|  |
| --- |
| **TECNICATURA UNIVERSITARIA EN PROGRAMACION**  **Laboratorio de Computación II - Ciclo lectivo 2025** |

|  |  |
| --- | --- |
| **Características del Práctico : Excepciones en Java** | |
| **Unidad** | **3** |
| **Tema** | **Excepciones en Java** |
| **Resultados de Aprendizajes** | RA1: Domina el sistema de excepciones Java, lanzamiento captura,  RA2: Domina la creación de tipos propios de excepciones. |
| **Objetivo** | El objetivo de este trabajo práctico es que los estudiantes adquieran un entendimiento del sistema de excepciones provisto por Java y puedan aplicarlo a sus propios desarrollos. |
| **Requisitos técnicos** | Codificar una solución básica en Java para probar el sistema de logging. |
| **Fecha de inicio** | Marzo de 2025 - |
| **Fecha de entrega:** | Marzo de 2025 |
| **Modalidad de entrega :** | Adjuntar el trabajo en formato Word al link correspondiente en el aula virtual. |
| **Comisiones** | **Comisiones: Mañana y tarde** |
| **Modalidad del Trabajo Práctico** | Desarrollo Grupal – Entrega en aula virtual |
|  | |
| **Conclusiones:**  Los estudiantes deben resumir lo que aprendieron durante la realización del trabajo práctico y destacar cómo el manejo de excepciones en Java permite un mejor seguimiento de la ejecución de la aplicación, como así también el manejo de errores y situaciones anómalas. | |
| Forma de Presentación | El trabajo debe presentarse en el link correspondiente al trabajo práctico alojado en el aula virtual. |
| Tipo de Evaluación | Formativa - Grupal |
| **Modalidad de Evaluación** | |  |  |  |  | | --- | --- | --- | --- | |  | **100** | **>70%** | **<50 %** | | **Lanzamiento de Excepciones** |  |  |  | | **Manejo de Excepciones** |  |  |  | | **Creación de tipos propios de excepciones** |  |  |  | | **RESULTADOS** | **Excelente** | **Satisfactorio** | **No Satisfactorio** |   **Se establece una sumatoria de cada ítem y se determina el porcentaje individual de cada uno. Posteriormente se establece la sumatoria de todos los ítems y se lo divide por la cantidad de los mismos para determinar en cuál de los valores de la escala se encuadra.**   |  |  |  |  | | --- | --- | --- | --- | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |
| Bibliografía | Effective Java 3rd Edición. [Joshua Bloch](https://www.amazon.com/-/es/Joshua-Bloch/e/B001CDCVUG/ref=dp_byline_cont_book_1). (2017)  Java: The Complete Reference, Twelfth Edition 12th Edición.  [Herbert Schildt](https://www.amazon.com/-/es/Herbert-Schildt/e/B001H6PSMG/ref=dp_byline_cont_book_1) (2021) |
| Profesores responsables del TP. | |  |  |  |  | | --- | --- | --- | --- | | Apellido y nombre | | Apellido y nombre | | | Julio Monetti | | Claudia Naveda | | |  |  | |  | |  |  | |

DESARROLLO

**PARTE A**

Ejercicio

Responda.

* 1. Cuál es el nombre en Java de la clase que define las excepciones ?
* En Java, la clase que define las excepciones se llama **Exception**.
* Forma parte de la jerarquía de clases de manejo de errores, y hereda de la clase Throwable, que es la superclase de todas las clases que pueden ser lanzadas como excepciones o errores.

b.Cuál es el nombre en Java de la clase que representa las excepciones que se producen al invocar un método que pertenece a un objeto nulo ?

La clase en Java que representa las excepciones que se producen al invocar un método sobre un objeto nulo se llama:

### NullPointerException

Esta es una excepción **no verificada** (unchecked), ya que hereda de RuntimeException.

c.Qué información aporta el método *printStackTrace* en el manejo de excepciones ?

El método **printStackTrace()** en Java se utiliza para **imprimir el seguimiento de la pila (stack trace)** de una excepción. Esta información es muy útil para depurar errores porque muestra:

1. **El tipo de excepción** lanzada (por ejemplo, NullPointerException).
2. **El mensaje asociado** a la excepción (si lo hay).
3. **La secuencia de llamadas de métodos** que llevaron a que se produjera la excepción, indicando:
   * Nombre del archivo,
   * Número de línea,
   * Nombre del método y clase donde ocurrió el error.

**Ejercicio**

Como obtiene el mensaje perteneciente a una excepción ?. De un ejemplo de código donde se utilice este dato.

Para obtener el **mensaje** asociado a una excepción en Java, se utiliza el método:

### getMessage()

Este método devuelve un **String** con el mensaje que fue especificado al lanzar la excepción (o un mensaje por defecto si no se proporcionó uno).

