PRG - ETSInf. TEORÍA. Curso 2015-16. Recuperación Parcial 2.
GII. GIINF - GADE.
17 de junio de 2016. Duración: 2 horas.

1. 1.5 puntos Se dispone de un array `lS` de objetos de tipo `String`, que representan valores en coma flotante.

   Si el array está correctamente formado, esto es, si cada uno de sus elementos es una `String` que contiene la representación de un `double` en Java, entonces, el siguiente código escribe correctamente el contenido del array:

```
public static void m1(String[] lS) {
    for (int i = 0; i < lS.length; i++) {
        System.out.print("Pos: " + i + ": ");
        if (lS[i].length() > 0) {
            double valor = Double.parseDouble(lS[i]);
            System.out.println("Valor: " + valor);
        }
        else { System.out.println("String de longitud cero."); }
    }
}
```

   Sin embargo, si alguna de las `Strings` del array no existe, o contiene un valor que no representa un `double`, se podrán producir, respectivamente, las excepciones: `NullPointerException` o `NumberFormatException`.

   En ese caso, en realidad, se desearía una salida **sin excepciones**. Por ejemplo, como la que se muestra a continuación, para el array: {`"1234.0"`, `"1.23456789E8"`, `null`, `"123xx9"`, `null`, `""`}.

```
Pos: 0: Valor: 1234.0
Pos: 1: Valor: 1.23456789E8
Pos: 2: String inexistente.
Pos: 3: Número mal formado.
Pos: 4: String inexistente.
Pos: 5: String de longitud cero.
```

   Se pide: reescribir el método `m1` para que, **tratando exclusivamente** las dos excepciones indicadas resuelva el problema efectuando una salida como la mostrada en el ejemplo.

   Solución:

```
public static void m1(String[] lS) {
    for (int i = 0; i < lS.length; i++) {
        System.out.print("Pos: " + i + ": ");
        try {
            if (lS[i].length() > 0) {
                double valor = Double.parseDouble(lS[i]);
                System.out.println("Valor: " + valor);
            }
            else { System.out.println("String de longitud cero."); }
        } catch(NullPointerException nP) {
            System.out.println("String inexistente.");
        } catch(NumberFormatException nF) {
            System.out.println("Número mal formado.");
        }
    }
}
```

2. 2.5 puntos **Se pide:** implementar un método estático tal que dada una `PilaIntEnla p` copie sus elementos uno por línea en un fichero de texto de nombre `"ContenidoDePila.txt"` en el orden en que fueron apilados.

Al finalizar la ejecución del método, la pila `p` debe quedar como estaba. Así, si tenemos la pila $\rightleftarrows$ 1 2 3 4 donde la cima se situa en el 1, en el fichero deberán guardarse, uno por línea, los valores 4 3 2 1. El método debe devolver como resultado el objeto `File` creado. Deberá tratarse la posible excepción `FileNotFoundException`, de modo que se muestre un mensaje de error en caso de que ésta se produzca.

Solución:

```
public static File pilaIntEnlaToTextFile(PilaIntEnla p) {
    PilaIntEnla aux = new PilaIntEnla();
    File res = new File("ContenidoDePila.txt");
    try {
        PrintWriter pw = new PrintWriter(res);
        while (!p.esVacia()) { aux.apilar(p.desapilar()); }
        while (!aux.esVacia()) {
            p.apilar(aux.desapilar());
            pw.println(p.cima());
        }
        pw.close();
    } catch (FileNotFoundException e) {
        System.out.println("No se puede crear el fichero");
    }
    return res;
}
```

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

```
public void recular(int x)
```

tal que:

- Busque la primera ocurrencia del elemento `x` dentro de la cola y, en caso de éxito en la búsqueda, haga que dicho elemento se traslade al final del todo y, por tanto, se quede como el último de la cola.
- En caso de fracaso en la búsqueda, la cola se queda como estaba.

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:

```
/** Si x está en la cola, lo pone el último de la cola. */
public void recular(int x) {
    NodoInt aux = primero, ant = null;
    while (aux != null && aux.dato != x) {
        ant = aux;
        aux = aux.siguiente;
    }

    if (aux != null && aux != ultimo) {
        if (aux == primero) { primero = primero.siguiente; }
        else { ant.siguiente = aux.siguiente; }

        ultimo.siguiente = aux;
        aux.siguiente = null;

        ultimo = aux;
    }
}
```

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 diferencia(ListaPIIntEnla lista1, ListaPIIntEnla lista2)
```

que devuelva una lista con los elementos de `lista1` que no están en `lista2`.

Por ejemplo, dadas la `lista1` $\rightarrow 7 \rightarrow 3 \rightarrow 9 \rightarrow 6 \rightarrow 2$ y la `lista2` $\rightarrow 8 \rightarrow 9 \rightarrow 5 \rightarrow 3 \rightarrow 2 \rightarrow 4$, entonces el resultado de `diferencia(lista1, lista2)` debe ser una lista con los elementos $\rightarrow 7 \rightarrow 6$.

Solución:

```
/** Precondición: lista1 y lista2 no contienen elementos repetidos. */
public static ListaPIIntEnla diferencia(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()) { result.insertar(x); }
        lista1.siguiente();
    }
    return result;
}
```