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

7 de julio de 2011

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

Ejercicio 1 — 1 punto

Para cada uno de los dos apartados siguientes, indica qué se escribe en la salida estándar al ejecutarlo.

a) **[0,25 puntos]**

```
public class Prueba {
   public static void main(String[] args) {
      int a = 1, b = 0, c = 1, d = 1;
      if (a++ == ++b || c++ == ++d && a++ == c++)
         a++;
      System.out.println(a + ", " + b + ", " + c + ", " + d);
   }
}
```

b) **[0,75 puntos]**

```
import java.util.Arrays;

class Contador {
   private int valor;
   public Contador(int valorInicial) { valor = valorInicial; }
   public Contador incrementar()     { return new Contador(valor + 1); }
   public String toString()          { return "" + valor; }
}
public class PruebaContador {
   public static void pruebaUno(Contador contador) {
      contador = contador.incrementar();
   }
   public static void pruebaDos(Contador contador) {
      contador.incrementar();
   }
   public static void pruebaTres(Contador[] contadores) {
      for (int i = 0; i < contadores.length; i++)
         contadores[i].incrementar();
   }
   public static void pruebaCuatro(Contador[] contadores) {
      for (int i = 0; i < contadores.length; i++)
         contadores[i] = contadores[i].incrementar();
   }
   public static void pruebaCinco(Contador[] contadores) {
      for (int i = 0; i < contadores.length; i++)
         pruebaUno(contadores[i]);
   }
   public static void main(String[] args) {
      Contador c = new Contador(0);
      pruebaUno(c);
      System.out.println("1: " + c);
      pruebaDos(c);
      System.out.println("2: " + c);

      Contador[] v = {new Contador(1), new Contador(-1)};
      pruebaTres(v);
      System.out.println("3: " + Arrays.toString(v));
      pruebaCuatro(v);
      System.out.println("4: " + Arrays.toString(v));
      pruebaCinco(v);
      System.out.println("5: " + Arrays.toString(v));
    }
}
```

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` y el tiempo de ejecución total del programa.

a) **[1 punto]**

```
public class EnigmaUno {
   public static int enigma(int n) {
      if (n <= 1)
         return n;
      else
         return enigma(n / 2) + n % 2;
   }
   public static int[][] rellena(int n) {
      int[][] matriz = new int[n][n];
      for (int i = 0; i < n; i++)
         for (int j = i; j < n; j++)
            matriz[i][j] = enigma(i + j);
      return matriz;
   }
   public static void main(String[] args) {
      int[][] matriz = rellena(5);
      for (int i = 0; i < 5; i++)
         System.out.println(i + " " + matriz[i][i]);
   }
}
```

b) **[1 punto]**

```
public class EnigmaDos {
   // Suponemos que la matriz es cuadrada
   private static boolean enigma(int[][] matriz, int i) {
      if ( i >= matriz.length )
         return true;
      for (int j = i + 1; j < matriz.length; j++)
         if ( matriz[i][j] != matriz[j][i] ) {
            System.out.println("(" + i + "," + j + ")");
            return false;
         }
      return enigma(matriz, i + 1);
   }
   public static boolean enigma(int[][] matriz) {
      return enigma(matriz, 0);
   }
   public static void main(String[] args) {
      int[][] m = new int[5][5];
      for (int i = 0; i < 5; i++)
         for (int j = 0; j < 5; j++)
            m[i][j] = i + j;
      System.out.println(enigma(m));
      m[3][3] = 0;
      System.out.println(enigma(m));
      m[3][2] = 0;
      System.out.println(enigma(m));
      m[3][1] = 0;
      System.out.println(enigma(m));
   }
}
```

Queremos desarrollar una aplicación que nos permita almacenar y consultar información sobre asignaturas. De cada asignatura nos interesa su nombre y la información de los estudiantes de la misma. Considera que ya disponemos de la siguiente clase `Estudiante`:

```
public class Estudiante {
   private String nombre;
   private double nota;

   public Estudiante(String unNombre) {
      nombre = unNombre;
   }
   public String getNombre() {
      return nombre;
   }
   public double getNota() {
      return nota;
   }
   public void setNota(double unaNota) {
      nota = unaNota;
   }
}
```

