Programación en Java

Tema 4. Excepciones en Java

Luis Rodríguez Baena

Universidad Pontificia de Salamanca (campus Madrid)
Facultad de Informática

Excepciones (I)

- Permiten la captura de errores en tiempo de ejecución.
  - El control de excepciones permite extraer el código de manejo de excepciones (controladores de excepciones) del código base del programa.
- Una excepción se lanza cuando el entorno de ejecución se encuentra una condición de excepción.
  - Se puede producir por:
    - ✓ Una condición anormal de ejecución (*excepción síncrona*) detectada por la JVM.
    - ✓ La ejecución del programa detecta una instrucción `throw`.
    - ✓ Un error interno de la JVM (*excepción asíncrona*) o la llamada al método `stop` de la clase `Thread`.

Excepciones (II)

- Cuando se lanza una excepción:
  - Se crea un objeto excepción que desciende de la clase `Trowable` o de alguna de sus subclases.
  - Se detiene el flujo normal de ejecución y se lanza una referencia al objeto excepción creado.
  - Se interpone el mecanismo de gestión de excepciones que busca un lugar para continuar la ejecución.
    - ✓ Si existe un gestor de excepciones se realizan sus instrucciones.
      - × El gestor que se ejecuta es el más cercano en la pila de ejecución de métodos.
    - ✓ Si no existe se lanza el gestor de excepciones por omisión que lanza un mensaje informativo y detiene la ejecución del programa.

Capturar una excepción

- Cuando un método guarda una excepción se asume que esta será capturada y tratada.
- Esa captura se produce en la *región guardada.*
- Cualquier método que pueda producir una excepción deberá introducirse en una región guardada.
  - El bloque `try` debe "envolver" cualquier código o invocación a un método que lanze una excepción.

```
try{
   // Código que produce una excepción
}
```

Gestionar una excepción (I)

- Los distintos tipos de excepción se gestionan mediante un *controlador de excepciones*.
- Los controladores aparecen después del bloque `try` y se identifican mediante la palabra clave `catch`.
- Cada controlador es un pequeño método que toma un único argumento de un tipo concreto que corresponde con el tipo de excepción capturada.

```
try{
...
}catch(TipoExcepción1 idExcepción1){
   //Tratamiento de la excepción TipoExcepción1
}
catch(TipoExcepción2 idExcepción2){
   //Tratamiento de la excepción TipoExcepción2
}
...
```

Gestionar una excepción (II)

- El entorno de ejecución buscará la cláusula `catch` cuyo tipo de argumento coincida con la excepción producida.
- Cuando la encuentra ejecuta las instrucciones del bloque.
  - En ocasiones el bloque puede modificar las condiciones que han producido el error y volver a ejecutar el método.
  - En otras ocasiones simplemente se podrá informar del error y detener la ejecuación.
- Bloque `finally`.
  - Aparece después de los gestores de excepciones.
  - Proporciona una sección de código que se ejecutará se produzca o no la excepción.

La clase Throwable (I)

- Superclase de todas las clases de excepción.
- Dos tipos heredados:
  - `Error`. Representa los errores del sistema y los errores en tiempo de compilación.
    - ✓ No se captura salvo en ocasiones especiales.
  - `Exception`. Tipo básico que puede lanzarse desde cualquier método de la biblioteca Java.
    - ✓ Es el utilizado para capturar y crear nuevas excepciones.

La clase Throwable (II)

- El subtipo `RuntimeException`.
  - El sistema es el encargado de lanzarla.
  - Cuando se produce una excepción de este tipo el sistema invoca al método `printStackTrace` que proporciona información sobre el método que produjo la excepción.
  - Junto con las excepciones de la clase Error, forman las *excepciones no comprobadas*.
    - ✓ No es necesario capturarlas.
  - Cualquier otra excepción es una *excepción comprobada* y es necesaria su captura.
    - ✓ El compilador da un error si no se captura una excepción comprobada.

