Exception Handling

What is Exception?

Exception Handling

An exception can occur for following reasons.

- User error
- Programmer error
- Physical error

```java
import java.util.*;
public class MyClass
{
     public static void main(String args[])
    {
        int x=10;
        int y=0;
        int z=x/y;
        System.out.println(z);
     }
}
```

Output

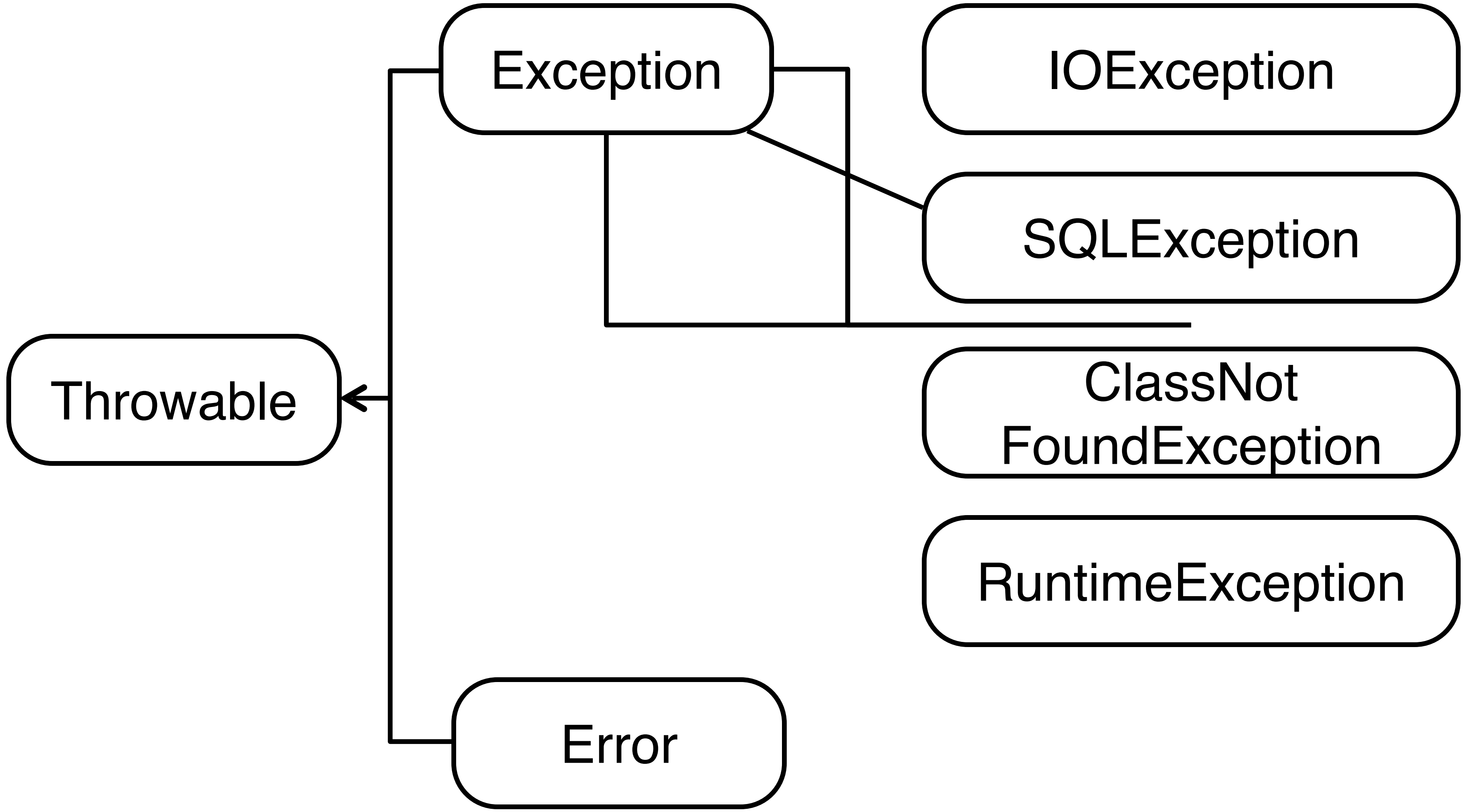
```
Exception in thread "main" java.lang.ArithmeticException: / by zero     at
MyClass.main(MyClass.java:6)
```

```java
1  import java.io.*;
2  public class Main {
3      public static void main(String[] args){
4          System.out.println("First line");
5          System.out.println("Second line");
6          int[] myIntArray = new int[]{1, 2, 3};
7          print(myIntArray);
8          System.out.println("Third line");
9      }
10     public static void print(int[] arr) {
11         System.out.println(arr[3]);
12         System.out.println("Fourth element successfully displayed!");
13     }
14 }
15
```

Output

```
First line
Second line
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
      at Main.print(Main.java:11)
      at Main.main(Main.java:7)
```

Hierarchy of Java Exception classes



Types of Exception

- Checked Exception
- Unchecked Exception

```java
//Example for Checked exception
import java.io.FileInputStream;
public class Main {
	public static void main(String[] args) {
		FileInputStream fis = null;
		fis = new FileInputStream("D:/myfile.txt");
		int k;
		while(( k = fis.read() ) != -1)
		{
			System.out.print((char)k);
		}
		fis.close();
	}
}
```

Output

```
Exception in thread "main" java.lang.Error: Unresolved compilation
problems:
Unhandled exception type FileNotFoundException
Unhandled exception type IOException
Unhandled exception type IOException
```

```java
//Example for Unchecked exception
import java.io.*;
public class Main {
    public static void main(String[] args){
        int[] arr = new int[]{7, 11, 30, 63};
        System.out.println(arr[5]);
    }
}
```

Output

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
at Main.main(Main.java:5)
```

Exception Handling

Exception Handling?

It is the process of responding to the occurrence during computation of exceptions.

Exceptional Handling Keywords

- try
- catch
- finally
- throw
- throws

```java
import java.util.*;
public class MyClass
{
    public static void main(String
                    args[])
   {
      int x=10;
      int y=0;
      int z=x/y;
      System.out.println(z);
    }
}
```

```java
import java.util.*;
public class MyClass
{
    public static void main(String
                    args[])
 {
      int x=10;
      int y=0;
      try
      {
        int z=x/y;
      }
      catch(Exception e)
      {
          System.out.print(e);
      }
   }
}
```

```java
import java.util.*;
public class MyClass
{
    public static void main(String
                  args[])
 {
        int x=10;
        int y=0;
        try
        {
          int z=x/y;
        }
        catch(Exception e)
        {
            System.out.print(e);
        }
     }
}
```

comment/pseudo code/output

Output:

```
java.lang.ArithmeticException: / by zero
```

Syntax:

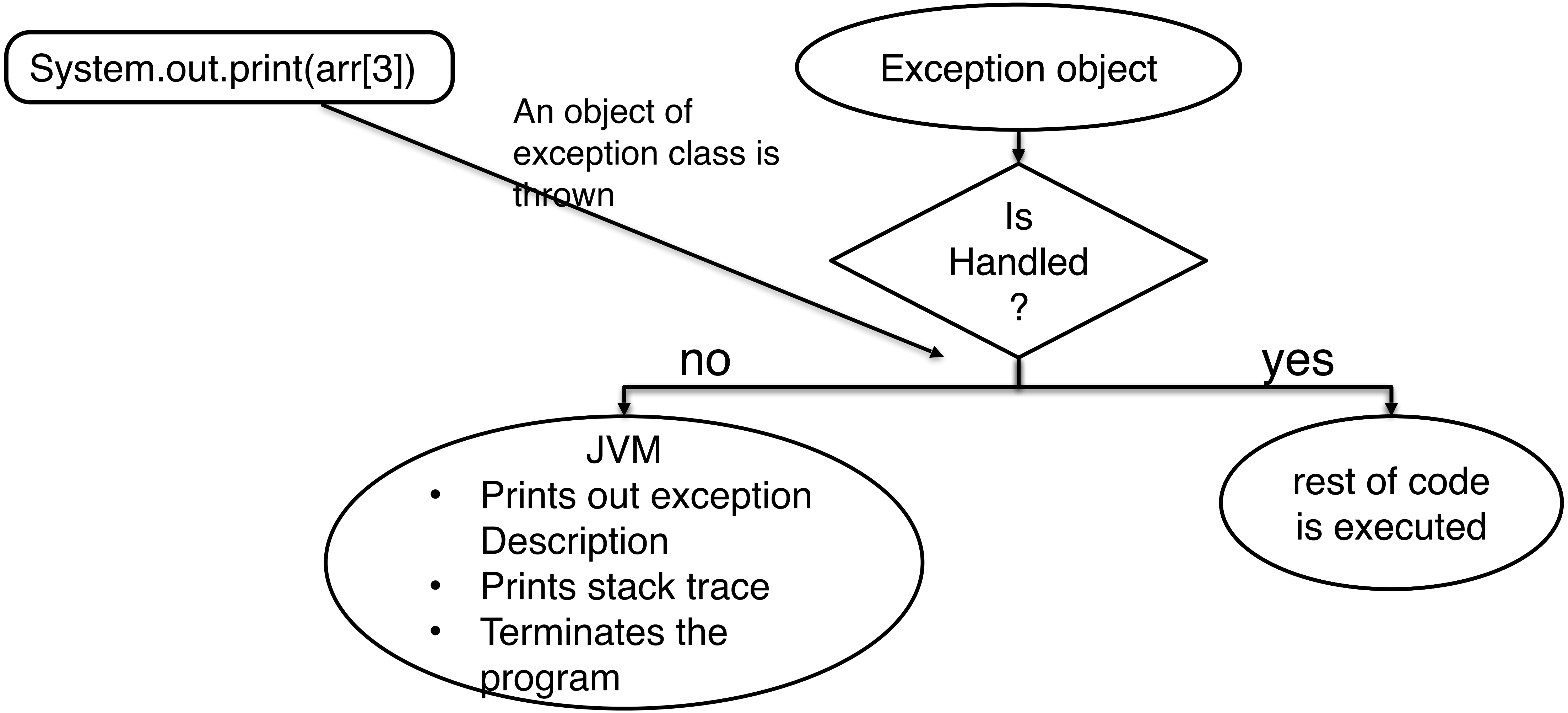
```
try
{
   // Protected code
}
catch (ExceptionName e1)
{
   // Catch block
}
```

```java
import java.io.*;
public class Main {
    public static void main(String[] args){
        System.out.println("First line");
        System.out.println("Second line");
        try{
            int[] myIntArray = new int[]{1, 2, 3};
            print(myIntArray);
        }
        catch (ArrayIndexOutOfBoundsException e){
              System.out.println("The array doesn't have fourth element!");
        }
        System.out.println("Third line");
    }
    public static void print(int[] arr) {
        System.out.println(arr[3]);
    }
}
```

OUTPUT

```
First line
Second line
The array doesn't have fourth element!
Third line
```

Internal Working of try-catch Block



```java
//Predict the Output
import java.util.*;
public class MyClass
{
	public static void main(String args[])
	{
		MyClass ob = new MyClass();
		try
		{
			ob.meth1();
		}
		catch(ArithmeticException e)
		{
			System.out.println("ArithmaticException thrown by meth1()
					method is caught in main() method");
		}
	}
```

```java
          public void meth1()
          {
            try
            {
                System.out.println(100/0);
            }
            catch(NullPointerException nullExp)
            {
                System.out.println("We have an Exception - "+nullExp);
            }
            System.out.println("Outside try-catch block");
          }
    }
```

```java
import java.util.*;
public class MyClass {
    public static void main(String[] args) {
        try{
            int arr[]=new int[5];
            arr[7]=100/0;
        }
        catch(ArithmeticException e)
        {
           System.out.println("Arithmetic Exception occurs");
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
           System.out.println("ArrayIndexOutOfBounds Exception occurs");
        }
        catch(Exception e)
        {
           System.out.println("Parent Exception occurs");
        }
        System.out.println("rest of the code");
    }
}
```

```
//Predict the Output
import java.util.*;
public class MyClass
{
    public static void main(String[] args)
    {
        String[] s = {"Hello", "423", null, "Hi"};
        for (int i = 0; i < 5; i++)
        {
            try
            {
                int a = s[i].length() + Integer.parseInt(s[i]);
            }
            catch(NumberFormatException ex)
            {
                System.out.println("NumberFormatException");
            }
```

```java
                catch (ArrayIndexOutOfBoundsException ex)
                {
                    System.out.println("ArrayIndexOutOfBoundsException");
                }
                catch (NullPointerException ex)
                {
                    System.out.println("NullPointerException");
                }
                System.out.println("After catch, this statement will be
                                                            executed");
            }
        }
    }
```

```java
// Pipe(|) operator
import java.util.*;
public class Main
{
    public static void main(String[] args)
    {
        String[] s = {"abc", "123", null, "xyz"};
        for (int i = 0; i < 5; i++)
        {
            try
            {
                int a = s[i].length() + Integer.parseInt(s[i]);
            }
            catch(NumberFormatException | NullPointerException |
                                ArrayIndexOutOfBoundsException ex)
            {
                System.out.println("Handles above mentioned three
                                                    Exception");
            }
        }
    }
}
```

```java
//Predict the output
import java.util.*;
public class Main
{
    public static void main(String[] args)
    {
        try
        {
            int i = Integer.parseInt("Thor");
        }
        catch(Exception ex)
        {
            System.out.println("This block handles all exception types");
        }
        catch(NumberFormatException ex)
        {
            System.out.print("This block handles NumberFormatException");
        }
    }
}


```

```java
//Predict the OUtput
import java.util.*;
public class Main
{
    public static void main(String[] args)
    {
        String[] str = {null, "Marvel"};
        for (int i = 0; i < str.length; i++)
        {
            try
            {
                int a = str[i].length();
                try
                {
                    a = Integer.parseInt(str[i]);
                }
                catch (NumberFormatException ex)
                {
                    System.out.println("NumberFormatException");
                }
            }

```

```
                catch(NullPointerException ex)
                {
                    System.out.println("NullPointerException");
                }
            }
        }
    }
```

```
//Syntax for finally block

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

Description

It contains all the crucial statements that must be executed whether exception occurs or not.

```java
public class MyClass{
  public static void main(String args[]){
    try{
            int data = 30/3;
            System.out.println(data);
        }
    catch(NullPointerException e)
    {
        System.out.println(e);
    }
    finally
    {
      System.out.println("finally block is always executed");
    }
  }
}
```

```java
public class MyClass
{
   public static void main(String args[]) {
      try{
            int num=121/0;
            System.out.println(num);
      }
      catch(ArithmeticException e){
         System.out.println("Number should not be divided by zero");
      }
      finally{
          System.out.println("This is finally block");
      }
      System.out.println("Out of try-catch-finally");
   }
}
```

```java
public class MyClass
{
   public static void main(String args[])
   {
      System.out.println(MyClass.myMethod());
   }
   public static int myMethod()
   {
      try
      {
        return 0;
      }
      finally
      {
        System.out.println("This is Finally block");
        System.out.println("Finally block ran even after return
                                                 statement");
      }
   }
}
```

```java
public class MyClass
{
   public static void main(String args[])
   {
      System.out.println(MyClass.myMethod());
   }
   public static int myMethod()
   {
      try
      {
        return 0;
      }
      finally
      {
        System.out.println("This is Finally block");
        System.out.println("Finally block ran even after return
                                                         statement");
      }
   }
}
```

Cases when the finally block doesn’t execute

- The death of a Thread.
- Using of the System. exit() method.
- Due to an exception arising in the finally block.

```java
public class MyClass{
   public static void main(String args[])
   {
      System.out.println(MyClass.myMethod());
   }
   public static int myMethod(){
       try {
                int x = 63;
                int y = 9;
                int z=x/y;
                System.out.println("Inside try block");
                ~~System.exit(0);~~
        }
        catch (Exception exp){
            System.out.println(exp);
        }
        finally{
            System.out.println("Java finally block");
            return 0;
        }
   }
}
```

```java
public class MyClass{
   public static void main(String args[])
   {
      System.out.println(MyClass.myMethod());
   }
   public static int myMethod(){
       try {
               int x = 63;
               int y = 0;
               int z=x/y;
               System.out.println("Inside try block");
               System.exit(0);
        }
        catch (Exception exp){
            System.out.println(exp);
        }
        finally{
            System.out.println("Java finally block");
            return 0;
        }
   }
}
```

```
//Syntax for throw block


throw new exception_class("error message");
```

Description

The Java throw keyword is used to explicitly throw an exception.

```
public class MyClass
{
   public static void validate(int age)
   {
     if(age<21 || age>27)
      throw new ArithmeticException("not eligible");
     else
      System.out.println("Eligible to attend Military Selection ");
   }
   public static void main(String args[])
   {
      validate(30);
      System.out.println("rest of the code...");
    }
}
```

OUTPUT

```
Exception in thread "main" java.lang.ArithmeticException: not eligible
at MyClass.validate(MyClass.java:6)
at MyClass.main(MyClass.java:12)
```

```
public class MyClass
{
    public static void validate(int age)
    {
      if(age<21 || age>27)
       throw new ArithmeticException("not eligible");
      else
       System.out.println("Eligible to attend Military Selection ");
    }
    public static void main(String args[])
    {
        try
        {
         validate(30);
        }
        catch(ArithmeticException e)
        {
            System.out.println(e);
        }
       System.out.println("rest of the code...");
     }
}
```

OUTPUT

```
java.lang.ArithmeticException: not eligible
rest of the code...
```

```java
public class MyClass
{
   public static void validate(int age)
   {
     if(age<21 || age>27)
      throw new ArithmeticException("not eligible");
     else
      System.out.println("Eligible to attend Military Selection ");
   }
   public static void main(String args[])
   {
       try
       {
        validate(21);
       }
       catch(ArithmeticException e)
       {
           System.out.println(e);
       }
      System.out.println("rest of the code...");
    }
}
```

OUTPUT

```
Eligible to attend Military Selection
rest of the code...
```

```
//Syntax for throw block


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

Description

The Java throw keyword is used to explicitly throw an exception.

```java
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
public class Main {
      public static void main(String[] args) {
            Read r = new Read();
            try {
                  r.read();
            }
            catch (Exception e) {
                  System.out.print(e);
            }
      }
}
public class Read {
      void read() throws FileNotFoundException
      {
                  BufferedReader bufferedReader = new BufferedReader(new
                                    FileReader("F://file.txt"));
      }
}
```

OUTPUT

```
java.io.FileNotFoundException: F:\file.txt (The system cannot find the
file specified)
```

```java
import java.io.*;
class MyMethod {
  void myMethod(int num)throws IOException, ClassNotFoundException{
      if(num==1)
         throw new IOException("IOException Occurred");
      else
         throw new ClassNotFoundException("ClassNotFoundException");
  }
}
public class MyClass{
  public static void main(String args[]){
   try{
        MyMethod obj=new MyMethod();
        obj.myMethod(1);
    }
    catch(Exception ex){
     System.out.println(ex);
    }
  }
}
```

OUTPUT

```
java.io.FileNotFoundException: F:\file.txt (The system cannot find the
file specified)
```

Throw vs. Throws

throw	throws
1. Used to explicitly throw an exception	1. Used to declare an exception
2. Checked exceptions cannot be propagated using throw only	2. Checked exceptions can be propagated
3. Followed by an instance	3. Followed by a class
4. Used within a method	4. Used with a method signature
5. Cannot throw multiple exceptions	5. Can declare multiple exceptions

Programming

Question 1

Write a program to split the string based on /(slash) symbol in a string. Perform exception handling and also include finally block which will print "Inside finally block".

Get a string from the user.

If the length of the string is > 2 then call user defined function splitString() and print the split string along with index.

Else print the NullPointerException.

Question 1

Sample Input:

```
Happy/coding/Java
```

Sample Output:

```
Splitted string at index 0 is : Happy
Splitted string at index 1 is : coding
Splitted string at index 2 is : Java
Inside finally block
```

```java
import java.util.*;
public class Main
{
    static void splitString(String text)
    {
        try
        {
            String[] splittedString =text.split("/");
            for(int i = 0; i < splittedString.length; i++)
            {
                System.out.println("Splitted string at index "+i+" is :
                                               "+splittedString[i]);
            }
        }
        catch(Exception e)
        {
            System.out.println("Exception : "+e.toString());
        }
        finally{
            System.out.println("Inside finally block");
        }
    }
    public static void main(String args[])
    {
        Scanner scanner = new Scanner(System.in);
        String text = scanner.nextLine();
        if(text.length()>2)
        {
            splitString(text);
        }
            else
        {
            splitString(null);
        }
    }
}
```

Question 2

You are required to compute the power of a number by implementing a calculator. Create a class MyCalculator which consists of a single method long power(int, int). This method takes two integers, n and p, as parameters and finds n^p. If either n or p is negative, then the method must throw an exception which says “n or p should not be negative.". Also, if both n and p are zero, then the method must throw an exception which says “n and p should not be zero."

For example, -4 and -5 would result in java.lang.Exception: n or p should not be negative

Complete the function power in class MyCalculator and return the appropriate result after the power operation or an appropriate exception as detailed above.

Question 2

Sample Input:

```
3 5
2 4
0 0
-1 -2
-1 3
```

Sample Output:

```
243
16
java.lang.Exception: n and p
should not be zero.
java.lang.Exception: n or p should
not be negative.
java.lang.Exception: n or p should
not be negative.
```

```java
import java.util.Scanner;
class MyCalculator {
    public static int power(int n,int p) throws Exception
    {
        if(n<0 || p<0)
        {
           throw new Exception ("n or p should not be negative.");
        }
        else if(n==0 && p==0)
        {
          throw new Exception ("n and p should not be zero.");
        }
        else
        {
          return((int)Math.pow(n,p));
        }
    }
}
```

```java
public class Solution {
    public static final MyCalculator my_calculator = new MyCalculator();
    public static final Scanner in = new Scanner(System.in);
    public static void main(String[] args) {
        while (in .hasNextInt()) {
            int n = in .nextInt();
            int p = in .nextInt();

            try {
                System.out.println(my_calculator.power(n, p));
            } catch (Exception e) {
                System.out.println(e);
            }
        }
    }
}
```

MCQ

```java
// Predict the output
public class Test{
    public static void main(String args[]){
        try{
                int i;
                return;
            }
        catch(Exception e){
            System.out.print("CatchBlock");
        }
        finally{
                System.out.println("FinallyBlock");
        }
    }
}
```

Question 1

A) CatchBlock

B) CatchBlock FinallyBlock

[✓ **C)** FinallyBlock]

D) No Output

```java
// Predict the output
class Bike{
    public void petrol() {}
}
public class Test{
    public static void main(String args[]){
        Bike fz = null;
        try{
                fz.petrol();
        }
        catch(NullPointerException e){
                System.out.print("There is a NullPointerException. ");
        }
        catch(Exception e){
                System.out.print("There is an Exception. ");
        }
        System.out.print("Everything went fine. ");
    }
}
```

Question 2

A) There is a NullPointerException. Everything went fine. ✓

B) There is a NullPointerException.

C) There is a NullPointerException. There is an Exception.

D) This code will not compile, because in Java there are no pointers.

```java
// Predict the output
public class MyClass{
      public static void main(String args[]){
            try{
                  String arr[] = new String[10];
                  arr = null;
                  arr[0] = "one";
                  System.out.print(arr[0]);
            }
            catch(Exception ex)
            {
                  System.out.print("exception");
            }
            catch(NullPointerException nex)
            {
                  System.out.print("null pointer exception");
            }
      }
}
```

Question 3

A) "one" is printed.

B) "null pointer exception" is printed.

[✓ **C)** Compilation fails saying NullPointerException has already been caught.]

D) "exception" is printed.

```java
// Predict the output
import java.lang.Exception;
public class MyClass{
    void m() throws Exception {
        throw new java.io.IOException("Error");
    }
    void n() {
        m();
    }
    void p() {
        try {
            n();
        } catch (Exception e) {
            System.out.println("Exception Handled");
        }
    }
    public static void main(String args[]){
        MyClass obj = new MyClass();
        obj.p();
        System.out.println("Normal Flow");
    }
}
```

Question 4

A) Error

B) Error
Exception Handled
Normal Flow

[✓ **C)** Compilation Error]

D) No Output

```java
// Predict the output
public class MyClass {
    public static void main(String[] args) {
        for (int i = 0; i < 3; i++)
        {
            try
            {
                execute(i);

            }
            catch (Exception e) {
                }
            System.out.print("-");
        }
    }
    public static void execute(int i) {
     p('S');
     try {
         p('H');
         t(i == 1);
         try{
             p('A');
             t(i == 3);
         }
         finally {
             p('R');
         }
     }
     catch (Exception e) {
         p('K');
         t(i == 3);
     }
}
```

```
    public static void p(char c)
    {
        System.out.print(c);
    }
    public static void t(boolean thrw)
    {
        if (thrw)throw new RuntimeException();
    }
}
```

Question 5

A) SHARK-SH-SHK-

B) [✓ SHAR-SHK-SHAR-]

C) Compilation Error

D) SHARK-SHARK-SHARK-

THANK YOU