Faculty : Mr. Tarek Mizan
Lab Instructor: Nazmul Alam Diptu
Email : nazmul.diptu@northsouth.edu
Class Timing: ST 1:00 PM – 2:30 PM (LIB-611)
Topic: Exceptions, Unit Test

Objective

1. Java Exception
2. Types of Exceptions
3. try-catch
4. Introduction to JUnit : TDD (Test Driven Development)

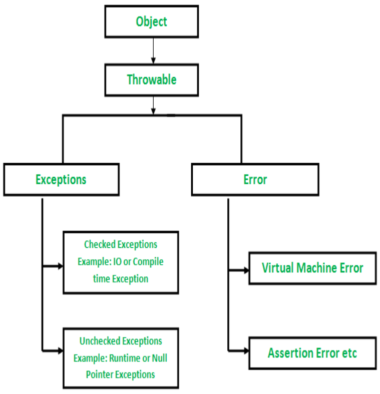
Exception: An exception is an unexpected event that occurs during program execution. An exception can occur for many reasons. Some of them are:

1. Invalid user input
2. Device failure
3. Loss of network connection
4. Physical limitations (out of disk memory)
5. Code errors
6. Opening an unavailable file

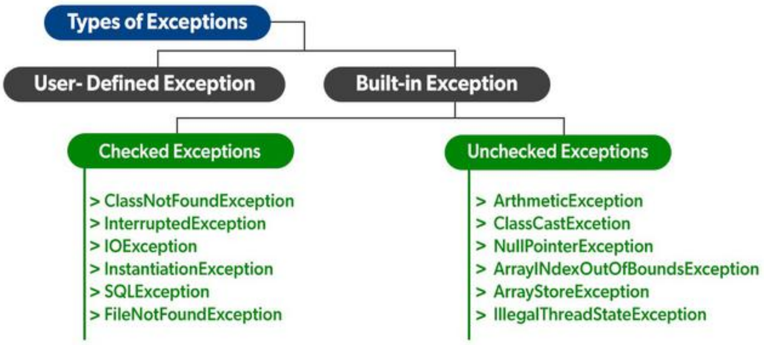
Errors represent irrecoverable conditions such as Java virtual machine (JVM) running out of memory, memory leaks, stack overflow errors, library incompatibility, infinite recursion, etc. Errors are usually beyond the control of the programmer and we should not try to handle errors.

Error: An Error indicates a serious problem that a reasonable application should not try to catch.
Exception: Exception indicates conditions that a reasonable application might try to catch.

Exception Hierarchy:


Object
Throwable
Exceptions
Error
Checked Exceptions Example: IO or Compile time Exception
Unchecked Exceptions Example: Runtime or Null Pointer Exceptions
Virtual Machine Error
Assertion Error etc

Types of Exceptions:



Exceptions can be Categorized in two ways:

1. Built-in Exception

- Checked Exception : Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler.
- Unchecked Exception : The unchecked exceptions are just opposite to the checked 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.

2. User-Defined Exceptions : Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, users can also create exceptions which are called 'user-defined Exceptions'.

The advantages of Exception Handling in Java are as follows:

1. Provision to Complete Program Execution
2. Easy Identification of Program Code and Error-Handling Code
3. Propagation of Errors
4. Meaningful Error Reporting
5. Identifying Error Types

Exception01.java

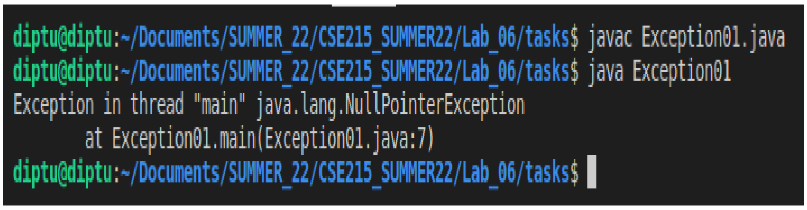
```java

public class Exception01 { // Save as "Exception01.java"

    public static void main(String args[]) {
        // declare a String variable and initialize it to null
        String str = null;
        // print the string
        System.out.println(str.length());

    }

}
```



```
diptu@diptu:~/Documents/SUMMER_22/CSE215_SUMMER22/Lab_06/tasks$ javac Exception01.java
diptu@diptu:~/Documents/SUMMER_22/CSE215_SUMMER22/Lab_06/tasks$ java Exception01
Exception in thread "main" java.lang.NullPointerException
        at Exception01.main(Exception01.java:7)
diptu@diptu:~/Documents/SUMMER_22/CSE215_SUMMER22/Lab_06/tasks$
```

`Exception02.java`

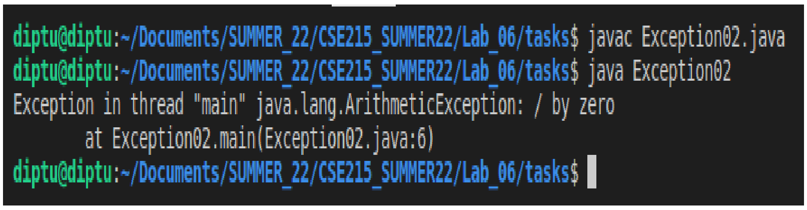
```java

public class Exception02 { // Save as "Exception02.java"

    public static void main(String args[]) {
        int num1 = 10;
        int num2 = 0;
        // divide both numbers and print the result
        int result = num1 / num2;
        System.out.println(result);

    }

}
```



```
diptu@diptu:~/Documents/SUMMER_22/CSE215_SUMMER22/Lab_06/tasks$ javac Exception02.java
diptu@diptu:~/Documents/SUMMER_22/CSE215_SUMMER22/Lab_06/tasks$ java Exception02
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Exception02.main(Exception02.java:6)
diptu@diptu:~/Documents/SUMMER_22/CSE215_SUMMER22/Lab_06/tasks$
```

```java
public class Exception02 {
    public static void main(String args[]) {
        try {
            // define two numbers
            int num1 = 100, num2 = 0;
            int result = num1 / num2; // divide by zero
            // print the result
            System.out.println("Result = " + result);
        } catch (ArithmeticException e) {
            System.out.println("ArithmeticException:Division by Zero");
        }

    }

}
```

