UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

Manejo de excepciones

En el proceso de aprendizaje de la programación, muchas veces se omite tener en cuenta una de las partes más importantes de un programa: el chequeo de errores.

- Existe un número considerable de errores que pueden ocurrir en tiempo de ejecución. No todos los errores pueden ser detectados en tiempo de compilación.

- El mecanismo de gestión de errores de Java con el uso de excepciones, es una forma de detectar e informar los errores.

UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

Manejo de excepciones

Ejemplo de un programa en pseudocódigo

Leer de la línea de comandos el parámetro de nombre de archivo
Abrir el archivo para lectura, obteniendo un “file handle”.
While(existen datos en el archivo)
{
 leer una línea del archivo
 escribir la línea en pantalla
}
cerrar archivo

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Manejo de excepciones

Este programa funciona, pero qué problemas podrían surgir en run-time?

- El usuario especifica mal los parámetros de la línea de comandos.
- El usuario da un nombre de archivo que no existe.
- El system file table del SO fue desbordado.
- El proceso excede el número de archivos abiertos que admite el SO.
- No se puede acceder bien, para lectura.
- Se puede leer, pero el archivo está vacío.
- Ocurre un error I/O en el dispositivo que realiza la lectura.
- Etc., etc...

Se podrían utilizar mensajes de error para controlar cada una de las situaciones expuestas, pero el código se volvería complicado.

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Manejo de excepciones

¿Por qué no usar mensajes de error en lugar de excepciones?

- Devolver una indicación de error cuando sucede una situación errónea no caracteriza el problema. La lógica se complica cuando el que efectuó la llamada tiene que comprobar el mensaje de error y efectuar alguna acción.
- En un solo valor de código de error se puede codificar una cantidad de información limitada.
- Se puede generar confusión, cuando una función puede devolver de la misma forma un código de error y su propio valor de no error.
- Los constructores no devuelven valores, por lo tanto no hay forma de que devuelvan un código de error.

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Manejo de excepciones

Java utiliza la alternativa de la excepción.

- El beneficio de usar excepciones es que simplifican el código de manejo de errores. Se maneja el problema en un solo lugar del programa llamado el *exception handler*.
- Una excepción es una condición de error que interrumpe el flujo del método o del ámbito en que se encuentra la ejecución.
- En Java, ocurre una excepción cuando el programa ejecuta la palabra clave *throw.*
- La sentencia *throw* pasa el control a un *bloque de captura* asociado.

UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

Manejo de excepciones

Cuando se lanza (throw) una excepción:

- Se crea un objeto en el heap con new.
- Se corta el camino normal de la ejecución
- El mecanismo de manejo de excepciones comienza a buscar un lugar apropiado para continuar la ejecución del programa.
- Ese lugar que maneja el tratamiento de la excepción es el exception handler.

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Manejo de excepciones

Cómo se agrega Manejo de Excepciones a un programa:

- Incluya el flujo normal de instrucciones en un bloque *try*.
- Notar qué errores pueden ocurrir en run-time durante la ejecución del flujo normal de instrucciones y asegurarse que cada uno lance (*throw*) una excepción.
- Se capturan y manejan todas las excepciones en un bloque *catch*.
- Se puede construir un solo bloque catch o especializar estos bloques para distintos tipos de excepciones.

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Manejo de excepciones

Si se hubiera utilizado manejo de excepciones con *try y throw*, el programa anterior se vería:

```
try
{
  Leer de la línea de comandos el parámetro de nombre de archivo
     Chequear que el parámetro no esté vacío
     Si el parámetro está vacío, throw excepción “parámetro vacío”
  try {
    Abrir el archivo para lectura, obteniendo un file handler
  } catch (file-table-overflowed excepction) {
    esperar un número arbitrario de milisegundos
    reintentar un número de veces
   si no se puede tener acceso avisar al usuario y salir
  }
  while(existen datos en el archivo)
  {
        leer una línea del archivo
        escribir la línea en pantalla
  }
  cerrar archivo
} catch (cualquier otra excepción) {
  decir al usuario el nombre de la excepción y salir
}
```

Manejo de excepciones

Las ventajas de esta versión son:

- Es más corta que con el tradicional chequeo de errores
- Es más fácil de leer, la lógica no se interrumpe por el chequeo de errores y es más fácil de depurar.
- Permite concentrarse en el problema que se trata de resolver en un lugar y y luego manejar los errores de ese código en algún otro lugar.

