Programación orientada a objetos

Abstracción

{desafío} latam_

Utilizar principios básicos de diseño orientado a objetos para la implementación de una pieza de software acorde al lenguaje Java para resolver un problema de baja complejidad

- Unidad 1: Flujo, ciclos y métodos
- Unidad 2: Arreglos y archivos
- Unidad 3: Programación orientada a objetos
- Unidad 4: Pruebas unitarias y TDD

Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Comprender una interface para separar código.*
- *Comprender las clases abstractas para generar herencia.*
- *Implementar polimorfismo para los principios de POO mediante herencia.*

{desafío} latam_

¿Dónde se puede utilizar el polimorfismo?

{desafío} latam_

/* Interfaces */

Interfaces

Se declaran utilizando la palabra reservada “interface” en lugar de “class”, y proveen una lista de prototipos de métodos, lo que significa que solo se declara tipo de retorno, nombre y parámetros de entrada.

Otras clases pueden implementar (heredar) a las interfaces, para lo que deben agregar la palabra “implements” después del nombre de la clase.

Al haber implementado una interface, la clase podrá contener sus propias versiones de los métodos de la interface.

Interfaces

```
     //Creación de interface
     public interface nombreInterface{
          void imprimirHola();
     }
     //Implementación de interface:
     public class nombreClase implements nombreInterface[,   nombreOtraInterface]{
     @Override
     public void imprimirHola(){
          System.out.println("hola");
          }
}
```

Interfaces

Además de estos prototipos de método, en las interfaces se pueden crear constantes que, como su nombre lo indica, sirven para almacenar valores (como las variables) que no cambiarán.

Las interfaces permiten conocer la lista de métodos que tendrán las clases que las implementen sin conocer el comportamiento específico de cada una, ya que cada implementación puede ser diferente.

Los métodos que se declaran en la interface deben existir en todas las clases que la implementen, por ende, ayudan a establecer la forma de las clases, lo que define protocolos de comunicación fácilmente, además de hacer la aplicación de fácil entendimiento para quienes tengan que modificarla.

Interfaces

Ejemplo

Al utilizar un control remoto de televisor, presionamos los botones sabiendo lo que hacen, pero no nos interesa saber cómo lo hacen, solo queremos que haga lo que le pedimos que haga.

Las interfaces no ayudan mucho con la reutilización de código, pero lo mantienen ordenado.

Ejercicio guiado

El juego

Crearemos una porción de la estructura de un juego donde solo se puede avanzar hacia adelante y saltar.

PASO 1
Crear un proyecto en Eclipse y crear la siguiente interface dentro de un package llamado Interfaces:

```
public interface Personaje {
     void mover(int x);
}
```

El juego

Crearemos una porción de la estructura de un juego donde solo se puede avanzar hacia adelante y saltar.

PASO 2
Crear un package Personajes y una implementación de Personaje llamada Protagonista.

```
public class Protagonista implements Personaje{
}
```

El juego

Crearemos una porción de la estructura de un juego donde solo se puede avanzar hacia adelante y saltar.

PASO 3

Importar la interface Personaje dentro de la clase Protagonista, y Eclipse arrojará un mensaje diciendo que debemos implementar los métodos de dicha interface, dándonos la opción de hacerlo automáticamente, tal como se muestra:

```
package Personajes;

import Interfaces.Personaje;

public class Protagonista implements Personaje {
}
```

El juego

Crearemos una porción de la estructura de un juego donde solo se puede avanzar hacia adelante y saltar.

PASO 4

Dar clic a la primera opción para que Eclipse genere las implementaciones por nosotros; la clase quedaría así:

```
public class Protagonista implements Personaje{
	@Override
		public void mover(int x) {
			// TODO Auto-generated method stub
		}
}
```

El juego

Crearemos una porción de la estructura de un juego donde solo se puede avanzar hacia adelante y saltar.

