Exception Handling

Introduction

- The **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.
- **When an exception can occur?** Exception can occur at **runtime** (known as runtime exceptions) as well as at **compile-time** (known Compile-time exceptions).

Difference between error and exception

- **Errors** indicate serious problems 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.**
- **Exceptions** are conditions within the code. A developer can handle such conditions and take necessary corrective actions. Few examples –
  - DivideByZero exception
  - NullPointerException
  - ArithmeticException
  - ArrayIndexOutOfBoundsException

Types of Exception

- There are mainly two types of exceptions:
  - Checked Exception
  - Unchecked Exception

1.Checked Exception

- **Checked exceptions**

All exceptions other than Runtime Exceptions are known as Checked exceptions as the compiler checks them during compilation to see whether the programmer has handled them or not. If these exceptions are not handled/declared in the program, it will give compilation error.

- **Examples of Checked Exceptions :-**

  ClassNotFoundException
  IllegalAccessException
  NoSuchFieldException
  EOFException etc.

- **Unchecked Exceptions**

  Runtime Exceptions are also known as Unchecked Exceptions as the compiler do not check whether the programmer has handled them or not but it’s the duty of the programmer to handle these exceptions and provide a safe exit. These exceptions need not be included in any method’s throws list because compiler does not check to see if a method handles or throws these exceptions.

- **Examples of Unchecked Exceptions:-**

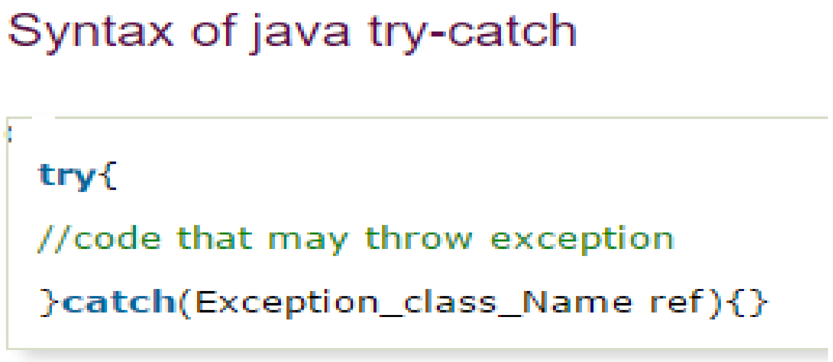
  ArithmeticException
  ArrayIndexOutOfBoundsException
  NullPointerException
  NegativeArraySizeException etc.

Java Exception Handling Keywords

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

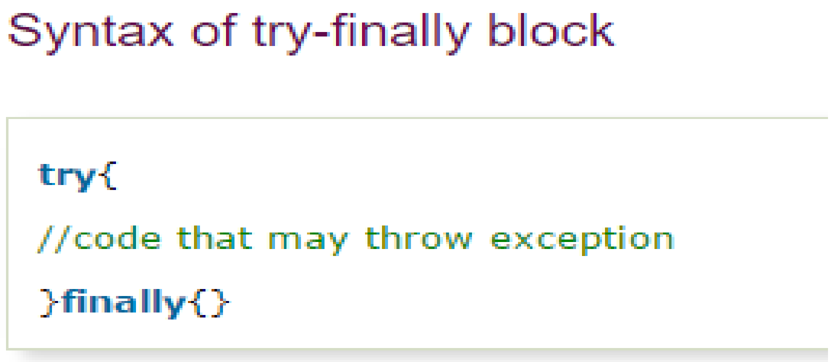
Java try block

- 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.



