UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Manejo de excepciones en Java

ELO329: Diseño y Programación Orientados a Objetos

Agustín J. González

Excepciones

- Una **excepción** es un evento que ocurre durante la ejecución de un programa que rompe el flujo normal de ejecución. Cuando se habla de excepciones nos referimos a **evento excepcional**.
- Muchas cosas pueden generar excepciones: Errores de hardware (falla de disco), de programa (acceso fuera de rango en arreglo), apertura de archivo inexistente, etc.
- Cuando se produce una excepción dentro de un método de Java, éste crea un objeto que contiene información sobre la excepción y lo pasa al código llamador.
- La rutina receptora de la excepción es responsable de reaccionar a tal evento inesperado.
- Cuando creamos un objeto para la excepción y lo pasamos al código llamador decimos que lanzamos una excepción (Throw an exception)
- Si el método llamador no tiene un manejador de la excepción se busca hacia atrás en la pila de llamados anidados hasta encontrarlo.
- Decimos que el manejador atrapa la excepción (catch the exception)

Ventajas de usar excepciones: Separar código de casos de error

- Supongamos que queremos hacer la tarea:

```
readFile {
    open the file;
    determine its size;
    allocate that much memory;
    read the file into memory;
    close the file;
}
```

Ventajas de usar excepciones: Separar código de casos de error

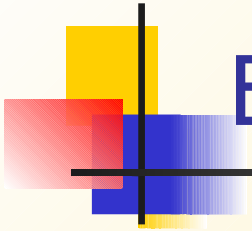
- Sin excepciones debemos hacer algo así:

```
errorCodeType readFile {
   initialize errorCode = 0;
   open the file;
   if (theFileIsOpen) {
      determine the length of the file;
      if (gotTheFileLength) {
         allocate that much memory;
         if (gotEnoughMemory) {
            read the file into memory;
            if (readFailed) {
               errorCode = -1;
            }
         } else {
            errorCode = -2;
         }
      } else {
         errorCode = -3;
      }
      close the file;
      if (theFileDidntClose && errorCode == 0) {
         errorCode = -4;
      } else {
         errorCode = errorCode and -4;
      }
   } else {
      errorCode = -5;
   }
   return errorCode;
}
```

Ventajas de usar excepciones: Separar código de casos de error

- Con excepciones:

```
readFile {
    try {
        open the file;
        determine its size;
        allocate that much memory;
        read the file into memory;
        close the file;
    } catch (fileOpenFailed) {
        doSomething;
    } catch (sizeDeterminationFailed) {
        doSomething;
    } catch (memoryAllocationFailed) {
        doSomething;
    } catch (readFailed) {
        doSomething;
    } catch (fileCloseFailed) {
        doSomething;
    }
}
```



Excepciones

- Otras ventajas de las excepciones son:
  - Propaga los errores hacia atrás en la secuencia de llamados anidados.
  - Se agrupan los errores según su naturaleza.
  - Ej:
    - Si hay más de un archivo el que se abre, basta con un código para capturar tal caso.
    - Si se lanzan excepciones que son todas subclases de una base, basta con capturar la base para manejar cualquiera de sus instancias derivadas.
- En Java los objetos lanzados deben ser instancias de clases derivadas de Throwable.
  Ej.
  Throwable e = new IllegalArgumentException("Stack underflow");
  throw e;
  O alternativamente
  throw new IllegalArgumentException("Stack underflow");
- Si una excepción no es procesada, debe ser relanzada.

Manejo de Excepciones

- El manejo de excepciones se logra con el bloque try

```
try {
        // código
} catch (StackError e )
{
        // código que se hace cargo del error reportado en e
}
```

- El bloque try puede manejar múltiples excepciones:

```
try {
        // código
} catch (StackError e )
{
        // código para manejar el error de stack
} catch (MathError me)
{
        // código para manejar el error matemático indicado en me.
}
```

Captura de Excepciones (completo)

- El bloque try tiene la forma general:

```
try {
        //sentencias
} catch (e-type1 e ) {
        // sentencias
} catch (e-type2 e ) {
        // sentencias
} ...
finally {
        //sentencias
}
```

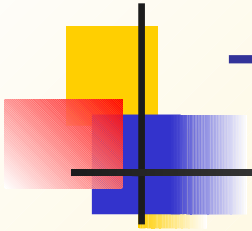
- La cláusula finally es ejecutada con posterioridad cualquiera sea la condición de término del try (sin o con error). Esta sección permite dejar las cosas consistentes antes del término del bloque try.

Captura de Excepciones: Ejemplo 1

```
public static void doio (InputStream in, OutputStream out) {
        int c;
        try { while (( c=in.read()) >=0 )
                { c= Character.toLowerCase( (char) c);
                   out.write( c );
                }
        } catch ( IOException e ) {
          System.err.println("doio: I/O Problem");
          System.exit(1);
        }
}
```

Captura de Excepciones: Ejemplo 2

- ```
  ......
  try { FileInputStream infile = new FileInputStream(argv[0]);
        File tmp_file = new File(tmp_name);
        ....
  } catch (FileNotFoundException e) {
        System.err.println("Can´t open input file "+ argv[0]);
        error = true;
  } catch (IOException e ) {
        System.err.println("Can´t open temporary file "+tmp_name);
        error = true;
  }finally {
        if ( infile != null) infile.close();
        if (tmp_file != null) tmp_file.close();
        if (error) System.exit();
  }
  ```
- El código de la sección finally es ejecutado no importando si el bloque try terminó normalmente, por excepción, por return, o break.
  ```



Tipos de Excepciones

- Las hay de dos tipos
- Aquellas generadas por el lenguaje Java. Éstas se generan cuando hay errores de ejecución, como al tratar de acceder a métodos de una referencia no asignada a un objeto, división por cero, etc.
- Aquellas no generadas por el lenguaje, sino incluidas por el programador.
- El compilador chequea por la captura de las excepciones lanzadas por los objetos usados en el código.
- Si una excepción no es capturada debe ser relanzada.

Reenviando Excepciones

