**Exercise 1:**

**Write a Java program that demonstrates handling an ArithmeticException by catching and printing an error message.**

**Answer 1:**

public class ArithmeticExceptionExample {

public static void main(String[] args) {

try {

int result = 10 / 0; // This will throw an ArithmeticException

} catch (ArithmeticException e) {

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

}

}

}

**Exercise 2:**

**Create a program that reads an integer from the user and handles InputMismatchException when the input is not an integer.**

**Answer 2:**

import java.util.InputMismatchException;

import java.util.Scanner;

public class InputMismatchExample {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

try {

System.out.print("Enter an integer: ");

int num = scanner.nextInt();

System.out.println("You entered: " + num);

} catch (InputMismatchException e) {

System.out.println("Error: Invalid input. Please enter an integer.");

} finally {

scanner.close();

}

}

}

**Exercise 3:**

**Write a program that uses a custom exception class to handle a scenario where a user is too young to access a certain feature (age < 18).**

**Answer 3:**

class UnderAgeException extends Exception {

public UnderAgeException(String message) {

super(message);

}

}

public class CustomExceptionExample {

public static void main(String[] args) {

int userAge = 15;

try {

if (userAge < 18) {

throw new UnderAgeException("Access denied: User is too young.");

}

System.out.println("Access granted.");

} catch (UnderAgeException e) {

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

}

}

}

**Exercise 4:**

**Create a program that reads a file and handles FileNotFoundException when the specified file does not exist.**

**Answer 4:**

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

public class FileNotFoundExample {

public static void main(String[] args) {

try {

File file = new File("nonexistent.txt");

Scanner scanner = new Scanner(file);

while (scanner.hasNextLine()) {

System.out.println(scanner.nextLine());

}

scanner.close();

} catch (FileNotFoundException e) {

System.out.println("Error: File not found.");

}

}

}

**Exercise 5:**

**Write a program that demonstrates a nested exception, where an IOException is caught within an SQLException.**

**Answer 5:**

import java.io.IOException;

import java.sql.SQLException;

public class NestedExceptionExample {

public static void main(String[] args) {

try {

try {

throw new IOException("IOException occurred.");

} catch (IOException e) {

throw new SQLException("SQLException occurred.", e);

}

} catch (SQLException e) {

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

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

}

}

}

**Exercise 6:**

**Write a program that demonstrates a NullPointerException.**

**Answer 6:**

public class NullPointerExceptionExample {

public static void main(String[] args) {

String str = null;

try {

int length = str.length(); // This will throw a NullPointerException

} catch (NullPointerException e) {

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

}

}

}

**Exercise 7:**

**Create a program that divides two numbers and handles both ArithmeticException and NumberFormatException.**

**Answer 7:**

public class MultipleExceptionsExample {

public static void main(String[] args) {

try {

int numerator = Integer.parseInt(args[0]);

int denominator = Integer.parseInt(args[1]);

int result = numerator / denominator;

System.out.println("Result: " + result);

} catch (ArithmeticException e) {

System.out.println("Error: Division by zero.");

} catch (NumberFormatException e) {

System.out.println("Error: Invalid input format.");

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Error: Insufficient arguments.");

}

}

}

**Exercise 8:**

**Write a program that demonstrates using the finally block to close resources.**

**Answer 8:**

public class FinallyBlockExample {

public static void main(String[] args) {

try {

System.out.println("Try block");

} finally {

System.out.println("Finally block");

}

}

}

**Exercise 9:**

**Create a program with a custom exception class to handle a scenario where a user provides an invalid username.**

**Answer 9:**

class InvalidUsernameException extends Exception {

public InvalidUsernameException(String message) {

super(message);

}

}

public class CustomExceptionExample2 {

public static void main(String[] args) {

String username = "user@";

try {

if (!username.matches("[a-zA-Z0-9\_]+")) {

throw new InvalidUsernameException("Invalid username format.");

