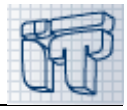


NOMBRE:
APELLIDOS:
NIA:
GRUPO:

# Primer parcial

# 2ª Parte: Problemas (7 puntos sobre 10)

Duración: 80 minutos
Puntuación máxima: 7 puntos
Fecha: 12 de marzo de 2019

Instrucciones para el examen:

- No se permite el uso de libros o apuntes, ni tener teléfonos móviles u otros dispositivos electrónicos encendidos. Incumplir cualquiera de estas normas puede ser motivo de expulsión inmediata del examen.
- Rellena tus datos personales antes de comenzar a realizar el examen.

## Ejercicio 1 (2 / 7 PUNTOS)

Escriba un programa que gestione una tienda de pantalones.

### *Apartado 1 (1 punto):*

Crear la clase pantalón con la siguiente información:

a) Tendrá que tener seis atributos privados, `Talla` (número entero expresado en centímetros), `Marca`, `Precio`, `Precio_Total` (atributo de clase que va a contener siempre la suma del precio de todos los Pantalones que existen en la tienda) y `Código`

b) Dos métodos constructores, uno sin parámetros y otro constructor donde se guarde la marca, el precio y la talla.

c) Los métodos set y get correspondientes.

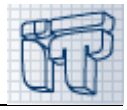
d) Un método `toString` que nos devuelve un `string` de la forma "Pantalon… marca: xxx talla: yyy precio: ppp".

### *Apartado 2 (1 punto):*

Crear el método `main` principal en el que hay que:

a) Crear un array con todos los pantalones que queramos (mínimo 5 pantalones) y los muestre por pantalla.

b) Mostrar por pantalla cual es el pantalón más caro (recorriendo el array).



## Ejercicio 2 (3 / 7 PUNTOS)

Una escudería de Fórmula 1 necesita un software de simulación de circuitos para optimizar el rendimiento de sus coches.

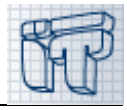
La idea es analizar la velocidad del coche en cada tramo del circuito, siendo cada tramo como si fuera una pieza de "*scalextric*".

Lo que se pretende es calcular la velocidad de salida del coche en cada tramo y realizar simulaciones con distintas variables (temperatura de los neumáticos, peso del coche, etc.) y así obtener el mejor rendimiento posible.

Para ello la escudería dispone de una interfaz de nombre *Sectionable* que tendrá un método *computeSpeed* que devolverá un *double* y tendrá como parámetro un objeto de la clase *RacingCar* que representará el coche de carreras. El valor devuelto será la velocidad del coche al finalizar el tramo.

La clase *RacingCar* tiene tres atributos que son la velocidad, el peso y la temperatura de sus neumáticos. Esta clase ya está implementada y tiene los siguientes métodos (*getters* y *setters* para sus atributos):

```
public double getSpeed()
public void setSpeed(double speed)
public double getWeight()
public void setWeight(double weight)
public double getTempWheels()
public void setTempWheels(double tempWheels)
```



## *Apartado 1 (0,5 puntos)*

Defina la interfaz descrita anteriormente.

## *Apartado 2 (2,5 puntos)*

Los tramos con los que trabaja el simulador serán *únicamente* rectas y curvas, pudiendo éstas últimas ser dextrógiras (a derechas) o levógiras (a izquierdas).

De cualquier tipo de tramo (recta o curva) el simulador necesita conocer el *grip* (adherencia), la humedad y la longitud del mismo. Todos estos datos son de tipo *double*.

Los ingenieros de la escudería han llegado a la conclusión de que la velocidad del coche al final del tramo se obtiene de la siguiente manera:

En el caso de la recta:

Velocidad de salida = longitud * (velocidad del coche – grip)

En el caso de la curva dextrógira:

Velocidad de salida = longitud * (velocidad del coche + temperatura de ruedas)

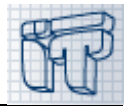
En el caso de la curva levógira:

Velocidad de salida = longitud * (velocidad del coche - humedad)

Diseñe las clases necesarias para manejar los diferentes tipos de tramo teniendo en cuenta que deben implementar la interfaz definida en el primer apartado y que el cálculo de la velocidad de salida del coche obtenido al invocar el método *computeSpeed* siga la especificación anterior.

Tenga en cuenta que su solución debe intentar que el código sea lo más reutilizable posible, pues los ingenieros pueden cambiar las fórmulas de cálculo. Es muy posible que se decidan añadir nuevos métodos que tendrán distinta implementación en cada tipo de tramo.

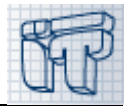
Suponga implementados los constructores de las clases y los métodos *getters* y *setters* de los atributos, no siendo por tanto necesario codificarlos. Tan sólo defina los atributos y codifique el método *computeSpeed* en las clases necesarias.



