Debugging & Exception Handling

Exception

Exception is an occurrence of an undesirable situation that can be detected during program execution.

- Division by zero and inputting invalid data are exceptions.
- Similarly, trying to open an input file that does not exist is an exception.
- In an array index that is outside the bounds of the array.

class Exception (directly or indirectly) is the superclass of all the exception classes in Java. When an exception occurs, an object of a particular exception class is created. There are two types of exceptions.

1. **Checked Exceptions**
    - Compiler check these types of exceptions
    - Compile Time Exceptions
    - FileNotFoundExceptions are checked exceptions
2. **Unchecked Exceptions**
    - Compiler does not check these types of exceptions
    - Runtime Exceptions
    - InputMismatchExceptions are unchecked exceptions

Exception Class	Description
`ArithmeticException`	Arithmetic errors such as division by zero
`ArrayIndexOutOfBoundsException`	Array index is either less than 0 or greater than or equal to the length of the array.
`FileNotFoundException`	Reference to a file that cannot be found
`IllegalArgumentException`	Calling a method with illegal arguments
`IndexOutOfBoundsException`	An array or a string index is out of bounds.
`NullPointerException`	Reference to an object that has not been instantiated
`NumberFormatException`	Use of an illegal number format
`StringIndexOutOfBoundsException`	A string index is either less than 0 or greater than or equal to the length of the string.
`InputMismatchException`	Input (token) retrieved does not match the pattern for the expected type, or the token is out of range for the expected type.

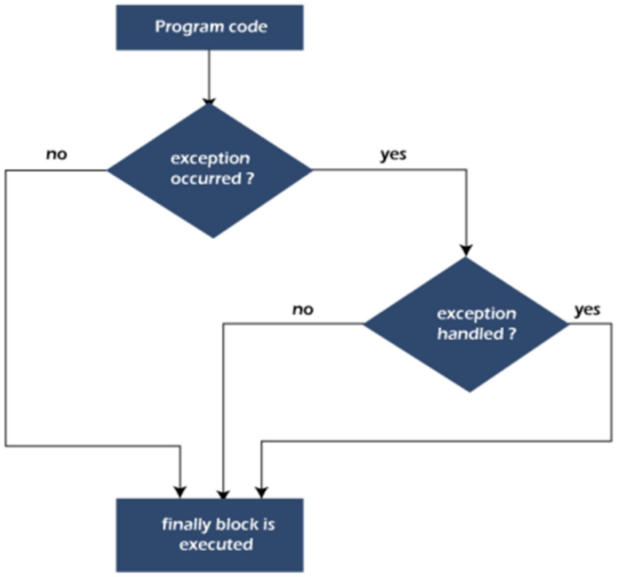
Error

Error is an illegal operation which results in the abnormal working of the program. There are three types of errors.

1. Syntax errors
    - The language rules are not followed
    - Missing semicolon, not declaring an object and using/calling it.
2. Logical errors
    - The program is syntactically correct but does not work as expected.
    - Writing program for addition when it is expected to return modulus.
3. Runtime errors
    - Occurs at runtime.
    - **Also called unchecked exceptions**
    - Input value of variable is other than the declared one.

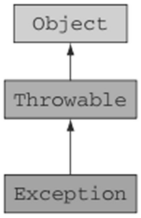
try/catch/finally Block

- Statements that might generate an exception are placed in a try block.
- The try block might also contain statements that should not be executed if an exception occurs.
- The try block is followed by zero or more catch blocks.
- A catch block specifies the type of exception it can catch and contains an exception handler.
- The last catch block may or may not be followed by a finally block.
- Any code contained in a finally block always executes, regardless of whether an exception occurs, except when the program exits early from a try block by calling the method System.exit.
- If a try block has no catch block, then it must have the finally block.



