Programación II - 2010/2011 - Universitat Jaume I
Examen final - Primera convocatoria

24 de mayo de 2011

La duración máxima de este examen es de 3,5 horas. No puedes consultar libros ni apuntes.

Ejercicio 1 — 1 punto

Considera la siguiente implementación de la clase `Par`:

```
public class Par {
   public int primero;
   public int segundo;

   public Par(int elPrimero, int elSegundo) {
      primero = elPrimero;
      segundo = elSegundo;
   }
   public Par(Par otro) {
      primero = otro.primero;
      segundo = otro.segundo;
   }
   public String toString() {
      return "(" + primero + "," + segundo + ")";
   }
}
```

Indica qué se escribe en la salida estándar al ejecutar el siguiente programa que hace uso de la clase `Par`:

```
import java.util.Arrays;

public class PruebaPar {
   public static void intercambia(Par p) {
      int aux = p.primero;
      p.primero = p.segundo;
      p.segundo = aux;
   }
   public static void intercambia(Par a, Par b) {
      Par aux = a;
      a = b;
      b = aux;
   }
   public static void intercambia(Par[] v) {
      Par aux = v[0];
      v[0] = v[1];
      v[1] = aux;
   }
   public static void main(String[] args) {
      Par miPar1 = new Par(2,3), miPar2 = miPar1, miPar3 = new Par(miPar2);

      intercambia(miPar1);
      intercambia(miPar2);
      intercambia(miPar3);
      System.out.println("E1: " + miPar1 + " " + miPar2 + " " + miPar3);

      Par[] misPares = { new Par(4,5), new Par(6,7) };

      intercambia(misPares[0]);
      System.out.println("E2: " + Arrays.toString(misPares));

      intercambia(misPares[0], misPares[1]);
      System.out.println("E3: " + Arrays.toString(misPares));

      intercambia(misPares);
      System.out.println("E4: " + Arrays.toString(misPares));
    }
}
```

Ejercicio 2 — 2 puntos

Para cada uno de los dos apartados siguientes:

- Indica qué se escribe en la salida estándar al ejecutarlo.
- Describe brevemente para qué sirve el método `enigma`.
- Expresa, empleando la notación O, el tiempo de ejecución del método `enigma`.

a) **[1 punto]**

```
public class EnigmaUno {

   public static int enigma(int n) {
      if (n < 0)
         return enigma(-n);
      else
         if (n <= 1)
            return 1;
         else
            return enigma(n/2) + 1;
   }

   public static void main(String[] args) {
      int[] vector = { 1, -4, 20, 60 };
      for (int i = 0; i < vector.length; i++) {
         System.out.println(enigma(vector[i]));
      }
   }

}
```

b) **[1 punto]**

```
public class EnigmaDos {

   public static int enigma(String[] vector, String valor) {
      return enigma(vector, valor, 0, vector.length - 1);
   }

   private static int enigma(String[] vector, String valor, int a, int b) {
      int aux = 0;
      if (a > b)
         return 0;
      if (vector[a].equals(valor))
         aux++;
      if (a != b && vector[b].equals(valor))
         aux++;
      return aux + enigma(vector, valor, a + 1, b - 1);
   }

   public static void main(String[] args) {
      String[] letras = { "a", "g", "c", "g", "b", "c" };
      System.out.println(enigma(letras, "b"));
      System.out.println(enigma(letras, "e"));

      String[] vocales = { "o", "e", "o", "e", "o" };
      System.out.println(enigma(vocales, "o"));
   }

}
```

Queremos desarrollar una aplicación que nos permita almacenar y consultar información sobre recetas de cocina. Cada receta tiene un nombre único que la identifica, un tiempo de elaboración y varios ingredientes. De cada ingrediente sólo nos interesa su nombre y la cantidad necesaria para preparar la receta. Considera que ya disponemos de la siguiente clase `Ingrediente`:

```
public class Ingrediente {
   private String nombre;
   private double cantidad;

   public Ingrediente(String unNombre, double unaCantidad) {
      nombre = unNombre;
      cantidad = unaCantidad;
   }
   public String getNombre()   { return nombre; }
   public double getCantidad() { return cantidad; }
}
```

Escribe en lenguaje Java la clase `Receta`, haciendo uso de la clase `Ingrediente`. Internamente, los ingredientes se deben guardar ordenados lexicográficamente en un vector[1]. Además de definir los atributos necesarios para representar una receta, te pedimos implementar los siguientes métodos, que se deben poder utilizar como ilustran los ejemplos:

a) **[0,25 puntos]** Un constructor que reciba como parámetros el nombre de la receta y el tiempo de elaboración expresado en minutos. Inicialmente, la receta no contendrá ningún ingrediente. A continuación se muestra un ejemplo de uso:

```
Receta receta = new Receta("Tallarines con guisantes", 30); // 30 minutos
```

La valoración de este apartado incluye la definición de los atributos de la clase `Receta`.

b) **[1,75 puntos]** Un método `insertar` que permita añadir a la receta un nuevo ingrediente dado su nombre y la cantidad necesaria del mismo. A continuación se muestra un ejemplo de uso:

```
receta.insertar("tallarines", 500); // 500 gr. de tallarines
receta.insertar("guisantes", 300);  // 300 gr. de guisantes
receta.insertar("jamón york", 150); // 150 gr. de jamón york
```

Observa que el usuario no tiene por qué proporcionar los ingredientes ordenados, pero internamente se han de guardar ordenados. Considera que nunca se insertarán ingredientes repetidos en una misma receta.

