OBJECT ORIENTED ANALYSIS & DESIGN
DATA STRUCTURES & ALGORITHMS

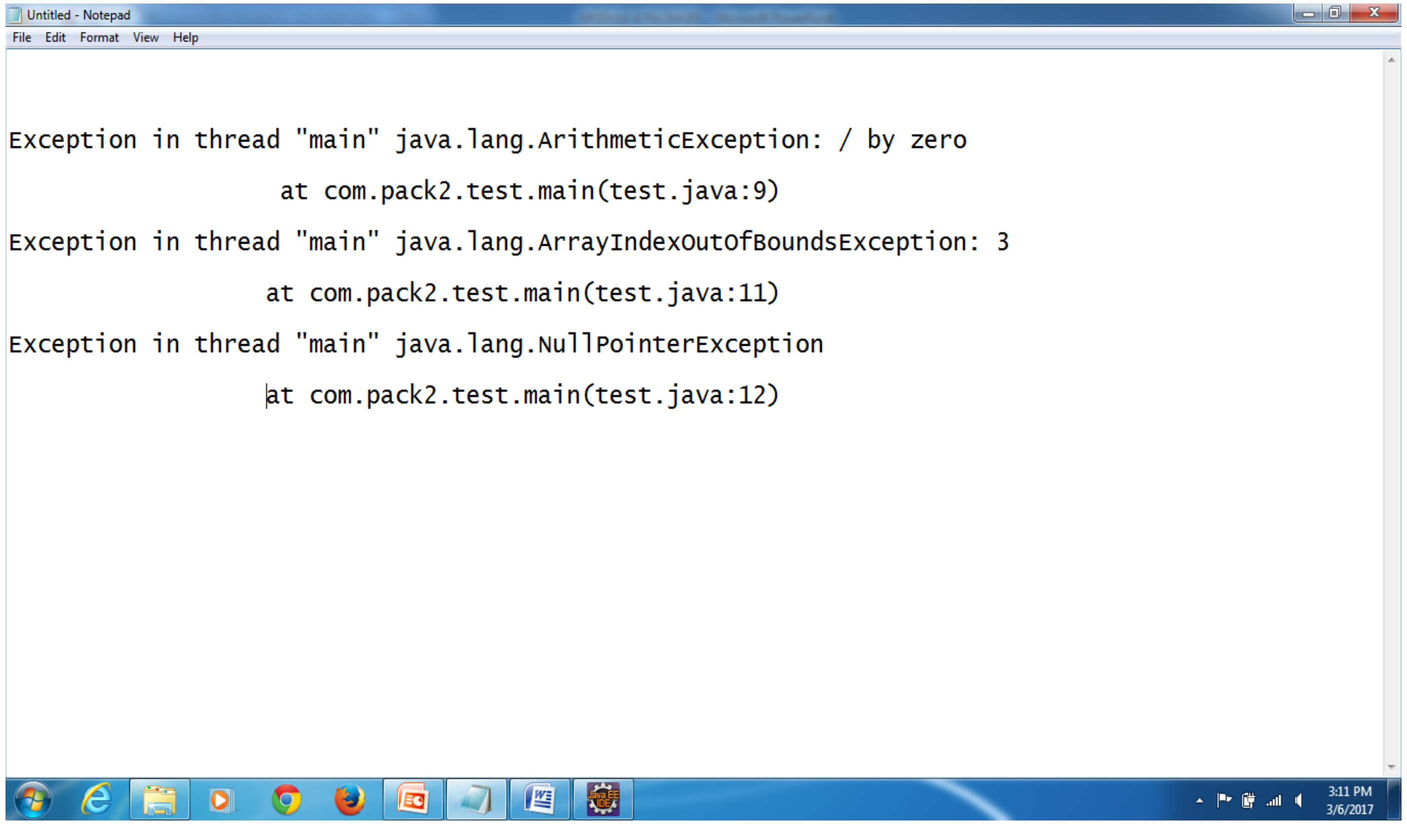
Exceptions

Exception

An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions—oracle.

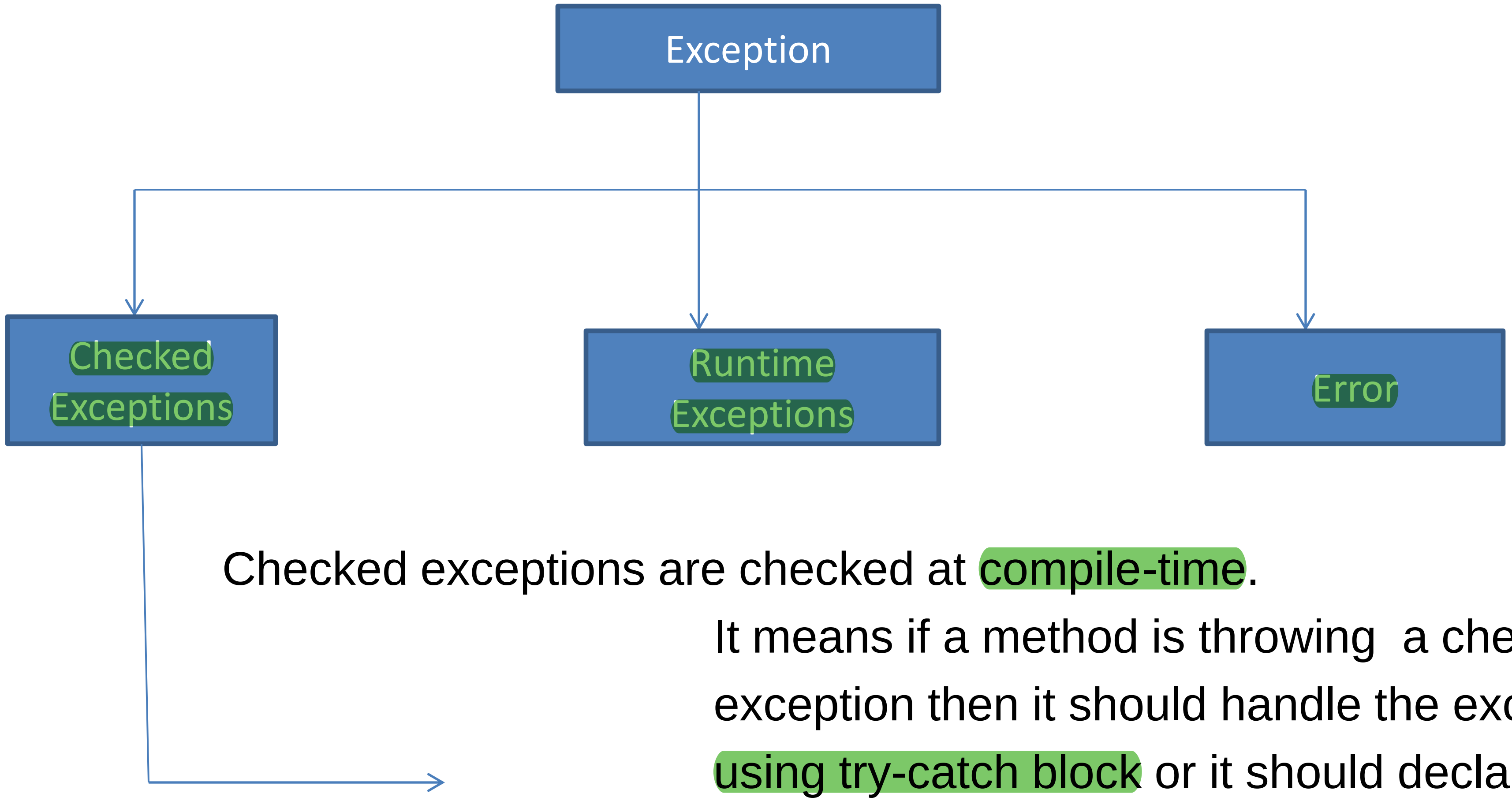
It is often referred to as run-time error.

