PRG - ETSInf. TEORÍA. Curso 2018-19. Parcial 2.
3 de junio de 2019. Duración: 2 horas.

Nota: El examen se evalúa sobre 10 puntos, pero su peso específico en la nota final de PRG es de **3 puntos**.

1. 3 puntos **Se pide:** escribir un método estático `void` de nombre `sumInt`, que reciba como parámetros un `String fileIn` con el nombre de un fichero de texto del sistema, y un `String fileOut`. Se supone que `fileIn` contiene una secuencia de enteros, y el método debe escribir en `fileOut`, línea a línea los valores leídos de `fileIn`, y al final la suma de todos los valores leídos. En el caso en que se produzca alguna excepción al leer un `int`, se debe escribir una línea con el formato `(Error:` *token incorrecto*`)`. Por ejemplo, si `fileIn` es un fichero de texto con los siguientes datos:

```
4  5
20  1  2x3  10
3
```

entonces el fichero de texto resultante debe contener

```
4
5
20
1
(Error: 2x3)
10
3
Suma: 43
```

Si alguno de los ficheros no se pudiese abrir, el método debe limitarse a propagar la excepción (comprobada) correspondiente.

Solución:

```
public static void sumInt(String fileIn, String fileOut) throws FileNotFoundException {
     File fI = new File(fileIn), fO = new File(fileOut);
     Scanner in = new Scanner(fI); PrintWriter out = new PrintWriter(fO);
     int sum = 0;
     while (in.hasNext()) {
         try {
             int n = in.nextInt();
             out.println(n);
             sum += n;
         } catch (InputMismatchException e) {
             out.println("(Error: " + in.next() + ")");
         }
     }
    out.println("Suma: " + sum);
    in.close(); out.close();
}
```

2. 3.5 puntos **Se pide:** añadir un método a la clase `QueueIntLinked` con perfil:

```
public void split(int x)
```

tal que, dado un entero `x`, busque la primera ocurrencia del elemento `x` en la cola y lo sustituya por el par de elementos `x / 2` y `x / 2 + x % 2`, uno a continuación del otro. Si `x` no aparece, la cola no debe cambiar.

Por ejemplo, si se invoca al método `q.split(9)` siendo `q` la cola de longitud 6 ← `1 -2 9 8 -3 5` ←, entonces `q` pasa a ser la cola de longitud 7 ← `1 -2 4 5 8 -3 5` ←.

IMPORTANTE: En la solución sólo se puede acceder a los atributos de la clase, quedando prohibido acceder a sus métodos.

Solución:

```
public void split(int x) {
     NodeInt aux = this.first;
     while (aux != null && aux.data != x) {
        aux = aux.next;
     }
     if (aux != null) {
         aux.data = x / 2;
         aux.next = new NodeInt( x / 2 + x % 2, aux.next);
         if (aux == this.last) { this.last = aux.next; }
         this.size++;
     }
}
```

3. 3.5 puntos **Se pide:** implementar un método estático `compress` tal que, dada una `ListPIIntLinked l` de la que se supone que sus elementos valen todos 0 o 1, devuelva otra lista de tamaño aproximadamente la mitad, tal que sus elementos representan a los de `l`, tomándolos de dos en dos y de izquierda a derecha. Así, en donde en `l` aparece una pareja de elementos seguidos `e1 e2`, en la nueva lista aparece `e1 * 2 + e2` (un valor `0`, `1`, `2`, o `3` para las parejas `00`, `01`, `10`, `11`, respectivamente). En el caso en que en `l` quedara al final un elemento `e` desemparejado, en la nueva lista aparecería al final `e - 2` (un valor `-1` o `-2`). Los elementos de la lista `l` no deben cambiar, aunque la posición del punto de interés sí que puede cambiar.

   Por ejemplo, si `l` es una lista con los elementos `0 0 0 1 1 0 1 1 0 0 1`, el método debe retornar otra lista con los elementos `0 1 2 3 0 -1`.

   IMPORTANTE: Se supondrá que el método se implementa en una clase diferente a `ListPIIntLinked`, por tanto, solo se podrán usar los métodos públicos de la clase.

Solución:

```
/** Precondición: los elementos de l valen 0 o 1. */
public static ListPIIntLinked compress(ListPIIntLinked l) {
    ListPIIntLinked result = new ListPIIntLinked();
    int n = l.size();
    l.begin();
    while (n >= 2) {
        int e1 = l.get(); l.next();
        int e2 = l.get(); l.next();
        result.insert(e1 * 2 + e2);
        n = n - 2;
    }
    if (n == 1) { result.insert(l.get() - 2); }
    return result;
}
```

ANEXO

Atributos de la clase `QueueIntLinked` y métodos de la clase `ListPIIntLinked`.

```
public class QueueIntLinked {
    private NodeInt first, last;
    private int size;
    ...


}
```

```
public class ListPIIntLinked {
    ...
    public ListPIIntLinked() { ... }
    public void begin() { ... }
    public void insert(int x) { ... }
    public int get() { ... }
    public int remove() { ... }
    public void next() { ... }
    public int size() { ... }
    public boolean empty() { ... }
    public boolean isEnd() { ... }
}
```