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...");

    }

}
```

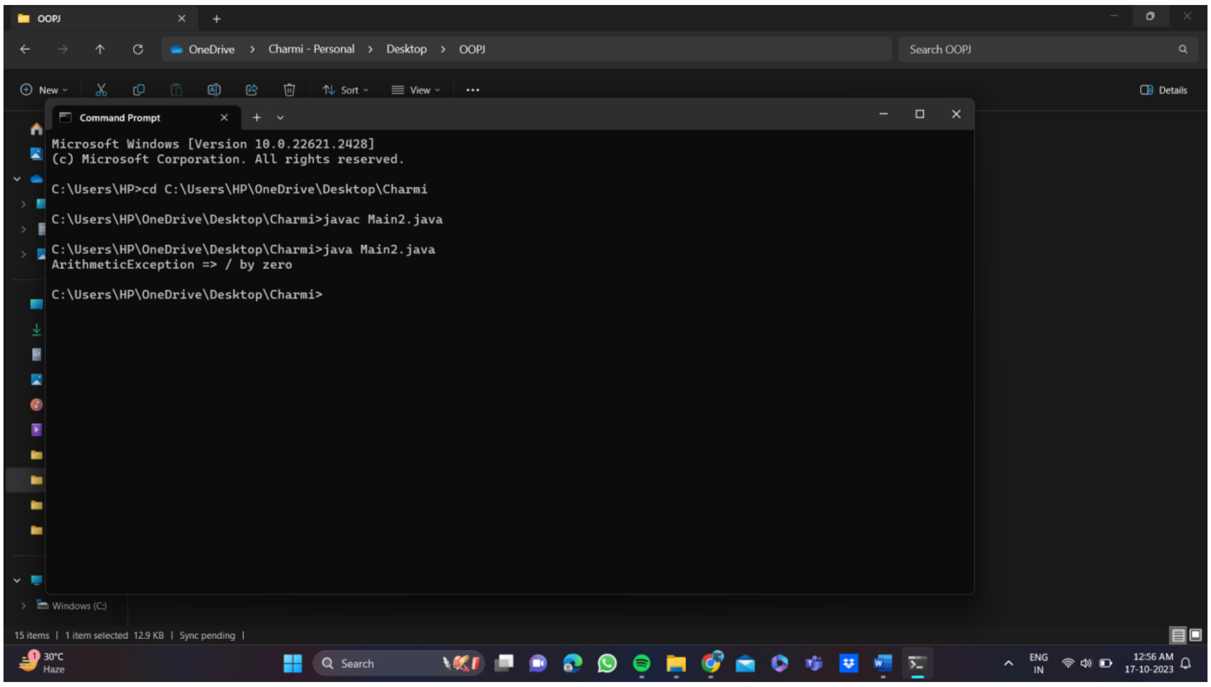
Output:

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

Code:

1} Try-catch

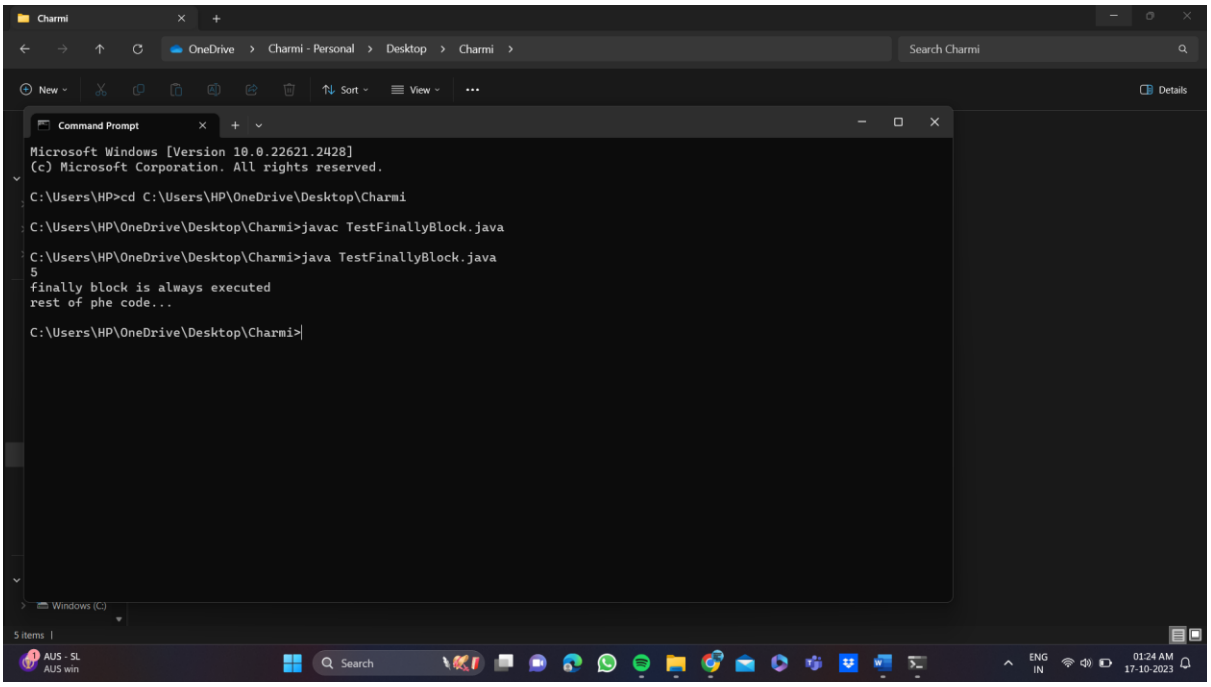
```
class Main2
{
public static void main(String args[])
{
try{
  int divideByZero = 8/0;
  System.out.println("Rest of code in try block");
    }

