UD 4: PROGRAMACIÓN ORIENTADA A OBJETOS 2

POLIMORFISMO

1. INTRODUCCIÓN

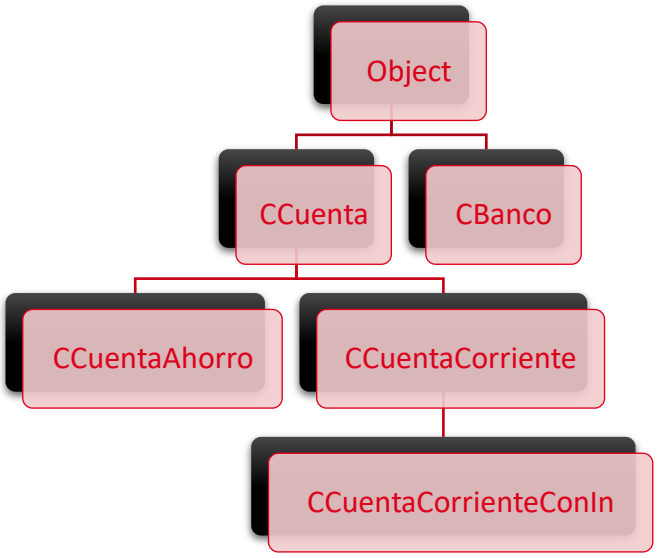
La facultad de llamar a una variedad de métodos utilizando exactamente el mismo medio de acceso, proporcionada por los métodos redefinidos en subclases, es a veces llamada polimorfismo.

La palabra "polimorfismo" significa "la facultad de asumir muchas formas", refiriéndose en Java a la posibilidad de llamar a muchos métodos diferentes utilizando una única sentencia.

Recordemos que cuando se invoca a un método que está definido en la superclase y redefinido en las subclases, la versión que se ejecuta depende de la clase del objeto referenciado, no del tipo de la variable que lo referencia.

Además, también "sabemos" que una referencia a una subclase puede ser convertida implícitamente por Java en una referencia a su superclase directa o indirecta. Esto significa que es posible referirse a un objeto de una subclase utilizando una variable del tipo de su superclase.

Supongamos la siguiente jerarquía de clases:



Pensemos en un array de referencias en la que cada elemento señale a un objeto de alguna de las subclases de la jerarquía construida anteriormente. ¿De qué tipo deben ser los elementos del array? Según nuestro ejemplo, deberían ser del tipo CCuenta, de esta forma el array podrá almacenar indistintamente, referencias a objetos de cualquiera de las subclases. Por ejemplo:

```
public class Test {
        public static void main(String[] args){
                CCuenta [] cliente= new Ccuenta [100];
                cliente [0]= new CCuentaAhorro ("cliente00", "3000123450", 10000, 2.5,30);
                cliente [1]= new CCuentaCorriente ("cliente01", "6000123450", 10000, 2.0,1.0,6);
                cliente [2]=new CCuentaCorrienteConIn ("cliente02", "4000123450", 10000, 3.5, 1.0,6);
```

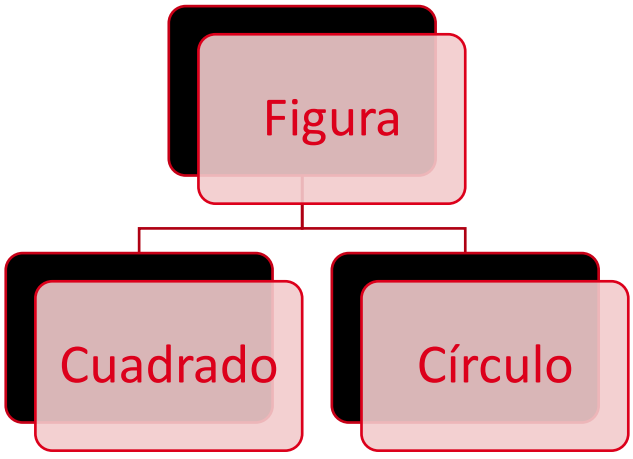
```
                for (int i=0; cliente [i]!=null; i++){
                        System.out.println(cliente[i].obtenerNombre ()+ ":");
                        System.out.println(cliente[i].intereses());
//Donde intereses pertenece a la clase CCuenta y está reescrito en las clases hijas CCuentaAhorro, //CCuentaCorriente y CCuentaCorrienteConIn
                }
        }
}
```

