Object Oriented Programming

Unit 5

Exceptions

Introduction: Features, Checked and unchecked exception, Java Approach to handle exceptions, Exception Hierarchy, Implement Exception, The Throw Keyword, Implement Checked Exception, Implement Unchecked Exception, Implement Custom Exception.

What is an exception?

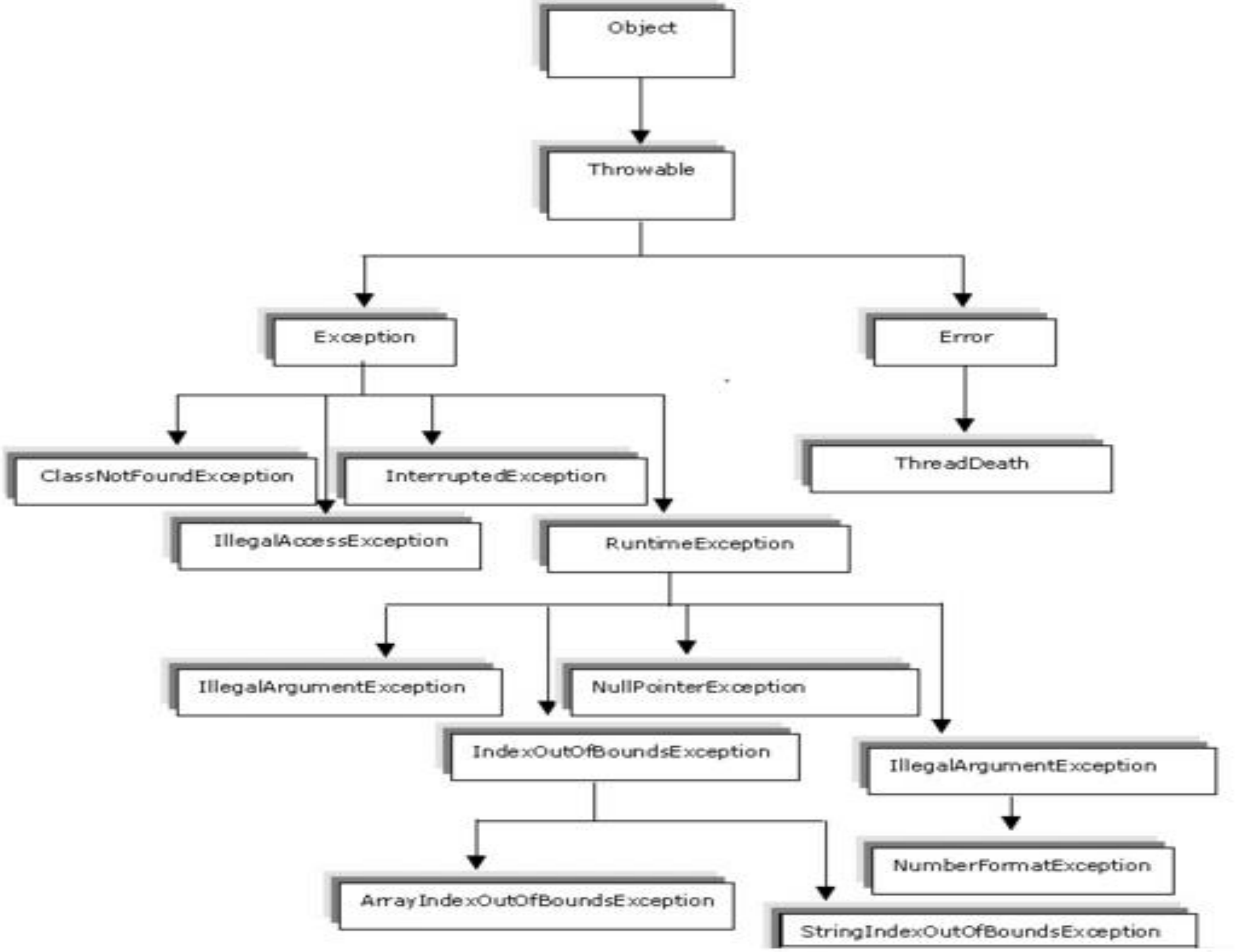
- ➢ An exception is an **unwanted or unexpected event**, which occurs during the execution of a program i.e. at run time, that disrupts the normal flow of the program's instructions.
- ➢ An exception can occur for many different reasons. Following are some scenarios where an exception occurs.
- ▪ A user has entered an invalid data.
- ▪ A file that needs to be opened cannot be found.
- ▪ A network connection has been lost in the middle of communications or the JVM has run out of memory.
- ➢ Exception can occur at **runtime (known as runtime exceptions) as well as at compile-time (known Compile-time exceptions)**.

Error vs Exception

- **Error:** An Error indicates **serious problem and abnormal conditions** that most applications should not try to handle. Error defines problems that are not expected to be caught under normal circumstances by our program. For example memory error, hardware error, JVM error etc.

- **Exception:** Exception indicates conditions that a **reasonable application might try to catch. Exceptions** are conditions within the code. A developer can handle such conditions and take necessary corrective actions.

Exception Hierarchy



Types of exceptions

1) Checked exceptions

- A checked exception is an exception that is checked (notified) by the compiler at compilation-time, these are also called **as compile time exceptions**. These exceptions cannot simply be ignored, the programmer should take care of (handle) these exceptions.
- For example, if you use **FileReader** class in your program to read data from a file, if the file specified in its constructor doesn't exist, then a *FileNotFoundException* occurs, and the compiler prompts the programmer to handle the exception.
- **Examples of Checked Exceptions :-**

ClassNotFoundException, IllegalAccessException, NoSuchFieldException, etc.

Types of exceptions(Contd.)

2) Unchecked exceptions

- An unchecked exception is an **exception that occurs at the time of execution**. These are also called as **Runtime Exceptions**. These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation.
- For example, if you have declared an array of size 5 in your program, and trying to call the 6th element of the array then anArrayIndexOutOfBounds Exception occurs.
- **Examples of Unchecked Exceptions:-** ArithmeticException, ArrayIndexOutOfBoundsException, NullPointerException, NegativeArraySizeException etc.

Catching Java Exceptions

- There are **5 keywords** used in java exception handling.
  - **try**
  - **catch**
  - **finally**
  - **throw**
  - **throws**

Java try block

- Java try block is used to enclose the code that might throw an exception. It must be used within the method.
- Java try block must be followed by either catch or finally block.

