ADRIAN MAXIMILIANO MUNOZ MARTINEZ

Laboratorio POO  Sabado 7 am – 9am

PRactica 11: Anotaciones

**Introducción**

Se desarrollará una aplicación en Java que en la que se implemente colecciones genéricas.

**Objetivo**

El objetivo de la practica es familiarizarnos con el concepto de colecciones genéricas y objetos genéricos, así como practicar su implementación y conocer sus ventajas y desventajas.

**Practica**

Practica11.java

package practica11;

import java.util.Scanner;

public class Practica11 {

public static void main(String[] args) {

Automovil auto = new Automovil();

Scanner sc = new Scanner(System.in);

// Imprir en donde se usar ciertas anotacions

System.out.println("@Deprected: Se utlizo para indicar que la funcion");

System.out.println("encender de la clase TransporteTerrestre esta en desuso");

System.out.println("@customAnnotation: Se utlizo para crear Autmovil");

// Pedir informacion al usario acerca del las caracteristicas del auto

System.out.print("\nIngresa la marca del automvil: ");

auto.setMarca(sc.nextLine());

System.out.print("Ingresa el modelo del automovil: ");

auto.setModelo(sc.nextLine());

System.out.print("Ingresa el precio del automovil: ");

auto.setPrecio(sc.nextDouble());

// Imprimir los atributos y llamar a los metodos del objeto auto tipo Automovil

System.out.println("\nDatos del Automovil");

System.out.println("Marca: " + auto.getMarca());

System.out.println("Modelo: " + auto.getModelo());

System.out.println("Precio: " + auto.getPrecio());

// Pedir al usario el argumento de la funcion avanzar

System.out.println("Ingres la cantidad de gas que desea ponerle al Automovil: ");

System.out.println(auto.avanzar(sc.nextInt()));

System.out.println(auto.apagar());

System.out.println("@Override: Se utlizo para cambiar el funcionamiento de Apagar");

System.out.println("en la clase automovil");

}

}

Automovil.java

package practica11;

//Anotacion personalizada para almacenar autores y fechas de modificacion

@CustomAnnotation(lastModified = "31/12/1999")

public class Automovil extends TransporteTerrestre {

public Double precio;

public String avanzar(int gas) {

// Validar que haya suficiente gas antes de avanzar

if (gas > 1) {

return "El auto se esta moviendo";

}

return "No hay suficiente gasolina";

}

@Override

public String apagar() {

return "El auto se ha apagado";

}

public void setPrecio(Double precio) {

this.precio = precio;

}

public Double getPrecio() {

return this.precio;

}

}

TransporteTerresetre.java

package practica11;

public abstract class TransporteTerrestre {

private String marca;

private String modelo;

// Atributo en desuso

@Deprecated

public int gas;

// Funcion en obsoleta por variable en desuso

@Deprecated

public void encender(int gas) {

this.gas = gas;

}

public String apagar() {

return "El vehiculo se ha apagado";

}

public void setMarca(String marca) {

this.marca = marca;

}

public String getMarca() {

return this.marca;

}

public void setModelo(String modelo) {

this.modelo = modelo;

}

public String getModelo() {

return this.modelo;

}

}

CustomAnnotation.java

package practica11;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Documented;

//Esta anotacion servira para especificar los autores y las modifcaciones

@Documented // Esta anotacion se documentara en Java Docs

@Retention(RetentionPolicy.RUNTIME) // Permitir el acceso de lo elementos de esta funcion en RUNTIME

public @interface CustomAnnotation {

String author() default "Jhon Lennon";

String lastModified();

}

**Screenshots**

**Explicación**

Se creo una aplicación que implemento un objeto Automóvil. El cual heredo parte de su funcionalidad de la clase TransporteTerrestre, algunos elementos de la clase Automovil y la clase TransporteTerrestre contaban con ciertas anotaciones.

Dentro de la clase TransporteTerrestre se usaron las anotaciones de @Deprecated para indicar que algún elemento estaba obsoleto y no debería de usarse en el código, ya que existía una mejor alternativa o este no era seguro.

La clase Automovil contaba con una anotación personalizada, @CustomAnnotation, esta anotación ayuda a mantener un registro de la fecha y los autores en la creación o modificación de las diferentes partes de la aplicación, fue creada usando las siguientes meta-anotaciones:

* @Documented la cual se asegura de que la anotacion sea documentada por aplicaciones como JavaDoc.
* @Retention para indicar que los elementos de esta anotacion podran ser accesados durante la ejecucion de la la aplicación.

Por ultimo la clase Automovil tambien cuenta con la anotacion @Override la cual indica que los metodos heredados de la clase abtracta TransporteTerrestre seran sobreescritos.

**Conclusión**

Las anotaciones son una forma de añadir metadatos y varias funcionalidades, algunas nos ayudan a evitar errores en el desarrollo de la aplicación como es el caso de @Override y @Deprecated. Mientras que otras nos permiten generar una mejor documentación como @Documented y @CustomAnnotation. Ya que Java cuenta con un amplio numero de anotaciones y nos facilita la creación de anotaciones personalizadas, las funciones de estas abarcan mucho mas de lo que pudimos ver en la practica.