

Programming in Java
Prof. Debasis Samanta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 26
Demonstration – X

So, let us have a demonstration regarding the Exception Handling Mechanism in Java.

(Refer Slide Time: 00:25)

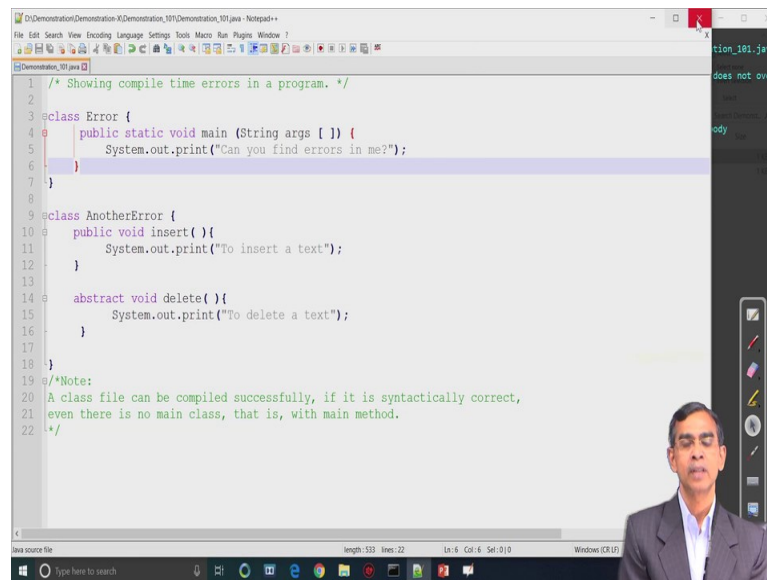
In today's demonstration ...

1. About compile-time error
2. About run-time error
3. Simple Try-Catch block
4. Try with multiple Catch
5. Multiple errors with single catch
6. Finally in try-catch block
7. Exception handling using Throws statement
8. Nested try-catch block

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES** | **DEBASIS SAMANTA**
CSE
IIT KHARAGPUR

And today's demonstration includes mainly the compile-time errors, and then what are the run time errors are there, and then how exception handling mechanism can be achieved by means of simple try-catch block, try with multiple catch, then multiple error errors with single catch. And there is another block that can be added with the try-catch is called the finally, so try-catch-finally block in along with the try-catch block, and then throw and then throw used in exception handling and then nested try-catch block also.

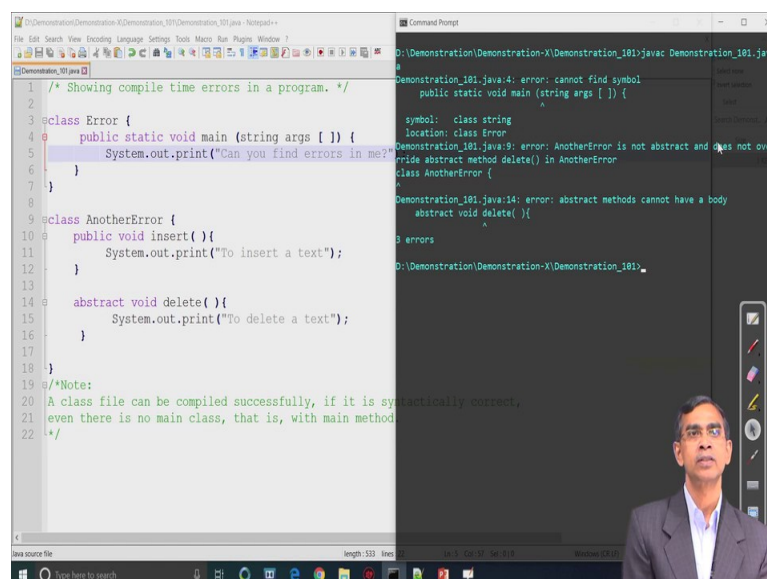
(Refer Slide Time: 01:08)



```
1  /* Showing compile time errors in a program. */
2
3  class Error {
4      public static void main (String args [ ]) {
5          System.out.print("Can you find errors in me?");
6      }
7  }
8
9  class AnotherError {
10     public void insert() {
11         System.out.print("To insert a text");
12     }
13
14     abstract void delete() {
15         System.out.print("To delete a text");
16     }
17 }
18
19 /*Note:
20 A class file can be compiled successfully, if it is syntactically correct,
21 even there is no main class, that is, with main method.
22 */
```

So, let us have the first look at the compile-time error. As you know if you run the program which is not as per the syntax of java programming language. So, the job java compiler reports all these things as an error as we see here in this program here. We can see a lot of errors are there and here we can see in the first statement itself start with error because java is a case sensitive. So, capital C in the class declaration key work is not allowed.

(Refer Slide Time: 01:38)



```
1  /* Showing compile time errors in a program. */
2
3  class Error {
4      public static void main (string args [ ]) {
5          System.out.print("Can you find errors in me?");
6      }
7  }
8
9  class AnotherError {
10     public void insert() {
11         System.out.print("To insert a text");
12     }
13
14     abstract void delete() {
15         System.out.print("To delete a text");
16     }
17 }
18
19 /*Note:
20 A class file can be compiled successfully, if it is syntactically correct,
21 even there is no main class, that is, with main method
22 */
```

```
D:\Demonstration\Demonstration-X\Demonstration_101>javac Demonstration_101.java
Demonstration_101.java:4: error: cannot find symbol
    public static void main (string args [ ]) {
                           ^
    symbol:   class string
    location: class Error
Demonstration_101.java:9: error: AnotherError is not abstract and does not override abstract method delete() in AnotherError
    class AnotherError {
    ^
Demonstration_101.java:14: error: abstract methods cannot have a body
    abstract void delete() {
    ^
3 errors
D:\Demonstration\Demonstration-X\Demonstration_101>
```

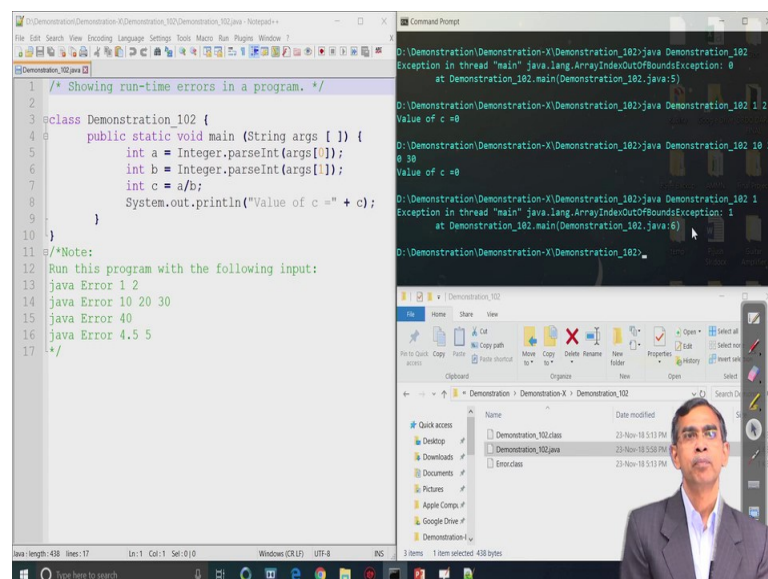
So, if we run this program as it is the first execution, so compiler will halt it here showing this error there. So, we will correct one error at a time as we see the compiler will report the class error here.

