Workshop Link: https://attendee.gotowebinar.com/register/2348474801872643416
Exception Handling:

Q)What is the difference between Error and Exception?

Ans:

Error is a problem , it will not allow us to execute applications.

There are two types of Errors.

1. Compilation Errors
2. Runtime Errors

Compilation Errors:
These errors are generated at compilation time.
There are three types of Compilation Errors.
1. Lexical Errors
2. Syntax Errors
3. Semantic Errors

Lexical Errors:
These errors are identified by the Lexical Analysis phase in Compilation.
EX: Mistakes in the tokens

```
EX:   int i = 10; ----> Valid
      nit i = 10; ----> Invalid, Lexical Error

EX:   for(int i = 0; i < 10; i++){
      }
      Status: Valid
      fro(int i = 0; i < 10; i++){
      }
      Status: Invalid, Lexical Error.
```

Syntax Errors:
These errors are identified in the Syntax Analysis phase in Compilation.
EX: Mistakes in the syntaxes.

```
EX:   int i = 10; --> Valid
      i 10 int; ----> Invalid, Syntax Error
EX:   while(i < 10){
            Sopln(i);
      }
      Status: Valid
      (i < 10) while{
            Sopln(i);
            i = i + 1;
      }
      Status: Invalid, Syntax Error.

EX:   int i = 10 ----->  Syntax Error
```

Semantic Error:
These errors are identified in the Semantic Analysis phase in compilation.
EX: Meaningless statements.
EX: Providing incompatible operands for the operators.

```
EX:   int i = 10;
      boolean b = true;
      char c = i + b;
```

Note: Along with the above three types of errors, all the programming languages are having their own errors as per the programming language rules and regulations.

```
EX:   Unreachable Statement
      Variable i might not have been initialized.
      Illegal Start of Expression
      -----
      ------
```

Runtime Errors or Error:
These errors are identified at runtime.

The errors which are identified at runtime and which are not having solutions programmatically then these errors are “Runtime Error”.

EX: Unavailability of IO Components.
EX: InsufficientMainMemory

Exception:
These are runtime errors identified at runtime and which are having solutions programmatically.
EX: ArithmeticException
EX: NullPointerException
EX: FileNotFoundException

Exception:
Exception is an unexpected event occurred at runtime of the application, which may be provided by the users while entering dynamic input to the java applications, which may be provided by the Databases when we perform database operations from java applications in JDBC applications, which may be provided by the network when we establish connection between client and Server in Distributed applications,....... Causes abnormal termination to the applications.

There are two types of Terminations for the applications,

1. Smooth Termination
2. Abnormal Termination

Smooth Termination: If the application execution is terminated at the end of the program then that termination is “Smooth Termination”.

Abnormal Termination: If the application execution is terminated in the middle of the program then that termination is called Abnormal Termination.

If the abnormal termination is available for the applications then it may provide the following serious problems

1. It may Corrupt the local Operating System.
2. It may hang the network if it is a network based application.
3. It may down the server if it is a server side application.
4. It may collapse the Database if it is a database related applications
   —----
   —-----

To avoid the above serious problems , we have to avoid Abnormal Terminations in our applications. To avoid abnormal terminations to our applications we have to handle the exceptions properly , here to handle exceptions properly we have to use a set of mechanisms explicitly called “Exception Handling Mechanism”.