Java catch block

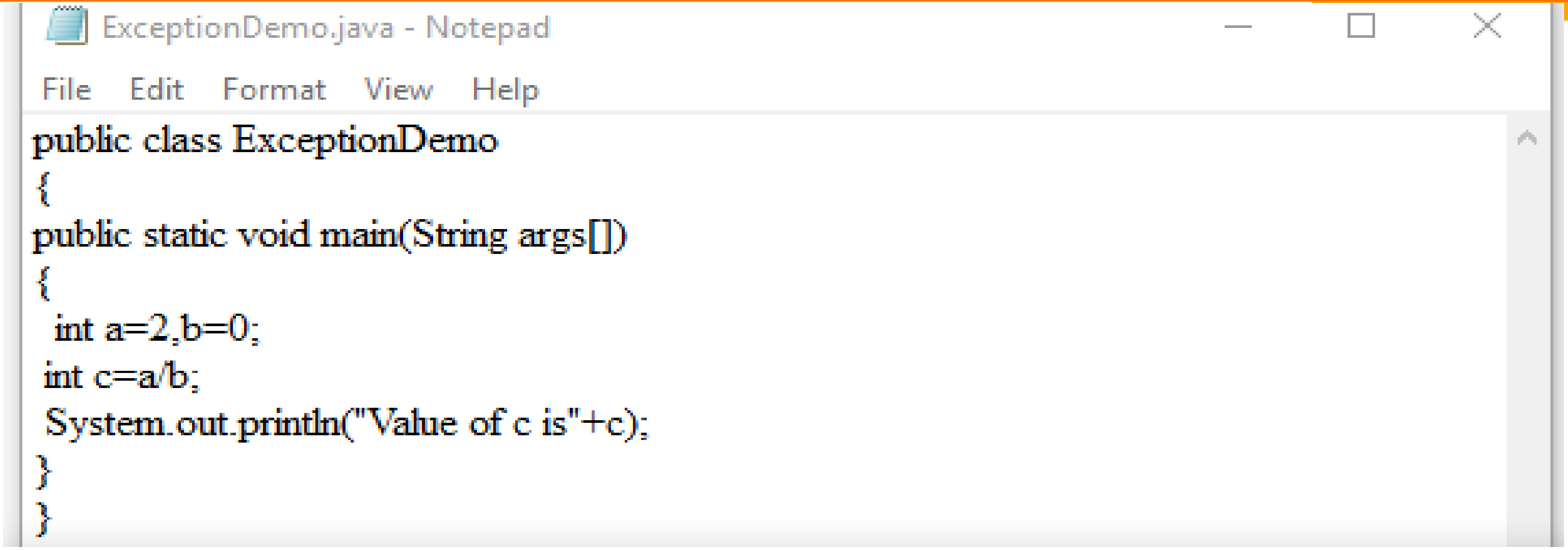
- Java catch block is used to handle the Exception. It must be used after the try block only.
- You can use multiple catch block with a single try.

Catching Java Exceptions

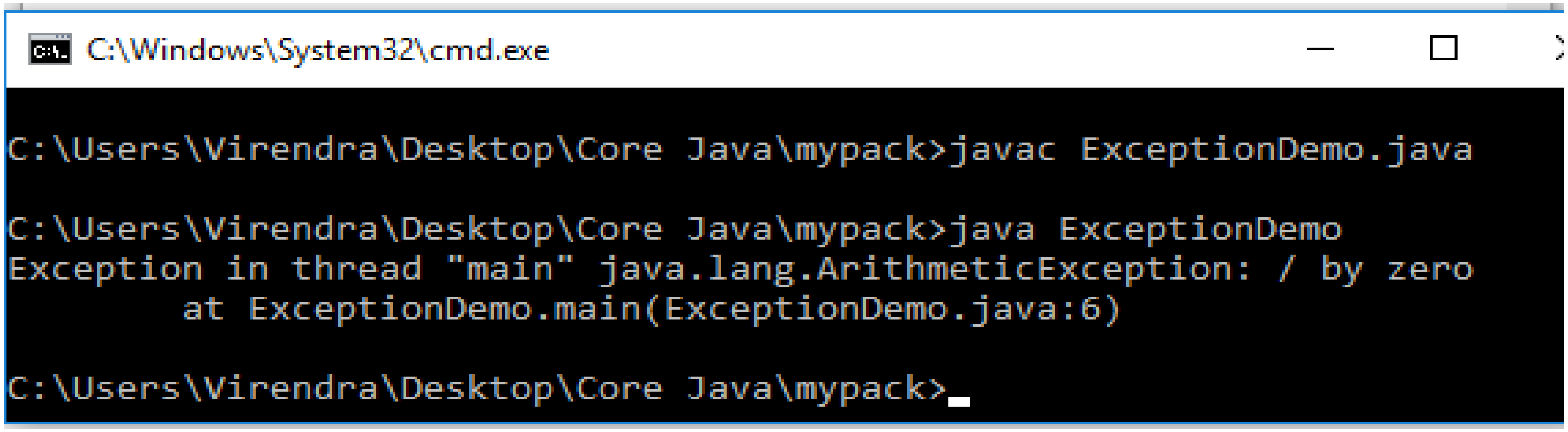
- **Syntax:**

```
try
{
   //Statements
}
catch(ExceptionName e)
{
//Statements
}
```

Without Java Exceptions

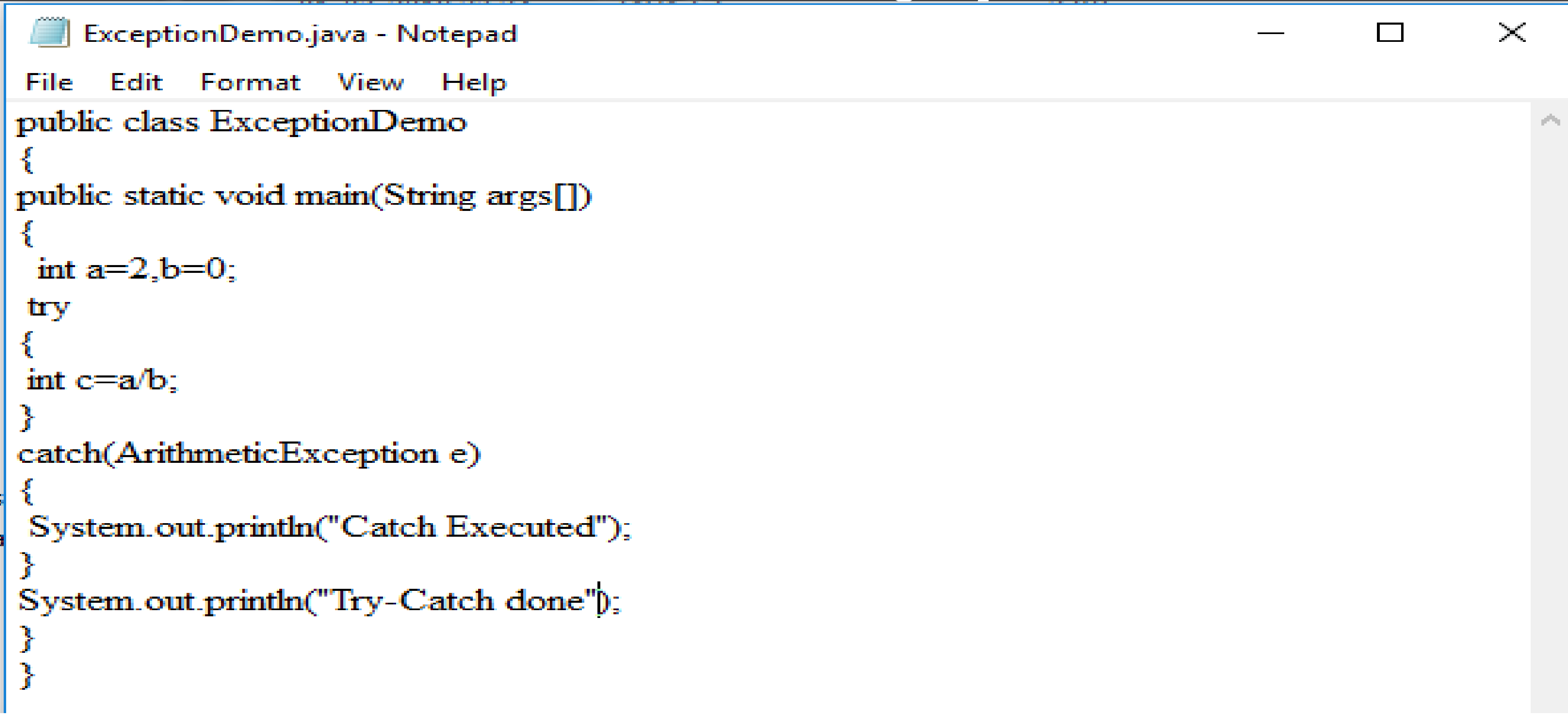


```
public class ExceptionDemo
{
public static void main(String args[])
{
  int a=2,b=0;
 int c=a/b;
 System.out.println("Value of c is"+c);
}
}
```

