# PRG - ETSInf. TEORIA. Curs 2015-16. Recuperació Parcial 2.
# 17 de juny de 2016. Duració: 2 hores.

1. [1.5 punts] Es disposa d'un array `lS` d'objectes de tipus `String`, que representen valors en coma flotant.

   Si l'array està correctament format, açò és, si cadascun dels seus elements és una `String` que conté la representació d'un `double` en Java, aleshores, el següent codi escriu correctament el contingut de l'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 zero."); }
    }
}
```

   No obstant això, si alguna de les `String` de l'array no existeix, o conté un valor que no representa un `double`, es podran produir, respectivament, les excepcions: `NullPointerException` o `NumberFormatException`.

   En aquest cas, en realitat, es desitjaria una sortida **sense excepcions**. Per exemple, com la que es mostra a continuació, per a l'array: {`"1234.0"`, `"1.23456789E8"`, `null`, `"123xx9"`, `null`, `""`}.

```
Pos: 0: Valor: 1234.0
Pos: 1: Valor: 1.23456789E8
Pos: 2: String inexistent.
Pos: 3: Nombre mal format.
Pos: 4: String inexistent.
Pos: 5: String de longitud zero.
```

   **Es demana:** reescriure el mètode `m1` perquè, **tractant exclusivament** les dues excepcions indicades resolga el problema efectuant una sortida com la mostrada en l'exemple.

   **Solució:**

```
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 zero."); }
        } catch(NullPointerException nP) {
            System.out.println("String inexistent.");
        } catch(NumberFormatException nF) {
            System.out.println("Nombre mal format.");
        }
    }
}
```

2. [2.5 punts] **Es demana:** implementar un mètode estàtic tal que donada una `PilaIntEnla p` copie els seus elements un per línia en un fitxer de text de nom `"ContingutDePila.txt"` en l'ordre en què van ser apilats. En finalitzar l'execució del mètode, la pila `p` ha de quedar com estava. Així, si tenim la pila $\rightleftarrows$ 1 2 3 4 on el

cim es troba a l'1, al fitxer s'han de guardar, un per línia, els valors 4 3 2 1. El mètode ha de retornar com a resultat l'objecte `File` creat. Haurà de tractar la possible excepció `FileNotFoundException`, de manera que es mostre un missatge d'error en cas que aquesta es produisca.

**Solució:**

```
public static File pilaIntEnlaToTextFile(PilaIntEnla p) {
    PilaIntEnla aux = new PilaIntEnla();
    File res = new File("ContingutDePila.txt");
    try {
        PrintWriter pw = new PrintWriter(res);
        while (!p.esBuida()) { aux.empilar(p.desempilar()); }
        while (!aux.esBuida()) {
            p.empilar(aux.desempilar());
            pw.println(p.cim());
        }
        pw.close();
    } catch (FileNotFoundException e) {
        System.out.println("No es pot crear el fitxer");
    }
    return res;
}
```

3. [3 punts] **Es demana:** afegir a la classe `CuaIntEnla` un mètode de perfil

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

tal que:

- Busque la primera ocurrència de l'element `x` dins de la cua i, en cas d'èxit en la cerca, faça que aquest element es trasllade al final del tot i, per tant, es quede com l'últim de la cua.
- En cas de fracàs en la cerca, la cua es queda com estava.

**Nota:** Només es permet accedir als atributs de la classe, quedant totalment prohibit l'accés als seus mètodes, així com a qualsevol altra estructura de dades auxiliar (incloent l'ús d'arrays).

**Solució:**

```
/** Si x està en la cua, el fica l'últim de la cua. */
public void recular(int x) {
    NodeInt aux = primer, ant = null;
    while (aux != null && aux.dada != x) {
        ant = aux;
        aux = aux.seguent;
    }

    if (aux != null && aux != ultim) {
        if (aux == primer) { primer = primer.seguent; }
        else { ant.seguent = aux.seguent; }

        ultim.seguent = aux;
        aux.seguent = null;

        ultim = aux;
    }
}
```

4. [3 punts] En una classe distinta a `LlistaPIIntEnla`, **es demana:** implementar un mètode amb el següent perfil i precondició:

```
/** Precondició: llista1 i llista2 no contenen elements repetits. */
public static LlistaPIIntEnla diferencia(LlistaPIIntEnla llista1, LlistaPIIntEnla llista2)
```

que torne una llista amb els elements de `llista1` que no estan en `llista2`.

Per exemple, donades la `llista1` $\rightarrow 7 \rightarrow 3 \rightarrow 9 \rightarrow 6 \rightarrow 2$ i la `llista2` $\rightarrow 8 \rightarrow 9 \rightarrow 5 \rightarrow 3 \rightarrow 2 \rightarrow 4$, aleshores el resultat de `diferencia(llista1, llista2)` ha de ser una llista amb els elements $\rightarrow 7 \rightarrow 6$.

**Solució:**

```
/** Precondició: llista1 i llista2 no contenen elements repetits. */
public static LlistaPIIntEnla diferencia(LlistaPIIntEnla llista1, LlistaPIIntEnla llista2) {
    LlistaPIIntEnla result = new LlistaPIIntEnla();
    llista1.inici();
    while (!llista1.esFi()) {
        int x = llista1.recuperar();
        llista2.inici();
        while (!llista2.esFi() && x != llista2.recuperar()) { llista2.seguent(); }
        if (llista2.esFi()) { result.inserir(x); }
        llista1.seguent();
    }
    return result;
}
```