Now, let us quickly remove this error now we can see there is more error and public also capital P it is not allowed, so let us small capital is there and then again the error is there, ok. Let us quick to fix it, right. Public and then the system as we see capital S should be there and then capital S, and then you see all the statement should be terminative semicolon. So, as there is no semicolon, so we should make the semicolon here again put the semicolon as we see. So, java compile compiler will compile the program and whenever it finds an error it will report it and then until this report is fixed, so this program will not lead to the successful execution of the program.

So, there are a few more errors as we see. Any method should not be declared and abstract inside a class method as we can see here because it is not an abstract class. So, there is an error, and then there are few more error as we see those basically compile-time errors actually.

Now, so after knowing this compile-time error let us have the run time error concept it is there. Our next illustration to explain the idea is about the run time error. So, let us go to demonstration 10.2. Here we can declare on class, the name of the class is error and ok, right.

(Refer Slide Time: 03:22)



The screenshot displays a Java IDE on the left and a Command Prompt on the right. The IDE shows a file named `Demonstration_102.java` with the following code:

```
1  /* Showing run-time errors in a program. */
2
3  class Demonstration_102 {
4      public static void main (String args [ ]) {
5          int a = Integer.parseInt(args[0]);
6          int b = Integer.parseInt(args[1]);
7          int c = a/b;
8          System.out.println("Value of c =" + c);
9      }
10 }
11
12 /*Note:
13 Run this program with the following input:
14 java Error 1 2
15 java Error 10 20 30
16 java Error 40
17 java Error 4.5 5
18 */
```

The Command Prompt shows the output of running the program. It displays several `java.lang.ArrayIndexOutOfBoundsException` errors, indicating that the program is crashing due to invalid array indices. The errors are as follows:

```
D:\Demonstration\Demonstration-X\Demonstration_102>java Demonstration_102
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
    at Demonstration_102.main(Demonstration_102.java:5)

D:\Demonstration\Demonstration-X\Demonstration_102>java Demonstration_102 1 2
Value of c =0

D:\Demonstration\Demonstration-X\Demonstration_102>java Demonstration_102 10 20 30
0 30
Value of c =0

D:\Demonstration\Demonstration-X\Demonstration_102>java Demonstration_102 1
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 1
    at Demonstration_102.main(Demonstration_102.java:6)

D:\Demonstration\Demonstration-X\Demonstration_102>
```

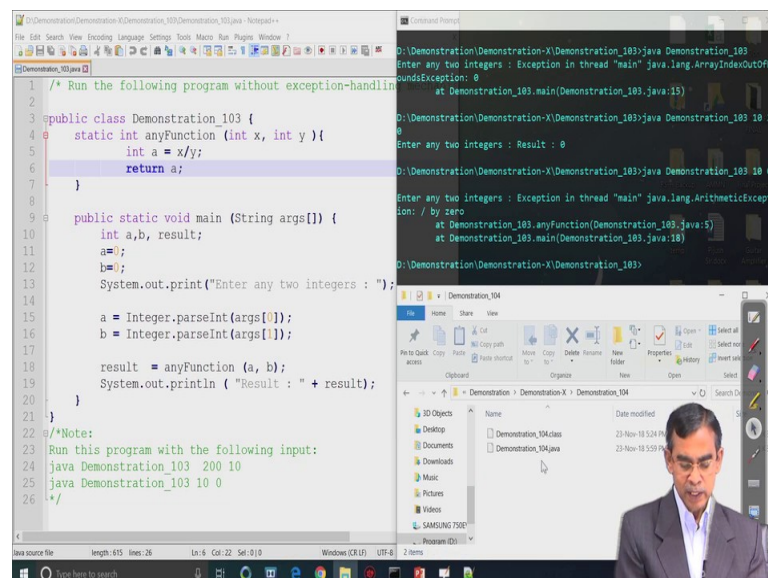
The IDE's file explorer on the right shows the project structure, including `Demonstration_102.class` and `Demonstration_102.java`.

As we see this is the one class declaration, let us follow let us have the quick look at the class declaration here. Now here we can see this program is perfectly there is no compilation error, as we have compiled it successfully; however, this program will run but will not run for all. Now, if we run with this one we will see it run for all not for all, but for some.

For example, for if we run this program with input 1 2 it will run, input 1 2 give, it runs correctly. Now, 10 20 30, as we see these program will gives an, it is 10 30; 10 20 30. So, this program also takes this input. Now, what will happen if we give only a single output? As we see it also ok. So, that is a single input, in this case, it gives an error. So, this error is basically array index out of bound exception array error it is there.

Now, again four 4.5 and 5 the different error it will basically it is called the exception. Also, we can see the number format exception error it is there. Now, we will see these are not actually errors they are called exceptions because in some situation this program not able to handle this, and this so program is there is no error in the program rather these programs cannot handle all input that is the cases of exception that is why they are called exception.