}

System.out.println("Username accepted.");

} catch (InvalidUsernameException e) {

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

}

}

}

**Exercise 10:**

**Write a program that demonstrates using multiple catch blocks for different exception types.**

**Answer 10:**

public class MultiCatchExample {

public static void main(String[] args) {

try {

int value = Integer.parseInt(args[0]);

int result = 10 / value;

System.out.println("Result: " + result);

} catch (ArithmeticException e) {

System.out.println("Error: Division by zero.");

} catch (NumberFormatException e) {

System.out.println("Error: Invalid input format.");

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Error: Insufficient arguments.");

} catch (Exception e) {

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

}

}

}

**Exercise 11:**

**Create a program that demonstrates throwing an exception explicitly using the throw keyword.**

**Answer 11:**

public class ThrowKeywordExample {

public static void main(String[] args) {

try {

int age = -5;

if (age < 0) {

throw new IllegalArgumentException("Age cannot be negative.");

}

System.out.println("Age: " + age);

} catch (IllegalArgumentException e) {

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

}

}

}

**Exercise 12:**

**Write a program that demonstrates chaining exceptions using the initCause method.**

**Answer 12:**

public class ChainedExceptionExample {

public static void main(String[] args) {

try {

try {

int result = 10 / 0; // This will throw an ArithmeticException

} catch (ArithmeticException e) {

throw new RuntimeException("Exception occurred.", e);

}

} catch (RuntimeException e) {

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

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

}

}

}

**Exercise 13:**

**Create a program that demonstrates using a throws clause in a method signature.**

**Answer 13:**

public class ThrowsClauseExample {

public static void main(String[] args) {

try {

validateAge(15);

} catch (IllegalArgumentException e) {

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

}

}

static void validateAge(int age) throws IllegalArgumentException {

if (age < 18) {

throw new IllegalArgumentException("Age should be at least 18.");

}

System.out.println("Age is valid.");

}

}

**Exercise 14:**

**Write a program that demonstrates rethrowing an exception from a catch block.**

**Answer 14:**

public class RethrowExceptionExample {

public static void main(String[] args) {

try {

processFile("missingfile.txt");

} catch (FileNotFoundException e) {

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

}

}

static void processFile(String fileName) throws FileNotFoundException {

try {

// Code to process the file

throw new FileNotFoundException("File not found: " + fileName);

} catch (FileNotFoundException e) {

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

throw e; // Rethrow the exception

}

}

}

**Exercise 15:**

**Write a program that demonstrates using the try-with-resources statement to automatically close resources.**

**Answer 15:**

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class TryWithResourcesExample {

public static void main(String[] args) {

try (BufferedReader reader = new BufferedReader(new FileReader("file.txt"))) {

String line;

while ((line = reader.readLine()) != null) {

System.out.println(line);

}

} catch (IOException e) {

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

}

}

}

**Exercise 16:**

**Create a program that demonstrates using a custom exception hierarchy with inheritance.**

**Answer 16:**

class BaseException extends Exception {

public BaseException(String message) {

super(message);

}

}

class DerivedException extends BaseException {

public DerivedException(String message) {

super(message);

}

}

public class ExceptionHierarchyExample {

public static void main(String[] args) {

try {

throw new DerivedException("Derived exception occurred.");

} catch (BaseException e) {

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

}

}

}

**Exercise 17:**

**Write a program that demonstrates catching multiple exceptions in a single catch block.**

**Answer 17:**

public class MultipleExceptionsSingleCatchExample {

public static void main(String[] args) {

try {

int value = Integer.parseInt(args[0]);

int result = 10 / value;

System.out.println("Result: " + result);

} catch (ArithmeticException | NumberFormatException e) {

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

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Error: Insufficient arguments.");

}

}

}

**Exercise 18:**

**Create a program that demonstrates creating and throwing a custom unchecked exception.**

**Answer 18:**

class CustomUncheckedException extends RuntimeException {

public CustomUncheckedException(String message) {

super(message);

