

Exception handling continued

In the previous session we saw 3 cases, here let's see the same 3 cases with the code.

Case1:

```
import java.util.Scanner;
class Demo
{
    public static void main(String[] args)
    {
        System.out.println("Connection is estd");
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the numerator:");
        int a = scan.nextInt();
        System.out.println("Enter the denominator:");
        int b = scan.nextInt();
        int c = a/b;
        System.out.println(c);
        System.out.println("Connection is terminated");
    }
}
```



Output:

```
Connection is estd
Enter the numerator:
100
Enter the denominator:
5
20
Connection is terminated
Press any key to continue . . .
```

This is the ideal case where without any mistakes the program is typed and the inputs given are also as expected by the programmer.

Case 2:

```
1 import java.util.Scanner;
2 class Demo
3 {
4     public static void main(String[] args)
5     {
6         System.out.println("Connection is estd");
7         Scanner scan = new Scanner(System.in);
8         System.out.println("Enter the numerator:");
9         int a = scan.nextInt();
10        System.out.println("Enter the denominator:");
11        int b = scan.nextInt();
12        int c = a/b; //--Exception would occur due to faulty input.
13        System.out.println(c);
14        System.out.println("Connection is terminated");
15    }
16 }
```

Output:



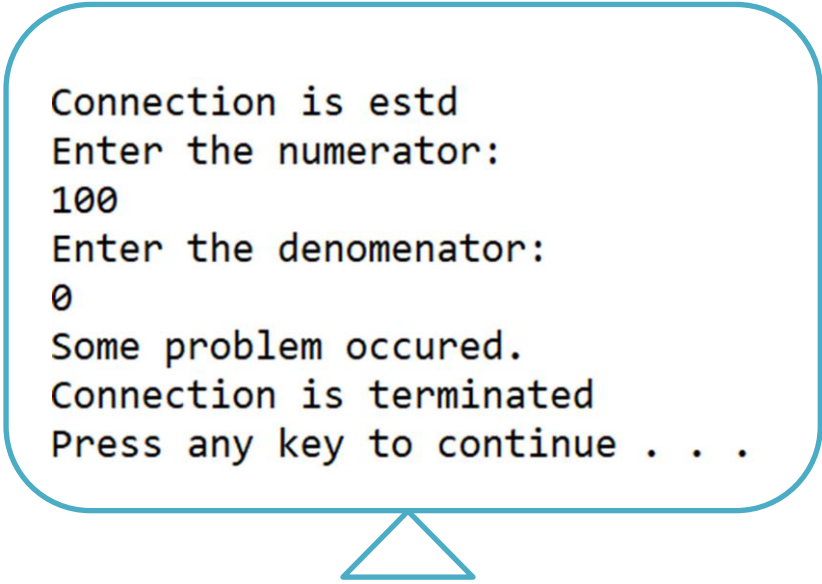
```
Connection is estd
Enter the numerator:
100
Enter the denominator:
0
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Demo.main(Demo.java:12)
Press any key to continue . . .
```

Here in the output we see that we have got an exception. This exception clearly states that it's arithmetic type exception which has occurred due to division by zero. And it is pointing to line 12. Note that the lines after 12th line did not execute, that's because the program got abruptly terminated when exception occurred and when the runtime system checked for a special code inside the program and didn't find any. Then the control was given to default exception handler and then the program was abruptly terminated without execution of the remaining lines.

Case 3:

```
import java.util.Scanner;
class Demo
{
    public static void main(String[] args)
    {
        System.out.println("Connection is estd");
        Scanner scan = new Scanner(System.in);
        try //lines which might generate exception are placed here
        {
            System.out.println("Enter the numerator:");
            int a = scan.nextInt();
            System.out.println("Enter the denominator:");
            int b = scan.nextInt();
            int c = a/b; //<--Exception would occur due to faulty input.
            System.out.println(c);
        }
        catch (Exception e) //If the error is occurred in try block it is caught here
        {
            System.out.println("Some problem occurred.");
        }
        System.out.println("Connection is terminated");
    }
}
```

Output:



```
Connection is estd
Enter the numerator:
100
Enter the denominator:
0
Some problem occurred.
Connection is terminated
Press any key to continue . . .
```

The **try** statement allows you to define a block of code to be tested for errors while it is being executed.

The **catch** statement allows you to define a block of code to be executed, if an error occurs in the try block. The **try** and **catch** keywords come in pairs

Providing try-catch is referred to as **user defined exception handler**. And prevents the program from abrupt termination.

