Programación II

Práctica 04: Objetos

Versión del 21/04/2022

Introducción

En la siguiente práctica se utilizarán los conceptos de: herencia, sobrescritura, polimorfismo, *abstract*, *extends* e *implements*.

Se pide Justificar cuando:

- Se utilicen métodos de clase.
- Se utilice *@Override.*

Notas:

- Si no utilizan esos conceptos en ningún momento, seguramente hay que corregir el ejercicio!
- Implementar el método ***toString()*** para realizar los testeos, en lugar de métodos "imprimir" particularizados.

Ejercicio 1

Realizar el diagrama de clases, incluir las relaciones existentes e implementar la siguiente situación.

a) Una computadora consta de una marca, un modelo y un año de fabricación. Para distinguir entre las computadoras, se les ha asignado un tipo: Tipo D para desktop, tipo A para All-in-one y tipo L para Laptop.

b) Cada computadora posee 3 características básicas, independiente de su tipo. Todas poseen un *disco rígido*, *un procesador* y *memoria*. El *disco rígido* se caracteriza por su marca, su capacidad en gigabytes, y su velocidad de operación en RPM[1]. Por su lado, los *procesadores* tienen marca, modelo y velocidad en gigahertz. De la *memoria* interesa solo su capacidad en Gigabytes.
   - Cada computadora debe tener la posibilidad de mostrar los valores suyos y de sus componentes, por ejemplo, si quiero imprimir los valores de una computadora, puedo imprimir: "Toshiba G480 -- Procesador: AMD -- Disco: 500gb -- Ram:4gb".

c) Probar las clases generadas creando un código cliente de éstas. Se espera que escriban una clase *main*, donde se crean computadoras y se muestra su información.

d) Agregarle a la computadora el comportamiento de encendido y apagado. La computadora debe permitir encenderla y apagarla, así como también consultar su estado actual. Recordar que, al encender una computadora, también se encienden su disco rígido y su procesador.

Ejercicio 2

Realizar el diagrama de clases y la implementación de las clases.

Utilizar **abstract** cuando sea necesario.

[1] Revoluciones Por Minuto

a) Realizar el diagrama de clases y la implementación de las clases:
   - Perro
   - Cocker
   - Caniche

   Con las operaciones :
   - String ladrar()
   - Int cantidadPatas()

b) Agregar los métodos necesarios para modelar el predicado “Perro que ladra no muerde”.
   - Se sabe que el cocker no ladra.
   - Se sabe que el caniche ladra.

c) El método ladrar debe informar la situación del perro.

Ejercicio 3

En los constructores se puede utilizar **super** cuando sea necesario.

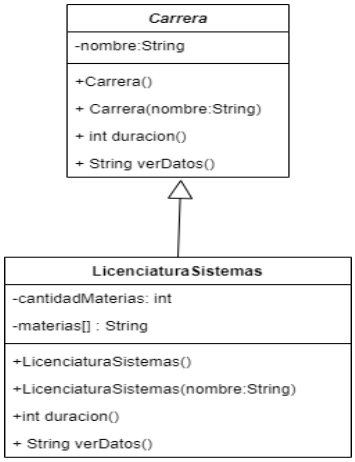
a) Realizar el diagrama de clases y la implementación de las clases:
   - Vehículo
   - Vehículo CuatroRuedas
   - Vehículo NRuedas
   - Automovil
   - JetSky//MotoNieve
   - Barco
   - Triciclo

   Con las operaciones:
   - Int cantidadRuedas()
   - String nombre()

b) Agregar los métodos necesarios para modelar:
   - La posición de cada vehículo en el mapa
   - Y la posibilidad de moverlos.

Ejercicio 4

a) Marcar en el siguiente ejemplo cuando se utilizan, sobrecarga, sobreescritura, métodos abstractos. ¿Porque la clase Carrera es abstracta?



Ejercicio 5

Como se rediseñaría el diagrama de clase del ejercicio 1 si sabemos que las computadoras de escritorio tienen monitor, los datos que tenemos del monitor son su tamaño en pulgadas, resolución y marca. Las laptops poseen batería de las cuales sabemos su duración en horas.

Modificar el diagrama de clases y luego cambiar la implementación en base al nuevo diagrama.

