CONTENIDOS

Impreso el marzo 17, 2021

Excepciones

Una excepción es un error que ocurre en tiempo de ejecución haciendo que el programa termine de forma inesperada. Veamos un ejemplo:

```
public class AddArguments {
  public static void main(String args[]) {
    int sum = 0;
    for ( String arg : args ) {
      sum += Integer.parseInt(arg);
    }
    System.out.println("Sum = " + sum);
  }
}
```

Si se introducen los datos adecuados al ejecutar la clase, el resultado será el esperado.

```
java AddArguments 1 2 3 4
Sum = 10
```

Ahora bien, que ocurre si no se introducen en el formato esperado? Se produce una excepción y la ejecución del programa se detiene sin llegarse a completar.

```
java AddArguments 1 two 3.0 4
Exception in thread "main" java.lang.NumberFormatException: For input string: "two"
    at java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
    at java.lang.Integer.parseInt(Integer.java:447)
    at java.lang.Integer.parseInt(Integer.java:497)
    at AddArguments.main(AddArguments.java:5)
```

Para evitar que el programa termine al generarse una excepción, tenemos la opción de capturarla y de esta forma la ejecución continúa.

Realmente una excepción es un objeto de tipo Exception. Cuando la máquina virtual se encuentra con un error que no puede resolver genera una instancia de una clase que hereda

de Exception. El tipo de clase elegido será el del tipo de problema encontrado. Si utilizamos un bloque try-catch para capturar excepciones, lo que realmente pretendemos es capturar ese objeto generado por la maquina virtual. De esta forma evitamos que se interrumpa la ejecución. Utilizamos el bloque try para encerrar aquellas instrucciones «peligrosas», nos referimos a las sentencias donde se podría generar una excepción. El bloque catch es donde capturamos la excepción. Veamos el ejemplo anterior mejorado.

```
public class AddArguments2 {
  public static void main(String args[]) {
    try {
      int sum = 0;
      for ( String arg : args ) {
        sum += Integer.parseInt(arg);
      }
      System.out.println("Sum = " + sum);
    } catch (NumberFormatException nfe) {
      System.err.println("One of the command-line "
                         + "arguments is not an integer.");
    }
  }
}
```

La ejecución daría el siguiente resultado:

```
java AddArguments2 1 two 3.0 4
One of the command-line arguments is not an integer.
```

Qué está ocurriendo? Por qué no muestra el resultado de la suma? El problema es el siguiente: Cuando se lee el primer argumento («1») se realiza la conversión a tipo entero y se acumula a la suma. Cuando se lee el segundo argumento («two»), al intentar hacer la conversión a tipo entero se genera una excepción de tipo NumberFormatException, en este momento se ejecuta el bloque catch para capturar dicha excepción.

El resto de los argumentos ya no se evalúan y tampoco se imprime el resultado final de la suma. Para evitar esto, todavía podríamos mejorar más nuestro código haciendo que sólo la instrucción peligrosa forme parte del bloque try. De esta forma, se evaluarán todos los argumentos y se mostrará la suma. Veamos cómo hacerlo.

```
public class AddArguments3 {
   public static void main(String args[]) {
     int sum = 0;
     for ( String arg : args ) {
        try {
          sum += Integer.parseInt(arg);
        } catch (NumberFormatException nfe) {
          System.err.println("[" + arg + "] is not an integer"
                             + " and will not be included in the sum.");
        }
     }
     System.out.println("Sum = " + sum);
   }
}
```

La ejecución dará ahora el siguiente resultado:

```
java AddArguments3 1 two 3.0 4
[two] is not an integer and will not be included in the sum.
[3.0] is not an integer and will not be included in the sum.
Sum = 5
```

Podemos utilizar múltiples bloques catch pero debemos seguir la regla de declararlos del más especifico al más genérico. En caso contrario se genera un error de compilación. También podemos utilizar try-catch-finally. El bloque finally nos asegura que se ejecutará siempre haya excepción o no. Un buen ejemplo de uso del bloque finally es cuando abrimos una conexión a la base de datos y siempre se debería de cerrar se haya producido una excepción o no. Pues bien, el cierre de la conexión iría dentro del bloque finally.

MANEJO DE EXCEPCIONES MÚLTIPLES

Cuando nos encontramos que podemos tener diferentes excepciones en una ejecución poemos incluirlas todas en un solo bloque.

```
1 public void newMu ltiCatch( ){
2 try{
3      methodThatThrowsThreeExceptions( );
4 }catch(ExceptionOne | ExceptionTwo | ExceptionThree e){
5 }
6 }
```

