

UNIT-3

Exception handling

Exception:

An abnormal event or situation which stops normal flow of execution of a program or task

An exception is an unexpected event that occurred during the execution of a program, and disrupts the normal flow of instructions.”

An exception (or exceptional event) is a problem that arises during the execution of a program.

Few real time examples

- Pen not functioning in the middle of the exam
- Running out of petrol in the middle of a journey
- running out of gas in the middle of cooking
- Entering restricted characters in e-mail address
- The amount passed to withdraw exceeds the account's balance.
- Power cut in the middle of using word document
- You issue a command to read a file from a disk, but the file does not exist there.
- You attempt to write data to a disk, but the disk is full
- - A file that needs to be opened cannot be found.
 - A network connection has been lost in the middle of communications

note:

- These errors are called exceptions because, presumably, they are not usual occurrences; they are “exceptional.
- Exception means special case

Programming example

1.

```
class array
{
    public static void main(String svec[])
    {
        int a[]={2,3,4,5,6};
        System.out.println(a[5]);
        System.out.println("I am an IT student");
        System.out.println("bye");
    }
}
```

Explanation:

- In above code, we tried to refer 'a[5]' which doesn't exist. It is called "ArrayIndexOutOfBoundsException" in java.
- Execution can't be proceeded further till the end so Execution gets terminated in the middle.

2.

class divide

```
{
    public static void main(String svec[])
    {
        int result;
        result=88/0;
        System.out.println(result);
        System.out.println("I am an IT student");
        System.out.println("bye");
    }
}
```

Explanation:

- In above code, we tried to divide 88 with 0 which is not valid. It is called "ArithmeticException" in java.
- Execution can't be proceeded further till the end so Execution gets terminated in the middle

Handling exceptions

- To detect that an exception has been thrown and prevent it from halting your application, Java allows you to create exception handlers.
- An *exception handler* is a section of code that gracefully responds to exceptions when they are thrown.
- The process of intercepting and responding to exceptions is called *exception handling*.

To handle an exception, you use a *try* and *catch* blocks.

```
try
{
    // here place the code which may raise an exception
}
catch (Exception class    exception object)
{
    //here place the code which handles raised exception
}
```

First the key word *try* appears. Next, a block of code appears inside braces, which are required.

This block of code is known as a *try block*. A *try block* is one or more statements that are executed and can potentially throw an exception.

Example code 1:

```
class divide
{
    public static void main(String svec[])
    {
        int result;
        try
        {
            result=88/0;
            System.out.println(result);
            System.out.println("NO EXCEPTION RAISED");
        }
        catch(ArithmeticException praneeth)
        {
            System.out.println("\nHi friend, an exception
raised :");
            System.out.println("\nI cant proceed
further.....");
        }
    }
}
```

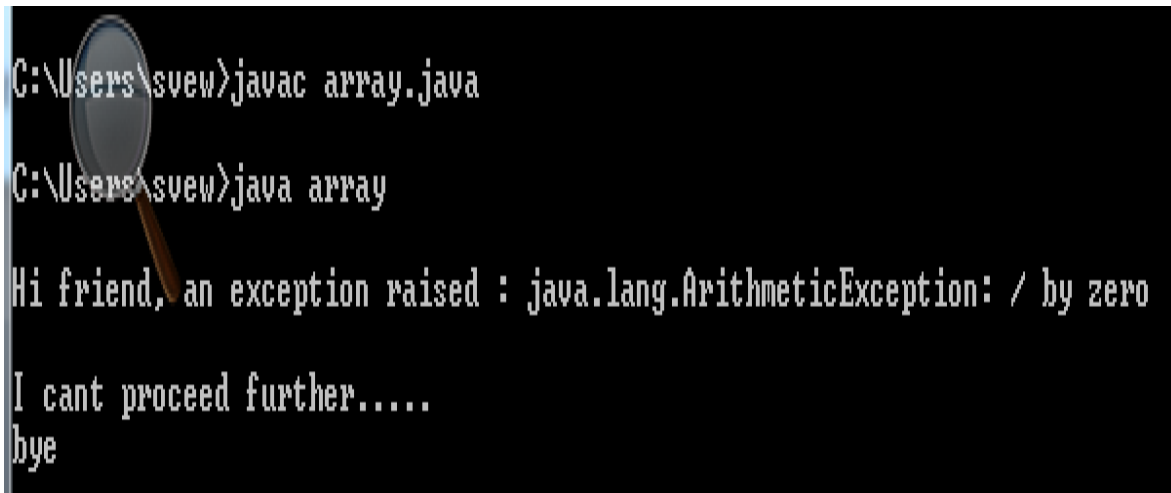
```

    }
    System.out.println("bye");
}

}

