PARTE A:

**Ejercicio**

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.

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?**

En Java, la clase que representa las excepciones que se producen al invocar un método en un objeto nulo se llama NullPointerException.

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

El método printStackTrace() en Java es una herramienta utilizada para manejar excepciones y errores.

**Ejercicio**

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

Respuesta:

En Java, puedes obtener el mensaje de una excepción utilizando el método getMessage(). Este método es parte de la clase Throwable y devuelve una cadena de texto que contiene el detalle del mensaje de la excepción. El mensaje puede ser null si no se proporcionó ningún mensaje cuando se creó la excepción

try {

// algunas acciones riesgosas

} catch (Exception e) {

String mensajeDeExcepcion = e.getMessage();

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

}

**Ejercicio**

**Observe el siguiente fragmento de código**

***String aux = “hola”;***

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

**¿Qué sucede al ejecutar el mismo?**

**Modifíquelo para realizar manejo de excepciones**

Al ejecutar el fragmento de código proporcionado, se producirá una *NumberFormatException*. Esto se debe a que estás intentando convertir la cadena de texto "hola" en un número entero utilizando el método Integer.parseInt(), lo cual no es posible ya que "hola" no es una representación válida de un número entero.

**CÓDIGO MODIFICADO:**

String aux = "hola";

try {

int aux2 = Integer.parseInt(aux);

} catch (NumberFormatException e) {

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

}

**Ejercicio**

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

import java.io.\*;

public class IOExceptionExample {

public static void main(String[] args) {

try {

*produceIOException*();

} catch (IOException e) {

System.*out*.println("Se ha producido una IOException: " + e.getMessage());

}

}

static void produceIOException() throws IOException {

throw new IOException("Es ta es una excepción de IO intencional");

}

}

**Ejercicio**

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

import java.io.\*;

public class ExcepcionesAnidadas {

public static void main(String[] args) {

try {

*produceExceptions*();

} catch (IOException e) {

System.*out*.println("Se ha producido una IOException: " + e.getMessage());

try {

*produceArithmeticException*();

} catch (ArithmeticException ae) {

System.*out*.println("Se ha producido una ArithmeticException: " + ae.getMessage());

try {

*produceNullPointerException*();

} catch (NullPointerException ne) {

System.*out*.println("Se ha producido una NullPointerException: " + ne.getMessage());

}

}

}

}

static void produceExceptions() throws IOException {

throw new IOException("Esta es una excepción de IO intencional");

}

static void produceArithmeticException() {

throw new ArithmeticException("Esta es una excepción aritmética intencional");

}

static void produceNullPointerException() {

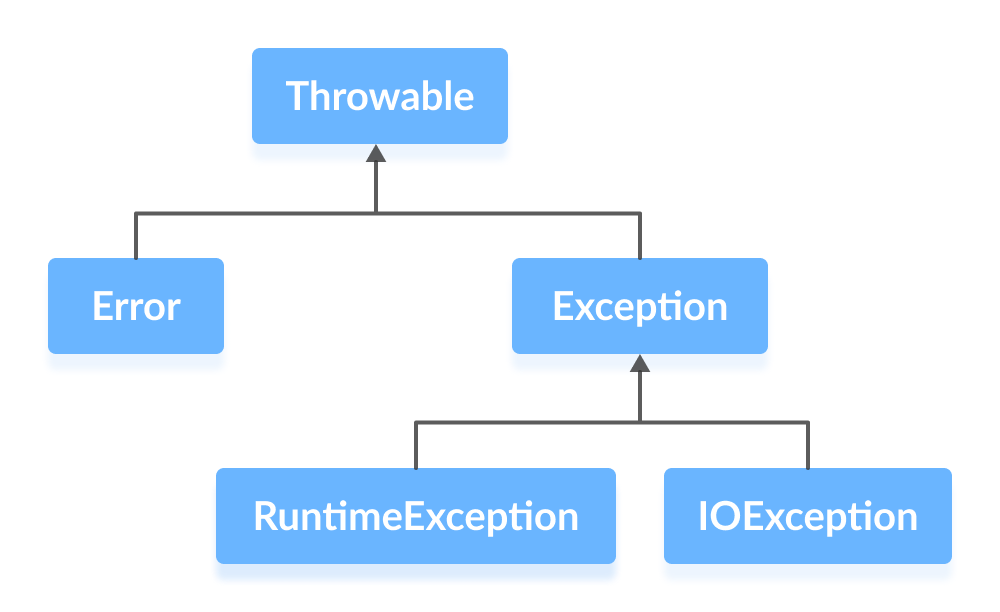
throw new NullPointerException("Esta es una excepción de puntero nulo intencional");

}

}

**Ejercicio**

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



**Throwable:**

La clase Throwable es la superclase de todos los errores y excepciones en Java.

