Lab 10
Exception Handling

Objective:

1. What & Why Exception Handling
2. Try, catch, finally and throw keyword.

1: Introduction of Exception Handling

There are two types of errors that occurs during the development process

1. **Compile time errors**

   Also called as Syntax errors

   Example: Instead of declaring `int a;` you mistakenly declared it as `in a;` for which compiler will throw an error.

2. **Runtime errors**

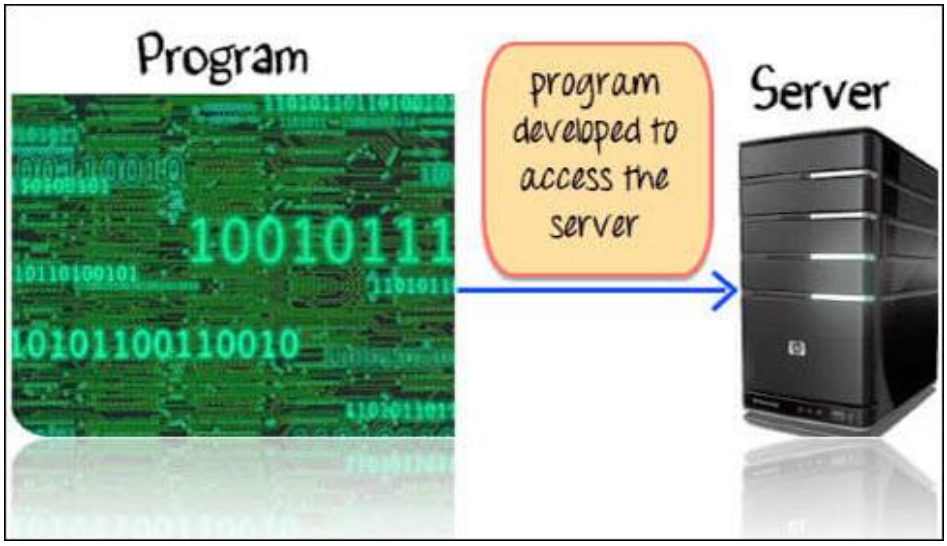
   A Runtime error is called an **Exception** error. It is any event that interrupts the normal flow of program execution.
   Example for exceptions are; arithmetic exception, Nullpointer exception, Divide by zero exception, stack overflow etc.

What is an Exception?

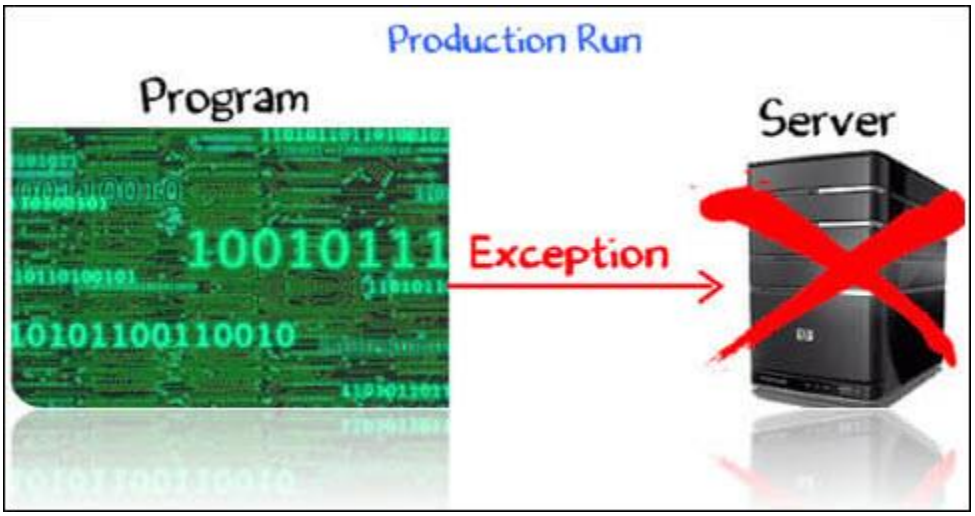
In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime by interpreter. In such cases we get a system generated error message. The good thing about exceptions is that they can be handled in Java

Why do we need Exception?

Suppose you have coded a program to access the server. Things worked fine while you were developing the code.



