ESCUELA DE
INGENIERÍA EN CIENCIAS Y SISTEMAS
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

ECYS | DIDT
DESARROLLO DE TRANSFERENCIA TECNOLÓGICA

Día, Fecha:	21/02/2023
Hora de inicio:	09:00

Introducción a la Programación y Computación 1 [Sección B]

Diego Fernando Cortez Lopez

Agenda

- Clase 4
- Dudas
- Ejemplo Práctico

Programación Orientada a Objetos (POO)

Clase 4

Modificadores

De Acceso y de No Acceso

Los modificadores se dividen en 2 tipos:

1. De clases
2. De atributos y métodos

Para ambos tipos de modificadores se subdividen en:

A. **De acceso:** Se utiliza para establecer niveles de acceso.
B. **De no acceso:** Son todos aquellos modificadores que no sirven para establecer niveles de acceso pero nos proveen de otras funcionalidades.

Modificadores de Acceso

De clases:

Modificador	Descripción
public	La clase es accesible para cualquier otra clase.
default	La clase sólo es accesible por clases en el mismo paquete. Esto se usa cuando no especifica un modificador.

Modificadores de Acceso

De atributos y métodos:

Modificador	Descripción
public	El código es accesible para todas las clases.
private	El código sólo es accesible dentro de la clase declarada.
protected	Solo se puede acceder al código en el mismo paquete. Esto se usa cuando no especifica un modificador.
default	Solo se puede acceder al código en el mismo paquete. Esto se usa cuando no especifica un modificador.

Modificadores de No Acceso

De clases:

Modificador	Descripción
final	La clase no puede ser heredada por otras clases.
abstract	La clase no se puede usar para crear objetos, para acceder a estos se debe heredar de otra clase.

Modificadores de Acceso

De atributos y métodos:

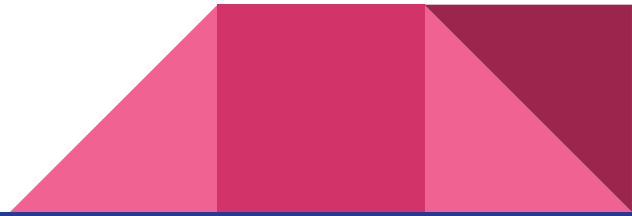
Modificador	Descripción
final	Los atributos y métodos no se pueden sobreescribir o modificar.
static	Los atributos y métodos pertenecen a la clase, en lugar de un objeto, es decir que se pueden utilizar sin instanciar un objeto de la clase.
abstract	Solo se puede usar en una clase abstracta y solo se puede usar en métodos. El método no tiene cuerpo, el cuerpo lo proporciona aquella clase que herede de la clase dueña de este método.

Pilares de la POO

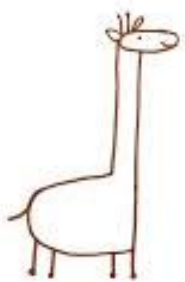
Pilares de POO

Los pilares de la programación orientada a objetos son cuatro conceptos fundamentales que se utilizan para diseñar y desarrollar sistemas de software basados en objetos. Estos pilares son:

- Abstracción
- Encapsulamiento
- Herencia
- Polimorfismo



Abstracción



Se refiere a la simplificación y generalización de los datos y comportamientos de los objetos.

La abstracción permite crear modelos más fáciles de entender y manipular, lo que ayuda a simplificar el desarrollo de software.

La abstracción se utiliza para identificar los atributos y métodos esenciales de los objetos y eliminar los detalles innecesarios.

Se puede utilizar para definir clases abstractas e interfaces y simplificar el desarrollo de software.

Clases Abstractas

```java
abstract class Figura {
    String nombre;

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

    public abstract double calcularArea();

    public void imprimirNombre() {
        System.out.println("Nombre de la figura: " + nombre);
    }
}
```

```java
class Circulo extends Figura {
    double radio;

    public Circulo(String nombre, double radio) {
        super(nombre);
        this.radio = radio;
    }

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

Notas:

- Las clases abstractas no se pueden utilizar para crear objetos, únicamente pueden ser heredadas por otras clases.
- Al heredar heredar de una clase abstracta se debe de sobrescribir cada uno de sus métodos abstractos de forma obligatoria utilizando la etiqueta @Override.
- Una clase abstracta puede tener métodos normales (con cuerpo) y abstractos (sin definición del cuerpo).
- Se debe colocar el modificador ‘abstract’ antes de una clase o método si se desea indicar que son abstractos. Los atributos no pueden ser abstractos.

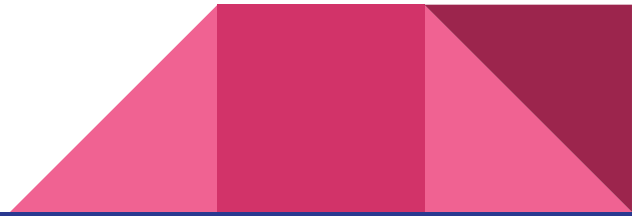
Interfaces

