Recuperación del segundo parcial de IIP - ETSInf.
Fecha: 23 de enero de 2015. Duración: 2:30 horas.

1. 6.5 puntos Una empresa de mensajería debe gestionar la distribución de envíos, que pueden ser de tipo paquete o documento. Cada envío tiene como datos su origen (`String`), su destino (`String`), una referencia (`String`) que lo identifica, un tipo (`int`), y una ubicación (`String`). Un envío se puede guardar en un almacén, en donde se le asigna como ubicación el lugar concreto o estante en donde se guarda el envío mientras permanece en dicho almacén. Cuando un envío se extrae de un almacén, se le da de baja cambiándole la ubicación a `"En tránsito"`.

   Para representar el problema se dispone de la clase `Envio` de la que se muestra a continuación un resumen de su documentación:

Field Summary

static int	**DOCUMENTO** Constante que indica que el envio es un documento.
static int	**PAQUETE** Constante que indica que el envio es un paquete.

Constructor Summary

Envio(String origen, String destino, String ref, int tipo) Crea un envio con origen, destino, referencia y tipo (PAQUETE o DOCUMENTO) dados; y con ubicacion por defecto "En transito".

Method Summary

boolean	**equals**(Object o) Comprueba si un envio es igual a otro o dado, esto es, si sus referencias son coinciden.
String	**getDestino**() Devuelve el destino del envio.
String	**getOrigen**() Devuelve el origen del envio.
String	**getRef**() Devuelve la referencia del envio.
int	**getTipo**() Devuelve el tipo del envio.
String	**getUbicacion**() Devuevle la ubicacion del envio.
void	**setUbicacion**(String nuevaUbic) Actualiza a nuevaUbic la ubicacion del envio.
String	**toString**() Devuelve un String con la informacion del envio.

   Observar que un objeto `Envio` se crea a partir de los `String` que indican su `origen`, `destino` y `referencia` y del `int` que indica su `tipo`. Según se ha indicado más arriba, un `Envio` tiene también otro atributo `String ubicacion`, que se inicia a `"En tránsito"` cuando se crea el objeto.

   La clase posee dos constantes `int` públicas, `DOCUMENTO` y `PAQUETE`, iniciadas a los valores enteros que se corresponden a los respectivos tipos de envíos.

   Notar que se dispone de métodos consultores para todos los datos del `Envio`, y un método modificador de la ubicación.

   Se pide: implementar la clase `Almacen` que representa un almacén de envíos mediante las componentes (atributos y métodos) que se indican a continuación.

   a) (0.5 puntos) Atributos:

   - `MAX_ENVIOS`, una constante de clase (o estática) que representa el número máximo de envíos que caben en el almacén, y que debe valer 2000. Recordar que esta constante, así como las de la clase `Envio`, se deben usar siempre que se requiera.
   - `numE`, un entero en el intervalo `[0..MAX_ENVIOS]` que representa el número de envíos guardados en el almacén en cada momento.

- `lista`, un array de tipo base `Envio`, de capacidad `MAX_ENVIOS`. Las componentes de este array contienen los envíos del almacén en posiciones consecutivas desde la `0` hasta la `numE-1`, y por orden de llegada, es decir, `lista[0]` es el envío más antiguo de los que permanecen en el almacén en un momento dado, y `lista[numE-1]` el más reciente. No existen envíos repetidos.
- `numDoc` y `numPaq`, enteros que representan el número de envíos en el almacén en un momento dado, que son de tipo documento o paquete respectivamente.

Los métodos que almacenen o extraigan un envío del almacén deberán mantener los envíos consecutivos en el array, por orden de llegada. Los contadores de envíos de cada tipo deberán actualizarse adecuadamente.

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

c) (1 punto) Un método con perfil:

```
private int posicionDe(String ref)
```

que devuelve la posición en la que aparece el envío con referencia `ref` en el array `lista`, o -1 si no está. Es un método de utilidad para los métodos `almacenar`, `buscar` y `darBaja` que se piden más adelante.

d) (1 punto) Un método con perfil:

```
public boolean almacenar(Envio e, String estante)
```

que actualiza el almacén añadiéndole el envío `e`. En el propio envío `e` se registra la información del estante en que se guarda, para lo que el método modifica la ubicación de `e` al `String estante` que se pasa como dato al método. El envío no se almacenará si existe otro igual (con la misma referencia) o el almacén está lleno, en cuyo caso el método devolverá `false`. En caso contrario, devolverá `true` indicando que el envío se ha podido guardar.

e) (0.75 puntos) Un método con perfil:

```
public Envio buscar(String ref)
```

que devuelve el envío del almacén cuya referencia es `ref`. Si no lo encuentra, devuelve `null`.

f) (1 punto) Un método con perfil:

```
public Envio darBaja(String ref)
```

que busca el `Envio` de referencia `ref`. Si lo encuentra, cambia su ubicación a `"En tránsito"`, lo extrae del almacén y lo devuelve como resultado. Si no hubiera ningún envío para esa referencia, devuelve `null`.