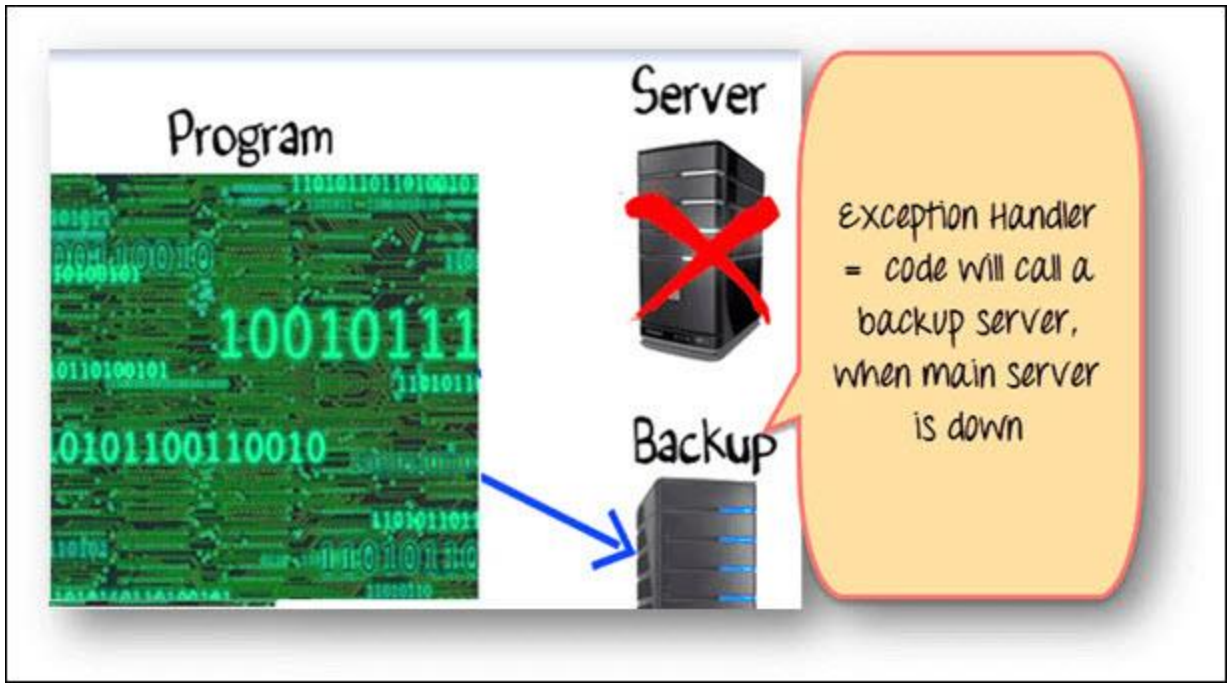
During the actual production run, the server is down. When your program tried to access it, an exception is raised.



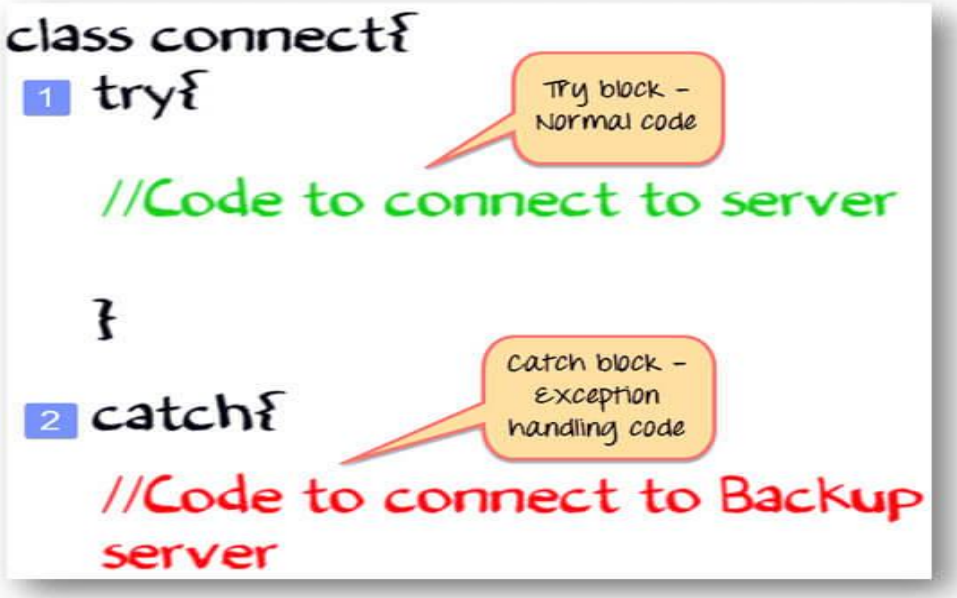
How to Handle Exception

A Robust Programming, which takes care of exceptional situations. Such code is known as Exception Handler. If an exception occurs, which has not been handled by programmer then program execution gets terminated and a system generated error message is shown to the user.

In this example, good exception handling would be, when the server is down, connect to the backup server.

