FACULTAD DE INGENIERIA
UNIVERSIDAD NACIONAL DE JUJUY

Exceptions – Sistemas de Trazas

Programación Orientada a Objetos

San Salvador de Jujuy

UNJu – Facultad de Ingeniería
Ing. José Zapana

¿Qué son las excepciones?

- Son eventos o condiciones inusuales y no deseadas que pueden ocurrir durante la ejecución de un programa y que interrumpen el flujo normal de instrucciones.
- Se utilizan para manejar situaciones imprevistas o errores en un programa y para evitar que el programa se bloquee o se comporte de manera impredecible.

Conceptos clave sobre manejo de excepciones

- **Lanzamiento de Excepciones:** Cuando ocurre una situación excepcional o un error, el programa puede "lanzar" una excepción.
- **Manejo de Excepciones:** tomar medidas específicas para tratar la situación excepcional.
- **Tipos de Excepciones:** Las excepciones pueden ser de varios tipos, dependiendo de la naturaleza del error.
- **Stack Trace:** traza de la pila (stack trace) que muestra la secuencia de llamadas de funciones o métodos que llevaron al error.
- **Instrucciones Try-Catch (o Try-Catch-Finally):** bloques para gestionar el código y las excepciones.
- **Creación de Excepciones Personalizadas:** para que el manejo de errores sea más específico y significativo.

Ventajas de las Excepciones

- **Manejo estructurado de errores:** legibilidad
- **Prevención de comportamientos inesperados:** error 500
- **Capacidad de recuperación**
- **Comunicación de errores**
- **Seguridad y Fiabilidad:** El uso adecuado de excepciones puede aumentar la seguridad y la fiabilidad de una aplicación al garantizar que los errores se manejen de manera controlada en lugar de propagarse sin control.

Obtención y manejo de excepciones

```
/**
 * Método ejemplo de exceptions
 */
public void unMetodo(){
    try {

        //secuencia de operaciones o acciones
        //que se desean controlar con la exception

    }catch(Exception e){
        //tratamiento de la exception
    }finally{
        //acciones que se realizan independiente de
        //si ocurrió o NO la exception
    }
}
```

Jerarquía de excepciones

- Object
  - Throwable
    - Exception
      - » ClassNotFoundException
      - » NoSuchMethodException
    - RuntimeExeption
      - » ClassCastException
      - » NullPointerException
    - IndexOfBoundException
      - » StringIndexOutOfBoundException
      - » ArrayIndexOutOfBoundException
    - IllgalArgumentException
      - » NumberFormatException
      - » IllegalThreadStateException

