Java

Programación Orientada a Objetos

Alfredo Raúl Teyseyre

UNICEN

Java

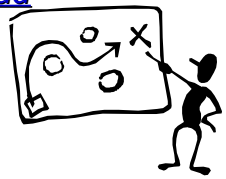
Tratamiento de Excepciones

Tratando con los Errores

El momento ideal para encontrar los problemas es en tiempo de compilación

pero..... *no todos los problemas se detectan o surgen en tie*

Se necesita contar con un mecanismo que en tiempo de ejecución permita al desencadenante del error pasar la información apropiada a alguien que sepa resolver esa dificultad adecuadamente.



Excepciones

Un evento que ocurre durante la ejecución de un programa que altera el flujo normal de las instrucciones

- Muchos tipos de errores pueden causar excepciones
  - Errores serios de hardware:rotura de disco
  - Simples errores de programación: acceso fuera de los limites de un arreglo
- Cuando un error ocurre se crea un objeto excepción
  - Representadas por instancias de la clase *java.lang.Exception*.
  - Creadas y disparadas cuando la condición ocurre
  - Pasadas al manejador apropiado
  - Pueden tener información sobre la condición

Beneficios del mecanismo de tratamiento de Excepciones de Java

- Separa el código funcional del código de manejo de errores
- Define un camino claro de propagación de errores
  - Si el método invocado encuentra una situación que no puede tratar, puede propagar una excepción y dejar que el método que lo invocó la trate
- Permite agrupar y diferenciar los tipos de error
- Soporte robusto: el compilador verifica que se traten las excepciones

Separa el código funcional del código de manejo de errores

Si se tratan los errores de forma tradicional su detección, reporte y recuperación usualmente llevan a un código spaghetti

- Ejemplo: lectura de un archivo

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

- A primera vista esta función es muy simple, pero ignora todos estos errores potenciales:
  - ¿ Que pasa si no se puede abrir el archivo?
  - ¿ Que pasa si no se puede determinar el tamaño del archivo?
  - ¿ Que pasa si no hay suficiente memoria?
  - ¿ Que pasa si falla la lectura?
  - ¿ Que pasa si no se puede cerrar el archivo?

Separa el código funcional del código de manejo de errores

....código spaghetti

```
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;}
```


Separa el código funcional del código de manejo de errores

Con Excepciones en Java

```
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;
    }
}
```

Propagación de Errores

- Supongamos que el método method1 es el único de los métodos interesado en los errores que ocurren en readFile.

```
method1 {
    call method2;
}
method2 {
    call method3;
}
method3 {
    call readFile;
}
```


Propagación de Errores

Sin Excepciones

```
method1 {
    errorCodeType error;
    error = call method2;
    if (error)
      doErrorProcessing;
    else proceed;}

errorCodeType method2 {
    errorCodeType error;
    error = call method3;
    if (error) return error;
    else proceed;}

errorCodeType method3 {
    errorCodeType error;
    error = call readFile;
    if (error)
        return error;
    else proceed;}
```

Con Excepciones

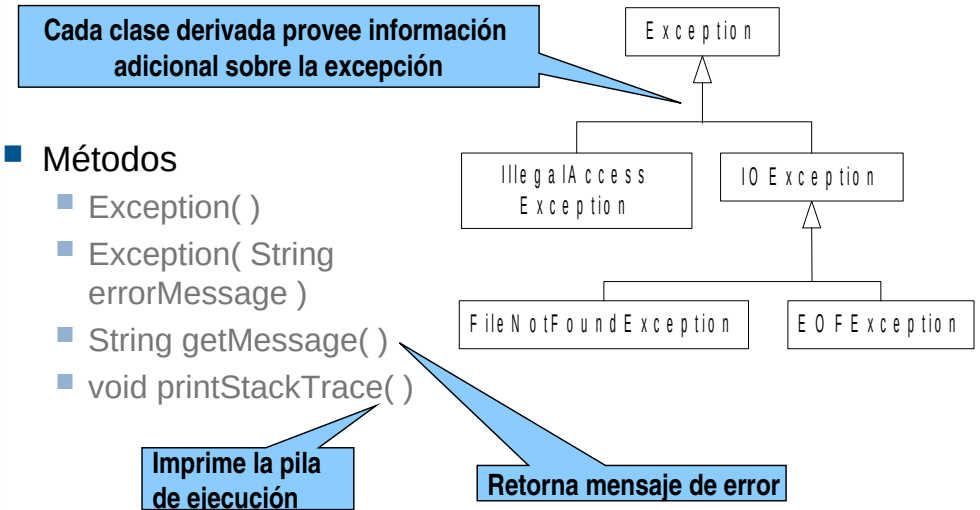
```
method1 {
    try {
        call method2;
    } catch (exception) {
        doErrorProcessing;
    }}

method2 throws exception {
    call method3;
}

method3 throws exception {
    call readFile;
}
```


Tipos de Excepciones

Señal indicando que una condición inusual ha ocurrido



- Métodos
  - Exception( )
  - Exception(String errorMessage)
  - String getMessage( )
  - void printStackTrace( )