Syntax of java try-catch

```
try{
//code that may throw exception
}catch(Exception_class_Name ref){}
```

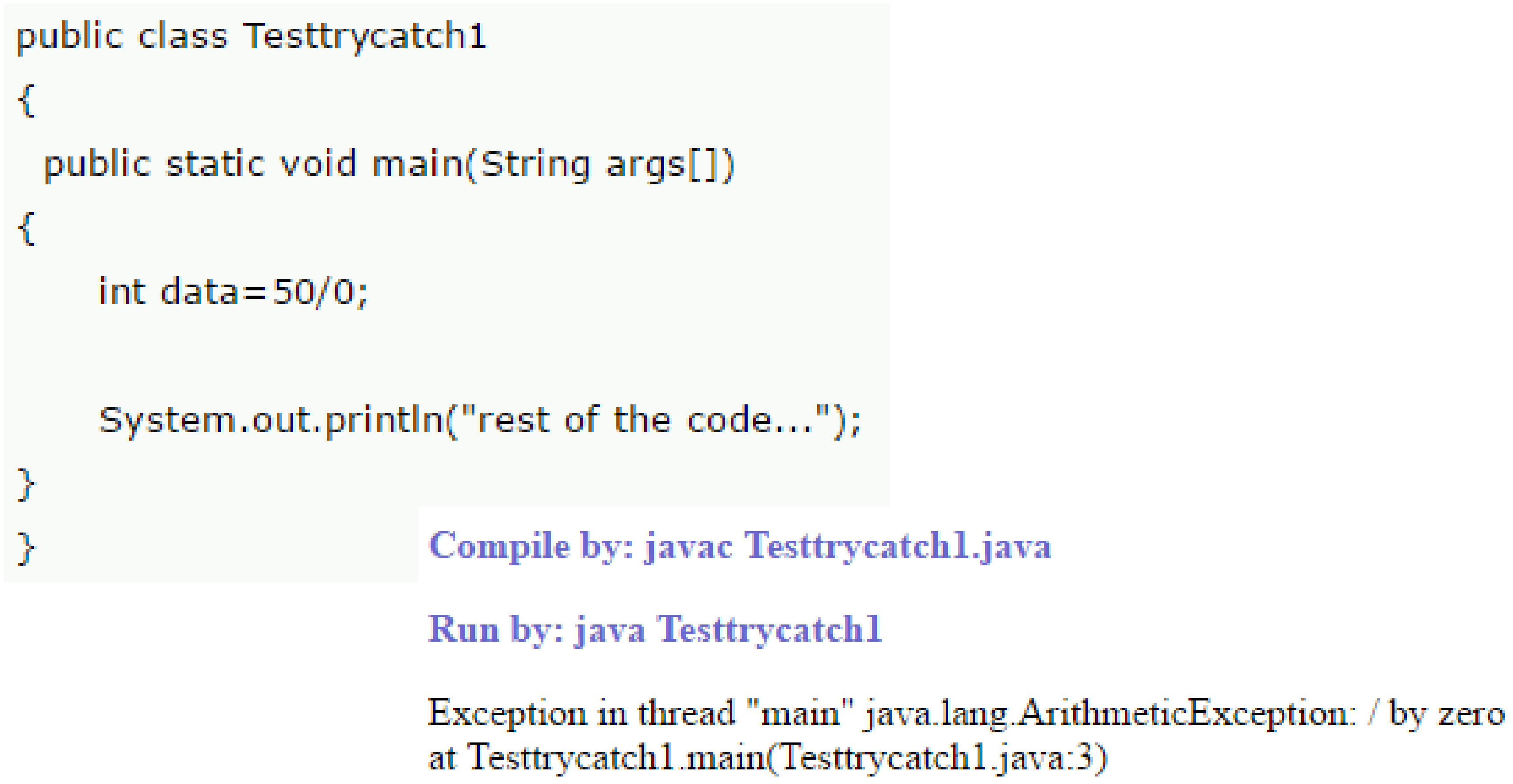


Syntax of try-finally block

```
try{
//code that may throw exception
}finally{}
```