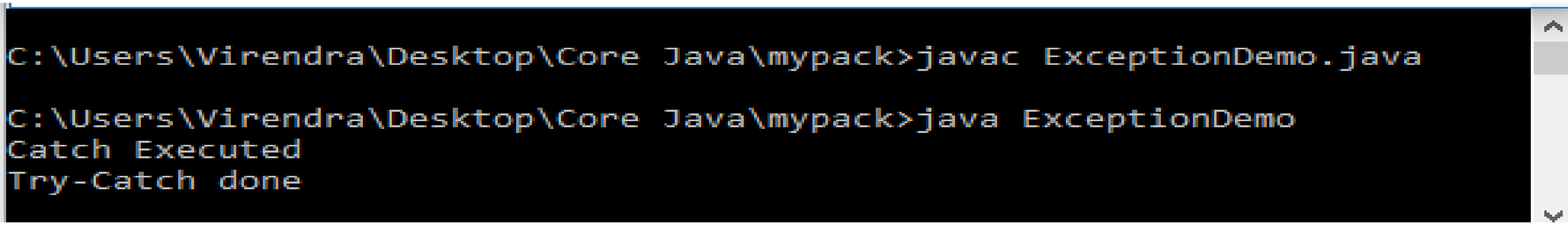


```
C:\Users\Virendra\Desktop\Core Java\mypack>javac ExceptionDemo.java

C:\Users\Virendra\Desktop\Core Java\mypack>java ExceptionDemo
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ExceptionDemo.main(ExceptionDemo.java:6)

C:\Users\Virendra\Desktop\Core Java\mypack>_
```

With Java Exceptions



```
public class ExceptionDemo
{
public static void main(String args[])
{
 int a=2,b=0;
 try
{
 int c=a/b;
}
catch(ArithmeticException e)
{
 System.out.println("Catch Executed");
}
System.out.println("Try-Catch done");
}
}
```



```
C:\Users\Virendra\Desktop\Core Java\mypack>javac ExceptionDemo.java

C:\Users\Virendra\Desktop\Core Java\mypack>java ExceptionDemo
Catch Executed
Try-Catch done
```

Examples of occurrences of Exception

ArithmeticException.java

```
class Example1
{
   public static void main(String args[])
   {
      try{
         int num1=30, num2=0;
         int output=num1/num2;
         System.out.println ("Result: "+output);
      }
      catch(ArithmeticException e)
      {
         System.out.println ("You shouldn't divide a number by zero");
      }
   }
}
```

Examples of occurrences of Exception

ArrayIndexOutOfBounds Exception

```
class ExceptionDemo2
{
   public static void main(String args[])
   {
      try{
        int a[]=new int[10];
        //Array has only 10 elements
        a[11] = 9;
      }
      catch(ArrayIndexOutOfBoundsException e)
     {
         System.out.println ("ArrayIndexOutOfBounds");
      }
   }
}
```

Examples of occurrences of Exception

NumberFormat Exception

```
class ExceptionDemo3
{
   public static void main(String args[])
   {
      try
    {
       int num=Integer.parseInt ("XYZ") ;
       System.out.println(num);
      }
       catch(NumberFormatException e)
      {
     System.out.println("Number format exception occurred");
       }
   }
}
```

Examples of occurrences of Exception

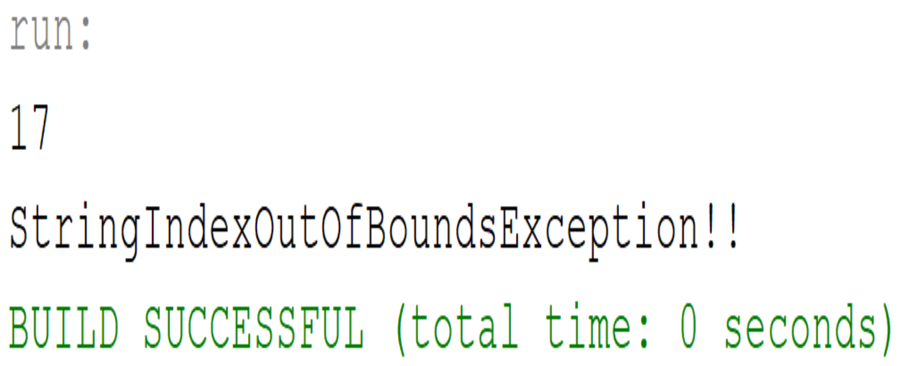
NullPointer Exception

```
class Exception2
{
   public static void main(String args[])
   {
              try
             {
                         String str=null;
                         System.out.println (str.length());
              }
        catch(NullPointerException e)
             {
                         System.out.println("NullPointerException..");
              }
   }
}
```

Examples of occurrences of Exception

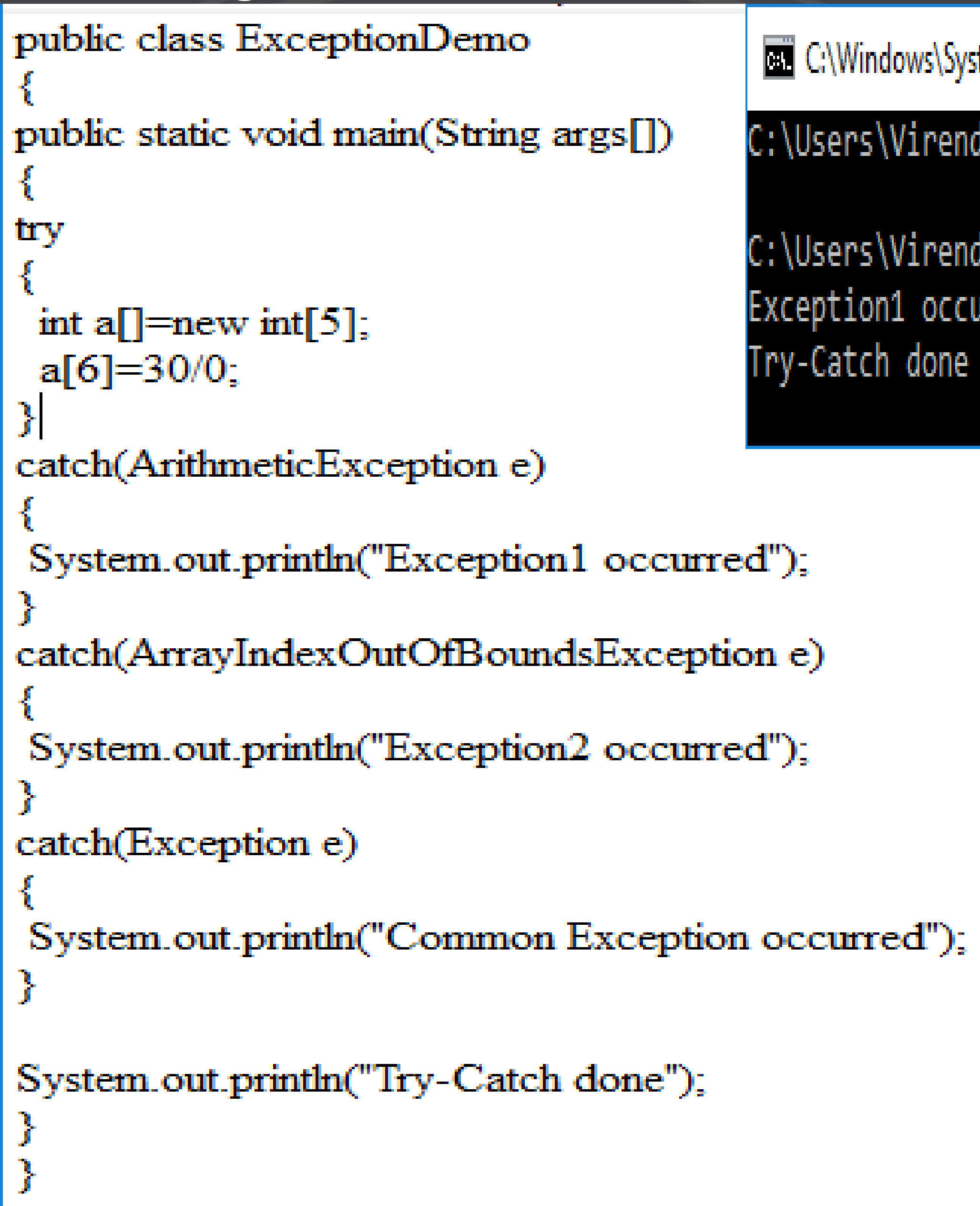
StringIndexOutOfBounds Exception

```
class ExceptionDemo4
{
   public static void main(String args[])
   {
      try
   {
        String str=“javabeginnersbook";
        System.out.println(str.length());
        char c = str.charAt(0);
        c = str.charAt(40);
        System.out.println(c);
      }
      catch(StringIndexOutOfBoundsException e)
     {
     System.out.println("StringIndexOutOfBoundsException!!");
      }
   }
}
```

