SWE 4202: OBJECT ORIENTED CONCEPTS I LAB

LAB_08: Exception Handling

PREPARED BY :

FARZANA TABASSUM || LECTURER
farzana@iut-dhaka.edu

ZANNATUN NAIM SRISTY || LECTURER
zannatunnaim@iut-dhaka.edu

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
ISLAMIC UNIVERSITY OF TECHNOLOGY

APRIL 1, 2024

1 Exceptions in Java

In Java, Exception is an unwanted or unexpected event, which occurs during the execution of a program, i.e. at run time, that disrupts the normal flow of the program's instructions. Exceptions can be caught and handled by the program. When an exception occurs within a method, it creates an object. This object is called the exception object. It contains information about the exception, such as the name and description of the exception and the state of the program when the exception occurred. Some major reasons why an exception may occur

- Invalid user input
- Device failure
- Loss of network connection
- Physical limitations (out-of-disk memory)
- Code errors
- Opening an unavailable file

1.1 Exception Hierarchy

All exception and error types are subclasses of the class Throwable, which is the base class of the hierarchy. One branch is headed by Exception.

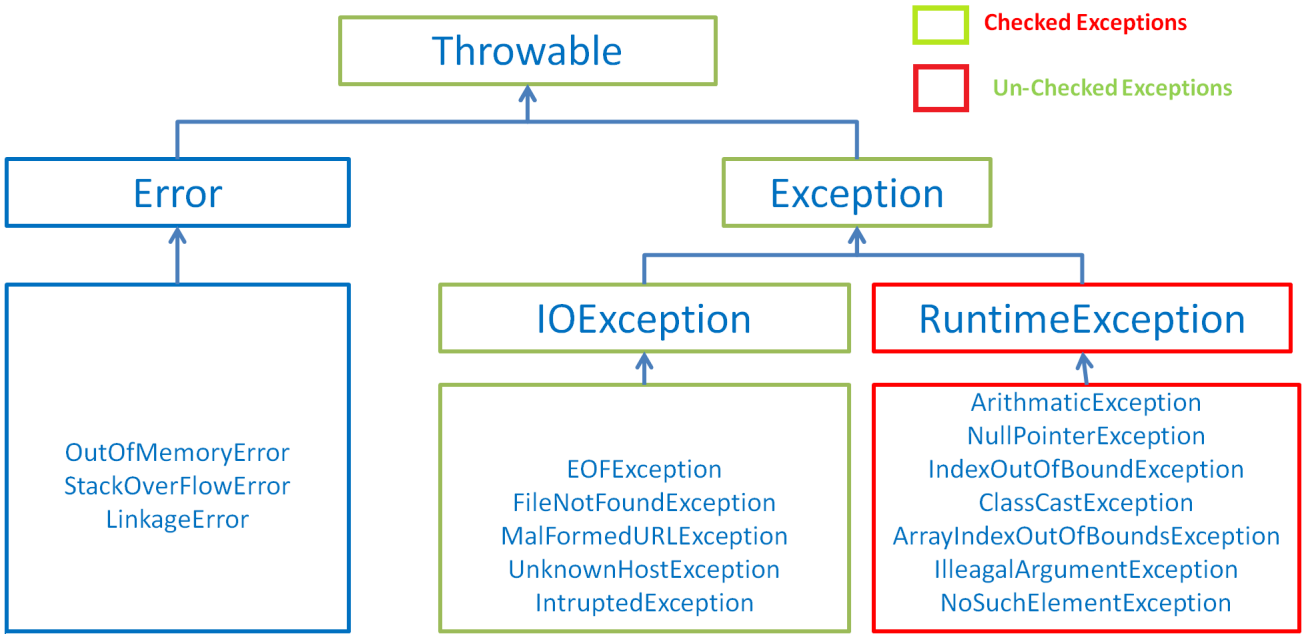

Figure 1: Exceptions

Checked Exceptions: Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler. Example: ClassNotFoundException, SQLException, IOException etc.
Unchecked Exceptions: The compiler will not check these exceptions at compile time. In simple words, if a program throws an unchecked exception, and even if we didn't handle or declare it, the program would not give a compilation error.
Error: Error is irrecoverable. Some example of errors are OutOfMemoryError, StackOverFlowError, AssertionError etc.