```
public static void doio (InputStream in, OutputStream out)
        throws IOException  // en caso de más de una excepción throws exp1, exp2
{
        int c;
        while (( c=in.read()) >=0 )
        {
                c= Character.toLowerCase( (char) c);
                out.write( c );
        }
}
```

Si la excepción no es capturada, se entiende reenviada

- **Alternativamente:**

```
public static void doio (InputStream in, OutputStream out) throws Throwable {
        int c;
        try { while (( c=in.read()) >=0 )
                { c= Character.toLowerCase( (char) c);
                  out.write( c );
                }
        } catch ( Throwable t ) {
                throw t;
        }
}
```

En este caso el método envía una excepción - que aquí corresponde al mismo objeto capturado -por lo tanto debe declarase en la cláusula throws.

- **!!! Si el método usa la cláusula throw debe indicarlo en su declaración con la cláusula throws.**

Creación de tus propias excepciones

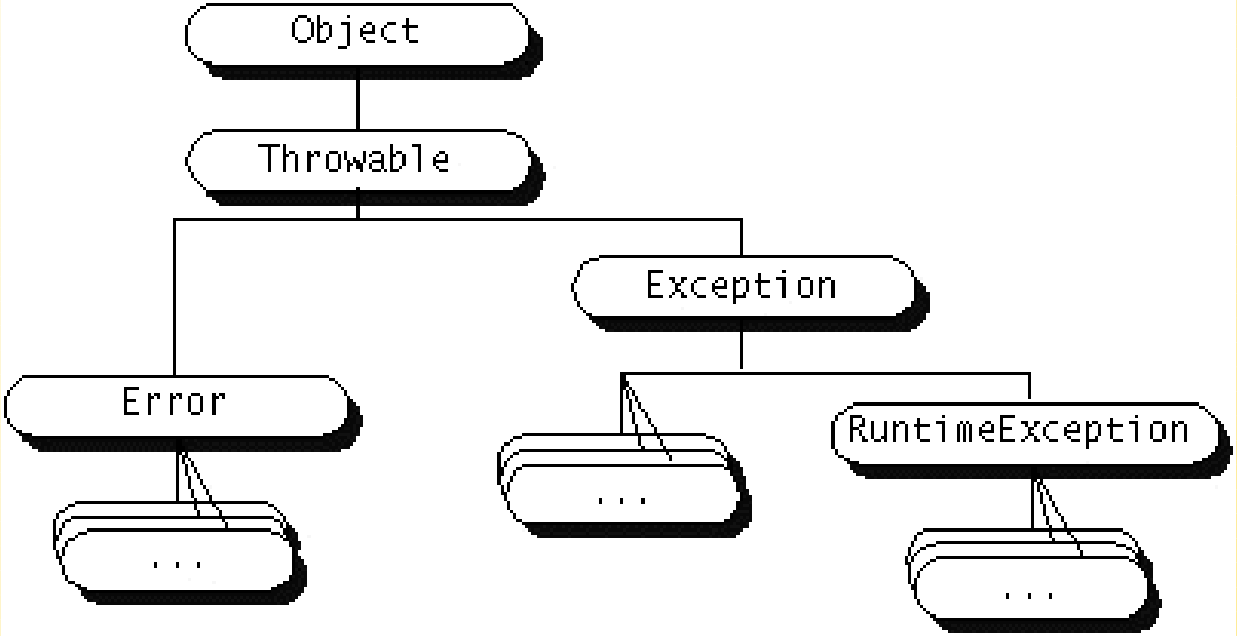
- **Siempre es posible lanzar alguna excepción de las ya definidas en Java (IOException por ejemplo).**
- **También se puede definir nuevas excepciones creando clases derivadas de las clases Error o Exception.**
- **class ZeroDenominatorException extends Exception**

```
{       private int n;
        public ZeroDenominadorException () {}
        public ZeroDenominadorException(String s) {
                super(s);
        }
        public setNumerator(int _n) { n = _n}
        // otros métodos de interés
}
```

- **Luego la podemos usar como en:**

```
....
public Fraction (int n, int d) throws ZeroDenominatorException {
        if (d == 0) {
                ZeroDenominatorException myExc = new
                 ZeroDenominatorExceptio("Fraction: Fraction with 0
denominator?");
myExc.setNumerator(n);
throw (myExc);
....
}
```

Jerarquía de Excepciones



- Java prohibe crear subclases de Throwable.
- Cuando creemos nuestras excepciones, serán subclases de Exception. Más sobre esto.
- Java no obliga a manejar o reenviar RuntimeException.

Cuando no podemos relanzar una excepción

- Hay situaciones en que estamos obligados a manejar una excepción. Consideremos por ejemplo:

```
class MyApplet extends Applet {
        public void paint (Graphics g) {
                FileInputStream in = new FileInputStream("input.dat"); //ERROR
                ....
        }
}
```

- Se crea aquí un problema porque dado que la intención es sobremontar un método de la clase Applet - método paint- el cual no genera ninguna excepción. Si un método no genera excepciones la función que lo sobremonta no puede lanzar excepciones (problema en Java).
- Lo previo obliga a que debamos hacernos cargos de la excepción.

```
 class MyApplet extends Applet {
        public void paint (Graphics g) {
          try {
                FileInputStream in = new FileInputStream("input.dat"); //ERROR
                ....
        } catch (Exception e) {//.....}
}
```