Segundo Parcial de IIP - ETSInf
Fecha: 17 de enero de 2014. Duración: 2:30 horas.

1. 6.5 puntos Se dispone de la clase `Astro` (que permite representar diferentes tipos de astros), ya conocida y de la que se muestra a continuación un resumen de su documentación:

Field Summary

Fields

Modifier and Type	Field and Description
`static int`	`ESTRELLA` Constante que indica que el `Astro` es de tipo estrella.
`static int`	`GALAXIA` Constante que indica que el `Astro` es de tipo galaxia.
`static int`	`NEBULOSA` Constante que indica que el `Astro` es de tipo nebulosa.

Constructor Summary

Constructors

Constructor and Description
`Astro(java.lang.String n, int t, double b, double d)` Crea un `Astro` con un nombre, tipo, brillo y distancia dados.

Method Summary

Methods

Modifier and Type	Method and Description
`boolean`	`equals(java.lang.Object o)` Comprueba si el `Astro` en curso es igual a otro dado; i.e. si coinciden en nombre, tipo, brillo y distancia.
`int`	`getTipo()` Devuelve el tipo del `Astro` en curso.
`int`	`masBrillante(Astro otro)` Devuelve 1 si el `Astro` en curso es más brillante en magnitud absoluta que un `Astro` dado, 0 si tienen la misma magnitud absoluta y -1 si el `Astro` dado es más brillante en magnitud absoluta que el `Astro` en curso.
`java.lang.String`	`toString()` Devuelve un `String` con la información del `Astro` en curso con el siguiente formato: "nombre: tipo (brillo, distancia)"; p.e., "Sirius: Estrella (-1.42, 8.70)".
`java.lang.String`	`visibleCon()` Devuelve un `String` que describe la forma en la que el `Astro` en curso puede ser observado ("a simple vista", "con prismáticos", "con telescopio" o "con grandes telescopios").

Se pide: implementar la clase `CatalogoAstronomico` que, como su nombre indica, representa un catálogo de astros mediante las componentes (atributos y métodos) que se indican a continuación.

a) (0.5 puntos) Atributos:

- `MAX_ASTROS`, una constante que representa el número máximo de astros de un catálogo: 120.
- `numAstros`, un entero en el intervalo `[0..MAX_ASTROS]` que representa el número de astros que tiene en cada momento el catálogo.
- `catalogo`, un array de tipo base `Astro`, de capacidad `MAX_ASTROS` y cuyas componentes se almacenan secuencialmente, en posiciones consecutivas del catálogo desde la `0` hasta la `numAstros-1`.
- `numEstrellasSimpleVista`, que representa el número de astros del catálogo que son estrellas visibles a simple vista.

b) (0.5 puntos) Un constructor por defecto (sin parámetros) que crea un catálogo vacío, con 0 astros.

c) (1 punto) Un método con perfil:

```
private int posicionDe(Astro a)
```

que, dado un `Astro a`, devuelve su posición en el catálogo, o -1 si no está.

d) (0.5 puntos) Un método con perfil:

```
private boolean esEstrellaSimpleVista(int i)
```

que, dada una posición válida `i` del catálogo ($0 \leq$ `i` $<$ `numAstros`), devuelve `true` si el `Astro` en dicha posición es una estrella visible a simple vista, o `false` si no lo es.

e) (1 punto) Un método con perfil:

```
public boolean añade(Astro a)
```

que devuelve `true` tras añadir el `Astro a` dado al catálogo. Si `a` no cabe o ya está en el catálogo, el método devuelve `false` para advertir que no se ha podido añadir. Se deben usar los métodos privados `posicionDe(Astro)` (para buscar el `Astro`) y `esEstrellaSimpleVista(int)` (para actualizar el atributo `numEstrellasSimpleVista` si procede).

f) (1 punto) Un método con perfil:

```
public Astro primeroMasBrillanteQue(Astro a)
```

que devuelve el primer `Astro` del catálogo que es más brillante en magnitud absoluta que el `Astro a` dado, o `null` si no hay ninguno.

g) (1 punto) Un método con perfil:

```
public Astro[] filtraEstrellasSimpleVista()
```

que devuelve un array de `Astro` con las estrellas visibles a simple vista que contiene el catálogo. La longitud de dicho array será igual al número de estrellas visibles a simple vista, o 0 si no hay ninguna. Se debe usar el método privado `esEstrellaSimpleVista(int)`.

h) (1 punto) Un método con perfil:

```
public Astro brillaMas()
```

que devuelve el `Astro` que es más brillante en magnitud absoluta de todos los del catálogo, o `null` si el catálogo está vacío.