PASO 5

Agregar una variable llamada xActual para indicar la posición del personaje y que mover() modifique esa variable.

```
public class Protagonista implements Personaje{
   private int xActual;
   @Override
   public void mover(int x){
   xActual = xActual + x;
  }
}
```

El juego

Crearemos una porción de la estructura de un juego donde solo se puede avanzar hacia adelante y saltar.

PASO 6

Crear otra clase llamada Enemigo en el package Personajes que implementa la interface Personaje. Esto con la idea de crear comportamientos específicos para Enemigo y Protagonista, es decir, el Enemigo avanzará desde el punto A al punto B de una forma progresiva y el Protagonista se moverá instantáneamente de un punto a otro.

```
public class Enemigo implements Personaje{
      private int xActual;
            @Override
            public void mover(int x){
                        while(xActual < x){
                                    xActual++;
                        }
            }
}
```

El juego

Crearemos una porción de la estructura de un juego donde solo se puede avanzar hacia adelante y saltar.

PASO 7

Crear una Interface de Jugador que le permita a las clases implementar un comportamiento de jugador.

```
package Interfaces;
   public interface Jugador {
          void saltar();
          void ejecutarAccion(String accion);
}
```

El juego

Crearemos una porción de la estructura de un juego donde solo se puede avanzar hacia adelante y saltar.

PASO 8

Implementar la interface Jugador en la clase Protagonista.

```
package Personajes;
import Interfaces.Jugador;
import Interfaces.Personaje;
public class Protagonista implements Personaje , Jugador {
     …
}
```

El juego

Crearemos una porción de la estructura de un juego donde solo se puede avanzar hacia adelante y saltar.

PASO 9

Agregar los métodos que no hemos implementado gracias a la opción de la imagen y la clase queda así:

```
private int xActual;
@Override
public void mover(int x){
	xActual = xActual + x;
}
@Override
public void saltar() {
	// TODO Auto-generated method stub
}
@Override
public void ejecutarAccion(String accion) {
	// TODO Auto-generated method stub
 }
```

El juego

Crearemos una porción de la estructura de un juego donde solo se puede avanzar hacia adelante y saltar.

PASO 10

Crear un comportamiento específico para el protagonista. Primero, modificar la implementación de saltar(), agregar una variable representando el eje "y" del plano (yActual), que actualmente tiene solo una dimensión y, posteriormente, haremos que el personaje suba y

```
private int yActual = 1;
@Override
public void saltar() {
	//Aumentamos hasta 5
		while(yActual < 5){
		yActual++;
		}
		//Cuando sea 5, disminuimos a 1 nuevamente
		while(yActual > 1){
				yActual--;
		}
}
```

El juego

Crearemos una porción de la estructura de un juego donde solo se puede avanzar hacia adelante y saltar.

PASO 11

Crear la implementación del método ejecutarAccion (String accion).

