Excepciones

Salvador López Mendoza

Noviembre de 2022

Introducción

Cuando escribimos programas para consumo propio sabemos qué tipos de datos se deben ingresar, así como los rangos válidos.

Podemos pensar que nuestros programas están bien hechos.

Funcionan correctamente *siempre que reciban los datos adecuados*.

¿Qué pasa cuando el programa es utilizado por una persona distinta al programador?

¿Qué pasa si usamos un programa después de mucho tiempo (y muchos programas) de haberlo programado?

Programas robustos

Los usuarios deben ser capacitados para poder usar un sistema.

A pesar de todo siempre es posible que se ingresen datos incorrectos.

Resultados de recibir un dato incorrecto:

- Se entrega un resultado incorrecto.
  Es posible que el usuario no lo sepa.
- Se genera un mensaje de error.
  Posiblemente el programa termina.
- **El programa se da cuenta y trata de recuperarse.**

Programas robustos (II)

Definición

Un programa es robusto si contempla los posibles errores que pueden ocurrir, debido principalmente al intentar trabajar con algún dato fuera del rango permitido.

Alternativas para hacer programas robustos

- Al programar considerar todos los posibles casos de error.
  En cada método se pregunta si cada parámetro está dentro del rango permitido.
- Aprovechar los mecanismos del lenguaje de programación.

Clases de errores

Existen distintos tipos de errores.

Errores de sintaxis. Los detecta el compilador.

Errores de semántica (o de lógica). Son difíciles de detectar.

Errores de lógica

Existen muchos tipos de errores, ¿cómo detectarlos?

Algunos ejemplos:

- Implementación errónea de un método.
- Variable fuera de rango.
  Índices de un arreglo.
- División por cero.
- Manejar un dato inexistente.
- Acceso a un archivo inexistente.

Para muchos de estos errores Java tiene un mecanismo que facilita su manejo.

Java envía un mensaje indicando el tipo de error.

También informa del método en que ocurre y la línea de código.

Excepciones

Definición

Una **excepción** es un evento que ocurre en cualquier momento de ejecución de un programa y que modifica el flujo normal de éste.

En Java, las excepciones son objetos de la clase `Exception`.

Estos objetos almacenan información que se regresa en caso de que ocurra una anormalidad.

Todos los métodos que hayan llamado al método en que se produce la excepción pueden darse por enterados y alguno de ellos o todos, tomar las medidas apropiadas.

