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

Datos personales	
Apellidos:	Prieels
Nombre:	Cédric

Órdenes Unix

Objetivos

Aprender el funcionamiento de algunas órdenes de la *shell* de Unix

Descripción

Indicar brevemente (máximo de tres líneas para cada una) para qué sirven las siguientes instrucciones de la *shell* de Unix. Poner un ejemplo de cada una indicando lo que hace (máximo otras 3 líneas para el ejemplo y su descripción):

- chgrp
- grep
- cat
- if

Respuesta:

1. chgrp es el comando "change group" que permite cambiar los usuarios que poseen un fichero o un directorio. Permite cambiar los usuarios que pueden modificar y escribir dentro de un fichero, por ejemplo.

   chgrp [option] grupo fichero (eg : chgrp –R nuevogrupo midirectorio)
   En este caso la opción –R permite cambiar el groupo de un directorio, "nuevogrupo" est el nuevogrupo que tendrá acceso a "midirectorio".

2. grep es un comando que permite saber si una palabra se encuentra o no dentro de un fichero dado.

   En el IFCA para saber si el trabajo en Gridui se ha acabado bien, se puede escribir (siendo job* los ficheros de salida, Done la palabra que no dice que todo ha salido bien) : cat job* | grep Done | wc

3. cat es el comando que permite poner en la pantalla el contenido de un fichero txt, por ejemplo, sin abrir un editor de texto del tipo gedit o emacs.

   En el ejemplo precedente, cat permite abrir los ficheros job*.txt para ver el contenido, y después se usa Grep para ver si la palabra Done se encuentra o no.

4. El comando if permite ejecutar una serie de instructiones después si se verifica una condición, que se escribe después de este "if".

   Por ejemplo, se puede escribir algo asi : if cmd (…) then <instructiones> (…) fi, que permite ejecutar las instrucciones si el comando cmd se ha pasado bien.

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 gestiona los jugadores de los distintos equipos de baloncesto
 */
public class Jugador
{
   private String nombre; //nombre con el que se conoce al jugador
   private int edad; //la edad en años del jugador
   private double altura; //la altura del jugador, en metros
   private int posicion; // posicion que ocupa el jugador de entre las posibles posiciones

   //Constantes de calse publicas que limitan las posibles posiciones que ocupa un jugador.
   public static final int BASE = 0;
   public static final int ESCOLTA = 1;
   public static final int ALERO = 2;
   public static final int PIVOT = 3;


   /**
    * Constructor de la clase Jugador
    * @param nombre    nombre con el que se conoce al jugador
    * @param edad      la edad en años del jugador
    * @param altura    la altura del jugador, en metros
    * @param posicion  0=BASE, 1=ESCOLTA, 2=ALERO, 3=PIVOT
    */
   public Jugador(String nombre, int edad, double altura, int posicion)
   {
      this.nombre = nombre;
      this.edad = edad;
      this.altura = altura;
      this.posicion = posicion;
   }

   /**
    * Metodo observador que devuelve el nombre del jugador
    * @return     nombre con el que se conoce al jugador
    */
   public String getNombre()
```

```
    {
        return nombre;
    }

     /**
     * Metodo observador que devuelve la edad del jugador
     * @return     la edad del jugador en años
     */
    public int getEdad()
    {
        return edad;
    }

     /**
     * Metodo observador que devuelve la altura del jugador
     * @return     altura del jugador en metros
     */
    public double getAltura()
    {
        return altura;
    }

     /**
     * Metodo observador que devuelve la posicion en la que juega el jugador
     * @return     posicion en la que juega el jugador
     */
    public int getPosicion()
    {
        return posicion;
    }

    public String toString()
    {
        String cadena = nombre + ". " + edad + "años. " + altura + "m. ";
        switch(posicion)
        {
            case BASE:
                cadena+="Base.";
                break;
            case ESCOLTA:
                cadena+="Escolta.";
                break;
            case ALERO:
                cadena+="Alero.";
                break;
            case PIVOT:
                cadena+="Pivot.";
                break;
            default:
                break;

