**Java Exceptions and Error Handling**

Task Description:

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

terms of class, method, and variable accessibility?

Java provides four access modifiers:

**1. public:**

* Members marked **public** are accessible from **anywhere** in your program, regardless of the package or class they belong to.
* Commonly used for classes, methods, and variables that need to be used by other parts of your application.

**2. private:**

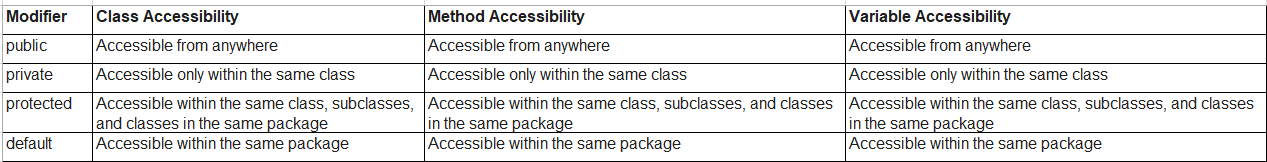
* Members marked **private** are **only accessible within the same class** where they are declared.
* Useful for encapsulating internal implementation details and protecting data from unintended access.

**3. Protected:**

* Members marked **protected** are accessible within the same class, all **subclasses** of that class (regardless of package), and **classes in the same package**.
* Used for providing limited access to specific methods or variables to subclasses while maintaining encapsulation.

**4. default (no keyword):**

* Members without an explicit access modifier have **default access.**
* They are accessible within the same package but **not** from other packages.
* Commonly used for members intended to be used within a specific module or set of collaborating classes



2. What is the difference between Exception and error?

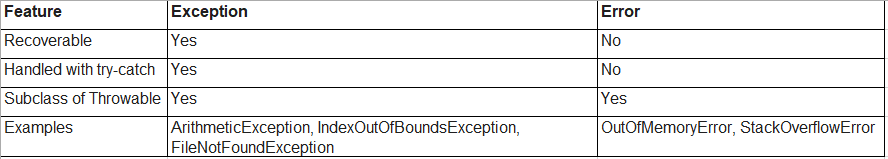
Both exceptions and errors represent problems that can occur during program execution in Java, but they differ in their nature and handling:

**1. Exceptions:**

* **Recoverable:** Exceptions signify **expected** problems that can be handled within the program code. They are thrown when conditions go wrong during normal execution, like trying to access an out-of-bounds array element or encountering a file not found.
* **Handled using try-catch blocks:** You can use try-catch blocks to anticipate and handle specific exceptions, preventing the program from crashing.
* **Subclasses of Throwable:** All exceptions are subclasses of thejava.lang.Throwableclass, inheriting methods for handling and providing information about the error.
* **Examples:** ArithmeticException, IndexOutOfBoundsException, FileNotFoundException, etc.

**2. Errors:**

* **Unrecoverable:** Errors represent **unexpected** problems that usually indicate severe issues beyond the program's control, like insufficient memory or system-level failures.
* **Not generally handled:** Due to their serious nature, recovering from errors within the program is typically not possible. They often lead to program termination.
* **Subclasses of Error:** Errors are also subclasses of java.lang.Throwable**,** but they are separate from the exception hierarchy.
* **Examples:** OutOfMemoryError, StackOverflowError, etc.



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

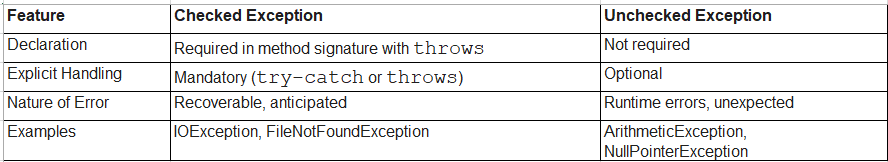
Java distinguishes between two types of exceptions based on how they are handled at compile time: **checked exceptions** and **unchecked exceptions**.

**1. Checked Exceptions:**

* **Declared** in the method signature using the throws keyword.
* **Force developers to handle** them explicitly using either a try-catch block or declaring them in the throws clause of the method signature.
* Designed to handle **recoverable** situations that the developer should anticipate and address to ensure program stability.
* Examples: IOException, FileNotFoundException, InterruptedException.

**2. Unchecked Exceptions:**

* **Not declared** in the method signature.
* **Do not require explicit handling** within the method itself.
* Represent **runtime errors** or unexpected conditions that are less predictable and often beyond the developer's immediate control.
* Examples: ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException.

