**EXCEPTION HANDLING**

**Keywords:**

**try -** Used to enclose the code that might throw exceptions and is followed by either a catch block or a finally block or both.

**catch -** Used to handle specific types of exceptions that occur within the corresponding try block.

**finally -** Used to execute code that should always run, regardless of whether an exception occurred or not in the try block.

**throw -** Used to explicitly throw an exception within a method.

**throws -** Used in method signatures to indicate that the method might throw certain types of exceptions, which are handled by the caller.

**throwable -** The base class for all exceptions and errors in Java. It includes both exception and error classes.

**--------------------------------------------------- CHECKED EXCEPTION ---------------------------------------------------**

Checked exceptions are exceptions that are checked at compile-time.

**IOException**

import java.io.FileWriter;

import java.io.IOException;

class Employee {

private String name;

public Employee(String name) {

this.name = name;

}

public void saveToFile(String filePath) throws IOException {

FileWriter writer = new FileWriter(filePath);

writer.write(this.name);

writer.close();

}

public static void main(String[] args) {

Employee emp = new Employee("Sneha");

try {

emp.saveToFile("employee.txt");

} catch (IOException e) {

e.printStackTrace();

} } }

**FileNotFoundException**

import java.io.FileReader;

import java.io.FileNotFoundException;

import java.io.IOException;

public class FileNotFoundExceptionExample {

public static void main(String[] args) {

try {

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

reader.close();

} catch (FileNotFoundException e) {

System.err.println("File not found: " + e.getMessage());

} catch (IOException e) {

e.printStackTrace();

} } }

**ClassNotFoundException**

public class ClassNotFoundExceptionExample {

public static void main(String[] args) {

try {

Class<?> clazz = Class.forName("java.util.Date");

System.out.println("Class found: " + clazz.getName());

} catch (ClassNotFoundException e) {

e.printStackTrace();

} } }

**InterruptedException**

public class InterruptedExceptionExample {

public static void main(String[] args) {

Thread thread = new Thread(() -> {

try {

System.out.println("Thread is sleeping...");

Thread.sleep(2000);

System.out.println("Thread woke up.");

} catch (InterruptedException e) {

e.printStackTrace();

}

});

thread.start();

} }

**-------------------------------------------------- UNCHECKED EXCEPTIONS ------------------------------------------------**

Unchecked exceptions are exceptions that are not checked at compile-time. They are typically a result of programming errors, such as logic errors or improper use of an API. Unchecked exceptions are subclasses of RuntimeException.

**NullPointerException**

public class NullPointerExceptionExample {

public static void main(String[] args) {

String str = null;

try {

System.out.println(str.length());

} catch (NullPointerException e) {

e.printStackTrace();

} } }

**ArrayIndexOutOfBoundsException**

public class ArrayIndexOutOfBoundsExceptionExample {

public static void main(String[] args) {

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

try {

System.out.println(numbers[3]);

} catch (ArrayIndexOutOfBoundsException e) {

e.printStackTrace();

} } }

**ArithmeticException**

public class ArithmeticExceptionExample {

public static void main(String[] args) {

try {

int result = 10 / 0;

} catch (ArithmeticException e) {

e.printStackTrace();

} } }

**NumberFormatException**

public class NumberFormatExceptionExample {

public static void main(String[] args) {

try {

int number = Integer.parseInt("abc");

} catch (NumberFormatException e) {

e.printStackTrace();

}}}

**IllegalArgumentException**

public class IllegalArgumentExceptionExample {

public static void main(String[] args) {

try {

Thread thread = new Thread(() -> System.out.println("Thread running..."));

thread.setPriority(11); // Invalid priority

} catch (IllegalArgumentException e) {

e.printStackTrace();

}}}