Ej:

public class EjemploExcepcion {

public static void main(String[] args) {

try {

int resultado = dividir(10, 0); // Esto lanza una excepción

System.out.println("Resultado: " + resultado);

} catch (ArithmeticException e) {

System.out.println("Ocurrió una excepción: " + e.getMessage());

}

}

public static int dividir(int a, int b) {

if (b == 0) {

throw new ArithmeticException("No se puede dividir por cero");

}

return a / b;

}

}

**Ejercicio**

Observe el siguiente fragmento de código

*String aux = “hola”;*

*int aux2 = Integer.parseInt( aux );*

Qué sucede al ejecutar el mismo ?

Este código **lanza una excepción en tiempo de ejecución**, porque intenta convertir la cadena "hola" a un número entero.

Modifíquelo para realizar manejo de excepciones.

public class EjemploParseo {

public static void main(String[] args) {

String aux = "hola";

try {

int aux2 = Integer.parseInt(aux);

System.out.println("Número convertido: " + aux2);

} catch (NumberFormatException e) {

System.out.println("Error: No se puede convertir '" + aux + "' a número.");

System.out.println("Mensaje de la excepción: " + e.getMessage());

}

}

}

**Ejercicio**

Cree un programa en Java que produzca una IOException. Lance y maneje la excepción adecuadamente.

import java.io.IOException;

public class EjemploIOException {

public static void main(String[] args) {

try {

// Simulamos una operación que lanza una IOException

lanzarIOException();

} catch (IOException e) {

System.out.println("Se capturó una IOException:");

System.out.println("Mensaje: " + e.getMessage());

e.printStackTrace(); // Opcional: muestra el stack trace para depuración

}

}

// Método que lanza una IOException de forma intencional

public static void lanzarIOException() throws IOException {

throw new IOException("Ocurrió un error de entrada/salida simulado.");

}

}

**Ejercicio**

Cree un método que capture al menos tres tipos de excepciones en forma anidada. (Cree también el método que las lanza).

public class ExcepcionesAnidadas {

public static void main(String[] args) {

try {

metodoQueLanzaExcepciones(2); // Cambia el número entre 1, 2 o 3 para probar

} catch (ArithmeticException e) {

System.out.println("Se capturó una ArithmeticException: " + e.getMessage());

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Se capturó una ArrayIndexOutOfBoundsException: " + e.getMessage());

} catch (NullPointerException e) {

System.out.println("Se capturó una NullPointerException: " + e.getMessage());

}

}

public static void metodoQueLanzaExcepciones(int tipo) {

switch (tipo) {

case 1:

int resultado = 10 / 0; // Lanza ArithmeticException

break;

case 2:

int[] arreglo = new int[3];

int x = arreglo[5]; // Lanza ArrayIndexOutOfBoundsException

break;

case 3:

String texto = null;

texto.length(); // Lanza NullPointerException

break;

default:

System.out.println("No se lanzó ninguna excepción.");

}

}

}

**Ejercicio**

Investigue en Internet y copie un árbol de excepciones de Java. Documente al menos cuatro de ellas.

java.lang.Object

└── java.lang.Throwable

├── java.lang.Error

│ ├── OutOfMemoryError

│ └── StackOverflowError

└── java.lang.Exception

├── java.io.IOException

├── java.sql.SQLException

└── java.lang.RuntimeException

├── NullPointerException

├── ArithmeticException

└── IndexOutOfBoundsException

Esta estructura muestra cómo las diferentes clases de excepciones y errores se derivan de la clase base Throwable. Las excepciones que heredan de Exception son las que los programas pueden manejar, mientras que las que heredan de Error indican problemas más serios que generalmente no se deben capturar.

### Documentación de Cuatro Excepciones

A continuación, se describen cuatro excepciones comunes en Java:

1. **IOException**
   * **Tipo**: Excepción verificada (checked exception).
   * **Descripción**: Se lanza cuando ocurre un error de entrada/salida, como al intentar leer un archivo que no existe o al fallar una operación de red.
   * **Ejemplo**:

java

CopiarEditar

try {

FileReader reader = new FileReader("archivo.txt");

} catch (IOException e) {

e.printStackTrace();

}

1. **NullPointerException**
   * **Tipo**: Excepción no verificada (unchecked exception).
   * **Descripción**: Ocurre cuando se intenta usar una referencia de objeto que es null, como llamar a un método o acceder a un campo de un objeto no inicializado.
   * **Ejemplo**:

java

CopiarEditar

String texto = null;

System.out.println(texto.length()); // Lanza NullPointerException

1. **ArithmeticException**
   * **Tipo**: Excepción no verificada (unchecked exception).
   * **Descripción**: Se lanza cuando ocurre una condición aritmética ilegal, como dividir por cero.
   * **Ejemplo**:

java

CopiarEditar

int resultado = 10 / 0; // Lanza ArithmeticException

1. **ArrayIndexOutOfBoundsException**
   * **Tipo**: Excepción no verificada (unchecked exception).
   * **Descripción**: Ocurre cuando se intenta acceder a un índice fuera de los límites de un arreglo.
   * **Ejemplo**:

java

CopiarEditar

int[] numeros = {1, 2, 3};

int numero = numeros[5]; // Lanza ArrayIndexOutOfBoundsException

**Ejercicio**

Cree una jerarquía de excepciones de dos niveles. De ejemplos de usos de la misma.

