Laboratorio de Software - 2025
Práctica n° 6

Temas
- Conceptos y uso de excepciones
- Try/catch/finally
- Definición de Excepciones

(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");
     }
   }
}
```

El código no funciona, se produce un error de compilación. En Java los bloques catch se revisan de arriba hacia abajo. Cuando surge una excepción de tipo Exception2, que es subclase de Exception1, el primer catch ya la captura. Entonces, el segundo catch es inalcanzable, lo que provoca el error.
Siempre hay que capturar primero las subclases y luego las superclases. Para solucionar el error basta con cambiar el orden de los catch.

(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());
 }
}
```

Ejecutarlo con los comentarios resulta en que el código sólo se ejecute hasta la línea return 1;, por lo que el resultado en consola es

```
Va a retornar 1
1
```

Ejecutarlo descomentado hace que se ejecute completo. Ingresa en el try y ejecuta ambas líneas, pero antes de completar el return se ejecuta el bloque finally, cambiando el valor de return. El resultado en consola es:

```
Va a retornar 1
Va a retornar 2
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");
          }
   }
}
```

La salida del programa es:

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

EL flujo es: try interno → excepción → ejecuta finally interno → excepción se propaga → try externo → excepción capturada → ejecuta catch → ejecuta finally externo

(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) {}
  }
}
```

La jerarquía de excepciones está bien formada.

La clase `Partido` declara correctamente las excepciones que puede lanzar.

La interface `Tormenta` también declara correctamente sus excepciones.

Los constructores de la clase `Encuentro` son correctas ya que pueden lanzar excepciones que son subtipos de las declaradas por `Partido` además de la excepción `Lluvia`.

El método `penal()` produce un error de compilación. El método original no lanza excepciones, al sobrescribir un método no se puede declarar que lanza nuevas excepciones que no están en la clase padre.
La solución puede ser quitar el `throws Mano` del método `penal()` en `Encuentro` o agregar la excepción en el método `penal()` en `Partido`.

El primer método `evento()` en `Encuentro` no compila, mientras que el segundo sí.
La clase `Encuentro` extiende una clase (`Partido`) e implementa una interfaz (`Tormenta`), y ambas definen un método `evento()`.
`Partido` declara que este método puede lanzar una excepción `FutbolException`, mientras que `Tormenta` declara que puede lanzar una excepción `Lluvia`.
El método en `Encuentro` que lanza `Lluvia` no cumple con el contrato de su clase padre, ya que `Lluvia` no es subtipo de `FutbolException`.

En cambio, el segundo método `evento()` (que no lanza ninguna excepción) sí compila, porque lanzar menos excepciones es más restrictivo y compatible con ambos contratos.
Sin embargo, no pueden existir dos métodos con la misma firma, por lo que definir ambos simultáneamente genera un error de duplicación.
La solución es dejar únicamente el método `evento()` que no declara `throws`.

El método jugada() throws Mano es válido porque la excepción Mano es subclase de Falta que sí está declarada en la clase padre.

(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.
El límite del for es incorrecto, debería ser args.length – 1.
Corregido eso, el código funciona correctamente.

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.
Four no se puede convertir a int.
Para solucionarlo podemos envolver la operación en un try-catch

```
try {
    suma += Integer.parseInt(args[i]);
} catch (NumberFormatException e) {
    System.out.println("Valor no numérico descartado: " + args[i]);
}
```

c- ¿Por qué no fue necesario capturar la excepción en el inciso a)?
Porque los datos ingresados eran correctos.