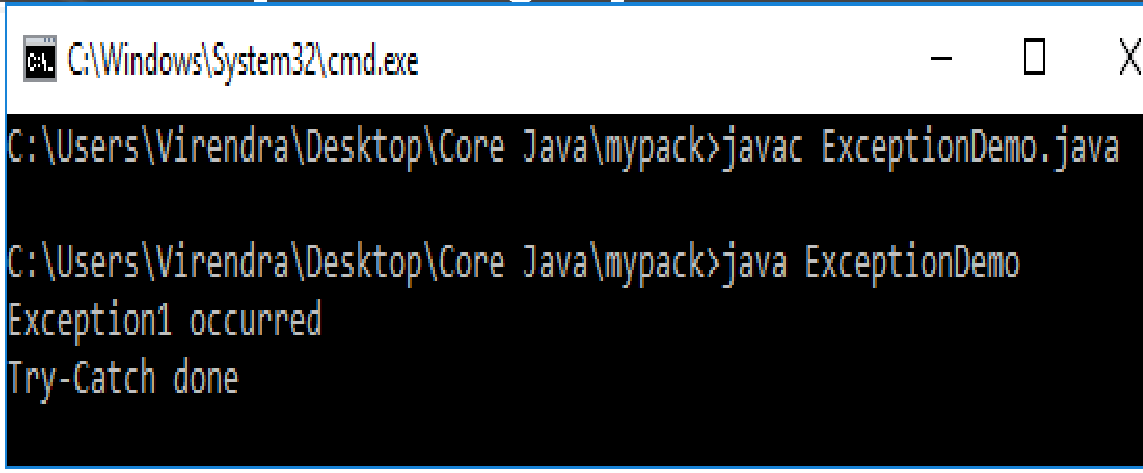


```
run:
17
StringIndexOutOfBoundsException!!
BUILD SUCCESSFUL (total time: 0 seconds)
```

Java catch multiple exceptions



```
public class ExceptionDemo
{
public static void main(String args[])
{
try
{
 int a[]=new int[5];
 a[6]=30/0;
}
catch(ArithmeticException e)
{
 System.out.println("Exception1 occurred");
}
catch(ArrayIndexOutOfBoundsException e)
{
 System.out.println("Exception2 occurred");
}
catch(Exception e)
{
 System.out.println("Common Exception occurred");
}

System.out.println("Try-Catch done");
}
}
```

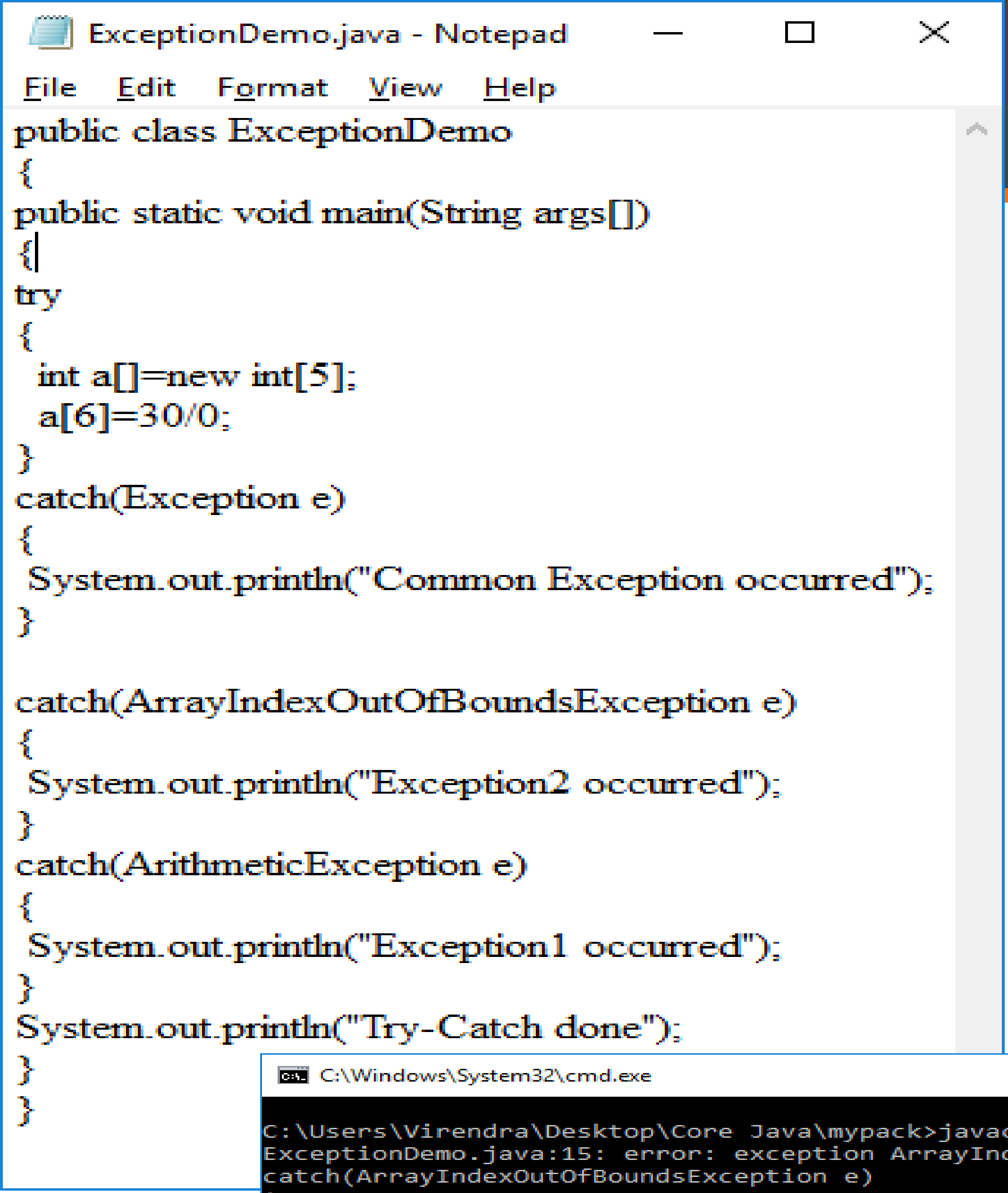


```
C:\Users\Virendra\Desktop\Core Java\mypack>javac ExceptionDemo.java

C:\Users\Virendra\Desktop\Core Java\mypack>java ExceptionDemo
Exception1 occurred
Try-Catch done
```