* MiExcepcionBase (clase base)
  + ExcepcionTipoA
  + ExcepcionTipoB

// Clase base

class MiExcepcionBase extends Exception {

public MiExcepcionBase(String mensaje) {

super(mensaje);

}

}

// Subclase 1

class ExcepcionTipoA extends MiExcepcionBase {

public ExcepcionTipoA(String mensaje) {

super(mensaje);

}

}

// Subclase 2

class ExcepcionTipoB extends MiExcepcionBase {

public ExcepcionTipoB(String mensaje) {

super(mensaje);

}

}

public class PruebaJerarquiaExcepciones {

public static void main(String[] args) {

try {

ejecutarOperacion("B"); // Cambia entre "A", "B", o "otro"

} catch (ExcepcionTipoA e) {

System.out.println("Se capturó ExcepcionTipoA: " + e.getMessage());

} catch (ExcepcionTipoB e) {

System.out.println("Se capturó ExcepcionTipoB: " + e.getMessage());

} catch (MiExcepcionBase e) {

System.out.println("Se capturó MiExcepcionBase: " + e.getMessage());

}

}

public static void ejecutarOperacion(String tipo) throws MiExcepcionBase {

if ("A".equals(tipo)) {

throw new ExcepcionTipoA("Ocurrió un error tipo A.");

} else if ("B".equals(tipo)) {

throw new ExcepcionTipoB("Ocurrió un error tipo B.");

} else {

throw new MiExcepcionBase("Error general.");

}

}

}

**Ejercicio**

Qué muestra el siguiente programa ? (identifique la salida sin ejecutarlo)

|  |
| --- |
| p*ublic class Main {*  *public static int devuelveNumero(int num) {*  *try {*  *if (num % 2 == 0) {*  *throw new Exception("Lanzando excepcion");*  *}*  *return 1;*  *} catch (Exception ex) {*  *return 2;*  *} finally {*  *return 3;*  *}*  *}*    *public static void main(String[] args) {*  *System.out.println(devuelveNumero(1));*  *}*    *}* |

Salida en consola:

3

**PARTE B**

**Ejercicio**

Cree un programa que lance una ArithmeticException.

public class EjemploArithmeticException {

public static void main(String[] args) {

int a = 10;

int b = 0;

// Esto lanza ArithmeticException: división por cero

int resultado = a / b;

System.out.println("Resultado: " + resultado);

}

}

Tipo de error que puede lanzar:

Exception in thread "main" java.lang.ArithmeticException: / by zero

at EjemploArithmeticException.main(EjemploArithmeticException.java:6)

Usando Try-Catch:

public class EjemploArithmeticExceptionManejo {

public static void main(String[] args) {

int a = 10;

int b = 0;

try {

int resultado = a / b;

System.out.println("Resultado: " + resultado);

} catch (ArithmeticException e) {

System.out.println("¡Error! División por cero no permitida.");

System.out.println("Mensaje de la excepción: " + e.getMessage());

}

}

}

**Ejercicio**

Codifique la siguiente aplicación



El método *acelera* lanza una excepción de tipo **miException** si alguno de los vehículos intenta acelerar a más de 120km/h. Agregar a esta clase la funcionalidad necesaria para mostrar por pantalla el nombre del objeto/clase que la produce (Auto o Camion).

Al crear un chofer, se debe lanzar una excepción **matriculaVencidaException** si la matrícula está vencida.

**Ejercicio**

Cree un programa que genere un número aleatorio e indique si el número generado es par o impar. El programa utilizará para ello el lanzamiento de una excepción.

import java.util.Random;

// Excepción personalizada para números pares

class NumeroParException extends Exception {

public NumeroParException(int numero) {

super("El número " + numero + " es PAR.");

}

}

// Excepción personalizada para números impares

class NumeroImparException extends Exception {

public NumeroImparException(int numero) {

super("El número " + numero + " es IMPAR.");

}

}

public class NumeroAleatorio {

public static void main(String[] args) {

Random rand = new Random();

int numero = rand.nextInt(100); // genera número entre 0 y 99

System.out.println("Número generado: " + numero);

try {

verificarParidad(numero);

} catch (NumeroParException | NumeroImparException e) {

System.out.println("Resultado: " + e.getMessage());

}

}

// Método que lanza excepción según si es par o impar

public static void verificarParidad(int numero) throws NumeroParException, NumeroImparException {

if (numero % 2 == 0) {

throw new NumeroParException(numero);

} else {

**Ejercicio**

Cree un programa que lance una excepción a través de una pila de llamada de métodos de longitud cuatro. El último método debe manejar la excepción.

public class PilaExcepciones {

public static void main(String[] args) {

metodo4(); // el método más externo

}

public static void metodo4() {

try {

metodo3();

} catch (Exception e) {

System.out.println("Excepción capturada en metodo4:");

e.printStackTrace(); // Muestra la pila de llamadas

}

}

public static void metodo3() {

metodo2();

}

public static void metodo2() {

metodo1();

}

public static void metodo1() {

// Aquí se lanza la excepción

throw new RuntimeException("Excepción lanzada desde metodo1");

}

}