Below are few of them :



```
Exception in thread "main" java.lang.ArithmeticException: / by zero
                   at com.pack2.test.main(test.java:9)
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
                  at com.pack2.test.main(test.java:11)
Exception in thread "main" java.lang.NullPointerException
                  at com.pack2.test.main(test.java:12)
```

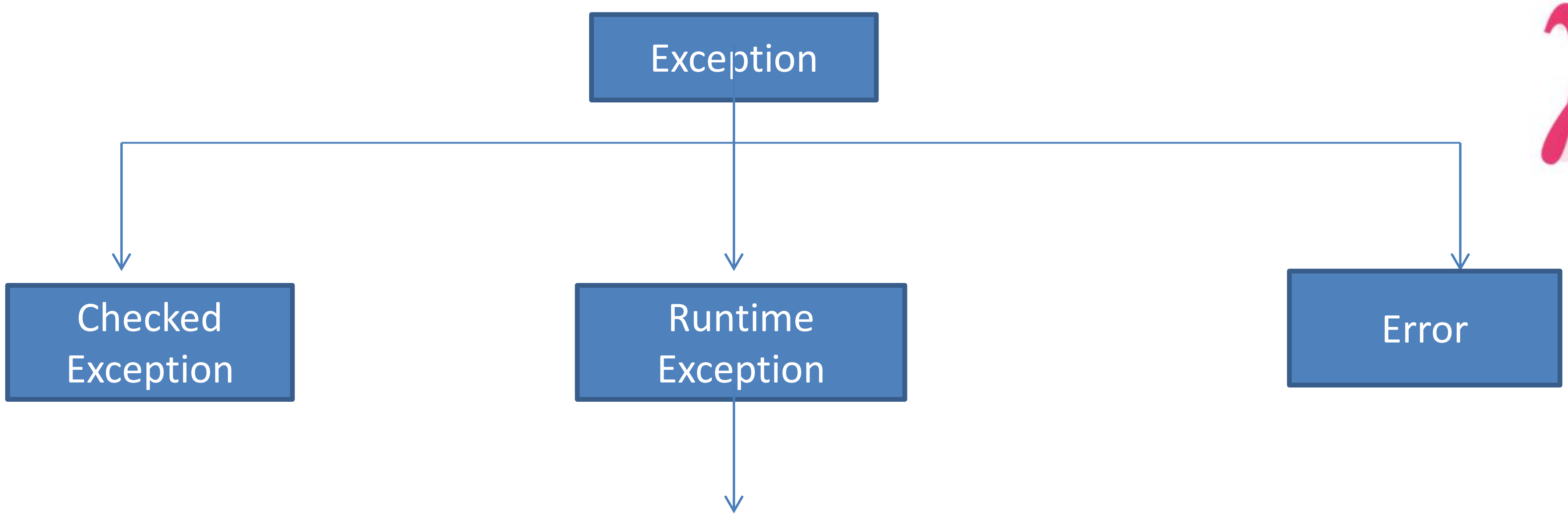
Types of Exception



Checked exceptions are checked at compile-time.

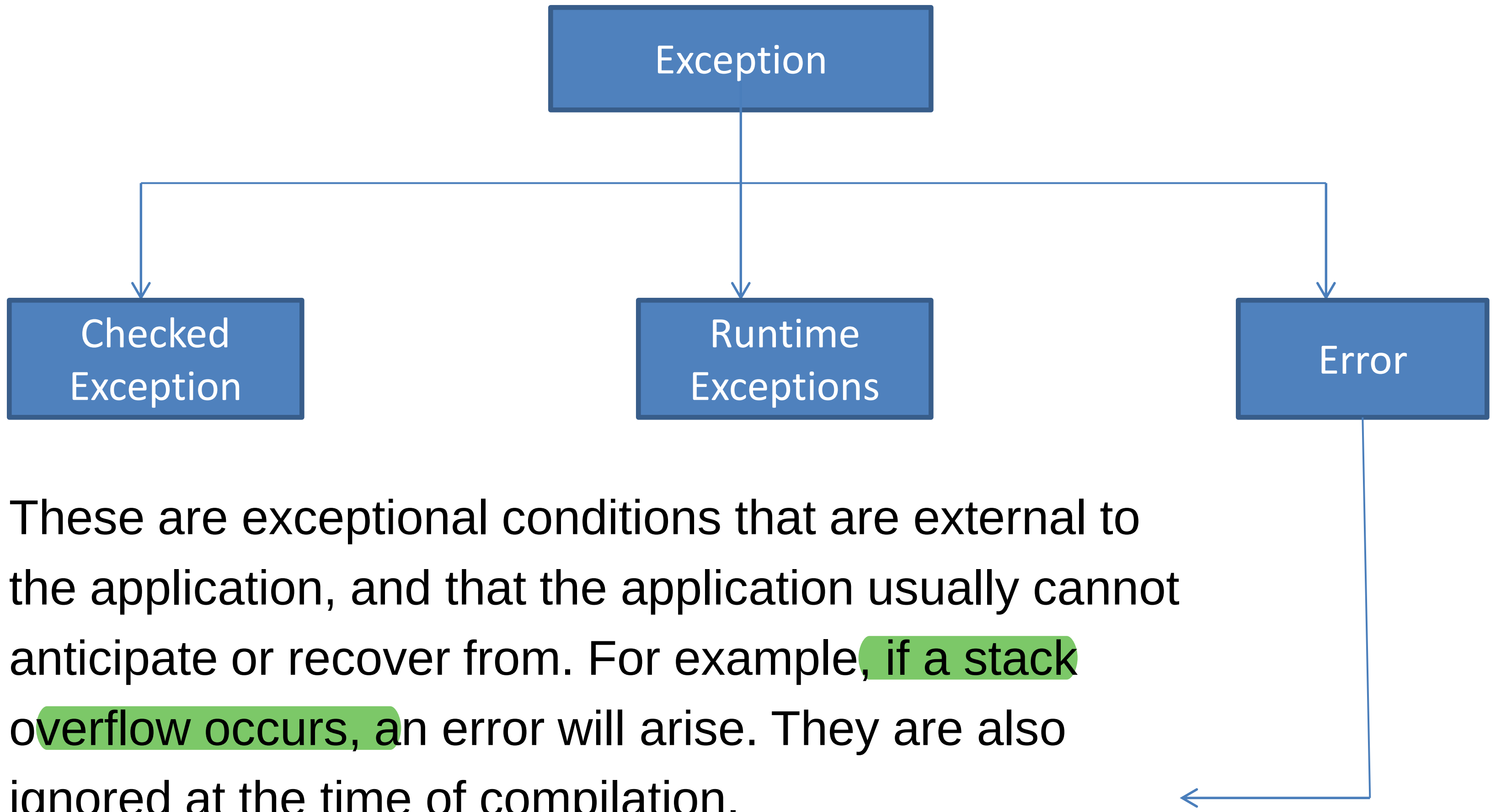
It means if a method is throwing a checked exception then it should handle the exception using try-catch block or it should declare the exception using throws keywords, otherwise the program will give a compilation error.