(Refer Slide Time: 05:10)



The screenshot displays a Java IDE with two windows. The left window shows the source code for `Demonstration_103.java`. The code defines a class `Demonstration_103` with a static method `anyFunction` that takes two integers `x` and `y` and returns `x/y`. The `main` method prompts the user to enter two integers, parses them using `Integer.parseInt`, and then calls `anyFunction` to calculate the result. A comment at the bottom instructs the user to run the program with inputs `200 10` and `10 0`. The right window shows the program's execution output. It displays the prompt "Enter any two integers : " and the result "Result : 20" for the input "200 10". For the input "10 0", it shows an "Exception in thread 'main' java.lang.ArithmeticException: / by zero" at line 18 of `Demonstration_103.java`. A file explorer window in the foreground shows the project structure with `Demonstration_104` and `Demonstration_104.java` files.

```
1 /* Run the following program without exception-handling
2
3 public class Demonstration_103 {
4     static int anyFunction (int x, int y) {
5         int a = x/y;
6         return a;
7     }
8
9
10    public static void main (String args[]) {
11        int a,b, result;
12        a=0;
13        b=0;
14        System.out.print("Enter any two integers : ");
15
16        a = Integer.parseInt(args[0]);
17        b = Integer.parseInt(args[1]);
18        result = anyFunction (a, b);
19        System.out.println ( "Result : " + result);
20    }
21 }
22
23 /*Note:
24 Run this program with the following input:
25 java Demonstration_103 200 10
26 java Demonstration_103 10 0
27 */
```

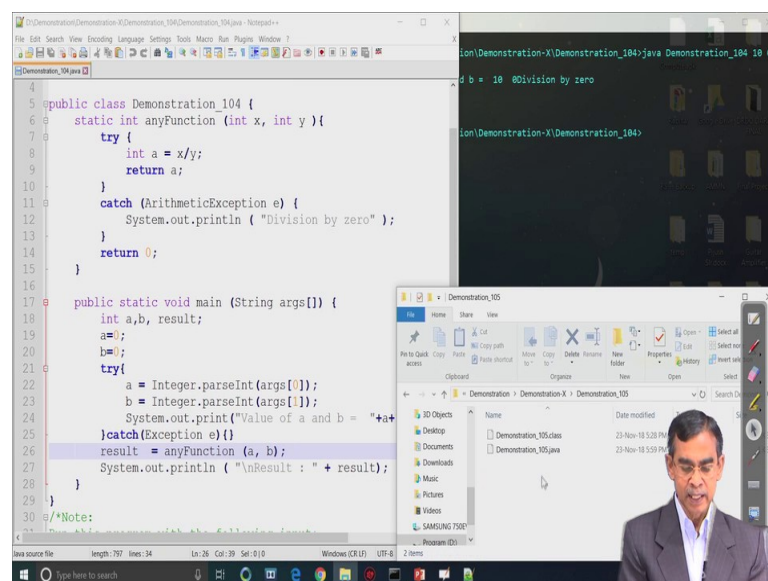
Enter any two integers : 200 10
Result : 20
Enter any two integers : 10 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
at Demonstration_103.anyFunction(Demonstration_103.java:5)
at Demonstration_103.main(Demonstration_103.java:18)

Now, let us see how we can make our program robust using the exception handling mechanism it is there. So, let us have the first program there. As we see this program 10.3, this program we see there are lot of exception scope of exception as we see for

example, in statement x; display, display yeah, ok. So, here we can see in that class demonstration 103 we see a equals to x divided by y. So, there is a scope of an exception to occur because if y is 0; that means, any function is called with two arguments with the second argument is 0 and then again there is a possible job exception in integer.percent in case suppose if we run this with non integer parameter passed through there right.

Now, anyway. So, this program has that it is, it is if we compile it or definitely there is no compile-time error this means the program does not have any error there. So, if we run it for two it is no error, but if we run with see first argument other than 0 then, ok; 10 and 0 for example, as we see here two inputs one is 10 and another is 0, because second input if it is 0 it will basically divide by 0 or arithmetic exception as we see here arithmetic exception error occurs here. So, this is a case of exception that we can see and we see this program is therefore, not so robust.

(Refer Slide Time: 07:10)

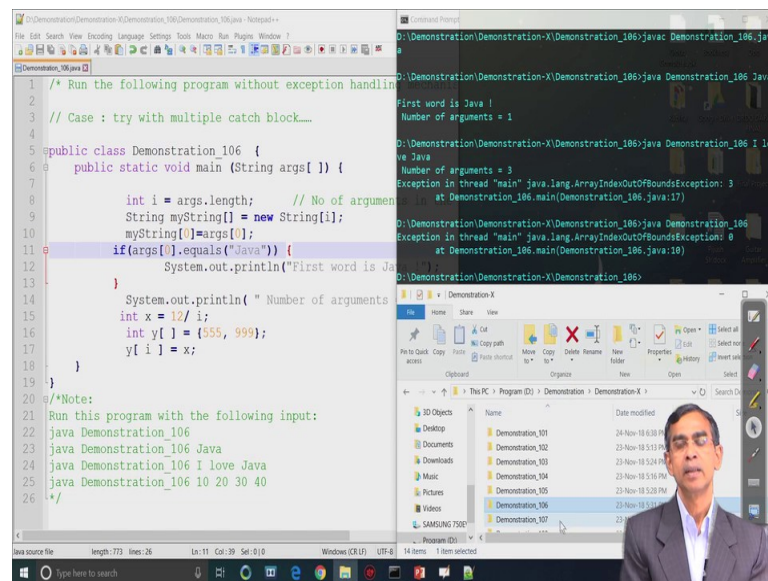


Now, towards the implementation robust program, we can use simple try-catch. So, this is the example to show how the simple try-catch can make the program robust.

So, here the point of exception where we just enclose it in the try-catch block, and there is another source of exception there a equals to integer.percent they are also a source of exception there. So, we have just put them in the try-catch. Now, this program will run for any input whatever the input that was not working in for the previous case.

Now, let us run this program with inputs 10 and 0 again. As it was not working there in this case you see it works there and divide by 0 it reported and then result in 0 returns this one, ok. So, this way we can; so in previous cases the program was not program was abnormally terminated rather, but in this case program run successfully, but exception which occurs it caught and then handle it.

(Refer Slide Time: 08:12)



The screenshot displays a Java IDE with two windows. The left window shows the source code for `Demonstration_106.java`. The code includes a `main` method that takes `String args[]` and performs several operations: it checks the number of arguments, prints the first word, calculates the number of arguments, and performs a division by `i`. A comment indicates that the program should be run without exception handling. The right window shows the output of the program, which includes the text "First word is Java", "Number of arguments = 1", "Number of arguments = 3", and several `java.lang.ArrayIndexOutOfBoundsException` errors. A file explorer window is also visible in the background, showing a directory structure with files named `Demonstration_101` through `Demonstration_107`.

```
1 /* Run the following program without exception handling
2 // Case : try with multiple catch block...
3
4
5 public class Demonstration_106 {
6     public static void main (String args[] ) {
7
8         int i = args.length; // No of arguments
9         String myString[] = new String[i];
10        myString[0]=args[0];
11        if(args[0].equals("Java")) {
12            System.out.println("First word is Java");
13        }
14        System.out.println(" Number of arguments");
15        int x = 10 / i;
16        int y[] = {555, 999};
17        y[i] = x;
18    }
19 }
20
21 /*Note:
22 Run this program with the following input:
23 java Demonstration_106
24 java Demonstration_106 I love Java
25 java Demonstration_106 10 20 30 40
26 */
```

