LAB # 12

Exception Handling

OBJECTIVE: Constructing a fault tolerant program by implementing exception handling techniques.

Exception:

An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e. at run time, that interrupts the normal flow of the program's instructions. When an exception occurs, program execution gets terminated. In such cases we get a system generated error message.

Exception Handling in Java:

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime. Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Advantage of exception handling:

Exception handling ensures that the flow of the program does not break when an exception occurs. For example, if a program has bunch of statements and an exception occurs mid-way after executing certain statements then the statements after the exception will not execute and the program will terminate abruptly. By handling we make sure that all the statements execute, and the flow of program does not break.

Java Exception Keywords:

There are 5 keywords which are used in handling exceptions in Java.

Keyword	Description
Try	The try block contains set of statements where an exception can occur. The try block must be followed by either catch or finally. It means, we can't use try block alone.
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 important code of the program. It is executed whether an exception is handled or not.
throw	The throw keyword is used to throw an exception. We can throw either checked or unchecked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception.

throws	The throws keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature. The Java throws keyword gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Try-catch block Example:

```
public class Example1 {
      public static void main(String args[]) {
            int num1, num2;
            try {
               /* We suspect that this block of statement can throw
                * exception so we handled it by placing these statements
                * inside try and handled the exception in catch block
                */
               num1 = 0;
               num2 = 62 / num1;
               System.out.println(num2);
               System.out.println("Hey I'm at the end of try block");
            }
            catch (ArithmeticException e) {
               /* This block will only execute if any Arithmetic exception
                * occurs in try block
                */
               System.out.println("You should not divide a number by zero");
            }
            catch (Exception e) {
       /* This is a generic Exception handler which means it can handle
            * all the exceptions. This will execute if the exception is not
                * handled by previous catch blocks.
                */
              //This catch block catches all the exceptions

               System.out.println("Exception occurred");
                  }
            System.out.println("I'm out of try-catch block in Java.");
         }

}
```

Output:

```
You should not divide a number by zero
I'm out of try-catch block in Java.
```

```
public class Use {

    public static void main(String[] args) {

        int i=50; int j=0; int data;

        try

        {  data=i/j; //may throw exception     }

       // handling the exception

        catch(Exception e)

        {   // resolving the exception in catch block

            System.out.println(i/(j+2));  }

       } }
```

Finally block example:

```
public class FinallyBlock {
      public static void main(String args[]) {
            try{
             int num=121/0;
             System.out.println(num);
            }
            catch(ArithmeticException e){
               System.out.println("Number should not be divided by zero");
            }
            /* Finally block will always execute
             * even if there is no exception in try block
             */
            finally{
             System.out.println("This is finally block");
            }
            System.out.println("Out of try-catch-finally");
         }

}
```

Output:

```
Number should not be divided by zero
This is finally block
Out of try-catch-finally
```

Throw Keyword example:

```
public class throwExample {
      static void checkEligibilty(int age, int weight){
            if(age<12 && weight<40) {
               throw new ArithmeticException(" Not Eligible for
registration");
            }
            else {
               System.out.println("Entry is Valid!!");
            }
         }

         public static void main(String args[]){
           System.out.println("Welcome to the Registration process!!");
           checkEligibilty(10, 39);

      }
}
```

Output:

```
Welcome to the Registration process!!
Exception in thread "main" java.lang.ArithmeticException: Not Eligible for registration
        at throwExample.checkEligibilty(throwExample.java:5)
        at throwExample.main(throwExample.java:14)
```

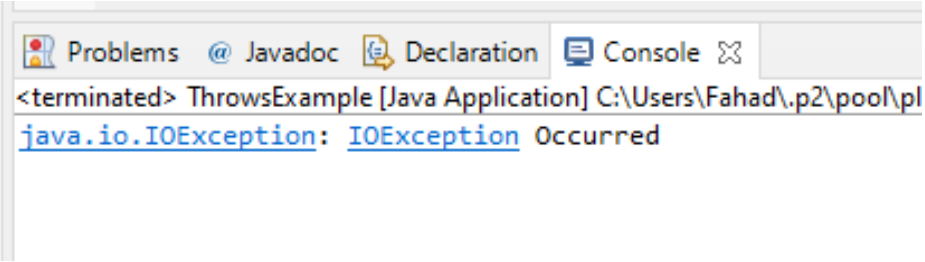
Throws Keyword example:

```
import java.io.IOException;

public class ThrowsExample {
      void myMethod(int num)throws IOException, ClassNotFoundException{
           if(num==1)
              throw new IOException("IOException Occurred");
           else
              throw new ClassNotFoundException("ClassNotFoundException");
        }

        public static void main(String args[]){
         try {
           ThrowsExample obj=new ThrowsExample();
            obj.myMethod(1);
                  }
         catch(Exception ex)
         {
           System.out.println(ex);
          }
        }
}
```

