

Programación II

Práctica 06: Objetos

Versión del 01/05/2016

Introducción

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

Justificar :

Cuando utilizar métodos de clase.

Cuando se utiliza *@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.

Ejercicio1

Utilizar **abstract** cuando sea necesario.

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

Perro

Cocker

Caniche

Con los métodos:

String ladrar()

Int cantidadPatas()

- b) Agregar los métodos necesarios para modelar el predicado "Perro que ladra no muerde".

Modelar que el cocker no ladra ni muerde

Modelar que el caniche ladra y no muerde

Ejercicio2

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

a) Realizar el diagrama de clases y la implementación de las clases:

Vehiculo
VehiculoCuatroRuedas
VehiculoNRuedas
Automovil
JetSky //MotoNieve
Barco
Triciclo

Con los métodos:

Int cantidadRuedas()
String nombre()

b) Agregar los métodos necesarios para modelar:

La posición de cada vehículo en el mapa
Y la posibilidad de moverlos.

Ejercicio3

Sobrecarga
Sobreescritura
Polimorfismo

a) Dar un ejemplo de cada uno.

b) Marcar en el siguiente ejemplo cuando se utilizan cada uno de los conceptos

Classs A
 metodo1(String a)
 metodo1(int a)
 metodo2(int a)

Classs B
 metodo2(int a)

Ejercicio4

Dar tres ejemplos distintos que utilicen los modificadores "final", "static", "protected".

Ejercicio5

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) Que versión de sumar se ejecutara 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).

Ejercicio6

Class UnidadMedida

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

Class Metro

```
int sumar(...)
```

Class Kilometro

```
int sumar(...)
```

ayuda: Asumir que un kilómetro son 1000 metros.

- a) Reimplementar el diagrama de clases de manera que se utilice:
 - a. 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)
```

Decidir en cada caso, que método se ejecuta (el método de la clase Persona o el de la clase Amigo)
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();
```