Experiment No. 9
Implement a program on Exception handling.
Date of Performance:
Date of Submission:

Aim: Implement a program on Exception handling.

Objective: To able handle exceptions occurred and handle them using appropriate keyword

Theory:

The Exception Handling in Java is one of the powerful mechanisms to handle the runtime errors so that the normal flow of the application can be maintained.

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

Java Exception Keywords

Java provides five keywords that are used to handle the exception. The following table 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.

```
public class JavaExceptionExample{

  public static void main(String args[]){

   try{

      //code that may raise exception

      int data=100/0;
      }catch(ArithmeticException e){System.out.println(e);}

       //rest code of the program

       System.out.println("rest of the code...");

      }

}
```

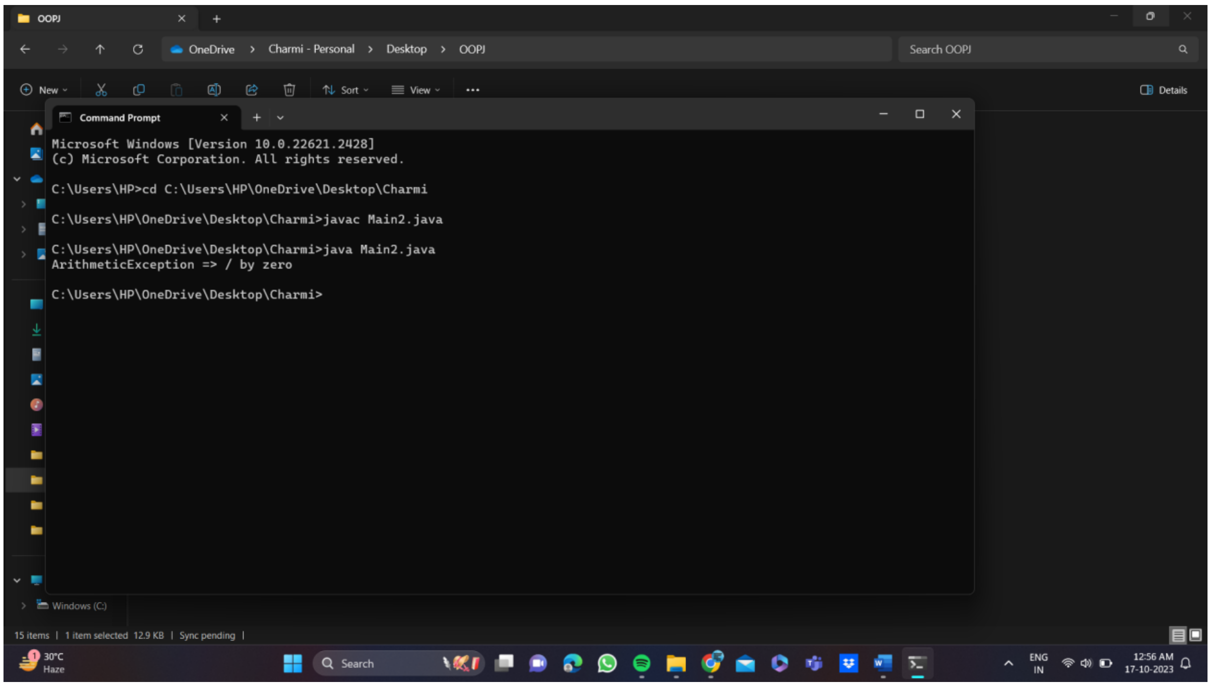
Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
rest of the code...
```

Code:

1} Try-catch

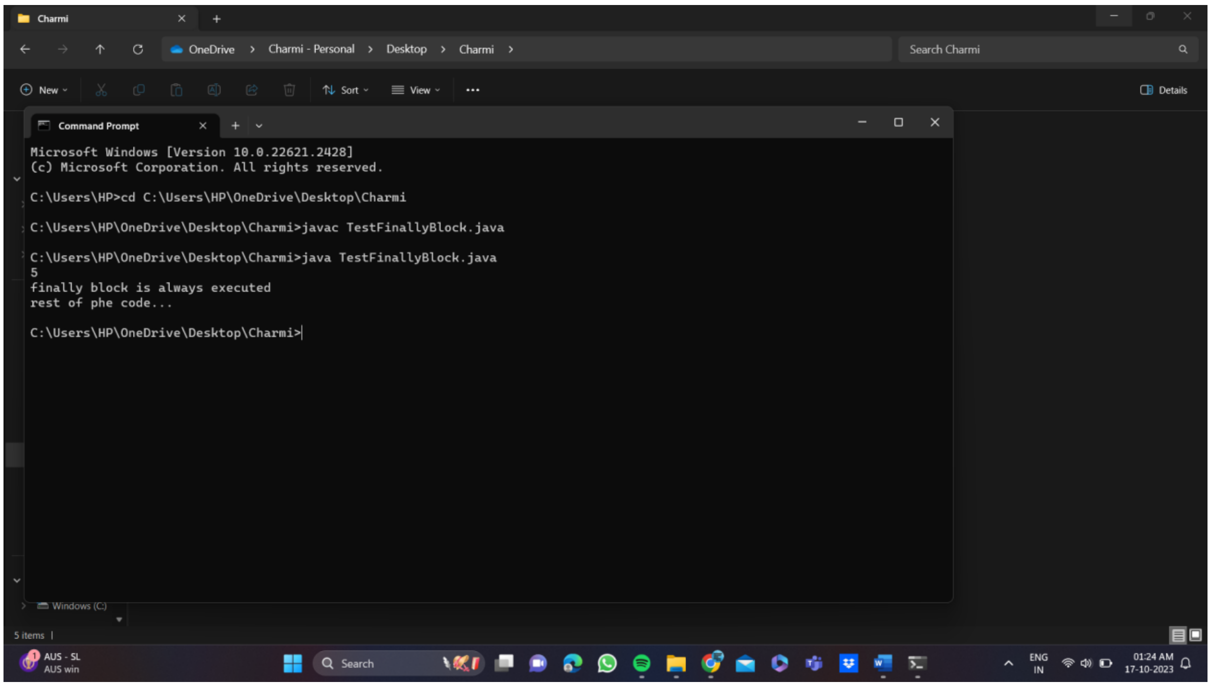
```
class Main2
{
public static void main(String args[])
{
try{
  int divideByZero = 8/0;
  System.out.println("Rest of code in try block");
    }