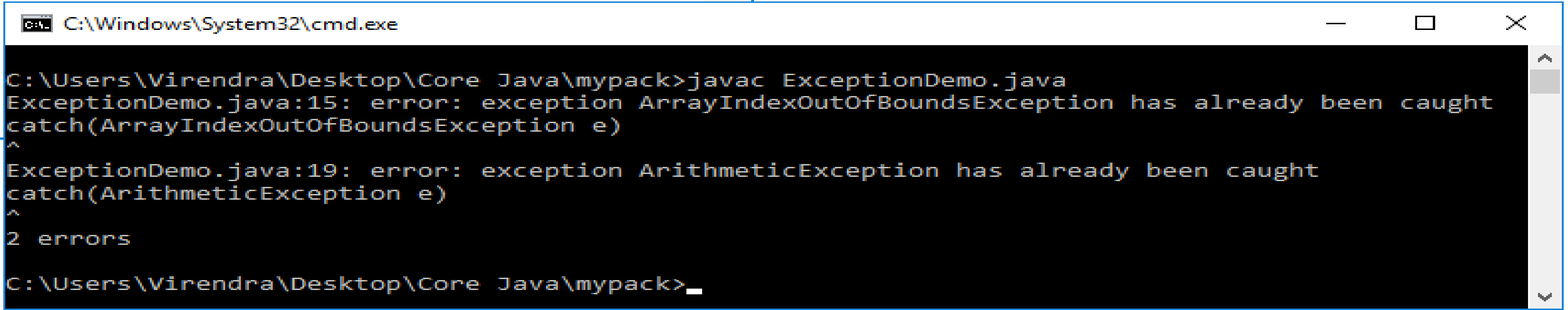
Rules in Exception Handling

- At a time **only one Exception occurs and at a time only one catch block is executed.**
- All catch blocks must be ordered from most specific to most general i.e. catch for ArithmeticException must come before catch for Exception .

Exception encountered



```
public class ExceptionDemo
{
public static void main(String args[])
{
try
{
 int a[]=new int[5];
 a[6]=30/0;
}
catch(Exception e)
{
 System.out.println("Common Exception occurred");
}

catch(ArrayIndexOutOfBoundsException e)
{
 System.out.println("Exception2 occurred");
}
catch(ArithmeticException e)
{
 System.out.println("Exception1 occurred");
}
System.out.println("Try-Catch done");
}
}
```



```
C:\Users\Virendra\Desktop\Core Java\mypack>javac ExceptionDemo.java
ExceptionDemo.java:15: error: exception ArrayIndexOutOfBoundsException has already been caught
catch(ArrayIndexOutOfBoundsException e)
^
ExceptionDemo.java:19: error: exception ArithmeticException has already been caught
catch(ArithmeticException e)
^
2 errors

C:\Users\Virendra\Desktop\Core Java\mypack>
```

The finally clause

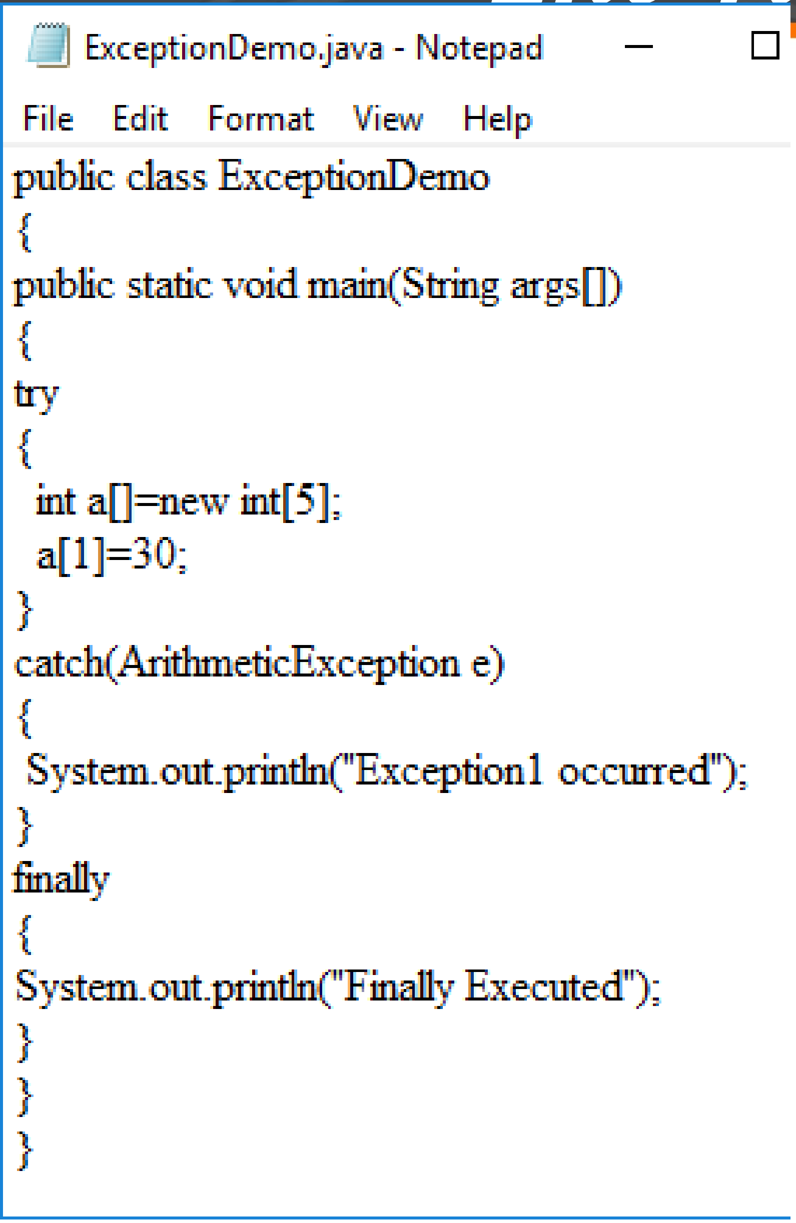
- **The finally block follows a try block or a catch block.**
- A finally block of code always executes, irrespective of occurrence of an Exception.
- Using a **finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code.**
- **Syntax:**

```
try
{ // Block of code with multiple exit points}
finally
{
    // Block of code that is always executed when the try block is
exited, no matter how the try block is exited
}
```