Output:



Throw vs Throws in java

Throws clause is used to declare an exception, which means it works similar to the try-catch block. On the other hand **throw** keyword is used to throw an exception explicitly.

Throw:

```
...
void myMethod() {
   try {
      //throwing arithmetic exception using throw
      throw new ArithmeticException("Something went wrong!!");
   }
   catch (Exception exp) {
      System.out.println("Error: "+exp.getMessage());
   }
```

```
}
...
```

Throws:

```
...
//Declaring arithmetic exception using throws
void sample() throws ArithmeticException{
   //Statements
}
...
```

You can **throw** one exception at a time, but you can handle multiple exceptions by declaring them using **throws** keyword.

Throw:

```
void myMethod() {
   //Throwing single exception using throw
   throw new ArithmeticException("An integer should not be divided by zero!!");
}
..
```

Throws:

```
//Declaring multiple exceptions using throws
void myMethod() throws ArithmeticException, NullPointerException{
   //Statements where exception might occur
}
```

Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked.

Checked Exception: The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

Unchecked Exception: The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime. Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Built-in Exceptions

Built-in exceptions are the exceptions which are available in Java libraries. These exceptions are suitable to explain certain error situations. Below is the list of important built-in exceptions in Java.

Arithmetic Exception:

```
public class ArithmeticE {
    public static void main(String args[])
    {
        try {
            int a = 30, b = 0;
            int c = a/b;  // cannot divide by zero
            System.out.println ("Result = " + c);
        }
        catch(ArithmeticException e) {
            System.out.println ("Can't divide a number by 0");
        }
    }
}
```

Output: Can't divide a number by 0

NullPointer Exception

```
public class NullpointerE {
      public static void main(String args[])
    {
        try {
            String a = null; //null value
            System.out.println(a.charAt(0));
        } catch(NullPointerException e) {
            System.out.println("NullPointerException..");
        }
    }
}
```

Output: NullPointerException..

StringIndexOutOfBound Exception

```
public class StringIndexOutOfBound_Demo {
    public static void main(String args[])
    {
        try {
            String a = "This is like chipping "; // length is 22
            char c = a.charAt(24); // accessing 25th element
            System.out.println(c);
        }
        catch(StringIndexOutOfBoundsException e) {
            System.out.println("StringIndexOutOfBoundsException");
        }
    }
}
```

Output: StringIndexOutOfBoundsException

ArrayIndexOutOfBounds Exception

```
public class ArrayIndexOutOfBound_Demo {
      public static void main(String args[])
    {
        try{
            int a[] = new int[5];
            a[6] = 9; // accessing 7th element in an array of
                      // size 5
        }
        catch(ArrayIndexOutOfBoundsException e){
            System.out.println ("Array Index is Out Of Bounds");
        }
    }
}
```

Output: Array Index is Out Of Bounds

Lab Task:

Question#01

Write a program that meets the following requirements:

- Creates an array with 10 randomly chosen integers.
- Prompts the user to enter the index of the array, then displays the corresponding element value. If the specified index is out of bounds, display the message Out of Bounds. (ArrayIndexOutOfBoundsException)

Question#02

Suppose you are developing a game for kids in which they are learning the division operation in math. Your game will take input from kids (2 integers) and then perform the division and displays the answer. Think and apply exception handling in this scenario. For e.g. Arithmetic Exception might occur here. Also suppose, this game has the limitation that it only performs division between integers so if it gets a decimal number as input, it **throws** an exception stating that the input is invalid, please give integer number etc.