POLIMORFISMO

AUTOR: WALTER MADRIGAL CHAVES

NOVIEMBRE: 2020

San Marcos

Contenido

INTRODUCCIÓN

La programación orientada a objetos tiene cuatro pilares fundamentales que le dan sustento a su lógica, estos son: la abstracción que es la que permite identificar las características de un objeto; el encapsulamiento que oculta la complejidad del código; la herencia cuya función es la reutilización del código y el polimorfismo.

En la siguiente lectura se estudiará al termino de polimorfismo orientado a la programación de software, el cual representa la capacidad que tienen los objetos dentro de una misma clase de cambiar o mutar según su entorno y necesidad, haciendo posible definir varios métodos o comportamientos de un objeto bajo un mismo nombre

Además, se ahondará en conceptos importantes como jerarquía, Class, interfaz, instanceof, java reflection, el fin es dar un mayor sustento y robustez a la utilización de este pilar fundamental llamado polimorfismo.

POLIMORFISMO

Según (Blasco, 2020), En una aplicación polimórfica nos encontramos con una "familia" de clases, todas ellas con un "ascendiente" común, que marca un patrón de comportamiento genérico, igual para todas las clases "descendientes", pero que, partiendo de ese patrón de comportamiento común, cada una de esas clases descendientes (Subclases) le puede dar "forma" diferente, es decir una implementación específica concreta.

Polimorfismo es la capacidad que tiene un objeto de adquirir varias formas, se puede aplicar sobre variables y métodos. Es un concepto que va de la mano con la herencia, debido a que es posible implementar polimorfismo en jerarquías de clasificación que se dan a través de la herencia.

Dentro de sus virtudes se denota la facilidad de simplificar el código y reducir el número de clases que los programadores deben conocer.

CLASIFICACIÓN

Se pueden clasificar en dos grupos: dinámicos y estáticos.

- **Polimorfismo dinámico:** también llamado polimorfismo paramétrico, su característica principal es que el código no indica el tipo de datos sobre el que se trabaja, dando la ventaja que puede ser utilizado por todo tipo de datos compatible.

- **Polimorfismo estático**: también conocido como polimorfismo ad hoc, al contrario del anterior en este si se requiere que los tipos involucrados en el polimorfismo sean declarados antes de poder ser utilizados.

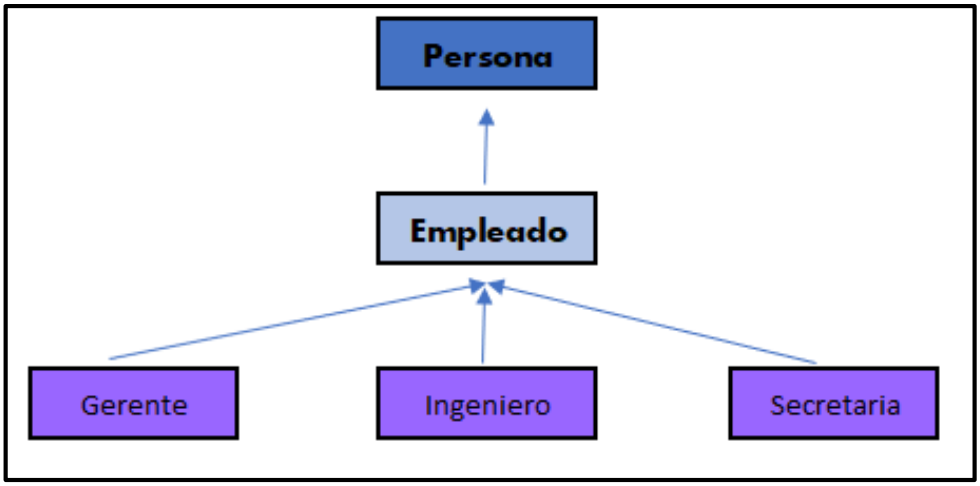
JERARQUÍA POLIMÓRFICA

Mantiene el mismo principio que la jerarquía de clases, solo que aplica el concepto de polimorfismo. Existe una clase padre que hereda los métodos y

atributos hacia debajo de la jerarquía en donde se encuentran las clases hijas. Las clases superiores describen un comportamiento más general. Las inferiores más específicas.

En la siguiente imagen se muestra la jerarquía de clases como hasta el momento se ha estudiado, en donde existe una clase Persona (padre) que hereda a una clase Empleado (hija) y esta a su vez hereda a tres clases más, Gerente, Ingeniero y Secretaria.

