| **TECNICATURA UNIVERSITARIA EN PROGRAMACION**  **Laboratorio de Computación II - Ciclo lectivo 2024** |
| --- |

| **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 2024 - |
| **Fecha de entrega:** | Marzo de 2024 |
| **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 ?

La clase que define las excepciones es la clase exception

* 1. 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 ?

El nombre es NullPointerException

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

Imprime el lugar donde aparece el error de la excepcion

**Ejercicio**

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

**Con getMessage()**

// como la llamada a nuestro método puede arrojar una excepción debemos

// encerrarla en un bloque try-catch

try

{

Equipo ganador = encuentro.obtenerEquipoGanador();

System.out.println(“El equipo ganador fue: ” + ganador);

}

catch (PartidoNoJugadoException ex)

{

System.out.println(“No se puede obtener el equipo ganador porque: ” +

ex.getMessage());

**Ejercicio**

Observe el siguiente fragmento de código

*String aux = “hola”;*

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

Qué sucede al ejecutar el mismo ?

Nos da un error de **java.lang.NumberFormatException: For input string: "hola"**

Modifíquelo para realizar manejo de excepciones.

String aux ="hola";  
try {  
 int aux2 = Integer.*parseInt*(aux);  
 System.*out*.println("El valor convertido es: " + aux2);  
}catch (NumberFormatException e) {  
 System.*out*.println("Se produjo una excepcion: " + e.getMessage());  
}

**Ejercicio**

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

import java.util.InputMismatchException;  
import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
  
 try {  
 int edad = sc.nextInt();  
 } catch (InputMismatchException e) {  
 System.*out*.println("Error: Entrada no válida. Se esperaba un número entero.");  
 }  
 }  
}

**Ejercicio**

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

import java.util.InputMismatchException;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.*in*);

try {

int edad = sc.nextInt();

throw new ArithmeticException ("Cuenta no permitida");

} catch (InputMismatchException e) {

System.*out*.println("Error: Entrada no válida. Se esperaba un número entero.");

} catch (ArrayIndexOutOfBoundsException e){

System.*out*.println("No hay array");

} catch (ArithmeticException e){

System.*out*.println("Error aritmético");

}

}

}

**Ejercicio**

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

**Throwable**

**├── Error**

**│ ├── AssertionError**

**│ ├── OutOfMemoryError**

**│ ├── StackOverflowError**

**│ └── ...**

**│**

**└── Exception**

**├── IOException**

**│ ├── FileNotFoundException**

**│ ├── EOFException**

**│ └── ...**

**│**

**├── RuntimeException**

**│ ├── NullPointerException**

**│ ├── ArrayIndexOutOfBoundsException**

**│ ├── IllegalArgumentException**

**│ └── ...**

**│**

**└── ...**

Aquí hay una breve descripción de cuatro de las excepciones mencionadas:

**IOException:**

1. Esta excepción es la superclase de las excepciones que se lanzan cuando ocurren problemas durante la operación de entrada/salida, como la lectura o escritura de archivos, operaciones de red, etc. Algunas subclases notables son FileNotFoundException, que se lanza cuando un archivo no se encuentra, y EOFException, que se lanza cuando se alcanza el final de un flujo de entrada.

**NullPointerException:**

1. Esta excepción se lanza cuando intentas acceder a un objeto o llamar a un método en una referencia de objeto que es null. Es una de las excepciones más comunes en Java y puede ocurrir cuando no inicializas una variable de referencia, o cuando intentas acceder a un miembro de un objeto que no ha sido inicializado.

**ArrayIndexOutOfBoundsException:**

1. Esta excepción se lanza cuando intentas acceder a un índice fuera del rango válido de un arreglo (array). Ocurre cuando intentas acceder a un índice negativo o mayor que el tamaño del arreglo.

**IllegalArgumentException:**

1. Esta excepción se lanza cuando un método recibe un argumento ilegal o inapropiado. Por ejemplo, puede ocurrir si un método espera un valor dentro de un rango específico y recibe un valor fuera de ese rango.

Estas son solo algunas de las excepciones disponibles en Java. Cada una está diseñada para manejar casos específicos de errores o situaciones excepcionales durante la ejecución del programa.

**Ejercicio**

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

**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));*  *}*    *}* |
| --- |

Analiza que el numero sea par y devuelve una excepcion que arroja (Lanzando excepcion)

Como el numero no se cumple, el ejercicio salta directamente al finally, El return debe ser un 3

**PARTE B**

**Ejercicio**

Cree un programa que lance una ArithmeticException.

import java.util.InputMismatchException;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.*in*);

try {

int c1 = sc.nextInt();

double c2 = c1 / 0;

throw new ArithmeticException ("Cuenta no permitida");

} catch (ArithmeticException e){

System.*out*.println("Error aritmetico");

}

}

}

**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;

public class Main {

public static void main(String[] args) {

try {

int numeroGenerado = *generarNumero*();

System.*out*.println("El número generado es: " + numeroGenerado);

if (*esPar*(numeroGenerado)) {

throw new Exception("El número generado es par.");

} else {

throw new Exception("El número generado es impar.");

}

} catch (Exception e) {

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

}

}

public static int generarNumero() {

Random rand = new Random();

return rand.nextInt(100) + 1;

}

public static boolean esPar(int numero) {

return numero % 2 == 0;

}

}

**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 Main {

public static void main(String[] args) {

try {

*metodoUno*();

} catch (Exception e) {

System.*out*.println("Excepción manejada en el método main: " + e.getMessage());

}

}

public static void metodoUno() throws Exception {

*metodoDos*();

}

public static void metodoDos() throws Exception {

*metodoTres*();

}

public static void metodoTres() throws Exception {

*metodoCuatro*();

}

public static void metodoCuatro() throws Exception {

throw new Exception("Excepción lanzada desde el métodoCuatro");

}

}