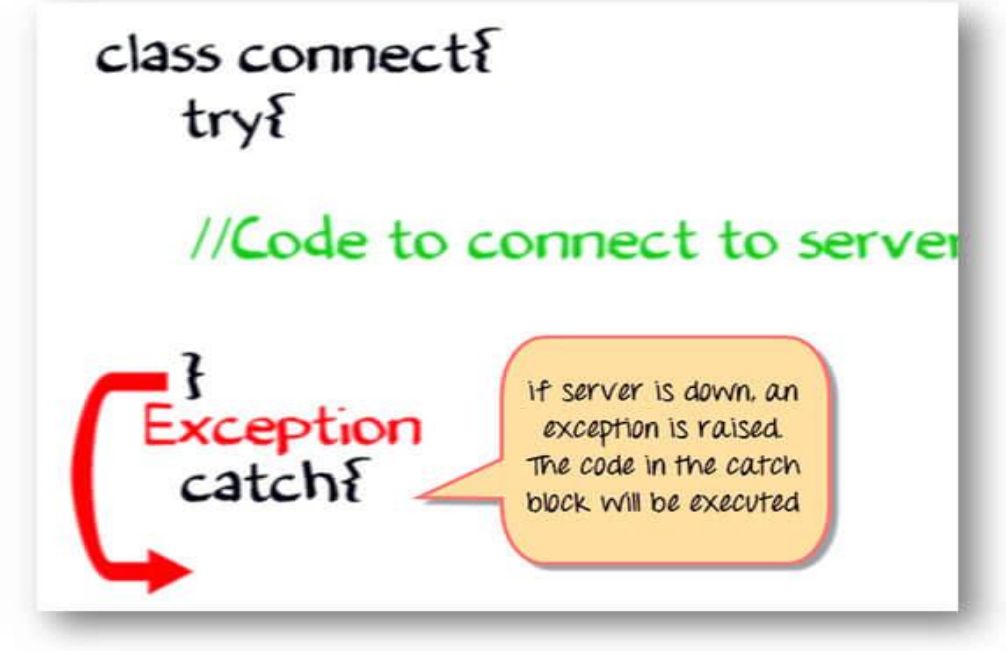


Try Catch Block Java provides an inbuilt exceptional handling. The normal code goes into a TRY block. The exception handling code goes into the CATCH block



In this example, TRY block will contain the code to connect to the server. CATCH block will contain the code to connect to the backup server.

In case the server is up, the code in the CATCH block will be ignored. In case the server is down, an exception is raised, and the code in catch block will be executed.



Syntax for using try & catch

```
try{
    statement(s)
}
catch (exceptiontype name){
        statement(s)
}
```

Example 1

Step 1) Copy the following code into notepad

```
class JavaException {
   public static void main(String args[]){
      int d = 0;
      int n = 20;
      int fraction = n/d;
     System.out.println("End Of Main");
   }
```

```
}
```

Step 2) Save the file & compile the code. Run the program using command, java JavaException

Step 3) An Arithmetic Exception - divide by zero is shown as below for line # 5 and line # 6 is never executed

Step 4) Now let's examine how try and catch will help us to handle this exception. We will put the exception causing the line of code into a **try** block, followed by a **catch** block. Copy the following code into the editor.

```
class JavaException {
 public static void main(String args[]) {
  int d = 0;
  int n = 20;
  try {
   int fraction = n / d;
   System.out.println("This line will not be Executed");
  } catch (ArithmeticException e) {
   System.out.println("In the catch Block due to Exception = " + e);
  }
  System.out.println("End Of Main");
 }
}
```

Step 5) Save, Compile & Run the code. You will get the following output

```
C:\workspace>java JavaException
In the catch clock due to Exception = java.lang.ArithmeticException: / by zero
End Of Main
```

As you observe, the exception is handled, and the last line of code is also executed. Also, note that Line #7 will not be executed because **as soon as an exception is raised the flow of control jumps to the catch block.**

Note: The AritmeticException Object "e" carries information about the exception that has occurred which can be useful in taking recovery actions.

Example 2

Step 1) Copy the following code into notepad

```
class JavaException {
 public static void main(String args[]) {
  try {
   int d = 1;
   int n = 20;
   int fraction = n / d;
   int g[] = {   1   };
   g[20] = 100;
  }
  /*catch(Exception e){
        System.out.println("In the catch block due to Exception = "+e);
  }*/
  catch (ArithmeticException e) {
   System.out.println("In the catch block due to Exception = " + e);
  } catch (ArrayIndexOutOfBoundsException e) {
   System.out.println("In the catch block due to Exception = " + e);
  }
  System.out.println("End Of Main");
 }
}
```

Step 2) Save the file & compile the code. Run the program using command, **java JavaException**.

