PRG - ETSInf. TEORÍA. Curso 2015-16. Parcial 2.
31 de mayo de 2016. Duración: 2 horas.

1. 1.5 puntos Se dispone de un array `lS` de objetos de tipo `String`, tal que no tiene inicializados todos sus elementos (esto es, alguna de las posiciones del array contienen un valor `null` en lugar de una `String` bien formada).

   Si se utiliza el método siguiente para imprimir la longitud de todas las `String` realmente existentes de `lS`:

   ```
   public static void m1(String[] lS) {
       int k = 0;
       boolean fin = false;
       while (!fin) {
           System.out.print("Posicion " + k + ": ");
           System.out.println(lS[k].length() + " caracteres");
           k++;
       }
   }
   ```

   se pueden producir las excepciones: `NullPointerException` y `ArrayIndexOutOfBoundsException`. Cuando, en realidad, se desearía una salida **sin excepciones** como la que se muestra (el ejemplo se efectúa con un array de 6 elementos):

   ```
   Posición 0: 4 caracteres
   Posición 1: 9 caracteres
   Posición 2: String no inicializada
   Posición 3: 11 caracteres
   Posición 4: String no inicializada
   Posición 5: 0 caracteres
   Posición 6: Inexistente. Fin del array
   ```

   Se pide: Reescribir el método `m1` para que, **tratando exclusivamente** las dos excepciones indicadas (sin hacer uso del atributo `length` del array ni de la constante `null`), resuelva el problema pedido efectuando una salida como la mostrada en el ejemplo.

Solución:

A continuación se muestran dos soluciones alternativas:

```
// Primera solución:
public static void m1(String[] lS) {
    int k = 0;
    boolean fin = false;
    try {
        while (!fin) {
            System.out.print("Posición " + k + ": ");
            try {
                System.out.println(lS[k].length() + " caracteres");
            } catch (NullPointerException np) {
                System.out.println("String no inicializada");
            }
            k++;
        }
    } catch (ArrayIndexOutOfBoundsException io) {
        System.out.println("Inexistente. Fin del array");
    }
}
```

```
// Segunda solución:
public static void m1(String[] lS) {
    int k = 0;
    boolean fin = false;
    while (!fin) {
        System.out.print("Posición " + k + ": ");
        try {
            System.out.println(lS[k].length() + " caracteres");
        } catch (NullPointerException np) {
            System.out.println("String no inicializada");
        } catch (ArrayIndexOutOfBoundsException io) {
            System.out.println("Inexistente. Fin del array");
            fin = true;
        }
        k++;
    }
}
```

2. 2.5 puntos **Se pide** implementar un método estático tal que:
   - Reciba como argumento una `String` que contiene la ruta y nombre de un fichero de texto.
   - Propague la excepción `FileNotFoundException` si no fuera posible abrir el fichero cuyo nombre se ha recibido.
   - Construya y devuelva un objeto `ListaPIIntEnla` con todos los números enteros contenidos en el fichero.
   - El fichero de texto recibido puede contener tokens que sean representaciones válidas de números enteros y tokens que no lo sean. Se desconoce cuántos tokens hay en el fichero.
   - Si el token leído es una representación válida de un número entero, dicho número se debe insertar en la lista a devolver.
   - Si el token leído no es una representación válida de un número entero, entonces se debe capturar la excepción `InputMismatchException` que se genera en tal caso, mostrando en la consola de error un mensaje que incluya el nombre de la excepción y el valor del token que la ha generado. Esta circunstancia no debe impedir que continúe la lectura del fichero.

Solución:

```
public static ListaPIIntEnla leer(String f) throws FileNotFoundException {
    ListaPIIntEnla li = new ListaPIIntEnla();
    Scanner sc = new Scanner(new File(f));
    while (sc.hasNext()) {
        try {
            li.insertar(sc.nextInt());
        } catch (InputMismatchException e) {
            System.err.println(e + "::" + sc.next());
        }
    }
    sc.close();
    return li;
}
```

3. 3 puntos En una cola, a veces puede ser de interés sacar algún elemento `x` indeseado. Por este motivo, **se pide** añadir a la clase `ColaIntEnla` un nuevo método con perfil:

```
public int desencolar(int x)
```

que saque de la cola la primera ocurrencia de `x`, y lo devuelva. En el caso en que la cola esté vacía debe lanzar la excepción `NoSuchElementException` con el mensaje `Cola vacia`, y si `x` no se encuentra debe lanzar una excepción del mismo tipo con el mensaje `x no esta en la cola`.

Nota: en la solución no se podrán usar los métodos de la clase.

Solución:

```
    public int desencolar(int x) {
        if (this.talla == 0) { throw new NoSuchElementException("Cola vacia"); }
        NodoInt ant = null, aux = this.primero;
        while (aux != null && aux.dato != x) {
            ant = aux; aux = aux.siguiente;
        }
        if (aux != null) {
            if (this.primero == aux) { this.primero = aux.siguiente; }
            else { ant.siguiente = aux.siguiente; }
            if (aux == this.ultimo) { this.ultimo = ant; }
            this.talla--;
            return x;
        } else { throw new NoSuchElementException(x + " no esta en la cola"); }
    }
```

4. 3 puntos En una clase distinta a `ListaPIIntEnla`, **se pide** implementar un método con el siguiente perfil y precondición:

```
/** Precondición: lista1 y lista2 no contienen elementos repetidos. */
public static ListaPIIntEnla eliminComunes(ListaPIIntEnla lista1, ListaPIIntEnla lista2)
```

que devuelva una lista con los elementos comunes de ambas listas, eliminando de `lista1` dichos elementos.

Ejemplo:
Sea una `lista1` con los valores `6 -5 4 8 -9`,
sea una `lista2` con los valores `21 8 5 -9 -5 16`,
entonces el resultado de `eliminComunes(lista1, lista2)` debe ser una lista con los elementos `-5 8 -9`, y debe dejar `lista1` con los elementos `6 4`.

Solución:

```
/** Precondición: lista1 y lista2 no contienen elementos repetidos. */
public static ListaPIIntEnla eliminComunes(ListaPIIntEnla lista1, ListaPIIntEnla lista2) {
    ListaPIIntEnla result = new ListaPIIntEnla();
    lista1.inicio();
    while (!lista1.esFin()) {
        int x = lista1.recuperar();
        lista2.inicio();
        while (!lista2.esFin() && x != lista2.recuperar()) { lista2.siguiente(); }
        if (lista2.esFin()) { lista1.siguiente(); }
        else { lista1.eliminar(); result.insertar(x); }
    }
    return result;
}
```