TALLER No. 11	PROGRAMACIÒN ORIENTADA A OBJETOS

Profesor: Jaime Alberto Guzmán Luna

Contenido del taller:

1. try, catch, finally (try, except, finally)
2. throw y throws (raise)
3. Herencia de excepciones

1. Ejercicio de clase

```
1  public class ObjTaller11A {
2
3        public static void main(String[] args) {
4              try {
5                    int[] v = new int[] {4, 5};
6                    System.out.println(v[0]);
7
8                    String s = null;
9                    System.out.println(s.length());
10
11             } catch (IndexOutOfBoundsException e) {
12                   System.out.println("Excepcion por indices fuera de
   rango");
13             } catch (NullPointerException e) {
14                   System.out.println("Puntero nulo");
15             } finally {
16                   System.out.println("Lo ultimo");
17             }
18             System.out.println("Sigo normal");
19       }
20 }
```

Responda:

A. ¿Qué imprime el programa? ¿Qué tipo de error se genera y por qué?
B. ¿Qué sucede si en la línea 6 trata de acceder a la posición 2 del arreglo? ¿Qué imprime ahora? ¿Por qué?
C. ¿Qué sucede si cambia el orden de los catch? ¿Afecta en este caso ese orden?
D. ¿Cuándo se ejecuta el código del bloque `finally`?
E. Elimine los dos catch y coloque el siguiente. ¿En qué cambia el código ahora? ¿Cuál opción le parece mejor?

```
catch (NullPointerException | IndexOutOfBoundsException ex) {
System.out.println("Excepcion de puntero null o índices fuera de
rango");
}
```

F. ¿Qué sucede si coloca de primero la siguiente clausula?

```
catch (Exception e) {
 System.out.println("Hubo una excepcion"); }
```

G. Cambie la línea 18 por el siguiente código. ¿Qué sucede? ¿Por qué?

```
System.out.println("Sigo normal " + v[0]);
```

2. Ejercicio de clase

```
1  import java.io.*;
2  import java.util.Scanner;
3
4  public class ObjTaller11B {
5
6        public static void main(String[] args) {
7              try {
8                    new Gestor().gestionar();
9              } catch (IOException ex) {
10                   try {
11                         new Creador().gestionar();
12                   } catch (IOException ex1) {
13                         System.out.println("Excepcion tipo IO");
14                         System.out.println(ex.getMessage());
15                   }
16             }
17       }
18 }
19
20 class Gestor {
21
22       void gestionar() throws FileNotFoundException, IOException {
23             FileInputStream fileInput = new
   FileInputStream("D://prueba.txt");
24             Scanner sc = new Scanner(fileInput);
25             System.out.println(sc.nextLine());
26             sc.close();
27       }
28 }
29
30 class Creador extends Gestor {
31
32       void gestionar() throws FileNotFoundException, IOException {
33             FileOutputStream fileOut = new
   FileOutputStream("D://prueba.txt");
34             OutputStreamWriter out = new OutputStreamWriter(fileOut);
35             out.write("Hola");
```

```
36              out.close();
37        }
38  }
```

NOTA: Tenga en cuenta que si su computador no tiene disco D debe cambiar la ruta por una válida (donde se tenga acceso de escritura).

Responda:

A. De manera general, ¿qué está haciendo el programa?

B. Quite la excepción *FileNotFoundException* de la cláusula *throws* del método *gestionar()* de la clase *Gestor*. ¿Se genera un error en el método *gestionar()* de la clase *Creador* por no declarar la misma cantidad y tipo de excepciones que el método de su padre?

C. Agregue la excepción *NullPointerException* a la cláusula *throws* del método *gestionar()* de la clase *Creador*. ¿Se genera un error por no declarar la misma cantidad y tipo de excepciones que el método de su padre?

D. Si el método *gestionar()* declara que puede lanzar dos excepciones, ¿por qué cree que en el *main* solo se hace el catch para una?

E. Cambie la ruta del archivo por una ruta en el disco W y observe que sucede. ¿Qué tipo de excepción se genera?

3. Ejercicio de clase

```
1   public class ObjTaller11C {
2
3         public static void main(String[] args) {
4               try {
5                     m2();
6               } catch(ExcepcionG ex) {
7                     System.out.println(ex.getMessage());
8                     ex.printStackTrace();
9               }
10        }
11
12        static void m1() throws ExcepcionG {
13              throw new ExcepcionG();
14        }
15
16        static void m2() throws ExcepcionP {
17              try {
18                    m1();
19              } catch (ExcepcionG ex) {
20                    throw new ExcepcionP();
21              }
22        }
23  }
24
25  class ExcepcionG extends Exception {
26
27        public ExcepcionG() {
28              this("Excepcion G");
```

```
29          }
30
31          protected ExcepcionG(String msg) {
32                super(msg);
33          }
34    }
35
36    class ExcepcionP extends Exception {
37
38          public ExcepcionP() {
39                super("Excepcion P");
40          }
41    }
```

Responda:

A. Es correcto que en el main solo se haga un catch de ExcepcionG? Explique su respuesta.
B. Agregue la siguiente clausula en el try-catch del *main*, después del catch de *ExcepcionG*. ¿Qué sucede y por qué? ¿Qué sucede si se cambia el orden de los catch?

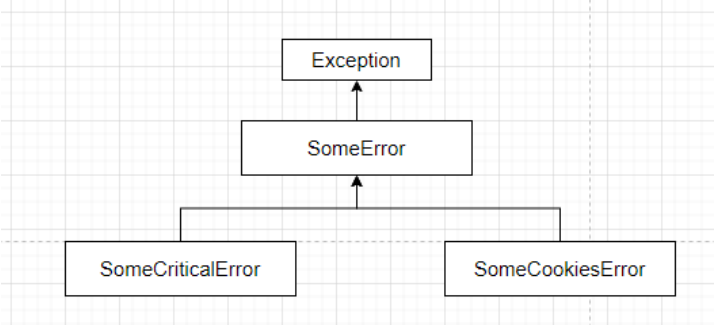
```
catch (ExcepcionP ex) {
 System.out.println(ex.getMessage());
}
```

C. ¿Qué sucede si el método *m2()* realiza un *throws* de *ExcepcionG* y no *ExcepcionP*, y el catch del *main* captura un *ExcepcionP* y no un *ExcepcionG*?
D. ¿Qué hace el método *getMessage()*? ¿Qué diferencia tiene con *printStackTrace()*?
E. Ponga la clase *ExcepcionG* a extender de *RuntimeException* y elimine los *throws* de la declaración de los métodos *m1()* y *m2()*. ¿Por qué no se genera error en dichos métodos si están lanzando excepciones pero no están diciendo que lo hacen?
F. ¿Por qué cree que no siempre es necesario hacer catchs de excepciones como *IndexOutOfBoundsException* o *NullPointerException* así sepamos que se pueden generar?

Python

4. Ejercicio de clase

Realizar la siguiente jerarquía de errores.



Cuando se lance una excepción la clase SomeError aportara la primera parte de lo que se mostrara como error y las clases que hereden de esta deberán poner el complemento de lo que se mostrara.
Para probar estas clases se deberá crear un archivo principal el cual se encargara de hacer el try except y deberá hacer un raise a cualquiera de las dos clases hijo para ser probado