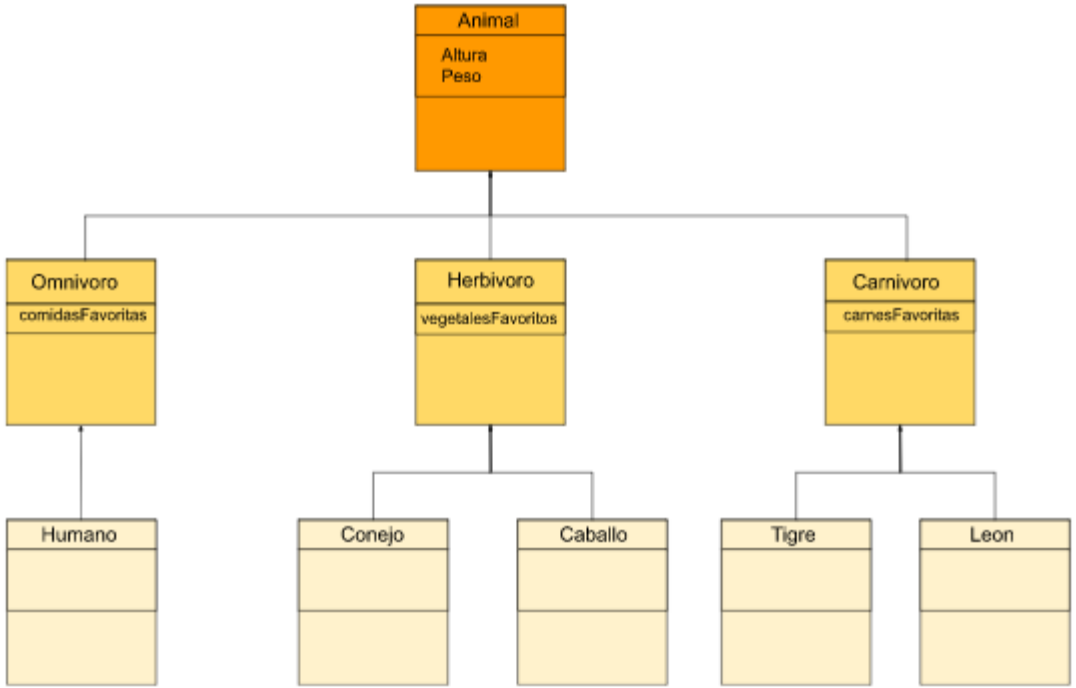
```
@Override
public void ejecutarAccion(String accion) {
     if(accion.equals("saltar") && yActual == 1){
                    saltar();
          } else if(accion.equals("avanzar")){
                    mover(1);
          }
}
```

/* Clases abstractas */

Clases abstractas

Son clases que no pueden instanciarse, dado que permiten categorizar a otras.

Por ejemplo, la clase Animal que posee una herencia, debiera ser abstracta, ya que la palabra Animal la asociamos al concepto de animal como tal, pero no es algo tangible.



Ejercicio guiado

Animales

Paso 1

Crear una clase llamada Animal que contiene la palabra reservada abstract antes de la palabra class. Esto significa que la clase no puede ser instanciada, generamos los getter and setter correspondientes de las variables altura y peso.

```
public abstract class Animal {
        private int altura;
                private int peso;
                public int getAltura() {
                                return altura;
                }
                public void setAltura(int altura) {
                                this.altura = altura;
                }
                public int getPeso() {
                                return peso;
                }
                public void setPeso(int peso) {
                                this.peso = peso;
                }
```

Animales

Paso 2

Terminar el árbol de carnívoros, creando la clase abstracta Carnivoro.

```
import java.util.List;
public abstract class Carnivoro extends Animal{
          List<String> carnesFavoritas;
     public List<String> getCarnesFavoritas() {
           return carnesFavoritas;
          }
     public void setCarnesFavoritas(List<String> carnesFavoritas) {
          this.carnesFavoritas = carnesFavoritas;
          }
}
```

Animales

Paso 3 y 4

Crear la clase Tigre y León que extienden de Carnivoro.

```
//Tigre
public class Tigre extends Carnivoro{
}
```

```
//Leon
public class Leon extends Carnivoro{
}
```

Animales

Paso 5

Agregar un método main para crear esta lista polimórfica con clases abstractas.

```
public class Main {
            public static void main(String[] args) {
                        Leon leon = new Leon();
                        Tigre tigre = new Tigre();
                        ArrayList<Animal> listaAnimales = new ArrayList<>();
                        ArrayList<Carnivoro> listaCarnivoros = new ArrayList<>();
                        listaAnimales.add(leon);
                        listaAnimales.add(tigre);
                        listaCarnivoros.add(leon);
                        listaCarnivoros.add(tigre);
                        System.out.println(listaAnimales);
                        System.out.println(listaCarnivoros);
            }
}
```

Abstracción en POO

La abstracción es la realización de polimorfismo en busca de características comunes en objetos, todo esto con la finalidad de trabajarlos dentro de un mismo contexto.

¿Cómo podemos implementar una interface?

¿Qué se puede lograr gracias al Polimorfismo?

¿Qué son las clases abstractas?

Próxima sesión...

- *Desafío guiado*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales