1.- Determine si el siguiente código es correcto. Si produce un error, observe de qué tipo es y soluciónelo.

```
class Excepcion1 extends Exception{}
class Excepcion2 extends Exception1{}
public class Test1 {
   public static void main(String[] args) {
     try {
             throw new Exception2();
     } catch(Excepcion1 e1) {
System.out.println("Se capturó la Excepción1");
     } catch( Excepcion2 e2) {
System.out.println("Se capturó la Excepción2");
     }
   }
}
```

Hay muchas cosas que estaban escritas mal pero asumo que no es a lo que iba el ejercicio. Para que copmpile tuve que borrar el segundo catch porque me dice que la excepción ya fue capturada.
Excepcion2 hereda de Excepcion1, por lo que cualquier excepción de tipo Excepcion2 también puede ser capturada por el catch de Excepcion1.

```
class Exception1 extends Exception{}
class Exception2 extends Exception1{}
public class Test1 {
   public static void main(String[] args) {
       try {
           throw new Exception2();
       } catch(Exception1 e1) {
           System.out.println("Se capturó la Excepción1");
       }
   }
}
```

Otra opci;on es invertirlos. Esto me deja compilar pero em dice que el de abajo es inaccesible pero creo que no deber;ia dar error?
Unreachable section: exception 'Exception2' has already been caught
Creo que la primera va a tomar las de e2 y la segunda solo e1

```
public class Test1 {
   public static void main(String[] args) {
       try {
           throw new Exception2();
       } catch (Exception2 e2) {
           System.out.println("Se capturó la Excepción2");
       } catch (Exception1 e1) {
           System.out.println("Se capturó la Excepción1");
```

```
        }
    }
}
```

2.- Ejecute el siguiente código. ¿Cuál es el resultado?. Elimine los comentarios y vuelva a ejecutarlo. ¿Cuál es el resultado?.

```
public class Test2 {
public int unMetodo(){
   //  try {
      System.out.println("Va a retornar 1");
      return 1;
   // } finally {
      System.out.println("Va a retornar 2");
      return 2;
   // }
 }
 public static void main(String[] args) {
    Test2 res = new Test2();
    System.out.println(res.unMetodo());
 }
}
```

sin los comentarios chilla porque pues hay 2 returns y el segundo no va a llegar nunca
El finally se ejecuta siempre
Va a retornar 1
Va a retornar 2
2
Se entra en el bloque try
👉 se imprime Va a retornar 1.
👉 el programa prepara retornar 1, pero aún no termina el método.

Antes de que se efectúe realmente el return 1, Java siempre ejecuta el bloque finally si existe.
👉 se imprime Va a retornar 2.
👉 hay un return 2 dentro del finally.

En Java, si hay un return en el finally, ese valor pisa cualquier valor anterior que estuviera por retornarse.
👉 el 1 preparado en el try se descarta y el método devuelve 2.

3.- Ejecute el siguiente código. ¿Cuál es la salida del programa?

```
public class Test3 {
   public static void main(String[] args) {
          System.out.println("Test3");
          try {
                System.out.println("Primer try");
                try {
                      throw new Exception();
                } finally {
                      System.out.println("Finally del 2° try");
                }
          } catch (Exception e) {

                   System.out.println("Se capturó la Excepción ex del 1° Primer try");
          } finally {
                System.out.println("Finally del 1° try");
          }
   }
}
```

Test3
Primer try
Finally del 2º try
Se capturó la Excepción ex del 1º Primer try
Finally del 1º try

4.- Analice el siguiente código y determine si es correcto. Si hay errores, escriba el motivo de cada uno y proponga una solución.

```
class FutbolException extends Exception{}

class Falta extends FutbolException{}

class EquipoIncompleto extends
                        FutbolException{}

class ClimaException extends Exception{}

class Lluvia extends ClimaException{}

class Mano extends Falta{}

class Partido {
  Partido() throws FutbolException{}
  void evento() throws FutbolException{}
  void jugada() throws EquipoIncompleto,
                       Falta{}
  void penal(){}
}

inteface Tormenta {
  void evento() throws Lluvia;
  void diluvio() throws Lluvia;
}
```

```
public class Encuentro extends Partido
                  implements Tormenta {

Encuentro() throws Lluvia,
                    FutbolException{..}
Encuentro (String fecha) throws Falta,
                    FutbolException{..}

void penal() throws Mano{..}
public void evento() throws Lluvia{..}
public void diluvio() throws Lluvia{..}
public void evento(){..}
void jugada() throws Mano{..}

public static void main (String[] args) {
     try {
        Encuentro enc = new Encuentro();
        enc.jugada();
     } catch(Mano e) {
     } catch(Lluvia e) {
     } catch(FutbolException e) {
     try {
        Partido par = new Encuentro();
        par.jugada();
     } catch(EquipoIncompleto e) {
     } catch(Falta e) {
     } catch(Lluvia e) {
     } catch(FutbolException e) {}
  }
}
```

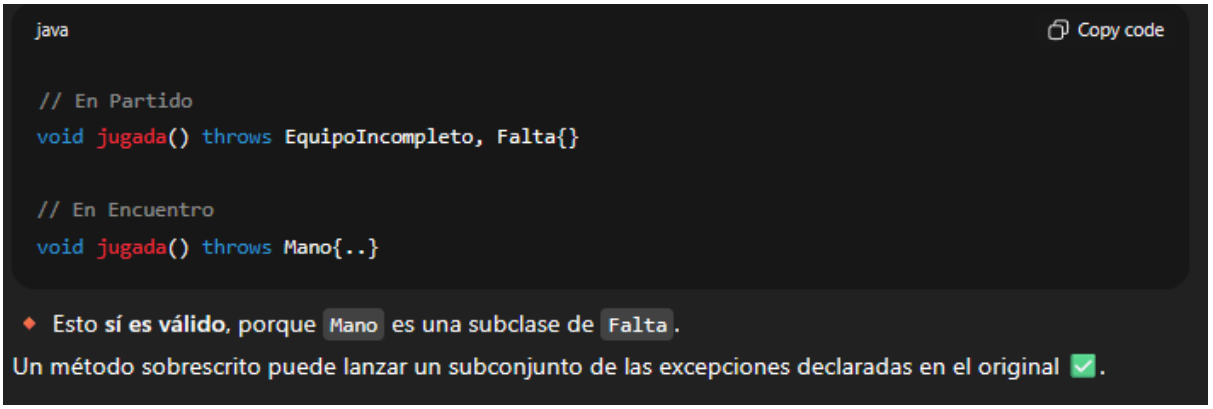
errores:

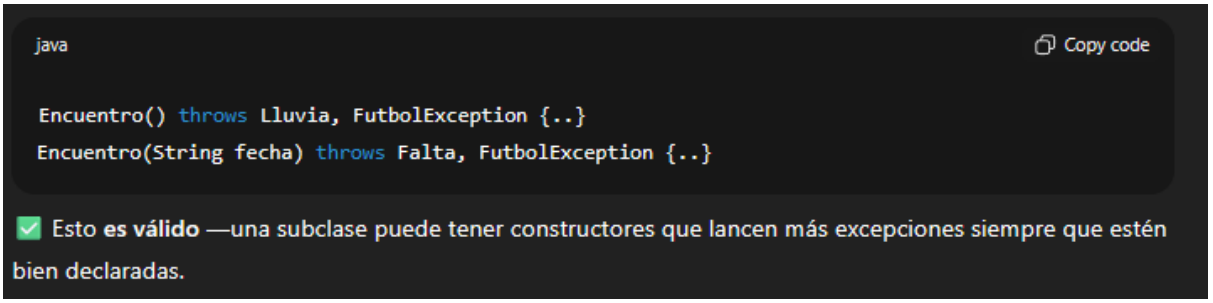
- tipeo en interface tormenta (dice interface)
- Evento en encuentro está dos veces y en ambos con la misma firma.
- Evento en partido dice que tira FutbolException pero después en encuentro tira lluvia que no es de ese tipo.
  soluciones:

public void evento() throws FutbolException { ... }
o en partido hacer void evento() throws FutbolException, Lluvia{}

- En penal de encuentro se dce que tira Mano pero en partido no tirana nada. no se puede.

“no podés agregar checked exceptions nuevas en la sobreescritura si el método original no lanza ninguna. (Mano es checked porque hereda de Exception → no compila.)”