Now let's see different types of Exception that can generate for the following example:

```
import java.util.Scanner;
class Demo
{
    public static void main(String[] args)
    {
        System.out.println("Connection is estd");
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the numerator:");
        int a = scan.nextInt();
        System.out.println("Enter the denominator:");
        int b = scan.nextInt();
        int c = a/b;
        System.out.println(c);

        System.out.println("Enter the size of the array:");
        int size = scan.nextInt();
        int arr[] = new int[size];
        System.out.println("Enter the element to be stored");
        int ele = scan.nextInt();
        System.out.println("Enter the index at which the element must be stored:");
        int index = scan.nextInt();
        arr[index] = ele;
        System.out.println(arr[index]);
        System.out.println("Connection is terminated");
    }
}
```

We have seen arithmetic exception in the previous example. Let's see some other exceptions

Output 1:

```
Connection is estd
Enter the numerator:
100
Enter the denominator:
2
50
Enter the size of the array:
-5
Exception in thread "main" java.lang.NegativeArraySizeException: -5
    at Demo.main(Demo.java:17)
Press any key to continue . . .
```

In the above output we see that the input given to an array size was negative. And when the exception object was given to the default exception handler it clearly showed type of exception created is **NegativeArraySizeException** and where is it pointing to.

Output 2:

```
Connection is estd
Enter the numerator:
100
Enter the denominator:
2
50
Enter the size of the array:
5
Enter the element to be stored
false
Exception in thread "main" java.util.InputMismatchException
    at java.base/java.util.Scanner.throwFor(Scanner.java:939)
    at java.base/java.util.Scanner.next(Scanner.java:1594)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
    at Demo.main(Demo.java:19)
Press any key to continue . . .
```

In the above output we see that the input given to the element stored is a string but the program expects integer type. And when the exception object was given to the default exception handler it clearly showed type of exception created is **InputMismatchException** and where is it pointing to.

Output3:

In the below output we see that the index at which the element must be stored is exceeding the size of array. And when executed the exception object was given to the default exception handler it clearly showed type of exception created is **ArrayIndexOutOfBoundsException** and where is it pointing to.

```
Connection is estd
Enter the numerator:
100
Enter the denominator:
2
50
Enter the size of the array:
5
Enter the element to be stored
99
Enter the index at which the element must be stored:
9
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 9 out of bounds for length 5
    at Demo.main(Demo.java:22)
Press any key to continue . . .
```



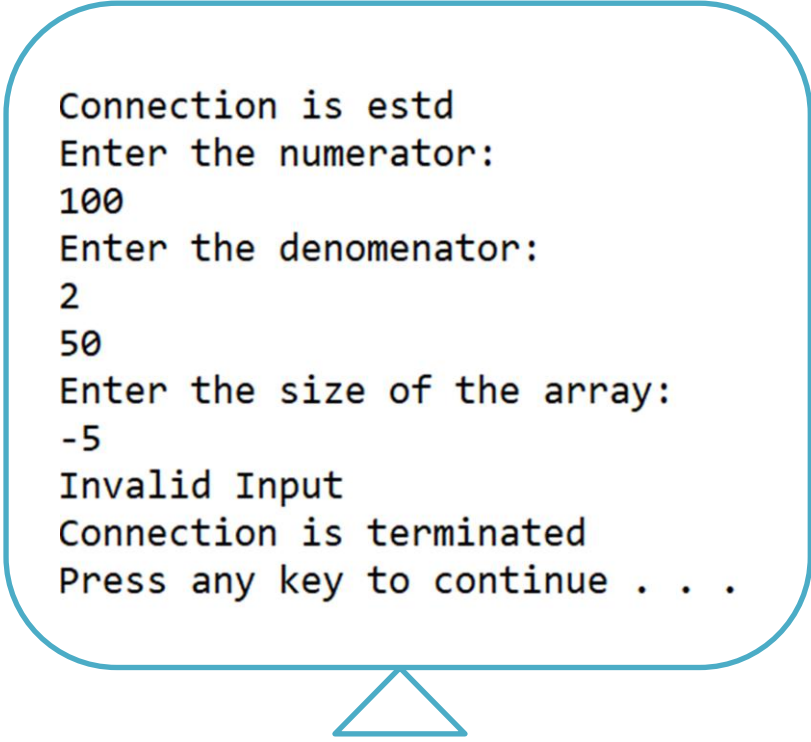
So now let us put all the statements that might create exception in try-catch block

```
1 import java.util.Scanner;
2 class Demo
3 {
4     public static void main(String[] args)
5     {
6         System.out.println("Connection is estd");
7         Scanner scan = new Scanner(System.in);
8         try
9         {
10            System.out.println("Enter the numerator:");
11            int a = scan.nextInt();
12            System.out.println("Enter the denominator:");
13            int b = scan.nextInt();
14            int c = a/b;
15            System.out.println(c);
16            System.out.println("Enter the size of the array:");
17            int size = scan.nextInt();
18            int arr[] = new int[size];
19            System.out.println("Enter the element to be stored");
20            int ele = scan.nextInt();
21            System.out.println("Enter the index at which the element must be stored:");
22            int index = scan.nextInt();
23            arr[index] = ele;
24            System.out.println(arr[index]);
25        }
26        catch (Exception e)
27        {
28            System.out.println("Invalid Input");
29        }
30        System.out.println("Connection is terminated");
31    }
32 }
```

Now if we give the same inputs then exception object generated will not be given to default exception handler instead it would be given to user defined exception handler.