}

}

public class UncheckedExceptionExample {

public static void main(String[] args) {

try {

throw new CustomUncheckedException("Custom unchecked exception occurred.");

} catch (CustomUncheckedException e) {

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

}

}

}

**Exercise 19:**

**Write a program that demonstrates using the assert statement to throw an AssertionError.**

**Answer 19:**

public class AssertionErrorExample {

public static void main(String[] args) {

int value = -5;

assert value >= 0 : "Value must be non-negative";

System.out.println("Value: " + value);

}

}

**Exercise 20:**

**Create a program that demonstrates handling a custom exception by rethrowing it as a built-in exception.**

**Answer 20:**

class CustomException extends Exception {

public CustomException(String message) {

super(message);

}

}

public class RethrowCustomExceptionExample {

public static void main(String[] args) {

try {

throwCustomException();

} catch (CustomException e) {

System.out.println("CustomException caught.");

}

}

static void throwCustomException() throws CustomException {

try {

throw new CustomException("Custom exception occurred.");

} catch (CustomException e) {

System.out.println("Rethrowing as RuntimeException.");

throw new RuntimeException(e);

}

}

}

**Exercise 21:**

**Write a program that demonstrates using the getSuppressed method to retrieve suppressed exceptions.**

**Answer 21:**

public class SuppressedExceptionExample {

public static void main(String[] args) {

try {

throw new Exception("Main exception");

} catch (Exception e) {

try {

throw new Exception("Suppressed exception 1");

} catch (Exception se1) {

e.addSuppressed(se1);

}

try {

throw new Exception("Suppressed exception 2");

} catch (Exception se2) {

e.addSuppressed(se2);

}

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

for (Throwable suppressed : e.getSuppressed()) {

System.out.println("Suppressed: " + suppressed.getMessage());

}

}

}

}

**Exercise 22:**

**Create a program that demonstrates using a custom exception with constructor overloading.**

**Answer 22:**

class CustomException2 extends Exception {

public CustomException2() {

super();

}

public CustomException2(String message) {

super(message);

}

}

public class CustomExceptionOverloadingExample {

public static void main(String[] args) {

try {

throw new CustomException2("Custom exception with message.");

} catch (CustomException2 e) {

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

}

}

}

**Exercise 23:**

**Write a program that demonstrates using the getStackTrace method to print the call stack.**

**Answer 23:**

public class StackTraceExample {

public static void main(String[] args) {

try {

divideByZero();

} catch (ArithmeticException e) {

e.printStackTrace();

}

}

static void divideByZero() {

int result = 10 / 0; // This will throw an ArithmeticException

}

}

**Exercise 24:**

**Create a program that demonstrates creating and throwing a custom checked exception.**

**Answer 24:**

class CustomCheckedException extends Exception {

public CustomCheckedException(String message) {

super(message);

}

}

public class CheckedExceptionExample {

public static void main(String[] args) {

try {

throw new CustomCheckedException("Custom checked exception occurred.");

} catch (CustomCheckedException e) {

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

}

}

}

**Exercise 25:**

**Write a program that demonstrates catching an exception using a try block without a catch block but with a finally block.**

**Answer 25:**

public class TryFinallyExample {

public static void main(String[] args) {

try {

int result = 10 / 0; // This will throw an ArithmeticException

} finally {

System.out.println("Finally block executed.");

}

}

}

**Exercise 26:**

**Write a program that demonstrates creating and throwing a custom runtime exception.**

**Answer 26:**

class CustomRuntimeException extends RuntimeException {

public CustomRuntimeException(String message) {

super(message);

}

}

public class CustomRuntimeExceptionExample {

public static void main(String[] args) {

try {

throw new CustomRuntimeException("Custom runtime exception occurred.");

