জাভা একটি **অবজেক্ট-ওরিয়েন্টেড প্রোগ্রামিং (OOP)** ল্যাঙ্গুয়েজ। OOP এর মূল ধারণাগুলি হলো **ক্লাস**, **অবজেক্ট**, **ইনহেরিটেন্স**, **পলিমরফিজম**, **অ্যাবস্ট্রাকশন**, এবং **এনক্যাপসুলেশন**। নিচে এই ধারণাগুলি এবং জাভাতে এগুলি বাস্তবায়নের জন্য ব্যবহৃত মেথড ও কনসেপ্টগুলি বাংলায় বিস্তারিত ব্যাখ্যা দেওয়া হলো:

**১. ক্লাস (Class):**

ক্লাস হলো একটি ব্লুপ্রিন্ট বা টেমপ্লেট যা অবজেক্ট তৈরি করতে ব্যবহৃত হয়। এটি ডেটা (প্রপার্টি) এবং মেথড (ফাংশন) ধারণ করে।

**উদাহরণ:**

java

Copy

class Car {

// প্রপার্টি (ডেটা)

String brand;

int year;

// মেথড (ফাংশন)

void start() {

System.out.println("Car is starting...");

}

}

**২. অবজেক্ট (Object):**

অবজেক্ট হলো ক্লাসের একটি ইনস্ট্যান্স। এটি ক্লাসের প্রপার্টি এবং মেথড ব্যবহার করে।

**উদাহরণ:**

java

Copy

public class Main {

public static void main(String[] args) {

Car myCar = new Car(); // অবজেক্ট তৈরি

myCar.brand = "Toyota";

myCar.year = 2020;

myCar.start(); // আউটপুট: Car is starting...

}

}

**৩. ইনহেরিটেন্স (Inheritance):**

ইনহেরিটেন্স হলো একটি ক্লাস অন্য ক্লাসের প্রপার্টি এবং মেথড ইনহেরিট (উত্তরাধিকার সূত্রে পাওয়া) করার ক্ষমতা। এটি কোড রিইউজেবিলিটি (পুনরায় ব্যবহারযোগ্যতা) বাড়ায়।

**উদাহরণ:**

java

Copy

class Vehicle {

void move() {

System.out.println("Vehicle is moving...");

}

}

class Car extends Vehicle { // Car ক্লাস Vehicle ক্লাস ইনহেরিট করে

void start() {

System.out.println("Car is starting...");

}

}

public class Main {

public static void main(String[] args) {

Car myCar = new Car();

myCar.move(); // আউটপুট: Vehicle is moving...

myCar.start(); // আউটপুট: Car is starting...

}

}

**৪. পলিমরফিজম (Polymorphism):**

পলিমরফিজম হলো একই মেথডের বিভিন্ন ফর্ম। এটি দুটি ভাবে বাস্তবায়িত হয়: **মেথড ওভারলোডিং** এবং **মেথড ওভাররাইডিং**।

**ক. মেথড ওভারলোডিং (Method Overloading):**

একই নামের মেথড, কিন্তু বিভিন্ন প্যারামিটার লিস্ট।

**উদাহরণ:**

java

Copy

class Calculator {

int add(int a, int b) {

return a + b;

}

double add(double a, double b) {

return a + b;

}

}

public class Main {

public static void main(String[] args) {

Calculator calc = new Calculator();

System.out.println(calc.add(2, 3)); // আউটপুট: 5

System.out.println(calc.add(2.5, 3.5)); // আউটপুট: 6.0

}

}

**খ. মেথড ওভাররাইডিং (Method Overriding):**

সাবক্লাসে সুপারক্লাসের মেথড রিডিফাইন করা।

**উদাহরণ:**

java

Copy

class Animal {

void sound() {

System.out.println("Animal is making a sound");

}

}

class Dog extends Animal {

@Override

void sound() {

System.out.println("Dog is barking");

}

}

public class Main {

public static void main(String[] args) {

Animal myDog = new Dog();

myDog.sound(); // আউটপুট: Dog is barking

}

}

**৫. অ্যাবস্ট্রাকশন (Abstraction):**

অ্যাবস্ট্রাকশন হলো জটিল ডিটেইলস লুকিয়ে শুধুমাত্র প্রয়োজনীয় তথ্য প্রদর্শন করা। এটি **অ্যাবস্ট্রাক্ট ক্লাস** এবং **ইন্টারফেস** ব্যবহার করে বাস্তবায়িত হয়।