Types of Exception



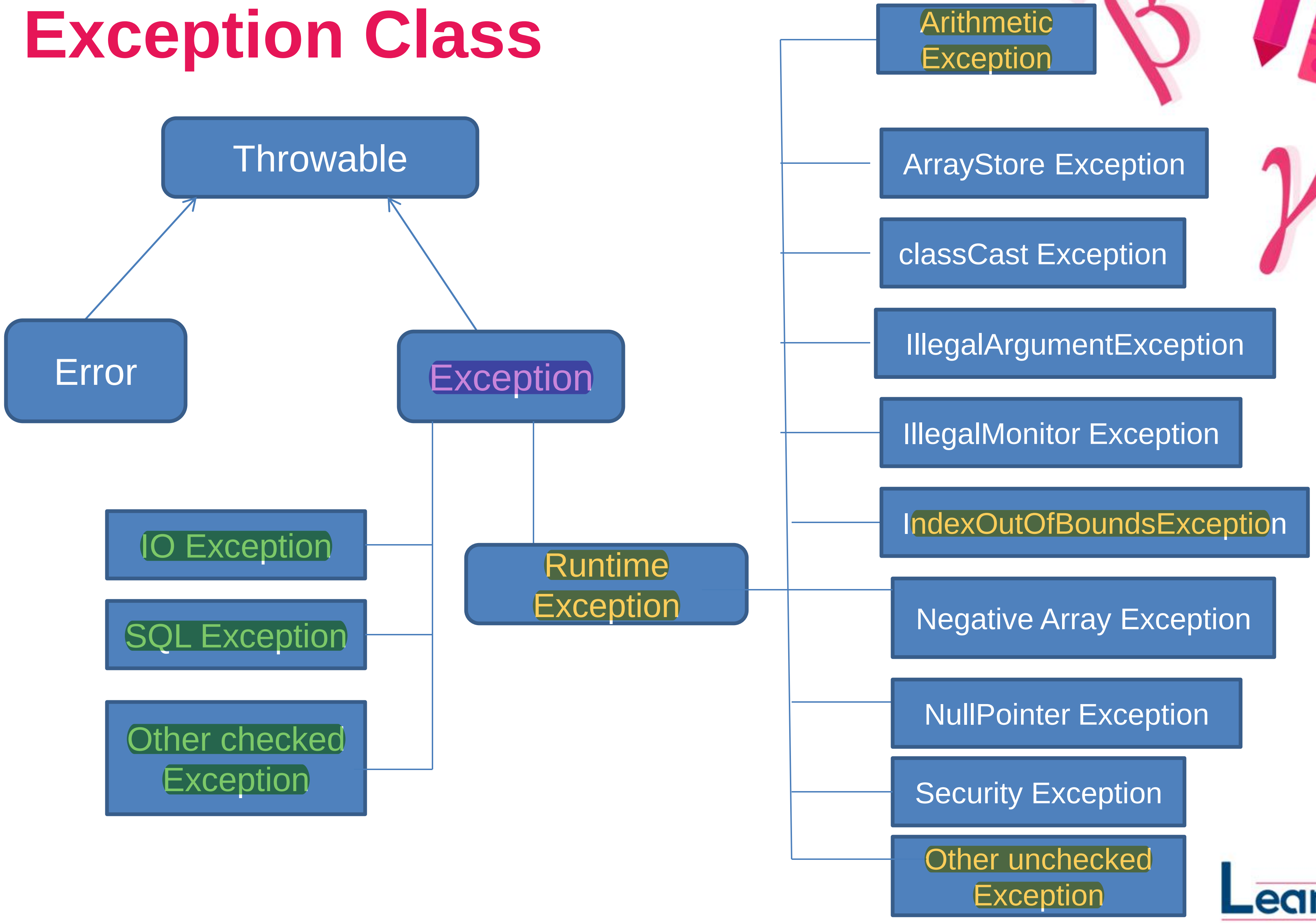
Unchecked exceptions are not checked at compile time . It means if your program is throwing an unchecked exception and even if you didn’t handle/declare that exception, the program won’t give a compilation error. It is up to the programmer to judge the conditions in advance, that can cause such exceptions and handle them appropriately. All unchecked exceptions are direct sub classes of RuntimeException class.

Types of Exception



These are exceptional conditions that are external to the application, and that the application usually cannot anticipate or recover from. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

Exception Class



Why Exceptional Handling?

```
// Divide by Zero Problem
int x= 5 / 0 ;
System.out.println(x);
```

output error program will halt

```
Exception in thread “main” java.lang.ArithmeticException : /by zero
        at com.pack2.test.main(test.java:9)
```

Array Index Out of Bound

```
int  arr[] = {1, 2, 3} ;

System.out.println(arr[3]) ;
```