Java catch block

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



```
public class Testtrycatch1
{
  public static void main(String args[])
{
     int data=50/0;

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

Compile by: javac Testtrycatch1.java

Run by: java Testtrycatch1

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
at Testtrycatch1.main(Testtrycatch1.java:3)
```

```
public class Testtrycatch2
{
  public static void main(String args[])
{
   try{
      int data=50/0;

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

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

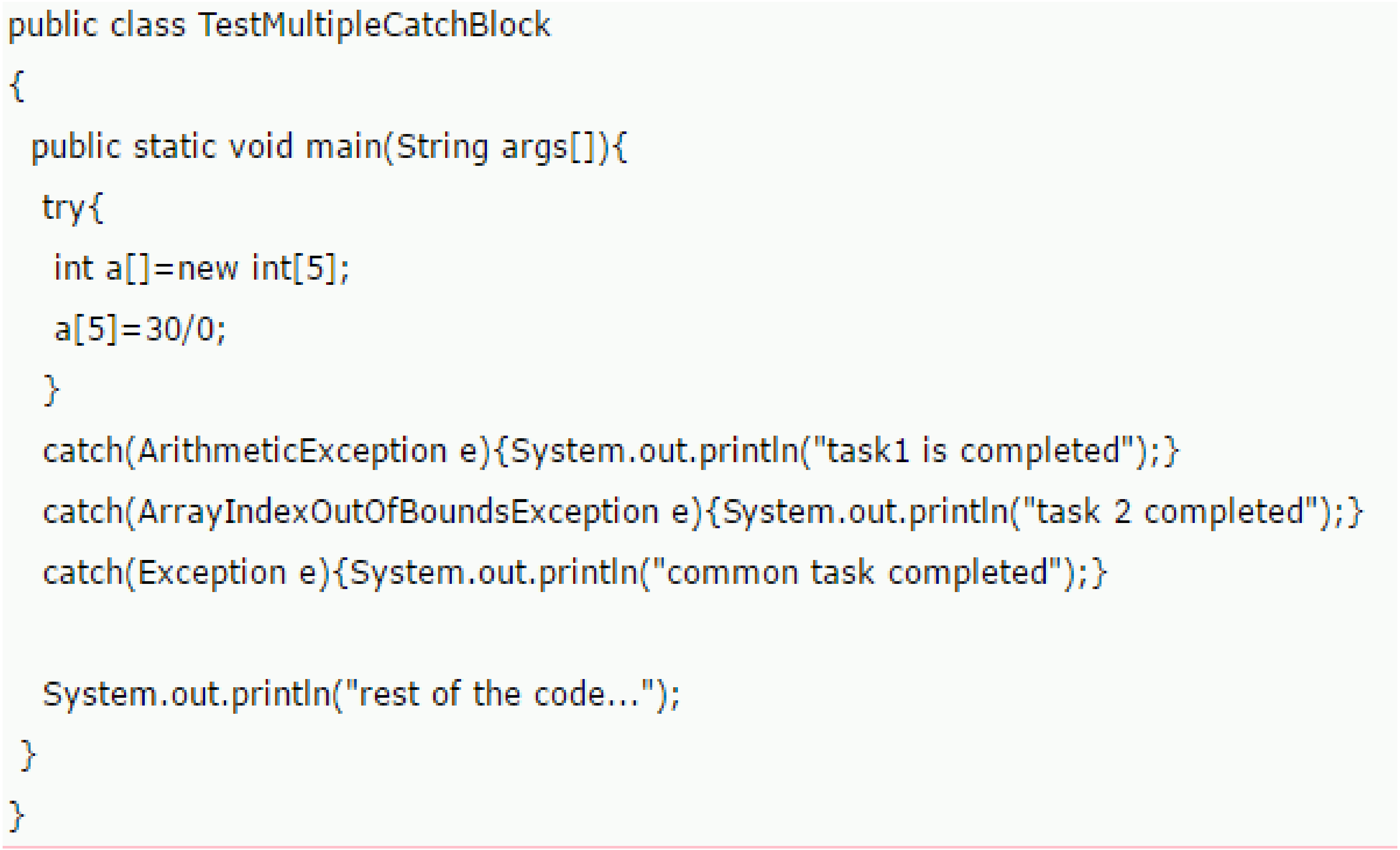
Output

Compile by: javac Testtrycatch2.java

Run by: java Testtrycatch2

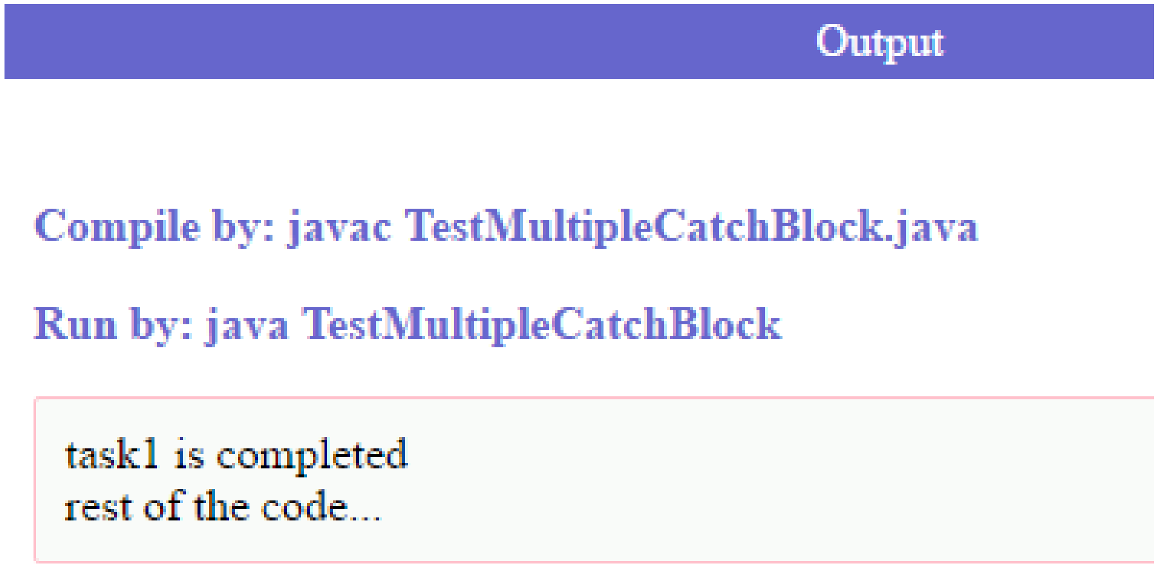
```
java.lang.ArithmeticException: / by zero
rest of the code...
```

Java Multi catch block