**ক. অ্যাবস্ট্রাক্ট ক্লাস:**

অ্যাবস্ট্রাক্ট ক্লাসে অ্যাবস্ট্রাক্ট মেথড (বডি ছাড়া মেথড) এবং কনক্রিট মেথড (বডি সহ মেথড) থাকতে পারে।

**উদাহরণ:**

java

Copy

abstract class Animal {

abstract void sound(); // অ্যাবস্ট্রাক্ট মেথড

void sleep() {

System.out.println("Animal is sleeping");

}

}

class Dog extends Animal {

@Override

void sound() {

System.out.println("Dog is barking");

}

}

public class Main {

public static void main(String[] args) {

Animal myDog = new Dog();

myDog.sound(); // আউটপুট: Dog is barking

myDog.sleep(); // আউটপুট: Animal is sleeping

}

}

**খ. ইন্টারফেস:**

ইন্টারফেস শুধুমাত্র অ্যাবস্ট্রাক্ট মেথড ধারণ করে (জাভা ৮ এর আগে)। এটি মাল্টিপল ইনহেরিটেন্স সাপোর্ট করে।

**উদাহরণ:**

java

Copy

interface Animal {

void sound(); // অ্যাবস্ট্রাক্ট মেথড

}

class Dog implements Animal {

@Override

public void sound() {

System.out.println("Dog is barking");

}

}

public class Main {

public static void main(String[] args) {

Animal myDog = new Dog();

myDog.sound(); // আউটপুট: Dog is barking

}

}

**৬. এনক্যাপসুলেশন (Encapsulation):**

এনক্যাপসুলেশন হলো ডেটা এবং মেথড কে একটি ইউনিট (ক্লাস) এ আবদ্ধ করা এবং ডেটা অ্যাক্সেস কন্ট্রোল করা। এটি **প্রাইভেট প্রপার্টি** এবং **পাবলিক মেথড** ব্যবহার করে বাস্তবায়িত হয়।

**উদাহরণ:**

java

Copy

class Person {

private String name; // প্রাইভেট প্রপার্টি

// পাবলিক মেথড (getter)

public String getName() {

return name;

}

// পাবলিক মেথড (setter)

public void setName(String name) {

this.name = name;

}

}

public class Main {

public static void main(String[] args) {

Person person = new Person();

person.setName("John");

System.out.println(person.getName()); // আউটপুট: John

}

}

**৭. কনস্ট্রাক্টর (Constructor):**

কনস্ট্রাক্টর হলো একটি বিশেষ মেথড যা অবজেক্ট তৈরি করার সময় কল হয়। এটি ক্লাসের নামের মতোই নামে এবং কোনো রিটার্ন টাইপ নেই।

**উদাহরণ:**

java

Copy

class Car {

String brand;

int year;

// কনস্ট্রাক্টর

Car(String brand, int year) {

this.brand = brand;

this.year = year;

}

void display() {

System.out.println("Brand: " + brand + ", Year: " + year);

}

}

public class Main {

public static void main(String[] args) {

Car myCar = new Car("Toyota", 2020);

myCar.display(); // আউটপুট: Brand: Toyota, Year: 2020

}

}

**সারসংক্ষেপ:**

| **OOP ধারণা** | **বর্ণনা** | **উদাহরণ** |
| --- | --- | --- |
| **ক্লাস** | অবজেক্ট তৈরি করার ব্লুপ্রিন্ট | class Car { ... } |
| **অবজেক্ট** | ক্লাসের ইনস্ট্যান্স | Car myCar = new Car(); |
| **ইনহেরিটেন্স** | একটি ক্লাস অন্য ক্লাসের প্রপার্টি এবং মেথড ইনহেরিট করে | class Car extends Vehicle { ... } |
| **পলিমরফিজম** | একই মেথডের বিভিন্ন ফর্ম | মেথড ওভারলোডিং এবং ওভাররাইডিং |
| **অ্যাবস্ট্রাকশন** | জটিল ডিটেইলস লুকিয়ে প্রয়োজনীয় তথ্য প্রদর্শন | abstract class Animal { ... } |
| **এনক্যাপসুলেশন** | ডেটা এবং মেথড কে একটি ইউনিটে আবদ্ধ করা | private String name; |
| **কনস্ট্রাক্টর** | অবজেক্ট তৈরি করার সময় কল হওয়া বিশেষ মেথড | Car(String brand, int year) { ... } |

এই ধারণাগুলি ব্যবহার করে আপনি জাভাতে অবজেক্ট-ওরিয়েন্টেড প্রোগ্রামিং এর সুবিধা নিতে পারবেন এবং কোডকে মডুলার, রিইউজেবল, এবং মেইনটেইনেবল করতে পারবেন। 😊