Escribe en lenguaje Java la clase `Asignatura`, haciendo uso de la clase `Estudiante`. Internamente, la información de los estudiantes se debe guardar en un vector de objetos de la clase `Estudiante`. Además de definir los atributos necesarios para representar una asignatura, te pedimos implementar los métodos que se enumeran a continuación, que se deben poder utilizar como ilustran los ejemplos. En este ejercicio no es necesario que el vector esté ordenado ni que tu solución sea lo más eficiente posible, pero debes expresar, empleando la notación O, el tiempo de ejecución de tu solución en cada apartado.

a) **[0,25 puntos]** Un constructor que reciba como parámetro el nombre de la asignatura. Inicialmente, la asignatura no contendrá ningún estudiante. A continuación se muestra un ejemplo de uso:

```
Asignatura redes1 = new Asignatura("Redes I: ataque");
Asignatura redes2 = new Asignatura("Redes II: defensa");
```

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

b) **[0,75 puntos]** Un método `insertar` que permita añadir a la asignatura un nuevo estudiante dado su nombre. A continuación se muestra un ejemplo de uso:

```
redes1.insertar("Edward Drummond 'Barbanegra'");
```

Considera que nunca se insertarán estudiantes con el mismo nombre en una misma asignatura.

c) **[0,5 puntos]** Un método `cambiarNota` que reciba como parámetros el nombre de un estudiante y su nota. El método debe poner esa nota a ese estudiante. Si no hay ningún estudiante con ese nombre en la asignatura, no debe cambiar nada. A continuación se muestra un ejemplo de uso:

```
redes2.cambiarNota("capitán Garfio", 9.5);
```

d) **[1,25 puntos]** Un método `eliminarAprobados` que elimine del vector de estudiantes todos los que tengan una nota igual o superior a 5. A continuación se muestra un ejemplo de uso:

```
redes1.eliminarAprobados();
```

e) **[1,25 puntos]** Un método `mostrarEstudiantesComunes` que reciba como parámetro otra asignatura y escriba en la salida estándar los nombres de todos los estudiantes que están en ambas asignaturas. A continuación se muestra un ejemplo de uso:

```
redes1.mostrarEstudiantesComunes(redes2);
```

Queremos desarrollar una aplicación que nos permita trabajar con recetarios de cocina. Cada recetario tiene un nombre que lo identifica y contiene una colección de recetas. Para ello ya disponemos de la clase `Receta` que, entre otros métodos, incluye los siguientes:

- Un método con perfil `public boolean contiene(String ingrediente)`, que recibe como parámetro el nombre de un ingrediente y devuelve como resultado `true` si la receta contiene ese ingrediente y `false` si no.
- Un método con perfil `public String getNombre()` que devuelve el nombre de la receta.

Escribe en lenguaje Java la clase `Recetario`, haciendo uso de la clase `Receta`. En la clase `Recetario` no puedes utilizar ningún otro método de la clase `Receta` aparte de los dos métodos anteriores. Internamente, las recetas se deben almacenar ordenadas lexicográficamente por su nombre en una lista enlazada de nodos, en la que cada nodo contiene una receta[1]. 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 recetario, te pedimos implementar los siguientes métodos, que se deben poder utilizar como ilustran los ejemplos:

a) **[0,5 puntos]** Un constructor que reciba como parámetro el nombre del recetario. Inicialmente, el recetario no contendrá ninguna receta. A continuación se muestra un ejemplo de uso:

```
Recetario recetario = new Recetario("Recetas para principiantes");
```

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

b) **[1,25 puntos]** Un método `insertar` que reciba como parámetro una nueva receta y la añada al recetario conservando el orden lexicográfico. A continuación se muestra un ejemplo de uso:

```
Receta receta;
// El código necesario para crear la receta no nos interesa
recetario.insertar(receta);
```

Si ya hay una receta con el mismo nombre, se debe lanzar una excepción de tipo `RecetaDuplicada`. Considera que la clase `RecetaDuplicada` ya está definida.

c) **[0,5 puntos]** Un método `mostrarRecetasCon` que reciba como parámetro un vector de cadenas con nombres de ingredientes y que escriba en la salida estándar los nombres de las recetas que contengan alguno de los ingredientes dados. A continuación se muestra un ejemplo de uso:

```
String[] antojos = { "maíz cocido", "anchoas" };
recetario.mostrarRecetasCon(antojos);
```

d) **[0,75 puntos]** Un método `eliminar` que reciba como parámetro el nombre de una receta y elimine dicha receta del recetario. Si el recetario no contiene ninguna receta con el nombre dado, no se debe modificar. A continuación se muestra un ejemplo de uso:

```
recetario.eliminar("Tallarines con guisantes");
```

[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.