Programación II - 2010/2011 - Universitat Jaume I
Evaluación continua - Módulo 2 de teoría - Recuperación

17 de mayo de 2011

La duración máxima de esta prueba es de 45 minutos. No puedes consultar libros ni apuntes.

Ejercicio 1 (4 puntos, 2 puntos por apartado)

Imagina que hubiésemos implementado así las clases `Punto` y `Segmento`. Ten en cuenta que hay importantes diferencias respecto de la implementación que conoces.

```
public class Punto {
   private double x, y;
   public Punto(double laX, double laY) {
      x = laX;
      y = laY;
   }
   public Punto mover(double dx, double dy) {
      return new Punto(x+dx, y+dy);
   }
   public String toString() {
      return "(" + x + "," + y + ")";
   }
}

public class Segmento {
   private Punto inicio, fin;
   public Segmento(Punto elInicio, Punto elFin) {
      inicio = elInicio;
      fin = elFin;
   }
   public void invertir() {
      Punto tmp = inicio;
      inicio = fin;
      fin = tmp;
   }
   public Segmento mover(double dx, double dy) {
      Punto nuevoInicio = inicio.mover(dx, dy);
      Punto nuevoFin = fin.mover(dx, dy);
      return new Segmento(nuevoInicio, nuevoFin);
   }
   public String toString() {
      return inicio + "-" + fin;
   }
}
```

Indica lo que se escribiría en la salida estándar al ejecutar cada uno de los siguientes programas utilizando esas versiones. No es necesario que expliques por qué.

(a)
```
import java.util.Arrays;

public class SegmentosEjercicioA {

   public static void invertirUnSegmento(Segmento unSegmento) {
      unSegmento.invertir();
   }
   public static void moverUnSegmento(Segmento unSegmento, double dx, double dy) {
      unSegmento = unSegmento.mover(dx, dy);
   }
   public static void main(String[] args) {
      Segmento [] misSegmentos = new Segmento[2];
      misSegmentos[0] = new Segmento(new Punto(2,3), new Punto(4,5));
      misSegmentos[1] = new Segmento(new Punto(6,7), new Punto(8,9));

      invertirUnSegmento(misSegmentos[0]);
      System.out.println("a1 = " + Arrays.toString(misSegmentos));

      moverUnSegmento(misSegmentos[1], 10, 20);
      System.out.println("a2 = " + Arrays.toString(misSegmentos));
    }
}
```

(b)
```
import java.util.Arrays;

public class SegmentosEjercicioB {

   public static void invertirDosSegmentos(Segmento [] dosSegmentos) {
      dosSegmentos[0].invertir();
      dosSegmentos[1].invertir();
   }
   public static void moverDosSegmentos(Segmento [] dosSegmentos,
                                        double dx, double dy) {
      dosSegmentos[0] = dosSegmentos[0].mover(dx, dy);
      dosSegmentos[1] = dosSegmentos[1].mover(dx, dy);
   }
   public static void main(String[] args) {
      Segmento [] misSegmentos = new Segmento[2];
      misSegmentos[0] = new Segmento(new Punto(2,3), new Punto(4,5));
      misSegmentos[1] = new Segmento(new Punto(6,7), new Punto(8,9));

      invertirDosSegmentos(misSegmentos);
      System.out.println("b1 = " + Arrays.toString(misSegmentos));

      moverDosSegmentos(misSegmentos, 10, 20);
      System.out.println("b2 = " + Arrays.toString(misSegmentos));
   }
}
```

Ejercicio 2 (6 puntos)

Queremos implementar una nueva clase `TareasPendientes` que nos permita gestionar tareas pendientes. De cada tarea nos interesan dos datos, su fecha y una descripción de la misma, y por ello vamos a utilizar la siguiente clase `Tarea`:

```
public class Tarea {
   private Fecha fecha;
   private String descripción;
   public Tarea(Fecha laFecha, String laDescripción) {
      fecha = laFecha;
      descripción = laDescripción;
   }
   public Fecha getFecha() {
      return fecha;
   }
   public String getDescripción() {
      return descripción;
   }
}
```

Implementa la clase `TareasPendientes` haciendo uso de esa clase `Tarea`, sin modificarla. Supón que dispones también de una implementacion de la clase `Fecha`, por ejemplo, cualquiera de las que has implementado en las clases prácticas. Internamente, las tareas se deben guardar en un vector de tareas ordenadas de menor a mayor fecha. Con ese requisito, debes escribir todo lo necesario para que el usuario de la clase pueda hacer lo siguiente (tal como ilustran los ejemplos, no de otra forma):

1. Crear un nuevo objeto de esa clase:

   ```
   TareasPendientes misTareas = new TareasPendientes();
   ```

2. Añadirle tareas:

   ```
   misTareas.insertar(new Tarea(new Fecha(24, 5, 2011),
                                "examen final PROGRAMACIÓN II"));
   misTareas.insertar(new Tarea(new Fecha(7, 6, 2011),
                                "celebración PROGRAMACIÓN II"));
   misTareas.insertar(new Tarea(new Fecha(25, 5, 2011),
                                "reciclar apuntes PROGRAMACIÓN II"));
   ```

   Observa que el usuario no tiene por qué proporcionar las tareas ordenadas, pero internamente se han de guardar ordenadas. Se acepta que el usuario inserte tareas con la misma fecha y/o con la misma descripción. Cuando haya varias tareas con la misma fecha, el orden entre ellas no importa.

3. Buscar la fecha de una determinada tarea dada su descripción:

   ```
   Fecha fechaImportante = misTareas.buscar("examen final PROGRAMACIÓN II");
   ```

   En caso de que haya más de una tarea con la misma descripción, debe devolver la fecha de la primera de ellas; si no hay ninguna con esa descripción, debe devolver `null`.

Responde brevemente a las siguientes cuestiones:

(a) Indica el tiempo de ejecución de cada uno de los métodos en tu implementación.

(b) Pon un ejemplo de método que, aprovechando que el vector está ordenado, se podría implementar más eficientemente que si no lo estuviese, e indica su coste en cada caso (no es necesario que lo implementes en este ejercicio).