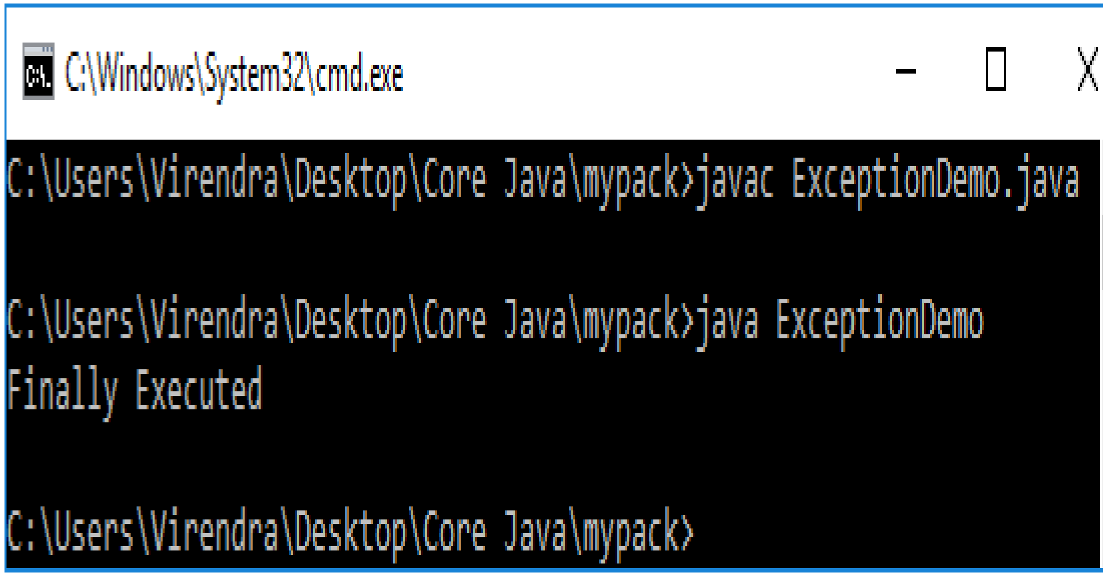
The finally clause

```
try
{
    //Statements that may cause an exception
}
catch
{
   //Handling exception
}
finally
{
   //Statements to be executed
}
```

The finally clause



```
public class ExceptionDemo
{
public static void main(String args[])
{
try
{
 int a[]=new int[5];
 a[1]=30;
}
catch(ArithmeticException e)
{
 System.out.println("Exception1 occurred");
}
finally
{
System.out.println("Finally Executed");
}
}
}
```



```
C:\Users\Virendra\Desktop\Core Java\mypack>javac ExceptionDemo.java

C:\Users\Virendra\Desktop\Core Java\mypack>java ExceptionDemo
Finally Executed

C:\Users\Virendra\Desktop\Core Java\mypack>
```

The throw keyword

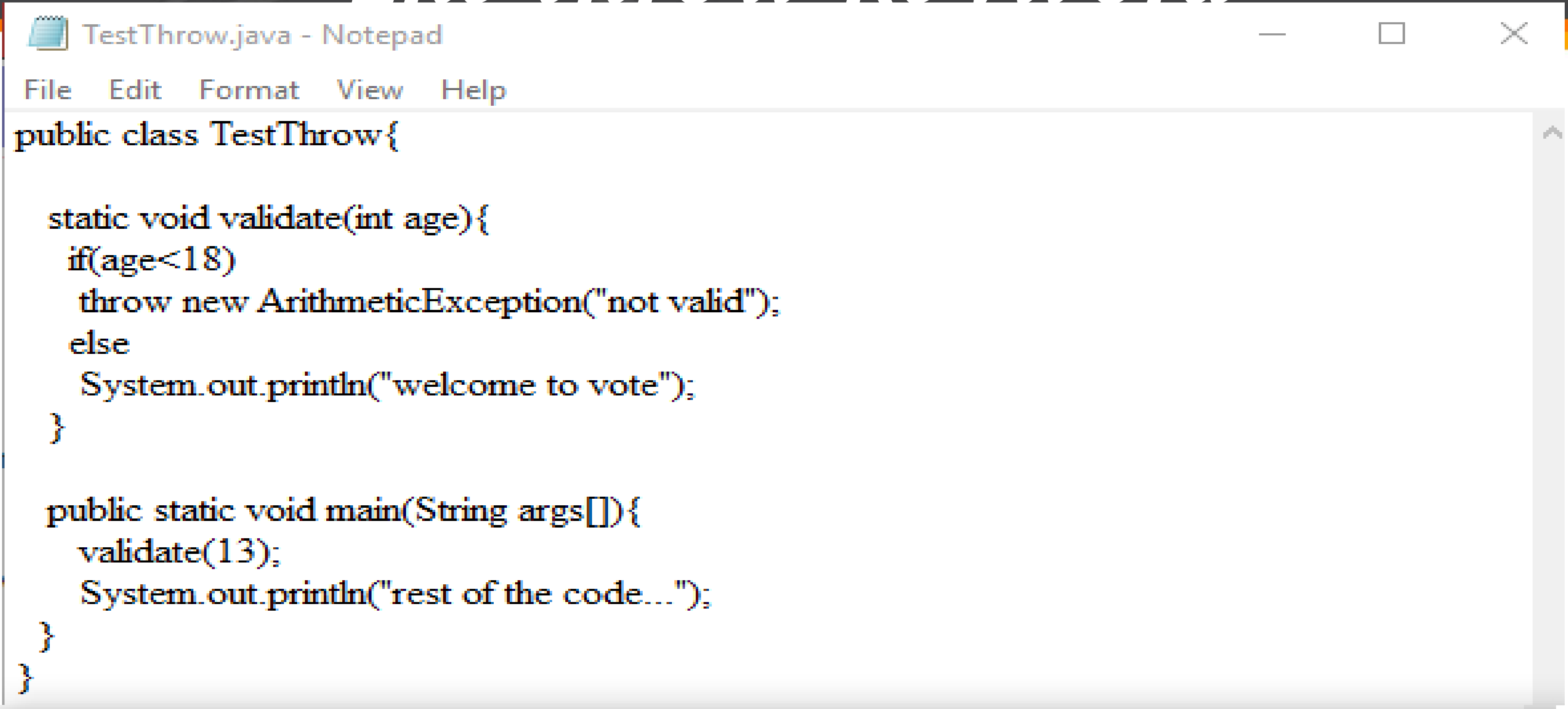
- The **Java throw keyword** is used to explicitly throw an exception. The throw keyword is mainly used to throw **custom exception.**
- The syntax of java throw keyword is given below.

  throw exception;

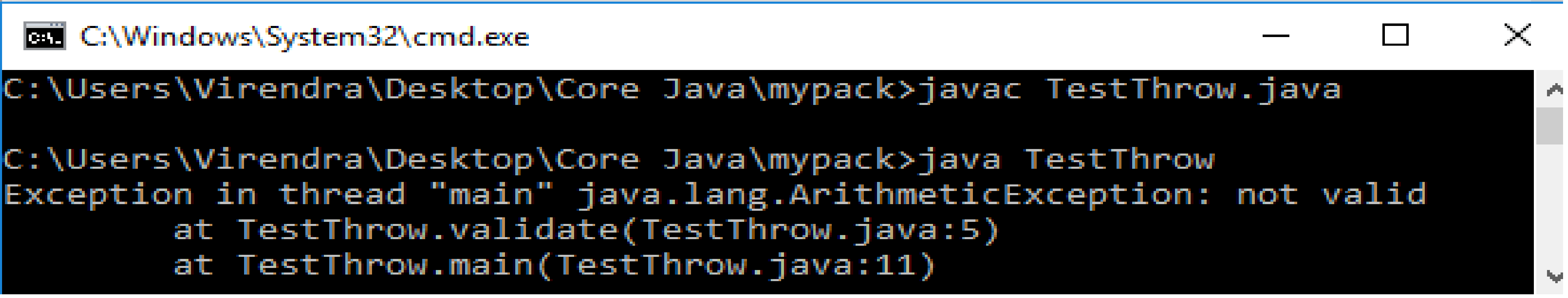
- Example of throw IOException:

  throw new IOException("sorry device error");