Recordar que el método debe mantener los envíos apareciendo consecutivos en las primeras componentes del array `lista`, y respetando el orden de llegada, de manera que cuando se saque un envío del array, los envíos a su derecha se moverán una posición hacia la izquierda.

g) (1 punto) Un método con perfil:

```
public Envio[] obtenerEnvios(int tipo)
```

que devuelve un array de `Envio` del tipo que indique el parámetro `tipo`. La longitud de este array será igual al número de envíos de dicho tipo o 0 si no hay ninguno (también se devolverá un array de talla 0 si el tipo no corresponde a ninguno de los tipos válidos de `Envio`).

h) (0.75 puntos) Un método con perfil:

```
public String toString()
```

que devuelve un `String` con la información de todos los envíos que haya en el almacén, por orden de antigüedad, cada uno en una línea distinta.

Solución:

```
public class Almacen {
    public static final int MAX_ENVIOS = 2000;
    private Envio[] lista;
    private int numE, numDoc, numPaq;

    public Almacen() {
        lista = new Envio[MAX_ENVIOS];
        numE = numDoc = numPaq = 0;
    }

    private int posicionDe(String ref) {
        int i = 0;
        while (i < numE && !ref.equals(lista[i].getRef())) i++;
        if (i < numE) return i; else return -1;
    }
```

```
    public boolean almacenar(Envio e, String estante) {
        if (posicionDe(e.getRef()) == -1 || numE == MAX_ENVIOS) return false;
        e.setUbicacion(estante);
        lista[numE++] = e;
        if (e.getTipo() == Envio.PAQUETE) numPaq++; else numDoc++;
        return true;
    }

    public Envio buscar(String ref) {
        int pos = posicionDe(ref);
        if (pos != -1) return lista[pos]; else return null;
    }

    public Envio darBaja(String ref) {
        int pos = posicionDe(ref);
        if (pos == -1) return null;
        Envio e = lista[pos];
        e.setUbicacion("En transito");
        for (int i = pos + 1; i < numE; i++) lista[i - 1] = lista[i];
        numE--;
        if (e.getTipo() == Envio.PAQUETE) numPaq--; else numDoc--;
        return e;
    }

    public Envio[] obtenerEnvios(int tipo) {
        int talla = 0;
        switch (tipo) {
            case Envio.DOCUMENTO: talla = numDoc; break;
            case Envio.PAQUETE: talla = numPaq; break;
        }
        Envio[] lEnv = new Envio[talla];
        for (int i = 0, j = 0; j < talla; i++)
            if (lista[i].getTipo() == tipo) {
                lEnv[j] = lista[i];
                j++;
            }
        return lEnv;
    }

    public String toString(){
        String result = "Listado de envios del almacen:\n";
        for (int i = 0; i < numE; i++)
            result += lista[i] + "\n";
        return result;
    }
}
```

2. 1.75 puntos **Se pide:** Implementar un método de clase (o estático) tal que, dados un array `a` de `double` (`a.length` > 0) y un entero `n` (`n` > 0), modifique todos los elementos del array de forma que el valor máximo del mismo sea `n` y el resto estén escalados con este valor. Es decir, si el máximo de `a` es `m`, la relación entre el valor nuevo y el antiguo de una componente del array debe ser `n/m`. Por ejemplo, si el array `a` es {`4.5, 27.0, 18.0, 1.5`} y `n = 9`, `a` debe cambiar a {`1.5, 9.0, 6.0, 0.5`}.

Solución:

```
/** a.length > 0, a[i] > 0 ∀i: 0 <= i < a.length y n > 0 */
public static void escalar(double[] a, int n) {
    double maximo = a[0];
    for (int i = 1; i < a.length ; i++)
        if (a[i] > maximo) maximo = a[i];
    for (int j = 0; j < a.length; j++)
        a[j] = (a[j] / maximo) * n;
}
```

3. [1.75 puntos] El postulado de Bertrand se enuncia así: "Si n es un número natural mayor que 3, entonces siempre existe un número primo p tal que $n < p < 2 * n - 2$ ". En una cierta clase se dispone de un método con el perfil:

```
/** n > 1 */
public static boolean es_primo(int n)
```

que, dado un `n` mayor que 1, comprueba si es un número primo.

Se pide: Implementar un método estático (que se escribiría dentro de la misma clase) que, dado un número natural `n` tal que `n` $>$ `3`, determine, usando el método `es_primo` anterior, qué número primo `p` cumple el postulado de Bertrand para `n`. De haber más de uno, debe devolver el más pequeño. Por ejemplo, para `n = 8`, los primos dentro del intervalo `[9,13]` son el `11` y el `13`, luego el método deber devolver `11`.

Solución:

```
/** n > 3 */
public static int bertrand(int n) {
    int p = n + 1;
    boolean enc = false;
    while (p < (2 * n - 2) && !enc)
           if (es_primo(p)) enc = true;
       else p++;
    return p;
}
```