Manejo de Excepciones

1.- Introducción

Una excepción es la indicación de un problema que ocurre durante la ejecución de un programa.

El manejo de excepciones nos permite crear aplicaciones que puedan resolver (o manejar) las excepciones.

Sólo las clases que extienden a **Throwable** (paquete java.lang) en forma directa o indirecta pueden usarse para manejar excepciones.

Ejemplos de excepciones utilizadas:
ArrayIndexOutOfBoundsException
IllegalArgumentException
ArithmeticException, **InputMismatchException**
(Archivos, flujos y serialización: **SecurityException**, **FileNotFoundException**, **IOException**, **ClassNotFoundException**, **IllegalStateException**, **FormatterClosedException**, **NoSuchElementException**)
(Colecciones genéricas: **ClassCastException**, **UnsupportedOperationException**, **NullPointerException**)

2.- Ejemplo: división entre cero sin manejo de excepciones

Rastreo de la pila: ruta de ejecución que condujo a la excepción, método por método.

Rastreo de la pila para una excepción **ArithmeticException**

Rastreo de la pila para una excepción **InputMismatchException**

Un programa puede continuar, aun cuando haya ocurrido una excepción y se imprima un rastreo de pila

3.- Ejemplo: manejo de excepciones tipo **ArithmeticException** e **InputMismatchException**

Esta versión de la aplicación utiliza el manejo de excepciones de manera que, si el usuario comete un error, el programa atrapa y maneja (es decir, se encarga de) la excepción

Encerrar código en un *bloque **try***

Un *bloque **try*** encierra el código que podría lanzar (**throw**) una excepción y el código que no debería ejecutarse en caso de que ocurra una excepción

Atrapar excepciones

Un *bloque **catch*** (también conocido como *cláusula **catch*** o *manejador de excepciones*) atrapa, i.e., recibe, y maneja una excepción.

Al menos debe ir un *bloque **catch*** o un *bloque **finally*** justo después del *bloque **try***.

Error común de programación (CPE): es un error de sintaxis colocar código entre un *bloque* ***try*** y sus correspondientes *bloques* ***catch***.

Cada *bloque* ***catch*** especifica entre paréntesis un parámetro de excepción, que identifica el tipo de excepción que puede procesar el manejador.

Cuando ocurre una excepción en un *bloque* ***try***, el *bloque* ***catch*** que se ejecuta es el primero cuyo tipo coincide con el tipo de la excepción que ocurrió

Multi-**catch**

Si los cuerpos de varios *bloques* ***catch*** son idénticos, puede usar la característica *multi-****catch*** (introducida en Java SE 7) para atrapar esos tipos de excepciones en un solo *manejador de excepciones* y realizar la misma tarea

catch (tipo1 | tipo2 | tipo3 e)

Excepciones no atrapadas

Una excepción no atrapada es una para la que no hay *bloques* ***catch*** cuyo tipo del parámetro de excepción NO coincida con el tipo de la excepción que se lanzó.

Modelo de terminación del manejo de excepciones

Si ocurre una excepción en un *bloque* ***try***, el *bloque* ***try*** termina de inmediato y el control del programa se transfiere al primero de los siguientes *bloques* ***catch*** en los que el tipo del parámetro de excepción coincide con el tipo de la excepción que se lanzó.

Una vez que se maneja la excepción, el control del programa no regresa al punto de lanzamiento, ya que el *bloque* ***try*** ha expirado (y se han perdido sus variables locales). En vez de ello, el control se reanuda después del último *bloque* ***catch***.

Esto se conoce como el *modelo de terminación del manejo de excepciones*.

Algunos lenguajes utilizan el *modelo de reanudación del manejo de excepciones* en el que, después de manejar una excepción, el control se reanuda justo después del *punto de lanzamiento*.

Si no se lanzan excepciones en el *bloque* ***try***, se omiten los *bloques* ***catch*** y el control continúa con la primera instrucción después de los *bloques* ***catch***.

El *bloque* ***try*** y sus correspondientes *bloques* ***catch*** o *bloque* ***finally*** forman en conjunto una *instrucción* ***try***.

Al igual que con cualquier otro bloque de código, cuando termina un *bloque* ***try***, las variables locales declaradas en ese bloque quedan fuera de alcance y ya no son accesibles; por ende, las variables locales de un *bloque* ***try*** no son accesibles en los correspondientes *bloques* ***catch***. Cuando termina un *bloque* ***catch***, las variables locales declaradas dentro de este bloque (incluyendo el parámetro de excepción de ese *bloque* ***catch***) también quedan fuera de alcance y se destruyen.

Uso de la *cláusula* ***throws***

Esta cláusula, que debe aparecer después de la lista de parámetros del método y antes de su cuerpo, contiene una lista separada por comas de los tipos de excepciones que pueden lanzarse mediante instrucciones en el cuerpo del método, o a través de métodos que se llamen desde ahí.

Tip para prevenir errores (EPT): Antes de usar un método en un programa, lea la documentación en línea de la API para saber acerca del mismo. La documentación especifica las excepciones que lanza el método (si las hay), y también indica las razones por las que pueden ocurrir dichas excepciones.