JUnit JUnit is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks which is collectively known as xUnit that originated with SUnit. JUnit is linked as a JAR at compile-time. We will be working with latest version of JUnit called JUnit 5.

> **JUint 5 Example**
>
> Create a new Java project e,g ***Junit Demo***
>
> - Create two packages e,g ***com.your_name.arithmetic*** & ***com.your_name.geometry***
>
> 1. Under ***com.your_name.arithmetic***
>
> > Create a java class CalculationUnit without main method
>
> 2. Under ***com.your_name.geometry***
>
> > Create a java class CircleUnit without main method

`J.java`

```
public class CalculationUnit { // Save as "CalculationUnit.java"

    public int addition(int a, int b) {
        return a + b ;
    }

}
```

> Now right click on CalculationUnit.java New > JUnit Test Case
>
> 1. Choose New JUnit Jupyter test from radio button if not already choosen.
>
> > Click finish > Add Junit library to built path > Ok
>
> 2. Under ***com.your_name.arithmetic***
>
> > CalculationUnitTest.java will be created

`CalculationUnitTest.java`

```
class CalculationUnitTest {

    @Test
    void testAddition() {
        CalculationUnit c = new CalculationUnit();

        int a = 7;
        int b = 3;
        int expected = 10;
        int actual = c.addition(a, b);
        String msg =  a + " + " + b + " = " + (a+b) ;
        assertEquals(expected, actual, msg);
    }
}
```

> To execute CalculationUnit.java as JUnit test: CalculationUnit.java > Run as > JUnit Test

`CircleUnit.java`

```
public class CircleUnit {

    public double diameter(double radious) {
        return 2 * radious;
    }
}
```

`CircleUnitTest.java`

```
class CircleUnitTest {

    CircleUnit c;

    @BeforeEach
    void init() {
        c = new CircleUnit();
    }

    @Test
    void testCircleDiameter(){

        double expected = 2 * radious;
        double actual = c.diameter(radious);
        // Lazy assert -> message created only if test fails.
        assertEquals(expected, actual,()-> "Diameter of a circle with
radious " + radious + " should be " + expected);
    }

}
```

> To execute All JUnit test: Java Project (Java Demo) > Run as > JUnit Test

We will continue on this next class

Faculty : Mr. Tarek Mizan
Lab Instructor: Nazmul Alam Diptu
Email : nazmul.diptu@northsouth.edu
Class Timing: ST 1:00 PM – 2:30 PM (LIB-611)
Quiz : 01, SET A, Time : 40 min, Points : 10

Tasks: Each problems carry same weights.

1. Write down a function power(x,y) that will take two integers x and y as parameter and will return x^y. Note that y can be negative. In your main function take two integers as input and use this function to determine and print the power. *Built in pow() is not allowed here.*

```
Sample Input/Output             Sample Input/Output
Enter x: 100                    Enter x: 5
Enter y: 500                    Enter y: -2
5^2 = 25                        5^-2 = 0.04
```

2. Write down a function that will take an integer as parameter and will return 1 if the integer is an Armstrong number of three digits and return 0 otherwise. Using this function write down a program that will print all Armstorng numbers between two ranges n1 and n2 that will be input to your program. Assume that n1 < n2 and both are of 3-digits. An Armstrong number of three digits is an integer such that the sum of the cubes of its digits is equal to the number itself. For example, 371 is an Armstrong number since 3^3 + 7^3 + 1^3 = 371.

```
Sample Input                Output:
Enter n1: 100               153 370 371 407
Enter n2: 500
```

Faculty : Mr. Tarek Mizan
Lab Instructor: Nazmul Alam Diptu
Email : nazmul.diptu@northsouth.edu
Class Timing: ST 1:00 PM – 2:30 PM (LIB-611)
Quiz : 01, SET B, Time : 40 min, Points : 10

Tasks: Each problems carry same weights.

1. Write down a function what will take an integer n as parameter and will return the summation of the following series:

   ```
   1!/1 +2!/2 +3!/3 +4!/4 ... ... ... + n!/n
   ```

```
Sample Input/Output
Enter n: 5
Result is: 34
```

2. Write down a function that will take an integer as parameter and will return 1 if the integer is a perfect number and return 0 otherwise. Using this function write down a program that will print all perfect numbers between two ranges n1 and n2 that will be input to your program. Assume that n1 < n2. Recall that a perfect number is a positive integer that is equal to the sum of its proper positive divisors, that is, the sum of its positive divisors excluding the number itself. For example, 6 is a perfect number because its positive divisors are 1, 2, 3 and the summation of these positive numbers are 1+2+3 = 6, which is the number itself.

```
Enter n1: 5
Enter n2: 20000
6 28 496 8128
```