Solución:

```
public class CatalogoAstronomico {
    public static final int MAX_ASTROS = 120;
    private Astro[] catalogo;
    private int numAstros, numEstrellasSimpleVista;

    public CatalogoAstronomico() {
       catalogo = new Astro[MAX_ASTROS];
       numAstros = 0; numEstrellasSimpleVista = 0;
    }

    private int posicionDe(Astro a) {
        int i = 0;
        while(i<numAstros && !catalogo[i].equals(a)) i++;
        if (i<numAstros) return i;
        else return -1;
    }
```

```
    private boolean esEstrellaSimpleVista(int i) {
        return catalogo[i].getTipo()==Astro.ESTRELLA &&
               catalogo[i].visibleCon().equals("a simple vista");
    }

    public boolean añade(Astro a) {
        boolean res = false;
        int pos = posicionDe(a);
        if (pos==-1) {
             if (numAstros!=MAX_ASTROS) {
                catalogo[numAstros++] = a;
                if (esEstrellaSimpleVista(numAstros-1)) numEstrellasSimpleVista++;
                res = true;
             }
        }
        return res;
    }

    public Astro primeroMasBrillanteQue(Astro a) {
        int i = 0;
        while(i<numAstros && catalogo[i].masBrillante(a)!=1) i++;
        if (i<numAstros) return catalogo[i];
        else return null;
    }

    public Astro[] filtraEstrellasSimpleVista() {
        Astro[] aux = new Astro[numEstrellasSimpleVista];
        int k = 0;
        for(int i=0; i<numAstros; i++)
            if (esEstrellaSimpleVista(i)) {
                aux[k] = catalogo[i];
                k++;
            }
        return aux;
    }

    public Astro brillaMas() {
        Astro masBrilla = catalogo[0];
        for(int i=1; i<numAstros; i++)
            if (catalogo[i].masBrillante(masBrilla)==1) masBrilla = catalogo[i];
        return masBrilla;
    }
}
```

2. 1.5 puntos **Se pide:** implementar un método estático que, dado un valor entero $n \geq 0$, muestre en la salida estándar la letra 'Z' en n líneas, tal como se ilustra en la siguiente figura para el caso en el que $n = 5$. Nótese que la primera y última línea tendrán n caracteres 'Z', mientras que el resto de líneas tendrán solo uno situado en su diagonal secundaria.

```
ZZZZZ
   Z
  Z
 Z
ZZZZZ
```

Solución:

```
/** n >= 0 */
public static void dibujaZ(int n) {
    for(int i=0; i<n; i++) System.out.print('Z');
    System.out.println();
    for(int i=0; j<n-2; i++) {
        for(int j=0; j<n-2-i; j++) System.out.print(' ');
        System.out.println('Z');
    }
    for(int i=0; i<n; i++) System.out.print('Z');
    System.out.println();
}
```

3. 2 puntos La norma $\|a\|$ de un vector $a = (a_1, a_2, \ldots, a_n)$ viene dada por $\|a\| = \sqrt{a_1^2 + a_2^2 + \ldots + a_n^2}$. El producto escalar $a \cdot b$ de dos vectores a y b de igual dimensión se calcula como $a \cdot b = a_1b_1 + a_2b_2 + \ldots + a_nb_n$. El coseno del ángulo θ que forman a y b de igual dimensión se puede calcular como $cos\ \theta = \frac{a \cdot b}{\|a\|\|b\|}$.

Teniendo en cuenta que un vector se representa como un array en Java, **se pide:**

a) Implementar un método estático privado `norma` que devuelva la norma de un array de enteros `a` dado, tal que `a.length`>0.

b) Implementar un método estático público `coseno` que, haciendo uso del método `norma`, devuelva el coseno del ángulo que forman dos arrays de enteros `a` y `b` dados, tales que `a.length`>0, `b.length`>0 y `a.length = b.length`.

Solución:

```
/** a.length>0 */
private static double norma(int[] a) {
        double res = 0;
        for(int i=0; i<a.length; i++) res += a[i]*a[i];
        return Math.sqrt(res);
}
/** a.length>0, b.length>0, a.length=b.length */
public static double coseno(int[] a, int[] b) {
       double num = 0;
       for(int i=0; i<a.length; i++) num += a[i]*b[i];
       double den = norma(a)*norma(b);
       return num/den;
}
```