Java is a Robust Programming Language, because

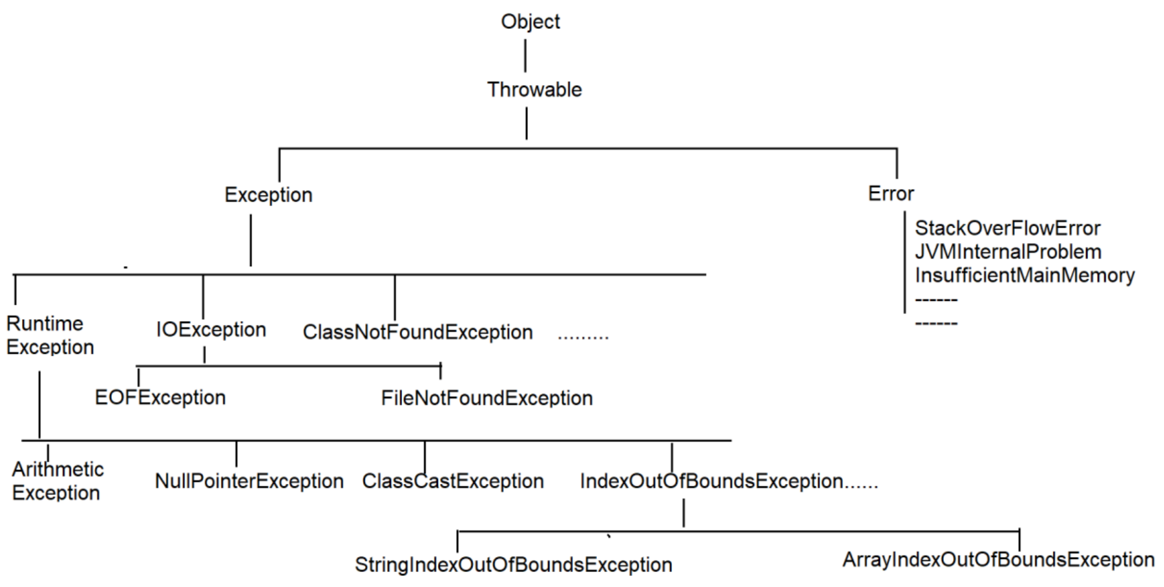
1. Java has a very good memory management system in the form of Heap memory management system, it is a dynamic memory management system , it allocates and deallocates memory for the objects at runtime as per the requirement.
2. Java has a very good Exception Handling mechanism , because Java has provided a very good predefined library to represent and handle almost all the exceptions which are coming frequently in Java applications.

There are two types of Exceptions in Java.

1. Predefined Exceptions
2. User defined Exceptions

Predefined Exceptions:

These exceptions are defined by the Java programming languages , Java has provided predefined classes for these exceptions in the Java predefined library.



There are two types of predefined exceptions.

1. Checked Exceptions
2. Unchecked Exceptions

Q)What is the difference between Checked Exceptions and Unchecked Exceptions?

Ans:
----
1. If any exception is identified by the compiler at compilation time then that exception is called Checked Exception.

Note: Really exceptions occur at runtime only, not at compilation time, but Compiler can recognize some exceptions which are going to generate at runtime then that exceptions are called “Checked Exceptions”.

If any exception is identified by the JVM at runtime , not by the compiler at compilation time then that Exception is called “Unchecked Exception”.
2. RuntimeException and its subclasses, Error and its subclasses are the examples for Unchecked Exceptions.

All the remaining Exceptions are the examples for Checked Exceptions.

There are two types of Checked Exceptions.

1. Pure Checked Exceptions
2. Partially Checked Exceptions

Q)What is the difference between Pure checked exceptions and Partially Checked Exceptions?

Ans:

If any checked Exception has only Checked Exceptions as subclasses then that checked exception is called “Pure Checked exception”.
EX: IOException

If any checked exception has at least one subclass as an unchecked exception then that checked exception is called “Partially Checked Exception”.
EX: Exception, Throwable

Overview of Predefined Exceptions:
----
1. ArithmeticException:
If we divide any number by zero then JVM will raise ArithmeticException.
EX:
```java
public class Main {
   public static void main(String[] args) {
       int i = 100;
       int j = 0;
       float f = i/j;
       System.out.println(f);
   }
}
```

If we run the above program then JVM will provide the following exception message.

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
	at Main.main(Main.java:5)
```

The above Exception message is divided into the following three parts.
1. Exception Name : java.lang.ArithmeticException
2. Exception Description : / by zero
3. Exception Location : Main.java: 5

2. NullPointerException:
If we access any instance variable or instance method on a reference variable containing null value then JVM will raise NullPointerException.

EX:
```
import java.util.Date;

public class Main {
   public static void main(String[] args) {
        Date date = null;
        System.out.println(date.toString());
   }
}
```

If we run the above program then JVM will provide the following exception details.
1. Exception Name: java.lang.NullPointerException
2. Exception Description : Cannot invoke "java.util.Date.toString()" because "date" is null
3. Exception Location: Main.java: 6

3. ArrayIndexOutOfBoundsException:
In Java applications, when we access an element from an array on the basis of the index value , when we insert an element in an array on the basis of the index value, where the provided index value is in the outside range of array indexes there JVM will raise ArrayIndexoutOfBoundsException.
EX:
```
import java.util.Date;

public class Main {
   public static void main(String[] args) {
        int[] a = {1,2,3,4,5};
        System.out.println(a[2]);// 3
        System.out.println(a[4]);// 5
        System.out.println(a[10]);
    }
}
```

If we run the above program then JVM will provide the following Exception details.

1. Exception Name : java.lang.ArrayIndexOutOfBoundsException
2. Exception Description : Index 10 out of bounds for length 5
3. Exception Location : Main.java: 7

4. StringIndexOutOfBoundsException:
In Java applications , when we are performing String operations on the basis of the index value , where the provided index value is in the outside range of the String indexes there JVM will raise StringIndexOutOfBoundsException .
EX:

```
import java.util.Date;

public class Main {
    public static void main(String[] args) {
        String str = new String("Durgasoft");
        System.out.println(str.charAt(6));
        System.out.println(str.charAt(20));
    }
}
```

If we run the above program then JVM will provide the following exception details.

1. Exception Name : java.lang.StringIndexOutOfBoundsException
2. Exception Description: String index out of range: 20
3. Exception Location : Main.java: 7

5. ClassNotFoundException:
In Java applications, if we want to load a particular class bytecode to the memory then we will use the Class.forName() method.

```
Class cls = Class.forName(“Employee”);
```

JVM will search for Employee.class at the current location, at the predefined library and at the locations referred by the classpath environment variable.

If the required Employee.class file is not available at all the above locations then JVM will raise a ClassNotFoundException.

EX:
```
public class Main {
   public static void main(String[] args) throws
Exception {
        Class cls = Class.forName("Employee");
   }
}
```

If we run the above program then JVM will provide the following exception details.

1. Exception Name : java.lang.ClassNotFoundException
2. Exception Description : Employee
3. Exception Location : Main.java: 4

6. InstantiationException:
In Java applications, by using Class.forName() method we are able to load a particular class bytecode to the memory, after loading class bytecode if we want to create an object for the loaded class we have to use the following method java.lang.Class.

```
public Object newInstance()throws InstantiationException,
IllegalAccessException
```

```
Class cls = Class.forName(“Employee”);
Employee emp = (Employee) cls.newInstance();
```

If we execute the above instruction, JVM will search for a 0-arg constructor in the Employee class. If no 0-arg constructor is available in the Employee class then JVM will raise InstantiationException .

EX:
```
class Employee{
   Employee(int i){
        System.out.println("Employee-Constructor");
   }
}
public class Main {
   public static void main(String[] args) throws
Exception {
        Class cls = Class.forName("Employee");
        Employee employee = (Employee) cls.newInstance();
   }
}
```

If we run the above program then JVM will provide the following details
1. Exception Name: java.lang.InstantiationException
2. Exception Description : Employee
3. Exception Location : Main.java: 9

7. IllegalAccessException:
In Java applications, by using Class.forName() method we are able to load a particular class bytecode to the memory, after loading class bytecode if we want to create an object for the loaded class then we have to use the following method from java.lang.Class.

```
public Object newInstance()throws InstantiationException,
IllegalAccessException
```

```
Class cls = Class.forName("Employee");
Employee emp = (Employee) cls.newInstance();
```

If we execute the above instruction, JVM will search for a non private constructor in the Employee class. If we have a private constructor in the Employee class then JVM will raise IllegalAccessException .
EX:

```
class Employee{
   private Employee(){
        System.out.println("Employee-Constructor");
   }
}
public class Main {
   public static void main(String[] args) throws
Exception {
        Class cls = Class.forName("Employee");
        Employee employee = (Employee) cls.newInstance();
   }
}
```

If we run the above code then JVM will provide the following exception details.

```
Exception Name : java.lang.IllegalAccessException
Exception Description :  class Main cannot access a member of class Employee
                         with modifiers "private"
Exception Location : Main.java: 9
```

8. ClassCastException:
In Java applications, we are able to keep subclass object reference value in superclass reference variable, but we are unable to keep superclass object reference value in subclass reference variable, if we keep superclass object reference value in subclass reference variable then JVM will raise ClassCastException.

EX:
```
class Employee{

}
class Manager extends Employee{

}
public class Main {
   public static void main(String[] args) throws
Exception {
        Employee employee = new Employee();
        Manager manager = (Manager) employee;
    }
}
```

If we run the above program then JVM will provide the following Exception details.