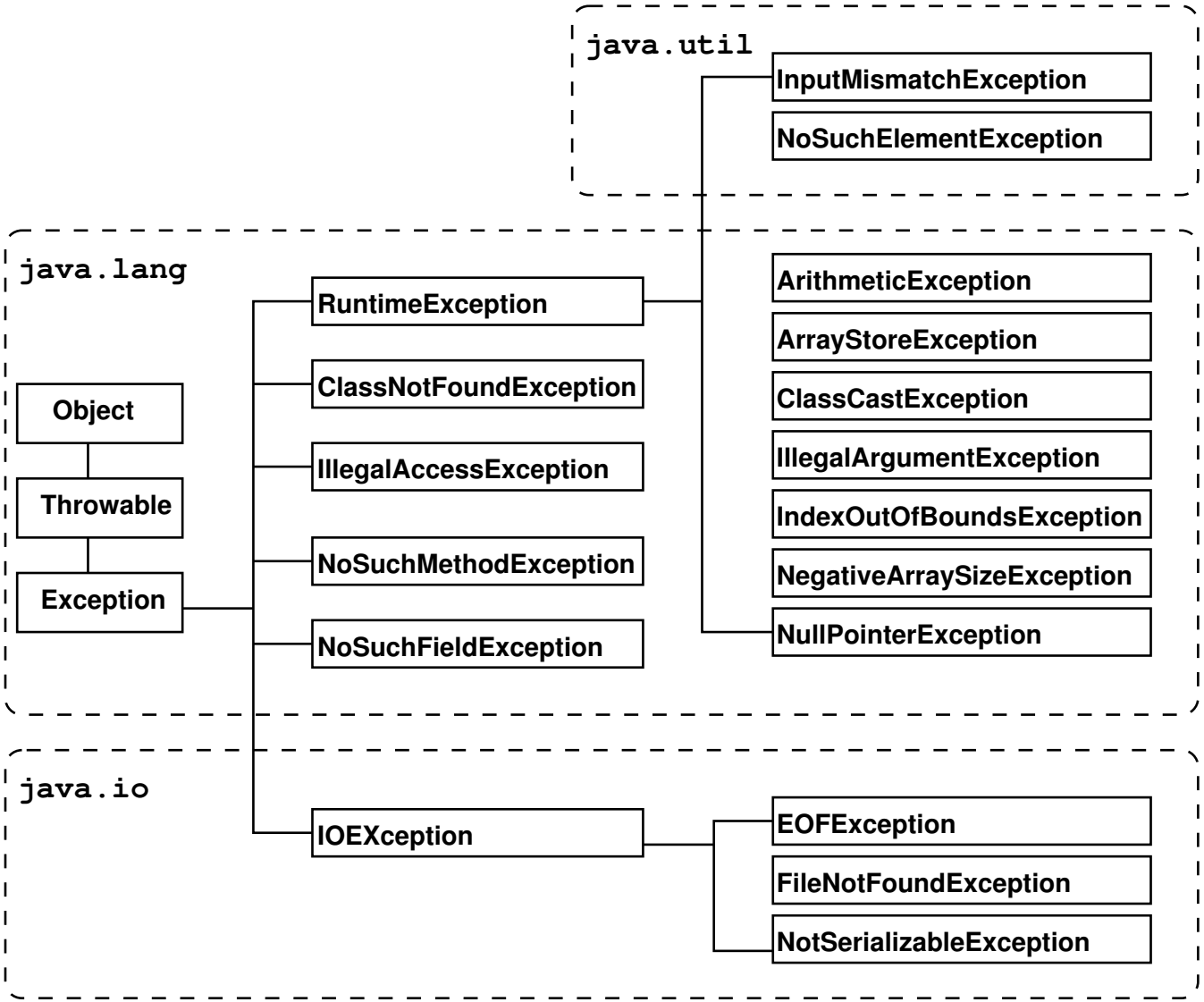
Jerarquía de excepciones

La clase `Exception` también se encuentra en el paquete `java.lang`.

Es la raíz de una jerarquía de clases para los errores más comunes.

También desciende de la clase `Object`.

Jerarquía de clases a partir de Exception



Estados de una excepción

Una excepción se activa para indicar que ocurrió una falla durante la ejecución de un método.

La excepción **se propaga** hasta encontrar un método en el cual se indica qué se debe hacer en circunstancias anómalas.

Este comportamiento se describe en términos técnicos definidos como **estados** por los que pasan las excepciones.

Estados de las excepciones

Disparo. Cuando ocurre un error se activa una excepción creando un objeto de la clase `Exception` con información como: tipo de excepción y estado del programa.

Atrapado. Una vez que un método dispara una excepción, el sistema de ejecución busca las instrucciones que especifican qué hacer para esa excepción.

Terminación. Si no se encuentra un método que maneje la excepción, termina la ejecución del programa.
En caso contrario, se dice que el manejador de excepciones atrapó la excepción y el programa se recupera de la excepción.
La ejecución del programa continúa en forma normal.

Disparo de excepciones

Disparar una excepción es la forma más efectiva para indicar que no es posible atender la petición del objeto que envía el mensaje.

El usuario no puede ignorar que ha ocurrido una excepción.

Se usa la instrucción `throw` con un objeto de la clase `Exception`.

Al ejecutar `throw` es porque ocurrió un error en el método y por lo tanto no puede continuar su ejecución.

Funcionamiento

Si en un método se puede disparar una excepción y ésta no es de la clase `RuntimeException` ni de sus descendientes, entonces al final de la firma del método se debe incluir la cláusula `throws` seguida del nombre de la excepción que se puede disparar.

Se advierte al usuario del método que tengan cuidado. Conviene tomar acciones necesarias para el caso en que se dispare una excepción.

La instrucción `throw` crea un objeto de la clase `Exception`, y luego se dispara la excepción.

La creación de un objeto para tratar la excepción siempre lleva una cadena de caracteres que se usa como mensaje de diagnóstico.

Al disparar una excepción termina la ejecución del método que la disparó.

El método valida el parámetro y en caso de **error** dispara la excepción.