La construcción del bloque try

Para comprender cómo se captura una excepción, se debe entender:

- Existe una sección de código que puede producir excepciones
- Seguida del código para manejar esas excepciones.

El bloque **try** es un ámbito ordinario precedido por la palabra clave *try*. Engloba una serie de sentencias desde las cuales puede originarse un lanzamiento de excepción.

```
try {
  // código que puede generar excepciones
}
```

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Manejo de excepciones

Exceptions handlers

Los bloques de captura siguen inmediatamente al bloque try y se denotan por la palabra clave *catch.* Es una sección de código diseñada para procesar una excepción lanzada (throw).

```
try {
  // código que puede genrar excepciones
} catch (Tipo1Excepción Id) {
  //maneja excepciones Tipo1
} catch (Tipo2Excepción Id) {
  //maneja excepciones Tipo2
} //....
```

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Manejo de excepciones

```
public class FileReader1
{
    public static void main(String[] args) {
  try {
        //Lee los parámetros de la línea de comandos
        // si parámetro vacío, throw la excepción "parámetro vacío"
        if(args.length == 0)
              throw(new Exception ("Uso: FileReader1 filename"));
        // Se abre el archivo para lectura, obteniendo un file handle
        java.io.BufferedReader theStream = new
            java.io.BufferedReader (new java.io.FileReader(args[0]));
        //Lee una línea del archivo
        String theInputString = new String();
        while((theInputString = theStream.readLine()) != null)
        {
              System.out.println(theInputString);//escribe la línea en pantalla
        }
        theStream.close();// cerrar el archivo
  } catch(Exception e) {
        System.err.println(e.getMessage());
        e.printStackTrace();
     }
  }
}
```

UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

Manejo de excepciones

Creación de las propias excepciones

En Java, hay definidas un gran número de clases de excepción. Sin embargo se puede definir la propia clase de excepción para ajustarse a las propias necesidades.

Pasos a seguir:

1. Revisar la jerarquía de clases de Java (docs/api/tree.html en el directorio JDK).
2. Si no existe una clase de Excepción JDK que satisfaga las necesidades, desarrollar una convención de nombres o de package que distinga las propias excepciones de las demás.
3. Agrupar las excepciones relacionadas en una jerarquía propia, derivada desde la más específica clase de Excepción de la jerarquía de JAVA.
4. Escribir dos constructores para cada clase nueva de excepción; uno sin argumentos y otro que tome un String como argumento.
5. Agregar cualquier otro constructor, datos miembro y métodos que se crean apropiados.

UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

Manejo de excepciones

Creación de las propias excepciones

En Java, hay definidas un gran número de clases de excepción. Sin embargo se puede definir la propia clase de excepción para ajustarse a las propias necesidades.

Pasos a seguir:

1. Revisar la jerarquía de clases de Java (docs/api/tree.html en el directorio JDK).
2. Si no existe una clase de Excepción JDK que satisfaga las necesidades, desarrollar una convención de nombres o de package que distinga las propias excepciones de las demás.
3. Agrupar las excepciones relacionadas en una jerarquía propia, derivada desde la más específica clase de Excepción de la jerarquía de JAVA.
4. Escribir dos constructores para cada clase nueva de excepción; uno sin argumentos y otro que tome un String como argumento.
5. Agregar cualquier otro constructor, datos miembro y métodos que se crean apropiados.

Manejo de excepciones

Para crear las propias excepciones, se debe heredar de algún tipo existente de excepción. La forma más trivial es:

```
class ExcepcionSimple extends Exception {}
public class MiClaseExcepcion
{
 public void f() throws ExcepcionSimple {
        System.out.println( "Throwing Excepcion Simple desde f()");
        throw new ExcepcionSimple();
 }
 public static void main(String[] args) {
        MiClaseExcepcion mce = new MiClaseExcepcion();
        try {
              mce.f();
        } catch(ExcepcionSimple e) {
              System.err.println("Capturada!!!!");
        }
 }
}
```

La salida en el Terminal Window de BlueJ es:

```
Throwing Excepcion Simple desde f()
```

En la consola de standard error de Java se imprime el resultado:

```
Capturada!!!
```

UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

Manejo de excepciones

También podríamos crear una clase de excepción que tome dos constructores: uno sin argumentos y uno con un argumento String.
Esto se muestra en el siguiente ejemplo:

```
class ExcepcionConst extends Exception {
 public ExcepcionConst() {}
 public ExcepcionConst(String m) {
      super(m);
 }
}
```

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Manejo de excepciones

```
public class DosConstructores {
  public void f() throws ExcepcionConst {
     System.out.println( "Throwing Excepcion con Const desde f()");
     throw new ExcepcionConst();
  }
  public void g() throws ExcepcionConst {
     System.out.println( "Throwing Excepcion con Const desde g()");
     throw new ExcepcionConst("originada en g()");
  }
  public static void main(String[] args) {
     DosConstructores dc= new DosConstructores();
     try {
        dc.f();
     } catch(ExcepcionConst e) {
     e.printStackTrace(System.err);
     }
     try {
        dc.g();
     } catch(ExcepcionConst e) {
        e.printStackTrace(System.err);
     }
}}
```

Manejo de excepciones

La salida en Terminal Window de BlueJ será:

Throwing Excepcion con Const desde f()
Throwing Excepcion con Const desde g()

El mensaje de error lo muestra en la consola standard error de Java.

Manejo de excepciones

Consideremos el caso de una clase Cuenta Bancaria. Supongamos que se ha extendido un cheque por más dinero que el efectivo en la cuenta, la función Retiro ignora la petición

```
class ExcepcionFondosInsuficientes extends Exception
{
 private CuentaBExcepcion _ba;
 private double _cantRetiro;
 ExcepcionFondosInsuficientes(CuentaBExcepcion ba, double cant)
 {
        super("No hay fondos suficientes en la cuenta");
        _ba=ba;
        _cantRetiro = cant;
 }
 public String toString()
 {
        StringBuffer sb = new StringBuffer();
        sb.append("No hay suficientes fondos en la cuenta");
        sb.append(_ba.id());
        sb.append("\nEl saldo era ");
        sb.append(_ba.saldo());
        sb.append("\nEl retiro era ");
        sb.append(_cantRetiro);
        return sb.toString();
 }}
```

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Manejo de excepciones

```
public class CuentaBExcepcion
{
 private static int aids = 0;
 private double _interes;
 private double _saldoActual;
 private int _nroCuenta; // el nro de cuenta del objeto actual

 // Constructor de la clase con saldo inicial
 CuentaBExcepcion(double saldoInicial)
 {
      _nroCuenta = ++aids; // asigna a este objeto el próximo nro
      _saldoActual = saldoInicial;
 }
 // devuelvo el identificador de la cuenta
 public int id()
 {
      return _nroCuenta;
 }
```

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Manejo de excepciones

```
void retiro(double cant) throws ExcepcionFondosInsuficientes
{
    if(_saldoActual < cant)
    {
        //lanzo una excepcion
        throw new ExcepcionFondosInsuficientes(this, cant);
    }
    _saldoActual -= cant;
}
double saldo()
{
// devuelvo el saldo actual redondeando al centavo mas próximo
    int nCents = (int)( _saldoActual * 100 + 0.5);
    return nCents / 100.0;
}
}
```

Manejo de excepciones

Escribimos una clase de prueba, forzando a que se produzca la excepción.

```
public class TestCuentaBExcepcion
{
  public static void main(String[] args) {
        try {
              //abro una cuenta bancaria con 50
              CuentaBExcepcion ba = new CuentaBExcepcion(50);

              //trato de sacar 100 pesos
              ba.retiro(100.00);

              System.out.println("\u00ad Retiro efectuado!");
        }
        catch(Exception e){
              //salida del mensaje de error de la excepción
              System.out.println(e.toString());
        }
  }
}
```

La salida será:

No hay suficientes fondos en la cuenta # 1
El saldo era 50.0
El retiro era 100.0

Manejo de excepciones

Gestión de múltiples excepciones

Un solo método puede lanzar más de una excepción. Podemos modificar el ej. anterior, suponiendo que debo encontrar una cuenta por su identificador.
Agrego un método de excepción para tratar la posibilidad de que no se encuentre la cuenta.

```
class ExcepcionNroCuenta extends Exception  //nueva Excepción
{
 ExcepcionNroCuenta(int nroCtaId) {
      super("Cuenta no encontrada "+ nroCtaId);
 }
}
class ExcepcionFondosInsuficientes extends Exception
{
 // Igual que en el ej. anterior
}
```