Output:

```
D:\Demonstration\Demonstration-X\Demonstration_106>java Demonstration_106
First word is Java
Number of arguments = 1
D:\Demonstration\Demonstration-X\Demonstration_106>java Demonstration_106 I lo
ve Java
Number of arguments = 3
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
at Demonstration_106.main(Demonstration_106.java:17)
D:\Demonstration\Demonstration-X\Demonstration_106>java Demonstration_106
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 8
at Demonstration_106.main(Demonstration_106.java:10)
D:\Demonstration\Demonstration-X\Demonstration_106>
```

Now, the next example is basically, so how the robust program can be developed. Again the same thing, if we run the different cases we can skip let us go to the 10.6 runs the following program, run the program without exception handling mechanism for some input.

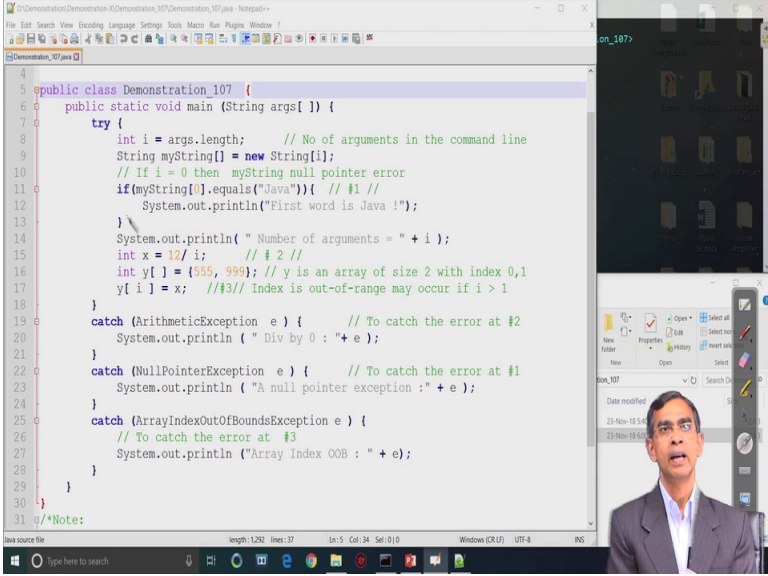
Let us see have the 106 programs there, yeah. 10.6 program as we see here this program includes again exception in many regions here in the, for example, string my string, new string, I, there is a possibility if `i` is 0 in all points are assignment and then next if my string 0 equals, ok. We can write it instead of my string 0 you know you just write `args 0` if `args 0`; no if `args 0` ok, that is fine it will work ok, if `args 0`.

Now, let us compile this program and as we see it will work for some input, but not all output, not all inputs, ok. And again if we run this program first say Java input is capital J a v a. So, it basically runs this program correctly and I love Java. Now, let us see you can see this program basically starts at the point here in the last statement. So, an error is there. It will not work for example, now what will happen if we run this program without

passing any input common line. So, just simply run. It is also giving an error because an array index out of bound error is there. So, we see this program is erroneous.

Now, let us implement this with multiple catch block.

(Refer Slide Time: 10:08)

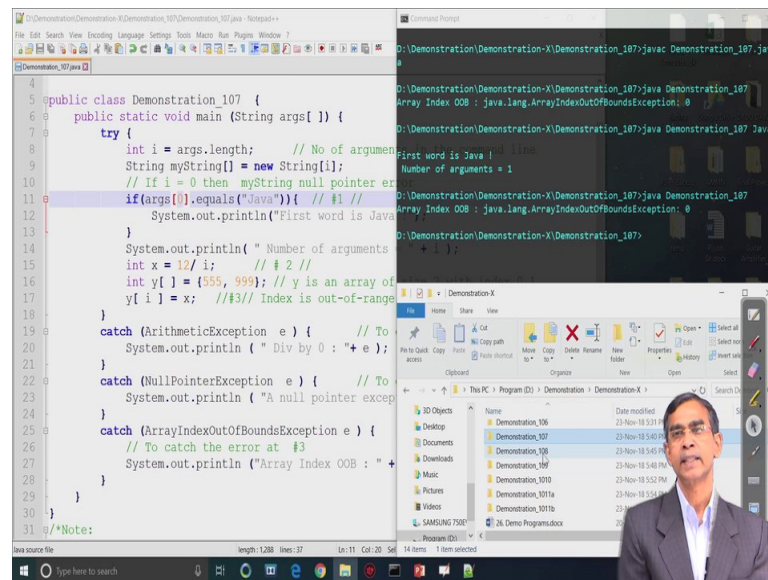


```
4
5 public class Demonstration_107 {
6     public static void main (String args[] ) {
7         try {
8             int i = args.length; // No of arguments in the command line
9             String myString[] = new String[i];
10            // If i = 0 then myString null pointer error
11            if(myString[0].equals("Java")){ // #1 //
12                System.out.println("First word is Java !");
13            }
14            System.out.println( " Number of arguments = " + i );
15            int x = 12/ i; // #2 //
16            int y[] = {555, 999}; // y is an array of size 2 with index 0,1
17            y[ i ] = x; // #3// Index is out-of-range may occur if i > 1
18        }
19        catch (ArithmeticException e) { // To catch the error at #2
20            System.out.println ( " Div by 0 : " + e );
21        }
22        catch (NullPointerException e) { // To catch the error at #1
23            System.out.println ( "A null pointer exception : " + e );
24        }
25        catch (ArrayIndexOutOfBoundsException e) {
26            // To catch the error at #3
27            System.out.println ( "Array Index OOB : " + e );
28        }
29    }
30 }
31 //Note:
```

So, this is an example to illustrate the try with multiple catch. So, all the errors all the exception that occurs there in the program can be handled using try or that means, try will check if exception if any occurs during run time and then corresponding catch will catch all the exception and then handle them with their proper code here. Here very simple code that we have used system print l n only.

Now, here there is point one where the error occurs, point two also the error occurs and point three also the error occurs here. And we just in the process of catching this here arithmetic exception and then if it is there so first catch is to catch the arithmetic exception, second catch to null pointer exception, and then the third catch is to array index out of bound exception. So, this way if we run this program again it will run successfully, again my string 0 you can check it arg 0 that is all. If the statement, is ok, fine.

