Excepciones

Una excepción es un evento no deseable en la ejecución de un programa.

Java identifica dos grandes grupos de excepciones:

* Excepciones verificadas
* Excepciones no verificadas

Excepciones verificadas

Son aquellas que se supone que el programador manejará en el programa y que se originan a partir de condiciones externas que ocurren en un programa que está ejecutando.

Ejemplos:

* Intentar acceder a un archivo que no se encuentra
* Falla en la red

Excepciones no verificadas

Las excepciones no verificadas se denominan de esta manera porque no es necesario realizar ninguna acción para manejarlas.

Pueden ser de dos tipos:

Excepciones en tiempo de ejecución (Runtime Exceptions): Representan bugs de programación. Por ejemplo intentar acceder más allá de la dimensión de un array.

Errores: Son excepciones que se originan como resultado de temas de entorno. Por ejemplo la ejecución de un programa sin memoria suficiente.

Clase Exception

La clase Exception es una clase que nos ofrece Java para administrar las excepciones, ya sean verificadas como no verificadas. Siempre va a ser preferible, ante la ocurrencia de una excepción, escribir un código para el manejo de la misma, en lugar de terminar el programa.

En el siguiente ejemplo desarrollamos una una clase, Numeros, que lo único que contiene es un método que se encarga de sumar los números que contiene un array de tipo String, los cuales se supone se guardarán siempre con valores numéricos de tipo int.



Código 1 – Clase Numeros

Al desarrollar la clase de test, provocamos una excepción en tiempo de ejecución, ingresando un valor no numérico en el array:



Código 2 – Clase TestExcepcionesVerificadas

La ejecución del método test arrojará la siguiente excepción: *java.lang.NumberFormatException: For input string: "Seguro falla"*

Ante la ocurrencia de una excepción, un método tiene dos alternativas:

* Lanzar la excepción al método que lo llamó
* Manejar la excepción
* Declarar las excepciones que un método puede lanzar

Lanzar la excepción al método que lo llamó

Si una excepción alcanza el tope de un hilo (se lanzó tantos métodos hacia arriba que ya no hay mas métodos dónde lanzarla), entonces el hilo es destruido y el programa se cierra.

Manejar la excepción

El lenguaje Java ofrece el mecanismo **try-catch** para manejar una excepción que ocurre durante la ejecución.

Con este mecanismo de maneja de excepciones, el código 2 se vería modificado como sigue:



Código 3 – Clase Numeros mejorada

Podemos tener tantos bloques catch como excepciones se puedan presentar en un bloque de código. Cada bloque atenderá la excepción que corresponda.



Código 4 – Clase Numeros con múltiple manejo de excepciones

El mecanismo try-catch también ofrece la posibilidad de utilizar la cláusula finally para realizar cualquier acción que sea necesaria hacer, ocurra o no la excepción (Por ej. Cerrar todas las conexiones que pudieran haber quedado abiertas).



Código 5 – Clase Numeros con la cláusula finally.

Existe una jerarquía de clases para el manejo de las excepciones. La clase padre de todos los objetos que pueden ser lanzados y atrapados usando el mecanismo de manejo de excepciones es java.lang.Throwable. Son subclases de la clase Throwable las clases:

* Error: Indica un problema severo del que recuperarse es difícil, si no, imposible. Por ejemplo quedarse sin memoria. No se espera que un programa se recupere de estas condiciones.
* RuntimeException: Indica un problema de diseño o implementación. La excepción indica una condición que nunca debería haber sucedido, si el programa funcionara adecuadamente. Por ejemplo, un NullPointerException, nunca debería ser lanzado.
* Exception: Es cualquier excepción que pueda aparecer por efectos del entorno. Por ejemplo no existe el archivo al que se quiere acceder o se ingreso de una URL no válida.

Declarar las excepciones que un método puede lanzar

Si al ejecutar un programa, aparece una RuntimeException(una excepción no verificada), y en ése bloque, no se incorporó ningún catch del tipo de la excepción ocurrida, se lanza la excepción al método llamador.

Ahora bien, para alentar la escritura de código robusto, Java requiere realizar algún tratamiento especial cuando pueda aparecer una excepción que sea del tipo verificada (subclase de Exception pero no de RuntimeException).

Para satisfacer este requerimiento existen las siguientes alternativas:

* Manejar la excepción utilizando un bloque try-catch. La cláusula catch debe atender la clase de la excepción esperada o una superclase de ella.
* Declarar las excepciones que un método puede lanzar.

Este último caso obliga a indicar que el método llamado no maneja la excepción que puede aparecer y que en consecuencia lanzará la excepción al método llamador, el cual debería tener el try-catch con la excepción lanzada.

Un método declara que una excepción puede ser lanzada en el cuerpo del método con la cláusula throws. El código 6 muestra un ejemplo de un método que delega el tratamiento de una excepción verificada al método que lo llamó.



Código 6 – Declaración de las excepciones que un método puede lanzar

De lo visto hasta este momento se desprende que no es necesario declarar el lanzamiento de excepciones de tiempo de ejecución (RuntimeException).

Creación de excepciones propias

Las excepciones definidas por el usuario se crean extendiendo de la clase Exception.

A modo de ejemplo, el código 7 muestra la manera en cómo se desarrollaría una excepción cuando se supere el descubierto disponible en una cuenta corriente.



Código 7 – Creación de un excepción

A través del método getMessage, el cual es heredado de la clase Exception, se podrá obtener el motivo por el cual se produjo la excepción.

El código 8 muestra un ejemplo de cómo se llamaría a la excepción por descubierto desde una clase CuentaCorriente.



Código 8 – Lanzamiento de la excepción creada por el usuario.



Código 9 – Manejo de una excepción definida por el usuario.