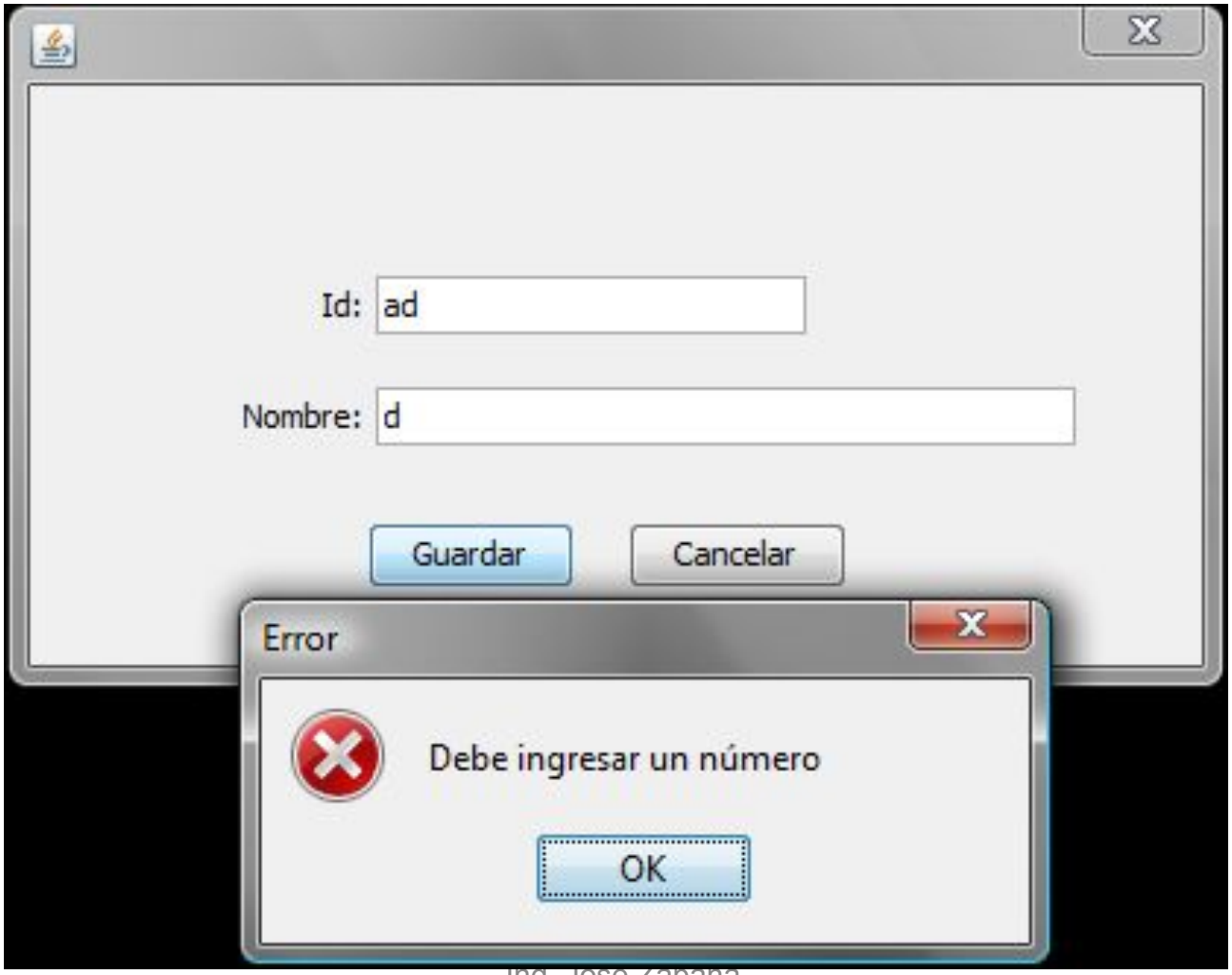
Obtención de una única excepcion

Ejemplo con java swing

```
private void btnGuardarActionPerformed(java.awt.event.ActionEvent evt) {
    try{
        String id = txtClienteId.getText();
        String nombre = txtNombre.getText();
        Cliente cliente = new Cliente(Integer.parseInt(id), nombre);
        ManagerUtil.addCliente(cliente);
        this.dispose();
    }catch(NumberFormatException e){
        JOptionPane.showMessageDialog(null, "Debe ingresar un número",
                                      "Error", JOptionPane.ERROR_MESSAGE);
    }

}
```

Obtención de una única excepción



Obtención de Varias Excepciones

```
/**
 * Método ejemplo de conexion a una URL
 */
public void conexion(String direccion){
    try {
        URL url = new URL(direccion);
        URLConnection conexion = url.openConnection();
        conexion.connect();
    }catch(MalformedURLException e){
        System.err.println("la URL no es válida: " + e);
    }catch(IOException e){
        System.err.println("No es posible conectarse" + e);
    }
}
```

Limpieza con un bloque **finally**

```java
/**
 * Método que cuenta los caracteres de un archivo de texto
 * @param path
 */
public void contar(String path){
    FileInputStream f;
    int cantidadCaracteres = 0;
    try{
        f = new FileInputStream(path);

        while(f.read() != -1){
            cantidadCaracteres++;
        }
        f.close();
    }catch(IOException e){
        System.err.println("Error al acceder al archivo: "+ e);
    }finally{
        System.out.println("Fin del proceso");
    }
}
```

Devolución de Excepciones

- Devuelva las excepciones mediante la palabra clave **`throw`**.
- Utilice **`throws`** en la declaración del método.
- Ventajas
  - Se traslada el manejo de la excepción hacia el cliente del método
  - La excepción podrá ser mostrada en el formato adecuado.

Devolución de Excepciones

- Ejemplo

```
/**
 *
 * @param vector: con elementos enteros
 * @param indice: posición desde donde se quiere obtener un valor
 * @return: elemento del vector en el indice indicado
 * @throws java.lang.IndexOutOfBoundsException
 */
public int getElementoVector(int[]vector, int indice)
                                        throws IndexOutOfBoundsException{

        if(indice < 0)
            throw new IndexOutOfBoundsException("Indice incorrecto");

        return vector[indice];
}
```

Excepciones Personalizadas

- Es posible también crear propias excepciones que permitan identificar y gestionar con mayor claridad una exception

```
class SaldoInsuficienteException extends Exception {
 public SaldoInsuficienteException(String mensaje) {
      super(mensaje);
 }
}
```