```
public class TestMultipleCatchBlock
{
  public static void main(String args[]){
   try{
    int a[]=new int[5];
    a[5]=30/0;
   }
   catch(ArithmeticException e){System.out.println("task1 is completed");}
   catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}
   catch(Exception e){System.out.println("common task completed");}

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



Output

Compile by: javac TestMultipleCatchBlock.java

Run by: java TestMultipleCatchBlock

```
task1 is completed
rest of the code...
```

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

Java Nested try block

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

```
....
try
{
    statement 1;
    statement 2;
    try
    {
        statement 1;
        statement 2;
    }
    catch(Exception e)
    {
    }
}
catch(Exception e)
{
}
....
```

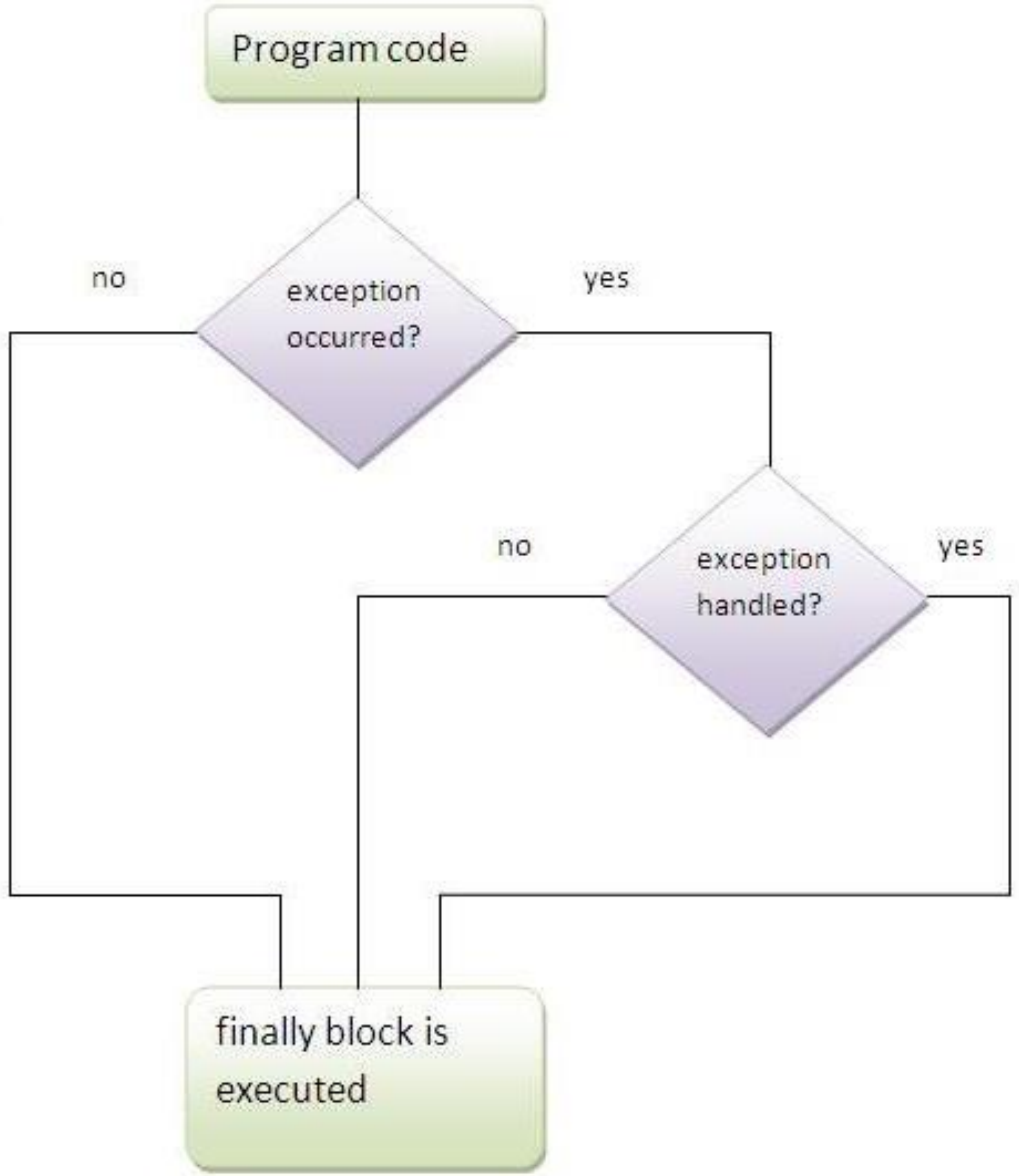
```
class Excep6{
 public static void main(String args[]){
  try{
    try{
     System.out.println("going to divide");
     int b =39/0;
    }catch(ArithmeticException e){System.out.println(e);}

    try{
    int a[]=new int[5];
    a[5]=4;
    }catch(ArrayIndexOutOfBoundsException e){System.out.println(e);}

    System.out.println("other statement);
  }catch(Exception e){System.out.println("handeled");}

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

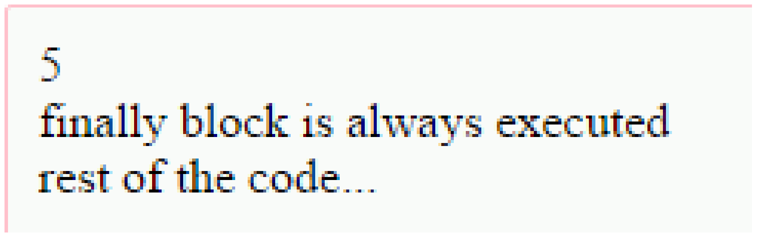
Java finally block