Utilizando las Excepciones

- Capturando las excepciones propagadas por los métodos
- Escribiendo métodos que declaran y disparan excepciones
- Definiendo subclases de Exception

Capturando Excepciones

- Encapsule el código que propaga excepciones utilizando try/catch/finally
- El código puede propagar variadas excepciones

```
void readFile(String fileName) {
    RandomAccessFile file;
    try {
      file = new RandomAccessFile(fileName, "r");
        while((line = file.readLine()) != null)
        ...
    }
    catch (EOFException e) { ... }
    catch (IOException e) { … }
    catch (Exception e) { … }
    finally {
        try{ file.close( ); }
        catch (IOException e){ e.printStackTrace(); }
    }
}
```

Opcional: para liberar recursos (siempre se ejecuta)

Los bloques catch se ordenan desde los más específicos a los menos específicos

Propagando Excepciones

Un método debe listar las excepciones que emite

```
public void readFile (String fileName)
        throws IOException, FileNotFoundException {
    char input = '\0';
    FileInputStream file = new FileInputStream ( fileName );
    while ((input = (char)file.read( )) != -1)    {    //clip….    }
}
```

Declara excepciones no tratadas

Estas líneas propagan excepciones no tratadas por el método

- *throws* difiere el tratamiento de los errores al método invocante
  - El método que invoca *readfile* debe tratar la excepción o listar la excepción en la clausula *throws*.

```
public void doit( ) {
    try{
        readFile( "filename.txt" );
    } catch( IOException e )    { … }
}
```


Información de la Excepción

La excepción contiene información útil

```
try {
    ...
    file.readLine( );
    ...
}
catch (IOException e)  {
    System.out.println(e);       // imprime la excepción
    String msg = e.getMessage( );     // mensaje de la
                                      //excepción
    e.printStackTrace( ); // imprime la pila de
                          // ejecución}
```


Creando una nueva Excepción

- Extienda la clase Exception o alguna de sus subclases

```
class IllegalDateException extends Exception  {
    public IllegalDateException( ) { super( ); }
    public IllegalDateException (String s) { super(s); }        }
```

- Disparando una excepción

```
class Employee {
    Date dateOfBirth;        // instance field
    ...
public void setDateOfBirth(Date date) throws IllegalDateException {
        Date today = new Date( );
        if ( date.after (today) )     // nació en el futuro
            throw new IllegalDateException ("Invalid DOB: "+ date);
        dateOfBirth = date;
    }
}
```

Fecha Actual

Propaga excepción.

Creando una nueva Excepción

```
public Employee makeEmp( Date d ) {
    Employee emp = new Employee( );

    try  {
       emp.setDateOfBirth( d );
    }
    catch( IllegalDateException e )    {
      emp.setDateOfBirth( new Date( ) );
      e.printStackTrace( );
    }
    return e;
}
```


Redisparando una Excepción

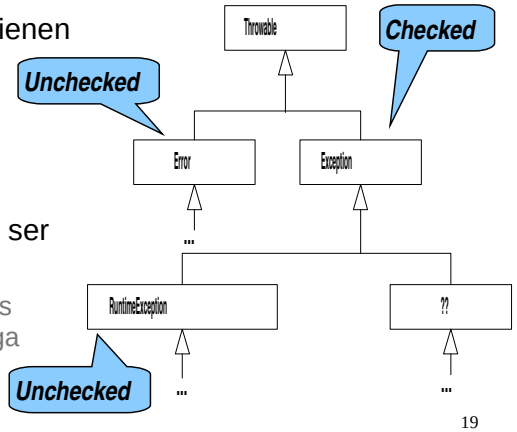
- Dentro de un bloque catch se puede repropagar una excepción
- Obliga al invocante a tratar la excepción nuevamente

```
try {
  ...
}
catch (IOException e) {
    // do some exception handling...
    throw e;
}
```


Excepciones Checked y Unchecked

Throwable Class Hierarchy		Checked
Error and its subclasses		no
Exception	**RuntimeException** and its subclasses	no
	All other **Exception** subclasses	yes

- Las excepciones Unchecked no tienen que ser tratadas
  - Error - Interno, irrecuperable
  - RuntimeException - "Falla del Programador"
- Las excepciones Checked deben ser tratadas o repropagadas
  - El método debe declarar todas las excepciones checked que propaga
  - Verificado por el Compilador



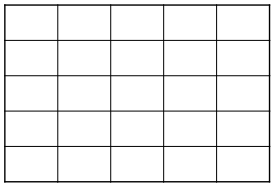
Consejos

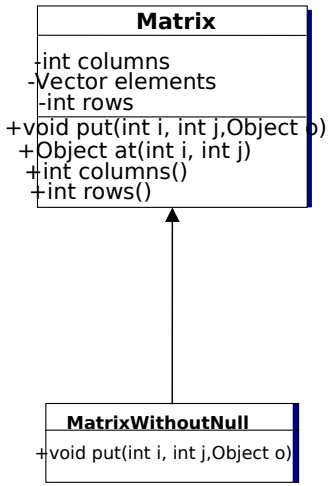
- No abuse de las excepciones, puede incurrir en una sobrecarga significativa
- No trate el excepciones por líneas de código reducidas
  - Utilice bloques try con múltiples bloques catch
- No ignore las excepciones
  - Ejemplo:

```
catch (Exception e) { }          // no hace
  nada
```

