dit UPM	ETSIT-UPM. PROGRAMACIÓN. CONVOCATORIA ORDINARIA. JUNIO 2013. APELLIDOS: NOMBRE: EMAIL:	

Problema 1. La mayoría de los reproductores multimedia permiten crear listas de reproducción; este problema trata de crear las clases necesarias para elegir las canciones que forman una lista de reproducción. Suponga que existe la clase *Cancion* con atributos *nombre*, *cantante* y *estilos*, de tipo *String* y que no pueden ser nulos; constructor con esos parámetros y métodos accesores para cada uno. El atributo *estilos* es una *String* en la que aparecen los nombres de los estilos musicales de la canción (una palabra cada uno) separados por un espacio blanco como en "jazz blues funk". Puede haber diferentes criterios para elegir las canciones.

Pregunta 1. (0.5 puntos) Escriba una interfaz *CriterioEleccion* que contiene un método para indicar si se elige o no una canción para una lista de reproducción.

```
interface CriterioEleccion {
  public boolean seleccionar(Cancion c);
}
```

Pregunta 2. (0.5 puntos) Escriba una clase que implemente la interfaz anterior, y que permita elegir las canciones de un determinado cantante. El nombre del cantante se indicará como parámetro del constructor.

```
class SeleccionCantante implements CriterioEleccion {
  private String cantante;
  public SeleccionCantante (String c) {
    this.cantante = c;
  }
  public boolean seleccionar (Cancion c) {
    return (c.getCantante().equals(this.cantante));
  }
}
```

Pregunta 3. (2 puntos) Escriba una clase que implemente la interfaz anterior, y permita elegir canciones de determinados estilos. El atributo de esta clase será un conjunto (*Set*) de *String* con los nombres de los estilos; debe tener método constructor sin parámetros y un método para añadir un estilo (*String*) al conjunto. Para que una canción se elija por estilo basta con que alguna de las palabras en su atributo estilo aparezca en el conjunto de estilos de esta clase. Por ejemplo, si el conjunto de estilos en esta clase contiene los estilos "jazz" "gospell" y "pop", una canción con los estilos "jazz blues funk" sí se elije, pero una canción con estilos "rock funk newage" no. Para extraer palabras (secuencias de caracteres cualesquiera, separadas entre sí por uno o más espacios en blanco) de una *String*, se puede usar la clase *Scanner*. El constructor de la clase, *Scanner(String s)*, toma como argumento la cadena de la que se van a extraer las palabras. El método *boolean hasNext()* indica si quedan caracteres por procesar en la *String* del objeto *Scanner* al que se aplica, y *String next()* devuelve una *String* con la siguiente palabra.

```
class SeleccionEstilos implements CriterioEleccion {
  private Set<String> estilos;
  public preferenciaEstilos () {
    this.estilos = new HashSet<String> ();
  }
  public void add (String e) {
    this.estilos.add (e);
  }
  public boolean seleccionar (Cancion c) {
    Scanner sc = new Scanner (c.getEstilos());
    while (sc.hasNext()) {
      if (this.estilos.contains(sc.next()))
        return true;
    }
    return false;
  }
}
```

dit UPM	ETSIT-UPM. PROGRAMACIÓN. CONVOCATORIA ORDINARIA. JUNIO 2013. APELLIDOS: NOMBRE: EMAIL:	

Pregunta 4. (2 puntos) Escriba un método de la clase *Reproductor* que reciba un *Scanner* como parámetro y devuelva un *CriterioEleccion*, sin lanzar excepciones. Si la primera palabra que se lee del *Scanner* es "cantante", tomará la siguiente palabra como el nombre del cantante y creará un criterio de selección por cantante; si la primera palabra es "estilos" creará un criterio de selección por estilo, tomando cada palabra restante del *Scanner* como un estilo. En el resto de casos devuelve *null*. Para obtener las palabras de un objeto Scanner puede usar los métodos *boolean hasNext()* y *String next()*. El *Scanner* toma las palabras de líneas de texto como estas:

cantante ElianeElias
estilos jazz bossanova rock flamenco

```
public CriterioEleccion leePreferencias (Scanner sc) {
    if (¡sc.hasNext())
      return null;
     String tipo = sc.next ();
     if ((tipo != null) && tipo.equals ("cantante")) {
          return new SeleccionCantante (sc.next());
     }
     if (tipo ¡= null) && tipo.equals("estilos")) {
          SeleccionEstilos se = new SeleccionEstilos ();
          while (sc.hasNext ())
            se.add (sc.next());
          return se;
     }
  return null;
}
```

dit UPM	ETSIT-UPM. PROGRAMACIÓN. CONVOCATORIA ORDINARIA. JUNIO 2013. APELLIDOS: NOMBRE: EMAIL:	

Problema 2. El sistema de D'Hondt es una fórmula electoral creada por Victor d'Hondt, que permite obtener el número de escaños que hay que asignar a las candidaturas en proporción a los votos que han conseguido.