****

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.

| **package** trainingtaskcompletion;  **import** java.util.Scanner;  **public** **class** DivisionbyZero {  **public** **static** **void** main(String[] args) {  // Create a Scanner object to read user input  Scanner scanner = **new** Scanner(System.***in***);  **try** {  // Prompt the user to enter two integers  System.***out***.print("Enter the first integer: ");  **int** numerator = scanner.nextInt();  System.***out***.print("Enter the second integer: ");  **int** denominator = scanner.nextInt();  // Perform division and display the result  **double** result = *divide*(numerator, denominator);  System.***out***.println("Result of division: " + result);  } **catch** (ArithmeticException e) {  // Handle the exception when the denominator is zero  System.***out***.println("Error: Division by zero is not allowed.");  } **finally** {  // Close the scanner to release resources  scanner.close();  }  }  // Method to perform division  **public** **static** **double** divide(**int** numerator, **int** denominator) {  // Check if the denominator is zero  **if** (denominator == 0) {  **throw** **new** ArithmeticException("Division by zero is not allowed.");  }  // Perform division and return the result  **return** (**double**) numerator / denominator;  }  } |
| --- |

**Output:-**

Enter the first integer: 10

Enter the second integer: 0

Error: Division by zero is not allowed.

Enter the first integer: 10

Enter the second integer: 2

Result of division: 5.0

5. Write the code of ArrayIndexOutOfBoundsException, StringIndexOutOfBoundsException?

| **package** trainingtaskcompletion;  **import** java.util.Scanner;  **public** **class** OutofBound {  **public** **static** **void** main(String[] args) {  // Create a Scanner object to read user input  Scanner scanner = **new** Scanner(System.***in***);  **try** {  // Example causing ArrayIndexOutOfBoundsException  System.***out***.print("Enter an index to access in the array: ");  **int** index1 = scanner.nextInt();  **int**[] array = { 1, 2, 3 };  **int** element1 = array[index1]; // Throws ArrayIndexOutOfBoundsException  System.***out***.println("Element at index " + index1 + " is: " + element1);  } **catch** (ArrayIndexOutOfBoundsException e) {  System.***out***.println("Caught ArrayIndexOutOfBoundsException: " + e.getMessage());  }  **try** {  // Example causing StringIndexOutOfBoundsException  System.***out***.print("Enter an index to access in the string: ");  **int** index2 = scanner.nextInt();  String str = "Hello";  **char** ch = str.charAt(index2); // Throws StringIndexOutOfBoundsException  System.***out***.println("Character at index " + index2 + " is: " + ch);  } **catch** (StringIndexOutOfBoundsException e) {  System.***out***.println("Caught StringIndexOutOfBoundsException: " + e.getMessage());  } **finally** {  // Close the scanner to release resources  scanner.close();  }  }  } |
| --- |

**Output:-**

Enter an index to access in the array: 10

Caught ArrayIndexOutOfBoundsException: Index 10 out of bounds for length 3

Enter an index to access in the string: 90

Caught StringIndexOutOfBoundsException: Index 90 out of bounds for length 5

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?

| **package** trainingtaskcompletion;  **import** java.util.Scanner;  **public** **class** EnterPwd {  **public** **static** **void** main(String[] args) {  // Create a Scanner object to read user input  Scanner scanner = **new** Scanner(System.***in***);  // Prompt the user to enter the password  System.***out***.print("Enter your password: ");  String enteredPassword = scanner.nextLine();  // create an object to the class loginSystem  LoginSystem loginSystem = **new** LoginSystem();  **try** {  // call the login system class to check the password is valid or not  loginSystem.login(enteredPassword);  }  // catch the bubbled InvalidPasswordException  **catch** (InvalidPasswordException e) {  System.***out***.println("Error: " + e.getMessage());  } **finally** {  // Close the scanner to release resources  scanner.close();  }  }  } |
| --- |

| **package** trainingtaskcompletion;  **public** **class** LoginSystem {  // Define the correct password as a constant  **private** **static** **final** String ***CORRECT\_PASSWORD*** = "correctPassword";  // Method to perform login  **public** **void** login(String enteredPassword) **throws** InvalidPasswordException {  // Check if the entered password matches the correct password  **if** (!enteredPassword.equals(***CORRECT\_PASSWORD***)) {  // If not, throw an InvalidPasswordException with an error message  **throw** **new** InvalidPasswordException("Incorrect password. Please try again.");  } **else** {  // If the passwords match, print a success message  System.***out***.println("Login successful!");  }  }  } |
| --- |

| **package** trainingtaskcompletion;  **import** java.lang.Exception;  // Import the necessary package for Exception class  @SuppressWarnings("serial")  **public** **class** InvalidPasswordException **extends** Exception {  // Constructor to create a new instance of InvalidPasswordException  **public** InvalidPasswordException(String message) {  // Call the constructor of the superclass (Exception) with the provided message  **super**(message);  }  } |
| --- |