Cuando una excepción se lanza desde un método, dicho método termina y no devuelve un valor, por lo que sus variables locales quedan fuera de alcance (y se destruyen). Si dicho método contiene variables locales que sean referencias a objetos y no hay otras referencias a esos objetos, éstos se marcan para la *recolección de basura* (*garbage collection*).

4.- Cuándo utilizar el manejo de excepciones

El manejo de excepciones está diseñado para procesar *errores sincrónicos*, que ocurren cuando se ejecuta una instrucción. El manejo de excepciones no está diseñado para procesar los problemas asociados con los *eventos asíncronos* (por ejemplo, completar las operaciones de E/S de disco, la llegada de mensajes de red, clics del ratón y pulsaciones de teclas), los cuales ocurren en paralelo con, e independientemente de, el flujo de control del programa.

Observación de ingeniería de software (SIO): Incorpore su estrategia de manejo de excepciones y recuperación de errores en sus sistemas, partiendo desde el principio del proceso de diseño. Puede ser difícil incluir esto después de haber implementado un sistema.

Observación de ingeniería de software (SIO): El manejo de excepciones proporciona una sola técnica uniforme para documentar, detectar y recuperarse de los errores. Esto ayuda a los programadores que trabajan en proyectos extensos a comprender el código de procesamiento de errores de los demás programadores.

5.- Jerarquía de excepciones en Java

Figura 11.4 (Deitel)

La clase **Exception** y sus subclases, representan situaciones excepcionales que pueden ocurrir en un programa en Java, y que pueden ser atrapadas por la aplicación.

La clase Error y sus subclases representan las situaciones anormales que ocurren en la JVM. La mayoría de los errores tipo **Error** ocurren con poca frecuencia y no deben ser atrapados por las aplicaciones.

Por lo general no es posible que las aplicaciones se recuperen de los errores tipo **Error**.

Comparación entre *excepciones verificadas* y *no verificadas*

Java clasifica a las excepciones en estas dos categorías. Esta distinción es importante, ya que el compilador de Java implementa requerimientos especiales para las excepciones verificadas. El tipo de una excepción determina si es verificada o no verificada.

Las excepciones **RuntimeException** son *excepciones no verificadas*.

Por lo general, se deben a los defectos en el código de nuestros programas, e.g., **ArrayIndexOutOfBoundsException** y **ArithmeticException**.

Excepciones verificadas

Todas las clases que heredan de la clase **Exception** pero no directa o indirectamente de la clase **RuntimeException** se consideran como excepciones verificadas.

Por lo general, dichas excepciones son provocadas por condiciones que no están bajo el control del programa, e.g., en el procesamiento de archivos, el programa no puede abrir un archivo si éste no existe.

El compilador y las *excepciones verificadas*

El compilador verifica cada una de las llamadas a un método, junto con su declaración, para determinar si el método lanza una *excepción verificada*. De ser así, el compilador asegura que la *excepción verificada* sea *atrapada* o sea *declarada* en una *cláusula* ***throws***. A esto se le conoce como *requerimiento de atrapar o declarar* (*catch-or-declare requirement*).

Para satisfacer la parte relacionada con atrapar (**catch**) del *requerimiento de atrapar o declarar*, el código que genera la excepción debe envolverse en un *bloque* ***try***, y debe proporcionar un manejador **catch** para el tipo de *excepción verificada* (o una de sus superclases).

Para satisfacer la parte relacionada con declarar (**declare**) del *requerimiento de atrapar o declarar*, el método que contiene el código que genera la excepción debe proporcionar una cláusula throws que contenga el tipo de excepción verificada, después de su lista de parámetros y antes de su cuerpo.

Si el requerimiento de atrapar o declarar no se satisface, el compilador emitirá un mensaje de error. Esto obliga a los programadores a pensar sobre los problemas que pueden ocurrir cuando se hace una llamada a un método que lanza excepciones verificadas.

Tip para prevenir errores (EPT): Los programadores deben atender las *excepciones verificadas*. Esto produce un código más robusto que el que se crearía si los programadores simplemente ignoran las excepciones.

Error común de programación (CPE): Si el método de una subclase sobrescribe al método de una superclase, es un error para el método de la subclase mencionar más expresiones en su *cláusula* ***throws*** de las que tiene el método sobrescrito de la superclase. Sin embargo, la *cláusula* ***throws*** de una subclase puede contener un subconjunto de la lista **throws** de una superclase.

Observación de ingeniería de software (SIO): Si su método llama a otros métodos que lanzan *excepciones verificadas*, éstas deben atraparse o declararse. Si una excepción puede manejarse de manera significativa en un método, éste debe atrapar la excepción en vez de declararla.

El compilador y las *excepciones no verificadas*

A diferencia de las *excepciones verificadas*, el compilador de Java no examina el código para determinar si una *excepción no verificada* es atrapada o declarada. Por lo general, las *excepciones no verificadas* se pueden evitar mediante una codificación apropiada.