1.2 Java Exception Keywords

Java provides five keywords that are used to handle the exception. The following Table1 describes each.

Keyword	Description
try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

Table 1: Java Exception Keywords

1.3 Java Exception Handling Example

```
1  class Main {
2    public static void main(String[] args) {
3
4      try {
5        System.out.println("try block is being executed");
6        System.out.println();
7        int divideByZero = 5 / 0;
8        System.out.println("Rest of code in try block");
9      }
10
11     catch (ArithmeticException e) {
12       System.out.println("ArithmeticException => " + e.getMessage());
13     }
14
15      finally {
16       System.out.println();
17       System.out.println("Finally block is always executed");
18     }
19   }
20 }
```

Output:

```
try block is being executed

ArithmeticException => / by zero

Finally block is always executed
```

In the above example, notice line **number 7**. Here, we are trying to divide a number by zero. In this case, an exception occurs. Hence, we have enclosed this code inside the *try* block.
When the program encounters this code, ArithmeticException occurs. And, the exception is caught by the *catch* block and executes the code inside the *catch* block. The *catch* block is only executed if there exists an exception inside the try block.
And lastly, the *finally* block is always executed whether there is an exception inside the *try* block or not.

1.4 Java Multi-catch block

For each *try* block, there can be **zero or more** *catch* blocks. Multiple *catch* blocks allow us to handle each exception differently. The syntax for this:

```
try{
  // code
}
catch(exception) {
  // code
}
catch(exception) {
  // code
}
```

Points to remember...

- At a time only one exception occurs and at a time only one catch block is executed.
- All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.

1.5 Java Custom Exception

In Java, according to user need you can also create your own exceptions that are derived classes of the Exception class. Creating your own Exception is known as custom exception or user-defined exception.
Example:

```
class InvalidAgeException  extends Exception
{
    public InvalidAgeException (String str)
    {
        super(str);
    }
}

// class that uses custom exception InvalidAgeException
class VoteEligibility
{
    public void validate (int age) throws InvalidAgeException{
       if(age < 18){

        // throw an object of user defined exception
        throw new InvalidAgeException("age is not valid to vote");
    }
       else {
        System.out.println("welcome to vote");
        }
     }
}

public class Main
{
    public static void main(String args[])
    {
        VoteEligibility ve = new VoteEligibility();
        try
        {
            // calling the method
            ve.validate(13);
        }
```

```
            catch (InvalidAgeException ex)
            {
                System.out.println("Caught the exception");

                // printing the message from InvalidAgeException object
                System.out.println("Exception occured: " + ex);
            }

            System.out.println("rest of the program...");
        }
    }
```

Output:

```
Caught the exception
Exception occured: InvalidAgeException: age is not valid to vote
rest of the program...
```

2 Problem Statement 01

Your task involves implementing functionalities for a basic banking system. You are provided with a 'BankAccount' class representing individual bank accounts and a 'BankingSystem' class containing the 'main' method to demonstrate the operations.
The 'BankAccount' class includes an account number and initial amount and two methods 'withdraw' and 'calculateAmountWithInterest'. The 'calculateAmountWithInterest' method takes principal, rate and time and return the total amount. The equation is provided for you-
$principal * (int)Math.pow(1 + rate/rate, time)$
There will be a custom exception 'InsufficientBalanceException'. If he withdrawal amount is greater than the available balance, then this exception will be thrown. In the BankingSystem class, incorporate the following code snippets to demonstrates the usage of the exception handling.

```
// You can incorporate these code snippets however you want in the main
   function.
    BankAccount account = new BankAccount("123", 1000);
    int interest = account.calculateInterest(principal, rate, time);

    BankAccount account1 = new BankAccount("123", 1000);
    account1.withdraw(1500);

    BankAccount[] accounts = new BankAccount[3];
    accounts[3] = new BankAccount("123", 1000);
```

Your completed code should accurately handle all potential exceptions gracefully to ensure the reliability of the banking system. Figure1 may help you to identify possible exceptions.