ADRIAN MAXIMILIANO MUNOZ MARTINEZ

Laboratorio POO  Sabado 7 am – 9am

PRactica 10: Java Generics

**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**

Practica10i.java

package practica10i;

import java.util.ArrayList;

import java.util.Scanner;

public class Practica10i {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int i, j;

Telefono nuevo\_movil;

String[] propiedades = { "Nombre", "Gamma", "Color" };

String[] elements = new String[3];

ArrayList<Telefono> telefonos = new ArrayList<>();

for (i = 0; i < 4; i++) {

System.out.println();

if ((i % 2) == 0) {

System.out.print("Creando telefono de tipo Apple: \n");

nuevo\_movil = new MovilApple();

} else {

System.out.print("Creando telefono de tipo Samsung: \n");

nuevo\_movil = new MovilSamsung();

}

System.out.print("\tIngresa el nombre: ");

elements[0] = sc.nextLine();

System.out.print("\tIngresa el modelo: ");

nuevo\_movil.setModelo(sc.nextLine());

System.out.print("\tIngresa la gamma: ");

elements[1] = sc.nextLine();

System.out.print("\tIngresa el color: ");

elements[2] = sc.nextLine();

System.out.print("\tEstas son las propiedades del telefono: ");

j = 0;

for (String element : nuevo\_movil.propiedades(elements)) {

System.out.print(String.format("\n\t%s: %s", propiedades[j], element));

j++;

}

System.out.println();

telefonos.add(nuevo\_movil);

}

// Imprimir los atributos y llamar las funciones de cada objeto Movil en el

// array list

j = 0;

for (Telefono movil : telefonos) {

// Ya que la declaracion generica de tipo dentro del ArrayList fue de tipo

// Telefono

// Solo podemos usar los metodos de encapsulamiento del atributo modelo

// Y el metodo de propiedades que fue usado arriba

// Los elementos que estan dentro del array solo pueden ser de tipo

// telefono

// Pero pueden instanciarse con sus clases hijas haciendo polimorfismo

// Y sobrescribiendo los metodos de Telefono en run-time

System.out.println(String.format("\nTelefono %d", ++j));

System.out.println(String.format("\tModelo del telefono %s", movil.getModelo()));

}

}

}

Dispositivoelectronico.java

package practica10i;

public interface DispositivoElectronico {

public abstract String encender();

public abstract String apagar();

}

Telefono.java

package practica10i;

public abstract class Telefono {

private String modelo;

public abstract String[] propiedades(String[] element);

public String getModelo() {

return modelo;

}

public void setModelo(String modelo) {

this.modelo = modelo;

}

}

MovilApple.java

package practica10i;

public class MovilApple extends Telefono implements DispositivoElectronico {

private int precio;

public String encender() {

return "El telefono se ha encendido";

}

public String apagar() {

return "El telefono se ha apagado";

}

public void hacerLlamada(int tel) {

System.out.println(String.format("Llamando a %d", tel));

}

public void hacerLlamada(String contacto) {

System.out.println(String.format("Llamando a %s", contacto));

}

public void finalizarLlamada() {

System.out.println("Finalizando llamada");

}

public void setPrecio(int precio) {

this.precio = precio;

}

public int getPrecio() {

return this.precio;

}

@Override

public String[] propiedades(String[] elemnts) {

return elemnts;

}

}

MovilSamsung.java

package practica10i;

public class MovilSamsung extends Telefono implements DispositivoElectronico {

private int precio;

public String encender() {

return "El telefono se ha encendido";

}

public String apagar() {

return "El telefono se ha apagado";

}

public void hacerLlamada(int tel) {

System.out.println(String.format("Llamando a %d", tel));

}

public void hacerLlamada(String contacto) {

System.out.println(String.format("Llamando a %s", contacto));

}

public void finalizarLlamada() {

System.out.println("Finalizando llamada");

}

public void setPrecio(int precio) {

this.precio = precio;

}

public int getPrecio() {

return this.precio;

}

@Override

public String[] propiedades(String[] elemnts) {

return elemnts;

}

}

**Screenshots**

**Imagen que contiene texto

Descripción generada automáticamenteImagen que contiene pájaro, ave

Descripción generada automáticamente**

**Explicación**

Se creo una aplicación que implemento un arreglo genérico de tipo Telefono, el cual almacenaría 4 dispositivos móviles instanciados con sus clases hijas, esto se hizo para poder almacenar objetos de tipo MovilApple y MovilSamsung dentro del arreglo.

Los atributos y las propiedades de cada objeto fueron ingresadas por el usuario.

Al final del programa se llamaron todos los métodos disponibles de los elementos del arreglo.

Ya que la declaración generica de tipo dentro del ArrayList fue de tipo Teléfono. Solo se llamaron los métodos de encapsulamiento del atributo modelo, y el método de propiedades que fue usado cuando se le pidió al usuario ingresar las propiedades.

Ya que estos son los únicos métodos que se pueden llamar desde un objeto de tipo Telefono, pero como estos objetos fueron instanciados con la hijas de Telefono el comportamiento de estos fue el que se sobrescribió en estas.

**Conclusión**

Las colecciones genéricas agilizan y simplifican mucho la implementación del código cuando sabemos que la colecciones solo va a almacenar un tipo de dato en especifico, pero al mismo tiempo nos roban un poco de la flexibilidad que una colección no genérica ofrece.