```
1. Exception Name : java.lang.ClassCastException:
2. Exception Description : class Employee cannot be cast to class Manager
3. Exception Location : Main.java: 10
```

9. FileNotFoundException:
In Java applications, when we are trying to read data by using FileInputStream or FileReader from a particular file and if the provided file is not available then JVM will raise FileNotFoundException.

EX:
```
import java.io.FileInputStream;

public class Main {
   public static void main(String[] args) throws
Exception {
        FileInputStream fileInputStream = new
FileInputStream("E:/abc/welcome.txt");
    }
}
```

If we run the above code then JVM will provide the following exception details.

```
1. Exception Name : java.io.FileNotFoundException
2. Exception Description :  E:\abc\welcome.txt (The system cannot find the
                              file specified)
3. Exception Location : Main.java: 5
```

‘throw’ Keyword:

‘throw’ is a Java keyword, it can be used to raise or generate an exception intentionally as per the application requirement.

Syntax: throw new ExceptionClassName([ParamValues]);

EX:
Student.java
```
package com.durgasoft.entities;

public class Student {
    private String sid;
    private String sname;
    private int smarks;

    public Student(String sid, String sname, int smarks) {
        this.sid = sid;
        this.sname = sname;
        this.smarks = smarks;
    }

    public void getStudentStatus(){
```

```java
        System.out.println("Student Details");
        System.out.println("------------------------");
        System.out.println("Student Id      : "+sid);
        System.out.println("Student Name    : "+sname);
        System.out.println("Student Marks   : "+smarks);
        String status = "";
        if( smarks >= 0 && smarks <= 100){
            if(smarks < 35){
                status = "FAIL";
            }else if(smarks >= 35 && smarks < 50){
                status = "THIRD CLASS";
            }else if(smarks >= 50 && smarks < 60){
                status = "SECOND CLASS";
            }else if(smarks >= 60 && smarks < 70){
                status = "FIRST CLASS";
            }else{
                status = "DISTINCTION";
            }
        }else{
            throw new RuntimeException("Invalid Marks");
        }
        System.out.println("Student Status  : "+status);
    }
}
```

Main.java

```java
import com.durgasoft.entities.Student;

public class Main {
    public static void main(String[] args) {
        Student student1 = new Student("S-111", "AAA",
77);
        student1.getStudentStatus();

        Student student2 = new Student("S-222", "BBB",
150);
        student2.getStudentStatus();
    }
}
```

Note:
If we provide any statement immediately after the throw statement then the compiler will raise an error like Unreachable Statement.

```
public class Main {
   public static void main(String[] args) {
       System.out.println("Before Exception");
       throw new RuntimeException();
       System.out.println("After Exception");
   }
}
```

Status: Compilation Error, Unreachable Statement

In Java applications, we are able to handle the exceptions in the following two ways.

1. By Using 'throws' keyword.
2. By using 'try-catch-finally' blocks.

'throws' keyword:

'throws' is a Java keyword, it can be used to bypass the generated exception from the present method to the caller method in order to handle that exception.

In general, throws keyword will be used for Checked Exceptions.

Note: In method declaration, along with throws keyword we have to provide an exception name ,it must be either the same as the generated exception or super class of the generated exception.

Note: In general, we will use throws keyword in the method declaration , along with throws keyword we are able to use more than one exception class.

EX:

```
void m1()throws Exception1, Exception2,... Exception_n{
      -------
}
```

EX:

```
import java.io.*;
class A{
      void add(){
            concat();
      }
      void concat(){
            throw new IOException("My IO Excxeption");
      }
}
class Test{
      public static void main(String[] args){
            A a = new A();
            a.add();
      }
}
```

```
D:\java6>javac Test.java
Test.java:7: error: unreported exception IOException; must be caught or
declared to be thrown
                throw new IOException("My IO Excxeption");
                ^
1 error

D:\java6>
```

EX:

```
import java.io.*;
class A{
      void add(){
            concat();
      }
      void concat()throws IOException {
            throw new IOException("My IO Excxeption");
      }
}
class Test{
      public static void main(String[] args){
            A a = new A();
            a.add();
      }
}
```

```
D:\java6>javac Test.java
Test.java:4: error: unreported exception IOException; must be caught or
declared to be thrown
                concat();
                      ^
1 error

D:\java6>
```

EX:

```
import java.io.*;
class A{
      void add()throws Exception{
            concat();
      }
      void concat()throws IOException {
            throw new IOException("My IO Excxeption");
      }
}
class Test{
      public static void main(String[] args){
            A a = new A();
            a.add();
      }
}
```

```
D:\java6>javac Test.java
Test.java:13: error: unreported exception Exception; must be caught or
declared to be thrown
                a.add();
                     ^
1 error

