PRG - GIINF+GADE.
TEORÍA. Curso 2017-18. Parcial 2.
28 de mayo de 2018. Duración: 2 horas.

1. 2.5 puntos Dentro de una clase cualquiera (y distinta de `StackInt`), implementa un método estático **recursivo** que calcule el valor máximo de una pila de números enteros (`StackInt p`) recibida como parámetro del método. En caso de que `p` fuera una pila vacía, entonces el método debe lanzar la excepción `NoSuchElementException`. Evidentemente, la pila `p` no debe ser modificada por el método, quedando intacta a la finalización del mismo.

Solución:

```
public static int maximo(StackInt p) {
    if (p.isEmpty()) { throw new NoSuchElementException(); }
    else {
        if (p.size() == 1) { return p.peek(); }
        else {
            int x = p.pop();
            int y = maximo(p);
            p.push(x);
            if (x > y) { return x; }
            else       { return y; }
        }
    }
}
```

2. 2.5 puntos Dada una secuencia de nodos encabezada por `NodeInt sec`, implementa un método de perfil

```
public static void deleteOddPositions(NodeInt sec)
```

tal que elimine de la secuencia todos los nodos que ocupan actualmente las posiciones impares de la misma. **Nota:** Recuerda que por convención el primer nodo de una secuencia ocupa la posición 0 (0 es un número par).

Por ejemplo, dada la secuencia de entrada siguiente: $\rightarrow$ 2 $\rightarrow$ 4 $\rightarrow$ 6 $\rightarrow$ 8 $\rightarrow$ `null`
tras ejecutar el método, la secuencia quedaría como: $\rightarrow$ 2 $\rightarrow$ 6 $\rightarrow$ `null`

Solución:

```
public static void deleteEvenPositions(NodeInt sec) {
    NodeInt ant = null;
    NodeInt aux = sec;
    int k = 0;
    while (aux != null) {
        if (k % 2 == 1) {
            ant.next = aux.next;
            aux = aux.next;
        }
        else {
            ant = aux;
            aux = aux.next;
        }
        k++;
    }
}
```

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

```
public void overwrite(int x)
```

- tal que sobreescriba el dato al que apunta el punto de interés como si se tratara de una inserción, es decir, el punto de interés y su antecesor se deben desplazar, respectivamente, al siguiente elemento de la lista.
- En caso de que el punto de interés se encontrara al final del todo, su comportamiento es como el de una inserción normal.

Nota: Sólo se permite acceder a los atributos de la clase, quedando terminantemente prohibido el acceso a sus métodos, así como a cualquier otra estructura de datos auxiliar (incluyendo el uso de arrays).

Solución:

```
    public void overwrite(int x) {
        if (PI != null) {
            PI.data = x;
            prevPI = PI;
            PI = PI.next;
        }
        else {
            NodeInt nuevo = new NodeInt(x);
            if (first == null) { first = nuevo; }
            else         { prevPI.next = nuevo; }
            prevPI = nuevo;
            size++;
        }
    }
```

4. 2.5 puntos Dado un archivo de texto `filename`, que contiene varios numéros por línea (enteros y reales), implementa un método estático en una clase arbitraria con el siguiente perfil:

```
public static QueueInt file2Queue(String filename)
```

tal que construya una cola de números enteros con los datos de tipo entero almacenados en el fichero de texto. El método debe gestionar la posible `InputMismatchException` generada al intentar leer un número entero, recuperándose de esa posible situación anómala, y continuar leyendo sin problemas el resto de datos del fichero. Si durante el *opening* del archivo saltara la excepción `FileNotFoundException`, el método tiene que propagarla.

Solución:

```
public static QueueIntLinked file2Queue(String filename) throws FileNotFoundException {
    Scanner s = new Scanner(new File(filename));
    QueueIntLinked q = new QueueIntLinked();
    while (s.hasNext()) {
        try {
            int n = s.nextInt();
            q.add(n);
        }
        catch (InputMismatchException e) { s.nextDouble(); }
    }
    s.close();
    return q;
}
```