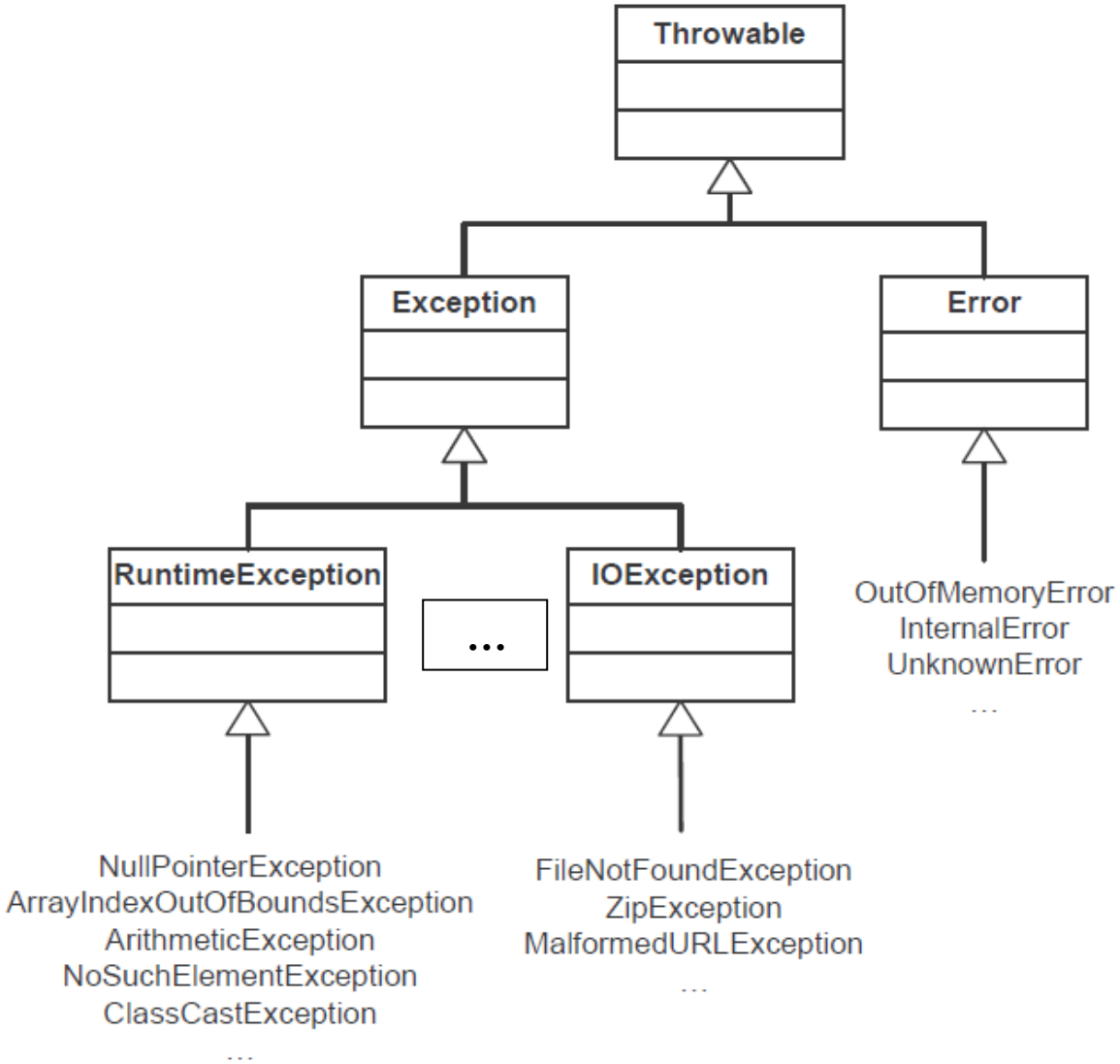
Programación Concurrente

USO DE EXCEPCIONES EN JAVA

En JAVA, cuando se produce un error en un método, "se dispara" un objeto Throwable.
Cualquier método que haya llamado al método puede "capturar la excepción" y tomar las medidas que estime oportunas.
Tras capturar la excepción, el control no vuelve al método en el que se produjo la excepción, sino que la ejecución del programa continua en el punto donde se haya capturado la excepción.

Java presenta la siguiente jerarquía de clases para el manejo de excepciones:



A continuación se explica brevemente cada una de las clases:

- THROWABLE:

  Es la clase base que representa todo lo que se puede "disparar" en JAVA.
  - Contiene una instantánea del estado de la pila en el momento en el que se creó el objeto ("stack trace" o "call chain").
  - Almacena un mensaje (variable de instancia de tipo String) que podemos utilizar para detallar que error se produjo.
  - Puede tener una causa, también de tipo Throwable, que permite representar el error que causó este error.

