Polimorfismo en Java: Programación orientada a objetos

por: Equipo Geek, 3 Feb 2020

Cuando hablamos de polimorfismo no es que nos hemos vuelto locos. Todos saben que las personas que nos dedicamos al desarrollo de software, tenemos un lenguaje propio y no es que un ser procedente del espacio exterior y congelado por miles de años en la Antártida hubiera vuelto a la vida. No, no hablamos de ninguna película de miedo de los 80 sino de un concepto que por su utilidad debería estar en el ABC de cualquier libro de programación.

Método de Polimorfismo Java

En **programación orientada a objetos**, polimorfismo es la capacidad que tienen los objetos de una clase en ofrecer respuesta distinta e independiente en función de los parámetros (diferentes implementaciones) utilizados durante su invocación. Dicho de otro modo el objeto como entidad puede contener valores de diferentes tipos durante la ejecución del programa.

En JAVA el término polimorfismo también suele definirse como 'Sobrecarga de parámetros', que así de pronto no suena tan divertido pero como veremos más adelante induce a cierta confusión. En realidad suele confundirse con el tipo de poliformismo más común, pero no es del todo exacto usar esta denominación.

Ejemplo de Polimorfismo

Un ejemplo clásico de **poliformismo** es el siguiente. Podemos crear dos clases distintas: Gato y Perro, que heredan de la superclase Animal. La clase Animal tiene el método abstracto makesound() que se implementa de forma distinta en cada una de las subclases (gatos y perros suenan de forma distinta). Entonces, un tercer objeto puede enviar el mensaje de hacer sonido a un grupo de objetos Gato y Perro por medio de una variable de referencia de clase Animal, haciendo así un uso polimórfico de dichos objetos respecto del mensaje mover.

Java

Certificaciones JAVA, la guía definitiva [ACTUALIZADO]

Java

Cómo descargar e instalar Java en Linux, Windows 10, o Mac OS

Java

Qué es y cómo utilizar instanceof en Java

Leer más

```
class Animal {
  public void makeSound() {
    System.out.println("Grr...");
  }
}
class Cat extends Animal {
  public void makeSound() {
    System.out.println("Meow");
  }
}
class Dog extends Animal {
  public void makeSound() {
    System.out.println("Woof");
  }
}
```

Como todos los objetos Gato y Perro son objetos Animales, podemos hacer lo siguiente:

```
public static void main(String[ ] args) {
  Animal a = new Dog();
  Animal b = new Cat();
}
```

Creamos dos variables de referencia de tipo Animal y las apuntamos a los objetos Gato y Perro. Ahora, podemos llamar a los métodos makeSound().

```
a.makeSound();
//Outputs "Woof"

b.makeSound();
//Outputs "Meow"
```

Como decía el polimorfismo, que se refiere a la idea de "tener muchas formas", ocurre cuando hay una jerarquía de clases relacionadas entre sí a través de la herencia y este es un buen ejemplo.

Por lo general diremos que existen 3 tipos de polimorfismo:

- Sobrecarga: El más conocido y se aplica cuando existen funciones con el mismo nombre en clases que son completamente independientes una de la otra.
- Paramétrico: Existen funciones con el mismo nombre pero se usan diferentes parámetros (nombre o tipo). Se selecciona el método dependiendo del tipo de datos que se envíe.
- Inclusión: Es cuando se puede llamar a un método sin tener que conocer su tipo, así no se toma en cuenta los detalles de las clases especializadas, utilizando una interfaz común.

En líneas generales en lo que se refiere a la POO, la idea fundamental es proveer una funcionalidad predeterminada o común en la clase base y de las clases derivadas se espera que provean una funcionalidad más específica.

Polimorfismo paramétrico

Antes habíamos visto un ejemplo clásico de polimorfismo basado en sobrecarga. Pero veamos ahora un ejemplo Paramétrico. Es importante entender que la conversión automática sólo se aplican si no hay ninguna coincidencia directa entre un parámetro y argumento

```
class Overload
{
    void demo (int a)
    {
       System.out.println ("a: " + a);
    }
    void demo (int a, int b)
    {
       System.out.println ("a and b: " + a + "," + b);
    }
    double demo(double a) {
       System.out.println("double a: " + a);
       return a*a;
    }
}
class MethodOverloading
{
    public static void main (String args [])
    {
        Overload Obj = new Overload();
        double result;
        Obj .demo(10);
        Obj .demo(10, 20);
        result = Obj .demo(5.5);
        System.out.println("O/P : " + result);
    }
}
```

Salida de datos:

```
a: 10
a and b: 10,20
double a: 5.5
O/P : 30.25
```

Polimorfismo de inclusión

La habilidad para redefinir por completo el método de una superclase en una subclase es lo que se conoce como polimorfismo de inclusión (o redefinición).