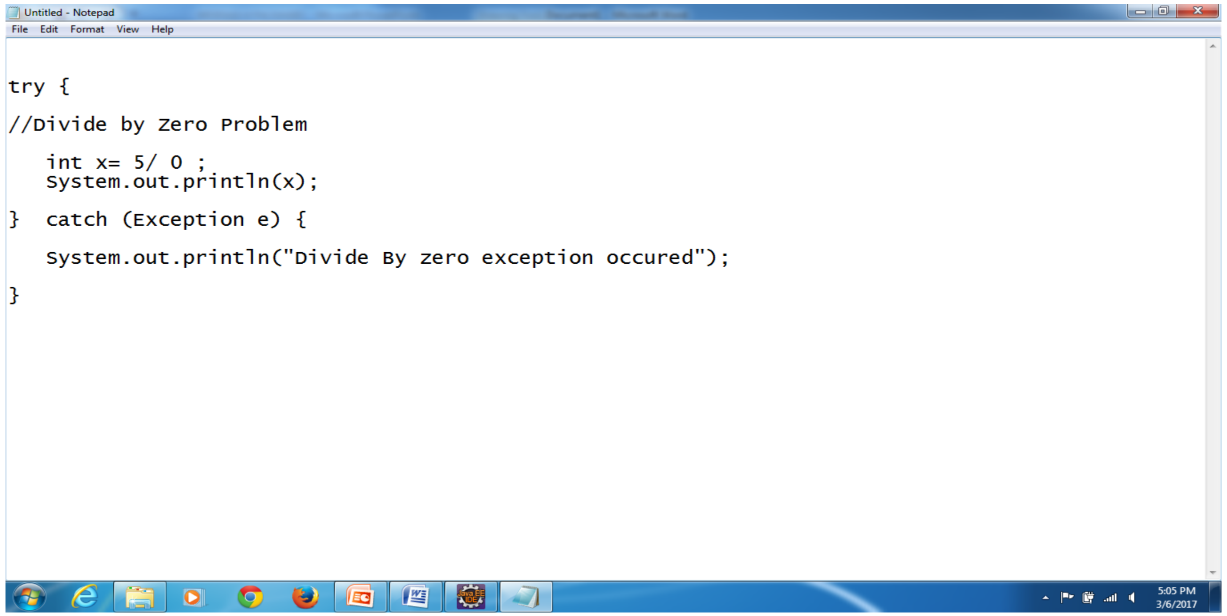
output

error!

program will halt

```
Exception in
 thread“main”java.lang.ArrayIndexOutOfBoundsException: 3
 at com.pack2.test.main(test.java:11)
```

Solution


```
try {
//Divide by Zero Problem
   int x= 5/ 0 ;
   System.out.println(x);
}  catch (Exception e) {
   System.out.println("Divide By zero exception occured");
}
```

Output

Divide By Zero exception occurred.

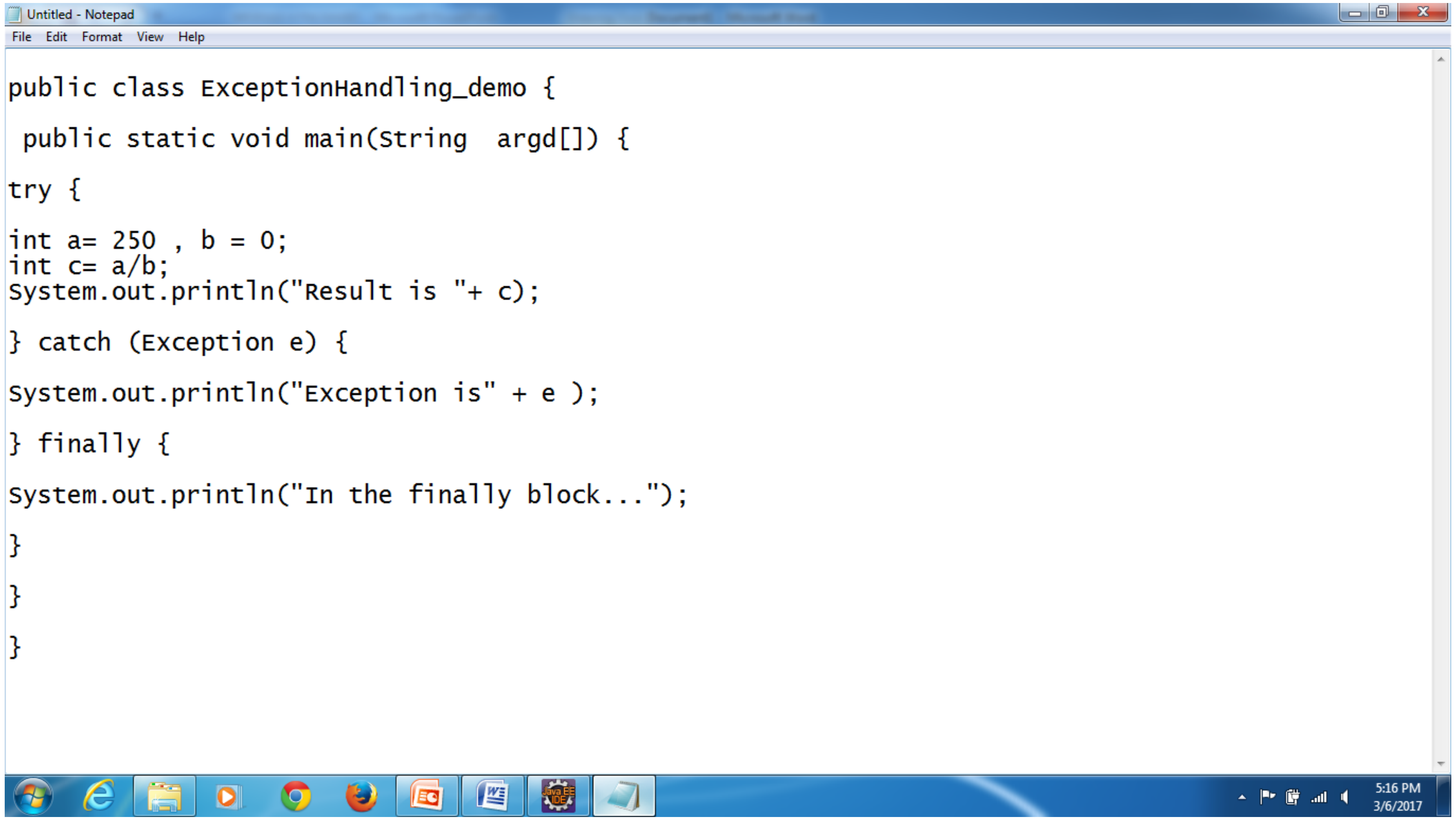
Exception Handling

- If there is a run time error then program is crashed and control out of the program.
- This issue can be solved by exception handling.
- Mainly, try, catch and finally are keywords for exception handling.

Exception Handling (contd.)

- **try:** All the statements to be executed should be placed in the try block.
- **catch:** If there are any issues or runtime errors, control comes in catch block.
- **finally:** Whether successful or unsuccessful execution , statements in the finally block gets executed.