Funcionamiento (II)

No se incluye la instrucción `throws` en el encabezado de un método si la excepción que se dispara es subclase de `RuntimeException`.

¿Qué pasa si en un método se pueden disparar varias excepciones distintas?

En el encabezado del método se especifican todas las clases de excepciones, separadas por comas.

Dentro del método debe haber al menos una instrucción `throw` para cada una de las diferentes clases de excepciones mencionadas en el encabezado.

Manejo de excepciones

¿Qué hacer cuando se dispara una excepción?

Se debe escribir (programar) un manejador de excepciones.

Se usa la instrucción `try`.

```
try {
    instrucciones ...
}
catch (Excepción-1 e-1) {
    instrucciones ...
}
...
catch (Excepción-n e-n) {
    instrucciones ...
}
finally {
instrucciones ...
}
```

Manejo de excepciones (II)

`try` Contiene código que incluye las instrucciones que pueden disparar la(s) excepción(es).

`catch` Tienen como parámetro un objeto de alguna clase de excepción.
En una instrucción `try` puede haber varias cláusulas `catch`.
En el cuerpo o bloque de cada una se coloca el código que implementa la acción a realizar en caso de que ocurra una excepción del tipo de su parámetro.

`finally` contiene el código para establecer un estado adecuado para continuar la ejecución del método donde aparece la instrucción `try`.
Es opcional.

Fin de la excepción

Se usa la cláusula `finally` para realizar tareas importantes (generalmente de recuperación), sin importar cuál de las excepciones se haya disparado.

Esta clausula se ejecuta también en caso de que no se dispare una excepción.

Creación de excepciones propias

En muchas ocasiones las clases de excepciones existentes no describan la naturaleza del error que se tiene, en ese caso y para dar mayor claridad a los programas, es posible crear excepciones propias.

Se extiende la clase `Exception` (usando la herencia).

Las nuevas clases requieren que se proporcione una cadena de diagnóstico al constructor.

Generalmente las nuevas clases no agregan nuevo comportamiento, simplemente sirven para mejorar la legibilidad del código.

Consideraciones generales

Se dice que el bloque de instrucciones dentro de la clausula `try` es un **bloque protegido**.

Cuando una clausula `try` puede disparar varias excepciones cada una de ellas requiere de una clausula `catch`.

Las clausulas `catch` aparecen secuencialmente, en el orden que defina el programador.

Al ejecutarse el bloque protegido, cuando se genera una excepción el intérprete busca secuencialmente (en el orden de escritura) hasta encontrar una clausula `catch` que atrape a la excepción.

Se recomienda que estas cláusulas se especifiquen en orden contrario al de su jerarquía de clases.

Consideraciones generales (II)

En caso de no tener considerado un tipo particular de excepción, se realice la cláusula `catch` de su superclase.

Puede incluirse como última cláusula `catch` la que atrapa `Exception`, puesto que es la raíz de la jerarquía y con el objetivo de asegurar que hay alguna cláusula que atrapa cualquier excepción y no termine el programa abruptamente.

En un programa puede haber tantas instrucciones `try` como se requieran, no se está limitado a una sola.

Propagación de excepciones

Se debe decidir si la excepción va a ser manejada en el método o bien si se va a propagar hacia los métodos de donde proviene la llamada al método que dispara la excepción.

Un método propaga la excepción si:

- En el encabezado se especifica la excepción que dispara.
- No contiene un manejador de excepciones en su cuerpo.

Recuperación de excepciones

El uso de excepciones permite notificar la existencia de algún error.

Es importante que el programa siga trabajando cuando hay excepciones.

La clausula `try` puede ser parte de un ciclo; por ejemplo para volver a pedir datos.

Ventajas de usar excepciones

1. Permiten separar el código para manejo de error, del código normal. Aunque esto no evita que se deba especificar cuáles son los posibles errores y qué hacer en caso de que ocurran.
2. Se pueden agrupar tipos de errores y también se pueden diferenciar errores.
   Por ejemplo, todos los que pueden ocurrir al trabajar con arreglos, con archivos, etc., pero también se puede trabajar con cada uno por separado.
3. Al crear excepciones propias se están creando códigos de error propios y son manejados por Java de manera idéntica a sus excepciones.