(Refer Slide Time: 11:12)



```
4 public class Demonstration_107 {
5     public static void main (String args[] ) {
6         try {
7             int i = args.length; // No of arguments
8             String myString[] = new String[i];
9             // If i = 0 then myString null pointer exception
10            if (args[0].equals("Java")) { // #1 //
11                System.out.println("First word is Java");
12            }
13            System.out.println(" Number of arguments");
14            int x = 12/ i; // #2 //
15            int y[] = {555, 999}; // y is an array of
16            y[i] = x; // #3// Index is out-of-range
17        }
18        catch (ArithmeticException e) { // To
19            System.out.println ( " Div by 0 : "+ e );
20        }
21        catch (NullPointerException e) { // To
22            System.out.println ( "A null pointer exception" );
23        }
24        catch (ArrayIndexOutOfBoundsException e) {
25            // To catch the error at #3
26            System.out.println ("Array Index OOB : " +
27                e.getMessage());
28        }
29    }
30 }
31 //Note:
```

Terminal Output:

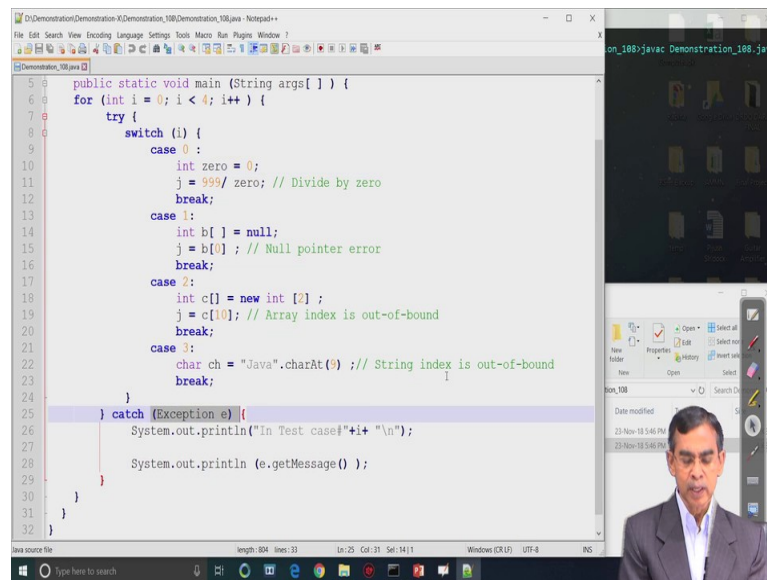
```
D:\Demonstration\Demonstration-X\Demonstration_107>java Demonstration_107.java
Array Index OOB : java.lang.ArrayIndexOutOfBoundsException: 0
D:\Demonstration\Demonstration-X\Demonstration_107>java Demonstration_107.java
First word is Java
Number of arguments = 1
D:\Demonstration\Demonstration-X\Demonstration_107>java Demonstration_107.java
Div by 0 : java.lang.ArithmeticException: / by zero
D:\Demonstration\Demonstration-X\Demonstration_107>
```

So, this program is now robust and whatever the situation happens here in the last case it was working java all the input it is where, but I love java it was not working there. A null pointer exception `java.lang.nullpointerexception` where it is? Ok, you just do my string whatever it is here, ok. Run this some of here is a program, yeah. So, no compile-time error, yeah. So, there is an error because no input, if we do not give input one error will be there, if we just simply java a null pointer java.lang (Refer Time: 12:12), yeah.

So, it was working correctly; there are some marks there we have resolving. Anyway, so now, again if we run this program java without any input, I mean this program without any input common line output these also it will catch the exception and accordingly program will not abruptly terminated. Array index out of bound this is why it gives the error, fine.

So, let us have the next one. This is also another example that multiple errors by a single catch. So, this example we can see the multiple errors here in the switch statement actually, in every switch there is an error is there. And then if we run this program as we have make the program robust by using single try, but the multiple errors are only one try and then catch is to handle all the catch here you can see catch exception `e` because of all the irrespective of the type of error it will catch it. This is the exception is there.

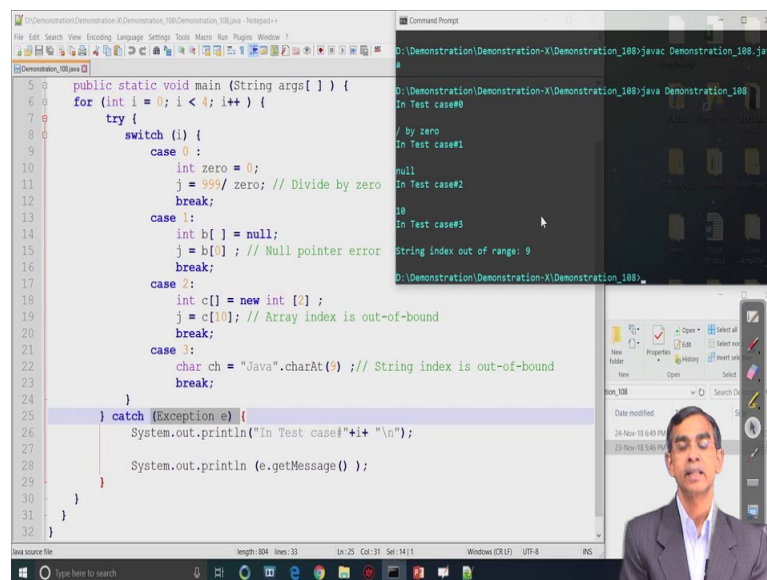
(Refer Slide Time: 13:26)



```
5 public static void main (String args[] ) {
6     for (int i = 0; i < 4; i++) {
7         try {
8             switch (i) {
9                 case 0 :
10                     int zero = 0;
11                     j = 999/ zero; // Divide by zero
12                     break;
13                 case 1:
14                     int b[] = null;
15                     j = b[0]; // Null pointer error
16                     break;
17                 case 2:
18                     int c[] = new int [2] ;
19                     j = c[10]; // Array index is out-of-bound
20                     break;
21                 case 3:
22                     char ch = "Java".charAt(9) ;// String index is out-of-bound
23                     break;
24             }
25         } catch (Exception e) {
26             System.out.println("In Test case#"+i+ " \n");
27             System.out.println( e.getMessage() );
28         }
29     }
30 }
31 }
32 }
```

And this program will; ok. If we run it, it in fact for each switch statement case statement it will face an exception and this accession will be caught by the catch statement.