```java
public interface Calculadora {
    double PI = 3.1416;

    double sumar(double a, double b);
    double restar(double a, double b);
}
```

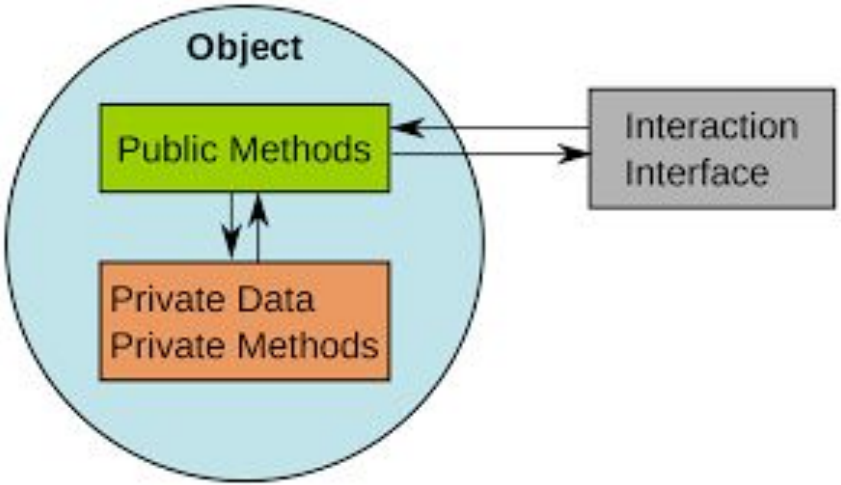
```java
public class CalculadoraBasica implements Calculadora {
    @Override
    public double sumar(double a, double b) {
        return a + b;
    }

    @Override
    public double restar(double a, double b) {
        return a - b;
    }

    public void imprimirPI() {
        System.out.println("El valor de PI es: " + Calculadora.PI);
    }
}
```

Notas:

- Los métodos de una interfaz no tienen cuerpo, estos son descritos en la clase que implementa la interfaz.
- Al implementar una interfaz se deben sobrescribir cada uno de sus métodos de forma obligatoria utilizando la etiqueta @Override.
- Al igual que las clases abstractas, las interfaces no se pueden utilizar para crear objetos.
- Los atributos declarados en una interfaz son por defecto públicos, estáticos y finales. Estos a diferencia de las clases no representan estados, sino que constantes.



Encapsulamiento



Se refiere a la técnica de ocultar la complejidad interna de los objetos y exponer solo una interfaz simplificada para interactuar con ellos.

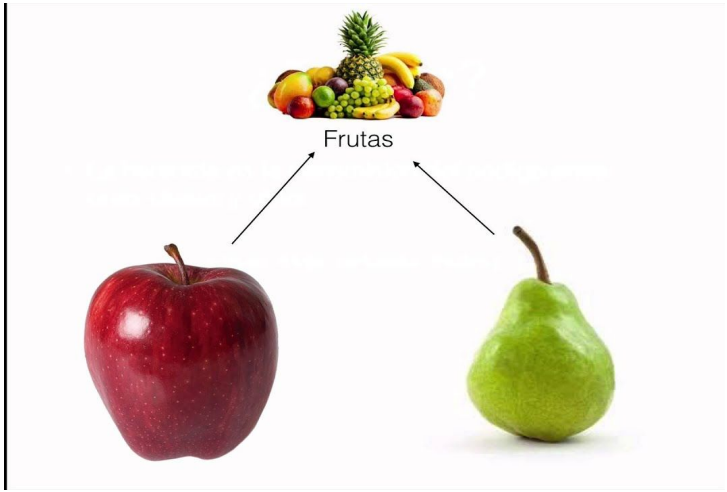
Mejora la seguridad y la integridad de los datos, y permite modificar la implementación interna del objeto sin afectar a los demás objetos.

El encapsulamiento se logra mediante la definición de acceso a los miembros del objeto

```java
public class Persona {
    private String nombre;