```
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 the code...");
  }
}
```



```
5
finally block is always executed
rest of the code...
```

java finally example where **exception occurs and not handled.**

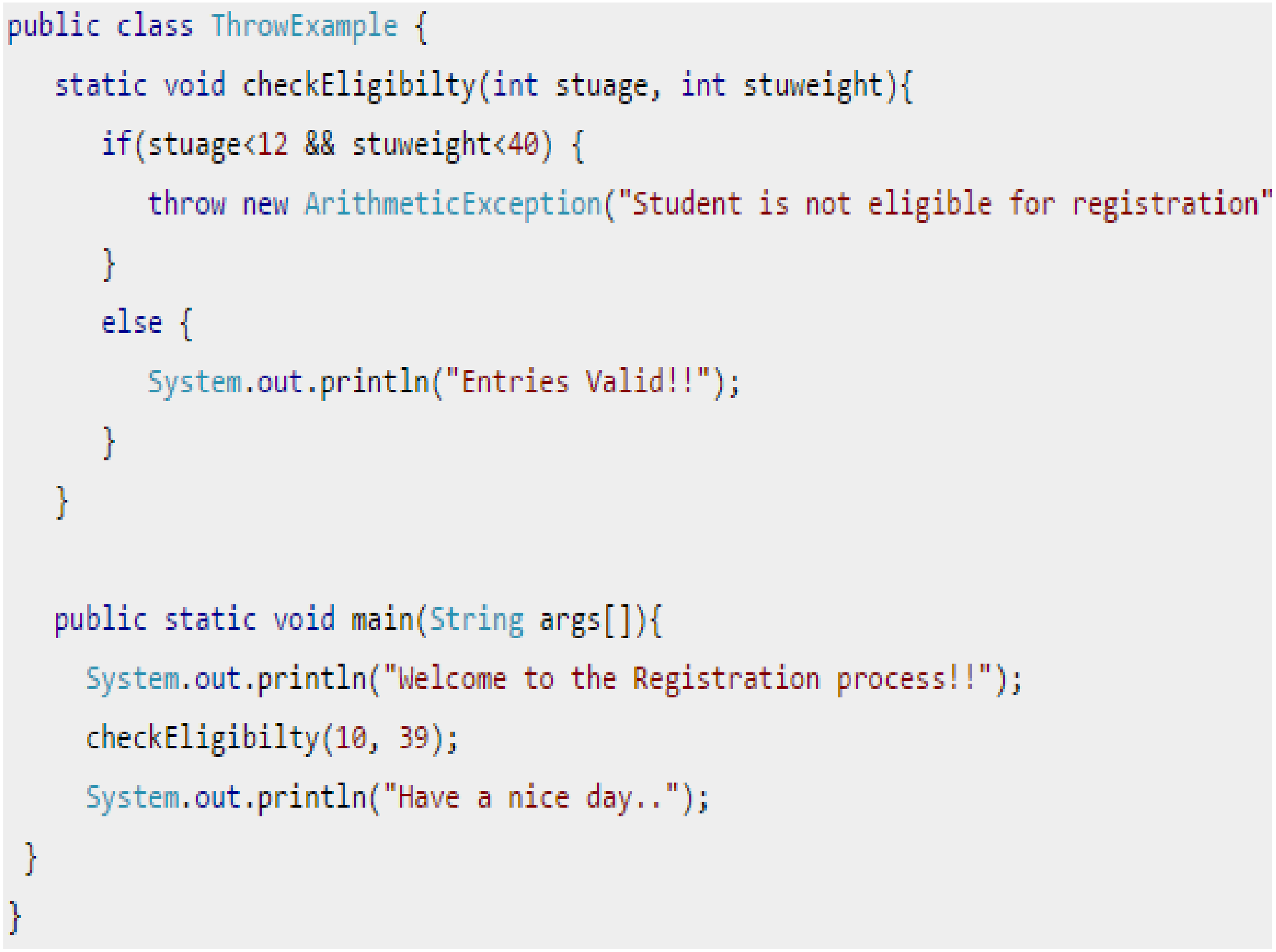
```
class TestFinallyBlock1{
  public static void main(String args[]){
  try{
   int data=25/0;
   System.out.println(data);
  }
  catch(NullPointerException e){System.out.println(e);}
  finally{System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

```
Output:finally block is always executed
       Exception in thread main java.lang.ArithmeticException:/ by zero
```