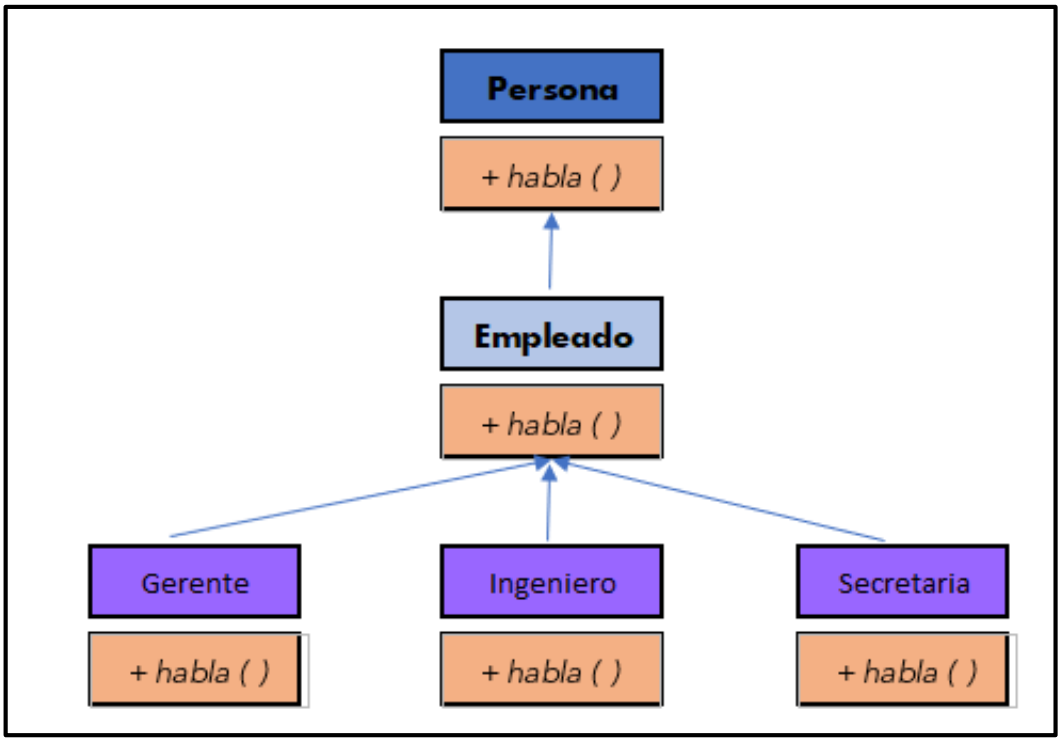
Imagen 1 Jerarquía de clases



Fuente: Elaboración propia

Se puede realizar un cambio en esta herencia para aplicar el polimorfismo, en la siguiente imagen se realiza una modificación a esta estructura.

Imagen 2 Jerarquía polimórfica



Fuente: Elaboración propia

Como se observa, se crea un nuevo método llamado "habla", es gracias a este nuevo miembro que se va a aplicar el concepto de polimorfismo. Todas las clases hijas heredaron dicho método, pero este se va a comportar diferente de acuerdo con la clase en que se encuentre, de ahí el concepto de que un objeto puede tomar muchas formas.

Lo que habla el gerente, es diferente a lo que puede hablar el ingeniero o la secretaria.

CLASS

Para (Blasco, 2020), la clase Class es el centro de las acciones del polimorfismo, es mediante sus métodos que se puede acceder a información relacionada con metadados de cualquier otra clase. Para ello, es necesario

disponer de un objeto que responda a la clase Class asociada a la clase de la que necesitamos obtener dichos metadatos.

Hay tres formas para obtener la información mediante Class:

- Invocando al método getClass () desde una referencia al objeto.
- Aplicando de forma estática Class al nombre de la clase
- Aplicando como argumento el método static forName (). un string que suponga el path y nombre de la clase. Este también es utilizado para ingresar a bases de datos relacionales

Este tipo de clase a diferencia de las demás no tiene constructores públicos y sus instancias son creadas por la Java Virtual Machine (JVM) cuando la clase es cargada. El objeto de clase tipo Class también es usado para representar clases, enumeraciones, interfaces y anotaciones corriendo en una aplicación Java. Los tipos primitivos como byte, short, char, int, float, double, boolean, así como la palabra reservada void también son representados como instancias de Class. Se puede obtener la instancia correspondiente a Class usando la palabra class, por ejemplo, int.class, float.class o boolean.class.

ABSTRACT CLASS VRS INTERFAZ

Hay algunas diferencias que se plantean al comparar a ambas estructuras, las cuales se plasman en la siguiente imagen.

Imagen 3 Diferencias entre clase Abstract y una Interfaz

Abstract Class	Interfaz
Tiene la posibilidad de heredar cualquier clase, independientemente de que esta sea abstracta o no lo sea.	Únicamente puede extender o implementar otras de su mismo tipo.
Hereda de una sola clase, sea abstracta o no.	Puede extender varias interfaces de una misma vez.
Puede definir métodos abstractos o que no lo sean.	Solo puede definir métodos abstractos.