Ejemplo: Una Especialización de Matriz

- Una matriz no debe permitir ingresar elementos nulos
  - Propagar una excepción cuando se agrega un elemento nulo





Ejemplo: MatrixWithoutNull Checked

```
public class MatrixWithoutNull extends Matrix  {
  public void put(int i, int j,Object o)
      throws NullMatrixAssignament {
    if (o != null ) super.put(i,j,o);
    else throw new NullMatrixAssignament();}
```

Checked Exception

Checked Exception

```
public class NullMatrixAssignament extends Exception
```

```
public static void main(String[] args) {
  MatrixWithoutNull m= new MatrixWithoutNull();
      try { m.put(1,1,"hello");}
      catch(NullMatrixAssignament e) {....      }
}
```


Ejemplo: MatrixWithoutNull Unchecked

```
public class MatrixWithoutNull extends Matrix  {
public void put(int i, int j,Object o)  {
    if (o != null ) super.put(i,j,o);
    else throw new NullMatrixAssignament();}

public static void main(String[] args) {
       MatrixWithoutNull m= new MatrixWithoutNull();
      m.put(1,1,"hello");}
}}
public class NullMatrixAssignament extends
                                   RunTimeException
```

Excepción Unchecked Sin cláusula throws

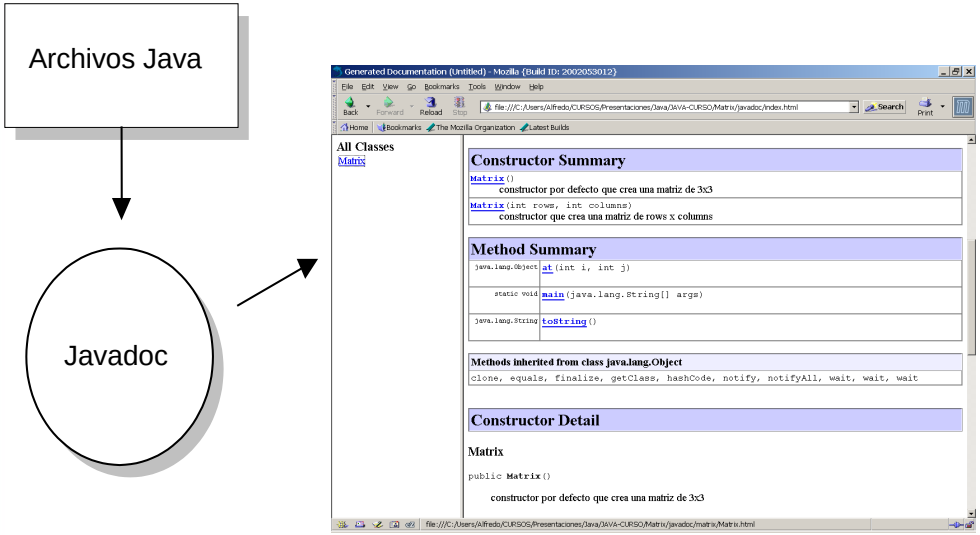
Excepción Unchecked

Java

Javadoc

Javadoc

Permite generar la documentación de las clases en formato html a partir de los comentarios



Documentación

```
/** Matrix Dodumentation
 * @author Alfredo Raul Teyseyre
 * @version 1.0
 */

public class Matrix {
  /** Vector que almacena los elementos de la matriz*/
  private Vector elements;

  /** dimensiones de la matriz */
  private int rows, columns;

  /** constructor por defecto que crea una matriz de 3x3 */
  public Matrix() {
    this(3,3);
  }
```


Documentación

```
/** constructor que crea una matriz de rows x columns
  * de elementos null
  * @param rows cantidad de filas
  * @param columns cantidad de columnas
  */

  public Matrix(int rows, int columns)  {
    this.rows=rows;
    this.columns=columns;
    elements= new Vector(rows);
    for (int i=0 ; i < rows; i++) {
      Vector aux=new Vector(columns);
      for (int j=0; j < columns; j++)
        aux.add(j,null);
      elements.add(i,aux);     }}
```


Documentación: HTML Embebido

```
/**
* You can <em>even</em> insert a list:
* <ol>
* <li> Item one
* <li> Item two
* <li> Item three
* </ol>
*/
```

Tags de Documentación

- **@see: referring to other classes**
  - @see classname
  - @see fully-qualified-classname
  - @see fully-qualified-classname#method-name
- **@version**
  - @version version-information
- **@author**
  - @author author-information
- **@since**
- **@param**
  - @param parameter-name description
- **@return**
  - @return description
- **@throws**
  - @throws fully-qualified-class-name description
- **@deprecated**