Segundo Parcial de IIP - ETSInf
Fecha: 9 de enero de 2017. Duración: 2h 30'

1. 6 puntos Se dispone de la clase `Edificio` que representa un edificio de la UPV mediante dos tipos de información: la asociada a su construcción (coordenadas GPS y código de identificación en un plano) y la asociada al uso que se le ha asignado (tipo de uso y nombre de la entidad que lo usa). Esta clase ya es conocida y se muestra, a continuación, un resumen de su documentación:

Field Summary

Fields

Modifier and Type	Field and Description
`static int`	`DEPARTAMENTO` Constante que representa el tipo de edificio departamental.
`static int`	`ESCUELA` Constante que representa el tipo de edificio dedicado a docencia como aularios, escuelas o laboratorios.
`static int`	`SERVICIOS` Constante que representa el tipo de edificio para otras actividades como cafeterias, oficinas, etc.

Constructor Summary

Constructors

Constructor and Description
`Edificio()` Crea un `Edificio` de codigo `"1F"`, usado por la entidad `"DSIC"`, de tipo departamental y en las coordenadas `(39.4625, -0.3472)`.
`Edificio(java.lang.String c, java.lang.String e, int t, Punto p)` Crea un `Edificio` de codigo `c`, usado por la entidad `e`, de tipo `t` y en las coordenadas dadas por el `Punto p`.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`boolean`	`equals(java.lang.Object o)` Devuelve `true` si `o` es un `Edificio` con el mismo codigo, y las mismas coordenadas que `this`; en caso contrario, devuelve `false`.
`java.lang.String`	`getEntidad()` Devuelve la entidad que usa el edificio `this`.
`int`	`getTipo()` Devuelve el tipo del edificio `this`.
`int`	`masCercaRectorado(Edificio e)` Dado un `Edificio e`, devuelve: `-1` si el edificio `this` esta mas cerca de rectorado que el edificio `e`; `1` si `e` esta mas cerca de rectorado que `this`; o `0` si ambos edificios estan igual de cerca de rectorado.

Se pide: implementar la clase `PlanoVera` para representar los edificios del campus de Vera de la UPV mediante las componentes (atributos y métodos) que se indican a continuación.

Recuerda que debes utilizar las constantes de las clases `Edificio` y `PlanoVera` siempre que se requiera.

a) (0.5 puntos) Atributos, de los cuales solo es público el primero:

- `MAX_EDIFS`, una constante de clase (o estática) que representa el numero máximo de edificios que puede haber en el plano y que vale 50.
- `numEdifs`, un entero en el intervalo `[0..MAX_EDIFS]` que representa el número de edificios del plano en cada momento.
- `edifs`, un array de tipo base `Edificio`, de capacidad `MAX_EDIFS`, para almacenar los edificios del plano en cada momento, dispuestos en posiciones consecutivas del array, desde la `0` hasta la `numEdifs - 1` inclusive, **ordenados ascendentemente por su cercanía a rectorado**, siendo `edifs[0]` rectorado, `edifs[1]` el más cercano y `edifs[numEdifs - 1]` el más lejano. Si dos edificios están **igual de cerca** de rectorado, ocuparán en el array dos posiciones consecutivas `i` e `i + 1`, $1 \leq$ `i` $<$ `numEdifs - 1`, siendo `edifs[i + 1]` un edificio añadido al array **con posterioridad** a `edifs[i]`.
- `numEscuelas`, un entero no negativo que representa el número de edificios docentes que hay en el plano en cada momento.

b) (1 punto) Un constructor por defecto (sin parámetros) que crea un objeto `PlanoVera` con 1 único edificio con las siguientes características: un edificio de servicios usado por la entidad `"Rectorado"`, con código `"3A"` y coordenadas `(39.4823, -0.3457)`.

c) (1.5 puntos) Un método con perfil:

```
private int posicionDe(Edificio e)
```

que, dado un `Edificio e`, devuelve la posición del primer edificio del array (de índice menor) que esté más lejos de rectorado que `e`, o `numEdifs` si no hay ningún edificio más lejos de rectorado que `e`.

Nota que debes usar el método `masCercaRectorado(Edificio)` de la clase `Edificio`.

d) (1.5 puntos) Un método con perfil:

```
public boolean anyadir(Edificio e)
```

que, dado un `Edificio e` que **no está en el plano**, lo añade, si cabe, de manera **ordenada** según su cercanía a rectorado, actualizando los atributos `numEdifs` y, si procede, `numEscuelas`. El método devuelve `true` si se ha añadido con éxito, o `false` si no caben más edificios en el plano.

Nota que, en el caso de que `e` quepa en el array, debes usar el método privado `posicionDe(Edificio)` para saber la posición del array `edifs` en la que situar el edificio `e`. Una vez encontrada dicha posición, hay que hacerle un hueco a `e` 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 del array `edifs` desde la posición `ini` a la posición `fin` inclusive ($0 \leq$ `ini` $\leq$ `fin` $\leq$ `numEdifs - 1` $<$ `edifs.length - 1`). Nota que, por precondición, si `ini` $>$ `fin`, no realiza ningún desplazamiento.

e) (1.5 puntos) Un método con perfil:

```
public Edificio[] filtrarTipoEscuela()
```

que devuelve un array de `Edificio` con los edificios docentes, o escuelas, del plano. La longitud de este array será igual al número de edificios de tipo docente, o 0 si no hay ningún edificio de dicho tipo en el plano.

Solución:

```
public class PlanoVera {
    public static final int MAX_EDIFS = 50;
    private int numEdifs;
    private Edificio[] edifs;
    private int numEscuelas;

    public PlanoVera() {
        edifs = new Edificio[MAX_EDIFS];
        edifs[0] = new Edificio("3A", "Rectorado",
            Edificio.SERVICIOS, new Punto(39.4823, -0.3457));
        numEdifs = 1;
        numEscuelas = 0;
    }

    private int posicionDe(Edificio e) {
        int i = 1;
        while (i < numEdifs && edifs[i].masCercaRectorado(e) <= 0) { i++; }
        return i;
    }

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

    /** Precondicion: e no esta en el plano */
    public boolean anyadir(Edificio e) {
        boolean res = false;
        if (numEdifs != MAX_EDIFS) {
            int pos = posicionDe(e);
            desplazarDcha(pos, numEdifs - 1);
            edifs[pos] = e;
            numEdifs++;
            if (e.getTipo() == Edificio.ESCUELA) { numEscuelas++; }
            res = true;
        }
        return res;
    }
```

```
    public Edificio[] filtrarTipoEscuela() {
        Edificio[] aux = new Edificio[numEscuelas];
        int k = 0;
        for (int i = 1; i < numEdifs && k < numEscuelas; i++) {
            if (edifs[i].getTipo() == Edificio.ESCUELA) {
                aux[k] = edifs[i];
                k++;
            }
        }
        return aux;
    }
}
```

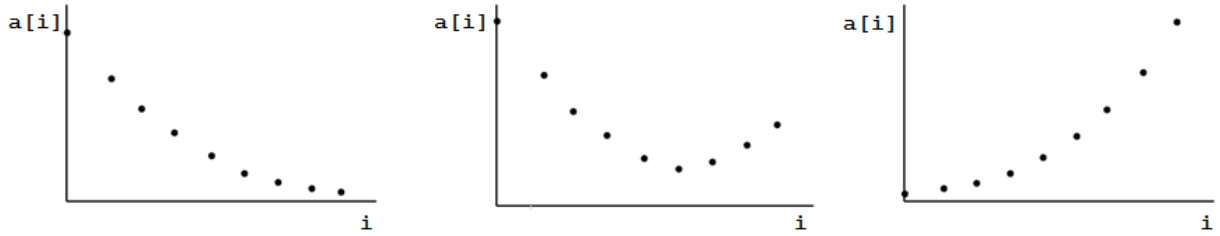
2. 2 puntos Sea un entero `n` $\geq$ `2`. **Se pide:** implementar un método estático que, para todos los enteros entre `2` y `n` inclusive, devuelva un `String` con la lista de sus divisores propios. Recuerda que los *divisores propios* de un entero son todos sus divisores excepto él mismo y la unidad. Por ejemplo, para `n = 18`, el método deberá producir el siguiente `String`:

```
Divisores propios de 2:
Divisores propios de 3:
Divisores propios de 4: 2
Divisores propios de 5:
Divisores propios de 6: 2 3
Divisores propios de 7:
Divisores propios de 8: 2 4
Divisores propios de 9: 3
Divisores propios de 10: 2 5
Divisores propios de 11:
Divisores propios de 12: 2 3 4 6
Divisores propios de 13:
Divisores propios de 14: 2 7
Divisores propios de 15: 3 5
Divisores propios de 16: 2 4 8
Divisores propios de 17:
Divisores propios de 18: 2 3 6 9
```

Solución:

```
/** Precondicion: n >= 2 */
public static String divPropios(int n) {
    String res = "";
    for (int i = 2; i <= n; i++) {
        res += "Divisores propios de " + i + ": ";
        for (int j = 2; j <= i / 2; j++) {
            if (i % j == 0) { res += j + " "; }
        }
        res += "\n";
    }
    return res;
}
```

3. 2 puntos Sea un array `a` de reales y longitud `n` $\geq$ `2`, tal que sus componentes se ajustan al perfil de una curva cóncava, es decir, existe un mínimo en una cierta posición `k`, `0` $\leq$ `k` $<$ `n` (esto es, los valores en `a[0..k]` son estrictamente decrecientes y los valores en `a[k..n - 1]` son estrictamente crecientes); el mínimo se puede encontrar en uno de los extremos del array. **Se pide:** implementar un método estático que, dado el array, devuelva la posición del mínimo. Por ejemplo, para los arrays de las siguientes figuras, el método debería devolver 8, 5 y 0, respectivamente.



Solución:

```
/** Precondicion: Las componentes de a, a.length >= 2,
 *  se ajustan al perfil de una curva concava.
 */
public static int minimoConcava(double[] a) {
    int i = 0;
    while (i < a.length - 1 && a[i] > a[i + 1]) { i++; }
    return i;
}
```