The throw keyword

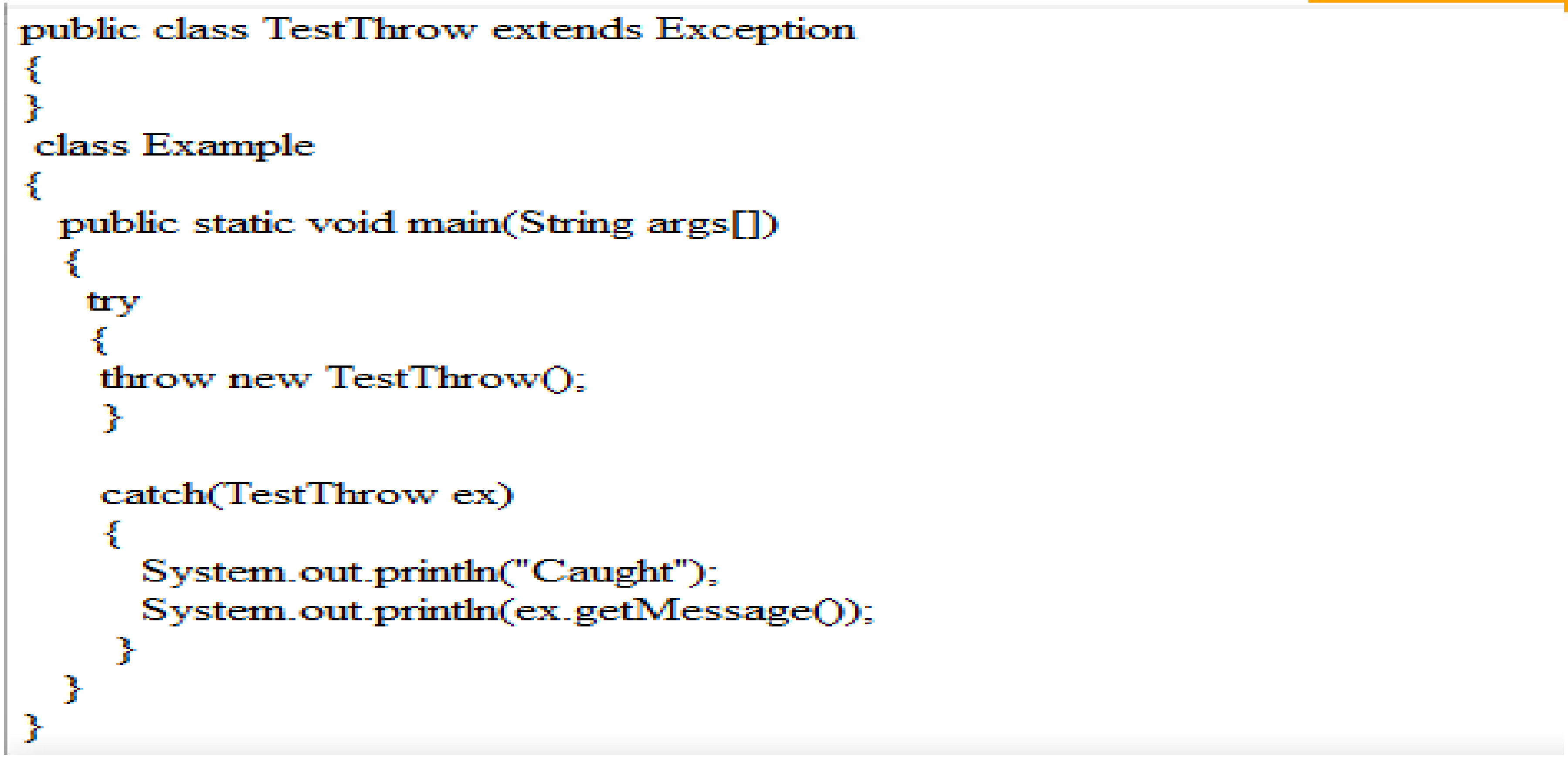


```
public class TestThrow{

  static void validate(int age){
   if(age<18)
    throw new ArithmeticException("not valid");
   else
    System.out.println("welcome to vote");
  }

  public static void main(String args[]){
    validate(13);
    System.out.println("rest of the code...");
  }
}
```



```
C:\Users\Virendra\Desktop\Core Java\mypack>javac TestThrow.java

C:\Users\Virendra\Desktop\Core Java\mypack>java TestThrow
Exception in thread "main" java.lang.ArithmeticException: not valid
        at TestThrow.validate(TestThrow.java:5)
        at TestThrow.main(TestThrow.java:11)
```

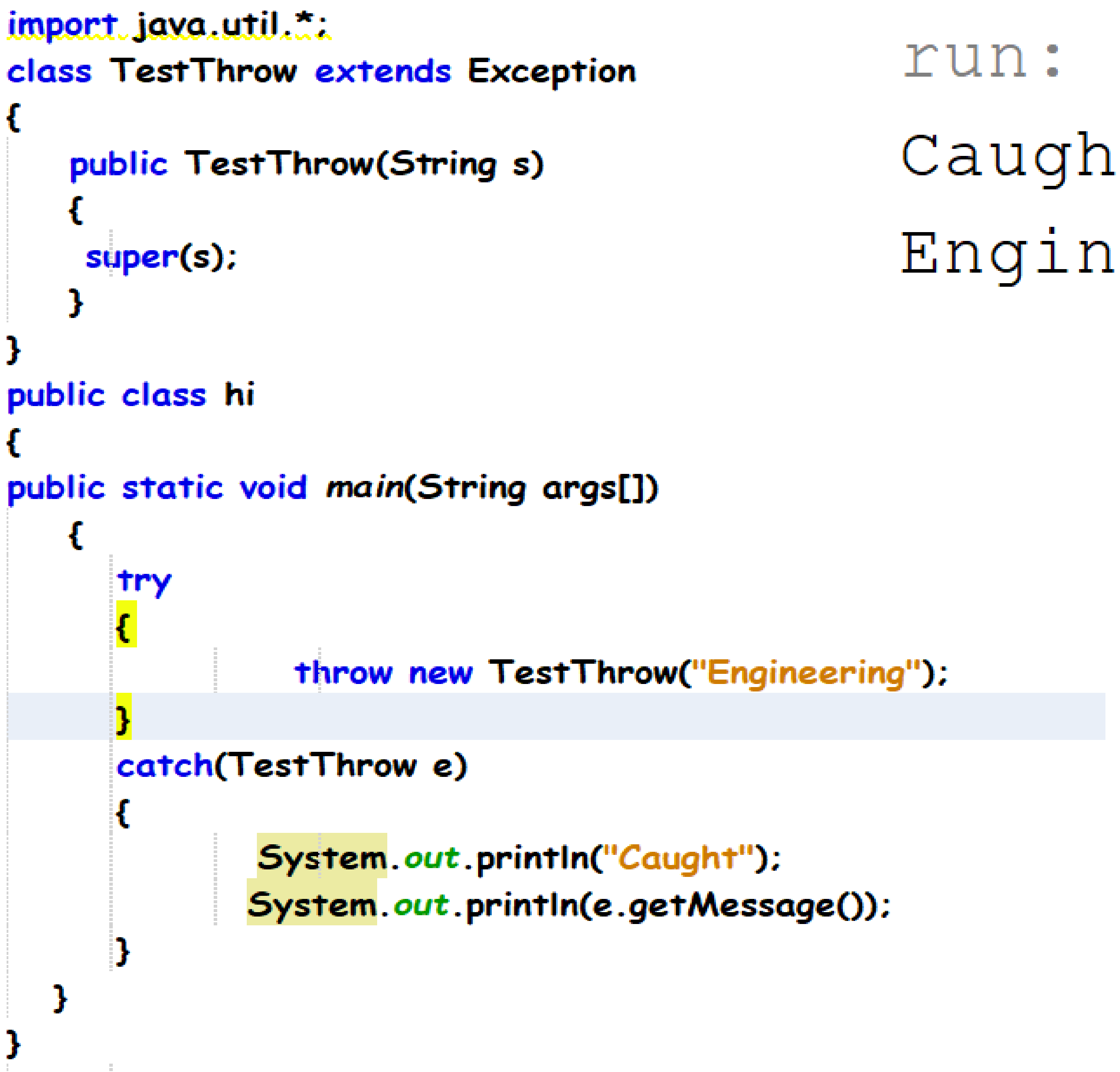
throw keyword



```
public class TestThrow extends Exception
{
}
 class Example
{
   public static void main(String args[])
   {
     try
     {
      throw new TestThrow();
      }

      catch(TestThrow ex)
      {
         System.out.println("Caught");
         System.out.println(ex.getMessage());
       }
   }
}
```

```
run:
Caught
null
BUILD SUCCESSFUL (total time: 0 seconds)
```

