Control de excepciones

Programación I

Grado en Ingeniería Informática
MDR, JCRdP y JDGD

Introducción

- Si durante la ejecución de un programa su produce un error
  - ¿Qué hacer después de que ocurra un error?
  - ¿Cómo manejar el error?
  - ¿Quién lo maneja?
  - ¿Puede recuperarse el programa del error o debe terminar?
- Una excepción es un evento (interno o externo) que ocurre durante la ejecución de un programa y que **impide** que continúe **el flujo normal** de instrucciones

Excepciones en Java

- Si se **produce** o **lanza** una excepción se **abandona la ejecución** en marcha (No hay forma de retornar al estado previo a la excepción)
- La excepción está **representada por un objeto** que contiene información sobre ella
- El tipo de objeto determina el tipo de excepción
- **La ejecución continua en un gestor de excepciones** o termina el programa

Excepciones en Java

- Para indicar que se desea gestionar las excepciones que se produzcan en una parte de la ejecución del programa se establece un bloque **try**
- El bloque try encierra la ejecución de código en el que se pueden producir excepciones
- Las excepciones se pueden producir directamente o indirectamente en métodos llamados durante la ejecución de ese código

Excepciones en Java

- Los gestores de excepciones que se encargaran de tratar las excepciones se definen con la instrucción **catch**
- Esta instrucción se sitúa después del bloque try y tiene la apariencia de un procedimiento (no devuelve nada) con un parámetro
- El tipo del parámetro determina el tipo de excepción que gestionará

Excepciones en Java

- Pueden existir varios **catch** asociados a un **try**
- Opcionalmente se puede tener un bloque **finally** que se sitúa después de los **catch** y se ejecuta siempre después del bloque **try** haya ocurrido o no una excepción
- Para lanzar (producir) una excepción se usa la instrucción **throw** objeto
- La instrucción **throw** puede aparecer en cualquier parte del código

Formato (1 /2)

```
try {
  //Código que puede lanzar excepciones
}
catch(TipoExcepción1 e) {
  //Gestor de la excepción de tipo TipoExcepción1
}
catch(TipoExcepción2 e) {
  //Gestor de la excepción de tipo TipoExcepción2
}
//Puede haber tantos catch como sean necesarios
```

Formato (2/2)

```
//...
finally {
  //Después de los catch
  //Opcionalmente se puede tener este bloque
}
//-------------------------------------
//...
throw objeto;
//La instrucción anterior lanza una excepción del tipo
//del objeto
//Puede aparecer en cualquier parte del código
```

¿Qué ocurre cuando se lanza una excepción?

- Se abandona la ejecución donde se produce o lanza la excepción
- Se salta al último **try** en el que se entró
- Si existe un **catch** para el tipo de excepción producida (o más general), se ejecuta éste
- Si no existe, se salta al **try** anterior al actual hasta encontrar un **catch** que gestione la excepción, si no se encuentra termina la ejecución del programa
- Siempre que se abandona un **try**, incluido el que gestione la excepción, se ejecutarán los posibles bloques **finally**

Ejemplo de lanzamiento de excepción

```
try{
    //…
    o.f();
}
catch(Exception1 e){

}
catch(RuntimeException e){

}
finally{
    //Siempre se ejecuta
}
```

```
public void f()
throws Exception1 {
    //…
    g();
    //…
}
private void g()
throws Exception1{
    //…
    throw new Exception1();
    //…
}
```

Jerarquía de clases de excepciones

¿Cuándo se necesita especificar que una excepción puede salir de un método?

- Algunas excepciones es obligatorio especificarlas o capturarlas
- Para especificar las excepciones se escribe, después del cierra paréntesis “)” de los parámetros y antes del abre llaves “{“, **throws** y las clases de excepciones que pueden salir de él
- Es obligatorio especificar que excepciones pueden salir de un método, ya sea porque se lanza con **throw** o porque no captura la excepción que puede lanzar otro método llamado desde éste
- Las excepciones que **no es obligatorio** capturar o especificar son **las derivadas de RuntimeException**

Excepciones comunes

clase	Descripción
ArithmeticException	Error en una operación aritmética, por ejemplo: división por cero de un entero. No obligatorio especificar
NullPointerException	Se intenta acceder a un objeto cuya referencia es null. No obligatorio especificar
IOException	Es una clase genérica de la que derivan clases especificas que se lanzan en caso de que se produzca un error de entrada y salida. Obligatorio especificar
NegativeArraySizeException	Intento de crear un vector de tamaño negativo. No obligatorio especificar

Ejemplo (1 /2)

```
public class Persona2{
   private String nombre;
   private long teléfono;
   public Persona2(String r) {
         nombre=r;
   }
   public Persona2(String r, long i) throws Exception {
         nombre=r;
         setTeléfono(i);
   }
   public void setTeléfono(long t) throws Exception {
     if(t<0) throw new Exception("Número de teléfono negativo");
     teléfono=t;
   }
   public long getTeléfono() {
         return teléfono;
   }
}
```

Ejemplo (2/2)

```
public class Excepcion {
    public static void main(String args[]){
        Persona2 p;
        String n="Nombre";
        int t=-928458733;
        try {
            p =  new Persona2(n,t);
        }
        catch(Exception e)
        {
            System.out.println(e.getMessage());
        }
    }
}
```