Métodos de Programación Orientada a Objetos. Polimorfismo

Practica Cinco

Ortega Ezquerra, Jesús Rodigo
Ingeniería Eléctrica y Electrónica.
Univesidad Nacional Autónoma de México
CDMX, México
jeroorez@gmail.com
https://orcid.org/0000-0002-8965-1189

Morales Medina, Antonio
Ingeniería Eléctrica y electrónica.
Univesidad Nacional Autónoma de México
CDMX, México
moralesmedinaa201@gmail.com

***Resumen*—En esta práctica se busca entender el uso del polimorfísmo en el lenguaje de java para la clase de Modelos de Programación Orientada a Objetos.**

***Index Terms*—POO, java 8, polimorfismo, instenceof, abstract, extends, super**

I. Introducción.

Para el previo de esta práctica se debían preparar tres clases:

1. Poligono(superclase)
2. Triangulo
3. Cuadrilatero

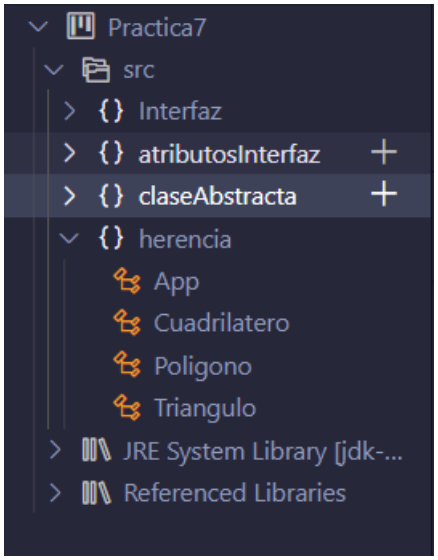

Figura 1. Estructura del paquete herencia.

La clase `App` se escribió en clase cómo `MPOOP7` para efectuar las actividades uno y dos

I-A. Objetivos.

1. Analizar la herencia y privacidad de clases base así como sus derivadas para relacionar sus instancias que la llevan a ser polomórficas.
2. Definir el funcionamiento de el método `instanceof` y conocer la instancia de los objetos creados.
3. Reconocer la diferencia de una clase abstracta y una privada
4. Conocer las interfaces en Java para su implementación.

II. Clases base.

> ”El concepto de herencia conduce a una estructura jerárquica de clases o estructura de árbol, lo cual significa que en la OOP todas las relaciones entre clases deben ajustarse a dicha estructura.” [1]

II-A. Tipos de clase.

1. *Clase Base*
   También llamada clase padre o superclase, es aquella que va a heredar a las clases hijas que derivan a partir de la clase padre. (Figura 2)
2. *Clase Derivada*
   Clase que reinstancía a la clase padre de la que deriva para ser una de sus subclases, cabe mencionar que hereda los métodos y variables de la clase padre. Se le conoce también como clase hija o subclase.

II-B. Actividad uno.

Se crea un objeto llamado `poligono` y se imprime la cadena de atributos. se crea otro objeto llamado `objeto` y se asocia a la clase arbol de java `Object.lang` y cuando se intenta imprimir en pantalla nos regresa la dirección de memoria dónde se encuentra almacenado `objeto` En cambio, cuando se asocia con la clase padre `poligono` que contiene el método `toString();` es cuando envía correctamente los valores de su clase en forma de cadena.Figura 3

II-C. Actividad dos.

Para la actividad dos se asocia a cada una de las clases para darle la característica de polimorfismo.

```java
package herencia;

public class Poligono {

    /**
     * Clase padre de las clases Trangulo y Cuadrilatero
     */

    public Poligono(){

    }

    public void area(){

    }

    public void perimetro(){

    }


    /**
     * @return String Cadena de Poligono
     */
    @Override
    public String toString(){
        return "Poligono {" + '}';
    }

}

```

Figura 2. Superclase Poligono.

```
1**************************
Poligono {}
Objetojava.lang.Object@68f7aae2
Objeto como Poligono Poligono {}
2**************************
Triangulo {alpha = 0beta = 0gamma = 0a = 0.0b = 0.0c = 0.0base = 0.0altura = 0.0
El objeto es un triangulo.
Cuadrilatero [a=0.0, alpha=0, altura=0.0, b=0.0, base=0.0, beta=0]
El objeto es un cuadrilatero.
Poligono {}
El objeto es un poligono.
```

Figura 3. Actividad uno y dos en consola.