  Uno de los métodos definido en la clase Throwable y heredado por todas sus subclases es el método getMessage. Este método permite obtener información de la excepción.
  El siguiente ejemplo ilustra su uso:

```
try {
  C c = new C();
  c.m1();
} catch (FileNotFoundException fnfe) {
  System.out.println(fnfe.getMessage());
}
```

  En el bloque `catch` se capturan las excepciones de tipo `FileNotFoundException`, que se producen cuando se intenta abrir un archivo que no existe. En caso de que se produzca una excepción de este tipo, en la salida estándar del programa se escribirá algo como: `noexiste.dat (No such file or directory)` (suponiendo que el archivo que se intenta abrir se llame `noexiste.dat`).

  Como se puede observar, el método `getMessage` permite obtener información útil sobre la excepción. El usuario puede utilizar esta información para intentar solucionar el error. Por ejemplo, en este caso, se muestra el nombre del archivo que se intenta abrir. Esta indicación podría servir para que el usuario recuerde que es necesario que el archivo de datos utilizado por el programa tenga un nombre concreto, este ubicado en algún directorio concreto, etc.

- ERROR:

  Subclase de Throwable que indica problemas graves que una aplicación **no debería** intentar solucionar. Ejemplos: Memoria agotada, error interno de JVM….
  Las excepciones tipo ERROR son condiciones excepcionales que son externas a la aplicación (son de hardware), y por ello la aplicación no puede anticiparse y recuperarse de ellas. Por ejemplo si una aplicación abre un archivo de entrada pero no puede leer el archivo porque hay un error de hw o del sistema, se lanzaría la excepción java.io.IOError, que aun

cuando pudiera capturarse en esta ocasión sería prudente terminar la ejecución.

- EXCEPTION:

  Exception y sus subclases indican situaciones de error que una aplicación debería tratar de forma razonable.
  Los dos tipos principales de excepciones son:
  - RunTimeException (errores de programación, como una división por cero o el acceso fuera de los límites de un array).
  - IOException (errores que no puede evitar el programador, generalmente relacionados con la E/S del programa).

  La mayoría de las clases derivadas de la clase **Exception** no implementan métodos adicionales ni proporcionan más funcionalidad, simplemente heredan de **Exception**. Por ello, la información más importante sobre una excepción se encuentra en la jerarquía de excepciones, el nombre de la excepción y la información que contiene la excepción.

  Las excepciones tipo *RuntimeException*. Son condiciones internas a la aplicación, y a las que la aplicación usualmente no se puede anticipar o recuperar. Generalmente indican errores de programación, lógicos o uso inapropiado de una API. No es obligación capturarlas y tratarlas, ni especificarlas. En la mayoría de los casos es apropiado o tiene más sentido eliminar el error de programación que causa la excepción . Se podría decir que se utilizan en la fase de testeo, no en la versión final.

Captura de Excepciones: try...catch

Al trabajar con excepciones se debe considerar 3 elementos básicos:

- El bloque protegido o bloque de código en el que existe riesgo de error
- El objeto excepción, que tiene la información respecto al error producido
- El manejador de la excepción, que es el bloque de código donde se trata la excepción

En Java se utiliza ***try*** … ***catch … finally***.

El primer paso para construir un manejador de excepción es definir cuál será el código protegido, encerrándolo en un bloque ***try***. En cuanto se produce la excepción, la ejecución del bloque ***try*** termina.

La cláusula ***catch*** recibe como argumento un objeto ***Throwable,*** que es el objeto correspondiente a la excepción que se disparó. Y el código dentro del bloque ***catch*** corresponde al manejador para tratar la excepción producida.

Ejemplo 1:

```
//Bloque1
```

```
try{
    //Bloque 2
}catch (Exception error){
    //Bloque 3
}
//Bloque 4
```

La ejecución puede ser:

- Sin excepción: se ejecutará Bloque1 - Bloque 2 - Bloque 4.
- Con una excepción en el Bloque 2: se ejecutará Bloque1 – Bloque 2* - Bloque 3 - Bloque 4.
- Con una excepción en el Bloque 1: se ejecutará Bloque 1*.

Ejemplo 2:

```
//Bloque1
try{
    //Bloque 2
}catch (ArithmeticException ae){
    //Bloque 3
}catch (NullPointerException ne){
   //Bloque 4
}
//Bloque 5
```

La ejecución puede ser:

- Sin excepción: se ejecutará Bloque1 - Bloque 2 - Bloque 5.
- Excepción de tipo aritmético: se ejecutará Bloque1 – Bloque 2* - Bloque 3 - Bloque 5.
- Acceso a un objeto nulo (null): se ejecutará Bloque 1 – Bloque 2* - Bloque 4 – Bloque 5.
- Excepción de otro tipo diferente: se ejecutará Bloque 1 – Bloque 2*.

