Programación II
Desarrollo en Java

Clase N° 6
Encapsulamiento.
Clases Abstractas.
Métodos abstractos.

Profesores Carolina Archuby
y Daniel Díaz.

UTNMDP
Regional Mar del Plata

TEMAS A DESARROLLAR

- Encapsulamiento.
- Clases Abstractas.
- Métodos abstractos.

ENCAPSULAMIENTO

El encapsulamiento en Java se refiere al **mecanismo de ocultar los detalles internos de una clase y proporcionar una interfaz pública para acceder a los datos y métodos de la clase.**

Ejemplo de encapsulamiento:

Tenemos que crear una **clase Libro** para manejar una biblioteca.
El atributo disponible está marcado como **privado**, lo que significa que el estado está encapsulado, su valor no puede ser accedido ni modificado directamente desde fuera de la clase Libro

```
private String titulo;
private String autor;
private boolean disponible;
```

En su lugar, se ofrecen métodos específicos para interactuar con este estado:

a) isDisponible() : Permite consultar si el libro está disponible para préstamo.

```java
// Método público para cambiar el estado de disponibilidad a no disponible
public void prestar() {
    if(disponible) {
        disponible = false;
        System.out.println("El libro ha sido prestado.");
    } else {
        System.out.println("El libro ya está prestado.");
    }
}
// Método público para cambiar el estado de disponibilidad a disponible
public void devolver() {
    if(!disponible) {
        disponible = true;
        System.out.println("El libro ha sido devuelto.");
    } else {
        System.out.println("Este libro no estaba prestado.");
    }
}
```

b) prestar() y c) devolver()
Son los únicos métodos que pueden modificar el estado de disponibilidad del libro, implementando reglas de negocio (como evitar marcar un libro ya prestado como prestado nuevamente).

Ventajas del encapsulamiento

=> **Ayuda a mantener la integridad de los datos de la clase**. En el ejemplo: Al encapsular el estado “disponible”, nos aseguramos de que sólo se modifique de maneras lógicamente coherentes, previniendo inconsistencias como un libro marcado erróneamente como disponible cuando está prestado.

=> **Permite que los cambios internos de la clase sean más fáciles de realizar**. En el ejemplo: Si en el futuro se necesita cambiar la lógica de cómo se gestiona la disponibilidad de un libro (por ejemplo, añadiendo un estado de "reservado"),este cambio se puede realizar internamente sin afectar a las clases que utilizan Libro.

=> **Facilita la reutilización del código**. En el ejemplo: Los usuarios de la clase Libro no necesitan entender cómo se gestiona internamente el estado de disponibilidad; solo necesitan interactuar con los métodos públicos proporcionados.

=> **Mejora la seguridad del código**.

Otro ejemplo

Clase Gato:

Definición de los atributos de clase :

```
private String nombre;
private String color;
private int edad;
private boolean isAdoptado;
```

Métodos públicos para acceder y modificar los datos :

```
public String getNombre() {
    return nombre;
}

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

*Debe crearse un método para obtener el dato (**get**) y otro para modificarlo (**set**) por cada uno de los atributos.*

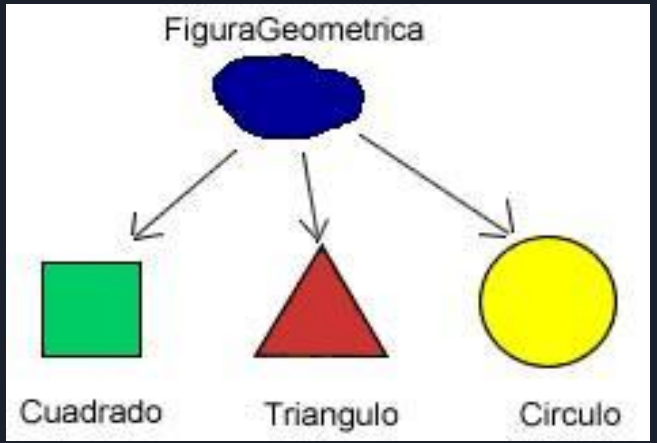
Utilizando los métodos públicos para acceder y modificar los datos del ejemplo anterior:

```
Gato gato1 = new Gato("Pelusa", "Gris", 1, true);

System.out.println("El nombre del gato es: " + gato1.getNombre());
System.out.println("La edad del gato es: " + gato1.getEdad());
System.out.println("El color del gato es: " + gato1.getColor());
System.out.println("Es adoptado : " + gato1.isAdoptado());

System.out.println("Modifique el nombre del gato");
gato1.setNombre("Rocky");
System.out.println("El nombre nuevo del gato es: " + gato1.getNombre());
```

¿QUÉ ES LA ABSTRACCIÓN?

* Es uno de los pilares de la POO

* Consiste en la capacidad de representar un objeto en su **forma más esencial, abstracta y general**, sin preocuparse por los detalles específicos de su implementación.



Ventajas de la abstracción

Permite **encapsular la complejidad de un objeto** y **separar la implementación de sus métodos públicos**, lo cual:

* facilita la **comprensión** y el **mantenimiento** del código

* permite una mayor **modularidad** y **reutilización** del mismo.

CLASES ABSTRACTAS

=> Una clase abstracta es una clase similar a una clase concreta (posee atributos y métodos), pero la gran diferencia es que es una clase que **NO SE PUEDE INSTANCIAR DIRECTAMENTE (no se pueden crear objetos de una clase abstracta)**, sino que se utiliza como una **plantilla o modelo** para definir las características y el comportamiento común de un conjunto de clases relacionadas.

=> Son clases que NO fueron pensadas para crear instancias de ellas sino exclusivamente para servir como superclase de otra.

CLASES ABSTRACTAS: Creación y reglas.

SE DEFINEN MEDIANTE **LA PALABRA CLAVE "ABSTRACT" EN LA DECLARACIÓN DE CLASE.**

CLASES ABSTRACTAS... ¿constructores??

Si las clases abstractas no se pueden instanciar... ¿pueden tener constructores?...

Sí!! **Es posible definir constructores en las superclases, pero no es posible crear instancias, sólo se usan para que a través de la herencia los utilicen las subclases.**

MÉTODOS ABSTRACTOS: ¿Qué son?

Un método abstracto se caracteriza por dos detalles:

=> Está precedido por la palabra clave abstract.

=> No tiene cuerpo y su encabezado termina con punto y coma

(es decir, se especifica su nombre, parámetros y tipo de retorno, pero no incluye código).