Program on Exception Handling



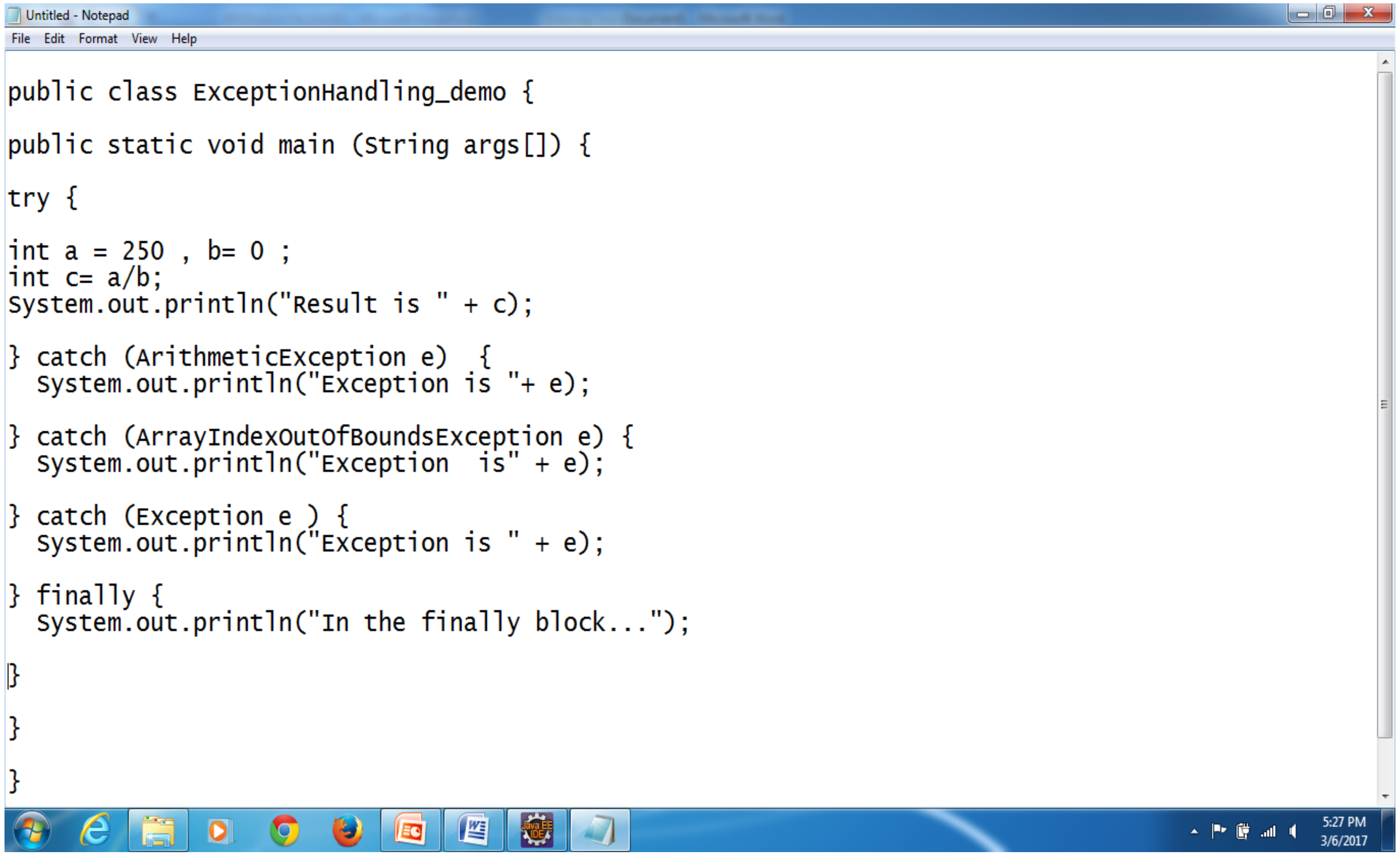
```
public class ExceptionHandling_demo {

 public static void main(String  argd[]) {

try {

int a= 250 , b = 0;
int c= a/b;
System.out.println("Result is "+ c);

} catch (Exception e) {

System.out.println("Exception is" + e );

} finally {

System.out.println("In the finally block...");

}

}

}
```

Exception Handling

- One try can have multiple catch blocks. In this scenarios, depends on the type of exception thrown corresponding catch blocks is invoked.

- Since all the exceptions are derived from Exception, catch (Exception e) should be placed at last. It can catch all the exceptions.

Program on Multiple Catch Blocks



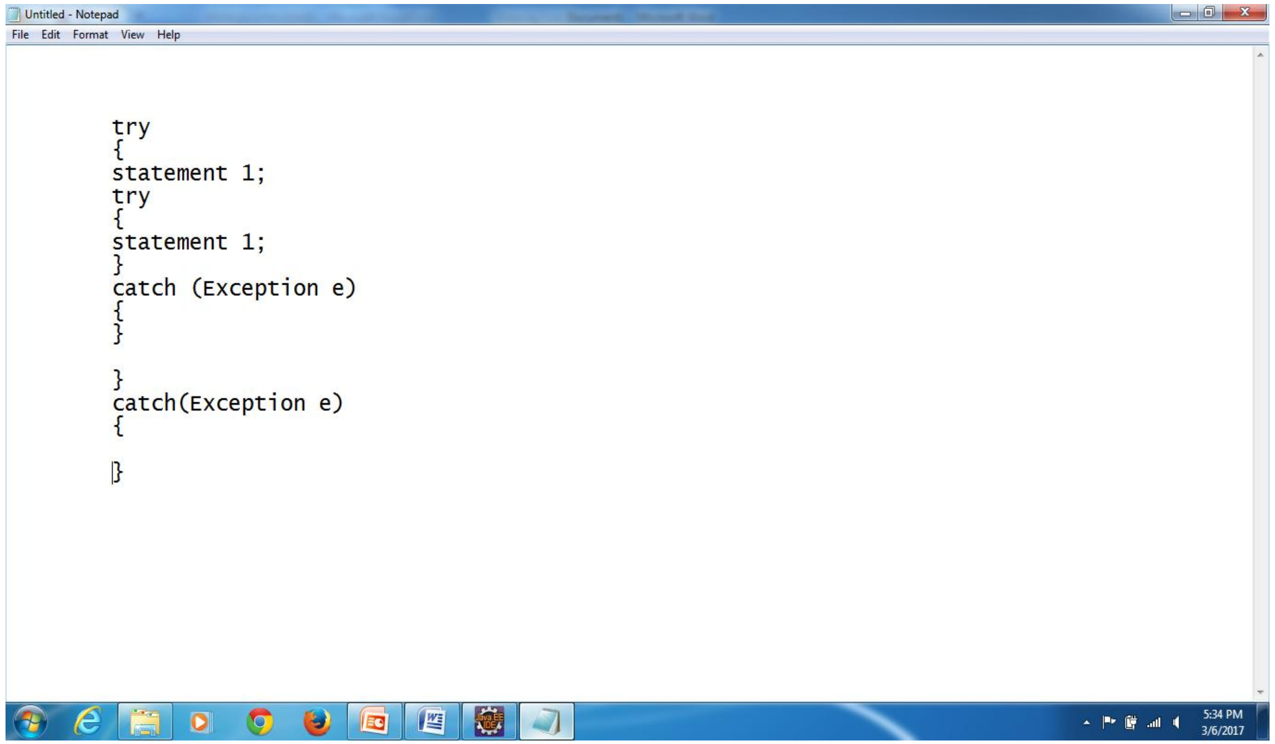
```
public class ExceptionHandling_demo {

public static void main (String args[]) {

try {

int a = 250 , b= 0 ;
int c= a/b;
System.out.println("Result is " + c);

} catch (ArithmeticException e)  {
   System.out.println("Exception is "+ e);

} catch (ArrayIndexOutOfBoundsException e) {
   System.out.println("Exception  is" + e);

} catch (Exception e ) {
   System.out.println("Exception is " + e);

} finally {
   System.out.println("In the finally block...");

}

}

}
```