    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

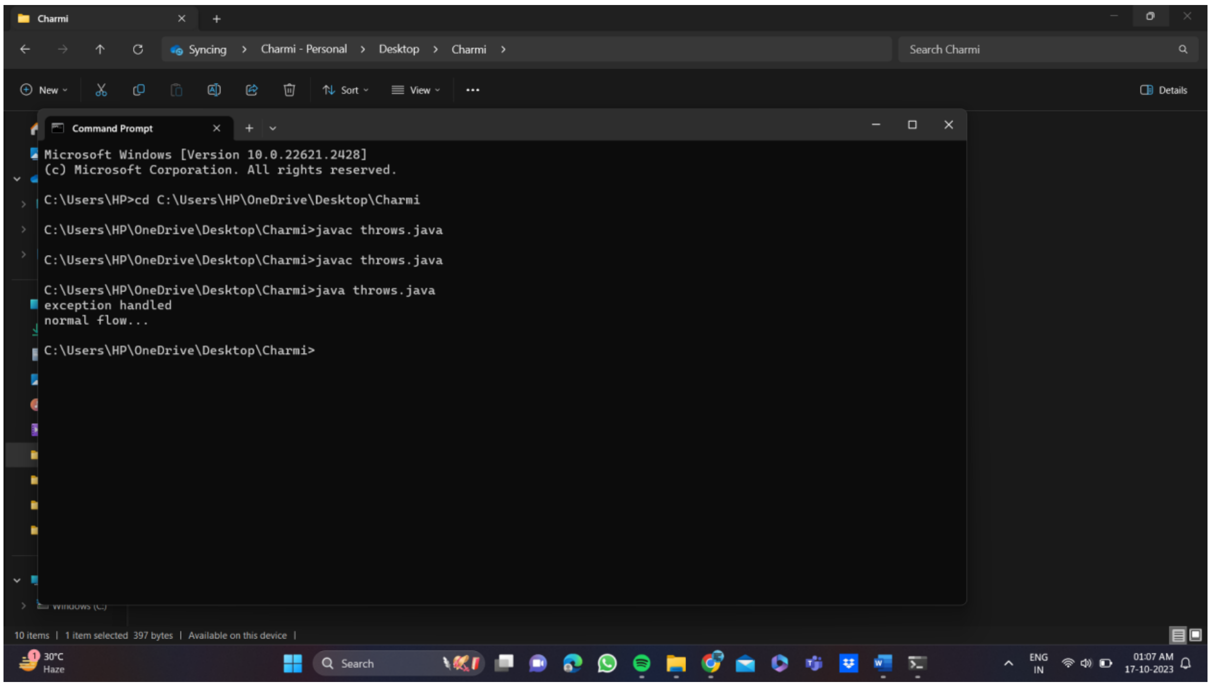
```
class TestFinallyBlock {
  public static void main(String args[]){
  try{
   int data=25/5;
   System.out.println(data);
  }
  catch(NullPointerException e){
System.out.println(e);
}
 finally {
System.out.println("finally block is always executed");
}

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

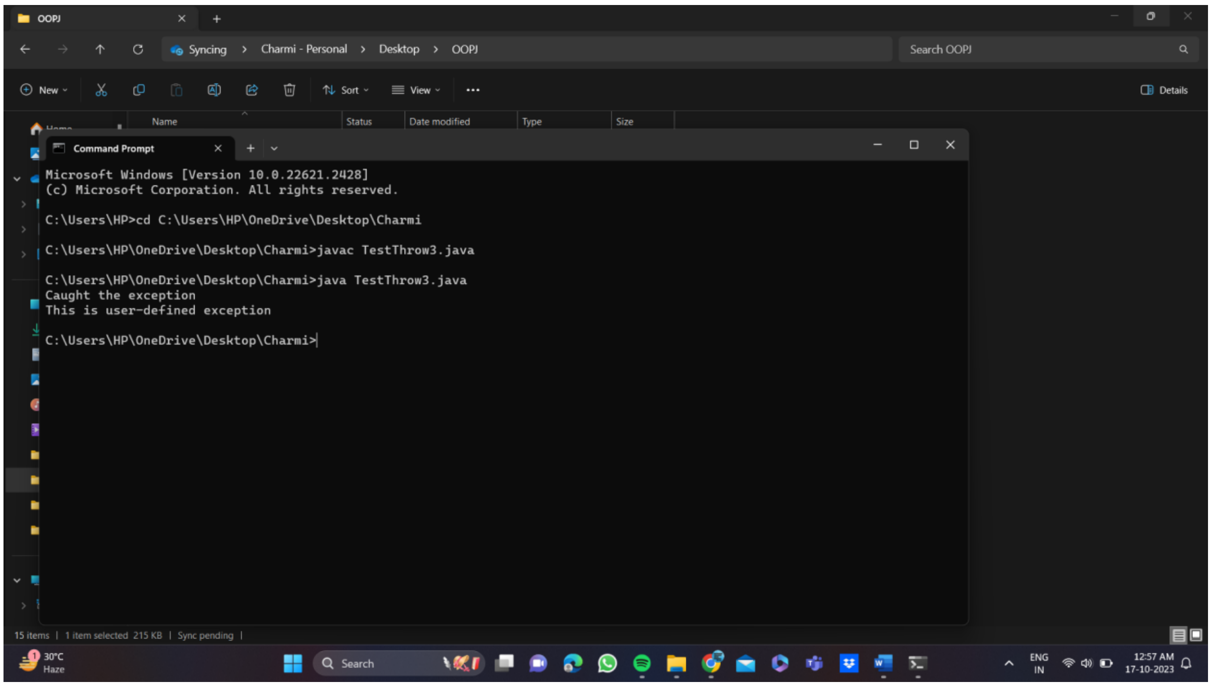
```
import java.io.IOException;
 class Testthrows2{
   public static void main(String args[]){
    try{
     M m=new M();
     m.method();
    }catch(Exception e){System.out.println("exception handled");}

    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:

Comment on how exceptions are handled in JAVA.

In Java, exceptions are handled using a combination of the try, catch, finally, and throw keywords.

Try-Catch Blocks (Using `try` and `catch`):

- The `try` block is used to encapsulate a section of code where an exception may occur. It defines the region in which you anticipate an exception.
- One or more `catch` blocks immediately follow the `try` block, and each `catch` block is associated with a specific type of exception. When an exception is thrown within the `try` block, the program flow transfers to the corresponding `catch` block that matches the exception type.
- In the `catch` block, you can handle the exception, log the error, or take corrective actions to deal with the exceptional situation. This helps in preventing the program from crashing due to runtime errors.

Finally Block (Using `finally`):

- The `finally` block, if provided, is executed regardless of whether an exception was thrown or not. It's useful for performing cleanup operations, like releasing resources (e.g., closing files or network connections), that need to be executed, regardless of the outcome of the `try` and `catch` blocks. This ensures that resources are properly managed and released.

Throwing Exceptions (Using `throw`):
- The `throw` keyword allows you to explicitly throw an exception within your code. This is often done when your code encounters a specific error condition that it cannot handle but wants to pass control to an exception handler.
- You can create and throw custom exceptions by instantiating an exception object and using `throw`. This is useful for creating and signaling application-specific exceptions that can be caught and handled by your code or higher-level error-handling mechanisms.

In summary, Java's exception handling mechanism provides a structured way to deal with runtime errors and exceptional situations. The `try` and `catch` blocks isolate potentially problematic code, the `finally` block ensures proper resource cleanup, and the `throw` keyword allows for explicit exception signaling. This approach helps maintain the stability and reliability of Java programs even when errors occur.