Segundo Parcial de PRG - ETSInf
Fecha: 7 de junio de 2012. Duración: 2 horas y 30 minutos

1. 1.5 puntos El siguiente método pide al usuario un valor entero, que lee como una String, `val`, la cual se transforma, mediante la operación `Integer.parseInt(val)` en el valor entero que contiene.

```
public static int leerInt() {
    Scanner tcl = new Scanner(System.in);
    System.out.print("Da el valor: ");
    String val = tcl.nextLine().trim();
    int ret = Integer.parseInt(val);
    return ret;
}
```

   El problema es que si lo que contiene la `String` no es un entero correctamente escrito (por ejemplo porque contiene alguna letra) se produce una excepción `NumberFormatException`, deteniéndose a continuación el proceso de lectura.

   Se pide: Modificar el método `leerInt()` para que, en caso de producirse una excepción `NumberFormatException` durante la transformación de la `String` a `int`, vuelva a pedirse el valor las veces que sea necesario hasta que sea un número correcto.

   Solución:

```
public static int leerInt() {
    Scanner tcl = new Scanner(System.in);
    int ret=0;
    boolean salir = false;
    while (!salir)
        try {
            System.out.print("Da el valor: ");
            String val = tcl.nextLine().trim();
            ret = Integer.parseInt(val);
            salir = true;
        } catch (NumberFormatException e) {System.out.println("ERROR");}
    return ret;
}
```

2. 1.5 puntos Dado cierto fichero de texto llamado en el sistema "origen.txt", así como cierto valor entero `umbral`, **se pide** diseñar un método que escriba en el sistema un nuevo fichero "destino.txt"que contenga exclusivamente y en el mismo orden de aparición, todas las líneas de texto del fichero original que tengan una longitud, en número de caracteres, mayor o igual que `umbral`.

   Si el fichero origen estuviera vacío el nuevo fichero generado también deberá estar vacío.

Solución:

```
public static void nuevoFichero(int umbral) throws IOException {
    String sin = "origen.txt"; String sout = "destino.txt";

    Scanner fin = new Scanner(new File(sin));
    PrintWriter fout = new PrintWriter(new File(sout));
    while (fin.hasNextLine()) {
        String aux = fin.nextLine();
        if (aux.length() >= umbral) fout.println(aux);
    }
    fin.close(); fout.close();
}
```

3. 3.0 puntos Se desea añadir a la clase `ListaPIIntEnla` un par de métodos nuevos, **internos** a la clase, que permitan obtener una lista a partir de los elementos de un array y viceversa. **Estas nuevas operaciones deberán escribirse sin usar las operaciones públicas ya existentes en la clase**.

   Gracias a estos métodos se podrá ordenar una lista con punto de interés transformándola en un array, el cual se puede ordenar con alguno de los métodos de ordenación conocidos, y a partir del cual se puede obtener la lista ordenada. En concreto, **se pide** resolver los siguientes apartados:

   1. Método `toArray()`, que deberá devolver un array que contenga exactamente y en el mismo orden los elementos de la `ListaPIIntEnla`.
   2. Método constructor `ListaPIIntEnla(int[] a)`, que dado un array de enteros construya una `ListaPIIntEnla` con los elementos del array, respetando su orden.
   3. Supóngase la clase `ListaPIIntEnla` completada con los dos métodos anteriores. Además, supóngase que en una clase `PruebaLista` se dispone ya realizado de un método de ordenación rápida de arrays con el perfil siguiente:

      `public static void ordFusión(int[] a, int ini, int fin)`

      Escribir las instrucciones necesarias para ordenar en dicha clase `PruebaLista` cierta `ListaPIIntEnla` almacenada en una variable `laLista`.

Solución:

```
public int[] toArray() {
    int[] aux = new int[talla];
    int k; NodoInt p;
    for (p=primero, k=0; p!=null; p=p.siguiente, k++) aux[k] = p.dato;
    return aux;
}

public ListaPIIntEnla(int[] a) {
    this.primero = null;
    for (int k=a.length-1; k>=0; k--) primero = new NodoInt(a[k],primero);
```

```
    this.talla = a.length;
    this.PI = primero; this.antPI = null;  // PI al inicio
}

// secuencia de instrucciones para ordenar laLista:
  int[] copia = laLista.toArray();
  ordFusion(copia,0,copia.length-1);
  laLista = new ListaPIIntEnla(copia);
```

4. 1.5 puntos Teniendo en cuenta la definición de `PilaIntEnla` vista en clase y accediendo a su representación interna, **se pide** escribir un método:

   `public void topeBase()`

   que intercambie el valor del elemento del tope de la pila con el de la base de la misma (el más antiguo). Será precondición del método que la pila no esté vacía.

Solución:

```
/** PRECONDICIÓN: !vacia() */
public void topeBase() {
    NodoInt aux = tope;
    int e = aux.dada;
    while(aux.siguiente!=null)
        aux = aux.siguiente;
    tope.dato = aux.dato;
    aux.dato = e;
}
```

5. 2.5 puntos Dadas dos listas con punto de interés `ListaPIIntEnla la` y `lb`, que están ordenadas ascendentemente y que no contienen elementos duplicados, **se pide** escribir un método `intersección` que devuelva una nueva `ListaPIIntEnla` que contenga únicamente los elementos que se encuentren tanto en `la` como en `lb`. Esta nueva lista tampoco deberá contener elementos duplicados.

   Si no existen elementos en común, la nueva `ListaPIIntEnla` estará vacía.

   Ejemplo: Si inicialmente se tiene que:

   ```
   la = 2 4 6 7 9 11 33 45 67 112 129 310 516 555 610
   lb = 8 11 22 33 44 45 46 112 113
   ```

   Como resultado del método pedido se tendrá la siguiente lista:

   ```
   11 33 45 112
   ```

Solución:

```
/** PRECONDICIÓN: la y lb ordenadas ascendentemente, sin repetidos */
public static ListaPIIntEnla intersección (ListaPIIntEnla la,
                                           ListaPIIntEnla lb) {
    ListaPIIntEnla lc = new ListaPIIntEnla();
    la.inicio(); lb.inicio();
    while (!la.fin() && !lb.fin()) {
        int a = la.recuperar(); int b = lb.recuperar();
        if (a<b) la.siguiente();
        else if (b<a) lb.siguiente();
        else {
            lc.insertar(a);
            la.siguiente(); lb.siguiente();
        }
    }
    return lc;
}
```