Nested try catch

Why use nested try block?

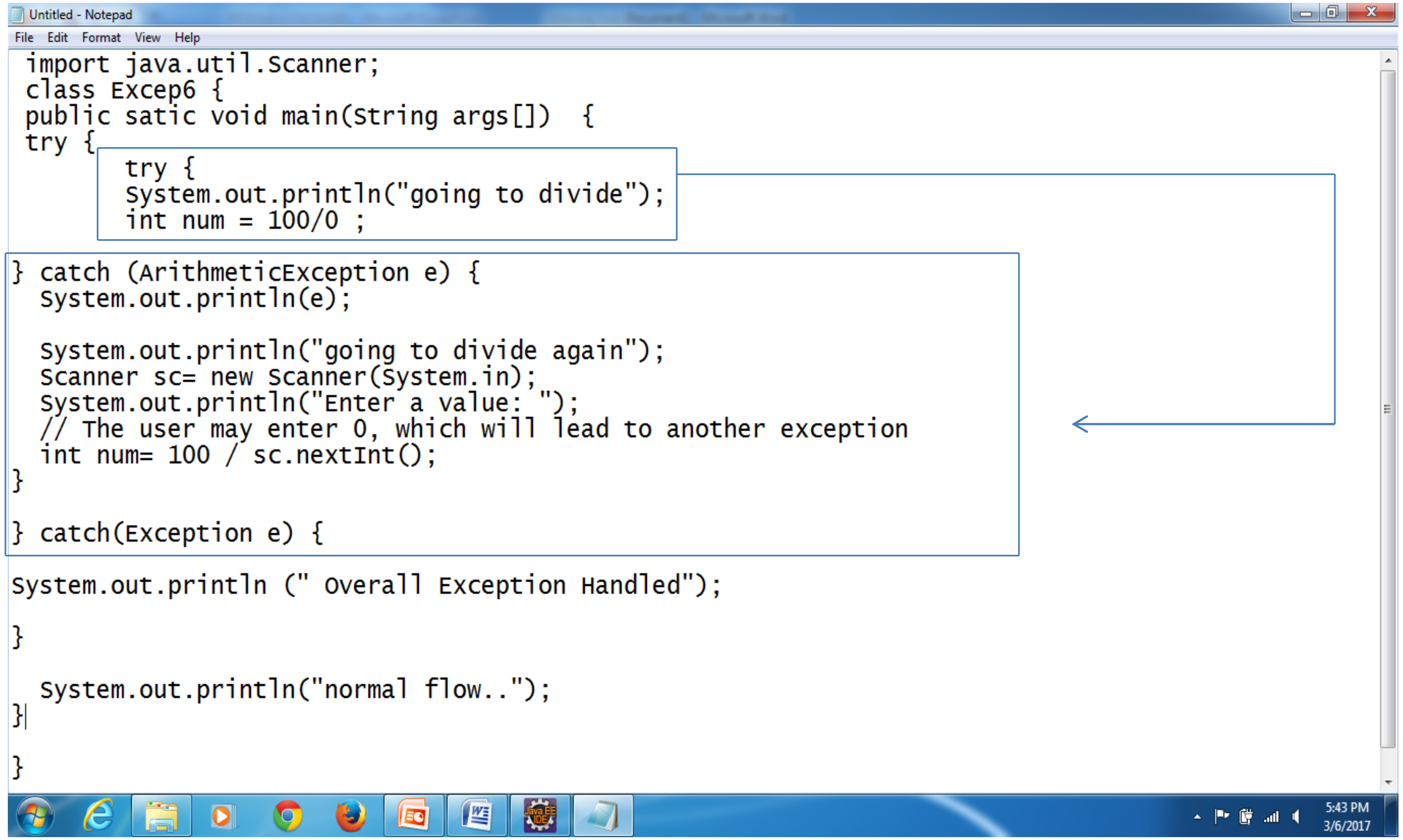
Sometimes a situation may arise where a part of a block may cause one error and the entire block itself ma cause another error. In such cases, exception handlers have to be nested.

Syntax:



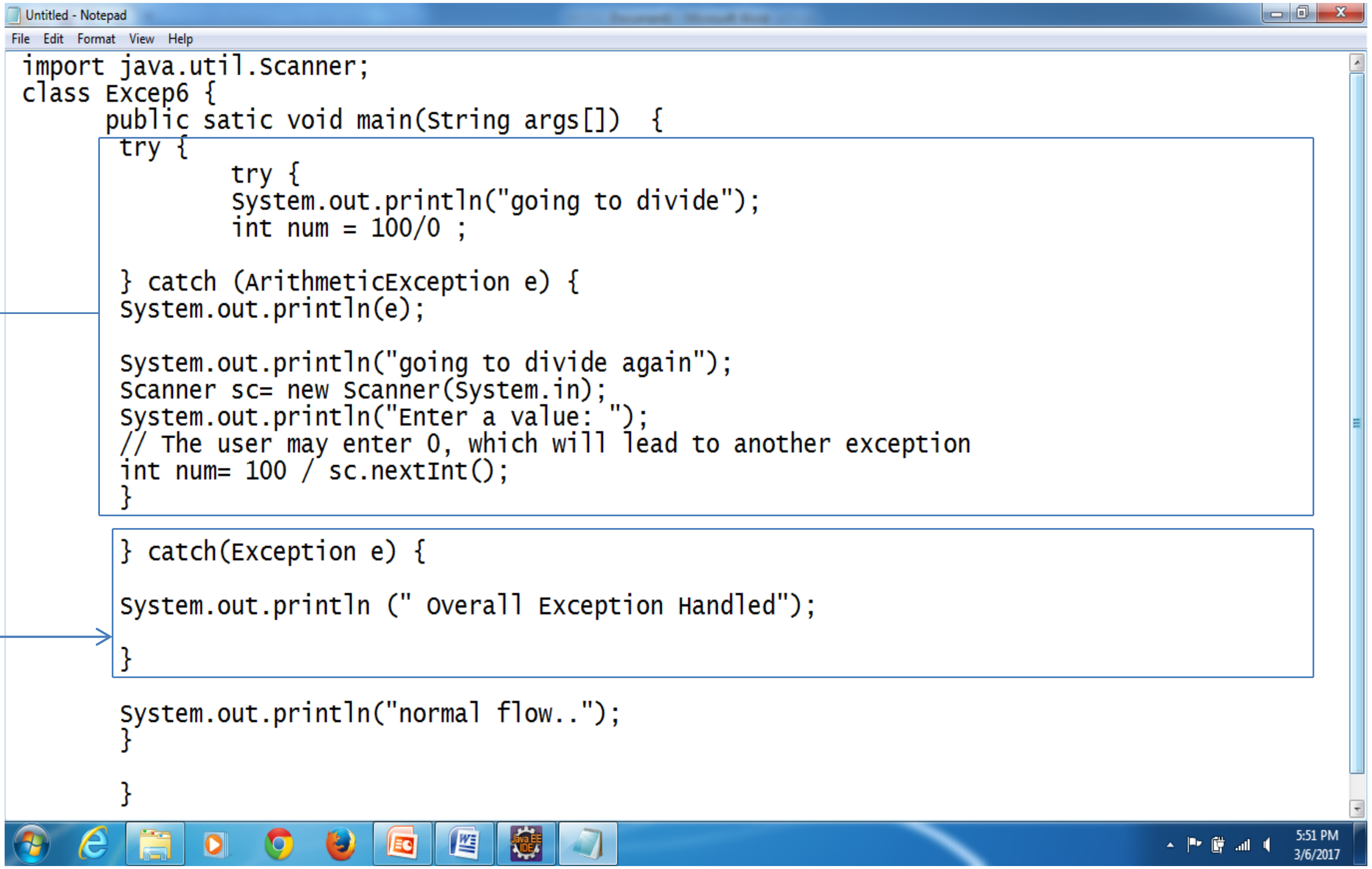
```
try
{
statement 1;
try
{
statement 1;
}
catch (Exception e)
{
}

}
catch(Exception e)
{

}
```

Program on Nested try catch



```
import java.util.Scanner;
class Excep6 {
public satic void main(String args[])  {
try {
       try {
       System.out.println("going to divide");
       int num = 100/0 ;

} catch (ArithmeticException e) {
  System.out.println(e);

  System.out.println("going to divide again");
  Scanner sc= new Scanner(System.in);
  System.out.println("Enter a value: ");
  // The user may enter 0, which will lead to another exception
  int num= 100 / sc.nextInt();
}

} catch(Exception e) {

System.out.println (" Overall Exception Handled");

}

  System.out.println("normal flow..");
}

}
```

Program on Nested try catch



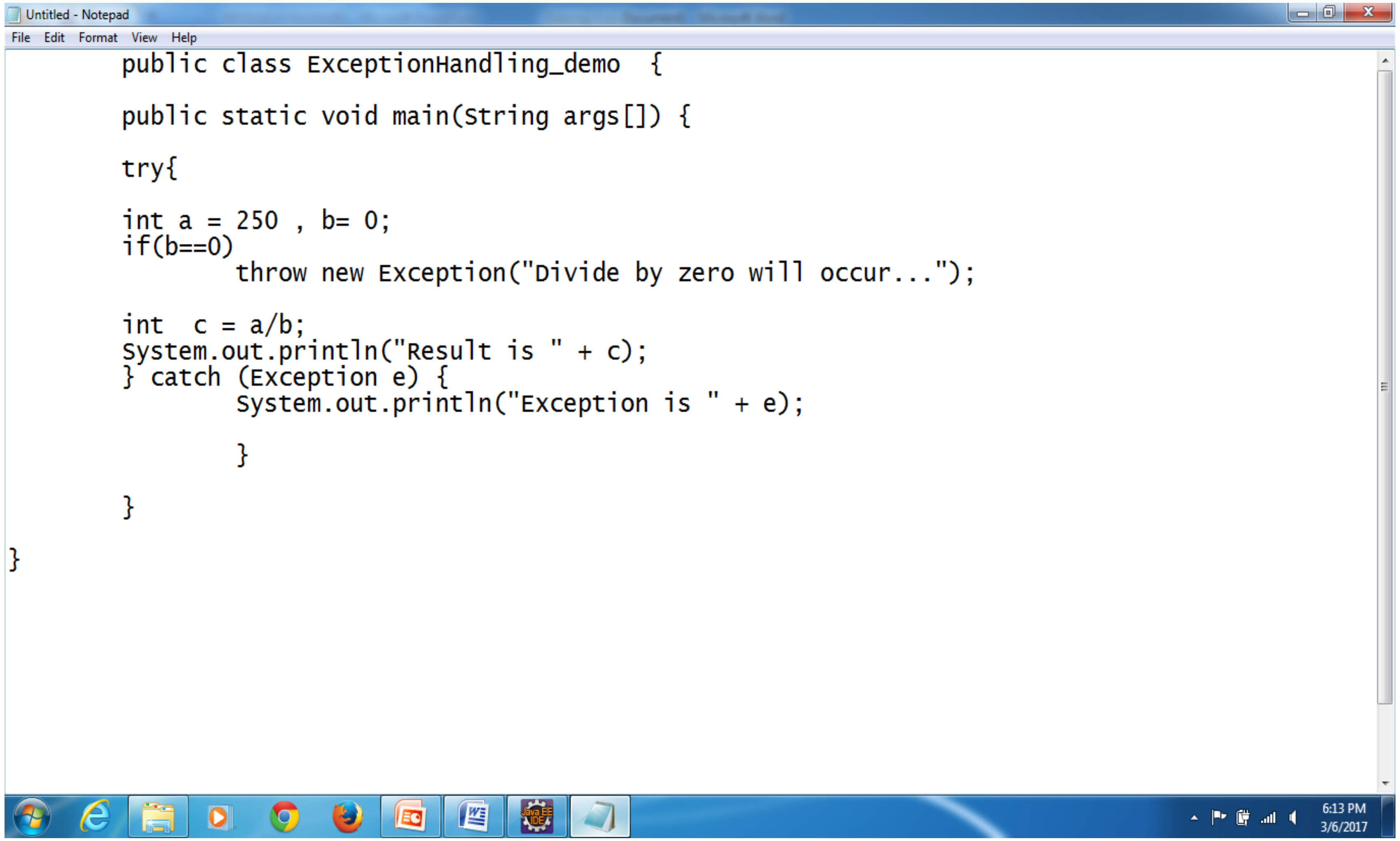
```
import java.util.Scanner;
class Excep6 {
        public satic void main(String args[])  {
         try {
                 try {
                 System.out.println("going to divide");
                 int num = 100/0 ;

         } catch (ArithmeticException e) {
         System.out.println(e);

         System.out.println("going to divide again");
         Scanner sc= new Scanner(System.in);
         System.out.println("Enter a value: ");
         // The user may enter 0, which will lead to another exception
         int num= 100 / sc.nextInt();
         }

         } catch(Exception e) {

         System.out.println (" Overall Exception Handled");

         }

         System.out.println("normal flow..");
         }

         }
```

Why throw?

