Segundo Parcial de IIP - ETSInf. Recuperación
Fecha: 24 de enero de 2017. Duración: 2h 30'

1. 6 puntos Se dispone de la clase `Parada` que representa una parada de un tren a lo largo de su trayecto. La información de la parada viene dada por el nombre de la ciudad, el momento del día en que se hace la parada y el tipo de la misma. La parada puede ser de tipo estación, apeadero o apartadero (un apartadero no permite la incorporación de viajeros).

   Se muestra, a continuación, un resumen de su documentación, con sus constantes y un extracto de sus métodos públicos:

Field Summary

static int	APARTADERO
static int	APEADERO
static int	ESTACION

Constructor Summary

Parada(java.lang.String c, iipUtil.Instante h, int t) Crea una Parada en la ciudad c a la hora h y de tipo t.

Method Summary

. . .	
java.lang.String	getCiudad() Devuelve la ciudad de la parada this.
iipUtil.Instante	getHoraParada() Devuelve la hora de parada de this.
int	getTipo() Devuelve el tipo de la parada this: ESTACION, APEADERO o APARTADERO.

   Se pide: implementar la clase `RecorridoTren` para representar la secuencia de paradas que realiza un tren entre un origen y un destino. Los atributos y métodos de la clase a implementar son los que se indican a continuación.

   a) (0.5 puntos) Atributos, de los cuales solo es público el primero:
   - `MAX_PARADAS`, una constante de clase (o estática) que representa el número máximo de paradas que puede haber en el recorrido y que vale `25`. Debes usar esta constante y las de la clase `Parada` siempre que se requiera.
   - `numParadas`, un entero en el intervalo `[0..MAX_PARADAS]` que representa el número de paradas que aparecen incluidas en el recorrido en cada momento.
   - `trayecto`, un array de tipo base `Parada`, de capacidad `MAX_PARADAS`, para almacenar las paradas que aparecen en el recorrido, dispuestas en posiciones consecutivas del array desde la `0` hasta la `numParadas - 1` inclusive, **ordenadas cronológicamente por la hora de parada**, siendo `trayecto[0]` la parada de origen y `trayecto[numParadas - 1]` la parada de destino.
   - `numSubidas`, un entero no negativo que representa el número de paradas que hay en el recorrido y que permiten la incorporación de viajeros, es decir, las paradas que son estaciones o apeaderos.

   b) (1 punto) Un constructor que, dada una `Parada origen` y otra diferente `destino`, siendo la hora de parada de `destino` posterior a la de `origen`, crea un objeto `RecorridoTren` cuyo trayecto consta inicialmente de este par de paradas.

   c) (1.5 puntos) Un método con perfil:

   ```
   private int posicion(Instante hP)
   ```

   que devuelve `-1` si el instante `hP` es anterior al del origen o posterior al del destino, o ya existe en `trayecto` una parada en el mismo instante; en otro caso, devuelve el índice de la primera parada en `trayecto` cuyo instante es posterior a `hP`.

   Recuerda que para comparar dos instantes `t1`, `t2`, se puede usar el método `compareTo` de su clase, de manera que `t1.compareTo(t2)` es < 0, 0 o > 0 si `t1` es anterior, igual o posterior a `t2`, respectivamente.

d) (1.5 puntos) Un método con perfil:

```
public boolean incluir(Parada p)
```

para incluir `p` en el recorrido. Si la hora de la parada `p` no coincide con la de ninguna otra parada del recorrido, es posterior a la del origen y anterior a la del destino, entonces se incluye `p` ordenadamente en el recorrido y se devuelve `true`. Si `p` no se puede incluir por razones de horario, o porque se excede el máximo número de paradas, se devuelve `false`.

Nota que, en el caso de que `p` se pueda incluir, debes usar el método privado `posicion` para averiguar dónde situar `p` en el array `trayecto`. Una vez encontrada dicha posición, hay que hacerle un hueco a `p` en el array. Para ello, debes usar un método privado, ya implementado, con el siguiente perfil:

```
private void desplazarDcha(int ini, int fin)
```

que desplaza una posición hacia la derecha los elementos de `trayecto[ini..fin]`, siendo $0 \leq$ `ini` $\leq$ `fin` $\leq$ `numParadas - 1` $<$ `trayecto.length - 1`.

e) (1.5 puntos) Un método con perfil:

```
public Parada[] subidas()
```

que devuelve un array con las paradas del recorrido que permiten la subida de viajeros, es decir, son estaciones o apeaderos. La longitud de este array será igual al número de paradas en las que pueden subir viajeros, o `0` si no hubiera ninguna.

Solución:

```
import iipUtil.Instante;
public class RecorridoTren {
    public static final int MAX_PARADAS = 25;
    private Parada[] trayecto;
    private int numParadas;
    private int numSubidas;

    /** Precondicion: origen y destino son paradas distintas, siendo
     *  la hora de parada de destino posterior a la de origen */
    public RecorridoTren(Parada origen, Parada destino) {
        trayecto = new Parada[MAX_PARADAS];
        trayecto[0] = origen;
        trayecto[1] = destino;
        numParadas = 2;
        if (origen.getTipo() != Parada.APARTADERO) { numSubidas++; }
        if (destino.getTipo() != Parada.APARTADERO) { numSubidas++; }
    }

    private int posicion(Instante hP) {
        if (trayecto[0].getHoraParada().compareTo(hP) >= 0
            || trayecto[numParadas - 1].getHoraParada().compareTo(hP) <= 0) {
            return -1;
        }
        int i = 1;
        while (i < numParadas - 1 && trayecto[i].getHoraParada().compareTo(hP) < 0) {
            i++;
        }
        if (trayecto[i].getHoraParada().compareTo(hP) == 0) { return -1; }
        else { return i; }
    }

    /** Precondicion: 0 <= ini <= fin <= numParadas - 1 < trayecto.length - 1 */
    private void desplazarDcha(int ini, int fin) {
        for (int pos = fin + 1; pos > ini; pos--) {
            trayecto[pos] = trayecto[pos - 1];
        }
    }
```

```
    public boolean incluir(Parada p) {
        if (numParadas == MAX_PARADAS) { return false; }
        int pos = posicion(p.getHoraParada());
        if (pos == -1) { return false; }
        desplazarDcha(pos, numParadas - 1);
        trayecto[pos] = p;
        numParadas++;
        if (p.getTipo() != Parada.APARTADERO) { numSubidas++; }
        return true;
    }

    public Parada[] subidas() {
        Parada[] result = new Parada[numSubidas];
        int j = 0;
        for (int i = 0; i < numParadas && j < numSubidas; i++) {
            if (trayecto[i].getTipo() != Parada.APARTADERO) {
                result[j] = trayecto[i];
                j++;
            }
        }
        return result;
    }
}
```

2. 2 puntos Sea un entero `a` $>$ `1`. **Se pide:** implementar un método estático que, usando `'*'`, muestre por pantalla una figura compuesta por un triángulo rectángulo isósceles de altura `a` y su imagen especular, encarados por la hipotenusa y con sus bases separadas por un espacio en blanco. Por ejemplo, para `a = 4`, el método debe producir la siguiente figura:

```
*       *
**     **
***   ***
**** ****
```

Solución:

```
/** Precondicion: a > 1 */
public static void dibujar(int a) {
    int b = a * 2 + 1; // base de la figura a dibujar
    for (int i = 1; i <= a; i++) {
        for (int j = 1; j <= i; j++) { System.out.print('*'); }
        int blancos = b - i;
        for (int j = i; j < blancos; j++) { System.out.print(' '); }
        for (int j = 1; j <= i; j++) { System.out.print('*'); }
        System.out.println();
    }
}
```

3. 2 puntos **Se pide:** implementar un método estático que, dados dos arrays de caracteres `a` y `b`, ambos sin repetidos, devuelva el número de caracteres comunes. Por ejemplo, si `a` es {`'C','T','A','G'`} y `b` es {`'T','U','C'`}, el método debe devolver `2`.

Solución:

```
/** Precondicion: a sin repetidos, b sin repetidos */
public static int numCharComunes(char[] a, char[] b) {
    int comunes = 0;
    for (int i = 0; i < a.length; i++) {
        int j = 0;
        while (j < b.length && a[i] != b[j]) { j++; }
        if (j < b.length) { comunes++; }
    }
    return comunes;
}
```