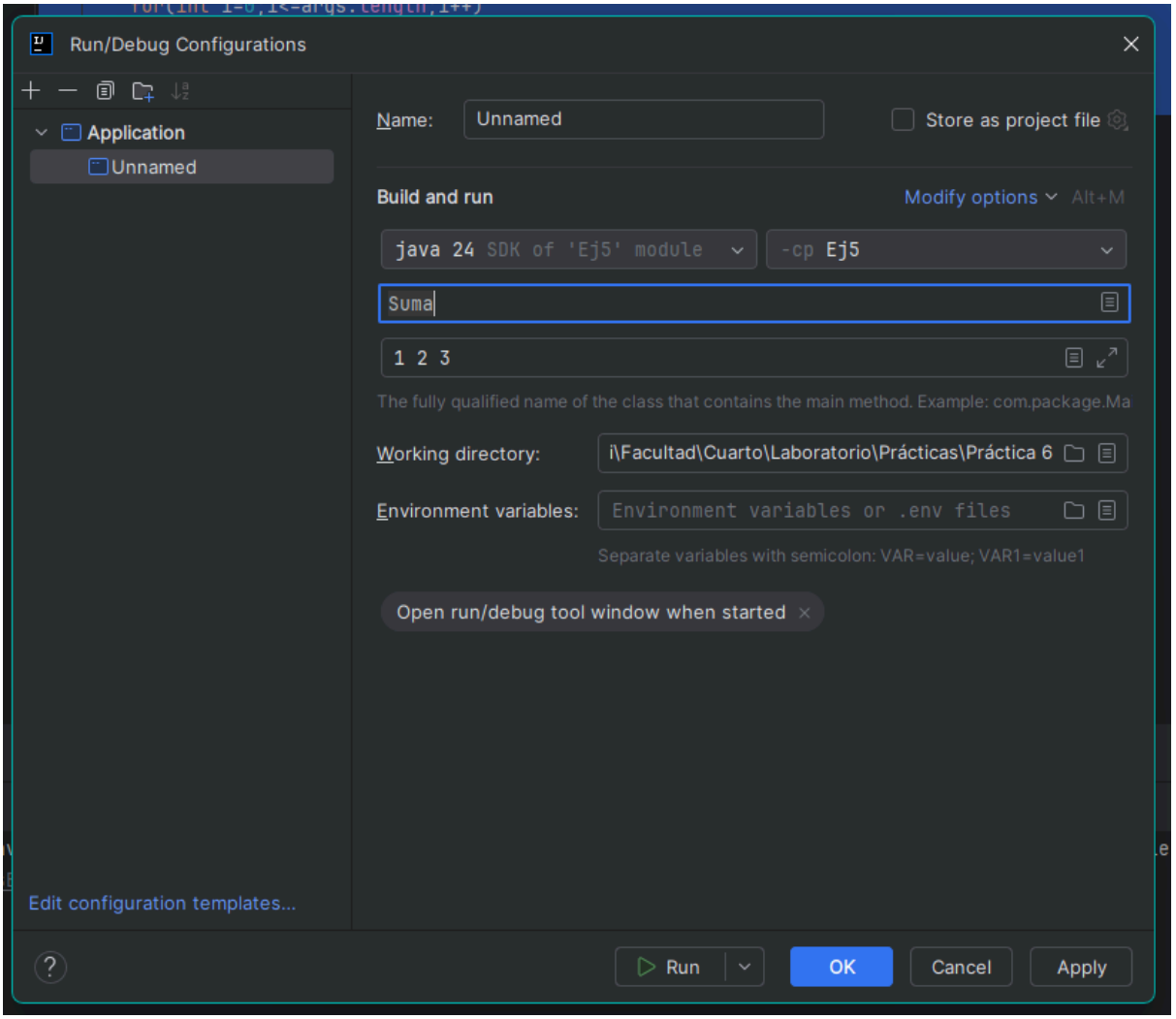


los try catch están bien, se va desde excepciones más concretas a más generales.

5.- Analice el siguiente código:

```
public class Suma {
    public static void main(String[] args){
      int suma=0;
      for(int i=0;i<=args.length;i++)
         suma+= Integer.parseInt(args[i]);
      System.out.print("La suma es:"+suma);
    }
}
```

a) Ejecútelo ingresando al menos 2 valores.
Explota porque se va out of bounds.



Para mandar parametros

b) Ahora ejecútelo ingresando: 2 3 four. ¿Qué pasó?. Solucione el problema de manera que los datos no numéricos sean impresos en la consola con un mensaje y descartados antes de ser sumados.

```java
public class Suma {  new *
    public static void main(String[] args){  new *
        int suma=0;
        for(int i=0;i<args.length;i++){

            try{
                suma+= Integer.parseInt(args[i]);
            }
            catch (Exception e){
                System.out.println((args[i])+" no es un número, error: "+e);
            };
        }
        System.out.print("La suma es:"+suma);
    }
}
```

c) ¿Por qué no fue necesario capturar la excepción en el inciso a) ?

En Java **no es obligatorio** capturar excepciones de tipo `RuntimeException` (como `NumberFormatException`), solo las *checked exceptions* deben capturarse o declararse con `throws`.

👉 `NumberFormatException` es una **unchecked exception**, por eso **no fue necesario** usar `try-catch` en el inciso a).

No había entendido que preguntaba esto xd