java finally example where **exception occurs and handled.**

```
public class TestFinallyBlock2{
  public static void main(String args[]){
  try{
   int data=25/0;
   System.out.println(data);
  }
  catch(ArithmeticException e){System.out.println(e);}
  finally{System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

```
Output:Exception in thread main java.lang.ArithmeticException:/ by zero
       finally block is always executed
       rest of the code...
```



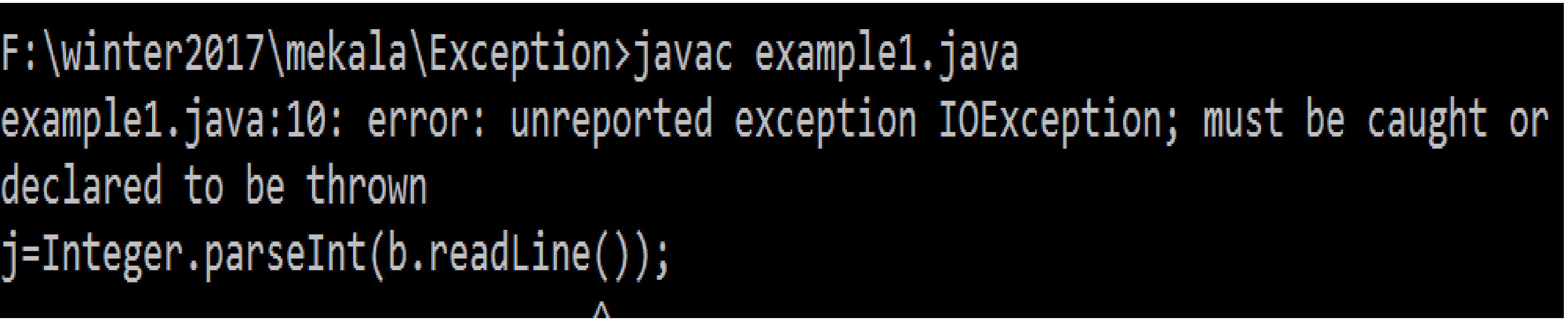
```java
public class ThrowExample {
   static void checkEligibilty(int stuage, int stuweight){
      if(stuage<12 && stuweight<40) {
         throw new ArithmeticException("Student is not eligible for registration"
      }
      else {
         System.out.println("Entries Valid!!");
      }
   }