```

OUTPUT: When you divide with 0

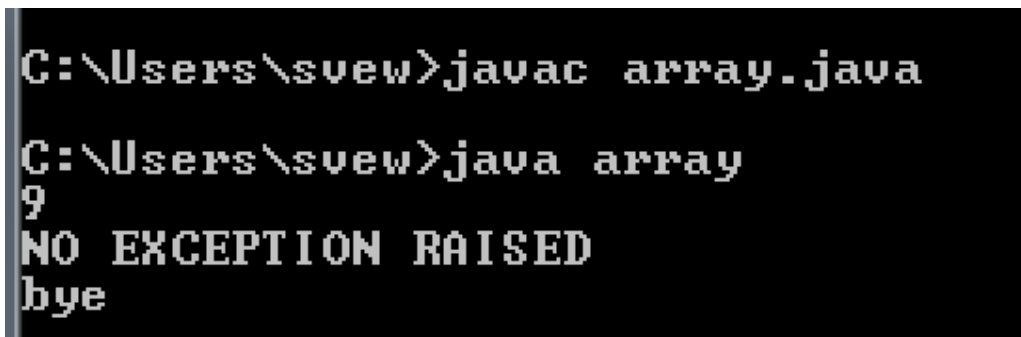


```

C:\Users\sveu>javac array.java
C:\Users\sveu>java array
Hi friend, an exception raised : java.lang.ArithmeticException: / by zero
I cant proceed further.....
bye

```

OUTPUT: When you divide with non 0



```

C:\Users\sveu>javac array.java
C:\Users\sveu>java array
9
NO EXCEPTION RAISED
bye

```

Explanation:

1. we only enter into catch block when you try to divide 88 with 0 in try block otherwise we wont.
2. if you try to divide 88 with 0 in try block, exception is thrown to catch block.

3. unless there is an exception in try block we wont enter into catch block

Example code 2:

```
import java.io.*; // For File class and FileNotFoundException
import java.util.Scanner; // For the Scanner class

/**
 * This program demonstrates how a FileNotFoundException
 * exception can be handled.
 */

class OpenFile
{
    public static void main(String[] args)
    {
        // Create a Scanner object for keyboard input.
        Scanner keyboard = new Scanner(System.in);

        // Get a file name from the user.
        System.out.print("Enter the name of a file: ");
        String fileName = keyboard.nextLine();

        // Attempt to open the file.
        try
        {
            // Create a File object representing the file.
            File file = new File(fileName);

            // Create a Scanner object to read the file.
            // If the file does not exist, the following
            // statement will throw a
            FileNotFoundException.
            Scanner inputFile = new Scanner(file);
            // If the file was successfully opened, the
            // following statement will execute.
            System.out.println("The file was found.");
        }
    }
}
```

```

    }
    catch (FileNotFoundException e)
    {
        // If the file was not found, the following
        // statement will execute.
        System.out.println("File not found.");
    }
    System.out.println("Done.");
}
}

```

Explanation:

1. Above code tries to open an entered text file. If it exists in your system, it displays "The file was found" if not you enter into catch block and it displays "The file was not found"
2. The name of the exception may raise in this code is "FileNotFoundException".

Multiple catch blocks for single try block

- A single try block can contain multiple catch blocks
- Based on type of exception raised corresponding catch block gets executed

Example code

```

import java.util.*;
class manycatches
{
    public static void main(String svec[])
    {
        int a,b,result;
        Scanner apple=new Scanner(System.in);
    }
}

```

```

try
{
    System.out.print("\nEnter dividend:");
    a=apple.nextInt();
    System.out.print("\nEnter divisor:");
    b=apple.nextInt();
    result=a/b;
    System.out.println("\nresult="+result);
    System.out.println("\nNO EXCEPTION RAISED");

}
catch(ArithmeticException praneeth)
{
    System.out.println("\nHi friend,division with 0 not defined");
    System.out.println("\nI cant proceed further....");
}
catch(InputMismatchException siri)
{
    System.out.println("\nHi friend,either dividend or divisor is
not in correct format:");
    System.out.println("\nI cant proceed further....");
}
System.out.println("\nbye");
}
}

```

Output 1


```
C:\Users\sview>javac manycatches.java
C:\Users\sview>java manycatches
Enter dividend:2
Enter divisor:0
Hi friend,division with 0 not defined
I cant proceed further.....
bye
```

Output 2:

```
C:\Users\sview>javac manycatches.java
C:\Users\sview>java manycatches
Enter dividend:9
Enter divisor:y
Hi friend,either dividend or divisor is not in correct format:
I cant proceed further.....
bye
```

Explanation:

- In above code 2 types of exceptions may raise
 - ArithmeticException
 - InputMismatchException
- “ArithmeticException” exception raises when divisor is 0
- “InputMismatchException” exception raises when you enter any non integer value for either a or b.

Nested try block

“a try block within another try block”

```
import java.util.*;
class nestedtry
{
    public static void main(String svec[])
    {
        int a,b,result;
        Scanner apple=new Scanner(System.in);
        try
        {
            System.out.print("\nEnter dividend:");
            a=apple.nextInt();
            System.out.print("\nEnter divisor:");
            b=apple.nextInt();

            try
            {
                result=a/b;
                System.out.println("\nresult="+result);
                System.out.println("\nNO EXCEPTION RAISED");
            }
            catch(ArithmeticException praneeth)
            {
                System.out.println("\nHi friend,division with 0 not
defined");
                System.out.println("\nI cant proceed further.....");
            }
        }
        catch(InputMismatchException siri)
        {
            System.out.println("\nHi friend,either dividend or divisor is
not in correct format:");
            System.out.println("\nI cant proceed further.....");
        }
        System.out.println("\nbye");
    }
}
```

```

import java.util.*;
class data
{
    public static void main(String[] args)
    {
        Scanner s=new Scanner(System.in);
        int i;
        try
        {
            try
            {
                try
                {
                    i=s.nextInt();
                    System.out.println(i);
                }
                catch(ArithmeticException e)
                {
                    System.out.println(e);
                }
            }
            catch(NegativeArraySizeException e)
            {
                System.out.println(e);
            }
        }
        catch(InputMismatchException z)
        {
            System.out.println(z);
        }
    }
}

```

“throw” keyword

So far we have seen ‘try’ block throwing exception to catch block for handling exceptions ,what if we manually throw an exception.Lets see how we would do that.

```

import java.util.*;
class throwkey

```

```

{
    public static void main(String svec[])
    {
        int a,b,result;
        Scanner apple=new Scanner(System.in);

        System.out.print("\nEnter dividend:");
        a=apple.nextInt();
        System.out.print("\nEnter divisor:");
        b=apple.nextInt();
        try
        {
            if(b==0)
            {
                ArithmeticException kohli=new ArithmeticException("divion is not
possible");
                throw kohli;//manually throwing exception
object
            }
            else
            {
                result=a/b;
                System.out.println("\nresult="+result);
                System.out.println("\nNO EXCEPTION RAISED");
            }
        }
        catch(ArithmeticException dhoni)
        {
            System.out.println("\nHi friend,division with 0 not
defined:"+dhoni);
            System.out.println("\nI cant proceed further....");
        }

        System.out.println("\nbye");
    }
}

```

- When the throw statement is executed, the execution of the surrounding try block is stopped and (normally) control is transferred to a catch block. The code in the catch block is executed next.

“Finally” keyword

- The finally block contains code to be executed whether or not an exception is thrown in a try block.
- The finally block, if used, is placed after a try block and its following catch blocks. The general syntax is as follows:

```
try
{
    ...
}
catch(ExceptionClass1      exception object)
{
    ...
}
.
.
.
catch(ExceptionClassN    exception object)
{
    ...
}
finally
{
    //Code to be executed whether or not an exception is thrown or
    caught
}
```

- The *finally block* is one or more statements that are always executed after the try block has executed and after any catch blocks have executed if an exception was thrown.

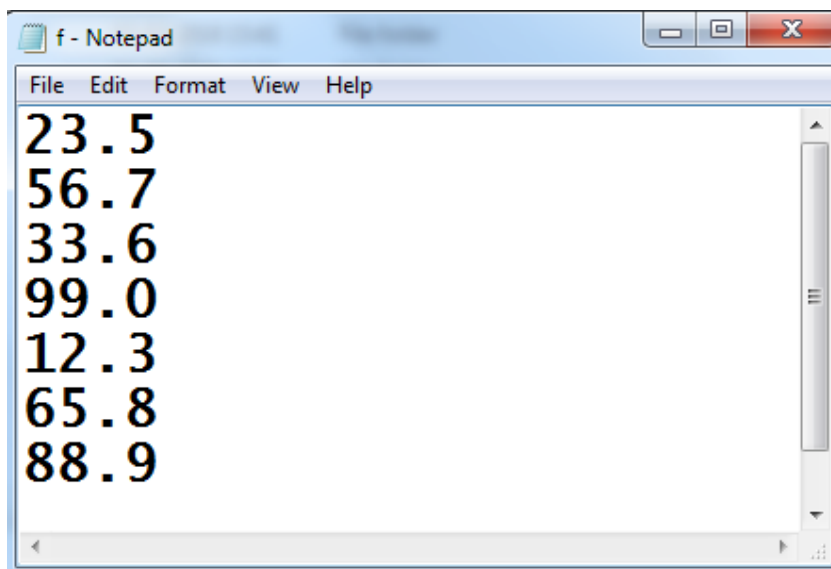
- The statements in the finally block execute whether an exception occurs or not.
- For example, the following code opens a file, which contains numbers of double data type and reads its contents. The outer try statement opens the file and has a catch clause that catches the FileNotFoundException.
- The inner try statement reads values from the file and has a catch clause that catches the InputMismatchException. The finally block closes the file regardless of whether an InputMismatchException occurs.

```
import java.util.*;
import java.io.*;
class usefinally
{
    public static void main(String svec[])
    {
        try
        {
            // Open the file.
            File lilly = new File("f.txt");
            Scanner inputFile = new Scanner(lilly);
            try
            {
                // Read and display the file's contents.
                while (inputFile.hasNext())
                {
                    System.out.println(inputFile.nextDouble());
                }
            }
            catch (InputMismatchException virat)
            {
                System.out.println("Invalid data found.");
            }
            finally
            {
                // Close the file.
                inputFile.close();
            }
        }
        catch (FileNotFoundException rose)
        {
            System.out.println("File not found.");
        }
    }
}
```

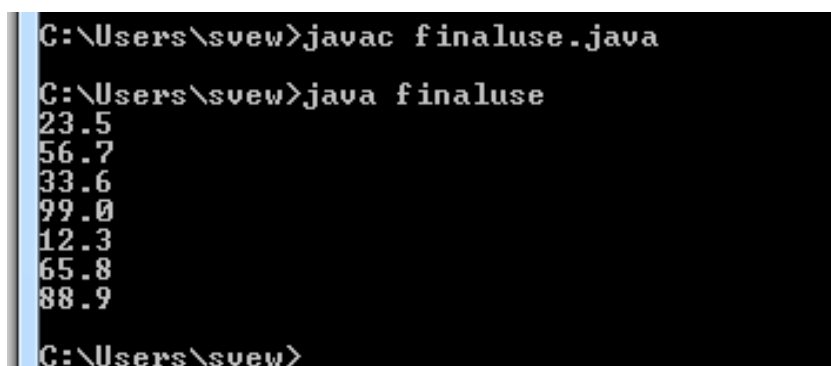
```
}  
}
```

Output 1:

If file “f.txt file” has following content



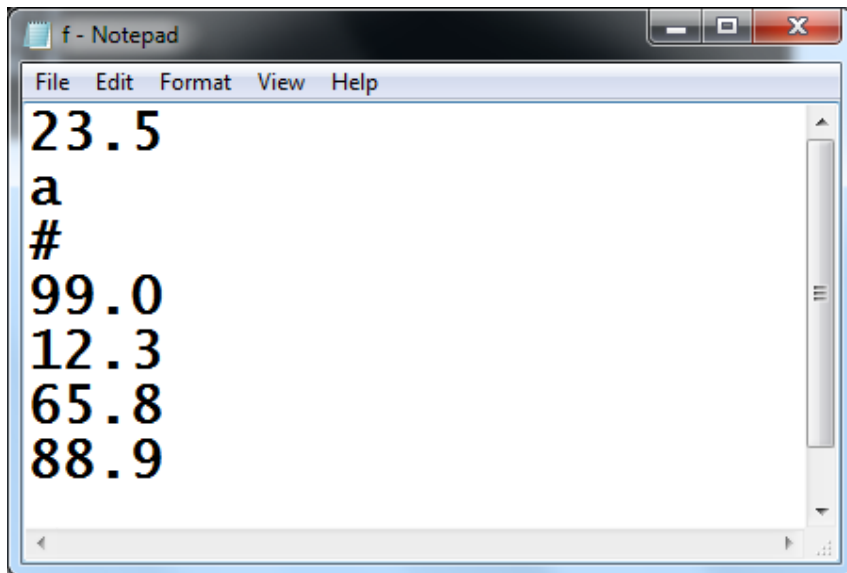
```
f - Notepad  
File Edit Format View Help  
23.5  
56.7  
33.6  
99.0  
12.3  
65.8  
88.9
```



```
C:\Users\sview>javac finaluse.java  
C:\Users\sview>java finaluse  
23.5  
56.7  
33.6  
99.0  
12.3  
65.8  
88.9  
C:\Users\sview>
```

Output 2:

If file “f.txt file” has following content



```
C:\Users\sview>javac finaluse.java  
  
C:\Users\sview>java finaluse  
23.5  
Invalid data found.  
  
C:\Users\sview>_
```

Types of exceptions

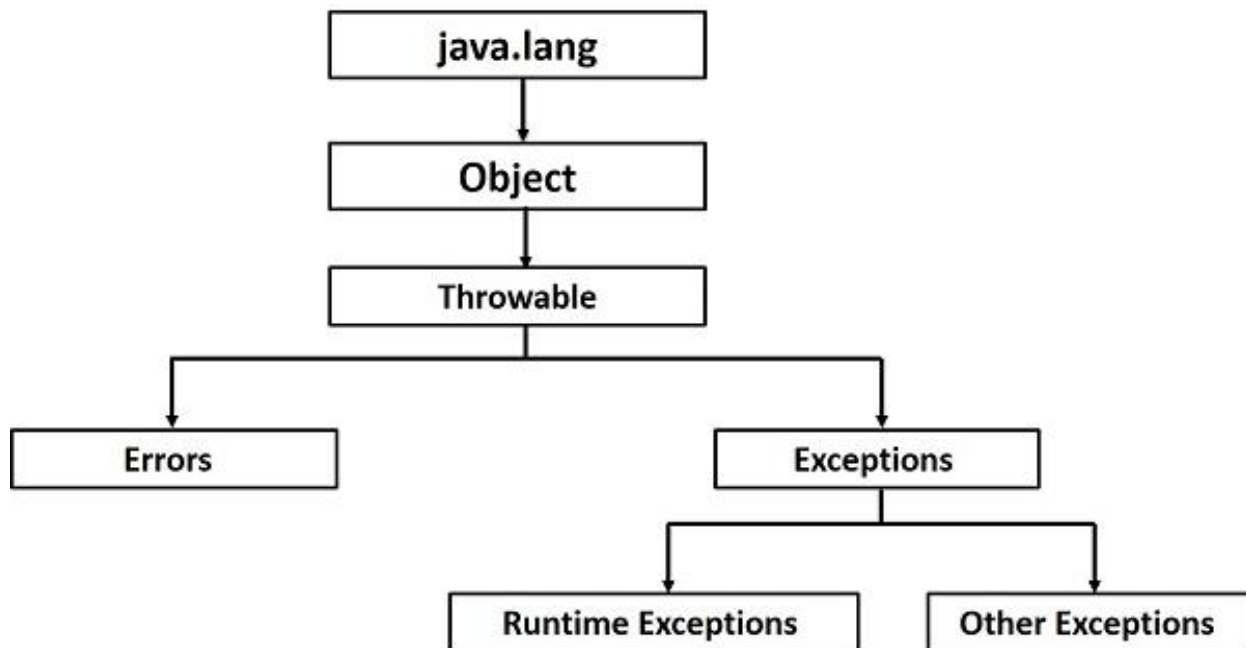
- There are two types of exceptions: **checked and unchecked**.
- A checked exception is an exception that is checked at compile time. All other exceptions are unchecked exceptions, also called runtime exceptions, because they are unchecked at compile time and are detected only at runtime.

- Trying to divide a number by 0 (ArithmeticException) and trying to convert a string with letters to an integer (NumberFormatException) are two examples of runtime exceptions.

Exception Hierarchy

All exception classes are subtypes of the java.lang.Exception class. The exception class is a subclass of the Throwable class. Other than the exception class there is another subclass called Error which is derived from the Throwable class.

Errors are abnormal conditions that happen in case of severe failures, these are not handled by the Java programs.



The Exception class has two main subclasses:

unchecked exceptions(runtime exception)

checked exceptions(compile time exception)

Built-in Exception classes:

Unchecked Exceptions

Exception Name	Meaning
ArithmeticException divide-by-zero.	Arithmetic error, such as
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
IllegalArgumentException method.	Illegal argument used to invoke a
NegativeArraySizeException	Array created with a negative size.
NullPointerException	Invalid use of a null reference.
NumberFormatException numeric format.	Invalid conversion of a string to a
StringIndexOutOfBoundsException bounds of a string.	Attempt to index outside the

eg 1:NullPointerException

```
class nullpoint
{
    public static void main(String svec[])
    {

        String b=null;
        try
        {
```

```

        System.out.println(b.charAt(0));
    }
    catch(Exception harikrishna)
    {
        System.out.println(harikrishna);
    }

}
}

```

NegativeArraySizeException

```

class negarray
{
    public static void main(String[] args)
    {
        try
        {
            int a[]=new int[-5];
        }
        catch(NegativeArraySizeException e)
        {
            System.out.println(e);
        }
    }
}

```

InputMismatchException

```

import java.util.*;
class data
{
    public static void main(String[] args)
    {

```

```

Scanner s=new Scanner(System.in);
int i;
try
{
    i=s.nextInt();
    System.out.println(i);
}
catch(InputMismatchException e)
{
    System.out.println(e);
}
}
}

```

checked exceptions

ClassNotFoundException	Class not found.
IllegalAccessException	Access to a class is denied.
InstantiationException abstract class or interface.	Attempt to create an object of an
NoSuchFieldException	A requested field does not exist.
NoSuchMethodException	A requested method does not exist.

throws Clause:

“If you define a method that might throw exceptions of some particular class, then normally either your method definition must include a catch block that will catch the exception or you must declare (that is, list) the exception class within a throws clause.”

```
class a
{
    int p,q,r;
    public void div(int x,int y)throws ArithmeticException
    {
        p=x;
        q=y;
        r=p/q;
        System.out.println(r);
    }
}
class divide
{
    public static void main(String svec[])
    {
        a a1=new a();
```

```

        try
        {
            a1.div(3,3);
        }
        catch(Exception harikrishna)
        {
            System.out.println(harikrishna);
        }
    }
}

```

```

class a
{
    public void check()throws IllegalAccessException
    {
        IllegalAccessException bob=new IllegalAccessException();
        throw bob;
    }
}
class throws
{
    public static void main(String svec[])
    {
        a a1=new a();
        try
        {
            a1.check();
        }
        catch(IllegalAccessException harikrishna)
        {
            System.out.println("illegal access bro/sis");
        }
    }
}

```

```

class a
{

```

```

    public void check()
    {
        try
        {
            IllegalAccessException bob=new IllegalAccessException();
            throw bob;
        }
        catch(IllegalAccessException harikrishna)
        {
            System.out.println("illegal access bro/sis");
        }
    }
}
class trows
{
    public static void main(String svec[])
    {
        a a1=new a();
        a1.check();
    }
}

```

Creating our own exception classes

So far we have seen pre defined or built-in exceptions in java like

- ArithmeticException
- ArrayIndexOutOfBoundsException
- InputMismatchException
- NumberFormatException

we can also create exceptions with the name we wish. To do so we require following steps

- Create an exception class that you wish and make it to inherit(extend) "Exception" class.
- Then throw an exception object of the class you have just created.

```
import java.util.Scanner;

class IHateNegativeException extends Exception
{
    /*public String toString()
    {
        return "don't be negative...";
    }*/
}

class myexception
{
    public static void main(String sachin[])
    {
        int num;

        Scanner enter=new Scanner(System.in);
        System.out.print("\nEnter a number:");
        num=enter.nextInt();

        try
        {
            if(num<0)
            {
                IHateNegativeException ramu = new
IHateNegativeException();
                throw ramu;
            }
        }
    }
}
```



```

        }
        else
        {
            System.out.print("\nBe positive....");
        }
    }
    catch(IHateNegativeException sai)
    {
        System.out.print("\n"+sai+"\n");
    }
    System.out.print("\n"+"bye"+"n");
}
}

```

```

class nesttry
{
    public static void main(String d[])
    {
        try
        {
            try
            {
                int r=10/0;
            }
            catch(ArrayIndexOutOfBoundsException vini)

```

```
        {  
            System.out.println(vini);  
        }  
    }  
    catch(ArithmeticException nani)  
    {  
        System.out.println(nani);  
    }  
}  
}
```

JAVA I/O

Java I/O (Input and Output) is used *to process the input and produce the output*.

Java uses the concept of a stream to make I/O operation fast.

The java.io package contains all the classes required for input and output operations.

Stream

A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.

OutputStream vs InputStream

The explanation of OutputStream and InputStream classes are given below:

OutputStream

Java application uses an output stream to write data to a destination;

it may be a file, an array, peripheral device

InputStream

Java application uses an input stream to read data from a source; it may be a file, an array, peripheral device or socket.

Let's understand the working of Java OutputStream and InputStream by the figure given below.