Para aplicar este sistema supondremos 5 Partidos (A,B,C,D,E) y 7 escaños a asignar. Si los votos de cada uno de los partidos fueran: 340.000, 280.000, 160.000, 60.000 y 15.000 respectivamente, se procedería así: se crea una matriz de 7 columnas (una por escaño) y 5 filas (una por partido); en la columna i se escribe el número de votos de cada partido dividido por i +1 (la numeración de las casillas de un array en Java comienza en 0); se seleccionan las casillas que tienen los 7 mayores valores y a cada partido se le asigna tantos escaños como casillas de su columna hayan sido seleccionadas.

La tabla muestra que al partido A le han correspondido 3 escaños (1,3 y 6) al B 3 escaños (2,5 y 7) y al C 1 escaño (el 4).

	1	2	3	4	5	6	7
A	[1] 340.000	[3] 170.000	[6] 113.333	85.000	68.000	56.667	48.571
B	[2] 280.000	[5] 140.000	[7] 93.333	70.000	56.000	46.667	40.000
C	[4] 160.000	80.000	53.333	40.000	32.000	26.667	22.857
D	60.000	30.000	20.000	15.000	12.000	10.000	8.571
E	15.000	7.500	5.000	3.750	3.000	2.500	2.143

Pregunta 1. (1 punto) Escriba el método *int [][] crearRellenarTabla (int [] votos, int puestos) throws Exception* de la clase *DHondt* que recibe como parámetros la lista de los votos de cada partido y el número de escaños a repartir y crea la tabla con los valores en cada casilla. Lanza excepciones si el parámetro *votos* es *null* o *puestos* es menor o igual que cero.

```
public int [][] crearRellenarTabla (int [] votos, int puestos) throws Exception{
        if ((votos == null) || (puestos<=0))
                throw new Exception();
        int [][] cocientes = new int [votos.length][puestos];
        for (int i= 0; i< cocientes.length ; i++){
                for (int j=0; j < puestos; j++){
                        cocientes [i][j] = votos [i]/(j+1);
                }
        }
        return cocientes;
}
```

Pregunta 2. (2 puntos) El método *int [] asignaPuestos (int [][] tabla, int puestos) throws Exception* devuelve un array con tantas posiciones como indique el parámetro *puestos* y donde estarán los valores más altos de la *tabla* que se pasa como parámetro. Para crear este método suponga que existe el método *int [] ordenaArray (int [] lista)* que a partir del parámetro devuelve otro array donde los valores están ordenados de mayor (en la posición 0) al menor. Lanza excepciones si el parámetro *tabla* es *null* o *puestos* menor o igual que cero.

```
        public int [] asignaPuestos (int [] [] tabla, int puestos) throws Exception {
                if ((tabla == null) || (puestos <= 0))
                        throw new Exception();
                int [] lista = new int [tabla.length * tabla[0].length];
                int c =0;
                for (int i=0; i < tabla.length; i++){
                        for (int j=0; j < tabla[i].length; j++){
                                lista [c] =tabla[i][j];
                                c++;
                        }
                }
                int [] l1 = this.ordenaArray (lista);
                int [] res = new int [puestos];
                for (int i = 0; i < puestos; i++)
                        res[i] = l1[i];
                return res;
        }
```

Pregunta 3. (2 puntos) Suponga que existe un método como el que aparece a continuación, y escriba uno o más métodos de prueba JUnit4 para ese método. Debe hacer pruebas, al menos, con valores que lancen excepciones y con los valores del ejemplo explicado en el enunciado del problema. Recuerde que puede usar el método *fail()* de JUnit4 para indicar que una prueba ha fallado, y que si el método de prueba acaba sin fallar, la prueba se da por válida. Suponga que dispone del método *void assertArrayEquals(int[]expected, int[]actuals)* para comprobar que dos arrays contienen los mismos valores.

```
public int[] aplicaDHondt (int[]votos, int puestos) throws Exception {
   return this. asignaPuestos (this.crearRellenarTabla (int[] votos, int puestos), int puestos);
}
```

```
@Test
public void testCrearTabla () {
try {
            DHondt d = new DHondt.aplicaDHondt (null, 1);
            fail();
}catch (Exception e) {
}
try {
            DHondt d = new DHondt.aplicaDHondt (new int[] { 0}, 0);
            fail();
}catch (Exception e) {
}
try {
            DHondt d = new DHondt();
            Int [] obtenido = d.aplicaDHondt  (new int [] {340000,280000,160000,60000,15000}, 7);
            assertArrayEquals(obtenido, new int[] {340000, 280000, 170000, 160000, 140000,113333, 93333});
} catch (Exception e) {
            fail();
}
}
```