} catch (CustomRuntimeException e) {

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

}

}

}

**Exercise 27:**

**Create a program that demonstrates handling exceptions in multi-level method calls.**

**Answer 27:**

public class MultiLevelExceptionExample {

public static void main(String[] args) {

try {

outerMethod();

} catch (Exception e) {

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

}

}

static void outerMethod() {

middleMethod();

}

static void middleMethod() {

innerMethod();

}

static void innerMethod() {

int result = 10 / 0; // This will throw an ArithmeticException

}

}

**Exercise 28:**

**Write a program that demonstrates catching an exception using a try block with multiple catch blocks.**

**Answer 28:**

public class MultipleCatchBlocksExample {

public static void main(String[] args) {

try {

int value = Integer.parseInt(args[0]);

int result = 10 / value;

System.out.println("Result: " + result);

} catch (ArithmeticException e) {

System.out.println("Error: Division by zero.");

} catch (NumberFormatException e) {

System.out.println("Error: Invalid input format.");

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Error: Insufficient arguments.");

}

}

}

**Exercise 29:**

**Create a program that demonstrates using the printStackTrace method to print exception information.**

**Answer 29:**

public class PrintStackTraceExample {

public static void main(String[] args) {

try {

int result = 10 / 0; // This will throw an ArithmeticException

} catch (ArithmeticException e) {

e.printStackTrace();

}

}

}

**Exercise 30:**

**Write a program that demonstrates using the getCause method to retrieve the root cause of an exception.**

**Answer 30:**

public class CauseExceptionExample {

public static void main(String[] args) {

try {

try {

int result = 10 / 0; // This will throw an ArithmeticException

} catch (ArithmeticException e) {

throw new RuntimeException("Runtime exception occurred.", e);

}

} catch (RuntimeException e) {

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

Throwable cause = e.getCause();

if (cause != null) {

System.out.println("Root cause: " + cause.getMessage());

}

}

}

}

**Exercise 31:**

**Write a program that demonstrates handling exceptions using a try block with a catch block for a specific subclass of an exception.**

**Answer 31:**

public class SpecificExceptionCatchExample {

public static void main(String[] args) {

try {

int result = Integer.parseInt("abc"); // This will throw a NumberFormatException

System.out.println("Result: " + result);

} catch (NumberFormatException e) {

System.out.println("Error: Invalid number format.");

}

}

}

**Exercise 32:**

**Create a program that demonstrates catching an exception using a try block with multiple catch blocks in a specific order.**

**Answer 32:**

public class OrderedCatchBlocksExample {

public static void main(String[] args) {

try {

int value = Integer.parseInt(args[0]);

int result = 10 / value;

System.out.println("Result: " + result);

} catch (ArithmeticException e) {

System.out.println("Error: Division by zero.");

} catch (NumberFormatException e) {

System.out.println("Error: Invalid input format.");

} catch (Exception e) {

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

}

}

}

**Exercise 33:**

**Write a program that demonstrates using the getLocalizedMessage method to retrieve the localized error message of an exception.**

**Answer 33:**

public class LocalizedMessageExample {

public static void main(String[] args) {

try {

int result = 10 / 0; // This will throw an ArithmeticException

} catch (ArithmeticException e) {

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

}

}

}

**Exercise 34:**

**Create a program that demonstrates creating and using a chained exception.**

**Answer 34:**

class ChainedException1 extends Exception {

public ChainedException1(String message) {

super(message);

}

}

class ChainedException2 extends Exception {

public ChainedException2(String message) {

super(message);

}

}

public class ChainedExceptionUsageExample {

public static void main(String[] args) {

try {

try {

throw new ChainedException1("Chained exception 1");

} catch (ChainedException1 e) {

throw new ChainedException2("Chained exception 2", e);