Manejo de excepciones

```
public class CuentaBExcepcion2
{
 // Todos los métodos del ejemplo anterior
 // El método find busca una cuenta por su id
public static CuentaBExcepcion2 find(int nroCuentaId) throws
ExcepcionNroCuenta {
 // procesos para encontrar la cuenta por su identificador
 // si no la encuentra, lanza una excepción
 }
 void retiro(int nroCuentaId, double cant) throws ExcepcionNroCuenta,
ExcepcionFondosInsuficientes
 {
        CuentaBExcepcion2 b = find(nroCuentaId);
        if(_saldoActual < cant)
        {
              //lanzo una excepcion
              throw new ExcepcionFondosInsuficientes(b, cant);
        }
        _saldoActual -= cant;
 }
}
```

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Manejo de excepciones

Especificación de la excepción

Cuando se escribe un método, se debe informar sobre las excepciones que pueden ser lanzadas (thrown) por ese método, mediante una especificación de excepción, que es parte de la declaración del método y aparece después de la lista de argumentos:

```
void f() throws DivPorCero, FondosInsuficientes { //....
```

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Manejo de excepciones

Captura de la excepción

Es posible crear un handler que capture cualquier tipo de excepción.
Esto se hace especificando en el argumento del método de captura, la clase base Exception, para que pueda recibir cualquier tipo de excepción que hereda de ella.

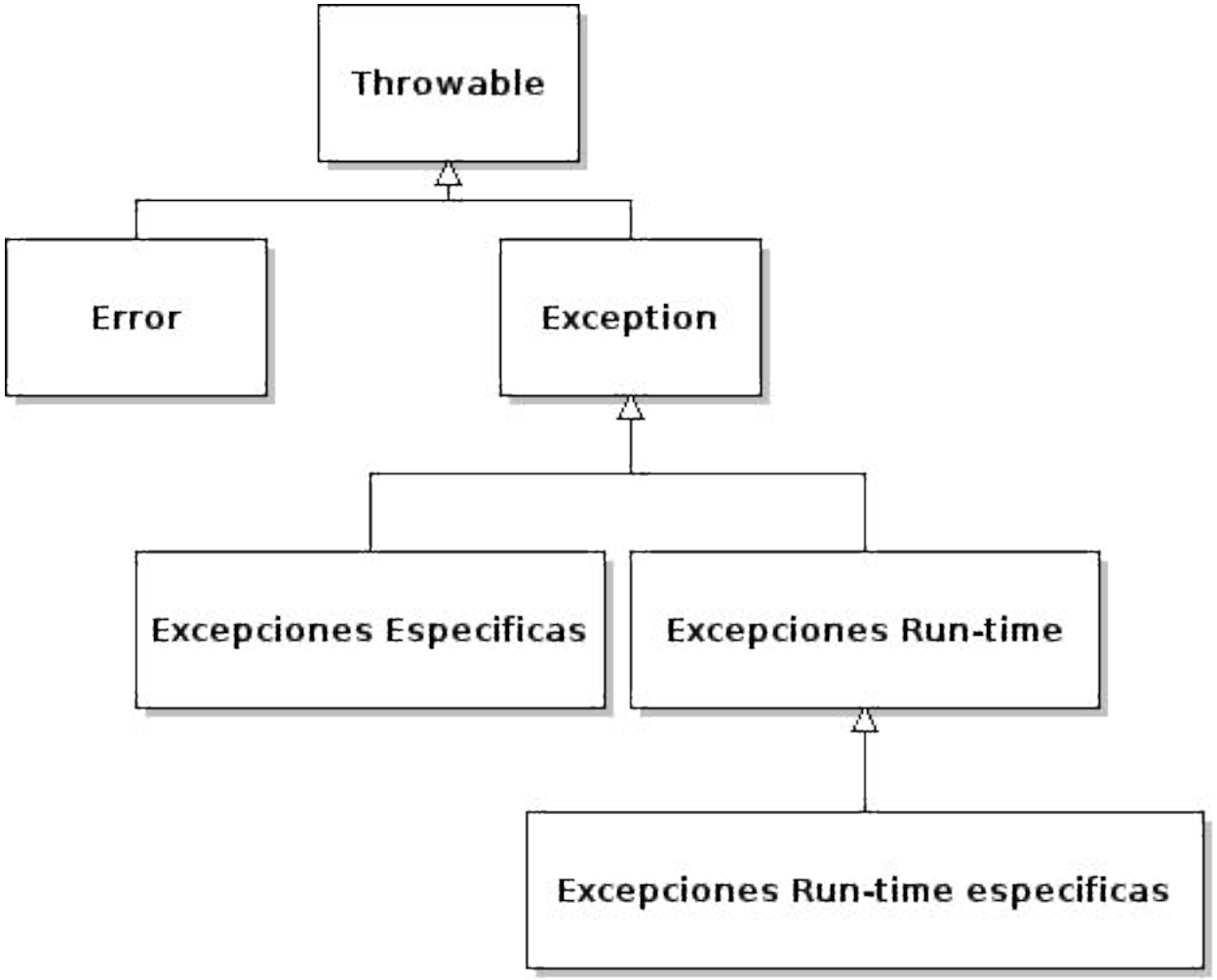
```
catch(Exception e) {
  System.err.println(“Capturada”);
}
```