D:\java6>
```

EX:

```
import java.io.*;
class A{
      void add()throws Exception{
            concat();// IOException
      }
      void concat()throws IOException {
            throw new IOException("My IO Excxeption");
      }
}
class Test{
      public static void main(String[] args)throws Throwable{
            A a = new A();
            a.add();// Exception
      }
}
```

```
D:\java6>javac Test.java

D:\java6>java Test
Exception in thread "main" java.io.IOException: My IO Excxeption
        at A.concat(Test.java:7)
        at A.add(Test.java:4)
        at Test.main(Test.java:13)

D:\java6>
```

If we provide throws keyword in the main() method declaration then Exception will be bypassed to the JVM, where JVM has a default Exception Handler, it will handle the exception and it will provide an Exception message at all the locations.

Q)What are the differences between throw keyword and throws keyword?

Ans:
-----

1. throw keyword can be used to raise an exception as per the application requirement.

   throws keyword can be used to bypass the generated exception from the present method to the caller method.

2. In general, we will use the throw keyword inside the method body.

   In general, we will use the throws keyword in the method declaration part.

3. throw keyword is able to allow only one exception name.

throws keyword is able to allow more than one exception name.

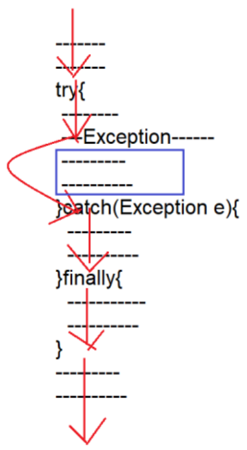
try-catch-finally:

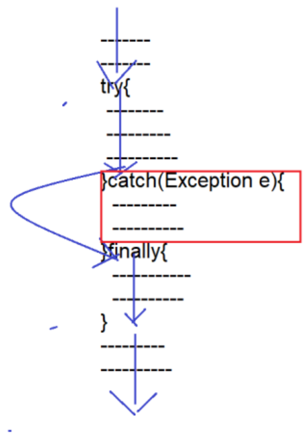
In Java applications, throws keyword is not handling the exceptions really, it is able to bypass the generated exception from the present method to the caller method in order to handle exceptions.

In java applications, we want to handle the exception right from the location where it is generated , for this we have to use “try-catch-finally”.

Syntax:
```
try{
  -----Instructions----
}catch(ExceptionName refVar){
  ----Instructions-----
}finally{
  -----Instructions-----
}
```

try block:
----
1. It will include the doubtful code where exceptions may or may not be generated.
2. If we have any exception in the try block then JVM will bypass flow of execution to the catch block by passing the generated exception object as parameter to the catch block and by skipping the remaining instructions after the exception in the try block.
3. If no exception is identified in try block then JVM will execute try block completely, at the end of the try block JVM will bypass flow of execution to the finally block by skipping catch block.





EX:

```
public class Main {
   public static void main(String[] args) {
       System.out.println("Before try block");
       try{
           System.out.println("Inside try block, before 
Exception");
           float f = 100/0;
           System.out.println("Inside try block, After 
Exception");
      }catch(Exception e){
           System.out.println("Inside catch block");
      }finally{
           System.out.println("Inside finally block");
      }
       System.out.println("After finally block");
   }
}
```

```
Before try block
Inside try block, before Exception
Inside catch block
Inside finally block
After finally block
```

EX:
```java
public class Main {
   public static void main(String[] args) {
        System.out.println("Before try block");
        try{
            System.out.println("Inside try block");

       }catch(Exception e){
            System.out.println("Inside catch block");
       }finally{
            System.out.println("Inside finally block");
       }
        System.out.println("After finally block");
   }
}
```

```
Before try block
Inside try block
Inside finally block
After finally block
```

catch block:

In try-catch-finally syntax, when we have an exception in try block the JVM will execute catch block.

The main purpose of catch block is to catch the exception from try block and to display the generated exception details on console.

To display the exception details on the console we have to use the following three approaches.

1. By Using printStackTrace() method:
----
public void printStackTrace()

It is able to display the exception details like Name of the exception, Description of the Exception and Location of the Exception.

2. By using System.out.println(e):

When we pass Exception object reference variable as parameter to System.out.println() method , JVM will execute toString() method internally , where in Exception classes Object class provided toString() method was overridden in such a way that to return a string that contains the exception details like name of the exception and description of the exception.