La clase Throwable (III)

- Constructores de la clase `Throwable`.
  - *`NombreClase`*`()` genera un objeto de la clase con un mensaje de error nulo.
  - *`NombreClase`*`(String` *`mensaje`*`)`, genera un objeto de la clase con un mensaje descriptivo.
- Métodos de la clase `Throwable`.
  - `String getMessage()`. Muestra el mensaje asociado a la excepción.
  - `String toString()`. Devuelve una descripción del objeto en forma de cadena.
  - `void printStackTrace(PrintStream)`. Imprime el objeto y la traza de llamadas lanzada.

Generación de excepciones

- El método `throw`.
  - Se utiliza para lanzar una excepción.
  - `throw expresión;`.
    - ✓ `expresión` debe tener como resultado un valor asignable a la clase `Throwable`.
- Métodos con la cláusula `throws`.
  - Todos los métodos capaces de lanzar excepciones, deben tener una cláusula `throws`.
  - `throws listaExcepcionesComprobadas`.
    - ✓ La lista contiene, separadas por comas, todas las excepciones que puede lanzar y que no son capturadas en el método.

Creación de excepciones

- Es posible crear nuevas excepciones ampliando la clase `Exception`.
- La nueva clase debe tener al menos un constructor que llame al constructor de la superclase.
  - Por ejemplo `Exception(String mensaje)`.

```
class NumeroNegativoException extends Exception {
  NumeroNegativoException(){
        super("El numero debe ser mayor o igual que 0");
  }
}
```

Ejemplo: método factorial

```
 static long Factorial(int n) throws NumeroNegativoException{
    if(n<=0)
       throw new NumeroNegativoException();
    int f=1;
    for(int i=1;i<=n;i++)
       f*=i;
    return f;
}
public static void main(String args[]){
   int i;
   BufferedReader entrada = new BufferedReader(
      new InputStreamReader(System.in));
   String s = new String();
   try{
      s=entrada.readLine();
      //Convertir la entrada en entero
      i = Integer.parseInt(s);
      System.out.println("Factorial de " + i + ": " + Factorial(i));
      } catch(NumeroNegativoException e){
           System.err.println(e.getMessage());
      } catch(IOException e){
     } catch(NumberFormatException e){ //Se lanza la cadena s no se puede convertir a entero
         System.err.println("Formato erroneo");
     }
   }
}
```

Apéndice: entrada de datos por teclado (I)

- Uno de los métodos que permite la entrada de datos por teclado es el método `readLine()`.
  - En una de sus versiones pertenece a la clase `BufferedReader`.
    - ✓ Desciende de la clase `Reader` que proporciona una entrada basada en caracteres Unicode.
    - ✓ Su constructor precisa un objeto de la clase `InputStreamReader`.
      - × Reciben con flujo (*stream*) de entrada un fluyo de bytes y lo convierten a caracteres Unicode.
      - × Para envolver la entrada de teclado (`System.in`) se utilizará un constructor similar a `new InputStreamReader(System.in)`.
  - La cabecera de `readLine()` contiene una cláusula `throws IOException` por lo que es necesario utilizar un bloque `try` para su utilización.
  - El método devuelve una cadena que deberemos convertir al formato apropiado.

Apéndice: entrada de datos por teclado (II)

- Todo junto.

```
package net.colimbo.util;
import java.io.*;
public class Teclado{
    private static String leer() throws IOException{
          BufferedReader teclado =
                    new BufferedReader(new InputStreamReader(System.in));
          return teclado.readLine();
    }
    //Leer un dato entero
    public static int leerInt() throws IOException{
          return Integer.parseInt(leer());
    }
    //Leer un dato double
    public static double leerDouble() throws IOException{
          return Double.parseDouble(leer());
    }
    //Leer una cadena
    public static String leerString() throws IOException{
          return leer();
    }
}
```