a single parent.

```
class Vehicle {
    void start() {
        System.out.println("Vehicle is starting...");
    }
}

class Car extends Vehicle {
    void drive() {
        System.out.println("Car is driving...");
    }
}

class Bike extends Vehicle {
    void ride() {
        System.out.println("Bike is riding...");
    }
}

public class HierarchicalInheritanceExample {
    public static void main(String[] args) {
        Car c = new Car();
        c.start();
        c.drive();

        Bike b = new Bike();
        b.start();
        b.ride();
    }
}
```

---

4. Multiple Inheritance (via Interfaces)

Java does not allow multiple inheritance with classes but supports it using interfaces.

```
interface Flyable {
    void fly();
}

interface Swimmable {
    void swim();
}

class Duck implements Flyable, Swimmable {
    public void fly() {
        System.out.println("Duck is flying...");
    }
    public void swim() {
        System.out.println("Duck is swimming...");
    }
}

public class MultipleInheritanceExample {
    public static void main(String[] args) {
        Duck d = new Duck();
        d.fly();
        d.swim();
    }
}
```

---

5. Hybrid Inheritance

A mix of two or more types of inheritance (achieved using **classes + interfaces** since Java doesn’t support it directly).

```
interface Musician {
    void playMusic();
}

class Person {
    void speak() {
        System.out.println("Person is speaking...");
    }
}

class Singer extends Person implements Musician {
    public void playMusic() {
        System.out.println("Singer is singing a song...");
    }
}

class Guitarist extends Person implements Musician {
    public void playMusic() {
        System.out.println("Guitarist is playing guitar...");
    }
}
```

```
public class HybridInheritanceExample {
    public static void main(String[] args) {
        Singer s = new Singer();
        s.speak();
        s.playMusic();

        Guitarist g = new Guitarist();
        g.speak();
        g.playMusic();
    }
}
```

✅ Summary Table

Inheritance Type	Example
Single Inheritance	Employee → Developer
Multilevel Inheritance	Person → Employee → Manager
Hierarchical	Vehicle → Car, Bike
Multiple	Duck implements Flyable, Swimmable
Hybrid	Mix of class + interfaces (Singer, Guitarist)