```
public String toString()
```

3. By using getMessage() method:

```
public String getMessage()
```

It is able to return the exception details like only Description of the exception.

EX:
```
public class Main {
    public static void main(String[] args) {
        try{
            float f = 100/0;
        }catch(Exception e){
            e.printStackTrace();
            System.out.println();

            System.out.println(e);
            System.out.println();

            System.out.println(e.getMessage());
        }finally{

        }
    }
}
```

OP:
```
java.lang.ArithmeticException: / by zero
      at Main.main(Main.java:4)
```

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

/ by zero
```

finally block:

Finally block is able to include a set of instructions which must be executed by JVM irrespective of having exceptions in try block and irrespective of executing catch block.

In try-catch-finally syntax, only finally block is giving guarantee for execution.

In a Java application, we may use some resources like streams, database connections, network connections,..... as per the requirement, if we work with these resources then we may get some exceptions , here to handle these exceptions if we use try-catch-finally then we must close the resources irrespective of getting exceptions or not in try block , so here to get the guarantee for resources closing operations we must provide the respective resources close operations inside the finally block, because finally block is giving guarantee for execution.

Q)Find the output from the following program?

```
class A{
   int m1(){
       try{
           return 10;
       }catch(Exception e){
           return 20;
       }finally{
           return 30;
       }
   }
}
public class Main {
   public static void main(String[] args) {
       A a = new A();
       System.out.println(a.m1());
```

```
    }
}
```

OP:
30

EX:

```
class A{
    int m1(){
        try{
            float f = 100/0;
            return 10;
        }catch(Exception e){
            return 20;
        }finally{
            return 30;
        }
    }
}
public class Main {
    public static void main(String[] args) {
        A a = new A();
        System.out.println(a.m1());
    }
}
```

OP:
30

Q)Is it possible to write try block without catch block?

Ans:

Yes, it is possible to write try block without catch block but we must provide a finally block.

```
try{
}finally{
}
```

In the above context, if we have any exception then JVM will execute both try and finally blocks then JVM will display the generated exception details.

EX:

```
public class Main {
   public static void main(String[] args) {
       try{
           System.out.println("Inside try block, before
exception");
           float f = 100/0;
       }finally{
           System.out.println("Inside finally block");
       }
   }
}
```

OP:

```
Inside try block, before exception
Inside finally block
Exception in thread "main" java.lang.ArithmeticException: / by zero
      at Main.main(Main.java:6)
```

Q)Is it possible to provide a try block without a finally block?

Ans:

Yes, it is possible to provide a try block without a finally block but we must provide a catch block.

```
try{
}catch(Exception e){
}
```

Q)Is it possible to write try-catch-finally inside the try block? Inside catch block? Inside finally block?

Ans:

Yes, It is possible to write try-catch-finally inside try block, inside catch block and inside finally block.

EX:
```
try{
      try{
      }catch(Exception e){
      }finally{
      }
}catch(Exception e){
}finally{
}
```

EX:
```
try{
}catch(Exception e){
      try{
      }catch(Exception e1){
      }finally{
      }

}finally{
}
```

EX:
```
try{
}catch(Exception e){
}finally{
      try{
      }catch(Exception e){
      }finally{
      }
}
```

EX:
```
import java.util.Date;

public class Main {
   public static void main(String[] args) {
        System.out.println("Before try block");
        try {
            System.out.println("Inside try, before nested
try");
            try{
                System.out.println("Inside try, inside
nested try");
            }catch(Exception e){
                System.out.println("Inside try, inside
nested catch");
            }finally{
                System.out.println("Inside try, inside
nested finally");
            }
            System.out.println("Inside try, after nested
finally");
            float f = 100/0;
        }catch(Exception e){
            try{
                System.out.println("Inside catch, inside
nested try");
                Date d = null;
                System.out.println(d.toString());
            }catch(Exception e1){
                System.out.println("Inside catch, inside
nested catch");
            }finally{
                System.out.println("Inside catch, inside
nested finally");
            }
            System.out.println("Inside catch, after nested
finally");
        }finally{
            try{
                System.out.println("Inside finally, inside
nested try");
            }catch(Exception e){
                System.out.println("Inside finally, inside
nested catch");
            }finally{
                System.out.println("Inside finally, inside
nested finally");
            }
            System.out.println("Inside finally, after
nested finally");
        }
        System.out.println("After finally");
    }
}
```

```
OP:
Before try block
Inside try, before nested try
Inside try, inside nested try
Inside try, inside nested finally
Inside try, after nested finally
Inside catch, inside nested try
Inside catch, inside nested catch
Inside catch, inside nested finally
Inside catch, after nested finally
Inside finally, inside nested try
Inside finally, inside nested finally
Inside finally, after nested finally
After finally
```

Q) Is it possible to provide more than one catch block for a single try block?

Ans:

Yes, it is possible to provide more than one catch block for a single try block with the following conditions.

1. We can write multiple catch blocks with the Exception class names , but if these Exception classes are having inheritance relation then we must provide all the catch blocks as per the exception classes' inheritance increasing order. If Exception classes are not having inheritance relation then it is possible to provide all catch blocks in any order.
2. If any catch block has a pure checked exception then the respective try block must raise the same pure checked exception.

```
EX1:
try{
----
}catch(ArithmeticException e){
}catch(NullPointerException e){
}catch(ArrayIndexOutOfBoundsException e){
}
Status: Valid

EX2:
try{
----
}catch(NullPointerException e){
}catch(ArithmeticException e){
}catch(ArrayIndexOutOfBoundsException e){
}
Status: Valid

EX2:
try{
----
}catch(ArithmeticException e){
}catch(RuntimeException e){
}catch(Exception e){
}
Status: Valid

EX2:
try{
----
}catch(Exception e){
}catch(RuntimeException e){
}catch(ArithmeticException e){
}
Status: Invalid

EX:
try{
      throw new IOException();
}catch(ArithmeticException e){
}catch(IOException e){
}catch(NullPointerException e){
}
```

Status: Valid

EX:
```
try{
      throw new ArithmeticException();
}catch(ArithmeticException e){
}catch(IOException e){
}catch(NullPointerException e){
}
```
Status: Invalid

Custom Exceptions or User Defined Exceptions:
——

These Exceptions are defined by the developers as per their application requirements.

In Java applications, if we want to prepare User defined exceptions we have to use the following steps.

1. Create a User Defined exception class.
   a. Declare an user defined class.
   b. Extend User defined class from java.lang.Exception .
   c. In user defined class , declare String parameterized Constructor.
   d. In the constructor access super class String parameterized constructor by using super(str).

Note: The main purpose of String parameterized constructor and accessing String parameterized super class constructor is to provide user defined exception description to the User defined Exception inorder to display User defined exception description through printStackTrace() method, System.out.println(e), System.out.println(e.getMessage()).

EX:
```
public class InsufficientFundsException extends Exception{
      public InsufficientFundsException(String exceptionDescription){
            super(exceptionDescription);
      }
}
```

super(exceptionDescription) will provide user defined exception description to the printStackTrace() method, toString() method and getMessage() method to display on console.

2. In java applications raise and handle User defined exceptions.
   To raise the exception we have to use the throw keyword.
   To handle the exception we have to use try-catch-finally.

EX:

```
try{
      throw new InsufficientFundsException("Funds Are Not Sufficient in your
      Account");
}catch(InsufficientFundsException e){
      e.printStacktrace();
}
```

EX:

InsufficientFundsException.java

```
package com.durgasoft.exceptions;

public class InsufficientFundsException extends Exception{
    public InsufficientFundsException(String message) {
        super(message);
    }
}
```

Account.java

```
package com.durgasoft.entities;

public class Account {
   private String accNo;
   private String accHolderName;
   private String accType;
   private int accBalance;



   public String getAccNo() {
       return accNo;
   }

   public void setAccNo(String accNo) {
       this.accNo = accNo;
   }

   public String getAccHolderName() {
       return accHolderName;
   }
```

```java
    public void setAccHolderName(String accHolderName) {
        this.accHolderName = accHolderName;
    }

    public String getAccType() {
        return accType;
    }

    public void setAccType(String accType) {
        this.accType = accType;
    }

    public int getAccBalance() {
        return accBalance;
    }

    public void setAccBalance(int accBalance) {
        this.accBalance = accBalance;
    }
}
```