Se define un array cliente de tipo CCuenta con 100 elementos inicialmente con valor null. Después se crean varios objetos de algunas de las subclases y se almacenan en los primeros elementos del array. Aquí, Java realizará una conversión implícita del tipo de la referencia devuelta por new al tipo CCuenta. Finalmente mostramos el nombre del cliente y los intereses que le corresponderán por mes. Si tenemos el método intereses redefinido de formas diferentes en cada una de las subclases, ¿En qué línea se aplica el polimorfismo?

En la última, porque invoca a las distintas definiciones del método intereses utilizando el mismo medio de acceso: una referencia a CCuenta que es el tipo del array cliente [i]. Así, un objeto tipo CCuenta se está comportando como una cuenta de ahorro, una cuenta corriente y una cuenta con intereses (polimorfismo).

Veremos todo esto con un clásico ejemplo sencillo usando de nuevo, Figura.

2. EJEMPLO 1



La palabra polimorfismo viene de “múltiples formas”, por lo tanto, las operaciones polimórficas son aquellas que hacen funciones similares con objetos diferentes.

```
public abstract class Figura {

	private String nombre, color;

/*Es interesante el hecho de que sea una clase abstracta puesto que una figura es un objeto general, no sabemos cómo calcular su área o su perímetro si no concretamos qué tipo de figura es de todas formas se podría hacer perfectamente sin necesidad de que la clase Figura sea abstracta*/

	public Figura ( ){

	}

	public Figura (String nombre, String color) {


		this.nombre = nombre;

		this.color = color;

	}

	@Override

	public String toString() {

		return "Figura [nombre=" + nombre + ", color=" + color + "]";

	}

	public abstract double calcularArea ();

	public abstract double  calcularPerimetro ();


	public void metodoSoloDeFigura ( ){

		System.out.println("Solo estoy en la clase Figura, sin sobrescribir en las hijas");

	}

}
```

Clase Cuadrado que extiende a Figura

```
public class Cuadrado extends Figura{

	private double lado;

	public Cuadrado ( ){

	}

	public Cuadrado(String nombre, String color, double lado) {

		super(nombre, color);

		this.lado=lado;

	}

	public double getLado() {

		return lado;

	}

	public void setLado(double lado) {

		this.lado = lado;

	}


	@Override
```

```
	public String toString() {
		return "Cuadrado [lado=" + lado + ", toString()=" + super.toString() + "]";
	}
	@Override
	public double calcularArea() {
		return lado*lado;
	}
	@Override
	public double calcularPerimetro() {
		return lado*4;
	}
	public void mostrarLados (){
		 System.out.println("Solo estoy en la clase Cuadrado porque los demás no tienen lados, en concreto tengo 4 lados ");
	}
}
```

Clase Círculo, que también es hija de Figura

```
public class Circulo extends Figura{
	private double radio;
	public Circulo (){
	}
	public Circulo(String nombre, String color, double radio) {
		super(nombre, color);
		this.radio = radio;
	}
	public double getRadio() {
		return radio;
	}
	public void setRadio(double radio) {
		this.radio = radio;
	}

	@Override
	public String toString() {
		return "Circulo [radio=" + radio + ", toString()=" + super.toString()+"]";
	}
	@Override
	public double calcularArea() {
		// TODO Auto-generated method stub
```

```
                return Math.PI* Math.pow(radio*2);
        }
        @Override
        public double calcularPerimetro() {
                // TODO Auto-generated method stub
                return 2*Math.PI*radio;
        }
        public void mostrarRadianes(){
                System.out.println("Solo estoy en la clase Círculo, porque los demás no pueden tener radianes en concreto tengo 2 pi radianes");
        }
}
```