}

} catch (ChainedException2 e) {

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

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

}

}

}

**Exercise 35:**

**Write a program that demonstrates using a custom exception with a constructor that takes a cause.**

**Answer 35:**

class CustomExceptionWithCause extends Exception {

public CustomExceptionWithCause(String message, Throwable cause) {

super(message, cause);

}

}

public class ExceptionWithCauseExample {

public static void main(String[] args) {

try {

try {

int result = 10 / 0; // This will throw an ArithmeticException

} catch (ArithmeticException e) {

throw new CustomExceptionWithCause("Custom exception with cause", e);

}

} catch (CustomExceptionWithCause e) {

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

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

}

}

}

**Exercise 36:**

**Create a program that demonstrates throwing an exception inside a loop and catching it outside the loop.**

**Answer 36:**

public class ExceptionInLoopExample {

public static void main(String[] args) {

try {

for (int i = 5; i >= -5; i--) {

int result = 10 / i; // This will throw an ArithmeticException for i=0

}

} catch (ArithmeticException e) {

System.out.println("Error: Division by zero.");

}

}

}

**Exercise 37:**

**Write a program that demonstrates handling multiple exceptions using a single catch block with instanceof.**

**Answer 37:**

public class MultipleExceptionsInstanceOfExample {

public static void main(String[] args) {

try {

int value = Integer.parseInt(args[0]);

int result = 10 / value;

System.out.println("Result: " + result);

} catch (Exception e) {

if (e instanceof ArithmeticException) {

System.out.println("Error: Division by zero.");

} else if (e instanceof NumberFormatException) {

System.out.println("Error: Invalid input format.");

} else if (e instanceof ArrayIndexOutOfBoundsException) {

System.out.println("Error: Insufficient arguments.");

} else {

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

}

}

}

}

**Exercise 38:**

**Create a program that demonstrates using the fillInStackTrace method to rethrow an exception with a new stack trace.**

**Answer 38:**

public class FillInStackTraceExample {

public static void main(String[] args) {

try {

int result = 10 / 0; // This will throw an ArithmeticException

} catch (ArithmeticException e) {

e.printStackTrace();

ArithmeticException newException = (ArithmeticException) e.fillInStackTrace();

newException.printStackTrace();

}

}

}

**Exercise 39:**

**Write a program that demonstrates handling an exception using a method with a throws clause.**

**Answer 39:**

public class ThrowsClauseMethodExample {

public static void main(String[] args) {

try {

performDivision(10, 0); // This will throw an ArithmeticException

} catch (ArithmeticException e) {

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

}

}

static void performDivision(int numerator, int denominator) throws ArithmeticException {

int result = numerator / denominator;

System.out.println("Result: " + result);

}

}

**Exercise 40:**

**Create a program that demonstrates using a custom exception class with overloaded constructors.**

**Answer 40:**

class CustomExceptionOverloaded extends Exception {

public CustomExceptionOverloaded(String message) {

super(message);

}

public CustomExceptionOverloaded(String message, Throwable cause) {

super(message, cause);

}

}

public class CustomExceptionOverloadedExample {

public static void main(String[] args) {

try {

try {

int result = 10 / 0; // This will throw an ArithmeticException

} catch (ArithmeticException e) {

throw new CustomExceptionOverloaded("Custom exception with message", e);

}

} catch (CustomExceptionOverloaded e) {

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

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

}

}

}

**Exercise 41:**

**Write a program that demonstrates handling an exception using nested try blocks.**

**Answer 41:**

public class NestedTryBlocksExample {

public static void main(String[] args) {

try {

try {

int result = 10 / 0; // This will throw an ArithmeticException

} catch (ArithmeticException e) {

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

}

} catch (Exception e) {

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

}

}

}

**Exercise 42:**

**Create a program that demonstrates catching and rethrowing an exception using a catch block.**

**Answer 42:**

public class RethrowExceptionInCatchExample {

public static void main(String[] args) {

try {

int result = 10 / 0; // This will throw an ArithmeticException

} catch (ArithmeticException e) {

System.out.println("Caught and rethrown.");

throw e; // Rethrow the exception

}

}

}

**Exercise 43:**

**Write a program that demonstrates catching and rethrowing an exception using a finally block.**

**Answer 43:**

public class RethrowExceptionInFinallyExample {

public static void main(String[] args) {

try {

int result = 10 / 0; // This will throw an ArithmeticException

} catch (ArithmeticException e) {

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

} finally {

System.out.println("Finally block executed.");

throw new RuntimeException("Exception in finally block");

