Colaboración y Herencia

¡Comencemos!

¿Qué aprenderás?

- Aplicar las técnicas de POO para crear clases que contengan otra clase conociendo técnicas de colaboración.
- Aplicar la sobrecarga de métodos con Herencia para entender la jerarquía de clases.

Introducción

A continuación conoceremos los conceptos de Herencia y Colaboración que son los pilares fundamentales de la POO (Programación Orientada a Objetos). Estos conceptos ya los hemos estudiado en capítulos anteriores, sin embargo, ahora conoceremos la codificación detrás, realizando clases robustas donde la colaboración será de forma directa e indirecta. Profundizaremos en la sobrecarga de método utilizando herencia y sus palabras reservadas en codificación Java.

¡Vamos con todo!

Colaboración - Declaración de objetos dentro de otro Objeto

Podemos declarar un atributo de tipo Objeto a nivel de clases y para esto necesitamos conocer el objeto y su relación con la clase, también podemos utilizar instancias de otros objetos solo en operaciones particulares de la clase, a esto se le llama atributos de tipo Objetos locales.

A continuación, se mostrará estos dos tipos de declaración:

```
public class Persona {

    private String nombre;
    private String rut;
    private double altura;
    // Asociación de dependencia - colaboración de objetos
    private Lapiz lapiz;
```

Se crea un atributo de tipo Objeto llamado `Lapiz` y se agrega al constructor con parámetros de la clase Persona.

```
public Persona(String nuevoNombre,  String nuevoRut,  double
nuevaAltura,Lapiz nuevoLapiz) {
       // A esto se le llama sobrecarga de métodos
       this(nuevoNombre,nuevaAltura);
       this.rut = nuevoRut;
       this.lapiz  = nuevoLapiz;

    }
```

Ahora tenemos colaboración directa entre la clase `Persona` y `Lapiz`, cabe destacar que no utilizamos la palabra `new` para crear un `Lapiz`, Java interpreta que el objeto Lapiz, que viene por parámetro, ya se instanció en algún lugar de la **Java Virtual Machine** por ende ese `new` viene en el parámetro de entrada.

Dentro de nuestra clase `Persona` podemos tener métodos u operaciones que necesiten un objeto solo para un evento en especial, aquí también tenemos colaboración, pero no a nivel de clases sino a nivel de método, donde el objeto nace y muere dentro del método.

```
   public Cuaderno reciboCuaderno(Cuaderno cuadernito){

        return cuadernito;
    }
```

- Aquí tenemos un método el cual recibe y retorna un Objeto `Cuaderno`.
- El Objeto `Cuaderno` nace y muere en ese método.
- El parámetro `cuadernito` no es atributo de la clase y forma parte de atributos locales o específicos.

Podemos ver que el objeto `Cuaderno` tiene sus propios atributos, pero a su vez se utiliza en clase `Persona`, no siendo parte de esta como tal, sino solo en su método.

Herencia - super

No es nada más que crear clases que heredan atributos y métodos desde otra. A las clases que heredan elementos desde otras se les llama subclase, y a las que heredan sus elementos a otras se les llama superclase. Para realizar la herencia en Java necesitamos utilizar la palabra reservada `extends` esta palabra le indica a Java que se está realizando herencia de dos objetos.

Esta palabra va al lado del nombre de clase y antes de los corchetes de apertura de clase:

```
public class Programador extends Persona {

    private  String lenguaje;
}
```

El uso de la palabra reservada `super` se utiliza para la sobrecarga de métodos con herencia, ya que Java interpreta este `super` como la llamada al constructor padre. Siguiendo el ejemplo de la Clase `Persona` llamaremos siempre a la firma del constructor padre y después se agregan los atributos propios de la clase hija.

```
    public Programador(String nombre, String rut, double altura,Lapiz
lapiz, String lenguaje) {

        super(nombre,rut,altura,lapiz);
        this.lenguaje= lenguaje;

    }
```

Los atributos `nombre`, `rut`, `altura` y `lapiz` son de la clase padre `Persona` y solo el atributo `lenguaje` es de su hija. Sin embargo, Java lo interpreta como una sola clase `Programador`. De esta forma la clase hija puede utilizar métodos u operaciones de la clase padre.

Ejercicio guiado: Botillería

Paso 1: Creamos las siguientes clases para el proyecto: Botella, Cerveza y Botillería.

Paso 2: En la clase Botella crearemos los siguientes atributos:

- tipo Botella: String
- marca: String

Para luego generar su constructor y “getters and setters” correspondientes.

```
public class Botella {

      private String tipoBotella;
      private String marca;

      public Botella(String tipoBotella, String marca) {
            super();
            this.tipoBotella = tipoBotella;
            this.marca = marca;
      }

      public String getTipoBotella() {
```

```
            return tipoBotella;
      }

      public void setTipoBotella(String tipoBotella) {
            this.tipoBotella = tipoBotella;
      }

      public String getMarca() {
            return marca;
      }

      public void setMarca(String marca) {
            this.marca = marca;
      }

}
```

Paso 3: En la clase Cerveza crearemos el siguiente atributo:

- precio: int

Para luego generar su constructor y "getter and setter" correspondiente. Sin embargo, `Cerveza` es una subclase de Botella. Por lo tanto, usaremos la palabra reservada `extends` para aplicar herencia desde la subclase `Cerveza` hacia la superclase `Botella`.

```
public class Cerveza extends Botella {
      private int precio;

      public Cerveza(String tipoBotella, String marca, int precio) {
            super(tipoBotella, marca);
            this.precio = precio;
      }

      public int getPrecio() {
            return precio;
      }

      public void setPrecio(int precio) {
            this.precio = precio;
      }
}
```

Paso 4: En la clase `Botilleria` crearemos los siguientes atributos:

- cerveza: Cerveza
- nombre: String

Para luego generar el constructor y "getters and setters" correspondientes, pero esta vez usaremos una clase `Cerveza` que viene extendida desde `Botella`.

```
public class Botilleria {

	private Cerveza cerveza;
	private String nombre;

	public Botilleria(Cerveza cerveza, String nombre) {
		super();
		this.cerveza = cerveza;
		this.nombre = nombre;
	}

	public Cerveza getCerveza() {
		return cerveza;
	}

	public void setCerveza(Cerveza cerveza) {
		this.cerveza = cerveza;
	}

	public String getNombre() {
		return nombre;
	}

	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
}
```

Resumen

Como hemos visto en esta unidad, la orientación a Objetos tiene énfasis en la construcción y desarrollo de ciertas variables para manejar distintas partes del programa. Esto con la idea de que el código sea más estable y se ejecute sin mayores problemas.