PRG - ETSInf. TEORÍA. Curso 2013-14. Parcial 2.
10 de junio de 2014. Duración: 2 horas.

1. 2 puntos Dado un fichero de texto, se quiere escribir su contenido línea a línea en la salida estándar, pero transformando sus caracteres alfabéticos a mayúsculas.

   Se pide escribir un método cuyo inicio sea el siguiente:

   ```
   public static void escribeConMayusculas(String nomF1) ...
   ```

   que escriba en la salida estándar, línea a línea, el contenido del fichero cuyo nombre está almacenado en `nomF1`, cambiando los caracteres alfabéticos en minúscula por sus correspondientes mayúsculas.

   En el caso de que `nomF1` no exista, se debe propagar una excepción del tipo `FileNotFoundException`.

   NOTA: Recuérdese que la operación `toUpperCase()` aplicada a una `String` devuelve una copia de la misma, en la que se han transformado a mayúsculas todos sus caracteres alfabéticos.

   Solución:

   ```
   public static void escribeConMayusculas(String nomF1) throws FileNotFoundException{
       Scanner sIn = new Scanner(new File(nomF1));
       while(sIn.hasNextLine())
           System.out.println(sIn.nextLine().toUpperCase());
       sIn.close();
   }
   ```

2. 1 punto Dada una `PilaIntEnla p`, escribir un método que muestre por la salida estándar el valor de la cima de la pila. Si la pila está vacía debe mostrar un mensaje de error. No se pueden utilizar los métodos `talla()` o `esVacia()` de `PilaIntEnla`. Se debe capturar la excepción `NoSuchElementException` que se produce en los métodos `cima()` o `desapilar()` cuando la pila está vacía.

   Solución:

   ```
   static void muestraCima(PilaIntEnla p){
       try{
           System.out.println(p.cima());
          }catch(NoSuchElementException ex){
           System.err.println("Error: Pila Vacía");
          }
      }
   ```

3. 2 puntos En una clase `ExamenEstructuras` se quiere añadir un método `incrementaPares` estático que dada una cola de enteros `c` implementada mediante una secuencia de nodos enlazados, devuelva una nueva cola de enteros en la que se habrán cambiado los números pares sumándoles uno, dejando el resto intactos. De manera que si la cola original es:

```
<- 5 10 14 13 22 31 <-
```

la cola a devolver debe contener los siguientes números (en cualquier orden):

```
<- 5 11 15 13 23 31 <-
```

NOTA: En la resolución de este problema no se podrán utilizar otras estructuras de datos adicionales.

a) (1.75 puntos) Implementa el método `incrementaPares`.

b) (0.25 puntos) Modifica el método anterior en caso de que la cola se implemente mediante un array.

Solución:

a) Una Solución posible (iterativa) entre muchas otras sería:

```
public static ColaIntEnla incrementaPares(ColaIntEnla q){
    ColaIntEnla q1 = new ColaIntEnla();
    int n = q.talla(), cont = 0;
    for (int i=0; i<n; i++){
        if (q.primero()%2==0) q1.encolar(q.desencolar()+1);
        else                  q1.encolar(q.desencolar());
    }
    return q1;
}
```

b) Bastaría cambiar en la cabecera y declaraciones de variables el nombre de la clase ColaIntEnla por ColaIntArray y usar también la constructora de ColaIntArray.

4. 3 puntos Se desea modificar el comportamiento de la operación `encolar(int)` en la clase `ColaIntEnla` de forma que sus elementos se mantengan de forma ordenada ascendente; esto es, el elemento **primero** de la cola será el de menor valor de entre todos, mientras que el **último** será el de valor mayor.

   Se pide escribir el método `encolar(int)` teniendo en cuenta la definición anterior.

Solución:

```
/** Encolar ordenadamente: versión primera. */
public void encolar(int val) {
    NodoInt q = null;
    NodoInt p = prim;

    while (p!= null && val > p.dato) { q = p; p = p.siguiente; }

    NodoInt nuevo = new NodoInt(val);
    if (q == null) {
        nuevo.siguiente = prim;
        prim = nuevo;
    } else {
        nuevo.siguiente = p;
```

```
            q.siguiente = nuevo;
        }

        if (p == null) ult = nuevo;
        talla++;
    }

    /** Encolar ordenadamente: versión segunda. */
    public void encolar2(int val) {
        NodoInt q = null;
        NodoInt p = prim;

        while (p!= null && val > p.dato) { q = p; p = p.siguiente; }

        NodoInt uevo;
        if (q == null) prim = nuevo = new NodoInt(val,prim);
        else q.siguiente = nuevo = new NodoInt(val,p);

        if (p == null) ult = nuevo;
        talla++;
    }
```

5. 2 puntos Se desea implementar un método llamado `contar` tal que:

   - Ha de recibir como argumentos una lista `l` perteneciente a la clase `ListaPIIntEnla`, y dos enteros `i`, `j`, que se considera que delimitan un rango de valores `[i,j]`.
   - Tiene que devolver el número de elementos en `l` que estén dentro del rango `[i,j]`.

   Se debe suponer que el método se está escribiendo en una clase diferente a `ListaPIIntEnla`.

Solución:

```
public static int contar(ListaPIIntEnla l,int i,int j) {
    int cont = 0;
    l.inicio();
    while ( !l.esFin() ) {
        int x = l.recuperar();
        if ( x>=i && x<=j ) cont++;
        l.siguiente();
    }
    return cont;
}
```