MÉTODOS ABSTRACTOS: ¿por qué?

Un método se define como abstracto

PORQUE EN ESE MOMENTO NO SE CONOCE

COMO HA DE SER SU IMPLEMENTACIÓN,

y serán las subclases de la clase abstracta

las responsables de darle “cuerpo”

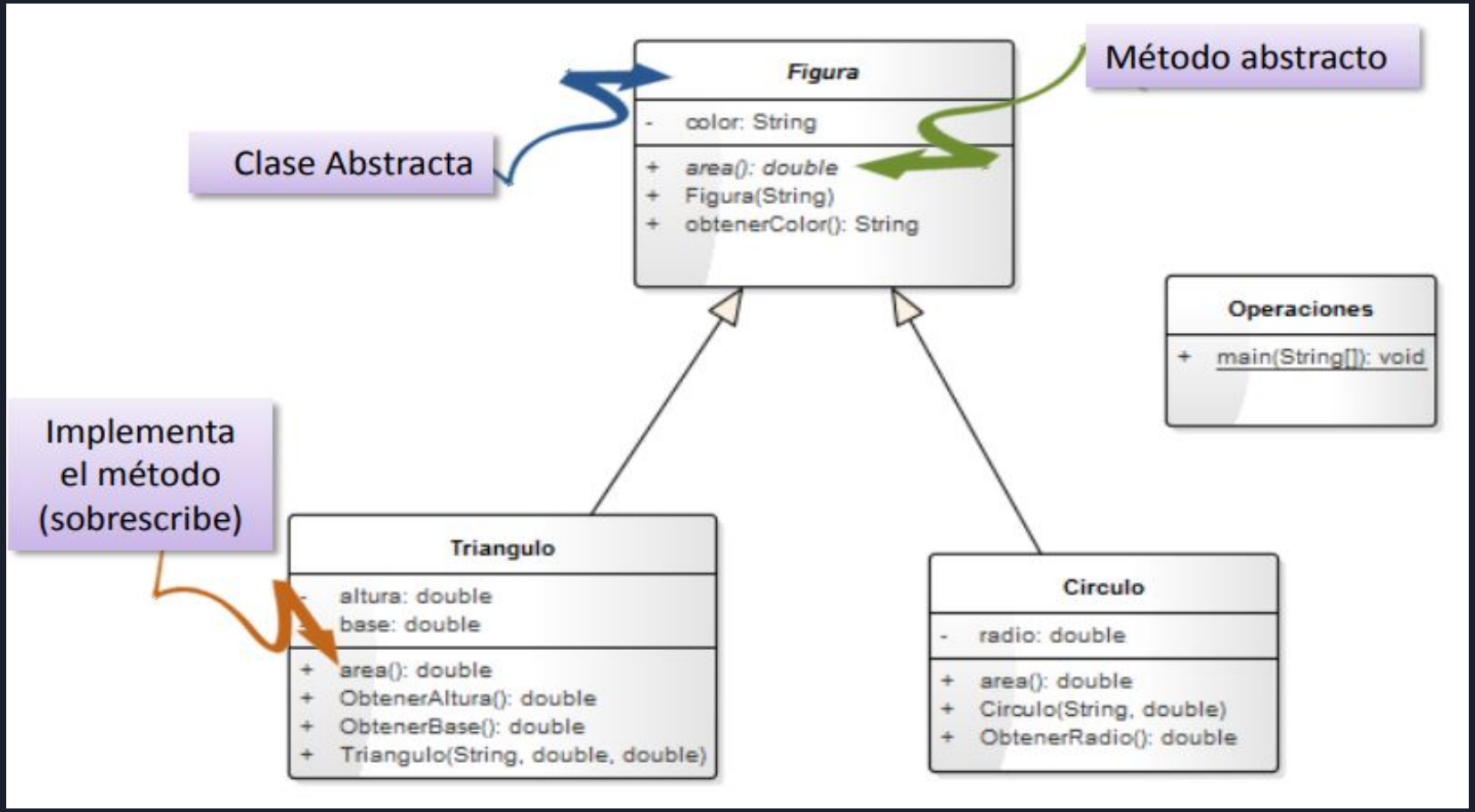
mediante la sobrescritura del mismo

MÉTODOS ABSTRACTOS: Reglas

=> Si un método se declara como abstracto, se debe marcar la clase como abstracta → No puede haber métodos abstractos en una clase concreta.

=> Los métodos abstractos deben implementarse en las subclases concretas (subclases) mediante @Override.
Las subclases de una clase abstracta están obligadas a sobrescribir todos los MÉTODOS ABSTRACTOS que heredan.
En caso de que no interese sobrescribir alguno de esos métodos, la subclase deberá ser declarada también abstracta.

Un ejemplo de clase abstracta y métodos abstractos



Un ejemplo de clase abstracta y métodos abstractos

```java
public abstract class Figura {
    private String color;
    public Figura(String color){
        this.color=color;
    }
    public String obtenerColor(){
        return color;
    }
    public abstract double area();
}
```

Implementa y sobrescribe el método abstracto **area()** (Polimorfismo)

```java
//Circulo.java
public class Circulo extends Figura{
    private int radio;
    public Circulo(int radio, String color){
        super(color);
        this.radio = radio;
    }
    public double area(){
        return Math.PI*radio*radio;
    }
    public int obtenerRadio(){
        return radio;
    }
}
```

Un ejemplo de clase abstracta y métodos abstractos

```
public class Triangulo extends Figura{
    private int base;
    private int altura;
    public Triangulo(int base,int altura,String color){
        super(color);
        this.base = base;
        this.altura = altura;
    }
    public double area(){
          return (base*altura)/2;
    }
    public int obtenerBase(){
        return base;
    }
    public int obtenerAltura(){
        return altura;
    }
}
```

Implementación del método **area()** aplicado en función a la fórmula para el Triángulo (Polimorfismo)

Representación de las clases abstractas en un UML