        }
        return cadena;
    }

}
```

Respuesta

Crear una lista de los datos que hay en la clase, agrupándolos según:

- atributos
- argumentos
- variables locales
- constantes literales

Para cada dato, indicar de qué tipo es (entero, real, carácter, booleano, String, ...)

ATRIBUTOS :

(private) String nombre
(private) int edad
(private) double altura
(private) int posicion

ARGUMENTOS :

String nombre
int edad
double altura
int posicion // Estos atributos también son argumentos del constructor

VARIABLES LOCALES :

String nombre
int edad
double altura
int posicion
String cadena

CONSTANTES LITERALES :

(public static final) int BASE
(public static final) int ESCOLTA
(public static final) int ALERO
(public static final) int PIVOT

Literales

Objetivos

Familiarizarse con los literales de los tipos primitivos

Descripción

Reescribir la siguiente clase Java para que los literales utilizados se correspondan con el tipo de la variable a la que se asigna o de la expresión en que aparece (aunque no sea necesario para que la clase compile correctamente).

Respuesta

Adaptar los literales de la siguiente clase para lograr una coincidencia de los tipos de sus expresiones:

```
/**
* Clase que sirve para practicar con los literales de los números enteros y reales
*/
public class Literales
{
   // atributos
   double x=3.0;
   float medida=-8.0;
   int i=4;
   byte b=6;
   long numero=73l;

   /**
    * Metodo que hace calculos (inutiles) y retorna un numero real
    */
   public double hazCalculos(double y, short s) {
      int j=i+s*3+b;
      numero=numero+8l;
      return x*j+y+numero*medida;
// Esta línea va a añadir dos tipos "double" y un tipo "long".  No lo cambio porque compilara bien asi, y si entiendo
                                                          // bien el ejercicio, no hay que modificar esto. Si no, hay que
                                                          // modificar el tipo de numero para que sea un "double".

   }

}
```

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?

Para estructurar el programa y ver más fácilmente que parte del código pertenece a qué función o bucle. El programa funciona lo mismo con o sin sangrados pero es más fácil para el usuario, permite ver más fácilmente cada bloque del código.

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

```
/**
 * Contiene los datos de un ciclista y métodos para estimar
```

```
 * su potencia y cambiar las condiciones
 *
 * @author (Michael)
 * @version (23-oct-2013)
 */
public class Ciclista
{
	// atributos, en unidades del sistema internacional; ángulos en grados
	private double m; // masa del conjunto ciclista+bicicleta, Kg
	private double ang; // angulo de la pendiente, grados
	private double vc; // velocidad del ciclista, m/s
	private double vViento; // velocidad del viento, m/s
	private double k=0.226; // coeficiente aerodinámico, Kg/m
	private double cr=0.038; // coeficiente de rodadura, m/s2
	private double g=9.8; // gravedad, m/s2

	/**
	* Pone los valores de los atributos obteniéndolos de los argumentos del
	* mismo nombre
	*/
	public Ciclista (double m, double ang, double vc, double vViento)
	{
		this.m=m;
		this.ang=ang;
		this.vc=vc;
		this.vViento=vViento;
	}

	/**
	* retorna la potencia total del ciclista
	*/
	public double potenciaTotal()
	{
		double vAire=vc+vViento;
		double pAero= k*vc*vAire*vAire;
		double pRod=cr*m*vc;
		double pAsc=m*g*vc*Math.sin(Math.toRadians(ang));
		return pAero+pRod+pAsc;
	}


	/**
	* modifica los atributos ang, vc y vViento haciéndolos iguales a nuevoAng,
	* nuevaVelCicl y nuevaVelViento
	*/
	public void cambiaCondiciones (double nuevoAng, double nuevaVelCicl,
	double nuevaVelViento)
	{
		this.ang=nuevoAng;
		this.vc=nuevaVelCicl;
```

```
            this.vViento=nuevaVelViento;
        }

}
```