Transaction.java

```java
package com.durgasoft.entities;

import com.durgasoft.exceptions.InsufficientFundsException;

public class Transaction {
    private String transactionId;

    public Transaction(String transactionId) {
        this.transactionId = transactionId;
    }

    public void withdraw(Account account, int withdrawAmount){
        try {
            System.out.println("Transaction Details");
            System.out.println("---------------------------");
            System.out.println("Transaction Id       : "+transactionId);
            System.out.println("Account Number       : "+account.getAccNo());
            System.out.println("Account Holder Name  :
"+account.getAccHolderName());
            System.out.println("Account Type         :
"+account.getAccType());
            System.out.println("Transaction Type     : WITHDRAW");
            int accBalance = account.getAccBalance();
            if(accBalance < withdrawAmount){
                System.out.println("Total Balance       :
"+account.getAccBalance());
```

```
                System.out.println("Transaction Status   : FAILURE");
                throw new InsufficientFundsException("Funds Are not sufficient
in your Account");
            }else{
                int newBalance = accBalance - withdrawAmount;
                account.setAccBalance(newBalance);
                System.out.println("Total Balance        :
"+account.getAccBalance());
                System.out.println("Transaction Status   : SUCCESS");
            }

        }catch (InsufficientFundsException e){
            System.out.println("Reason    : "+e.getMessage());
        }finally{
            System.out.println("**********ThankQ, Visit Again************");
        }
    }
}
```

Main.java

```
import com.durgasoft.entities.Account;
import com.durgasoft.entities.Transaction;

import java.io.IOException;

public class Main {
   public static void main(String[] args) {
        Account account1 = new Account();
        account1.setAccNo("a111");
        account1.setAccHolderName("Durga");
        account1.setAccType("Savings");
        account1.setAccBalance(20000);

        Transaction transaction1 = new Transaction("t111");
        transaction1.withdraw(account1, 5000);

        Account account2 = new Account();
        account2.setAccNo("a222");
        account2.setAccHolderName("Anil");
        account2.setAccType("Savings");
        account2.setAccBalance(10000);
        Transaction transaction2 = new Transaction("t222");
        transaction2.withdraw(account2, 15000);

    }
}
```

JAVA 7 features in Exception Handling:
----

1. Multi-Catch Block
2. Try-with-resources

Multi-Catch Block:

In the Multi-Catch block, we are able to handle multiple exceptions by using a single catch block.

Syntax:

```
try{
}catch(Exception-1 | Exception-2 | ……. | Exception-n e){
}
```

The above catch block is able to catch any of the specified exceptions.

In the Multi-Catch block, all the provided exception classes must not have inheritance relation, if inheritance relation is available between exception classes then the compiler will raise an error.

EX:

```
import java.util.Date;

public class Main {
   public static void main(String[] args) {
       try{
           //float f = 100/0;
           /*
           Date d = null;
           System.out.println(d.toString());
            */
           int[] a = {10,20,30,40};
           System.out.println(a[10]);
       }catch(ArithmeticException | NullPointerException |
ArrayIndexOutOfBoundsException e){
           e.printStackTrace();
       }
   }
}
```

EX:

```
import java.util.Date;

public class Main {
   public static void main(String[] args) {
       try{

       }catch(ArithmeticException | RuntimeException | Exception e){
           e.printStackTrace();
       }
   }
}
```

Status: Compilation Error.

try-with-resources:

In general, in java applications, we may use a number of resources like Streams, database Connections, Network Connections,......

If we use the above resources in java applications , these resources may generate some exceptions. To handle these exceptions if we use try-catch-finally then we have to use the following conventions.

1. Declare the resources before try block.
2. Create the resources inside the try block.
3. Close the resources inside the finally block.

```
FileInputStream fis = null;
Connection con = null;
URLConnection urlConn = null;
try{
      fis = new FileinputStream(“abc.txt”);
      con = DriverManager.getConnection(“---”,”--”,”---”);
      urlConn = socket.openConnection();
}catch(Exception e){
      e.printStacktrace();
}finally{
      try{
            fis.close();
            con.close();
            urlConn.close();
      }catch(Exception e){
            e.printStacktrace();
      }
}
```

With the above approach, we are able to get the following problems.

1. We must close the resources explicitly, which may not be guaranteed.

2. Providing try-catch-finally inside the finally block is somewhat confusing.

To overcome the above problems we have to use try-with-resources .

Syntax:
```
try(Resource-1; Resource-2;...... Resource-n;){
}catch(Exception e){
}
```

Where all the resources must implement java.lang.AutoCloseable marker interface.

In the try-with-resources, all the resources will be closed automatically when flow of execution is coming out from the try block.

EX:
```
try(
FileInputSteam fis = new FileInputStream("abc.txt");
Connection con = DriverManager.getConnection("--","--","--");
URLConnection uc = socket.openConnection();
){
----
-----
}catch(Exception e){
      e.printStackTrace();
}
```

==