Ejemplo 3:

```
//Bloque1
try{
    //Bloque 2
}catch (ArithmeticException ae){
    //Bloque 3
}catch (Exception error){
   //Bloque 4
}
//Bloque 5
```

La ejecución puede ser:

- Sin excepción: se ejecutará Bloque1 - Bloque 2 - Bloque 5.
- Excepción de tipo aritmético: se ejecutará Bloque1 – Bloque 2* - Bloque 3 - Bloque 5.
- Excepción de otro tipo diferente: se ejecutará Bloque 1 – Bloque 2* - Bloque 4 – Bloque 5.

Es importante tener en cuenta que las cláusulas se comprueban en orden, por lo que se sugiere capturar las excepciones en orden creciente de generalidad, es decir de la más específica a la más general:

Ejemplo 4:

```
//Bloque1
try{
    //Bloque 2
}catch (Exception error){
    //Bloque 3
}catch (ArithmeticException ae){
   //Bloque 4
}
//Bloque 5
```

La ejecución puede ser:

- Sin excepción: se ejecutará Bloque1 - Bloque 2 - Bloque 5.
- Excepción de tipo aritmético: se ejecutará Bloque1 – Bloque 2* - Bloque 3 - Bloque 5.
- Excepción de otro tipo diferente: se ejecutará Bloque 1 – Bloque 2* - Bloque 3 – Bloque 5.

Por lo tanto el Bloque 4 nunca se ejecutará.

La cláusula finally

En ocasiones, nos interesa ejecutar un fragmento de código independientemente de si se produce o no una excepción (por ejemplo, cerrar un archivo en el que estemos trabajando.

Ejemplo:

```
//Bloque1
try{
    //Bloque 2
}catch (ArithmeticException ae){
    //Bloque 3
}finally{
   //Bloque 4
}
```

//Bloque 5

La ejecución puede ser:

- Sin excepción: se ejecutará Bloque1 - Bloque 2 - Bloque 4 – Bloque 5.
- Excepción de tipo aritmético: se ejecutará Bloque1 – Bloque 2* - Bloque 3 - Bloque 4 – Bloque 5.
- Excepción de otro tipo diferente: se ejecutará Bloque 1 – Bloque 2* – Bloque 4.

Si el cuerpo del bloque try llega a comenzar su ejecución, el bloque finally siempre se ejecutará…

- Después del bloque try si no se producen excepciones.
- Después del bloque catch si éste captura la excepción.
- Justo después de que se produzca la excepción si ninguna cláusula catch captura la excepción y antes de que la excepción se propague hacia arriba.

La sentencia throw

Se utiliza en Java para lanzar objetos de tipo Throwable, es decir para disparar una excepcion:

```
throw  new Exception (“Mensaje de error……”);
```

Cuando se dispara una excepción:

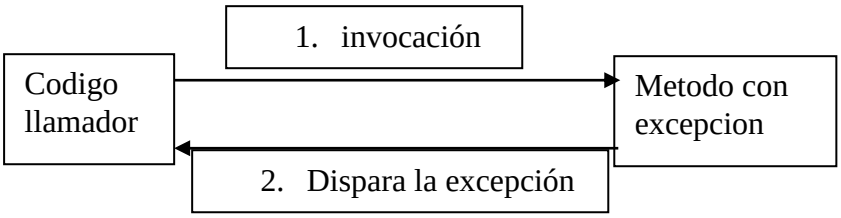
- Se sale inmediatamente del bloque de código actual.
- Si el bloque tiene asociada una cláusula catch adecuada para el tipo de la excepción generada se ejecuta el cuerpo de la cláusula catch.
- Si no, se sale inmediatamente del bloque (o método) dentro del cual está el bloque en el que se produjo la excepción y se busca una cláusula catch apropiada.
- El proceso continúa hasta llegar al método ***main*** de la aplicación. Si ahí tampoco existe una cláusula catch adecuada, la máquina virtual Java finaliza su ejecución con un mensaje de error.

Propagación de excepciones (throws)

Después que se lanza una excepción, el sistema de ejecución debe buscar el manejador para la excepción. ¿Cuales son los métodos posibles capaces de manejar la excepción? es el conjunto de métodos de la pila de llamadas del método donde ocurrió la excepción.

La búsqueda se hace hacia atrás, desde el método donde ocurrió el error hasta que encuentra el método que contiene el manejador de excepción adecuado.