Un método general de captura como este, se coloca al final de todos los handlers específicos, para evitar que se ejecute éste en lugar del deseado.

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Manejo de excepciones

Jerarquía de las excepciones: La clase Exception es del tipo Throwable. Existen muchas clases de Java derivadas de la clase Exception.



UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Manejo de excepciones

- Throwable es la clase base de Error y Exception.
- Describe tipos que pueden “lanzarse” mediante sentencias throw.
- No deben tratar de captarse Errors mediante bloques catch o declararse, ya que representan errores compile-time o del sistema.
- La excepción es el tipo básico que puede ser lanzado desde cualquier método de la librería standard de Java o desde los propios métodos.
- Existe una variedad de excepciones en la librería estándar. Cada nombre representa el problema que ocurre.
- Se pueden invocar los siguientes métodos, heredados de la clase base Throwable:

String getMessage()
String getLocalizedMessage()
Retorna el mensaje detallado o ajustado a particularidades locales

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Manejo de excepciones

String toString()
Retorna una breve descripción del Throwable incluyendo el mensaje detallado, si hay alguno.

void printStackTrace()
void printStackTrace(PrintStream)
void printStackTrace(PrintWriter)
Imprime el Throwable y el rastro de llamadas a Throwable en la pila.
La primera versión imprime en al standard error, la segunda y tercera a un stream elegido.

Throwable fillInStackTrace()
Registra información dentro de este objeto Throwable acerca del estado actual de la pila.

Manejo de excepciones

El siguiente ejemplo muestra el uso básico de estos métodos:

```
public class MetodosException
{
 public static void main(String[] args) {
      try{
           throw new Exception("Esta es mi excepcion");
      } catch (Exception e) {


System.err.println("Excepcion capturada");
           System.err.println("e.getMessage(): "+ e.getMessage());
           System.err.println("e.getLocalizedMessage(): "+
e.getLocalizedMessage());
           System.err.println("e.toString(): "+ e);
           System.err.println("e.printStackTrace(): ");
           e.printStackTrace(System.err);
      }
 }
}
```

Manejo de excepciones

La salida en la standard error de Java será:

```
Excepcion capturada
e.getMessage(): Esta es mi excepcion
e.getLocalizedMessage(): Esta es mi excepcion
e.toString(): java.lang.Exception: Esta es mi excepcion
e.printStackTrace():
java.lang.Exception: Esta es mi excepcion
        at MetodosException.main(MetodosException.java:12)
        at __SHELL0.run(__SHELL0.java:19)
        at
bluej.runtime.ExecServer.suspendExecution(ExecServer.java:125)
        at bluej.runtime.ExecServer.main(ExecServer.java:69)
```

Manejo de excepciones

- El compilador tampoco requiere que se capturen o declaren excepciones que son de tipo Runtime. La clase RuntimeException es un caso particular, ya que será encargada de chequear gran parte de los problemas en tiempo de ejecución, para aliviar al programador de este trabajo. Por ej. en el siguiente código:

```
if(t == null)
 throw new NullPointerException();
```

No podría pensarse en chequear todas las referencias nulas pasadas a un método. Esto lo hace la rutina de chequeos estándar de Java. Hay un extenso grupo de tipos de excepción en esta categoría, agrupados en una clase base llamada RuntimeException.
No hace falta decirle a un método que debe lanzar una excepción de este tipo, ni capturarlas en algún bloque.