En Java para definir un método dentro de una clase abstracta es indispensable la palabra “abstract”.	La palabra “Interfaz” no es necesaria ya que Java infiere en el concepto de interfaz
Los métodos abstractos pueden ser public o protected.	La interfaz solo puede tener métodos públicos.
Permite variables static, final o static final con cualquier modificador de acceso (public, private, protected o default).	Solo tener constantes public, static, final.

Fuente: Elaboración propia

OPERADOR INSTANCEOF

El operador instanceof permite comprobar si el objeto al que apunta una referencia es una instancia de una clase o interfaz concreta, es una herramienta útil cuando existe una colección de objetos y no hay seguridad de que tipo específico es. Se compone de tres partes:

- Una referencia.
- La palabra clave instanceof.
- El nombre de la clase o interfaz a comprobar.

Referencia instanceof clase = Dato1 instanceof persona

Para una mejor compresión se detalla el siguiente ejemplo:

Imagen 4 Implementación instanceof

```
1  public class Main {
2      public static void main (String[] args) {
3          Empleado empleado = new empleado();
4          System.out.print(empleado instanceof Persona);
5      }
6  }
7
8  class Persona {}
9  class Empleado extends Persona {}
10 class Profesion {}
11
```

Fuente: Elaboración propia

En el método principal se imprime el resultado de la validación realizada por el comando instanceof, el resultado puede ser "true" o "false", en términos simples

lo que hace el sistema es realizar la pregunta ¿Es el objeto empleado una instancia de la clase persona? Siguiendo el ejemplo la respuesta sería true, porque toda clase de la instancia Empleado es también una instancia de la clase Persona.

Como nota importante la instanceof solo devuelve un valor false cuando:

- La referencia no apunta a ningún objeto.
- Cuando la referencia es de una superclase.

JAVA REFLECTION

Java Reflection es un API muy versátil del lenguaje Java que tiene la capacidad de observar y modificar su estructura de manera dinámica. Brinda la posibilidad de inspeccionar clases, métodos, interfaces y campos en tiempo de ejecución. Además, da la posibilidad de crear instancias de nuevos objetos, invocar métodos y establecer valores de campo mediante la reflexión.

API - Application Programming Interfaces: Es conjunto de comandos, funciones y protocolos para crear programas para ciertos sistemas operativos

GENÉRICOS

Son tipos parametrizados, es gracias a ellos que se pueden crear clases, interfaces y métodos en los que el tipo de datos sobre los que operan se especifica como parámetro. Una clase, interfaz o método que utiliza un tipo de parámetro se denomina genérico:

- Clase genérica.
- Interfaz genérica.
- Método genérico.

La importancia de trabajar con código genérico es que se puede crear un

algoritmo, independientemente de cualquier tipo específico de datos, y luego aplicar ese algoritmo a una amplia variedad de tipos de datos sin ningún esfuerzo adicional.

Java ofrece la posibilidad de crear clases, interfaces y métodos generalizados operando a través de referencias del tipo Object. El tipo Object es la clase padre de todas las demás clases, una referencia de Object puede referirse a cualquier tipo de objeto. Así, en el código genérico, las clases, interfaces y métodos generalizados utilizaban referencias a objetos para operar en varios tipos de datos.

Para ahondar más en el tema de Polimorfismo debe realizar la lectura de las páginas 329 a la 408 del libro: Programación orientada a objetos con Java. (2020) de Francisco Blasco.

CONCLUSIONES Y RECOMENDACIONES

- El polimorfismo está en un escalón más elevado que la herencia, es muy flexible para jerarquizar y generar patrones de comportamiento común a una serie de objetos que heredan de la misma clase.

- Es gracias al polimorfismo que los programadores pueden apartar los objetos que cambian de los que no cambian, logrando con esto hacer más fácil la ampliación, el mantenimiento y la reutilización de los programas. Una implementación correcta del concepto genera código simplificado, fácil de entender y de probar.

- Debido a sus virtudes de flexibilidad y gran independencia de la jerarquía de clases estándar, las interfaces ayudan en mucho a aplicar las características propias del polimorfismo.

- El polimorfismo en la actualidad es poco usado, pero más por un asunto de desconocimientos que por utilidad. Si se hace un apego a las buenas prácticas de programación, sin duda su aplicación sería una regla.

- La desventaja más representativa del polimorfismo es que se necesitas crear un número mayor de clases que en una implementación normal.

REFERENCIAS BIBLIOGRÁFICAS

Barnes, D., & Kölling, M. (2017). *Programación oerientada a objetos con Java usando Bluej 6a. Ed.* Madrid: Pearson.

Blasco, F. (2020). *Programación orientada a objetos en Java.* Bogotá: Ediciones la U.

Invarato, R. (25 de 11 de 2020). *Jarroba*. Obtenido de https://jarroba.com/reflection-en-java/

San Marcos

www.usanmarcos.ac.cr

San José, Costa Rica