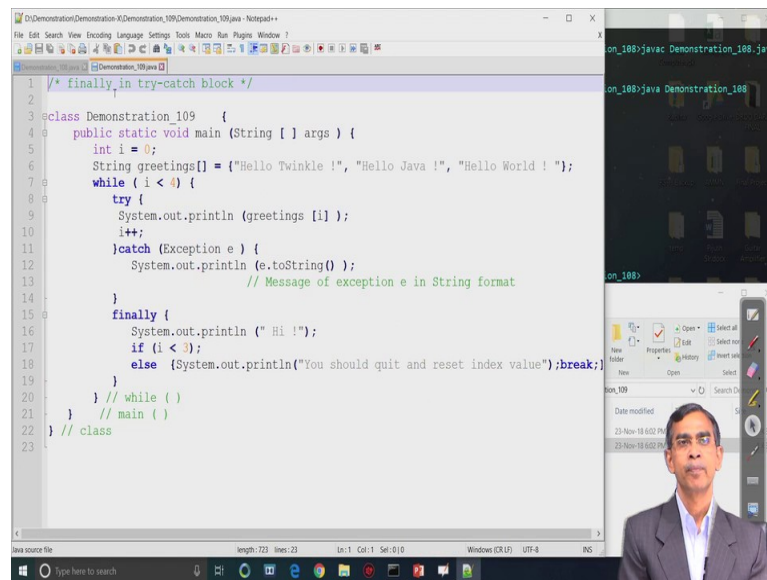
(Refer Slide Time: 13:33)



```
0:\Demonstration\Demonstration-X\Demonstration_108>javac Demonstration_108.java
0:\Demonstration\Demonstration-X\Demonstration_108>java Demonstration_108
In Test case#0
// by zero
In Test case#1
null
In Test case#2
10
In Test case#3
String index out of range: 9
0:\Demonstration\Demonstration-X\Demonstration_108>
```

Yeah as we see so all exceptions that occur in the four case statements are caught in the single catch statement.

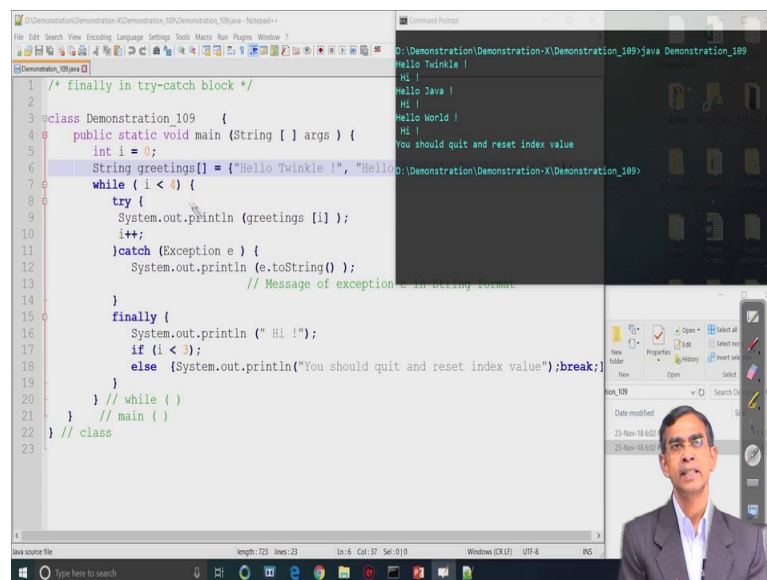
(Refer Slide Time: 13:58)



```
1  /* finally in try-catch block */
2
3  class Demonstration_109 {
4      public static void main (String [] args) {
5          int i = 0;
6          String greetings[] = {"Hello Twinkle !", "Hello Java !", "Hello World ! "};
7          while ( i < 4 ) {
8              try {
9                  System.out.println (greetings [i] );
10                 i++;
11             } catch (Exception e) {
12                 System.out.println (e.toString() );
13                 // Message of exception e in String format
14             }
15             finally {
16                 System.out.println (" Hi !");
17                 if ( i < 3 );
18                 else {System.out.println("You should quit and reset index value");break;}
19             }
20         } // while ( )
21     } // main ( )
22 } // class
```

Now, so there is another block that can be added with the try-catch block called the finally,. So, here is the next program that we can see, ok.

(Refer Slide Time: 14:03)



```
1  /* finally in try-catch block */
2
3  class Demonstration_109 {
4      public static void main (String [] args) {
5          int i = 0;
6          String greetings[] = {"Hello Twinkle !", "Hello
7          while ( i < 4 ) {
8              try {
9                  System.out.println (greetings [i] );
10                 i++;
11             } catch (Exception e) {
12                 System.out.println (e.toString() );
13                 // Message of exception e in String format
14             }
15             finally {
16                 System.out.println (" Hi !");
17                 if ( i < 3 );
18                 else {System.out.println("You should quit and reset index value");break;}
19             }
20         } // while ( )
21     } // main ( )
22 } // class
```

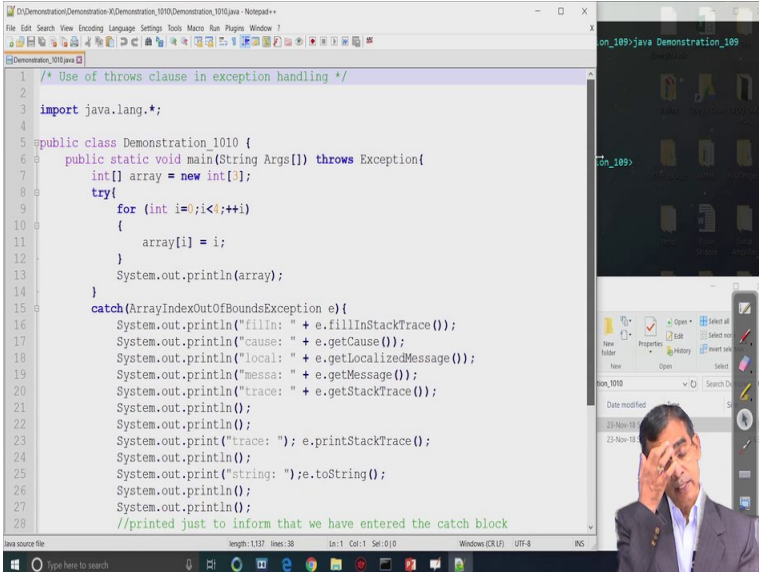
```
D:\Demonstration\Demonstration-X\Demonstration_109>java Demonstration_109
Hello Twinkle !
Hi !
Hello Java !
Hi !
Hello World !
Hi !
You should quit and reset index value
D:\Demonstration\Demonstration-X\Demonstration_109>
```

We can see the block with finally, so all the exception that can be caught by try here the try statement and then corresponding catch, and then we add it finally, statement here. And as we see whatever the exception whether occur or not this finally statement will be executed and it will accordingly the code will be, code will run and then gives the output.