throw keyword



```
import java.util.*;
class TestThrow extends Exception
{
    public TestThrow(String s)
    {
     super(s);
    }
}
public class hi
{
public static void main(String args[])
    {
       try
       {
                throw new TestThrow("Engineering");
       }
       catch(TestThrow e)
       {
                 System.out.println("Caught");
                System.out.println(e.getMessage());
       }
    }
}
```

```
run:
Caught
Engineering
```

Implementing a custom exception

```
 class MyUserDefinedException extends Exception
 {
    public MyUserDefinedException(String s)
    {
        super(s);
    }
 }

public class MyUserDefinedExceptionDemo
{
   public static void main(String args[])
   {
       try
      {
          throw new MyUserDefinedException("My user defined Exception class");
       } catch (MyUserDefinedException ex){
          System.out.println("Caught user defined exception");
          System.out.println(ex.getMessage());
       }
   }
}
```

Output

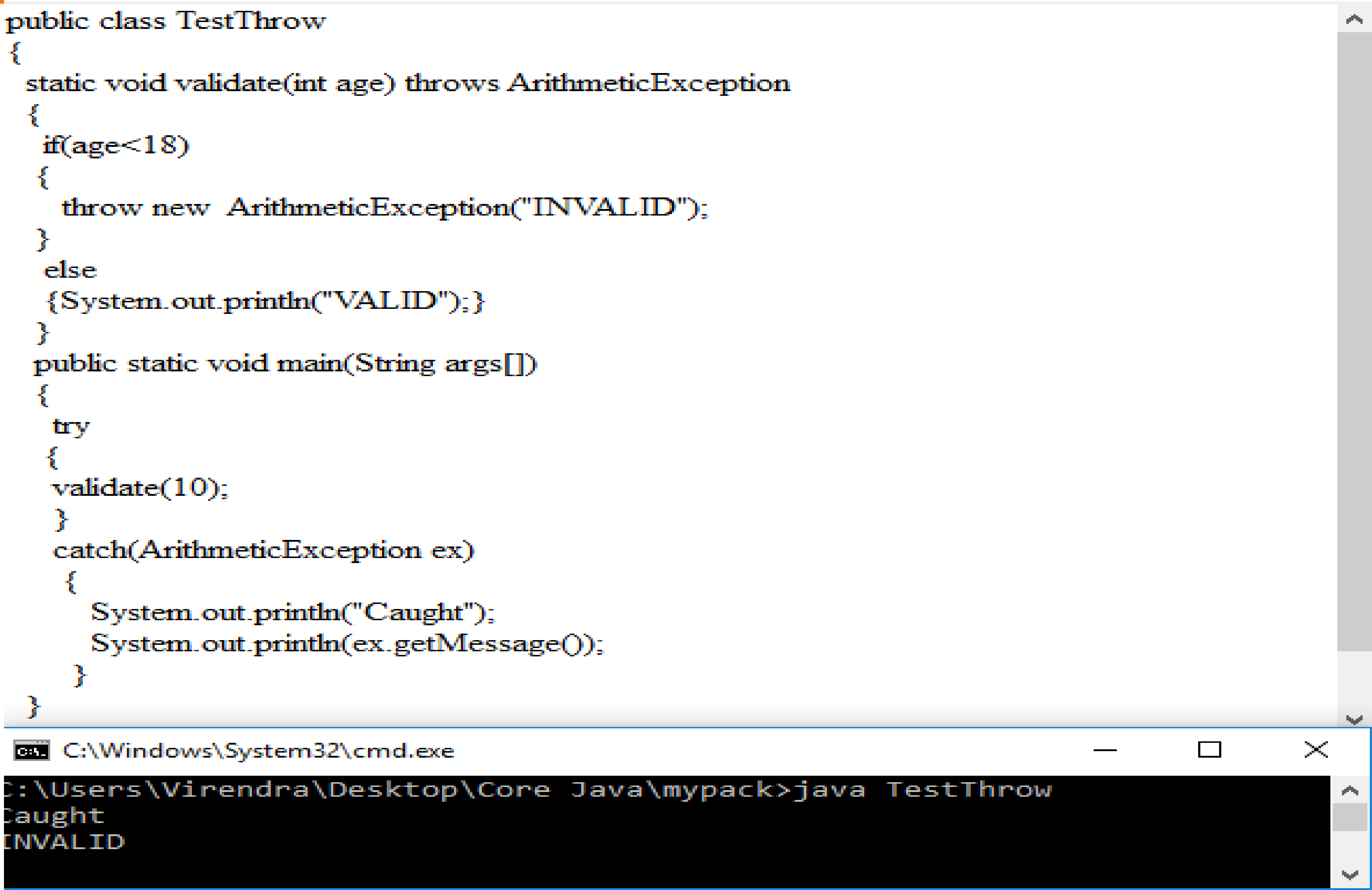
```
Caught user defined exception
My user defined Exception class
```

The throws clause

- The Java **throws keyword** is used to **declare an exception**. It 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.
- **Syntax:**

```
return_type method_name() throws exception_class_name
{
        //method code
}
```

throw and throws keyword



```
public class TestThrow
{
  static void validate(int age) throws ArithmeticException
  {
   if(age<18)
   {
     throw new  ArithmeticException("INVALID");
   }
   else
   {System.out.println("VALID");}
   }
  public static void main(String args[])
   {
    try
    {
    validate(10);
     }
    catch(ArithmeticException ex)
     {
        System.out.println("Caught");
        System.out.println(ex.getMessage());
      }
  }
```

C:\Windows\System32\cmd.exe

```
:\Users\Virendra\Desktop\Core Java\mypack>java TestThrow
aught
INVALID
```

Advantages of Exception handling

- **detect errors easily** without writing additional code to test return values
- exception-handling code is clearly **separated from exception-generating code**
- the same exception-handling code can deal with several possible exceptions
- code to handle an exception that may occur in the governed region needs to be written only once