## Ejercicio 3 (2 / 7 PUNTOS)

El método `PasswordCheck.measure(String)` mostrado a continuación calcula una medida de calidad de la contraseña que recibe por parámetro. Se pide completar una tabla con las contraseñas (y su medida de calidad correspondiente a cada una) que sean necesarias para que un test de caja blanca consiga una cobertura de código del 100% sobre el método `measure`.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class PasswordCheck {
/* El método measure(String) de la clase PasswordCheck devuelve un número de tipo int
con la medida de la calidad de la contraseña (mayor valor --> mejor contraseña).
Para este algoritmo, se definen las siguientes clases de caracteres:
   -  letras minúsculas (A..Z)
   -  letras mayúsculas (a..z)
   -  números (0..9)
   -  resto de caracteres (símbolos, espacios, ...)
El método realiza una serie de comprobaciones y va sumando o restando puntos según
la siguiente tabla (todas las condiciones se comprueban y calculan
independientemente):
   -  Si tiene estrictamente más de 8 caracteres de longitud: +1
   -  Si tiene estrictamente menos de 8 caracteres de longitud: -2
   -  Si tiene 12 o más caracteres de longitud: +1

   -  Si tiene dos o más mayúsculas: +1
   -  Si tiene dos o más minúsculas: +1
   -  Si tiene dos o más dígitos numéricos: +1
   -  Si tiene uno o más caracteres de otro tipo: +1 por cada carácter de otro tipo
Se calcula la suma de las puntuaciones y se devuelve el resultado de la suma. */
    public static int measure(String password) {
        int result = 0;

        int countUp = countCharClass(password, "[A-Z]");
        int countDown = countCharClass(password, "[a-z]");
        int countNum = countCharClass(password, "[0-9]");
        int countOther = countCharClass(password, "[^A-Za-z0-9]");

        if (password.length()  >  8) result = result + 1;
        if (password.length()  <  8) result = result - 2;
        if (password.length() >= 12) result = result + 1;

        if (countUp    >= 2) result = result + 1;
        if (countDown  >= 2) result = result + 1;
        if (countNum   >= 2) result = result + 1;
        if (countOther >= 1) result = result + countOther;

        return result;
    }

    /* Devuelve la cantidad de caracteres de tipo charClass presentes en password */
    private static int countCharClass(String password, String charClass) {
        int result = 0;
        Pattern p = Pattern.compile(charClass);
        Matcher m = p.matcher(password);
        while (m.find()) result++;
```



```
        return result;
    }
}
```

# Soluciones

## Ejercicio 1. Rúbrica

La clase Pantalón correcta valdrá 0,6 puntos desglosado en:

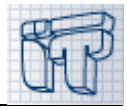
- Los atributos perfectamente inicializados todos. 0,1 puntos
- El método constructor sin parámetro. 0,05 puntos
- El método constructor con parámetros. 0,15 puntos
- Los métodos SET Y GET correspondientes. 0,15 puntos (0,01875 por método, como hay 8 métodos, todos ellos implementados correctamente suma 0,15 puntos)
- El método toString 0,15 puntos

El programa principal correcto valdrá 1,4 puntos desglosado en:

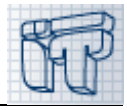
- Crear un array con todos los pantalones que queramos (mínimo 5 pantalones) y los muestre por pantalla. (0.4 puntos)
- Cuál es el pantalón más caro. (1 puntos)

## Ejercicio 1. Solución

```
package tienda;

public class Pantalon {
    private int talla = 0;
    private String marca = "";
    private double precio = 0;
    // El precio total con un static
    private static float precioTotal = 0;
    // el codigo unico del objeto
    private int codigo = 0;

    public Pantalon(String marca,double precio, int talla)
    {   this.talla = talla;
        this.precio = precio;
        this.marca = marca;
        // El precio total con un static
        this.precioTotal += precio;
    }

    public Pantalon()
    {
    }

    /**
     * @return the talla
     */
    public int getTalla()
    {
        return talla;
    }

    /**
     * @param talla the talla to set
     */
    public void setTalla(int talla)
    {
```



```
        this.talla = talla;
    }

    /**
     * @return the marca
     */
    public String getMarca()
    {
        return marca;
    }

    /**
     * @param marca the marca to set
     */
    public void setMarca(String marca)
    {
        this.marca = marca;
    }

    /**
     * @return the precio
     */
    public double getPrecio()
    {
        return precio;
    }

    /**
     * @param precio the precio to set
     */
    public void setPrecio(double precio)
    {
        // El codigo unico del objeto
        // Quito el precio que hubiera y pongo el nuevopara que cuadre
        this.precioTotal -= this.precio;
        this.precioTotal += precio;
        this.precio = precio;
    }

    // get de los campos static
    // no pongo el set, pues en este programa estos campos no se cambian
    /**
     * @return the precioTotal
     */
    public static double getPrecioTotal()
    {
        return precioTotal;
    }

    /**
     * @return the codigo
     */
    public int getCodigo()
    {
        return codigo;
    }