Una clase normal, sin estar en la jerarquía de herencia para ver el uso de métodos

```
public class OperacionesFiguras {
        public double calcularElAreaDeUnaFigura (Figura f){
                return f.calcularArea();
        }

/*1) Dentro del método sumarAreas, se está llamando al método calcularElAreaDeUnaFigura, al que se le pasa como parámetro una figura del array
/*2) El método calcularElAreaDeUnaFigura es el que distingue a qué versión de calcularArea hay que llamar según lo que haya en el listado, un cuadrado o un círculo*/
        public double sumarAreas (Figura [] listado){
                double resultado=0;
                for (int i=0; i<listado.length;i++){
                        resultado=resultado+calcularElAreaDeUnaFigura(listado[i]);
                }
                return resultado;
        }
}
```

Clase Test para probar todo

```
public class PruebaPolimorfismo {
        public static void main(String[] args) {
                //Prueba con objetos "sueltos"
                //Figura f1=new Figura (); No se puede crear porque Figura es abstracta
                Cuadrado cu1= new Cuadrado("Primer cuadrado", "Rojo", 1.0);
                Circulo ci1= new Circulo ("Primer Círculo", "Azul", 1.0);
                System.out.println(cu1);
                System.out.println(ci1);
                System.out.println("**********Área y perímetro**********");
                System.out.println("Área del primer cuadrado: "+cu1.calcularArea());
```

```
System.out.printf("Perímetro del primer círculo: %.2f",ci1.calcularArea());


//Prueba con polimorfismo

System.out.println("\n\n\n**************************************************");

Figura f1= new Cuadrado ("Segundo cuadrado", "verde", 2.0);

Figura f2= new Circulo ("Segundo círculo", "Amarillo", 2.0);


System.out.println(f1);

System.out.println(f2);

System.out.println("**********Áreas**********");

System.out.println("Usan el método de cada clase concreta porque está sobrescrito");

System.out.println("Área del segundo cuadrado: "+f1.calcularArea());

System.out.printf("Perímetro del segundo círculo: %.2f",f2.calcularArea());


System.out.println("\nPero ahora no pueden usar los métodos que solo están en cuadrado y círculo porque son figuras\n\n");

//System.out.println(f1.mostrarLados ()); Error de compilación

//System.out.println(f2.contarRadianes ());Error de compilación


System.out.println("**********************************************************");

System.out.println("*******Vamos ahora con el array de objetos**********\n\n");


//Prueba con array de objetos

Figura lista []= new Figura [4];

//Cargamos el array con objetos cuadrado y círculo indistintamente

//No podemos hacerlo con figuras porque la clase Figura es abstracta, si no lo fuera sí se podría hacer


lista[0]= new Cuadrado ("Un mísero cuadrado", "negro", 2.0);

lista[1]= new Circulo ("Un mísero círculo", "blanco", 2.0);

lista[2]= new Circulo ("Un círculo grisáceo", "gris", 2.0);

lista[3]= new Cuadrado ("Un cuadrado desnudo", "transparente", 2.0);


//Una prueba simple de lo que hay en el array es llamar al toString. Dependiendo de //lo que haya guardado en el

//array se llamará a uno u otro toString



for (int i=0; i<lista.length;i++){

        System.out.println(lista[i]);

}
```

```
//Llamamos a los métodos de la clase OperacionesFiguras para calcular las áreas individualmente y la suma de todas las áreas

//Creamos un objeto de la clase, aunque debería estar creado arriba, se pone aquí para no liar


                    OperacionesFiguras of= new OperacionesFiguras ();

                    for (int i=0; i< lista.length;i++){

                                System.out.printf("El área del "+ (i+1)+ " es: %.2f \n",of.calcularElAreaDeUnaFigura (lista[i]));

                    }

                    System.out.printf("La suma de todas las áreas es: %.2f", of.sumarAreas(lista));

//Dejo para vosotros el probar con el método calcularPerímetro

          }

}
```