Mis disculpas por el error. Aquí tienes el código con las breves explicaciones para cada concepto, junto con el índice:

Índice

---

1. Encapsulación

La **encapsulación** agrupa datos y métodos que operan sobre esos datos dentro de una clase. Esto protege los detalles internos del objeto y asegura que los datos solo se manipulen de manera controlada a través de métodos públicos.

Ejemplo:

```
class Persona {
    private String nombre;
    private int edad;

    // Constructor
    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    // Getter para nombre
    public String getNombre() {
        return nombre;
    }

    // Setter para nombre
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    // Getter para edad
    public int getEdad() {
        return edad;
    }

    // Setter para edad
    public void setEdad(int edad) {
        if (edad > 0) {
            this.edad = edad;
```

```
        }
    }
}
```

---

2. Abstracción

La **abstracción** permite representar los objetos del mundo real en términos de clases y objetos, ocultando la complejidad innecesaria y mostrando solo los detalles esenciales.

Ejemplo:

```
abstract class Vehiculo {
    protected String marca;

    public Vehiculo(String marca) {
        this.marca = marca;
    }

    // Método abstracto
    public abstract void encender();
}

class Coche extends Vehiculo {

    public Coche(String marca) {
        super(marca);
    }

    @Override
    public void encender() {
        System.out.println("El coche de marca " + marca + " está encendido.");
    }
}

class Moto extends Vehiculo {

    public Moto(String marca) {
        super(marca);
    }

    @Override
    public void encender() {
        System.out.println("La moto de marca " + marca + " está encendida.");
    }
}
```

---

3. Herencia

La **herencia** permite que una clase (subclase) herede atributos y métodos de otra clase (superclase), promoviendo la reutilización de código.

Ejemplo:

```
class Animal {
    public void sonido() {
        System.out.println("El animal hace un sonido.");
    }
}

class Perro extends Animal {
    @Override
    public void sonido() {
        System.out.println("El perro ladra.");
    }
}

class Gato extends Animal {
    @Override
    public void sonido() {
        System.out.println("El gato maúlla.");
    }
}
```

---

4. Polimorfismo

El **polimorfismo** permite que un objeto tome múltiples formas. El mismo método puede tener diferentes implementaciones dependiendo del contexto.

Ejemplo:

```
class Empleado {
    public void trabajar() {
        System.out.println("El empleado está trabajando.");
    }
}

class Desarrollador extends Empleado {
    @Override
    public void trabajar() {
        System.out.println("El desarrollador está escribiendo código.");
    }
}

class Diseñador extends Empleado {
    @Override
    public void trabajar() {
```

```
        System.out.println("El diseñador está creando gráficos.");
    }
}

public class Empresa {
    public static void main(String[] args) {
        Empleado empleado1 = new Desarrollador();
        Empleado empleado2 = new Diseñador();

        empleado1.trabajar(); // Salida: El desarrollador está escribiendo
código.
        empleado2.trabajar(); // Salida: El diseñador está creando gráficos.
    }
}
```