Now, let us run this program as we see the greetings has the three, so this while loop is successful for three, first three loops, I mean three iterations 0 1 3, but three for the three because less than 4 means this loop will roll till 3 and for the last one it will report an error and then we will see the finally, will print this statement accordingly. So, here we can see it will print high statements all the time, but the next if i less than 3 it will do nothing but it will be if i greater than 3, ok. So, as we see this is the output here and the last statement finally, whenever I loop is there fine. So, here we can see that finally block if it is added with try-catch. So, for every try-catch occurrence this finally will be executed, this is the main use of the final statement finally statement.

Now, our next illustration to highlights the throw clause in a program and here is basically the demonstration of the throws.

(Refer Slide Time: 15:35)



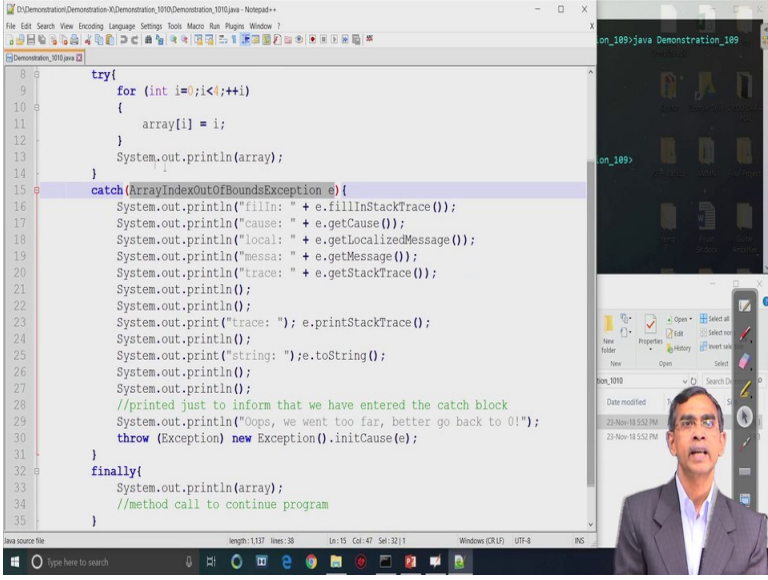
```

1  /* Use of throws clause in exception handling */
2
3  import java.lang.*;
4
5  public class Demonstration_1010 {
6      public static void main(String Args[]) throws Exception{
7          int[] array = new int[3];
8          try{
9              for (int i=0;i<4;++i)
10             {
11                 array[i] = i;
12             }
13             System.out.println(array);
14         }
15         catch(ArrayIndexOutOfBoundsException e){
16             System.out.println("fillin: " + e.fillInStackTrace());
17             System.out.println("cause: " + e.getCause());
18             System.out.println("local: " + e.getLocalizedMessage());
19             System.out.println("messa: " + e.getMessage());
20             System.out.println("trace: " + e.getStackTrace());
21             System.out.println();
22             System.out.println();
23             System.out.print("trace: "); e.printStackTrace();
24             System.out.println();
25             System.out.print("string: "); e.toString();
26             System.out.println();
27             System.out.println();
28             //printed just to inform that we have entered the catch block

```

As we see as we have discussed in our theoretical discussion a throws clause can throw an exception explicitly and that exception then can be caught in the try-catch block here. Now here is a small program segment as we see the name of the class is demonstration and the score 1010, and here array is a declaration of size 3 and then try for integer i equal to 0, i less than 4, so 0 1 2 3. As it is an array size 3 definitely for the first array it is an array index out of bound exception it will be there. So, for i equal to 0 no exception, i equal to 1 2 no exception, for i equal to 3 the exception will be there.

(Refer Slide Time: 16:27)

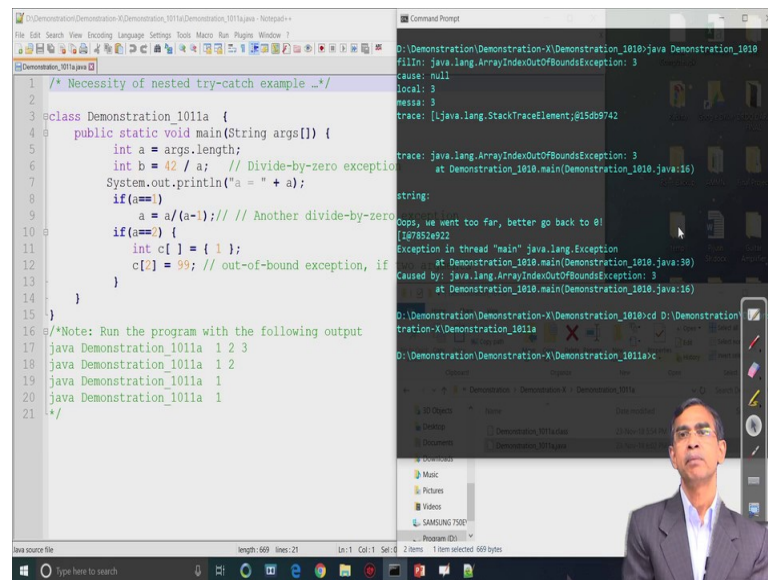


```
8      try{
9          for (int i=0;i<10;i++)
10             {
11                 array[i] = i;
12             }
13             System.out.println(array);
14         }
15         catch (ArrayIndexOutOfBoundsException e) {
16             System.out.println("fillIn: " + e.fillInStackTrace());
17             System.out.println("cause: " + e.getCause());
18             System.out.println("local: " + e.getLocalizedMessage());
19             System.out.println("messa: " + e.getMessage());
20             System.out.println("trace: " + e.getStackTrace());
21             System.out.println();
22             System.out.println();
23             System.out.print("trace: "); e.printStackTrace();
24             System.out.println();
25             System.out.print("string: "); e.toString();
26             System.out.println();
27             System.out.println();
28             //printed just to inform that we have entered the catch block
29             System.out.println("Oops, we went too far, better go back to 0!");
30             throw (Exception) new Exception().initCause(e);
31         }
32         finally{
33             System.out.println(array);
34             //method call to continue program
35         }
```

Now, this exception will be caught in the try block and then catch will do it and here actually it will see the different methods those are there in the exception class is invoked here. For example, `getCause`, then `fillInStackTrace`, then `getLocalizedMessage`, `getMessage` whatever it is there. So, it is basically the implementation of exception class in `java.lang.throwable` to package and according to this, we can access this method in our program.