    catch (ArithmeticException e) {
      System.out.println("ArithmeticException => " + e.getMessage());
    }
  }
}
```



```
Microsoft Windows [Version 10.0.22621.2428]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>cd C:\Users\HP\OneDrive\Desktop\Charmi

C:\Users\HP\OneDrive\Desktop\Charmi>javac Main2.java

C:\Users\HP\OneDrive\Desktop\Charmi>java Main2.java
ArithmeticException => / by zero

C:\Users\HP\OneDrive\Desktop\Charmi>
```

2} finally

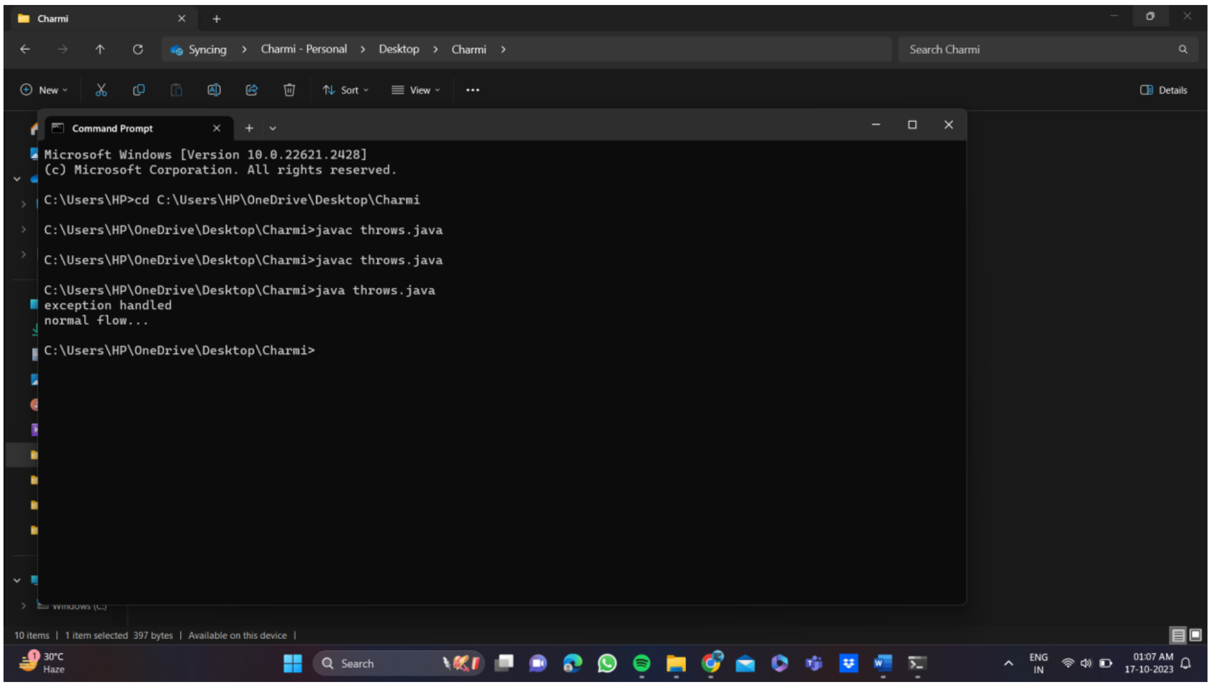
```
class TestFinallyBlock {
  public static void main(String args[]){
  try{
   int data=25/5;
   System.out.println(data);
  }
  catch(NullPointerException e){
System.out.println(e);
}
 finally {
System.out.println("finally block is always executed");
}

System.out.println("rest of phe code...");
  }
}
```



```
Microsoft Windows [Version 10.0.22621.2428]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>cd C:\Users\HP\OneDrive\Desktop\Charmi

C:\Users\HP\OneDrive\Desktop\Charmi>javac TestFinallyBlock.java

C:\Users\HP\OneDrive\Desktop\Charmi>java TestFinallyBlock.java
5
finally block is always executed
rest of phe code...