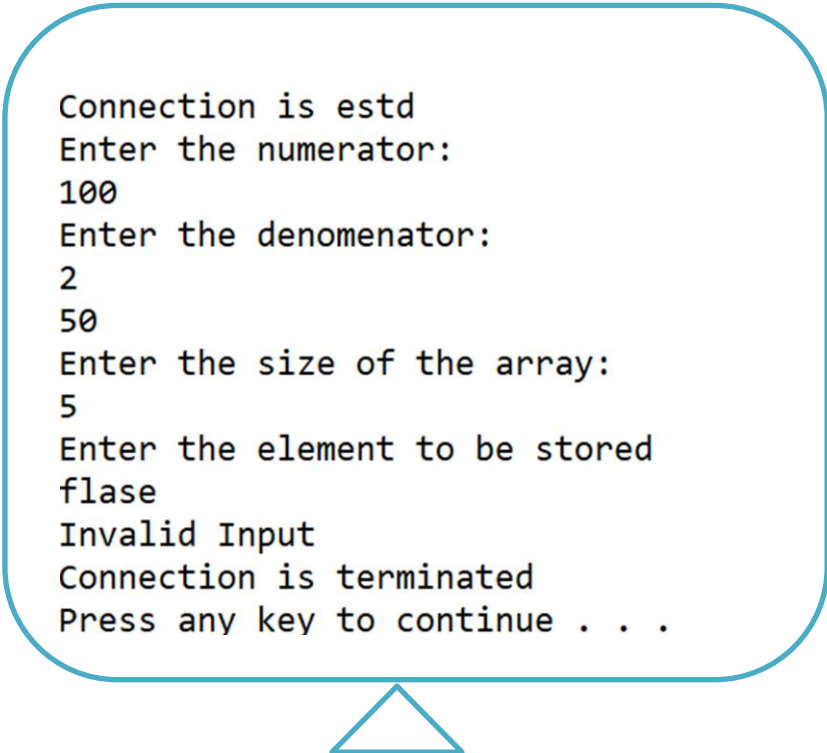
Let's see the outputs with the same inputs given earlier.

Output1:



```
Connection is estd
Enter the numerator:
100
Enter the denomenator:
2
50
Enter the size of the array:
-5
Invalid Input
Connection is terminated
Press any key to continue . . .
```

Output2:



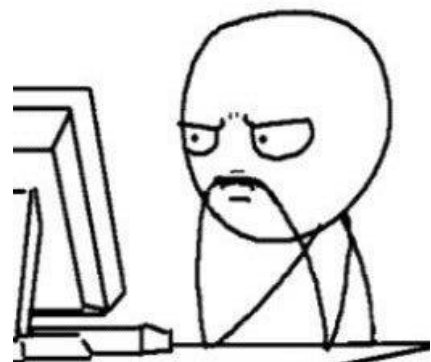
```
Connection is estd
Enter the numerator:
100
Enter the denomenator:
2
50
Enter the size of the array:
5
Enter the element to be stored
flase
Invalid Input
Connection is terminated
Press any key to continue . . .
```

Output3:

```
Connection is estd
Enter the numerator:
100
Enter the denominator:
2
50
Enter the size of the array:
5
Enter the element to be stored
99
Enter the index at which the element must be stored:
9
Invalid Input
Connection is terminated
Press any key to continue . . .
```

In all the above outputs we saw that there was no abrupt termination, all lines were executed and program got terminated. But the disadvantage here was every exception object was going to the same exception handler and the type of exception occurring was not clear. Which can be overcome by using multiple exception handler or in other words multiple catch blocks.

Below is how multiple catch blocks to be used to catch multiple exceptions.




```

catch (ArithmeticException ae)
{
    System.out.println("Enter non zero denominator");
}
catch (NegativeArraySizeException nas)
{
    System.out.println("Enter a positive size");
}
catch (InputMismatchException ime)
{
    System.out.println("Enter the valid input");
}
catch (ArrayIndexOutOfBoundsException ao)
{
    System.out.println("Enter the elements within limit");
}
catch (Exception e)
{
    System.out.println("Some problem occurred");
}
System.out.println("Connection is terminated");

```

Now whenever the exception object is generated it checks for all the exception handler or all the catch blocks and the respective message is displayed.

If by chance an exception occurred but it doesn't belong to any of the above mentioned exceptions then it goes to general exception handler.

