Introducción al Software, Curso 2017-2018

Problema 1: Órdenes de la *shell*; Datos de una clase; Literales; Sangrado

Datos personales	
Apellidos:	
Nombre:	

1 Datos de una clase

Objetivos

Distinguir entre los diferentes datos que pueden encontrarse en una clase.

Descripción

Indicar qué datos se encuentran en el código Java de la siguiente clase, indicando para cada uno si es un atributo, argumento, variable local o constante literal, así como el tipo de dato.

Observar que los datos variables siempre se definen en Java con el formato:

```
tipo nombreVariable
```

donde en ocasiones se pone el modificador "private" delante. El tipo puede ser un tipo predefinido (como int, double, …) o una clase (como String).

Por otro lado, las constantes literales se expresan directamente con su valor.

```
/**
 * Clase que representa un coche y su movimiento
 */
public class Coche
{
   // aceleración y frenada maxima en metros por segundo cuadrado
   public static final double acelMax=4.0, frenadaMax=10.0;

   // velocidad en m/s, posicion en m, tiempo en s
   private double velocidad, posicion, tiempo;

   // modelo
   private String modelo;

   /**
    * Constructor al que se le pasa el modelo de coche y que deja vel, pos y t a 0
    */
   public Coche(String modeloCoche)
   {
      velocidad= 0;
      posicion=0;
      tiempo=0;
```

```
    modelo="Modelo: "+modeloCoche;
}

/**
 * Cambia de via, colocando la posicion en la nueva vía a cero
 * Retorna el tiempo empleado en la via anterior, en segundos
 */
public double cambiaVia()
{
    double tiempoEmpleado=tiempo;
    tiempo=0;
    posicion=0;
    return tiempoEmpleado;
}

/**
 * Avanza un cierto tiempo, intentando alcanzar la velocidad destino
 * Se indican el tiempo a avanzar, en segundos y la velocidad destino en m/s
 */
public void avanza (double t, double velDestino)
{
    tiempo=tiempo+t;
    double acel;
    double acelDeseada=(velDestino-velocidad)/t;
    acel=acelDeseada;
    velocidad=velocidad+acel*t;
    posicion=posicion+velocidad*t+0.5*acel*t*t;
}

/**
 * Avanza un cierto tiempo, sin sobrepasar el punto destino. Se
 * especifican el tiempo en segundos, la posicion de destino en
 * metros y la velocidad maxima en m/s
 */
public void avanzaHastaPos (double t, double posDestino, double velMax)
{
    double distHastaDestino=posDestino-posicion;
    double velDeseada;
    avanza(t,velDeseada);
}

/**
 * posicion, en metros
 */
public double getPosicion()
{
    return posicion;
}

/**
 * velocidad, en metros/s
 */
public double getVelocidad()
{
    return velocidad;
}
```

```
   /**
    * tiempo, en segundos
    */
   public double getTiempo()
   {
      return tiempo;
   }

   /**
    * Modelo de coche
    */
   public double getModelo()
   {
      return modelo;
   }
}
```

Respuesta

- atributos:
- argumentos:
- variables locales:
- constantes literales:

2 Literales

Objetivos

Familiarizarse con los literales de los tipos primitivos

Descripción

Escribir los literales del número 23 de las siguientes formas:

- como número entero del tipo `int`
- como número entero del tipo `int` en octal
- como número entero del tipo `int` en hexadecimal
- como número entero del tipo `long`
- como número real del tipo `float`
- como número real del tipo `double`
- como un texto

Respuesta

<Poner aquí los literales del número 23 en los formatos indicados>

3 Sangrado

Objetivos

Familiarizarse con el concepto de sangrado.

Descripción

Contestar a una pregunta sobre la importancia del sangrado y hacer un ejercicio para sangrar correctamente las instrucciones de una clase Java.

Respuesta

a) ¿Para qué es importante el sangrado del código fuente?

<poner aquí la respuesta, máximo dos líneas>

b) Adaptar la siguiente clase utilizando el sangrado que te parezca más adecuado:

Nota 1: En Java es posible partir instrucciones largas en dos o más líneas, para que quepan en el papel o en la pantalla. Las líneas posteriores a la primera se sangran un nivel más a la derecha.

Nota 2: Por sencillez, la clase que aparece abajo no está completa.

```
/**
 * Almacena la informacion estadistica de una via urbana
 * de un solo sentido para trafico de automoviles
 */
public abstract class Via
{
// Nombre de la via
private String nombre;

// Numero de coches que han atravesado la via
private int numCoches;

// Tiempo total, maximo y minimo que han necesitado
// los coches para atravesar la via, en segundos
private double tiempoTotal, tiempoMaximo, tiempoMinimo;

/**
 * Constructor al que se le pasa el nombre de la via en metros
 */
public Via(String nombre)
{
this.nombre=nombre;
numCoches=0;
tiempoTotal=0.0;
tiempoMaximo=0.0;
tiempoMinimo=Double.MAX_VALUE;
}

/**
* Muestra en pantalla la cabecera de la estadistica de una via
*/
public static void muestraCabecera()
{
System.out.println();
```

```
System.out.println
("   Nombre  t.med  t.max  t.min  v.med  num. coches");
}

}
```

4 Órdenes Unix

Objetivos

Aprender el funcionamiento de algunas órdenes del intérprete de órdenes `bash` de Unix

Descripción

Consultar el funcionamiento de las siguientes órdenes de la *shell* `bash` de Unix, resumir el funcionamiento en menos de tres líneas por orden y escribir un breve ejemplo que las utilice haciendo lo que se pide:

- `if` (instrucción condicional): el ejemplo debe mostrar un mensaje en pantalla (con la orden `echo`) si un fichero determinado existe y otro mensaje distinto si no existe
- `for` (bucle para repetir instrucciones): el ejemplo debe mostrar en pantalla, uno por línea, los nombres que aparecen en una variable definida como nombres='Pedro Andres Laura Ana Felipe'
- `grep` (búsqueda de texto en varios ficheros): el ejemplo debe mostrar los nombres de los ficheros de un directorio que contienen la palabra "`pedro`"

Nota: Para hacer pruebas de los ejemplos se puede crear un directorio y en él crear los ficheros que se necesiten para probar.

Respuesta:

<poner aquí el resumen de lo que hace cada orden seguido del ejemplo de uso que se pide>

if

for

grep