Ejercicio 5

En una empresa hay empleados, con distintos regímenes. Empleados comunes que tienen un sueldo básico y sobre este se aplica un descuento de 11% por aporte jubilatorio. Hay vendedores que también son empleados, que sobre el sueldo básico cobran una comisión sobre las ventas realizadas, cada mes se registra importe vendido para poder calcular cuánto se les debe abonar en concepto de comisión, también se aplica el descuento de 11%. En ambos casos se tienen los datos básicos, nombre, apellido, DNI, teléfono. Por último también hay empleados contratados, estos no poseen sueldo básico, se les paga un valor hora y se registra cuantas horas trabajan mensualmente, también se poseen los datos de estos empleados, en este caso no se realizan descuentos jubilatorios.

a) Modelar las clases necesarias e implementar. ¿Qué conceptos se aplicaron?

b) La empresa registra todos sus empleados y necesita imprimir los sueldos de éstos. (Para simular la impresión de sueldos, mostrar la información por pantalla). Modificar el diagrama de clases y la implementación. ¿Qué conceptos se estarían aplicando?

c) Todos los empleados tienen una cuenta bancaria, entonces además de ser empleados también son clientes de banco y por esto saben extraer dinero. Para extraer dinero de su cuenta se debe indicar el importe, y automáticamente se debitará de su cuenta. Las cuentas de banco solo tienen un número y el saldo.

Ejercicio 6

Considere las clases:

Tupla: Que representa un vector de dos elementos
Coordenada: Que representa una coordenada cartesiana.
Pixel: Que agrega un color a la coordenada.

```
public class Tupla<E1,E2> {

	private E1 e1;
	private E2 e2;

	public Tupla(E1 e1, E2 e2){
		this.e1= e1;
		this.e2 = e2;
	}

	public E1 getE1() {		return e1;	}

	public void setE1(E1 e1) {		this.e1 = e1;}

	public E2 getE2() {		return e2;	}

	public void setE2(E2 e2) {		this.e2 = e2;}

	public void sumar(Tupla t){
		//Implementacion de: setE1(getE1 + t.getE1)
		if (t.getE1() instanceof String && getE1() instanceof String){
			setE1((E1)(getE1().toString()+ t.getE1().toString()));
		}

		if (t.getE2() instanceof String && getE2() instanceof String){
			setE2((E2)(getE2().toString()+ t.getE2().toString()));
		}
	}

}

public class Coordenada extends Tupla{

	public Coordenada(Integer x, Integer y){
		super(x,y);
	}

	@Override
	public void sumar(Tupla t){
		super.setE1((Integer)super.getE1() + (Integer)t.getE1());
		super.setE2((Integer)super.getE2() + (Integer)t.getE2());
	}

}
```

```
public class Pixel extends Coordenada{

	private int color;

	public Pixel(Integer x, Integer y, Integer color){
		super(x,y);
		this.color = color;
	}

	@Override
	public void sumar(Tupla t){
		super.setE1((Integer)super.getE1() + (Integer)t.getE1());
		super.setE2((Integer)super.getE2() + (Integer)t.getE2());
	}

}

public class Test {
	public static void main(String[] args) {
		Tupla<String,String> t1 = new Tupla<String,String>("a","b");
		Tupla<String,String> t2 = new Tupla<String,String>("c","d");

		Coordenada c1 = new Coordenada(1,2);
		Coordenada c2 = new Coordenada(1,2);

		t1.sumar(t2);

		c1.sumar(c2);

		Tupla<Integer,Integer> t3 = new Pixel(1,2,3);
		Tupla<Integer,Integer>t4 = new Pixel(1,2,3);

		t3.sumar(t3);
		t3.sumar(c2);

		System.out.println((String)t1.getE1());
		System.out.println(c1.getE1());

	}

}
```

a) ¿Qué versión de sumar se ejecutará en cada caso?
b) Implementar un toString eficiente (que reutilice código) en cada clase
c) Que sucede si declaro
   Pixel x = new `Tupla<Integer,Integer>`
   ¿Porque?

d) Implementar la suma de Pixel.
e) Implementar la comparación de tuplas “coordenada a coordenada”.

Realizar las modificaciones necesarias en Tupla, de manera que sus coordenadas sean comparables, para poder realizar el punto e).

Ejercicio 7

Class UnidadMedida
int sumar(UnidadMedida u) // suma **u** a **this** y devuelve el resultado

Class Metro
int sumar(…)

Class Kilometro
int sumar(…)

(Ayuda: Un kilómetro son 1000 metros)

a) Reimplementar el diagrama de clases de manera que se utilice:
   - herencia, sobrecarga y sobrescritura.
b) Armar un ejemplo donde se utilice sobrecarga y otro distinto donde se utilice sobrescritura.

Ejercicio7

UpCasting

Class Persona
Public void asignarNombre(String nombre)
Public void asignarEdad(int edad)

Class Amigo extends Persona
@Override
Public void asignarEdad(int edad)
Public void asignarTelefono(int telefono)

a. Decidir en cada caso, que método se ejecuta (el método de la clase Persona o el de la clase Amigo)
b. En qué casos da un "*error de compilación*", y en qué casos da un "*error en tiempo de ejecución*".

```
Persona p = new Amigo();

p.asignarNombre("Juan");
p.asignarEdad(22);
 p.asignarTelefono(44444444);
p.toString();
```