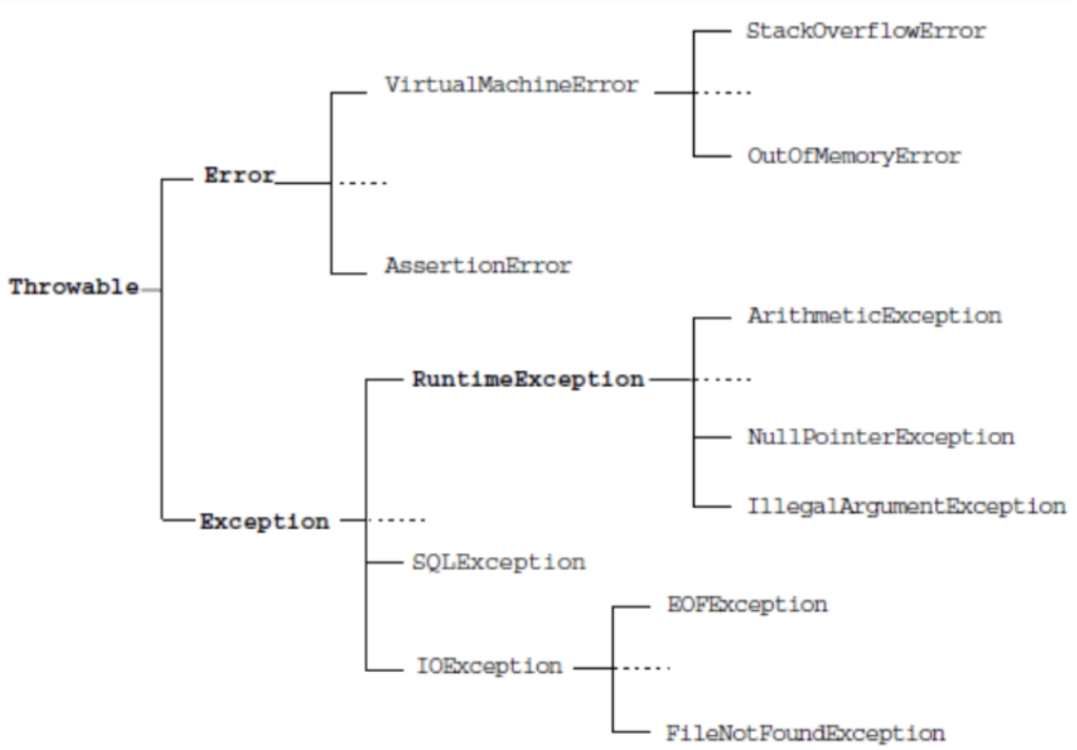
Sin embargo, si tienes excepciones que pertenecen a distintos tipos que deben manejarse de distinta manera, se puede hacer lo siguiente:

```
1 public void newMultiMultiCatch( ){
2 try{
3      methodThatThrowsThreeExceptions( );
4 }catch (ExceptionOne e){
5      //maneja ExceptionOne
6 }catch (ExceptionTwo | ExceptionThree e){
7     //maneja ExceptionTwo y ExceptionThree
```

```
8  }
9  }
```

API EXCEPTION

El API Exception nos permite ver las diferentes clases de las excepciones y la forma en que están organizadas. Como vemos en el diagrama, la clase Exception hereda de la clase Throwable y hermana de ella tenemos la clase Error. Los errores no se pueden capturar. Cuando ocurren el programa termina sin que podamos hacer nada.



PROPAGAR EXCEPCIONES

Si no queremos capturar una excepción generada tenemos otra opción que es lanzarla. Para lanzar una excepción añadimos la clausula throws en el método donde se puede generar la excepción. En este caso, la excepción se propaga a la instrucción donde se ha efectuado la llamada al método. Veamos ejemplos:

```
void trouble() throws IOException { ... }
void trouble() throws IOException, MyException { ... }
```

REGLAS DE SOBREESCRITURA DE METODOS QUE LANZAN EXCEPCIONES

Si queremos sobrescribir o redefinir un método con clausula throws aquí tenemos las reglas a seguir:

- Se puede sobrescribir el método eliminado la cláusula throw
- No se puede sobrescribir para que lance más excepciones que las definidas
- Puede lanzar excepciones de una subclase de la definida originalmente.
- No puede lanzar excepciones que sean de la superclase de la definida en un principio.

EXCEPCIONES PERSONALIZADAS

A pesar de que el API Exception nos ofrecen muchas clases para manejar excepciones, muchas veces nos vamos a encontrar con la situación de querer crear nuestra propia excepción, a eso lo denominamos excepciones personalizadas o propias. Gracias al mecanismo de herencia, para generar nuestra propia excepción es suficiente con crear una clase que herede de la clase Exception. Veamos un ejemplo:

```
public class ServerTimedOutException extends Exception {
  private int port;

  public ServerTimedOutException(String message, int port) {
    super(message);
    this.port = port;
  }

  public int getPort() {
    return port;
  }
}
```

```
public void connectMe(String serverName)
      throws ServerTimedOutException {
  boolean successful;
  int portToConnect = 80;

  successful = open(serverName, portToConnect);

  if ( ! successful ) {
    throw new ServerTimedOutException("Could not connect",
                                      portToConnect);
  }
}
```

En este ejemplo estamos propagando la excepción con throws. También podríamos haberla capturado con un bloque try-catch o try-catch-finally.

Código de ejemplo Excepciones