Expresa, empleando la notación O, el tiempo de ejecución de tu solución.

c) **[2 puntos]** Un método `contieneAlguno` que reciba como parámetro un vector con los nombres de varios ingredientes. El método debe devolver `true` si la receta contiene alguno de los ingredientes cuyos nombres aparecen en ese vector y `false` si no contiene ninguno de ellos. Si en la receta todavía no se ha insertado ningún ingrediente o si el vector dado como parámetro es vacío, el método debe devolver `false`. A continuación se muestra un ejemplo de uso:

```
String[] alimentosInsanos = { "palomitas", "salchichas", "mayonesa" };
if (receta.contieneAlguno(alimentosInsanos)) {
    ...
}
```

En este apartado sólo optarás al 50 % de la nota si tu solución no tiene el mejor tiempo de ejecución posible, teniendo en cuenta que el vector de ingredientes que forman la receta está ordenado pero el vector que se le pasa al método no tiene por qué estar ordenado.

Expresa, empleando la notación O, el tiempo de ejecución de tu solución.

[1] Te recordamos que, si `c1` y `c2` son de tipo `String`, `c1.compareTo(c2)` devuelve un número negativo, cero o un número positivo según `c1` sea menor que, igual o mayor que `c2`, respectivamente, de acuerdo con el orden lexicográfico de las cadenas.

Queremos desarrollar una aplicación para tiendas de alquiler de videojuegos y necesitamos implementar una clase que nos permita gestionar la información de un catálogo de videojuegos. De cada juego nos interesan tres datos: un código que lo identifica, su nombre y la cantidad de ejemplares disponibles. Considera que ya disponemos de la siguiente clase `Juego`:

```
public class Juego {
   private int código;
   private String nombre;
   private int cantidad;

   public Juego(int unCódigo, String unNombre, int unaCantidad) {
      código = unCódigo;
      nombre = unNombre;
      cantidad = unaCantidad;
   }
   public int    getCódigo()   { return código; }
   public String getNombre()   { return nombre; }
   public int    getCantidad() { return cantidad; }
   public void   setCantidad(int unaCantidad) { cantidad = unaCantidad; }
}
```

Escribe en lenguaje Java la clase `CatálogoDeJuegos`, haciendo uso de la clase `Juego`. Internamente, los juegos se deben almacenar en una lista enlazada de nodos en la que cada nodo contiene un juego. Puedes utilizar el tipo de lista que quieras (simplemente enlazada, doblemente enlazada, con referencia al primer nodo, con referencia al primero y al último, etc.) pero no puedes utilizar ninguna de las proporcionadas en las bibliotecas del lenguaje Java. Además de los atributos y métodos que necesites para representar cada nodo y de los atributos que necesites para representar el catálogo, te pedimos implementar los siguientes métodos, que se deben poder utilizar como ilustran los ejemplos:

a) **[0,5 puntos]** Un constructor sin parámetros. Inicialmente, el catálogo no contendrá ningún juego. A continuación se muestra un ejemplo de uso:

```
CatálogoDeJuegos catálogo = new CatálogoDeJuegos();
```

La valoración de este apartado incluye la definición de la clase que representa los nodos y de los atributos de la clase `CatálogoDeJuegos`.

b) **[0,5 puntos]** Un método `darDeAlta` que permita añadir al catálogo un juego dado como parámetro. A continuación se muestra un ejemplo de uso:

```
Juego últimaAdquisición = new Juego(14525, "Tetris 10D", 2);
catálogo.darDeAlta(últimaAdquisición);
```

Si ya hay un juego con el mismo código, se debe lanzar una excepción de tipo `JuegoDuplicado`[2].

c) **[1,25 puntos]** Un método `darDeBaja` que permita eliminar del catálogo un juego dando su código. A continuación se muestra un ejemplo de uso:

```
catálogo.darDeBaja(14525);
```

Si el juego no se encuentra en el catálogo, no debe cambiar nada.

d) **[0,75 puntos]** Un método `prestar` que permita prestar un juego dando su código. A continuación se muestra un ejemplo de uso:

```
if ( catálogo.prestar(14525) ) ...
```

Si el juego no se encuentra en el catálogo, se debe lanzar una excepción de tipo `JuegoNoEncontrado`[2].

Si se encuentra, hay que comprobar si la cantidad de ejemplares disponibles de ese juego es positiva: si lo es, se debe decrementar y el método debe terminar devolviendo `true` para indicar que el préstamo se ha realizado; si es cero, no debe cambiar nada y el método debe terminar devolviendo `false` para indicar que el préstamo no se ha realizado.

[2] Considera que las clases `JuegoDuplicado` y `JuegoNoEncontrado` ya están definidas.