}

}

}

**Exercise 44:**

**Create a program that demonstrates throwing a custom unchecked exception with a cause.**

**Answer 44:**

class CustomUncheckedExceptionWithCause extends RuntimeException {

public CustomUncheckedExceptionWithCause(String message, Throwable cause) {

super(message, cause);

}

}

public class UncheckedExceptionWithCauseExample {

public static void main(String[] args) {

try {

try {

int result = 10 / 0; // This will throw an ArithmeticException

} catch (ArithmeticException e) {

throw new CustomUncheckedExceptionWithCause("Custom unchecked exception", e);

}

} catch (CustomUncheckedExceptionWithCause e) {

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

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

}

}

}

**Exercise 45:**

**Write a program that demonstrates catching and handling multiple exceptions using a catch block with | (OR) in Java 7.**

**Answer 45:**

public class MultiExceptionOrInCatchExample {

public static void main(String[] args) {

try {

int value = Integer.parseInt(args[0]);

int result = 10 / value;

System.out.println("Result: " + result);

} catch (ArithmeticException | NumberFormatException e) {

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

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Error: Insufficient arguments.");

}

}

}

**Exercise 46:**

**Create a program that demonstrates throwing an exception inside a loop and catching it within the loop.**

**Answer 46:**

public class ExceptionInLoopWithCatchExample {

public static void main(String[] args) {

for (int i = 5; i >= -5; i--) {

try {

int result = 10 / i; // This will throw an ArithmeticException for i=0

} catch (ArithmeticException e) {

System.out.println("Error: Division by zero.");

}

}

}

}

**Exercise 47:**

**Write a program that demonstrates using a custom exception with a toString method.**

**Answer 47:**

class CustomExceptionWithToString extends Exception {

public CustomExceptionWithToString(String message) {

super(message);

}

@Override

public String toString() {

return "CustomExceptionWithToString: " + getMessage();

}

}

public class ExceptionToStringExample {

public static void main(String[] args) {

try {

throw new CustomExceptionWithToString("Custom exception with toString");

} catch (CustomExceptionWithToString e) {

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

}

}

}

**Exercise 48:**

**Create a program that demonstrates using the getStackTrace method to retrieve and print the call stack.**

**Answer 48:**

public class StackTraceUsageExample {

public static void main(String[] args) {

try {

divideByZero();

} catch (ArithmeticException e) {

StackTraceElement[] stackTrace = e.getStackTrace();

for (StackTraceElement element : stackTrace) {

System.out.println("Class: " + element.getClassName());

System.out.println("Method: " + element.getMethodName());

System.out.println("File: " + element.getFileName());

System.out.println("Line: " + element.getLineNumber());

System.out.println();

}

}

}

static void divideByZero() {

int result = 10 / 0; // This will throw an ArithmeticException

}

}

**Exercise 49:**

**Write a program that demonstrates creating and handling a custom exception with multiple constructors.**

**Answer 49:**

class CustomExceptionMultipleConstructors extends Exception {

public CustomExceptionMultipleConstructors() {

super();

}

public CustomExceptionMultipleConstructors(String message) {

super(message);

}

public CustomExceptionMultipleConstructors(Throwable cause) {

super(cause);

}

public CustomExceptionMultipleConstructors(String message, Throwable cause) {

super(message, cause);