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}
```

- La clase Persona tiene un campo privado nombre, y los métodos getNombre() y setNombre() para acceder y modificar este campo desde fuera de la clase.
- La propiedad nombre está encapsulada dentro de la clase, y no se puede acceder directamente desde fuera de ella. En su lugar, se accede a través de los métodos de acceso definidos en la clase, lo que permite controlar el acceso a la propiedad y aplicar validaciones o restricciones si fuera necesario.

Herencia



Se refiere a la capacidad de una clase de heredar atributos y métodos de otra clase.

Permite crear clases que son variaciones o extensiones de otras clases existentes, lo que ayuda a simplificar y reutilizar el código.

La herencia se establece mediante la creación de una clase derivada o subclase a partir de una clase base o superclase. La subclase hereda los atributos y métodos de la superclase y puede agregar sus propios atributos y métodos, o modificar los existentes. La subclase también puede reutilizar los métodos heredados de la superclase, lo que permite ahorrar tiempo y reducir errores de programación.

```java
// Clase padre o superclase
class Persona {
  protected String nombre;
  protected int edad;

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

  public void saludar() {
    System.out.println("Hola, soy " + nombre + " y tengo " + edad + " años.");
  }
}
```

```java
// Clase hija o subclase
class Estudiante extends Persona {
  private int matricula;

  public Estudiante(String nombre, int edad, int matricula) {
    super(nombre, edad);
    this.matricula = matricula;
  }

  public void estudiar() {
    System.out.println(nombre + " está estudiando.");
  }
}
```

```java
// Uso de las clases
public class Main {
  public static void main(String[] args) {
    Persona p = new Persona("Juan", 30);
    p.saludar(); // Salida: "Hola, soy Juan y tengo 30 años."

    Estudiante e = new Estudiante("Pedro", 20, 12345);
    e.saludar(); // Salida: "Hola, soy Pedro y tengo 20 años."
    e.estudiar(); // Salida: "Pedro está estudiando."
  }
}
```

Polimorfismo

Se refiere a la capacidad de objetos de diferentes clases de responder al mismo mensaje de manera diferente.

Se logra mediante la definición de métodos en las clases bases o superclases y la implementación de estos métodos en las subclases de manera diferente.

Permite que las clases se implementen de manera diferente, pero aun puedan compartir una interfaz común.

Simplifica el código, mejora la modularidad y permite la flexibilidad en la programación

```java
public class Calculadora {
  public int sumar(int a, int b) {
    return a + b;
  }

  public double sumar(double a, double b) {
    return a + b;
  }

  public int sumar(int a, int b, int c) {
    return a + b + c;
  }

  public static void main(String[] args) {
    Calculadora c = new Calculadora();
    int suma1 = c.sumar(2, 3); // llamada al primer método sumar
    double suma2 = c.sumar(2.5, 3.7); // llamada al segundo método sumar
    int suma3 = c.sumar(2, 3, 4); // llamada al tercer método sumar
    System.out.println(suma1); // Salida: 5
    System.out.println(suma2); // Salida: 6.2
    System.out.println(suma3); // Salida: 9
  }
}
```

Sobrecarga de Métodos

Sobreescritura de Métodos

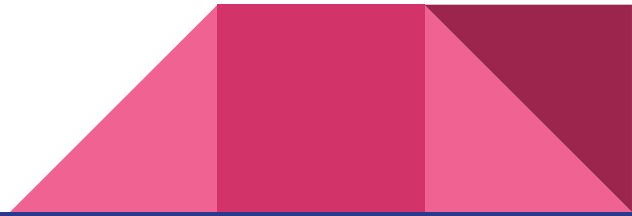
```java
// Clase padre o superclase
class Animal {
  public void hacerSonido() {
    System.out.println("El animal hace un sonido");
  }
}
```

```java
// Clase hija o subclase
class Gato extends Animal {
  @Override
  public void hacerSonido() {
    System.out.println("El gato maulla");
  }
}

// Clase hija o subclase
class Perro extends Animal {
  @Override
  public void hacerSonido() {
    System.out.println("El perro ladra");
  }
}
```

```java
// Uso de las clases
public class Main {
  public static void main(String[] args) {
    Animal a1 = new Gato();
    Animal a2 = new Perro();

    a1.hacerSonido(); // Salida: "El gato maulla"
    a2.hacerSonido(); // Salida: "El perro ladra"
  }
}
```

Recursos

- https://ferestrepoca.github.io/paradigmas-de-programacion/poo/poo_teoria/concepts.html
- Página de W3schools are de POO: https://www.w3schools.com/java/java_oop.asp



¿Dudas?