En él, una subclase define un método que existe en una superclase con una lista de argumentos (si se define otra lista de argumentos, estaríamos haciendo sobrecarga y no redefinición).

Un ejemplo muy básico en donde la clase Bishop sobreescribe el método move. Esto es el polimorfismo de inclusión.

```
abstract class Piece{
    public abstract void move(byte X, byte Y);
}

class Bishop extends Piece{
 @Override
  public void move(byte X, byte Y){

  }
}
```

Diferencias entre polimorfismo y sobrecarga

El polimorfismo presenta unas claras ventajas aplicado desde las interfaces, ya que nos permite crear nuevos tipos sin necesidad de modificar las clases ya existentes. Esta con recompilar todo el código que incluye los nuevos tipos añadidos sin retocar la clase anteriormente creada para añadir una nueva implementación lo que podría suponer una revisión completa de todo el código donde se instancia la clase.

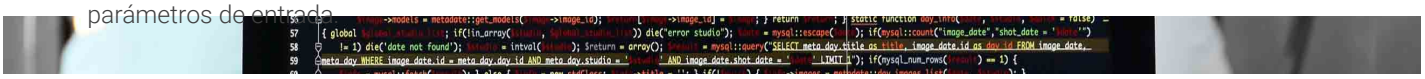
Por contra, un método está sobrecargado si dentro de una clase existen dos o más declaraciones de dicho método con el mismo nombre pero con parámetros distintos, por lo que no hay que confundirlo con polimorfismo.

Esto puede parecer un poco confuso pero en definitiva el Polimorfismo consiste en redefinir un método de una clase padre en una clase hija. Mientras que sobrecarga es definir un nuevo método igual que otro viejo, pero cambiando el tipo o la cantidad de parámetros.

El compilador, viendo los parámetros, sabe a qué método llamar en función del parámetro que estás pasando. La sobrecarga se resuelve en tiempo de compilación utilizando los nombres de los métodos y los tipos de sus parámetros; el polimorfismo se resuelve en tiempo de ejecución del programa, esto es, mientras se ejecuta, en función de la clase a la que pertenece el objeto.

Preguntas Frecuentes sobre polimorfismo en Java

- **¿Qué es el término firma?** Como ya se ha tratado en este artículo, en Java dos o más métodos dentro de la misma clase pueden compartir el mismo nombre, siempre que sus declaraciones de parámetros sean diferentes. Cuando esto sucede, se dice que estamos usando sobrecarga de métodos (method overloading). En general sobrecargar un método consiste en declarar versiones diferentes de él. Y aquí es donde el compilador se ocupa del resto y donde el término firma cobra importancia. Una firma es el nombre de un método más su lista de parámetros. Por lo tanto cada método en una misma clase, en términos de sobrecarga, obtiene una firma diferente.
- **¿Diferencias entre los términos Overloading y Overriding?** Overloading significa que un mismo método tiene diferentes firmas mientras que Overriding es el mismo método, por tanto misma firma, al que diferentes clases conectan a través de la herencia. En algunos textos encontramos otra explicación en donde se resume la sobrecarga como un ejemplo de polimorfismo en tiempo de compilación y la sobreescritura como un ejemplo de polimorfismo en tiempo de ejecución.
- **¿Se pueden sobrecargar métodos estáticos?** Sí, es posible tener dos más métodos estáticos con el mismo nombre siempre que se diferencien en los parámetros de entrada.



- ¿Es posible sobrecargar un método en Java? Si es así, definamos correctamente un método sobrecargado de ejemplo.

```
class Demo{

    public static void main(String[] args) {
        System.out.println("Hello Folks"); // Hello Folks
        Demo.main("Ducks");
    }

    // Sobrecargando
    public static void main(String arg1) {
        System.out.println("Hello, " + arg1); // Hello Ducks
        Demo.main("Dogs","Cats");
    }
    public static void main(String arg1, String arg2) {
        System.out.println("Hello, " + arg1 + " and "  + arg2); //
    }
}
```


Java Polimorfismo

Java

Web scraping con javascript y nodejs

Java

Machine Learning con Java. Fundamentos básicos

Java

Recursos y cursos para convertirte en un ninja de la programación con JavaScript

Leer más

La comunidad de

if [geek] then

ifgeekthen.nttdata.com es un blog del grupo NTT DATA. Un proyecto que busca compartir con el mundo de los desarrolladores artículos relacionados con la tecnología y el universo geek.

MuleSoft celebra un meetup híbrido y presenta: API Governance y Anypoint Flex Gateway

Primeros pasos hacia MLOps - Machine Learning Operations (1 de 3)

Qué es la Servitización de productos y por qué es importante

Java | Javascript | Azure | AWS | Salesforce | Formación | Inteligencia Artificial (AI) | API | Oracle | Lambda | AngularJS | OutSystems | CSS | React | serverless | IDE | Transformación Digital | Big Data | CRM | Dynamic 365

IR