PRG - ETSInf. TEORÍA. Curso 2017-18. Recuperación Parcial 2.
19 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 que recupere los datos enteros de un fichero de texto, cuyo nombre se proporcione como parámetro, y los almacene en una `StackIntLinked`, que será el resultado, de manera que el primer dato entero que se encuentre en el fichero debe quedar en la base de la pila.

   Se debe tener en cuenta lo siguiente:

   - El fichero puede contener datos que no sean enteros, por ello la excepción `InputMismatchException` se debe capturar, y avanzar la línea cuando se produzca.
   - Si el fichero no contiene enteros, o si no existe, se devolverá una pila vacía.
   - En caso de que el fichero no exista, la excepción correspondiente debe tratarse localmente escribiendo por la salida estándar el mensaje `"No se encontró el fichero"`.
   - En cualquier caso, si se ha creado correctamente el scanner de lectura, este deberá cerrarse.

Solución:

```
public static StackIntLinked fileToStack(String name) {
    Scanner s = null;
    StackIntLinked stack = new StackIntLinked();
    try {
        s = new Scanner(new File(name));
        while (s.hasNextLine()) {
            try {
                stack.push(s.nextInt());
            } catch (InputMismatchException n) {
                s.nextLine();
            }
        }
    } catch (FileNotFoundException e) {
        System.out.println("No se encontró el fichero");
    } finally {
        if (s != null) { s.close(); }
    }
    return stack;
}
```

2. 2.5 puntos **Se pide:** implementar un método estático iterativo que copie los elementos de `l`, una lista `ListPIIntLinked` pasada como parámetro, en una nueva lista, resultado a devolver, cuyos elementos estén en el mismo orden, pero restando a todos los elementos el valor mínimo de la lista `l`. Por ejemplo, si la lista original `l` es `[12] 50 10 120` , el mínimo es `10` y, por tanto, se debe obtener la lista `2 40 0 110 [ ]` .

   Además, se debe tener en cuenta lo siguiente:

   - Si la lista `l` está vacía, se debe devolver `null`.
   - En cualquier caso, el contenido inicial de la lista `l` debe conservarse, a excepción de la posición de su punto de interés (la cual se permite modificar).
   - **REQUISITO:** El método pedido es de una clase diferente a `ListPIIntLinked`. Por ello, su implementación debe hacerse usando los métodos públicos de `ListPIIntLinked` exclusivamente.

Solución:

```
public static ListPIIntLinked subtractMinimumToList(ListPIIntLinked l) {
    if (l.empty()) { return null; }
    ListPIIntLinked res = new ListPIIntLinked();
    l.begin();
    int min = l.get();
    while (!l.isEnd()) {
        if (l.get() < min) { min = l.get(); }
        l.next();
    }
    for (l.begin(); !l.isEnd(); l.next()) {
        res.insert(l.get() - min);
    }
    return res;
}
```

3. 2.5 puntos **Se pide:** implementar un método estático que reciba una secuencia enlazada `NodeInt` y devuelva otra secuencia enlazada `NodeInt` que contenga solamente los datos pares de la secuencia recibida. Por ejemplo, si la secuencia recibida contiene los siguientes datos: `4 7 2 8 9 3 6` , se debe devolver la siguiente secuencia: `4 2 8 6` . Se ha de tener en cuenta que si la secuencia recibida es `null` o si no contiene datos pares, se debe devolver `null`.

Solución:

```
public static NodeInt evenSubsequence(NodeInt seq) {
    NodeInt first = null;
    NodeInt last = null;
    while (seq != null) {
        if (seq.data % 2 == 0) {
            if (first == null) {
                last = new NodeInt(seq.data);
                first = last;
            }
            else {
                last.next = new NodeInt(seq.data);
                last = last.next;
            }
        }
        seq = seq.next;
    }
    return first;
}
```

4. 2.5 puntos **Se pide:** implementar en la clase `QueueIntLinked` un método de instancia que divida una cola en dos mitades, con perfil: `public QueueIntLinked divideQueue()`
Teniendo en cuenta lo siguiente:
   - Precondición: la cola inicial (`this`) tiene al menos dos elementos.
   - La división se realiza de forma que la cola inicial se queda con la primera mitad de los elementos y se devuelve una cola con el resto de los elementos.
   - Las colas resultantes mantienen el orden de los elementos en la cola inicial.
   - Si la cola inicial tiene un cantidad impar de elementos, será la cola devuelta la que tendrá una longitud superior en una unidad.

Ejemplos:

Cola inicial	Cola inicial modificada	Cola devuelta
1 2 2 4 3 1	1 2 2	4 3 1
1 2 2 4 3 1 1	1 2 2	4 3 1 1

REQUISITO: En el método pedido debe usarse exclusivamente los atributos de `QueueIntLinked`, su constructor, y referencias a `NodeInt`. Por tanto, NO se permite ninguna invocación a los métodos de la clase.

Solución:

```
    public QueueIntLinked divideQueue() {
        QueueIntLinked nq = new QueueIntLinked();
        int middle = size / 2;
        NodeInt aux = this.first;
        for (int i = 0; i < middle - 1; i++) {
            aux = aux.next;
        }
        nq.first = aux.next;
        nq.last = this.last;
        aux.next = null;
        this.last = aux;
        nq.size = this.size - middle;
        this.size = middle;
        return nq;
    }
```

ANEXO

Métodos de las clases `StackIntLinked` y `ListPIIntLinked`, y atributos de la clase `QueueIntLinked`.

```
public class StackIntLinked {
        ...
    public StackIntLinked() { ... }
    public boolean empty() { ... }
    public int size() { ... }
    public void push(int x) { ... }
    public int pop() { ... }
    public int peek() { ... }
    public boolean equals(Object o) { ... }
    public String toString() { ... }
}

public class ListPIIntLinked {
    ...
    public ListPIIntLinked() { ... }
    public boolean empty() { ... }
    public int size() { ... }
    public boolean isEnd() { ... }
    public void begin() { ... }
    public void next() { ... }
    public void insert(int x) { ... }
    public int remove() { ... }
    public int get() { ... }
    public boolean equals(Object o) { ... }
    public String toString() { ... }
}

public class QueueIntLinked {
    private NodeInt first;
    private NodeInt last;
    private int size;
    public QueueIntLinked() { }
    ...
}
```