Exception Handling

Click to Continue

How can Exception be handled?

Exception can be handled using a **exception handler**.

What is a Exception Handler?

A set of code which can handle an error conditions in a program systematically by taking necessary action

Exception Handling Techniques:

- **Option I:** try-catch-finally
- **Option II:** throws.

Exception Handler

Click to Continue

Try-Catch:

When a program performs an operation that causes an exception, an **exception** will be **thrown**. Exception can be handled by using the **try** and **catch** blocks.

How is it implemented?

- The suspected code is embraced in the **try** block, followed by the catch block to handle the exception.
- In the catch block, the programmer can also write the code to recover from the exception and can also print the exception.
- A catch block is executed only if it catches the exception coming from within the corresponding try block.

Finally Block

Click to Continue

Finally block is the last block of the exception handling. It comes after all the catch blocks.

- The finally block will execute whether or not an exception is thrown.
- The finally block is optional.
- If a finally block is associated with a try, the finally block will be executed upon conclusion of the try
- Finally block can be useful for freeing up resources that might have been allocated in the method.
- A try can contain multiple catch block , but only one finally block.

Exception Handling Syntax

Click to Continue

Syntax:

```
try{

     // The code that is prone to throw exception
}
catch(Exception exception){
     // The thrown exception is handled
}
finally{
     //Any logic which needs to be executed irrespective of
     exception thrown or not
}
```

Exception Handling Syntax

Click to Continue

Syntax:

```
try{

    // The co
}
catch(Excep
    // The thr
}
finally{
    //Any log
    exceptio
}
```

```
public void divide(int a,int b){
        int quotient=0;
        try{
                quotient=a/b;
        }
        catch(ArithmeticException exception){
                System.out.println("Exception Occurred" +exception.getMessage());
        }
        finally{
                System.out.println("The quotient is "+quotient);
        }
}
```

Here, there is a possibility that an Arithmetic Exception can be thrown when b = 0.

Exception Handling Syntax

Click to Continue

Syntax:

try{

// The co

}

catch(Excep

// The th

}

finally{

//Any log

exceptio

}

```
public void divide(int a,int b){
        int quotient=0;
        try{
                quotient=a/b;
        }
        catch(ArithmeticException exception){
                System.out.println("Exception Occurred" +exception.getMessage());
        }
        finally{
                System.out.println("The quotient is "+quo
        }
}
```

Here, there is a possibility that an Arithmetic Exception can be thrown when b = 0.

The catch block is called when Arithmetic exception occurs. Here we are printing the error message

Exception Handling Syntax

Click to Continue

Syntax:

try{

// The co

}

catch(Excep

// The thr

}

finally{

//Any log

exceptio

}

```
public void divide(int a,int b){
        int quotient=0;
        try{
                quotient=a/b;
        }
        catch(ArithmeticException exception){
                System.out.println("Exception Occurred" +exception.getMessage());
        }
        finally{
                System.out.println("The quotient is "+quo
        }
}
```

Here, there is a possibility that an Arithmetic Exception can be thrown when b = 0.

The catch block is called when Arithmetic exception occurs. Here we are printing the error message

This statement will execute irrespective of the exception being thrown or not

Try catch exception flow

Click to Continue

```
public divide(int dividend, int divisor){
    try{
        result = dividend/ divisor;
        // other statements..
    }
    Catch(ExceptionObject)
    {
        // Exception handled
    }
        // other statements..
}
```

When divisor is zero, exception raised.

Exception is Caught

Exception is handled

Other statements till the end of the method will be triggered.

Try catch finally exception flow

Click to Continue

```
public divide(int dividend, int divisor){
    try{
        result = dividend/ divisor;
        // other statements..
    }
    Catch(ExceptionObject)
    {
        // Exception handled
    }
    finally
    {
        //some logic
    }
        // other statements..
}
```

- `result = dividend/ divisor;` — If divisor is zero exception raised.
- `Catch(ExceptionObject)` — Exception is caught
- `// Exception handled` — Exception is handled
- `//some logic` — Finally block statements invoked
- `// other statements..` — Other statements till the end of the method will be trigerred.

Try catch exception flow – NO exception

Click to Continue

```
public divide(int dividend, int divisor){
      try{
            result = dividend/ divisor;
            // other statements..
      }
      Catch(ExceptionObject)
      {
            // Exception handled
      }
      finally
      {
            //some logic
      }
            // other statements..
}
```

If NO exception raised in the method

Finally block statements invoked

Other statements till the end of the method will be trigerred.

Multiple Catch Blocks

Click to Continue

Multiple catch Blocks: If block of code can generate different types of exception. Then all the exceptions need to caught.

Example:

```
try {
    int den = Integer.parseInt(args[0]);
    System.out.println(3/den);
}
```

This can lead to **ArrayIndexOutOfBounds** Exception

This can lead to **ArithmeticException**

Handling Multiple Catch blocks:

```
try{
    int den = Integer.parseInt(args[0]);
    System.out.println(3/den);
}catch(ArrayIndexOutOfBoundsException ab){
    // Exception a handled here
}catch(Arithmetic Exception ar){
    // Exception b handled here
}
```

Here we have two catch blocks for each of the exceptions.
Based on the exception thrown in the try block, the appropriate catch block will be executed

Nested Try Blocks

Click to Continue

Nested try blocks can be used , here a try catch block is placed inside another try catch block.

Example of Nested Try block:

```
try{
        int den = Integer.parseInt(args[0]);
        try{
                System.out.println(3/den);
        } catch(ArithmeticException ar){
                // Exception a handled here
        }
}catch(ArrayIndexOutOfBoundsException ab){
        // Exception b handled here
}
```

This is the nested TRY. Arithmetic Exception thrown here will be caught inside this block itself.