Excepciones Personalizadas

```
public CuentaBancaria(double saldoInicial) {
    saldo = saldoInicial;
}

public void retirar(double cantidad) throws SaldoInsuficienteException {
    if (cantidad <= 0) {
        throw new IllegalArgumentException("La cantidad a retirar debe ser mayor que cero");
    }
    if (cantidad > saldo) {
        throw new SaldoInsuficienteException("Saldo insuficiente para realizar el retiro");
    }
    saldo -= cantidad;
    System.out.println("Retiro de $" + cantidad + " exitoso. Saldo restante: $" + saldo);
}
}
```

Excepciones Personalizadas

```
public class ExcepcionPersonalizada {
 public static void main(String[] args) {
     CuentaBancaria cuenta = new CuentaBancaria(1000.0);

     try {
         cuenta.retirar(500.0); // Intentamos retirar $500
         cuenta.retirar(800.0); // Esto lanzará una SaldoInsuficienteException
     } catch (SaldoInsuficienteException e) {
         System.err.println("Error: " + e.getMessage());
     } catch (IllegalArgumentException e) {
         System.err.println("Error: " + e.getMessage());
     }
 }
}
```

LOG4J: Introducción

- A menudo usamos System.out.println(…) para escribir en la consola el valor de una variable
- O bien para mostrar un mensaje “pasó por aquí”, como seguimiento de la ejecución del programa.
- Inconvenientes:
  - Si no se controlan pueden causar demora en los procesos
  - Los mensajes enviados NO indican en donde se produjo el error
  - En ambiente de producción deben ser controlados

Qué es el loggin de las aplicaciones

- El "logging" (registro o registro de eventos) en aplicaciones de programación se refiere al proceso de registrar información significativa o eventos relevantes que ocurren durante la ejecución de una aplicación
- Este registro puede realizarse en un archivo o en otro medio de registro.
- Los registros son una herramienta esencial para el desarrollo, el mantenimiento y la operación de aplicaciones de software.

Propósitos del logging

- **Depuración y diagnóstico:** ver datos del programa
- **Auditoría y seguridad:** transacciones bancarias
- **Rendimiento y optimización:** tiempos de respuesta
- **Trazabilidad y Monitorización:** seguir el flujo de ejecución a través de diferentes componentes y microservicios
- **Registro de Actividades:** acciones realizadas por los usuarios en una aplicación
- **Cumplimiento de Normativas:** registros detallados para cumplir con regulaciones y normativas específicas

¿Qué es Log4j?

- **Log4j** es una biblioteca open source desarrollada en Java por la Apache Software Foundation
- Permite a los desarrolladores de software elegir la salida y el nivel de granularidad de los mensajes
- La configuración de salida y granularidad de los mensajes es realizada en tiempo de ejecución mediante el uso de archivos de configuración externos.
- Log4J ha sido implementado en otros lenguajes como: C, C++, C#, Perl, Python, Ruby y Eiffel.

Pasos para la implementación

1. Agregar dependencia log4j
2. Crear archivo de configuración (por ej. log4j.properties)
3. Implementar en las clases del proyecto

Log4J – Niveles de mensajes

- Cuadro comparativo

Logging Level	Detail Included in Logs: Debug statements	Informational messages	Warning statements	Error statements	Fatal statements
ALL	Included	Included	Included	Included	Included
DEBUG	Included	Included	Included	Included	Included
INFO	Excluded	Included	Included	Included	Included
WARN	Excluded	Excluded	Included	Included	Included
ERROR	Excluded	Excluded	Excluded	Included	Included
FATAL	Excluded	Excluded	Excluded	Excluded	Included
OFF	Excluded	Excluded	Excluded	Excluded	Excluded

LOG4J: Appenders

- Son las salidas de destino que se pueden configurar.
- Existen varios appenders disponibles y configurados, aunque también podemos crear y configurar nuestros propios appenders.
  - Fichero de texto *.log* (*FileAppender*, *RollingFileAppender*)
  - Servidor remoto donde almacenar registros (*SocketAppender*)
  - Dirección de correo electrónico (*SMTPAppender*),
  - Base de datos (*JDBCAppender*).

log4.properties - Ejemplo

```
# Root logger option
log4j.rootLogger=DEBUG, stdout, file

# Redirect log messages to console
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n
log4j.logger.ar.edu.unju.fi.poo=error

# Redirect log messages to a log file, support file rolling.
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=log4j-application.log
log4j.appender.file.MaxFileSize=5MB
log4j.appender.file.MaxBackupIndex=10
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n
```

Configuración

- Pom.xml

```
<dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>1.2.17</version>
 </dependency>
```

Referencias

- log4j 1.2 official page
- log4j pattern layout
- Wikipedia : log4j
- Spring MVC + log4j example
- log4j.properties examples
-