Java Exception Hierarchy

The class Throwable, which is derived from the class Object, is the superclass of theclass Exception



Exception Handling Techniques

1. Terminate the Program
2. Fix the error and continue
3. Log the error and continue.

Review the example

```
import java.util.*;

public class ExceptionExample1
{
    static Scanner console = new Scanner(System.in);

    public static void main(String[] args)
    {
        int dividend, divisor, quotient;

        System.out.print("Line 2: Enter the "
                       + "dividend: ");
        dividend = console.nextInt();
        System.out.println();
        System.out.print("Line 5: Enter the "
                       + "divisor: ");
        divisor = console.nextInt();
        System.out.println();

        quotient = dividend / divisor;

        System.out.println("Line 9: Quotient = "
                         + quotient);
    }
}
```

Sample Run 1:

```
Line 2: Enter the dividend: 12

Line 5: Enter the divisor: 5

Line 9: Quotient = 2
```

Sample Run 2:

```
Line 2: Enter the dividend: 24

Line 5: Enter the divisor: 0

Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ExceptionExample1.main(ExceptionExample1.java:22)
```

Sample Run 3:

```
Line 2: Enter the dividend: 2e
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Unknown Source)
        at java.util.Scanner.next(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at ExceptionExample1.main(ExceptionExample1.java:14)
```

The problem above can be handled with “if – else” block but it is not the most effective and efficient method. Exceptions are effectively handled with “Try-catch” block, throw, and throws statements.

```
import java.util.*;

public class ExceptionExample3
{
    static Scanner console = new Scanner(System.in);

    public static void main(String[] args)
    {
        int dividend, divisor, quotient;
         try
         {
             System.out.print("Line 4: Enter the "
                              + "dividend: ");
             dividend = console.nextInt();
             System.out.println();

             System.out.print("Line 7: Enter the "
                              + "divisor: ");
             divisor = console.nextInt();
             System.out.println();

             quotient = dividend / divisor;
             System.out.println("Line 11: Quotient = "
                                + quotient);
         }
         catch (ArithmeticException aeRef)
         {
             System.out.println("Line 13: Exception "
                                + aeRef.toString());
         }
         catch (InputMismatchException imeRef)
         {
             System.out.println("Line 15: Exception "
                                 + imeRef.toString());
         }
     }
 }
```

Review the following example

```
import java.io.*;

public class PrintStackTraceExample1
{
    public static void main(String[] args)
    {
        try
        {
            methodA();
        }
        catch (Exception e)
        {
            System.out.println(e.toString() + " caught in main");
            e.printStackTrace();
        }
    }

    public static void methodA() throws Exception
    {
        methodB();
    }

    public static void methodB() throws Exception
    {
        methodC();
    }

    public static void methodC() throws Exception
    {
        throw new Exception("Exception generated in method C");
    }
}
```

TASK

Write a program that prompts the user to enter the length in feet and inches and outputs the equivalent length in inches and in centimeters. If the user enters a negative number or a non-digit number, throw and handle an appropriate exception and prompt the user to enter another set of numbers. Use the process of debugging.

Consider the following Java code:

```
int lowerLimit;
int divisor;
int result;

try
{
    System.out.println("Entering the try block.");

    result = lowerLimit / divisor;

    if (lowerLimit < 100)
        throw new Exception("Lower limit violation.");

    System.out.println("Exiting the try block.");
}
catch (ArithmeticException e)
{
    System.out.println("Exception: " + e.getMessage());

    result = 110;
}
catch (Exception e)
{
    System.out.println("Exception: " + e.getMessage());
}

System.out.println("After the catch block");
```

What is the output if:

a. The value of `lowerLimit` is `50` and the value of `divisor` is `10`?
b. The value of `lowerLimit` is `50` and the value of `divisor` is `0`?
c. The value of `lowerLimit` is `150` and the value of `divisor` is `10`?
d. The value of `lowerLimit` is `150` and the value of `divisor` is `0`?

Correct any compile-time errors in the following code:

```
import java.io.*;
import java.util.*;

public class SAverage
{
    public static void main(String[] args)
    {
        double test1, test2, test3, test4;
        double average;
```

```
        try
        {
            Scanner inFile = new
                 Scaner(new FileReader("test.txt"));

            PrintWriter outFile =
                        new PrintWriter("testavg.out");

            test1 = inFile.nextDouble();
            test2 = inFile.nextDouble();
            test3 = inFile.nextDouble();
            test4 = inFile.nextDouble();

            outFile.printf("Test scores: %.2f %.2f %.2f %.2f %n",
                           test1, test2, test3, test4);

            average = (test1 + test2 + test3 + test4) / 4.0;
            outFile.println("Average test score: %.2f",
                            average);

            outFile.close();

        }
        catch (Exception e)
        {
            System.out.println(e.toString());
        }
        catch (FileNotFoundException e)
        {
            System.out.println(e.toString());
        }
    }
}
```