Step 3) An ArrayIndexOutOfBoundsException is generated. Change the value of int d to 0. Save, Compile & Run the code.

Step 4) An ArithmeticException must be generated.

Step 5) Uncomment line #10 to line #12. Save, Compile & Run the code.

Step 6) Compilation Error? This is because Exception is the base class of ArithmeticException Exception. Any Exception that is raised by ArithmeticException can be handled by Exception class as well.So the catch block of ArithmeticException will never get a chance to be executed which makes it redundant. Hence the compilation error.

Java Finally Block

The finally block is executed irrespective of an exception being raised in the try block. It is optional to use with a try block.

```
try {
 statement(s)
} catch (ExceptiontType name) {

 statement(s)

} finally {

 statement(s)

}
```

In case, an exception is raised in the try block, finally block is executed after the catch block is executed.

Example 3

Step 1) Copy the following code into notepad

```
class JavaException {
   public static void main(String args[]){
    try{
       int d = 0;
       int n =20;
       int fraction = n/d;
    }
  catch(ArithmeticException e){
    System.out.println("In the catch block due to Exception = "+e);
  }
  finally{
        System.out.println("Inside the finally block");
  }
}
}
```

Step 2) Save, Compile & Run the Code.

Step 3) Expected output. Finally block is executed even though an exception is raised.

Step 4) Change the value of variable d = 1. Save, Compile and Run the code and observe the output.Bottom of Form

Java Throw keyword

If programmer do not handle the exception in a try catch block, compiling will fail. But almost every other method in the java library or even user defined may throw an exception or two.

So, java provides an option, wherein whenever you are using a risky piece of code in the method definition you declare it throws an exception without implementing try catch.

```
public class JavaException {
  static void checkAge(int age) {
    if (age < 18) {
      throw new ArithmeticException("Access denied - You must be at least 18 years
old.");
    }
    else {
      System.out.println("Access granted - You are old enough!");
    }
  }

  public static void main(String[] args) {
    checkAge(15); // Set age to 15 (which is below 18...)
  }
}
```

Java throws keyword

```
class JavaException {

        static void checkAge(int age) throws ArithmeticException{
            if (age < 18) {
                                System.out.println("Access denied!");

            }
            else {
                System.out.println("Access granted - You are old enough!");
            }
        }

        public static void main(String[] args) {

        checkAge(20);

    }

}
```

Lab Tasks

Marks: 10

Exercise 1

Write a program that can serve as a simple calculator. This calculator keeps track of a single number (of type double) that is called result and that starts out as 0.0 . Each cycle allows the user to repeatedly add, subtract, multiply, or divide by a second number. The result of one of these operations becomes the new value of result . The calculation ends when the user enters the letter R for "result" (either in upper- or lowercase). The user is allowed to do another calculation from the beginning as often as desired. The input format is shown in the following sample dialogue. If the user enters any operator symbol other than + , −, * , or / , then an UnknownOperatorException is thrown and the user is asked to reenter that line of input. Defining the class UnknownOperatorException is part of this project.

Sample output:

Calculator is on.

```
result = 0.0

+5

result + 5.0 = 5.0

new result = 5.0

* 2.2

result * 2.2 = 11.0

updated result = 11.0

% 10

% is an unknown operation.

Reenter, your last line:

* 0.1

result * 0.1 = 1.1

updated result = 1.1

r
```

```
Final result = 1.1

Again? (y/n)

yes

result = 0.0

+10

result + 10.0 = 10.0

new result = 10.0

/2

result / 2.0 = 5.0

r

Final result = 5.0

Again? (y/n)

N
```

End of Program

Exercise 2

Here is a snippet of code that inputs two integers and divides them:

```
Scanner scan = new Scanner(System.in);

int n1, n2;

double r;

n1 = scan.nextInt();

n2 = scan.nextInt();

r = ( double) n1 / n2;
```

Place this code into a try-catch block with multiple catches so that different error messages are printed if we attempt to divide by zero or if the user enters textual data instead of integers (`java.util.InputMismatchException`). If either of these conditions occurs, then the program should loop back and let the user enter new data.

Exercise 3

Modify the previous exercise so that the snippet of code is placed inside a method. The method should be named ReturnRatio, read the input from the keyboard, and throw different exceptions if there is a division by zero or an input mismatch between text and an integer. Create your own exception class for the case of division by zero. Invoke ReturnRatio from your main method and catch the exceptions in main. The main method should invoke the ReturnRatio method again if any exception occurs.

Post lab questions to ponder

1. Is it possible to have multiple try blocks with one catch block?
2. Is it possible to have try block without catch?

END