Ejercicio Código 2

Autor: José Guillermo Sandoval Huerta

Fecha: 27 octubre 2023

Implementación Polimorfismo, clase Abstracta e interfaz.

El polimorfismo nos dice que emplea el **"mismo mensaje, pero comportamiento diferente"** esto nos indica que las clases hijas van a trabajar con los mismos atributos de la clase padre pero la forma en cómo van a trabajar con esos atributos va a ser diferente.

La clase Abstracta nos dice que, para ser abstracta, por lo menos uno de sus métodos debe de ser de tipo abstracto.

Y una interfaz nos dice que, al ser implementado, los métodos que estén declarados deben de ser obligatoriamente utilizados en las clases donde se implemente la interfaz.

Dado la explicación anterior un buen ejemplo de polimorfismo se ve reflejado en la película "Spiderman Into the Spiderverse" donde todos los personajes son spiderman, lo que los diferencia es su tipo de habilidad y su origen. Por lo que se implementó lo siguiente:

Se tiene una clase abstracta **SpiderPeopleVerso** la cual funciona como clase padre, en esta, se declaran sus atributos y tiene un método abstracto **tipoHabilidad()** el cual debe ser implementado por cada uno de sus clases hijas. Cabe mencionar que esta clase implementa la interfaz **Poder** la cual tiene un solo método llamado **origenPoder()**, de la misma manera, este tiene que ser implementado por las clases hijas.

Código 1. Clase abstracta SpiderPeopleVerso.

```
…

public abstract class SpiderPeopleVerso implements Poder{
    …

    abstract String tipoHabilidad(int tipo);

    …
}
```

Código 2. Interfaz Poder.

```
…

public interface Poder {
        void origenPoder ();
}
```

La implementación del polimorfismo esta en crear clases que hereden las características de la clase padre, teniendo así tres diferentes clases hijas **Spiderman**, **SpiderAnimal** y **Simbionte**.

Código 3. Clase Spiderman.

```
…

public class Spiderman extends SpiderPeopleVerso{

	public Spiderman(String nombreCivil, String nombreSpider, int universo) {
		super(nombreCivil, nombreSpider, universo);
	}

	@Override  // Implementación clase abstracta
	String tipoHabilidad(int tipo) {
		…
		return habilidad;
	}

	@Override  // Implementación interfaz
	public void origenPoder() {
		System.out.println("Origen Poder: Picado por una araña radioactiva\n");
	}

}
```

Código 4. Clase SpiderAnimal.

```
…

public class SpiderAnimal extends SpiderPeopleVerso{

	public SpiderAnimal (String nombreCivil, String nombreSpider, int universo) {
		super(nombreCivil, nombreSpider, universo);
	}

	@Override  // Implementación clase abstracta
	String tipoHabilidad(int tipo) {
		…
		return habilidad;
	}

	@Override  // Implementación interfaz
	public void origenPoder() {
		System.out.println("Origen Poder: Experimento Laboratorio\n");
	}

}
```

Código 5. Clase Simbionte.

```
…

public class Simbionte extends SpiderPeopleVerso{

	public Simbionte (String nombreCivil, String nombreSpider, int universo) {
		super(nombreCivil, nombreSpider, universo);
	}

	@Override  // Implementación clase abstracta
	String tipoHabilidad(int tipo) {
		…
		return habilidad;
	}

	@Override  // Implementación interfaz
	public void origenPoder() {
		System.out.println("Origen Poder: Uso de simbionte\n ");
	}
}
```

Donde en la clase **Principal** se declara un arreglo padre de clase **SpiderPeopleVerso**, que contiene elementos de sus clases hijas y en un método estático **personajes**, propio de la clase Principal, que imprimen las características y comportamientos de cada uno de las clases hijas.

Código 6. Clase Principal.

```
…

public class Principal {

	public static void main(String[] args) {

		SpiderPeopleVerso[] spiderVerso = {
				new Spiderman("Peter Parker", "Spiderman", 19999)
				, new SpiderAnimal("Peter Porker", "Spider-Ham", 8311)
				, new Simbionte("Eddie Brock", "Venom", 616)
		};

		personajes(spiderVerso);  // Imprimir caracteristicas
	}

	private static void personajes(SpiderPeopleVerso[] spiderVerso) {
		…
		for(SpiderPeopleVerso sv : spiderVerso) {
			…
			System.out.println(sv);
			…
		}
	}
}
```