Programación II - 2011/2012 - Universitat Jaume I
Examen final - Primera convocatoria

31 de mayo de 2012

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

Ejercicio 1 — 0,8 puntos

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 incrementa(int a, int b) {
      a++;
      b++;
   }
   public static void incrementa(Par par) {
      par.primero++;
      ++par.segundo;
   }
   public static void incrementa(Par[] v) {
      for (int i = 0; i < v.length; i++)
         incrementa(v[i]);
   }
   public static void main(String[] args) {
      Par miPar1 = new Par(2, 6), miPar2 = new Par(miPar1), miPar3 = miPar2;
      Par[] misPares = {miPar1, miPar2, miPar3};

      incrementa(miPar1.primero, miPar1.segundo);
      System.out.println("E1: " + miPar1 + " " + miPar2 + " " + miPar3);
      System.out.println("E2: " + Arrays.toString(misPares));

      incrementa(miPar2);
      System.out.println("E3: " + miPar1 + " " + miPar2 + " " + miPar3);
      System.out.println("E4: " + Arrays.toString(misPares));

      incrementa(misPares[2]);
      System.out.println("E5: " + miPar1 + " " + miPar2 + " " + miPar3);
      System.out.println("E6: " + Arrays.toString(misPares));

      incrementa(misPares);
      System.out.println("E7: " + miPar1 + " " + miPar2 + " " + miPar3);
      System.out.println("E8: " + Arrays.toString(misPares));
    }
}
```

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) **[0,6 puntos]**

```
public class EnigmaUno {

   public static void main(String[] args) {
      System.out.println("E1: " + enigma(2, 8));
      System.out.println("E2: " + enigma(2, 9));
      System.out.println("E3: " + enigma(3, 4));
      System.out.println("E4: " + enigma(2, -1));
   }

   public static double enigma(double x, int n) {
      if (n < 0)
         return 1.0 / enigma(x, -n);
      else if (n == 0)
         return 1.0;
      else if (n % 2 == 0) {
         double parcial = enigma(x, n / 2);
         return parcial * parcial;
      } else {
         double parcial = enigma (x, (n - 1) / 2);
         return parcial * parcial * x;
      }
   }

}
```

b) **[0,6 puntos]**

```
public class EnigmaDos {

   public static void main(String[] args) {
      int[] v1 = {5, 1, 2, 8, 3, 9, 7};
      System.out.println("E5: " + enigma(v1));
      int[] v2 = {1, 5, 2, 8, 3, 9, 7};
      System.out.println("E6: " + enigma(v2));
      int[] v3 = {9, 7, 2, 8, 3, 5, 1};
      System.out.println("E7: " + enigma(v3));
   }

   public static int enigma(int[] v) {
      return misterio(v, v.length - 1);
   }

   public static int misterio(int[] v, int i) {
      if (i <= 0)
         return 0;
      else if (v[i] > v[0])
         return misterio(v, i - 1) + 1;
      else
         return misterio(v, i - 1);
    }

}
```

Ejercicio 3 — 4 puntos

Queremos desarrollar una aplicación que nos permita gestionar información de una serie de cursos de formación. Cada curso tiene un nombre y la relación de estudiantes que participan en el mismo. De cada estudiante solo almacenaremos su DNI y su nota. Considera que ya disponemos de la siguiente clase `Estudiante`:

```
public class Estudiante {
   private String dni;
   private double nota;
   public Estudiante(String unDni, double unaNota) {
      dni  = unDni;
      nota = unaNota;
   }
   public String getDni()  { return dni;  }
   public double getNota() { return nota; }
   public void setNota(double unaNota) { nota = unaNota; }
}
```

Escribe en lenguaje Java la clase `Curso`, haciendo uso de la clase `Estudiante` (sin modificarla). Internamente, los estudiantes se deben guardar ordenados lexicográficamente por DNI en un vector de objetos de la clase `Estudiante`. Además de definir los atributos necesarios para representar un curso, te pedimos implementar los siguientes métodos públicos, que se deben poder utilizar como ilustran los ejemplos:

a) **[1 punto]** Un constructor que reciba como parámetros el nombre de un curso y un vector ordenado de cadenas con los DNIs de los estudiantes que participan en ese curso. Puedes suponer que el vector de cadenas dado no está vacío. El constructor debe verificar que el vector está ordenado de menor a mayor y que no contiene elementos repetidos[1]. De no ser así, se debe lanzar una excepción de tipo `ExcepciónDatosNoVálidos`[2]. Si los datos son válidos, se debe almacenar un objeto de la clase `Estudiante` por cada DNI dado. Inicialmente, todos los estudiantes deben tener -1 como nota para indicar que todavía no han sido evaluados.

   Ejemplo:
   ```
   String[] vectorDNI = {"12345678X", "21098765Y", "56789012B", "5679321A"};
   Curso cursoSeguridad = new Curso("Seguridad informática", vectorDNI);
   ```

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

b) **[1 punto]** Un método `cambiarNota` que reciba como parámetros un DNI y una nota y cambie la nota del estudiante correspondiente por la nota dada. Si el DNI dado no figura entre los participantes, el curso no se debe modificar y se debe lanzar una excepción del tipo `ExcepciónDniNoEncontrado`[2].

   Ejemplo: `cursoSeguridad.cambiarNota("21098765Y", 7.5);`

   Solo se aceptan soluciones eficientes, que tengan en cuenta que el vector de estudiantes está ordenado por DNI. Expresa, empleando la notación O, el tiempo de ejecución de tu solución.

c) **[1 punto]** Un método `contarEstudiantes` que reciba como parámetro otro curso y devuelva como resultado la cantidad de estudiantes diferentes que están participando en cualquiera de los dos cursos, es decir, si un estudiante (identificado por su DNI) participa en los dos cursos debe contarse solo una vez.

   Ejemplo: `int totalEstudiantes = cursoSeguridad.contarEstudiantes(otroCurso);`

   Solo se aceptan soluciones cuyo tiempo de ejecución sea $O(N)$, siendo N la suma de las cantidades de estudiantes de ambos cursos.

d) **[1 punto]** Un método `crearCursoDeRefuerzo` que reciba como parámetros un nuevo nombre y una nota de corte y devuelva como resultado un nuevo curso, con ese nuevo nombre, en el que participen aquellos estudiantes cuya nota es inferior a la nota de corte. Si no hay ningún estudiante que cumpla eso, el método debe devolver `null`. Inicialmente en el nuevo curso los estudiantes tendrán -1 como nota.

   Ejemplo: `Curso refuerzoSI = cursoSeguridad.crearCursoDeRefuerzo("Refuerzo S.I.", 4);`

   Solo se aceptan soluciones cuyo tiempo de ejecución sea $O(N)$, siendo N la cantidad de estudiantes.

[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.

[2] Considera que las clases `ExcepciónDatosNoVálidos` y `ExcepciónDniNoEncontrado` ya están definidas.

Queremos desarrollar una aplicación para gestionar rutas que visitan una serie de localidades. Para cada localidad dispondremos de varios datos pero, en este ejercicio, solo nos interesa su nombre. Además, la posición de cada localidad en la ruta es relevante. Considera que ya disponemos de la siguiente clase `Localidad`:

```
public class Localidad {
   private String nombre;
   public Localidad(String unNombre) { nombre = unNombre; }
   public Localidad(Localidad otraLocalidad) { nombre = otraLocalidad.nombre; }
   public String getNombre() { return nombre; }
}
```

Escribe en lenguaje Java la clase `Ruta`, haciendo uso de la clase `Localidad` (sin modificarla). Internamente, las localidades se deben almacenar en una lista enlazada de nodos en la que cada nodo contenga una localidad, de modo que la posición del nodo en la lista coincida con la posición de la localidad en la ruta. 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 la ruta, te pedimos implementar los siguientes métodos públicos, que se deben poder utilizar como ilustran los ejemplos:

a) **[0,5 puntos]** Un constructor sin parámetros. Inicialmente, la ruta no contendrá ninguna localidad.

   Ejemplo: `Ruta altoMaestrazgo = new Ruta();`

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

b) **[1 punto]** Un método `insertarLocalidad` que permita añadir a la ruta una localidad en la posición indicada, teniendo en cuenta que numeramos las posiciones desde cero consecutivamente. Si ya hay una localidad en esa posición, tanto ella como las que le siguen ven incrementada su posición. Si la posición dada es menor o igual que cero, la nueva localidad se debe añadir al principio de la ruta. Si la posición dada es mayor o igual que la talla de la ruta, la nueva localidad se debe añadir al final de la ruta.

   Ejemplo:
   ```
   Localidad origen = new Localidad("Puertomingalvo");
   altoMaestrazgo.insertarLocalidad(origen, 0);
   ```

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

c) **[0,5 puntos]** Un método `consultarLocalidadPorPosición` que reciba como parámetro una posición y devuelva como resultado la localidad que ocupa esa posición en la ruta. Si la posición dada no es válida, el método debe devolver `null`.

   Ejemplo: `Localidad destino = altoMaestrazgo.consultarLocalidadPorPosición(5);`

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

d) **[1 punto]** Un método `borrarRango` que reciba como parámetros dos posiciones dentro de la ruta y elimine de la misma todas las localidades comprendidas entre las posiciones dadas (ambas incluidas). Por simplicidad, puedes suponer que las dos posiciones dadas son válidas y que la primera es menor o igual que la segunda.

   Ejemplo: `altoMaestrazgo.borrarRango(2, 5);`

   Solo se aceptan soluciones cuyo tiempo de ejecución sea $O(N)$, siendo N la talla de la ruta.

e) **[1 punto]** Un método `copiarRango` que reciba como parámetros dos posiciones dentro de la ruta y devuelva como resultado una nueva ruta formada por copias de las localidades comprendidas entre las dos posiciones dadas (ambas incluidas). Como en el apartado anterior, puedes suponer que las dos posiciones dadas son válidas y que la primera es menor o igual que la segunda.

   Ejemplo: `Ruta centroAltoMaestrazgo = altoMaestrazgo.copiarRango(2, 4);`

   Solo se aceptan soluciones cuyo tiempo de ejecución sea $O(N)$, siendo N la talla de la ruta.