Final CAP

Curs 2021-22 (20/1/2022) **Duració: 3 hores**

1.- (2 punts) Escriviu un programa que, passant el nom d'una classe com a paràmetre de la línia de comandes, digui si algun dels seus mètodes (propis o heretats, sense importar si són **`public`**, **`protected`** o **`private`**) és capaç de generar (*throw*) més d'un tipus diferent d'excepció.

Solució:

```
import java.lang.reflect.*;

public class Pregunta_1 {

    public static void main(String args[]) throws Exception {
        if (pregunta_1(Class.forName(args[0]))) {
            System.out.println("SI, n'hi ha algun");
        } else {
            System.out.println("NO, no n'hi ha cap");
        }
    }

    private static boolean pregunta_1 (Class c) {
        if (c != null) {
            try {
                Method m[] = c.getDeclaredMethods();
                for (int i = 0; i < m.length; i++) {
                    Class exc[] = m[i].getExceptionTypes();
                    if (exc.length > 1) {
                           return true;
                    }
                }
                return pregunta_1(c.getSuperclass());
            }
            catch (Throwable e) {
                System.err.println(e);
            }
        }
        return false;
    }
}
```

2.- (2 punts) Executa aquest codi i **justifica** el resultat, és a dir, el valor de la variable **`result`** (*la pregunta val 0 punts si no es justifica*):

```
function misteri(n){
  let secret = 4;
  n += 2;
  function misteri2(mult) {
    mult *= n;
    return secret * mult;
  }
  return misteri2;
}

function misteri3(param){
```

```
  function misteri4(bonus){
    return param(6) + bonus;
  }
  return misteri4;
}

let h = misteri(3);
let j = misteri3(h);
let result = j(2);
```

Solució:
En executar aquest codi obtenim el resultat **122**. Les funcions en Javascript són *closures*, per tant capturen el context lèxic en el moment de la seva creació. Això és fonamental per entendre aquest exercici.
Aleshores: primer cridem a **misteri**, que retorna la funció **misteri2** amb variables capturades **n** (que val 5) i **secret** (que val 4). Aquesta funció és assignada a **h**.
Ara cridem **misteri3**, que retorna **misteri4** i l'assigna a **j**. **misteri4** captura **param** (que val **misteri2**, recordem que estem treballant amb funcions d'ordre superior).
Finalment cridem **j(2)**, és a dir, **misteri4(2)**, que retorna **param(6)+2**. **param**, és a dir, **misteri2**, es crida amb paràmetre 6, i executa **6*n = 6*5 = 30** (valor de **mult**) i retorna **secret * 30 = 4 * 30 = 120**. Així, el valor assignat a **result** serà **120+2 = 122**.

3.- (2 punts) Tenim la funció recursiva **my_reduce(f,array,v)**, on **f** és una funció amb dos paràmetres **f ≡ f(x,total)**, de manera que

my_reduce(f,[x_1,x_2,x_3,...,x_N],x_0) = f(x_1,f(x_2,f(x_3,...f(x_N,x_0)...)))

La seva implementació és:

```
function my_reduce(f,arr,v) {
    if (arr.length === 0) {
        return v
    } else {
       let [car, ...cdr] = arr
       return f(car,my_reduce(f,cdr,v))
    }
}
```

Aleshores: **my_reduce((x,y) => x*y,[1,2,3,4,5,6,7,8,9],1)** retorna **362880** (que és **9!**) o **my_reduce((x,y) => x+y,['h','o','l','a'],'')** retorna **'hola'**. Trobeu una funció equivalent recursiva final. Utilitzeu la transformació a *Continuation Passing Style* (CPS).

Solució: Aplicant CPS, obtenim la funció equivalent:

```
function my_reduce_CPS(f,arr,v,cont) {
    if (arr.length === 0) {
        return cont(v)
    } else {
       let [car, ...cdr] = arr
       return my_reduce_CPS(f,cdr,v,function(r) {
                                        return cont(f(car,r))
                                    })
    }
}
```

4.- (2 punts) Tenim una funció recursiva *final* **`my_filter_index(arr,f,res,i)`**, que aplica un predicat **`f`** a tots els elements d'un *array*, i retorna un altre *array* amb ***l'índex*** d'aquells elements **`e`** tal que **`f(e)`** és **`true`**. Si **`arr`** és l'*array* al que volem aplicar **`f`**, s'ha de cridar **`my_filter_index(arr,f,[],0)`**:

```
function my_filter_index (arr, f, res, i) {
    if (arr.length === 0) {
        return res
    } else {
        let [car, ...cdr] = arr
        if (f(car)) {
           res.push(i)
        }
        return my_filter_index(cdr,f,res,i+1)
    }
}
```

Aleshores: **`my_filter_index([1,2,3,4,5,6,7,8,9],x => x%2 === 0,[],0)`** retorna **`[1,3,5,7]`**

Sabem que si fem servir Node.js, que *no* fa TCO, tindrem problemes amb la pila. Si fem , per exemple, **`my_filter_index([...Array(10000).keys()], x => x % 2 === 0, [], 0),`** obtindrem un error **`RangeError: Maximum call stack size exceeded`**. Apliqueu la técnica del *trampolining* per obtenir una versió de **`my_filter_index`** que no tingui problemes amb la mida de la pila.

Solució: Apliquem l'esquema general que us vaig passar quan vam explicar el *trampolining*:

```
const my_filter_index_tramp = (function () {
   function __f(a, f, r, i) {
       if (a.length > 0) {
           return function() {
                    let [car, ...cdr] = a
                    if (f(car)) {
                       r.push(i)
                    }
                    return __f(cdr,f,r,i+1)
                };
       };
       return r
    };
    return function (arr, f, res, i) {
        return trampoline(__f(arr,f,res,i))
    }
})()
```

suposant que tenim la funció que ja coneixem:

```
function trampoline (fun) {
    while (typeof fun == 'function') {
        fun = fun();
    }
    return fun;
}
```

Així, si fem **`my_filter_index_tramp([...Array(10000).keys()], x => x+1, [])`** obtenim el resultat esperat sense problemes.

5.- (2 punts) Suposem que tenim tres funcions constructores **A**, **B** i **C**. Volem que els objectes construïts per la funció **C** puguin utilitzar les funcionalitats que proporcionen les funcions constructores **A** i **B** (en un món OO amb classes i herència simple, diriem que **C** és una subclasse de **B** i que **B** és una subclasse d'**A**). Per exemple, si els objectes creats amb **A** tenen una propietat anomenada **propA** (de contingut inicial **"a"**), els objectes creats per **B** tenen una propietat anomenada **propB** (de contingut inicial **"b"**) i els objectes creats amb **C** tenen una propietat anomenada **propC** (de contingut inicial **"c"**), el resultat d'executar:

```
let c = new C();
console.log(c.propA);
console.log(c.propB);
console.log(c.propC);
```

seria

```
a
b
c
```

Dóna una possible definició per a **A**, **B** i **C** per a que es verifiquin les condicions de l'enunciat.

Solució: Una solució seria:

```
function A() {
      // . . . el que sigui
}
A.prototype.propA = "a";

function B() {
      // . . . el que sigui
}
B.prototype = Object.create(A.prototype);
B.prototype.constructor = B;
B.prototype.propB = "b";

function C() {
      // . . . el que sigui
}
C.prototype = Object.create(B.prototype);
C.prototype.constructor = C;
C.prototype.propC = "c";
```