**Output:-**

Enter your password: correctPassword

Login successful!

Enter your password: \*\*\*\*\*\*\*

Error: 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.

| **package** trainingtaskcompletion;  **import** java.util.Scanner; // Import the Scanner class from java.util package  // Main class to demonstrate exception handling  **public** **class** AgeValidation {  // Method to validate the age  **public** **static** **void** validateAge(**int** age) **throws** InvalidAgeException {  // Check if the age is less than 18  **if** (age < 18) {  // If age is less than 18, throw the custom exception  **throw** **new** InvalidAgeException("Age must be 18 or older.");  }  }  // Main method to handle exception  **public** **static** **void** main(String[] args) {  // Create a Scanner object to read user input  Scanner scanner = **new** Scanner(System.***in***);  // Prompt the user to enter their age  System.***out***.print("Enter your age: ");  // Try block to catch the exception  **try** {  // Read the age input from the user  **int** age = scanner.nextInt();  // Validate the age  *validateAge*(age);  // If age is valid, display success message  System.***out***.println("Age is valid: " + age);  } **catch** (InvalidAgeException e) {  // If InvalidAgeException is caught, display error message  System.***out***.println("Error: " + e.getMessage());  } **catch** (Exception e) {  // If any other exception occurs, display a generic error message  System.***out***.println("Error: Invalid input. Please enter a valid age.");  } **finally** {  // Close the Scanner object to prevent resource leak  scanner.close();  }  }  } |
| --- |

| **package** trainingtaskcompletion;  **import** java.lang.Exception;  //Import the necessary package for Exception class  @SuppressWarnings("serial")  **public** **class** InvalidAgeException **extends** Exception {  // Constructor to initialize the exception with a custom message  **public** InvalidAgeException(String message) {  **super**(message);  }  } |
| --- |

**Output:-**

Enter your age: -10

Error: Age must be 18 or older.

Enter your age: 18

Age is valid: 18

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.

| **package** trainingtaskcompletion;  **import** java.io.File;  **import** java.util.Scanner;  **public** **class** FileNotFound {  **public** **static** **void** main(String[] args) {  Scanner scanner = **new** Scanner(System.***in***); // Create a Scanner object to read user input  System.***out***.print("Enter the file path: "); // Prompt the user to enter the file path  String filePath = scanner.nextLine(); // Read the file path entered by the user  File file = **new** File(filePath); // Create a File object with the provided file path  **if** (file.exists()) { // Check if the file exists  **try** {  Scanner fileScanner = **new** Scanner(file); // Create a Scanner object to read from the file  // Read and display the contents of the file line by line  **while** (fileScanner.hasNextLine()) {  String line = fileScanner.nextLine();  System.***out***.println(line);  }  fileScanner.close(); // Close the file Scanner object  } **catch** (Exception e) {  // If any exception occurs during file reading, display an error message  System.***out***.println("Error: Unable to read the file. Please try again.");  } **finally** {  // Close the Scanner object to prevent resource leak  scanner.close();  }  } **else** {  // If the file does not exist, display an error message  System.***out***.println("Error: File not found. Please make sure the file exists.");  }  }  } |
| --- |

**Output:-**

Enter the file path: "C:\Users\Admin\Desktop\TestcaseTemplate"

Error: File not found. Please make sure the file exists.

Enter the file path: C:\Users\Admin\Desktop\UsrAuth.csv

Test Case ID,Module,Test Case Description,Prerequisites,Steps to Execute,Expected Results

TC1,Test Case 1: User Registration,Validate Unique Email Addresses,None,1. Attempt to register a new user with an email address that is already registered. <br> 2. Submit registration form with a unique email address.,"The system should display an error message indicating that the email address is already in use for the first step. For the second step, the user should be successfully registered."