Hereda a las clases Error y Exception.

Puede capturar errores y excepciones producidos por la Máquina Virtual Java, ya sea errores de seguimiento de pila o excepciones de tiempo de ejecución o compilación.

Algunos métodos importantes de la clase Throwable son:

toString(): Devuelve la descripción del error en formato de cadena.

getMessage(): Obtiene el mensaje de error en formato de cadena.

getCause(): Devuelve la causa de la instancia actual de un Throwable.

getStackTrace(): Proporciona información sobre el seguimiento de la pila.

printStackTrace(): Imprime la instancia actual de Throwable.

**Error:**

La clase Error incluye errores que detienen la ejecución del programa.

Ejemplo: StackOverflowError.

**Exception:**

La clase Exception es una subclase de Throwable.

Representa excepciones que pueden ser manejadas.

Ejemplo: IOException.

**RuntimeException:**

Es una subclase de Exception.

Excepciones de tiempo de ejecución que no necesitan ser declaradas en la cláusula throws.

Ejemplo: NullPointerException

**Ejercicio**

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

**1 -Nivel Superior:**

* MiExcepcionPersonalizada: Una excepción personalizada que hereda de la clase Exception.
* Puedes crear esta excepción para manejar situaciones específicas en tu aplicación.
* Ejemplo de uso:

public class MiExcepcionPersonalizada extends Exception {

public MiExcepcionPersonalizada(String mensaje) {

super(mensaje);

}

}

**2 - Nivel Inferior:**

* DivisionPorCeroException: Una excepción específica para manejar errores de división por cero.
* Hereda de MiExcepcionPersonalizada.
* Ejemplo de uso:

public class DivisionPorCeroException extends MiExcepcionPersonalizada {

public DivisionPorCeroException() {

super("Error: División por cero.");

}

}

**3 - main:**

* En el main tenemos 2 variable numerador y denominador.
* Si el denominador es 0, entra en el DivisionPorCeroException, y entrará en el catch.
* sino muestra el resultado por pantalla.

public class EjemploExcepsiones {

public static void main(String[] args) {

try {

int numerador = 10;

int denominador = 0;

if (denominador == 0) {

throw new DivisionPorCeroException();

}

int resultado = numerador / denominador;

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

} catch (DivisionPorCeroException e) {

System.*err*.println(e.getMessage());

}

}

}

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

El valor que devuelve sera 3, ya que la sentencia finallyse ejecutara despues del bloque try, catch sin importar que ha y o hubo alguna excepcion. Por lo cual siempre devolvera 3 sin importar si se encuentra alguna excepcion en el codigo.

PARTE B:

**Ejercicio**:

Cree un programa que lance una ArithmeticException:

public static void main(String[] args) {

// Inicializa dos variables: a y b

int a = 10;

int b = 0; // b es igual a 0, lo que provocará una excepción cuando intentemos dividir a por b

// Intenta dividir a por b y captura la excepción si se produce

try {

// Esta línea lanza una ArithmeticException porque b es 0

int resultado = a / b;

System.*out*.println("El resultado de la división es: " + resultado);

} catch (ArithmeticException e) {

// Captura la excepción y muestra un mensaje indicando que ocurrió una excepción

System.*out*.println("Ocurrió una ArithmeticException: No se puede dividir por cero.");

}

}

En este main, lo ue hacemos es dividir a por b, como es una división por cero nos va a dar error, y por lo tanto se va a ir por el arithmeticException y largara un mensaje diciendo que no se puede dividir por cero.