No es obligatorio que se enumeren las *excepciones no verificadas* en la *cláusula* ***throws*** de un método; aún si se hace, no es obligatorio que una aplicación atrape dichas excepciones.

Observación de ingeniería de software (SIO): Aunque el compilador no implementa el requerimiento de atrapar o declarar para las *excepciones no verificadas*, usted deberá proporcionar un código apropiado para el manejo de excepciones cuando sepa que podrían ocurrir. Esto hará que sus programas sean más robustos.

Atrapar excepciones de subclases

Si se escribe un manejador **catch** para atrapar objetos de excepción de un tipo de superclase, también se pueden atrapar todos los objetos de las subclases de esa clase. Esto permite que un bloque catch maneje las excepciones relacionadas mediante el polimorfismo. Si estas excepciones requieren un procesamiento distinto, puede atrapar individualmente a cada subclase.

Sólo se ejecuta la primera *cláusula **catch*** que coincida

Si hay varios *bloques **catch*** que coinciden con un tipo específico de excepción, sólo se ejecuta el primer *bloque **catch*** que coincida cuando ocurra una excepción de ese tipo. Es un error de compilación atrapar el mismo tipo exacto en dos *bloques **catch*** distintos asociados con un *bloque **try*** específico.

Error común de programación (CPE): Al colocar un *bloque **catch*** para un tipo de excepción de la superclase antes de los demás *bloques **catch*** que atrapan los tipos de excepciones de las subclases, evitamos que esos *bloques **catch*** se ejecuten, por lo cual se produce un error de compilación.

Tip para prevenir errores (EPT): Al colocar un *bloque **catch*** para el tipo de la superclase después de los *bloques **catch*** de todas las otras subclases aseguramos que todas las excepciones de las subclases se atrapen en un momento dado.

Observación de ingeniería de software (SIO): En la industria no se recomienda lanzar o atrapar el tipo **Exception**; aquí lo usamos sólo para demostrar la mecánica del manejo de excepciones. En capítulos subsiguientes, por lo general lanzaremos y atraparemos tipos de excepciones más específicos.

6.- *Bloque **finally***

Los programas que obtienen ciertos recursos deben devolverlos al sistema para evitar las denominadas *fugas de recursos* (resource leaks).

En lenguajes de programación como C y C++, el tipo más común de *fuga de recursos* es la *fuga de memoria* (memory leak).

Java realiza la *recolección automática de basura* (automatic garbage collection) en la memoria que ya no es utilizada por los programas, evitando así la mayoría de las fugas de memoria.

Sin embargo, pueden ocurrir otros tipos de fugas de recursos en Java. Por ejemplo, los archivos, las conexiones de bases de datos y conexiones de red que no se cierran apropiadamente cuando ya no se necesitan, podrían no estar disponibles para su uso en otros programas.

Tip para prevenir errores (EPT): Hay una pequeña cuestión en Java: no elimina completamente las fugas de memoria. Java no hace recolección de basura en un objeto, sino hasta que no existen más referencias a ese objeto. Por lo tanto, si los programadores mantienen por error referencias a objetos no deseados, pueden ocurrir fugas de memoria.

El *bloque **finally*** (que consiste en la palabra clave **finally**, seguida de código encerrado entre llaves) es opcional, y algunas veces se le llama *cláusula **finally***. Si está presente, se coloca después del último *bloque **catch***. Si no hay *bloques **catch***, el bloque finally sigue justo después del bloque try .

El bloque finally se ejecutará, independientemente de que se lance o no una excepción en el bloque try correspondiente.

El *bloque **finally*** también se ejecutará si un *bloque **try*** se sale mediante el uso de una instrucción **return**, **break** o **continue**, o simplemente al llegar a la llave derecha de cierre del *bloque **try***.

El único caso en donde el *bloque **finally*** no se ejecutará es si la aplicación sale antes de tiempo de un *bloque **try***, llamando al método **System.exit**. Este método, que demostraremos en el capítulo 15, termina de inmediato una aplicación.

Si una excepción que ocurre en un bloque try no puede ser atrapada por uno de los manejadores catch de ese bloque try, el programa omite el resto del bloque try y el control pasa al bloque finally. Luego, el programa pasa la excepción al siguiente bloque try exterior (por lo general en el método que hizo la llamada), en donde un bloque catch asociado podría atraparla.

Como un bloque finally casi siempre se ejecuta, por lo general contiene código para liberar recursos (si haya atrapado o no la excepción.)

Tip para prevenir errores (EPT): El *bloque **finally*** es un lugar ideal para liberar los recursos adquiridos en un *bloque **try*** (como los archivos abiertos), lo cual ayuda a eliminar fugas de recursos.

Tip de rendimiento: Siempre debe liberar cada recurso de manera explícita y lo antes posible, una vez que ya no sea necesario. Esto hace que los recursos estén disponibles para que su programa los reutilice lo más pronto posible, con lo cual se mejora la utilización de recursos y el rendimiento de los programas.