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

Throws Keyword

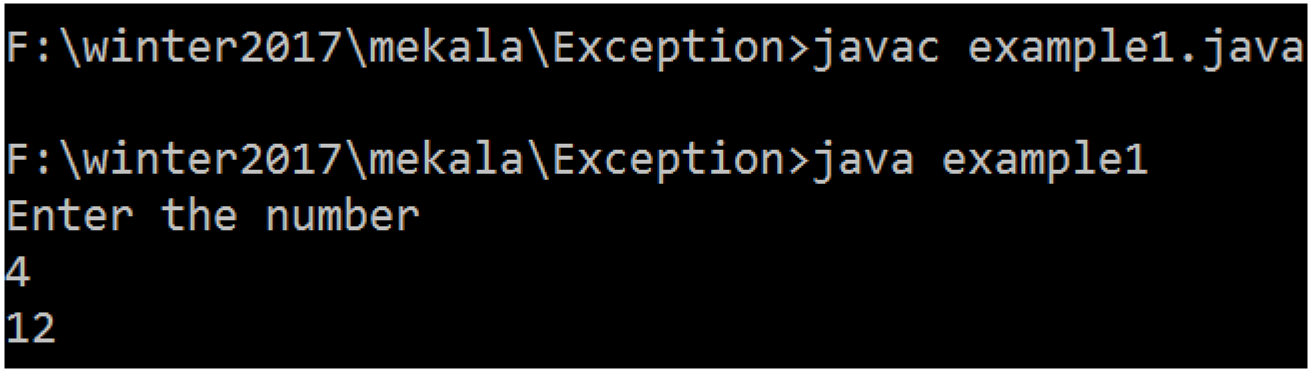
- throws is a keyword in Java which is used in the signature of method to indicate that this method might throw one of the listed type exceptions.
- The caller to these methods has to handle the exception using a try-catch block.

```java
import java.io.*;
class example1
{
public static void main(String args[])
{
int i, j,k=0;
BufferedReader b=new BufferedReader(new InputStreamReader(System.in));
i=8;
System.out.println("Enter the number");
j=Integer.parseInt(b.readLine());
k=i+j;
System.out.println(k);
}
}
```



```
F:\winter2017\mekala\Exception>javac example1.java
example1.java:10: error: unreported exception IOException; must be caught or
declared to be thrown
j=Integer.parseInt(b.readLine());
                              ^
```

```java
import java.io.*;
class example1
{
public static void main(String args[])throws Exception
{
int i, j,k=0;
BufferedReader b=new BufferedReader(new InputStreamReader(System.in));
i=8;
System.out.println("Enter the number");
j=Integer.parseInt(b.readLine());
k=i+j;
System.out.println(k);
}
}
```



```
F:\winter2017\mekala\Exception>javac example1.java

F:\winter2017\mekala\Exception>java example1
Enter the number
4
12
```

- Throws keyword doesnot handle the error but it supress the error

```java
// Java program to demonstrate working of throws
class ThrowsExecp
{
    static void fun() throws IllegalAccessException
    {
        System.out.println("Inside fun(). ");
        throw new IllegalAccessException("demo");
    }
    public static void main(String args[])
    {
        try
        {
            fun();
        }
        catch(IllegalAccessException e)
        {
            System.out.println("caught in main.");
        }
    }
}
```

```java
import java.io.*;
public class ThrowExample {
  void mymethod(int num)throws IOException, ClassNotFoundException{
     if(num==1)
        throw new IOException("Exception Message1");
     else
        throw new ClassNotFoundException("Exception Message2");
  }
}

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

User defined exception

- **User defined exceptions in java** are also known as **Custom exceptions**.
- Most of the times when we are developing an application in java, we often feel a need to create and throw **our own exceptions**.
- These exceptions are known as **User defined or Custom exceptions.**

User defined Exception

- User defined exception needs to **inherit (extends)** Exception class in order to **act as an exception**.
- throw keyword is used to throw such exceptions.

```
import java.lang.Exception;
class MyException extends Exception{
        MyException(String message)
        {     super(message);     }
}

class TestMyException{
public static void main(String arg[]){
        int x=5,y=1000;
        try
        {
          float z=(float)x/(float)y;
          if(z<0.01){
                throw new MyException("Number is too small");
          }
        }catch(MyException e){
            System.out.println("Caught My Exception");
            System.out.println(e);
}        finally{
           System.out.println("Always executed");
        }}}
```