NOTE: If the inner try does not have a matching catch statement for a particular exception, the control is transferred to the outer try statement's catch handlers where it searches for a matching catch statement

Rules For Try catch finally

Click to Continue

Rules for writing the try-catch-finally:

- The try block must be followed by either a catch block or a finally block, or both.
- The try block alone is not complete. It should either have a catch or finally block.
- Any catch block must immediately follow a try block. Cannot exist alone,
- The finally block must immediately follow the last catch block (or) the try block if there is no catch block.

Try it out – Try/Catch, throws

Click to Continue

1: Create a class, ***ExceptionDemo*** with a method ***divide*** with two int parameters

- Dividend
- Divisor

▪This method should divide the dividend by divisor and return the result.

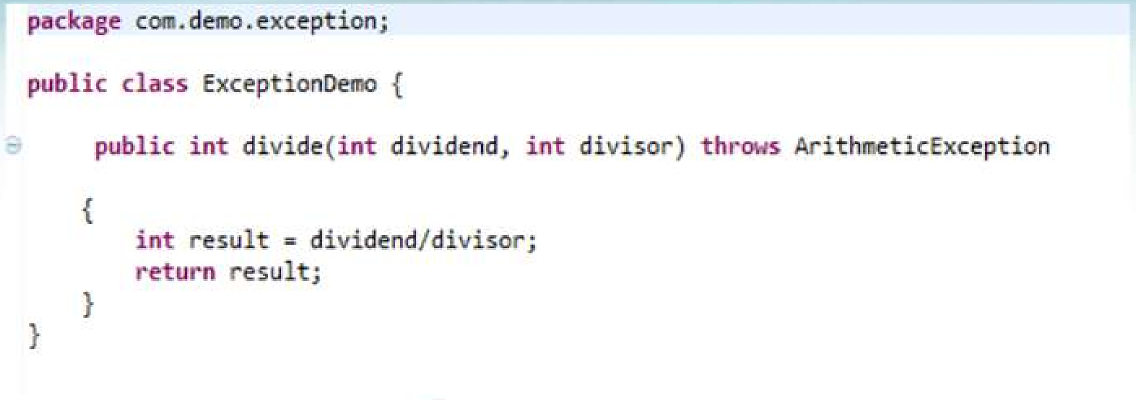
▪This method should also throw an ***ArithmeticException*** to the calling method.

2: Create a class, ***ThrowsDemo*** with a main method

- The main method should invoke the ***divide*** method in ***ExceptionDemo*** class.
- The main method should handle the exception by ***catching*** the ***ArithmeticException*** and print the Exception "Arithmetic Exception Thrown"
- Add a finally block which prints a message " The result is" <Result> where result is the value returned.

Try it out - solution

Click to Continue



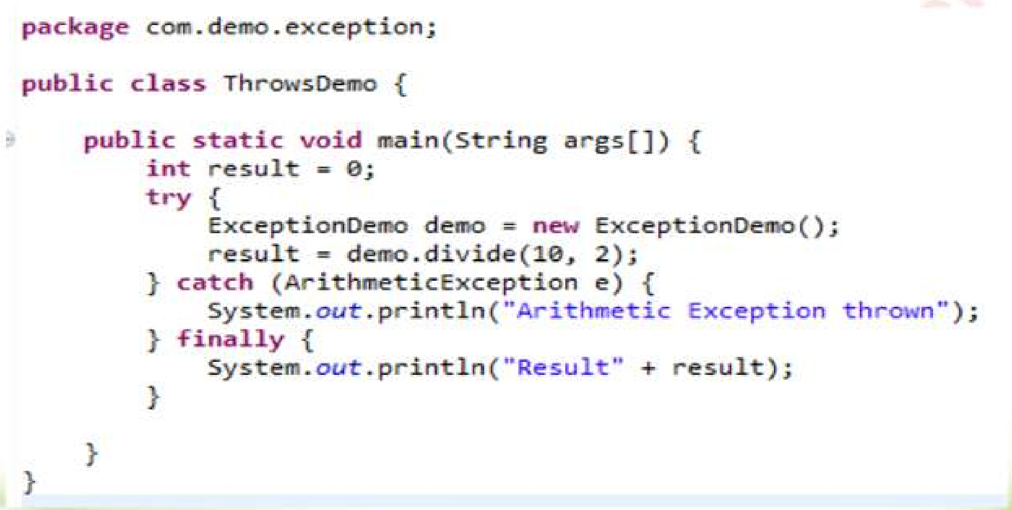
```
package com.demo.exception;

public class ExceptionDemo {

     public int divide(int dividend, int divisor) throws ArithmeticException

    {
        int result = dividend/divisor;
        return result;
    }
}
```

Execute the program for two values of divisor, 2 and zero.



```
package com.demo.exception;

public class ThrowsDemo {

    public static void main(String args[]) {
        int result = 0;
        try {
            ExceptionDemo demo = new ExceptionDemo();
            result = demo.divide(10, 2);
        } catch (ArithmeticException e) {
            System.out.println("Arithmetic Exception thrown");
        } finally {
            System.out.println("Result" + result);
        }

    }
}
```

Option II - throws

Click to Continue

Throws keyword is used to throw exception object from a method to the calling method.

In java , this delegation is done by using the ***throw*** clause.

The exception thrown by the method needs to be handled by the calling method.

Syntax of throws

Click to Continue

Syntax:

```
 <access specifier><return type><method name>() throws Exception-list
{
        //some code here which can throw
        //any type of exception specified in Exception-list
}
```

Exception-list is a comma-separated list of the exceptions that a method can throw.

The method invoking this method should handle the exceptions in the exception list using try catch block.