III. Método instanceof.

III-A. `Instanceof`

”The instanceof keyword compares an object to a class or interface type.It also looks at subclasses and subinterfaces. x instanceof Object returns true unless x is null” [2]

Lo que hace el método `instanceof` es comparar el objeto de una clase o interfaz. También se utiliza para subclases y subinterfaces. Para la actividad uno, usaremos la clase poligono cómo clase padre para extender otras figuras geométricas. (Figura 4)

```java
    /**
     * @param poligono herencia.Poligono
     */
    public static void selectorPoligonos(Poligono poligono) {
        if (poligono instanceof Triangulo) {
            System.out.println("El objeto es un trangulo.");
        } else if (poligono instanceof Cuadrilatero) {
            System.out.println("El objeto es un cuadrilatero.");
        } else if (poligono instanceof Poligono) {
            System.out.println("El objeto es un poligono.");
        } else {
            System.out.println("El objeto es una figura.");
        }

    }
}

```

Figura 4. Condicional anidada con el el método instanceof.

IV. Clase abstracta.

IV-A. `abstract`

”In software engineering and programming language theory, the abstraction principle (or the principle of abstraction) is a basic dictum that aims to reduce duplication of information in a program (usually with emphasis on code duplication) whenever practical by making use of abstractions provided by the programming language or software libraries. The principle is sometimes stated as a recommendation to the programmer, but sometimes stated as requirement of the programming language, assuming it is self-understood why abstractions are desirable to use.” [3]

En pocas palabras, la abstracción es una forma de ahorrar espacio en memoria así como líneas de código.

IV-B. *Actividad tres.*

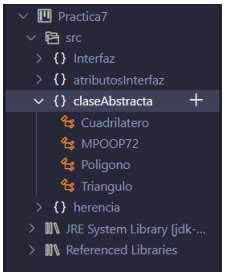

Figura 5. Arbol de jararquía de claseAbstracta.

Esta vez se hace exactamente lo mismo que el paquete de `herencia` (Figura 5), sólo que a diferencia de la actividad uno y dos, en la actividad tres su modificador de acceso se declara cómo `abstract` para analizar su comportamiento. (Figura 6)

```java
package claseAbstracta;

public abstract class Poligono {


    /**
     * @param toString(
     * @return float
     */
    public abstract float area();

    /**
     * @param toString(
     * @return float
     */
    public abstract float perimetro();


    /**
     * @return String
     */
    @Override
    public String toString() {
        return "Poligono {" + '}';
    }


}

```

Figura 6. Poligono para claseAbstracta.

Observamos que las clases abstractas no se pueden instanciar pero sus sublases si lo hacen.(Figura 7)

Algunos autores se reducen a definir una clase abstracta cómo aquella que no sabremos como emplear, pero obligatoriamente debe estar ahí. Una clase abstracta podría ser

```java
package claseAbstracta;

public class MPOOP72 {

    /**
     * @param args
     */
    Run | Debug
    public static void main(String[] args) {
        System.out.println("3**********************");
        //Poligono poligono = new Poligono(); X
        //Syntaxis incorrecta -> Poligono es abtrracta, no se puede instanciar.
        Poligono poligono = new Triangulo();
        System.out.println(poligono);

       poligono = new Cuadrilatero();
       System.out.println(poligono);
    }

}
```

Figura 7. Main para claseAbstracta.

aquella que permite generar sublases que en cierto modo van a compartir métodos o parámetros.(Figura 8) Al final la abstracción de una clase es encontrar una clase arbol que sabremos que derivará en subclases con distintos parámetros que podrían o no comprtir sus parámetros y clases, sean estas abractas o privadas.

```
3**********************
Triangulo {alpha = 0beta = 0gamma = 0a = 0.0b = 0.0c = 0.0base = 0.0altura = 0.0
Cuadrilatero [a=0.0, alpha=0, altura=0.0, b=0.0, base=0.0, beta=0]
```

Figura 8. Actividad tres en consola.

V. Interfaz.

”Una interfaz es una especie de plantilla para la construcción de clases. Normalmente una interfaz se compone de un conjunto de declaraciones de cabeceras de métodos (sin implementar, de forma similar a un método abstracto) que especifican un protocolo de comportamiento para una o varias clases. Además, una clase puede implementar una o varias interfaces: en ese caso, la clase debe proporcionar la declaración y definición de todos los métodos de cada una de las interfaces obien declararse como clase abstract. Por otro lado, una interfaz puede emplearse también para declarar constantes que luego puedan ser utilizadas por otras clases.” [4]

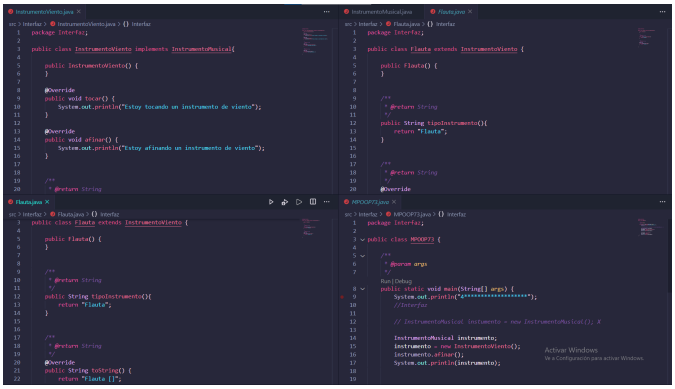

Figura 9. Clases que componen la paquetería interfaz.