}

}

public class ExceptionMultipleConstructorsExample {

public static void main(String[] args) {

try {

try {

int result = 10 / 0; // This will throw an ArithmeticException

} catch (ArithmeticException e) {

throw new CustomExceptionMultipleConstructors("Custom exception with multiple constructors", e);

}

} catch (CustomExceptionMultipleConstructors e) {

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

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

}

}

}

**Exercise 50:**

**Create a program that demonstrates using the initCause method to associate a cause with an exception.**

**Answer 50:**

public class InitCauseExample {

public static void main(String[] args) {

try {

try {

int result = 10 / 0; // This will throw an ArithmeticException

} catch (ArithmeticException e) {

RuntimeException customException = new RuntimeException("Custom exception");

customException.initCause(e);

throw customException;

}

} catch (RuntimeException e) {

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

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

}

}

}

**Exercise 51:**

**Create a program that demonstrates throwing a custom exception when a user provides an invalid email address.**

**Solution 51:**

class InvalidEmailException extends Exception {

public InvalidEmailException(String message) {

super(message);

}

}

public class InvalidEmailExample {

public static void main(String[] args) {

try {

String email = "invalid-email";

if (!email.contains("@")) {

throw new InvalidEmailException("Invalid email address.");

}

System.out.println("Email accepted: " + email);

} catch (InvalidEmailException e) {

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

}

}

}

**Exercise 52:**

**Write a program that demonstrates throwing a custom exception when a user tries to withdraw more money from an account than the available balance.**

**Solution 52:**

class InsufficientBalanceException extends Exception {

public InsufficientBalanceException(String message) {

super(message);

}

}

class BankAccount {

private double balance;

public BankAccount(double initialBalance) {

balance = initialBalance;

}

public void withdraw(double amount) throws InsufficientBalanceException {

if (amount > balance) {

throw new InsufficientBalanceException("Insufficient balance.");

}

balance -= amount;

System.out.println("Withdrawal successful. Remaining balance: " + balance);

}

}

public class BankAccountExample {

public static void main(String[] args) {

BankAccount account = new BankAccount(1000);

try {

account.withdraw(1500);

} catch (InsufficientBalanceException e) {

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

}

}

}

**Exercise 53:**

**Create a program that demonstrates using a user-defined exception for validating a password that must contain at least one uppercase letter.**

**Solution 53:**

class WeakPasswordException extends Exception {

public WeakPasswordException(String message) {

super(message);

}

}

public class PasswordValidationExample {

public static void main(String[] args) {

try {

String password = "password123";

if (!password.matches(".\*[A-Z].\*")) {

throw new WeakPasswordException("Password must contain at least one uppercase letter.");

}

System.out.println("Password accepted.");

} catch (WeakPasswordException e) {

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

}

}

}

**Exercise 54:**

**Write a program that demonstrates throwing a custom exception for a student registration system, where a student must provide a valid student ID.**

**Solution 54:**

class InvalidStudentIdException extends Exception {

public InvalidStudentIdException(String message) {

super(message);

}

}

class StudentRegistration {

public void registerStudent(String studentId) throws InvalidStudentIdException {

if (!studentId.matches("[0-9]{5}")) {

throw new InvalidStudentIdException("Invalid student ID format.");

}

System.out.println("Student registered successfully.");

}

}

public class StudentRegistrationExample {

public static void main(String[] args) {

StudentRegistration registration = new StudentRegistration();

try {

registration.registerStudent("123456"); // Invalid student ID format

} catch (InvalidStudentIdException e) {

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

}

}

}

**Exercise 55:**

**Create a program that demonstrates throwing a custom exception when a user tries to access a restricted area without proper authorization.**

**Solution 55:**

class UnauthorizedAccessException extends Exception {

public UnauthorizedAccessException(String message) {

super(message);

}

}

class RestrictedArea {

public void access(String username) throws UnauthorizedAccessException {

if (!username.equals("admin")) {

throw new UnauthorizedAccessException("Unauthorized access.");