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Manejo de excepciones

Utilización de finally

El control sale de un bloque try tan pronto como el programa ejecuta un retorno o se lanza una excepción. Java permite al usuario definir un bloque de código para que sea ejecutado antes de que el control salga del método.
Este bloque se llama de finalización (finally). Se define usando la palabra finally al final de todos los exceptions handlers.
Finally se utiliza cuando se necesita que ciertas actividades se realicen siempre, tanto si se dispara o no una excepción, que sean distintas de limpieza de memoria (funciona el garbagge collector).

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Manejo de excepciones

```
try {
 // Actividades que pueden disparar a A,B o C
 // este proceso se aborta cuando se lanza una excepción o se
// encuentra un retorno
} catch(A a) {
 // manejador de situaciones A
} catch(B b) {
      // manejador de situaciones B
} catch(C c) {
      // manejador de situaciones C
} finally {
 // actividades que siempre suceden
 //este bloque se ejecuta antes de que el control salga del método
 //independientemente que el bloque try incluya un retorno o
 //lance una excepción.
}
```

Manejo de excepciones

Rethrowing

A veces se quiere re-lanzar una excepción, es decir enviarla a los exception handlers en el siguiente contexto más alto que el actual:

```
catch(Exception e)
 System.err.println("Se lanza una excepción");
 throw e;
}
```

Se ignoran todas las cláusulas catch siguientes para el mismo bloque try (se ejecuta el finally si existe).
Se preserva la información del objeto, de modo que el handler en el siguiente contexto que captura el tipo específico de excepción, puede extraer toda la información de ese objeto.
Si simplemente se relanza la excepción actual, la información que se imprime sobre la excepción en **printStackTrace(),** pertenecerá al origen de la excepción, no al lugar donde se relanza.

Manejo de excepciones

Si se quiere instalar nueva información sobre el rastro de la pila, se hace llamando **fillInStackTrace()** que retorna un objeto Exception que crea llenando el viejo objeto Exception con información de la pila actual.

```
public class Rethrowing {
 public static void f() throws Exception{
    System.out.println("se origina la excepción en f()");
    Throw new Exception("lanzada desde f()");
 }
 public static void g() throws Throwable {
    try {
         f();
    } catch(Exception e) {
      System.err.println("Dentro de g(), e.printStackTrace()");
    e.printStackTrace(System.err);
    throw e; // línea 12
    }
```

Manejo de excepciones

```
public static void main(String[] args) throws Throwable {
   try {
      g();
   } catch(Exception e) {
         System.err.println ("Capturada en main, e.printStackTrace()");
      e.printStackTrace(System.err);
}
```

Con la línea 12 escrita como está, la salida sería:

```
se origina la excepción en f()
Dentro de g(), e.printStackTrace()
Java.lang.Exception: thrown from f()
 at Rethrowing.f(Rethrowing.java:4)
 at Rethrowing.g(Rethrowing.java:8)
 at Rethrowing main(Rethrowing.java:16)
Capturada en main, e.printStackTrace()
Java.lang.Exception: thrown from f()
 at Rethrowing.f(Rethrowing.java:4)
 at Rethrowing.g(Rethrowing.java:8)
 at Rethrowing main(Rethrowing.java:16)
```

Manejo de excepciones

Como se observa, el rastro de la pila de excepción siempre recuerda su punto de origen.

Si cambiamos la línea 12 de la forma:
throw e.fillInStackTrace();

el resultado es:

```
se origina la excepción en f()
Dentro de g(), e.printStackTrace()
Java.lang.Exception: thrown from f()
 at Rethrowing.f(Rethrowing.java:4)
 at Rethrowing.g(Rethrowing.java:8)
 at Rethrowing main(Rethrowing.java:16)
Capturada en main, e.printStackTrace()
Java.lang.Exception: thrown from f()
 at Rethrowing.g(Rethrowing.java:12)
 at Rethrowing main(Rethrowing.java:16)
```

UNTDF
Universidad Nacional de Tierra del Fuego,
Antártida e Islas del Atlántico Sur

Manejo de excepciones

Por la existencia de fillInStackTrace(), la línea 12 se vuelve el nuevo punto de origen de la excepción.

También es posible relanzar una excepción diferente de la que se capturó.

Hacer esto tiene el mismo efecto que usar fillInStackTrace(): se pierde la información acerca del sitio original de la excepción y lo que se deja es la información perteneciente al nuevo throw.