Polimorfismo

Para definir esta palabra, me remontaré a sus raíces, que son griegas. “Polys” significa muchos, y “morfo” forma. De ahí que polimorfismo significa “cualidad de tener muchas formas”.

En el paradigma OO, es la habilidad de que un objeto pueda tomar muchas formas.

Dicho de otra manera, es la capacidad que tienen los objetos de comportarse de forma diferente, aunque conserven su mismo nombre.

Debido a que el comportamiento de un objeto está relacionado con los métodos que se le definen, generalmente la capacidad polimórfica de una clase se establece al crear métodos con el mismo nombre, pero con diferentes comportamientos.

¿Y para qué se necesita eso?, te preguntarás. Recuerda que el paradigma POO busca modelar un problema lo más cercano a como se observa en la vida real.

Observa la figura 1, en la que te muestro a dos personas. Puedes pensar entonces en la clase Persona. Obviamente cada persona tiene la habilidad de hablar. Pero como sucede en la vida real, las personas pueden hablar diferentes idiomas. Entonces piensa por ejemplo en un método llamado saludo. Una persona que habla inglés saludará como el joven de la derecha, y una que habla español como la de la izquierda. Es exactamente el mismo método, pero el resultado es diferente. Por eso necesitamos métodos que tengan nombres iguales, pero comportamientos diferentes.

Tipos de polimorfismo.

Existen diversas formas de clasificar los tipos de polimorfismo. Para algunos autores, los tipos de polimorfismo son dos: paramétrico o dinámico, y estático o ad hoc.

Otros autores definen los tipos de polimorfismo como: Sobrecarga, *ad hoc* y dinámico. Y otros más los dividen en dos categorías como ***Polimorfismo ad hoc***, que incluye a la sobrecarga de métodos y de operadores, y el ***polimorfismo universal,*** el cuál es controlado por la herencia y/o parámetros.

Los lenguajes de POO pueden implementarlos de manera diferente, pues algunos lenguajes dependen del grado de definición de su paradigma OO en su estructura.

Para escribir este artículo investigué los nombres “oficiales” –por así decirlo- de los tipos de polimorfismo en Java. No porque no los conociera, como programadora sé hacer uso de éstos, pero a veces los vicios del idioma parece que no les hacen justicia a los conceptos, como el hecho de que en muchas ocasiones sólo se piensa que la sobrecarga es la única forma de polimorfismo que se puede implementar.

Por ello, la clasificación más coherente o más aproximada a la esencia del paradigma OO en Java, me parece que es la siguiente:

1. Polimorfismo en tiempo de compilación.
2. Polimorfismo en tiempo de ejecución.

Vamos a analizar cada tipo a continuación.

Polimorfismo en tiempo de compilación.

A este polimorfismo se le conoce también como polimorfismo estático o paramétrico. Este polimorfismo se resuelve durante el tiempo de compilación, de ahí su nombre de polimorfismo estático. En Java se logra a través de la sobrecarga de métodos

La sobrecarga de métodos (method overloading) es una estrategia de Java que permite que existan varios métodos en una clase con el mismo nombre, pero con diferentes tipos y/o número de parámetros. Observa el siguiente ejemplo:

```
public class Clase {

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

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

En este polimorfismo, existen métodos con el mismo nombre (sumar), pero se usan diferentes tipos de datos en los parámetros, o diferente cantidad de parámetros. Cuando se ejecuten estos métodos, se seleccionará el que concuerde con el tipo y la cantidad de datos de los parámetros que se envíen, por lo que en el llamado de los métodos Java sabrá cuál de los tres debe ser ejecutado revisando la firma de éstos, como se observa a continuación:

```
public static void main(String[] args) {

     Clase objeto = new Clase();
     System.out.println("Resultado del llamado a sumar: " + objeto.sumar(1,2));
     System.out.println("Resultado del llamado a sumar: " + objeto.sumar(1,2,3));

}
```

Como puedes observar, en tiempo de compilación Java puede saber cuál de los métodos invocados se debe ejecutar sólo revisando la firma de los métodos. De hecho, si lo piensas, la posibilidad de crear diversos métodos constructores en una clase, es un ejemplo de sobrecarga de métodos.

Polimorfismo en tiempo de ejecución.

El polimorfismo en tiempo de ejecución, también llamado polimorfismo dinámico, sucede en Java a través de la sobre-escritura de métodos.

A la sobre-escritura de métodos también se le conoce como anulación de métodos (method overriding), y es una característica del lenguaje que permite que una clase hija o derivada proporcione una implementación especializada a un método que ya existe en la clase padre. De esta forma, la subclase anula la implementación de la súper clase, proporcionando un método con la misma firma (esto incluye el mismo nombre del método, cantidad y tipo de parámetros iguales, mismo tipo de dato de retorno), que reemplaza el método de la clase padre.

Esta característica del lenguaje Java es un polimorfismo en tiempo de ejecución porque el método que se ejecutará depende del objeto que es usado para invocarlo.

Para ejemplificar este tipo de polimorfismo, he creado la súper clase siguiente:

```
public class FiguraGeometrica {

     private String nombreFigura;
     private double area;
     private double perimetro;

     public FiguraGeometrica(String nombre){
          this.nombreFigura = nombre;
     }

     public String getNombreFigura() {
          return nombreFigura;
     }

     public void setNombreFigura(String nombreFigura) {
          this.nombreFigura = nombreFigura;
     }

     public double getArea() {
          return area;
     }

     public void setArea(double area) {
          this.area = area;
     }

     public double getPerimetro() {
          return perimetro;
     }

     public void setPerimetro(double perimetro) {
          this.perimetro = perimetro;
     }

     @Override
     public String toString() {
          return "FiguraGeometrica{" + "nombreFigura=" + nombreFigura + ", area=" + area + ", perimetro=" + perimetro + '}';
     }

}
```

Como puedes observar, se han creado los métodos *getters* y *setters* correspondientes a los miembros área y perímetro. Pero en realidad en este nivel de abstracción no podemos calcular el área de la figura geométrica, puesto que la fórmula para realizar ese cálculo depende del tipo de figura geométrica del que se trate (la fórmula para obtener el área de un cuadrado no es igual a la necesaria para obtener el área de un triángulo). Por lo tanto, claramente necesitamos que la clase hija anule el método de la clase padre y proporcione una implementación adecuada según sea necesario.

Para ilustrar esto, te presento a continuación una clase hija de FiguraGeometrica:

```
public class Cuadrado extends FiguraGeometrica{

     private double lado;

     public Cuadrado(double l){
          super("cuadrado");
```

```
          this.lado = l;
     }

     @Override
     public double getArea() {
          return this.lado * this.lado;
     }

     @Override
     public double getPerimetro() {
          return 4 * this.lado;
     }

     @Override
     public String toString() {
          return super.toString() + "Cuadrado{" + "lado=" + lado + '}';
     }
}
```

Como puedes ver, he mejorado la funcionalidad del método *getArea* de la súper clase. Ahora no sólo regresa el valor del área, sino que también la calcula, y esto debido a que específicamente ha sido implementado para la fórmula del área de un cuadrado. Entonces si después se requiere el área de un triángulo, se creará una nueva clase, que heredará de *FiguraGeometrica*, y nuevamente modificaré el método *getArea*, ahora ajustándolo a la fórmula para obtener el área de un triángulo.

En este tipo de polimorfismo, existen métodos con el mismo nombre, pero la clase a la que pertenece la instancia dictaminará cuál de ellos se ejecutará:

```
public static void main(String[] args) {
      FiguraGeometrica unaFigura = new FiguraGeometrica("una figura");
      Cuadrado unCuadrado = new Cuadrado(20);
      System.out.println("Area de la figura: " + unaFigura.getArea());
      System.out.println("Area del cuadrado: " + unCuadrado.getArea());
      unCuadrado.setArea(unCuadrado.getArea());
      System.out.println(unaFigura.toString());
      System.out.println(unCuadrado.toString());
}
```

Como puedes observar, en la primera llamada a *getArea()*, el tipo de referencia es *FiguraGeometrica*. Por lo que al momento en que la llamada a *getArea()* se realiza, Java espera hasta el tiempo de ejecución para determinar cuál objeto está apuntando al método referenciado, en este caso, es *unaFigura*, una instancia de *FiguraGeometrica*.

Otro ejemplo común del polimorfismo dinámico es el método *toString*, el cuál puedes ver que estoy usando tanto en las clases padre e hija.

En estos ejemplos de polimorfismo parece muy obvio entender cuál método será llamado, porque se han creado objetos de cada una de las clases creadas. Sin embargo, la clase *FiguraGeometrica* no tiene en sí ninguna funcionalidad, más que servir de referencia a futuras clases de figuras que se vayan creando en el futuro. Es decir, se establece que toda figura geométrica tiene área y perímetro, pero nada más. Esto podría haberse establecido en una forma más elegante y adecuada usando una interfaz.

Por ello, ahora considera ahora el siguiente ejemplo, en donde no he creado un objeto *FiguraGeometrica*:

```
public static void main(String[] args) {
     FiguraGeometrica c = new Cuadrado(20);
     System.out.println("Área del cuadrado: " + c.getArea());
}
```

En este ejemplo, en la llamada a *getArea()* el tipo referenciado es *FiguraGeometrica*, pero el objeto que está siendo referenciado es *Cuadrado*. Ahora parece más obvio que cuando *getArea()* es llamado, Java necesita esperar hasta el tiempo de ejecución para determinar a cuál método apunta realmente la referencia, en este caso, la clase *Cuadrado*. Por eso la ejecución muestra el resultado de *getArea* de la clase *Cuadrado*:

Como el método a llamar se determina en tiempo de ejecución, esto se denomina enlace dinámico o enlace tardío.

Algunas reglas a considerar.

1. Para el polimorfismo estático:
   - Obviamente, los métodos deben llamarse iguales para que realmente puedan cumplir con ser polimórficos.
   - El número de parámetros, y su tipo de datos, deben ser diferentes en cada método definido como polimórfico.
2. Para el polimorfismo dinámico:
   - Nuevamente, se debe cumplir el primer inciso del punto anterior.
   - Los parámetros del método que sobre-escribe deben ser iguales a los del método que está anulando, tanto en tipo como en número. Si esto no se cumple, entonces éste método no anula al de la clase padre, por lo tanto, se trata de polimorfismo estático y no dinámico.

Bibliografía:

- Extraído de <<¿Qué es polimorfismo en el paradigma orientado a objeto?>>. Consultado el 28 de noviembre del 2022