V-A. *Actividad cuatro*

Para el ejemplo de las interfaces así cómo una clase, no es posible instanciar, por lo tanto hay que declararlo de la siguiente forma(Figura 11)

Es una especie de contenedor de variables y métodos que permiten mejor encapsulamiento, por lo tanto nivel de seguridad.

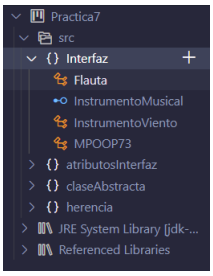

Figura 10. Jerarquía del paquete interfaz.

Una interface puede ser el medio para operar datos y otras clases.

```
4********************
Estoy afinando un instrumento de viento
InstrumentoViento []
Estoy afinando un instrumento de viento
Flauta []
```

Figura 11. Actividad cuatro en consola

Una interface también puede tener atributos.

VI. Atributos de interfaz.

VI-A. *Actividad cinco*

Es sólo un ejemplo corto pero significativo:

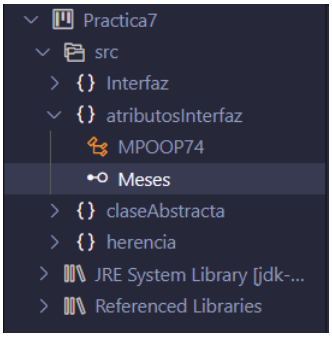

Figura 12. Jerarquía del paquete atributosInterfaz.

Se usan por ejemplo nombres de los días del mes que se podría incluso para hacerlas en distintos idiomas.(Figura 13) También podrían ser atributos para una interfaz gráfica, una skin para un personaje o incluso algoritmos de patrones en distintos tipos de poblaciones.

```java
package atributosInterfaz;

public class MPOOP74 {

    /**
     * @param args
     */
    Run | Debug
    public static void main(String[] args) {
        System.out.println("5************************");
        System.out.println("El mes de cumpleaños de Miguel es el número " + Meses.CINCO);
        System.out.println("La que corresponde al mes de " + Meses.NOMBRES_MESES[Meses.CINCO]);
    }
}
```

Meses.java

```java
package atributosInterfaz;

public interface Meses {
    //Atributos de una interfaz
    //public static final
    int UNO=1, DOS= 2, TRES=3, CUATRO=4, CINCO=5, SEIS=6, SIETE=7, OCHO=8, NUEVE=9, DIEZ=10;
    int ONCE=11, DOCE=12;
    String[] NOMBRES_MESES = {"", "enero", "febrero", "marzo", "abril", "mayo", "junio",
                              "julio", "agosto", "septiembre", "octubre","noviembre", "diciembre"};
}
```

Figura 13. Main para actividad cinco e interfaz meses.

```
5************************
El mes de cumpleaños de Miguel es el número 5
La que corresponde al mes de mayo
```

Figura 14. Actividad cinco en consola.

VII. Conclusiones.

VII-A. Polimorfismo.

Entendemos que cada superclase puede crear instancias pero no viceversa. Por lo tanto de infiere que cada superclase se puede comportar cómo una subclase.

VII-B. `instanceof`

El método `instanceof` devuelve las intancias de clases

VII-C. `abstract` *y* `private`

Las diferencias entre una clase `abstract` y una `private` es que una se puede instanciar, mientras las otras solo permiten atributos dentro de un concepto abstracto.

VII-D. `interface`

Una interface es un pequeño modelo que permite el almacenamiento de variables y métodos para usar en distintsas clases.

VIII. Repositorios.

- Github https://github.com/RhoKratos/MPOOP7
- Gitlab https://gitlab.com/jeroorez/practica7

Referencias

[1] I. C. A. R. Zamitiz. Programación orientada a objetos con java. Facultad de ingeniería, UNAM. CDMX, México, Ciudad universitaria. [Online]. Available: http://profesores.fi-b.unam.mx/carlos/java/indice.html

[2] Jeanne Boyarsky, Scott Selikoff, *Oracle Certified Professional Java SE 8 Programmer II Study Guide*, ser. OCA. United States, CA: Sybex, 2018.

[3] H. Hamill, *Java Abstraction*, ser. The Ultimate Guide. United States, CA: Java Code Geeks, 2006.

[4] J. A. A. García-Beltrán, *Programación Orientada a Objetos con Java*. Madrid, España: Universidad Técnica de Madrid, 2009, ch. 18.