}

System.out.println("Access granted.");

}

}

public class RestrictedAreaExample {

public static void main(String[] args) {

RestrictedArea area = new RestrictedArea();

try {

area.access("user"); // Unauthorized access

} catch (UnauthorizedAccessException e) {

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

}

}

}

**Exercise 56:**

**Write a program that demonstrates using a user-defined exception for validating a product code that must start with a letter followed by four digits.**

**Solution 56:**

class InvalidProductCodeException extends Exception {

public InvalidProductCodeException(String message) {

super(message);

}

}

public class ProductCodeValidationExample {

public static void main(String[] args) {

try {

String productCode = "A1234";

if (!productCode.matches("[A-Z][0-9]{4}")) {

throw new InvalidProductCodeException("Invalid product code format.");

}

System.out.println("Product code accepted.");

} catch (InvalidProductCodeException e) {

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

}

}

}

**Exercise 57:**

**Create a program that demonstrates throwing a custom exception when a user provides an invalid phone number format.**

**Solution 57:**

class InvalidPhoneNumberException extends Exception {

public InvalidPhoneNumberException(String message) {

super(message);

}

}

public class PhoneNumberValidationExample {

public static void main(String[] args) {

try {

String phoneNumber = "123-456"; // Invalid phone number format

if (!phoneNumber.matches("\\d{3}-\\d{3}-\\d{4}")) {

throw new InvalidPhoneNumberException("Invalid phone number format.");

}

System.out.println("Phone number accepted: " + phoneNumber);

} catch (InvalidPhoneNumberException e) {

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

}

}

}

**Exercise 58:**

**Write a program that demonstrates using a user-defined exception for a library checkout system, where a book must be available to be checked out.**

**Solution 58:**

class BookUnavailableException extends Exception {

public BookUnavailableException(String message) {

super(message);

}

}

class Library {

private boolean bookAvailable = false;

public void checkOutBook() throws BookUnavailableException {

if (!bookAvailable) {

throw new BookUnavailableException("Book is not available for checkout.");

}

System.out.println("Book checked out.");

bookAvailable = false;

}

public void returnBook() {

bookAvailable = true;

System.out.println("Book returned.");

}

}

public class LibraryCheckoutExample {

public static void main(String[] args) {

Library library = new Library();

library.returnBook();

try {

library.checkOutBook(); // Book is not available

} catch (BookUnavailableException e) {

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

}

}

}

**Exercise 59:**

**Create a program that demonstrates using a user-defined exception for a ticket booking system, where a customer must provide a valid age.**

**Solution 59:**

class InvalidAgeException extends Exception {

public InvalidAgeException(String message) {

super(message);

}

}

class TicketBooking {

public void bookTicket(int age) throws InvalidAgeException {

if (age < 0 || age > 120) {

throw new InvalidAgeException("Invalid age.");

}

System.out.println("Ticket booked successfully.");

}

}

public class TicketBookingExample {

public static void main(String[] args) {

TicketBooking booking = new TicketBooking();

try {

booking.bookTicket(150); // Invalid age

} catch (InvalidAgeException e) {

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

}

}

}

**Exercise 60:**

**Write a program that demonstrates using a user-defined exception for a classroom grading system, where a student's grade must be within a valid range.**

**Solution 60:**

class InvalidGradeException extends Exception {

public InvalidGradeException(String message) {

super(message);

}

}

class Classroom {

public void assignGrade(int grade) throws InvalidGradeException {

if (grade < 0 || grade > 100) {

throw new InvalidGradeException("Invalid grade.");

}

System.out.println("Grade assigned: " + grade);

}

}

public class ClassroomGradingExample {

public static void main(String[] args) {

Classroom classroom = new Classroom();

try {

classroom.assignGrade(-5); // Invalid grade

} catch (InvalidGradeException e) {

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

}

}

}

All the best.