O que é OOP? OOP (Programação Orientada a Objetos) é um paradigma de programação que organiza o código em torno de objetos, permitindo a reutilização e a modularidade do código, além de facilitar a representação do mundo real e o trabalho em equipe.

Assim como toda boa estrutura, a OOP tem seus pilares. Sendo eles:

- **Encapsulamento**: Consiste em agrupar dados e métodos relacionados em objetos, ocultando a implementação interna e permitindo o acesso controlado aos mesmos.
- **Herança**: Permite criar novas classes com base em classes existentes, herdando seus atributos e comportamentos, promovendo a reutilização de código e a criação de hierarquias de classes.
- **Polimorfismo**: Permite que um objeto possa ser referenciado de várias maneiras, possibilitando o uso de um mesmo código para diferentes tipos de objetos, aumentando a flexibilidade e a extensibilidade do sistema.
- **Abstração**: Permite representar conceitos complexos do mundo real em termos de objetos e suas interações, simplificando a implementação e facilitando a compreensão do sistema.

Exemplo - Encapsulamento:

```
public class Car {
    private String model;
    private int year;

    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        this.model = model;
    }

    public int getYear() {
        return year;
    }

    public void setYear(int year) {
        this.year = year;
    }
}
```

No exemplo acima, a classe Car encapsula os atributos model e year, tornando-os privados (private). Os métodos getModel() e getYear() permitem acessar os valores dos atributos, enquanto setModel() e setYear() permitem modificar esses valores. O encapsulamento protege os atributos, controlando o acesso a eles através de métodos.

Exemplo - Herança:

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

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

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.eat(); // Chamando método da classe Animal através da herança
        dog.bark(); // Chamando método específico da classe Dog
    }
}
```

No exemplo acima, a classe Dog herda da classe Animal. Isso significa que a classe Dog possui todos os métodos e atributos da classe Animal, incluindo o método eat(). Além disso, a classe Dog adiciona o método bark(). Ao criar um objeto Dog e chamá-lo, é possível acessar tanto o método eat() da classe Animal quanto o método bark() da classe Dog.

Exemplo - Polimorfismo:

```
public class Animal {
    public void makeSound() {
        System.out.println("The animal makes a sound.");
    }
}

public class Dog extends Animal {
    public void makeSound() {
        System.out.println("The dog barks.");
    }
}

public class Cat extends Animal {
    public void makeSound() {
        System.out.println("The cat meows.");
    }
}

public class Main {
```

```
    public static void main(String[] args) {
        Animal animal1 = new Dog();
        Animal animal2 = new Cat();

        animal1.makeSound(); // Chama o método makeSound() da classe Dog
        animal2.makeSound(); // Chama o método makeSound() da classe Cat
    }
}
```

No exemplo acima, as classes Dog e Cat herdam da classe Animal e substituem o método makeSound(). Ao criar objetos do tipo Dog e Cat e chamar o método makeSound(), o comportamento específico de cada classe é executado, demonstrando o polimorfismo.

Exemplo - Abstração:

```
public abstract class Shape {
    public abstract double calculateArea();
}

public class Circle extends Shape {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    public double calculateArea() {
        return Math.PI * radius * radius;
    }
}

public class Rectangle extends Shape {
    private double width;
    private double height;

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    public double calculateArea() {
        return width * height;
    }
}

public class Main {
    public static void main(String[] args) {
        Shape circle = new Circle(5.0);
```

```
        Shape rectangle = new Rectangle(4.0, 6.0);

        System.out.println("Circle area: " + circle.calculateArea());
        System.out.println("Rectangle area: " + rectangle.calculateArea());
    }
}
```

No exemplo acima, a classe abstrata Shape define um método abstrato calculateArea(), que é implementado pelas classes concretas Circle e Rectangle. A classe Shape representa uma abstração genérica de uma forma geométrica, enquanto as classes Circle e Rectangle são implementações específicas dessa abstração. Ao chamar o método calculateArea() em objetos das classes Circle e Rectangle, a área é calculada de acordo com a fórmula correspondente a cada forma.

- **Classes**: Uma classe é um modelo ou uma descrição que define características e comportamentos comuns a um grupo de objetos.
- **Atributos**: Os atributos são variáveis ou dados que representam as características de uma classe. Eles armazenam informações sobre os objetos criados a partir dessa classe.
- **Métodos**: Os métodos são funções ou blocos de código associados a uma classe. Eles representam os comportamentos ou ações que os objetos dessa classe podem realizar.
- **Objetos**: Os objetos são instâncias específicas de uma classe. Eles são criados a partir de uma classe e possuem atributos e comportamentos definidos por essa classe.
- **Instanciação**: A instanciação é o processo de criar um objeto específico a partir de uma classe. É quando um objeto real é criado na memória, usando a definição da classe como base.

Exemplo - Classe:

```
public class Car {
    // código da classe Car
}
```

Exemplo - Atributos:

```
public class Car {
    String model;
    int year;
    double price;
}
```

Exemplo - Métodos:

```
public class Car {
    public void startEngine() {
        // código para iniciar o motor do carro
    }
```

```
    public void accelerate() {
        // código para acelerar o carro
    }

    public void brake() {
        // código para frear o carro
    }
}
```

Exemplo - Objetos:

```
public class Main {
    public static void main(String[] args) {
        Car car1 = new Car();
        Car car2 = new Car();
    }
}
```

Exemplo - Instanciação:

```
public class Main {
    public static void main(String[] args) {
        Car car1 = new Car();
        car1.model = "Honda Civic";
        car1.year = 2022;
        car1.price = 25000.0;

        Car car2 = new Car();
        car2.model = "Ford Mustang";
        car2.year = 2021;
        car2.price = 45000.0;
    }
}
```