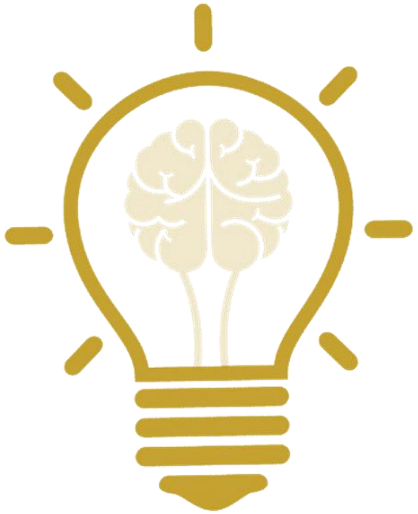
PROGRAMACIÓN EN JAVA

Materia PJA-001 material creado por el Ing. Jim Requena

Programación Orientada a Objetos en JAVA
Parte Final
(Clases Abstractas e Interfaces)



Materia PJA-001 material creado por el Ing. Jim Requena

Objetivos de la sesión

- Comprender el concepto de clases abstractas en Java
- Aprender a crear y utilizar interfaces
- Entender la diferencia entre clases abstractas e interfaces
- Familiarizarnos un poquito con las expresiones lambda y su uso con interfaces funcionales
- Practicar con ejercicios de clases abstractas e interfaces

Clases Abstractas

- No se pueden instanciar directamente
- Pueden contener métodos abstractos y concretos
- Se utilizan como base para otras clases

Sintaxis:

```
public abstract class ClaseAbstracta {
    // Métodos abstractos y concretos
}
```

iTDS | Profesionales en Educación

Ejemplo Abstractas en archivos separados 1/2

```
public abstract class Figura {
        protected String nombre;

        public Figura(String nombre) {
                this.nombre = nombre;
        }

        public abstract double calcularArea();

        public void mostrarNombre() {
                System.out.println("Esta figura es un " + nombre);
        }
}
```

iTDS | Profesionales en Educación

Ejemplo Abstractas en archivos separados 2/2

```
public class Circulo extends Figura {
        private double radio;

        public Circulo(double radio) {
                super("Círculo"); this.radio = radio;
        }
        @Override
        public double calcularArea() {
                return Math.PI * radio * radio;
        }
}
```

Interfaces

- Define un contrato de métodos que una clase debe implementar
- Todos los métodos son implícitamente públicos y abstractos
- Desde Java 8, puede contener métodos default y static con implementación

Ejemplo:

```
public interface Dibujable {
        void dibujar();

        default void mostrarInformacion() {
                System.out.println("Este es un objeto dibujable");
        }
}
```

Rectangulo.java:

```
public class Rectangulo extends Figura implements Dibujable {
        private double base;
        private double altura;

        public Rectangulo(double base, double altura) {
                super("Rectángulo");
                this.base = base;
                this.altura = altura;
        }
        @Override
        public double calcularArea() {
                return base * altura;
        }
        @Override
public void dibujar() {
                System.out.println("Dibujando un rectángulo");
        }
 }
```