C:\Users\HP\OneDrive\Desktop\Charmi>
```

3}throws

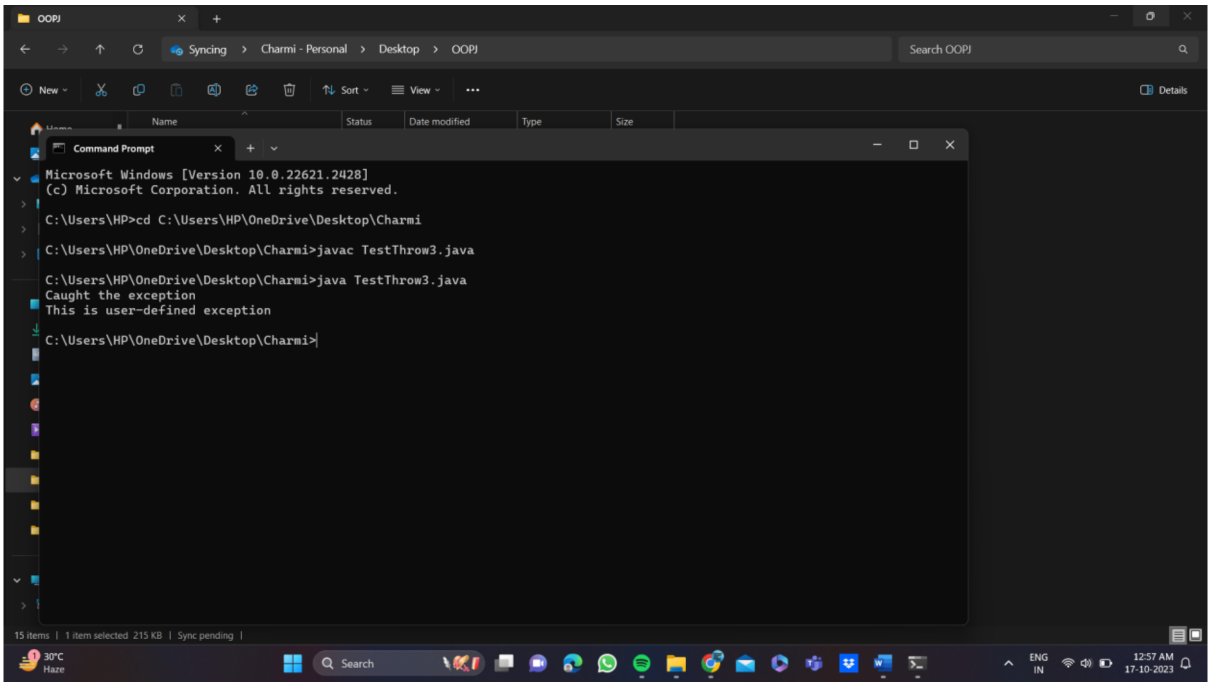
```
import java.io.IOException;
 class Testthrows2{
   public static void main(String args[]){
    try{
     M m=new M();
     m.method();
    }catch(Exception e){System.out.println("exception handled");}

    System.out.println("normal flow...");
  }
}
class M {
    void method() throws IOException {
        throw new IOException("device error");
    }
}
```



```
Microsoft Windows [Version 10.0.22621.2428]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>cd C:\Users\HP\OneDrive\Desktop\Charmi

C:\Users\HP\OneDrive\Desktop\Charmi>javac throws.java

C:\Users\HP\OneDrive\Desktop\Charmi>javac throws.java

C:\Users\HP\OneDrive\Desktop\Charmi>java throws.java
exception handled
normal flow...

C:\Users\HP\OneDrive\Desktop\Charmi>
```

4} throw

```
 class TestThrow3
{
   public static void main(String args[])
   {
      try
      {
         throw new UserDefinedException("This is user-defined exception");
      }
      catch (UserDefinedException ude)
      {
         System.out.println("Caught the exception");
         System.out.println(ude.getMessage());
      }
   }
}
class UserDefinedException extends Exception
{
   public UserDefinedException(String str)
   {
      super(str);
   }
}
```



```
Microsoft Windows [Version 10.0.22621.2428]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>cd C:\Users\HP\OneDrive\Desktop\Charmi

C:\Users\HP\OneDrive\Desktop\Charmi>javac TestThrow3.java

C:\Users\HP\OneDrive\Desktop\Charmi>java TestThrow3.java
Caught the exception
This is user-defined exception

C:\Users\HP\OneDrive\Desktop\Charmi>
```

Conclusion:

Exception handling in Java involves the utilization of essential keywords, such as try, catch, finally, and throw. This mechanism is pivotal for maintaining the robustness and reliability of your Java programs when dealing with runtime errors.

Try-Catch Blocks (try and catch): The foundation of exception handling is formed by the try-catch block. It allows you to encapsulate potentially error-prone code within a try block while providing catch blocks to handle specific types of exceptions.

Block (finally): Supplementary to try-catch blocks, the finally block comes into play. It ensures that certain code is executed, irrespective of whether an exception was thrown or not. Typically, it's employed for essential cleanup operations, like resource closure.

Throwing Exceptions (throw): The throw keyword empowers you to deliberately raise an exception within your code. This action is typically taken when your code encounters an exceptional circumstance beyond its control, and you need to transfer control to an exception handler.