Example1 : If there is a chance of a serious logic error or operational error then developer can also throw an exception .For example , if we are developing software for elections. For voting, minimum age required is 18. If the voter's age is below 18 then we can not continue any further, as the basic requirement itself is not met, hence developer can throw an exception.

Example 2 : In banking application, one user account is blocked or closed and if the bank gets the cheque to clear the amount from this account then it is not possible to continue any further hence developer can throw an exception. All the possible scenarios, developer has to use the throw keyword to throw an exception.

Throw



```
        public class ExceptionHandling_demo  {

        public static void main(String args[]) {

        try{

        int a = 250 , b= 0;
        if(b==0)
                throw new Exception("Divide by zero will occur...");

        int  c = a/b;
        System.out.println("Result is " + c);
        } catch (Exception e) {
                System.out.println("Exception is " + e);

                }

        }

}
```

Why throws?

- Design requirement: In an organization, employees provide the service. If there are any issues , in some scenarios , it is not possible for the employees to handle and it has to be escalated to the management to handle it . For example, contract signatures, handling legal issues etc.

- Similarly in Java, method which provides the service may not be required to handle certain exceptions and those exceptions should be handled by the calling function.

Throws

Throws will be used next to a function declaration statement as given below:

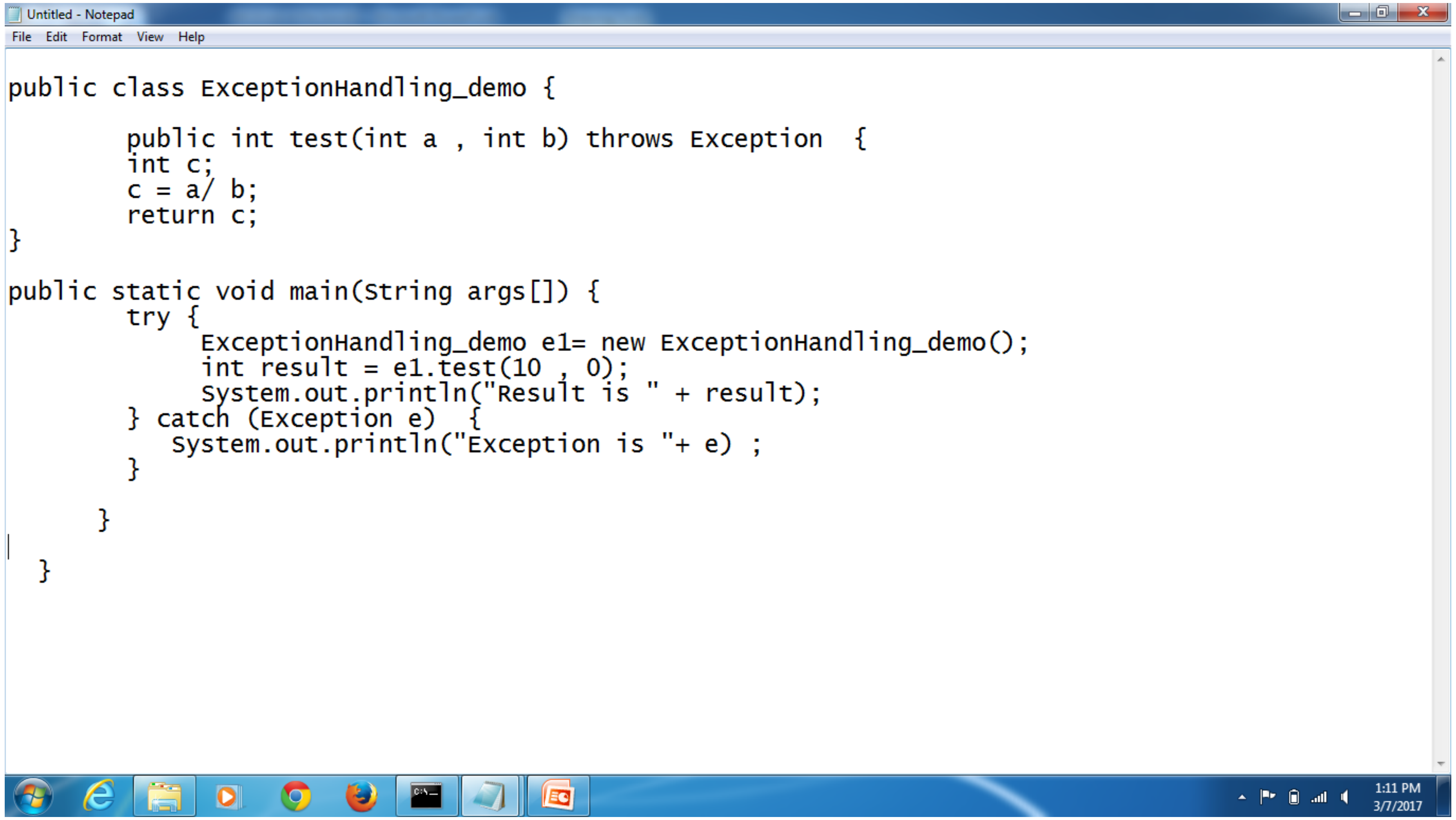
```
Public void test()throws IOException
```

This statement states that the function test() will not handle IO exception and

the calling function will handle these IOException. Calling function is responsible for IOExceptions. Many exceptions can be added by adding comma operator as given below:

```
Public void function() throws IOException,
ArrayIndexOutOfBoundsException
```

Program on Throws



```
public class ExceptionHandling_demo {

        public int test(int a , int b) throws Exception  {
        int c;
        c = a/ b;
        return c;
}

public static void main(String args[]) {
        try {
             ExceptionHandling_demo e1= new ExceptionHandling_demo();
             int result = e1.test(10 , 0);
             System.out.println("Result is " + result);
        } catch (Exception e)  {
           System.out.println("Exception is "+ e) ;
        }

      }

  }
```

Thank You!