Programación Concurrente



Un método captará la excepción que otro método lanza. Una excepción SIEMPRE es lanzada hacia el llamador. El método que dispara la excepción tiene que declarar que podría disparar la excepción.

Si en el cuerpo de un método se puede disparar una excepción (de un tipo derivado de la clase Exception), en la cabecera del método hay que añadir una cláusula ***throws*** que incluye una lista de los tipos de excepciones que se pueden producir al invocar el método.

Ejemplo:

```
public String leerArchivo (String nombreArchivo)  throws IOException
…
```

El compilador de Java chequea por todo tipo de excepción, excepto las RUNTIME EXCEPTION, que se consideran no chequeadas (UNCHECKED). Si un método puede disparar una excepción no chequeada, esta situación debe ser declarada con ***throws*** en la declaración del método.

Si desde el método A se invoca a un método B que puede disparar una excepción XX (o sea el método B lo declara con throws en su encabezado), en A se debe indicar que se esta atento a la posibilidad de la excepción, utilizando un bloque try/catch o try/catch/finally, o declarándola con ***throws*** en el encabezado de A (en este último caso pasándola hacia afuera en la cadena de invocaciones).

En el caso de las excepciones de tipo RuntimeException (que son muy comunes) no es necesario declararlas en la cláusula throws, ni es obligación capturarlas ya que no son verificadas por el compilador.

Al implementar un método, hay que decidir si las excepciones se propagarán hacia arriba (throws) en la cadena de llamadas o se capturarán en el propio método (catch).

1. Método que propaga una excepción:

```
public void f() throws IOException{
    //Fragmento de código que puede lanzar una excepción de tipo IOException.
}
```

   Un método puede lanzar una excepción porque cree explícitamente un objeto Throwable y lo lance con throw, o bien porque llame a un método que genere la excepción y no la capture.

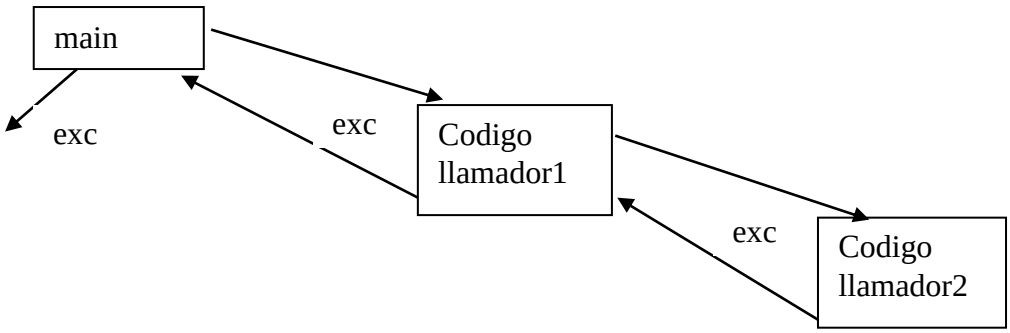
2. Método equivalente que no propaga la excepcion:

```
public void f(){
  //Fragmento de código libre de excepciones
  try{
        //Fragmento de código que puede lanzar una excepción de tipo IOException
  }catch (IOException error){
     //Tratamiento de la excepcion
  }finally{
    //liberar recursos
  }
}
```

Un método puede disparar más de una excepción, y un método puede estar preparado para capturar más de una excepción.

Cuando un método dispara más de una excepción debe separarlas con "," en la declaración del método, en la cláusula ***throws***. Cuando se tiene una jerarquía de excepciones en el método se puede declarar que dispara la excepción con el tipo más general. Cuando se considera la cláusula "catch" se puede estar preparado para capturar cualquier tipo de excepción de la jerarquía. Múltiples bloques catch deberían estar ordenados del más específico al más general. En caso contrario solo se ejecutará siempre el primer catch (correspondiente al tipo mas general)

Cuando no se quiere tratar una excepción solo hay que pasarla hacia afuera declarándola. Pero esto solo retrasa lo inevitable, tarde o temprano alguien tiene que tratarla. Pero ... ¿qué sucede si el main() la pasa?.



Creación de nuevos tipos de excepciones:

Un nuevo tipo de excepción puede crearse fácilmente: basta con definir una subclase de un tipo de excepción ya existente.

```
public ElementoInexistente extends RuntimeException {
        public ElementoInexistente(String message){
              super(message)
        }
}
```

Una excepción de este tipo puede entonces lanzarse como cualquier otra excepción:

```
public Object  recuperar (int pos) throws ElementoInexistente {
      if (pos >  this.longitud )
            throw new ElementoInexistente("Error!!!!!!!");
     return( … );
}
```

Las aplicaciones suelen definir sus propias subclases de la clase Exception para representar situaciones excepcionales específicas de cada aplicación.