Task Description:

Java Exceptions and Error Handling

1. What are the four access modifiers available in Java and what is their significance in

terms of class, method, and variable accessibility?

1. **Public**: Public class, method or variable can be accessed from anywhere in the program. This is used for components that need to be used in other parts of the code.
2. **Private**: Private class, methos or variable is opposite of public class which provide restricted access, this can be accessed only within the class it is declared and cannot be accessed from any other part of the code.
3. **Protected**: Protected class, method or variable provide access within the same package where the class is defined and in the subclasses in other packages. It is used for shared functionality between class and its inheritors.
4. **Default(Package-Private)**: This is the default access modifier if none of the above is specified. This can be accessed by other classes within the same package, but for other packages it is hidden.

2. What is the difference between Exception and error?

|  |  |
| --- | --- |
| **Exception** | **Error** |
| * Exceptions are recoverable events. * This is caused by program’s logic or unexpected user input. * The program can handle exceptions using try-catch blocks to intercept it and take action on it. * Exceptions often arise from problem within the code. * Example: Accessing element outside array’s bounds, converting string to number with invalid characters. * Exceptions are categorized as checked or unchecked. * Checked exceptions must be declared in the method, forcing programmer to anticipate and handle them. * Unchecked exceptions are not required to be declared but still considered for robust error handling. * Exception is a subclass of Throwable class in Java. Exception belong to **java.lang.Exception** class hierarchy. | * Errors are unrecoverable events. * This is caused by system malfunction or corrupt data. * The program cannot be able to continue execution and may need to be terminated. * Error arise from external factors outside the code. * Example: Running out of memory, disk errors, network issues. * Errors are not designed to be caught in the program. * Errors are logged for debugging purposes and the program is terminated upon encountering error. * Error is a subclass of Throwable class in Java. * Error belong to **java.lang.Error** class hierarchy. |

3. What is the difference between checked Exception and unchecked Exception?

|  |  |
| --- | --- |
| **Checked exception** | **Unchecked exception** |
| * Checked exceptions must be declared in the method, forcing programmer to anticipate and handle them. * Checked exceptions are enforced at compile time. * Checked exception indicates recoverable issues related to external resources or expected program conditions. * The compiler ensures the method whether it handles them or declares it using throws keyword. * Must be handled using try-catch block or by declaring throws keyword. * Example: IOException, FileNotFoundException, InterruptedException. | * Unchecked exceptions are not required to be declared but still considered for robust error handling. * Unchecked exceptions are not checked at compile time. * Unchecked exception represents programming errors or unexpected conditions within the program. * The compiler allows the code to compile without handling the. * Not mandatory to handle, but recommended for robust error handling * Example: ArithmeticExceprion, NullPointerException, IndexOutofBoundsException. |

4. Write a Java program that reads user input for two integers and performs division. Handle

the exception that is thrown when the second number is zero, and display an error

message to the user.

import java.util.Scanner;

public class DivisionException {

public static void main(String[] args) {

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

System.***out***.println("Enter two integers: ");

int num1=sc.nextInt();

int num2=sc.nextInt();

try {

int result = num1/num2;

System.***out***.println("The result is: "+result);

}catch(ArithmeticException e){ // ArithmeticException is used to handle the exception

System.***out***.println("Error: Division by zero is not allowed.");

}

}

}

Output:

Enter two integers:

12

0

Error: Division by zero is not allowed.

5. Write the code of ArrayIndexOutOfBoundsException &

StringIndexOutOfBoundsException?

//importing required package

import java.io.\*;

import java.lang.\*;

import java.util.\*;

public class IndexExceptionsExample {

public static void main(String[] args) {

try {

int[] array = new int[5];

System.***out***.println(array[5]);

//accessing an index greater than length of array

}catch(ArrayIndexOutOfBoundsException e) {

System.***out***.println("Error: Array index is out of bounds.");

}

try {

String str ="Vinoth";

System.***out***.println(str.charAt(6));

//accessing index greater than string length

}catch(StringIndexOutOfBoundsException e) {

System.***out***.println("Error: String index is out of bounds.");

}

}

}

Output:

Error: Array index is out of bounds.

Error: String index is out of bounds.

6. You are building a login system for a website using Java. If the user enters an incorrect password, you want to display a message informing them of the error. How would you use exception handling to handle this situation?

import java.util.Scanner;

public class LoginSystem {

private String correctPassword = "password123"; // predefined correct password

public void login(String enteredPassword) {

try {

checkPassword(enteredPassword);

// proceed with the login if user input matches predefined password

System.***out***.println("Login successful!");

} catch (Exception e) {

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

}

}

private void checkPassword(String enteredPassword) throws Exception {

if (!enteredPassword.equals(correctPassword)) {

throw new Exception("Incorrect password. Please try again.");

//print incorrect password if user input does not match predefined password

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.***in***);// reading input from user

System.***out***.println("Enter your password:");

String enteredPassword = scanner.next();

LoginSystem loginSystem = new LoginSystem();

loginSystem.login(enteredPassword);

}

}

Output:

Enter your password:

123password

Incorrect password. Please try again.

7. Create a custom exception in Java called "InvalidAgeException" that is thrown when the user enters an age less than 18. Implement exception handling in a Java program to catch the "InvalidAgeException" and display an error message.

import java.util.Scanner;

// InvalidAgeException

class InvalidAgeException extends Exception {

public InvalidAgeException(String message) {

super(message);

}

}

// Main Class

public class AgeValidator {

public static void main(String[] args) {

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

System.***out***.println("Enter your age:");

int age = scanner.nextInt();

try {

*checkAge*(age);

} catch (InvalidAgeException e) {

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

}

}

public static void checkAge(int age) throws InvalidAgeException {

if (age < 18) {

throw new InvalidAgeException("Age must be 18 or above.");

} else {

System.***out***.println("Valid age.");

}

}

}

Output:

Enter your age:

17

Age must be 18 or above.

8. Implement exception handling in a Java program that reads data from a file. If the file does not exist, throw a "FileNotFoundException" and display an error message to the user.

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

public class FileSearch {

public static void main(String[] args) {

try {

File file= new File("filename.txt");//entering file name

Scanner sc = new Scanner(file);

while(sc.hasNextLine()) {

String line= sc.nextLine();

System.out.println(line);

}

}catch(FileNotFoundException e) {

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

}

}

}Output:

Error: File not found.