**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 Camión).

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

**Respuestas:**

* **Vehículo y sus clases hijas (Auto, Camión):**

public abstract class Vehiculo {

private String nombre;

private int velocidad;

public Vehiculo(String nombre, int velocidad) {

this.nombre = nombre;

this.velocidad = velocidad;

}

public String getNombre() {

return nombre;

}

public void setNombre(String nombre) {

this.nombre = nombre;

}

public int getVelocidad() {

return velocidad;

}

public void setVelocidad(int velocidad) {

this.velocidad = velocidad;

}

public abstract void acelera(int incremento) throws miException;

}

public class Auto extends Vehiculo{

public Auto(String nombre, int velocidad) {

super(nombre, velocidad);

}

@Override

public void acelera(int incremento) throws miException {

int velocidad = getVelocidad();

setVelocidad(velocidad += incremento);

if (getVelocidad() > 120) {

throw new miException("El vehículo " + getNombre() + " (Auto) está acelerando demasiado rápido: " + getVelocidad() + " km/h");

}

System.*out*.println("El vehículo " + getNombre() + " (Auto) ha acelerado a " + getVelocidad() + " km/h");

}

}

public class Camion extends Vehiculo{

public Camion(String nombre, int velocidad) {

super(nombre, velocidad);

}

@Override

public void acelera(int incremento) throws miException {

int velocidad = getVelocidad();

setVelocidad(velocidad += incremento);

if (getVelocidad() > 120) {

throw new miException("El vehículo " + getNombre() + " (Camion) está acelerando demasiado rápido: " + getVelocidad() + " km/h");

}

System.*out*.println("El vehículo " + getNombre() + " (Camion) ha acelerado a " + getVelocidad() + " km/h");

}

}

* **miException:**

class miException extends Exception {

public miException(String mensaje) {

super(mensaje);

}

}

* **Clase Chofer:**

public class Chofer {

private String nombre;

private boolean matriculaVencida;

public Chofer(String nombre, boolean matriculaVencida) throws matriculaVencidaException {

this.nombre = nombre;

this.matriculaVencida = matriculaVencida;

if (matriculaVencida) {

throw new matriculaVencidaException("La matrícula del chofer " + nombre + " está vencida.");

}

}

public String getNombre() {

return nombre;

}

public void setNombre(String nombre) {

this.nombre = nombre;

}

public boolean isMatriculaVencida() {

return matriculaVencida;

}

public void setMatriculaVencida(boolean matriculaVencida) {

this.matriculaVencida = matriculaVencida;

}

}

* **matriculaVencidaException:**

public class matriculaVencidaException extends Exception {

public matriculaVencidaException(String mensaje) {

super(mensaje);

}

}

* **main:**

public class main {

public static void main(String[] args) {

// Creación de objetos Auto y Camion

Auto auto = new Auto("Toyota", 50);

Camion camion = new Camion("Volvo", 30);

// Manejo de excepción para aceleración de Auto y Camion

try {

// Acelerar Auto por encima de 120 km/h

auto.acelera(130);

} catch (miException e) {

System.*out*.println("Excepción capturada en Auto: " + e.getMessage());

}

try {

// Acelerar Camion por encima de 120 km/h

camion.acelera(150);

} catch (miException e) {

System.*out*.println("Excepción capturada en Camion: " + e.getMessage());

}

// Creación de un chofer con matrícula vencida

try {

Chofer chofer = new Chofer("Juan", true); // La matrícula está vencida (true)

} catch (matriculaVencidaException e) {

System.*out*.println("Excepción capturada al crear Chofer: " + e.getMessage());

}

// Creación de un chofer sin matrícula vencida

try {

Chofer choferSinVencida = new Chofer("Pedro", false); // La matrícula no está vencida (false)

System.*out*.println("Chofer " + choferSinVencida.getNombre() + " creado exitosamente sin matrícula vencida.");

} catch (matriculaVencidaException e) {

System.*out*.println("No se esperaba esta excepción, la matrícula no debería estar vencida.");

}

}

}

**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 {

// Inicia la cadena de llamadas a métodos

*metodo1*();

} catch (Exception e) {

// Captura la excepción lanzada por metodo4

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

}

}

// Método 1 llama a método 2

public static void metodo1() throws Exception {

*metodo2*();

}

// Método 2 llama a método 3

public static void metodo2() throws Exception {

*metodo3*();

}

// Método 3 llama a método 4

public static void metodo3() throws Exception {

*metodo4*();

}

// Método 4 lanza una excepción

public static void metodo4() throws Exception {

// Lanza una excepción con un mensaje personalizado

throw new Exception("Excepción lanzada por metodo4");

}

}

En el main llamamos al metodo1(), que a su vez llama al metodo2(), que llama al metodo3(), que llama al metodo4() que al final muestra el mensaje de Exception y entra en el catch y muestra el mensaje.