    // Sobrescribo el toString
    @Override
    public String toString()
    {
        String s="Pantalon... ";
        s += " marca : " + marca;
        s += " talla : " + talla;
        s += " precio : " + precio;
        return s;
    }

}

package tienda;
```



```
public class Tienda {

    public static void main(String[] args) {

        Pantalon[] arr = new Pantalon[5];
        arr[0] = new Pantalon("LEWIS",24.5,42);
        arr[1] = new Pantalon("Malboro",32.5,56);
        arr[2] = new Pantalon("Diesel",28.5,50);
        arr[3] = new Pantalon("LEWIS",36.5,40);
        arr[4] = new Pantalon("LEWIS",30.5,44);
        System.out.println("LOS PANTALONES DE LA TIENDA SON ");
        System.out.println("****************************** ");
        for (int i=0; i<arr.length;i++){
            System.out.print(" Marca  "+arr[i].getMarca());
            System.out.print(" Talla  "+arr[i].getTalla());
            System.out.print(" Precio "+arr[i].getPrecio());
            System.out.println("");
        }

        Pantalon masCaro = arr[0];
        for(int i=1;i<arr.length;i++)
        {
           if (arr[i].getPrecio()>masCaro.getPrecio())
           {
               masCaro =  arr[i];
           }
        }
        System.out.println("");
        System.out.println("EL PANTALON MAS CARO DE LA TIENDA ES  : ");
        System.out.println("****************************************** ");
        System.out.println(masCaro);
        System.out.println("");

}
```

# Ejercicio 2. Rúbrica y Solución

### Apartado 1:

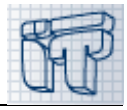
```
public interface Sectionable {
    double computeSpeed(racingCar car);
}
```

Criterios de corrección:

- 0 si en general no se sabe hacer: si se pone “class”, o “abstract” o “implement” o lo que sea distinto de “interface”
- 0 si se implementa el método
- Restar 0,25 si no se pone “public”
- Restar 0,25 si en el método se pone {} en vez de ;
- aceptar poner “public”, “abstract” o ambos al método

### Apartado 2:

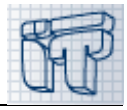
```
public abstract class RoadSection  implements Sectionable {

    private double grip;
```



```
    private double humidity;
    private double length;

/* YA IMPLEMENTADOS, NO SE PIDEN
    public roadSection(double grip, double humidity, double length){

        this.grip=grip;
        this.humidity=humidity;
        this.length=length;
    }

    public double getLength() {
        return length;
    }
    public void setLength(double length) {
        this.length = length;
    }
    public double getGrip() {
        return grip;
    }
    public void setGrip(double grip) {
        this.grip = grip;
    }
    public double getHumidity() {
        return humidity;
    }
    public void setHumidity(double humidity) {
        this.humidity = humidity;
    }
*/

}

public class StraightSection extends RoadSection {


    public double computeSpeed(RacingCar car) {

        return this.getLength() * (car.getSpeed()-this.getGrip());
    }
}

public class DextrorotatoryTurn extends RoadSection {

    public double computeSpeed(RacingCar car) {

        return this.getLength() * (car.getSpeed()+car.getTempWheels()) ;
    }
}

public class LevorotatoryTurn extends roadSection{

    public double computeSpeed(RacingCar car) {

        return this.getLength() * (car.getSpeed()- this.getHumidity());
    }
}
```

Criterios de corrección:

- 0 si en general no se sabe hacer.
- 0,5 si se hace bien la división de clases con la herencia correcta, incluyendo la implementación de la interfaz
- 0,5 si define bien la cabecera (abstract e implementa la interfaz) y define bien los atributos en la clase base.



- 0,5 por cada implementación correcta de computeSpeed en cada clase.
- Se resta -0,25 por clase si no usa bien los getters, o si los define en alguna clase.

## Ejercicio 3. Rúbrica y soluciones

Rúbrica:

- +1 punto por cada `if` de los 7 que existen, por los que se consiga pasar con las contraseñas del alumno (un alumno podría poner varias contraseñas que solapen los `if`s por los que se pasan, por eso se puntúa sobre los condicionales por los que se pasa). No es necesario que pongan un mínimo concreto de tuplas; sólo las que sean necesarias para pasar por todos los condicionales en algún momento. Lo normal que cabría esperar sería siete, una por cada condicional.
- –0.25 puntos por cada valor de calidad mal calculado que provoque que falle el `assertEquals(int,int)` en esa contraseña (por esto podría tener puntuación total negativa: dejar como mínimo con 0 puntos).

Esto otorga una puntuación de 0..7, normalizarlo a una escala de 0..2; (nota = puntos * 2.0 / 7.0)

Posible solución:

| **Contraseña** | **calidad** |
|---|---|
| `aaaabbbbc` | 2 |
| `aaaabbb` | -1 |
| `aaaabbbbcccc` | 3 |
| `AA` | -1 |
| `aa` | -1 |
| `00` | -1 |
| `=$` | 0 |