PRG - ETSInf. TEORÍA. Curso 2017-18. Parcial 2.
4 de junio de 2018. 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. 2.5 puntos **Se pide:** implementar un método estático tal que, dado un array de `int`, copie sus elementos, uno por línea, en un fichero de texto de nombre `"ArrayElements.txt"`. Así, si el array es {`5, 2, 8, 4`}, en el fichero se almacenarán, uno por línea, los valores `5`, `2`, `8` y `4`. El método debe devolver como resultado el objeto `File` creado.

   Deberá tratar la posible excepción `FileNotFoundException`, mostrando un mensaje de error en caso de que esta se produzca.

Solución:

```
public static File fromArrayToTextFile(int[] a) {
    File res = new File("ArrayElements.txt");
    PrintWriter pw = null;
    try {
        pw = new PrintWriter(res);
        for (int i = 0; i < a.length; i++) {
            pw.println(a[i]);
        }
    } catch (FileNotFoundException e) {
        System.err.println("Error al abrir " + res);
    } finally {
        if (pw != null) { pw.close(); }
    }
    return res;
}
```

2. 2.5 puntos **Se pide:** implementar un método estático tal que, dado un `String s`, lo convierta en una secuencia enlazada de caracteres donde el primer elemento de la secuencia debe ser el primer carácter del `String s`. Si el `String s` está vacío o es `null`, la secuencia resultante será también `null`; si no, devolverá el primer nodo de la secuencia. Para ello, se supone accesible una clase `NodeChar` idéntica a la clase `NodeInt` utilizada en el paquete `linear`, salvo en el tipo del dato.

   Por ejemplo, dado el `String s = "Examen"`, la secuencia será: → `'E'` → `'x'` → `'a'` → `'m'` → `'e'` → `'n'`

Solución:

```
    public static NodeChar fromStringToSeq(String s) {
        NodeChar res = null;
        if(s != null) {
           int i = s.length() - 1;
           while (i >= 0) {
               char c = s.charAt(i--);
               res = new NodeChar(c, res);
           }
        }
        return res;
    }
```

3. 2.5 puntos **Se pide:** añadir un método a la clase `ListPIIntLinked` con perfil:

```
public void append(int x)
```

tal que, dado un entero `x`, lo inserte en la posición siguiente al punto de interés, avanzando este al nuevo elemento introducido. Si el cursor ya está al final en el momento de invocar al método, este debe lanzar la excepción `NoSuchElementException` con el mensaje `"Cursor al final"`.

Por ejemplo, si se invoca al método `l.append(5)` siendo `l` una lista (donde el elemento distinguido es el marcado entre corchetes) `1 4 [7] 8 3 4`, entonces `l` queda de la forma `1 4 7 [5] 8 3 4`.

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

Solución:

```
public void append(int x) {
    if (pI != null) {
        prevPI = pI;
        pI.next = new NodeInt(x, pI.next);
        pI = pI.next;
        size++;
    } else { throw new NoSuchElementException("Cursor al final"); }
}
```

4. 2.5 puntos **Se pide:** implementar un método estático `fusion` tal que, dadas dos colas `QueueIntLinked q1` y `q2`, devuelva una nueva cola en la que se hayan fusionado los elementos de `q1` y `q2` de manera que aparezcan por orden de cola, pero alternándose un elemento de una y otra cola. Los elementos que sobraran de alguna de las colas aparecerán al final de la cola resultado. La colas `q1` y `q2` deben quedar en su estado inicial.

Por ejemplo, si `q1` es $\leftarrow$ `3 6 20 1 -3 4 -5` $\leftarrow$ y `q2` es $\leftarrow$ `10 9 8` $\leftarrow$ el método devuelve la cola $\leftarrow$ `3 10 6 9 20 8 1 -3 4 -5` $\leftarrow$.

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

Solución:

```
public static QueueIntLinked fusiona(QueueIntLinked q1, QueueIntLinked q2) {
    QueueIntLinked res = new QueueIntLinked();
    int i = Math.min(q1.size(), q2.size());
    for (int j = 0; j < i; j++) {
        res.add(q1.element()); q1.add(q1.remove());
        res.add(q2.element()); q2.add(q2.remove());
    }
    while (i < q1.size()) { res.add(q1.element()); q1.add(q1.remove()); i++; }
    while (i < q2.size()) { res.add(q2.element()); q2.add(q2.remove()); i++; }
    return res;
}
```

ANEXO

Métodos de la clase `QueueIntLinked`, atributos de la clase `ListPIIntLinked` y clase `NodeChar`.

```
public class QueueIntLinked {
    ...
    public QueueIntLinked() {...}
    public void add(int x) {...}
    public int remove() {...}
    public int element() {...}
    public int size() {...}
    public boolean empty() {...}
    public boolean equals(Object o) {...}
    public String toString() {...}
}
```

```
public class ListPIIntLinked {
    private int size;
    private NodeInt first, pI, prevPI;
}

class NodeChar {
    char data;
    NodeChar next;
    NodeChar(char c) { data = c;  next = null; }
    NodeChar(char c, NodeChar n) {
        data = c; next = n;
    }
}
```