Diferencias entre Clases Abstractas e Interfaces

1. Una clase puede implementar múltiples interfaces, pero solo puede extender una clase abstracta.

2. Las clases abstractas pueden tener constructores, las interfaces no.

3. Las clases abstractas pueden tener métodos con implementación por defecto y campos no estáticos.

4. Las interfaces solo pueden tener métodos abstractos, default, o static, y campos estáticos finales.

Diferencias entre Clases Abstractas e Interfaces

1. Una clase puede implementar múltiples interfaces, pero solo puede extender una clase abstracta.

2. Las clases abstractas pueden tener constructores, las interfaces no.

3. Las clases abstractas pueden tener métodos con implementación por defecto y campos no estáticos.

4. Las interfaces solo pueden tener métodos abstractos, default, o static, y campos estáticos finales.

Interfaces Funcionales y Expresiones Lambda

1. Interfaces funcionales: interfaces con un solo método abstracto
2. Se pueden implementar usando expresiones lambda

Ejemplo:

```
@FunctionalInterface
interface Calculable {
    double calcular(double a, double b);
}

public class Main {
    public static void main(String[] args) {
        // Implementación usando una expresión lambda
        Calculable suma = (a, b) -> a + b;
        System.out.println("Suma: " + suma.calcular(5, 3)); //

        //Otra implementación lambda Calculable multiplicacion = (a, b) -> a * b;
        System.out.println("Multiplicación: " + multiplicacion.calcular(4, 7));
        }
    }
```

Ejercicio práctico

1. Definir una clase abstracta Empleado con métodos abstractos **calcularSalario()** y **mostrarDetalles()**

2. Crear subclases **EmpleadoTiempoCompleto** y **EmpleadoTiempoParcial**

3. Definir una interfaz **Bonificable** con método **calcularBono()**

4. Implementar la interfaz en **EmpleadoTiempoCompleto**

5. Usar una expresión lambda para implementar un comparador de empleados por salario

Tarea

1. TareaAñade una nueva subclase `EmpleadoPorContrato` que extienda de `Empleado`

2. Crea una interfaz `Evaluable` con un método `evaluarDesempeño()`

3. Implementa `Evaluable` en todas las subclases de `Empleado`

4. Añade un método default `aumentarSalario(double porcentaje)` a la interfaz `Evaluable`

5. Implementa un comparator usando una expresión lambda para ordenar los empleados por nombre

Recursos adicionales

- Documentación oficial de Java sobre Clases Abstractas: docs.oracle.com/javase/tutorial/java/IandI/abstract.html
- Documentación oficial de Java sobre Interfaces: docs.oracle.com/javase/tutorial/java/IandI/createinterface.html
- Tutorial sobre Expresiones Lambda en Java: www.baeldung.com/java-8-lambda-expressions-tips

Gracias!

Convenciones de Java

- Documentación oficial de Oracle sobre convenciones de código Java: https://www.oracle.com/java/technologies/javase/codeconventions-contents.html

- Google Java Style: https://google.github.io/styleguide/javaguide.html

- Baeldung - Java Naming Conventions: https://www.baeldung.com/java-naming-conventions

Ejemplo Figura 1/2

Para ejecutar todo el código en un solo archivo (por ejemplo, para propósitos de demostración o en un entorno limitado)
Ejemplo:

```
abstract class Figura {
    public abstract double calcularArea();
}

class Circulo extends Figura {
    private double radio;
    public Circulo(double radio) {
        this.radio = radio;
    }

    @Override
    public double calcularArea() {
        return Math.PI * radio * radio;
    }
}
```

```
class Rectangulo extends Figura {
    private double base;
    private double altura;

    public Rectangulo(double base, double altura) {
        this.base = base;
        this.altura = altura;
    }

    @Override
    public double calcularArea() {
        return base * altura;
    }
}
```

Ejemplo Figura 2/2

```
class Triangulo extends Figura {
    private double base;
    private double altura;

    public Triangulo(double base, double altura) {
        this.base = base;
        this.altura = altura;
    }

    @Override
    public double calcularArea() {
        return 0.5 * base * altura;
    }
}
```

```
public class Figuras {
    public static void main(String[] args) {
        Figura[] figuras = new Figura[3];
        figuras[0] = new Circulo(5);
        figuras[1] = new Rectangulo(4, 6);
        figuras[2] = new Triangulo(3, 4);
    for (Figura figura : figuras) {
        System.out.println("Área: " + figura.calcularArea());
        }
    }
}
```

Convenciones de Java

Es importante notar que, aunque este enfoque funciona para ejemplos pequeños o demostraciones, en proyectos reales y más grandes, se recomienda seguir la convención de una clase por archivo, especialmente para clases públicas. Esto mejora la organización del código, facilita el mantenimiento