So, this catch whenever it occurs it catch the array index out of bound exception, for that error it will basically give the all message trace and then regarding the complete information about the exception that it occurs there via the different methods in that throwable class. Now, let us run this program and also it has finally method.

(Refer Slide Time: 17:19)



```
1  /* Necessity of nested try-catch example ...*/
2
3  class Demonstration_1011a {
4      public static void main(String args[]) {
5          int a = args.length;
6          int b = 42 / a; // Divide-by-zero exception
7          System.out.println("a = " + a);
8          if(a==1)
9              a = a/(a-1); // Another divide-by-zero
10         if(a==2) {
11             int c[] = { 1 };
12             c[2] = 99; // out-of-bound exception, if
13         }
14     }
15 }
16
17 //Note: Run the program with the following output
18 java Demonstration_1011a 1 2 3
19 java Demonstration_1011a 1 2
20 java Demonstration_1011a 1
21 +/
```

```
D:\Demonstration\Demonstration-X\Demonstration_1010>java Demonstration_1010
FillIn: java.lang.ArrayIndexOutOfBoundsException: 3
cause: null
local: 3
message: 3
trace: [Ljava.lang.StackTraceElement;@150b9742

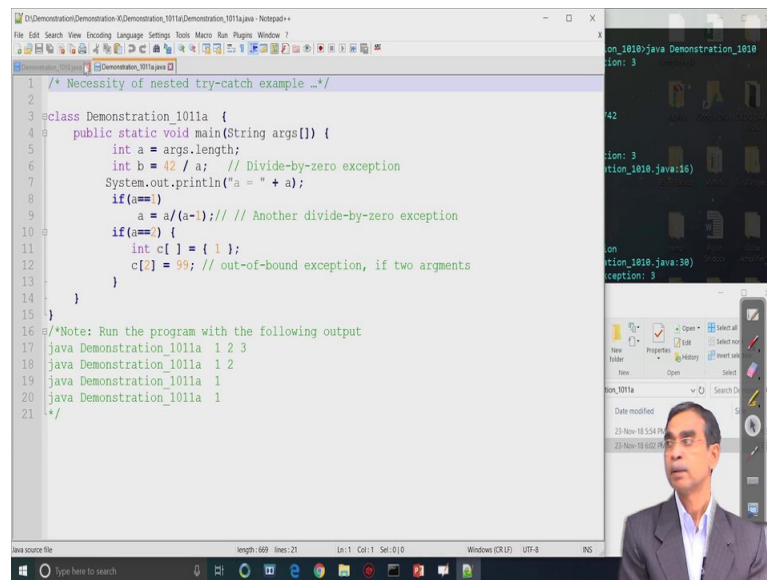
Trace: java.lang.ArrayIndexOutOfBoundsException: 3
      at Demonstration_1010.main(Demonstration_1010.java:16)

string:
Oops, we went too far, better go back to 0!
[167852e922
Exception in thread "main" java.lang.Exception
      at Demonstration_1010.main(Demonstration_1010.java:30)
Caused by: java.lang.ArrayIndexOutOfBoundsException: 3
      at Demonstration_1010.main(Demonstration_1010.java:16)
```

And here you can see within this catch block we include the throw statement it basically throw the exception to the caller. So, this method if it is called here the method is basically the name of the method is named the here method is ok. So, here is no caller of course, so it will not return anything to any other, but if this method is the try-catch block is inside a method, which basically throw then the throw an exception which basically returns to the caller here, fine.

So, this is the complete methods are there, if we run it then we can see how this output it will give for us. So, it is basically showing the exception as we see for others it will give it. Where is a program compilation? It is here, yeah. So, java array index out of bound exception for 3 and it gives all the detail information about the exception it is there. So, if you want to scrutinize this program for further details and then the cause and effects and everything then all those information will be useful.

(Refer Slide Time: 18:48)

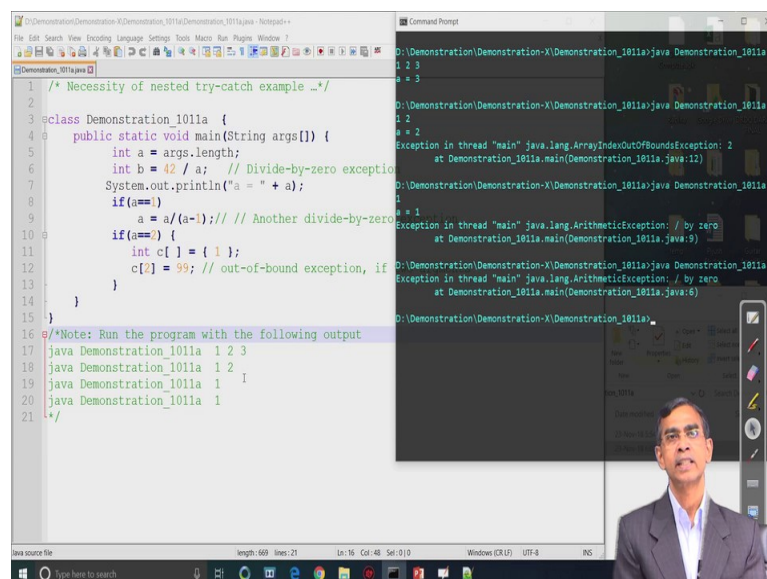


```
1  /* Necessity of nested try-catch example ...*/
2
3  class Demonstration_1011a {
4      public static void main(String args[]) {
5          int a = args.length;
6          int b = 42 / a; // Divide-by-zero exception
7          System.out.println("a = " + a);
8          if(a==1)
9              a = a/(a-1); // Another divide-by-zero exception
10         if(a==2) {
11             int c[] = { 1 };
12             c[2] = 99; // out-of-bound exception, if two arguments
13         }
14     }
15 }
16
17 /*Note: Run the program with the following output
18 java Demonstration_1011a 1 2 3
19 java Demonstration_1011a 1 2
20 java Demonstration_1011a 1
21 */
```

The screenshot shows a Notepad++ window with the above Java code. To the right, a Windows File Explorer window is open, showing the file 'Demonstration_1011a.java' in the 'D:\Demonstration\Demonstration-X\Demonstration_1011a' directory. The file's date modified is 23-Nov-19 5:54 PM.

Now, our next example is basically a nested try-catch example. As we have mentioned that a try-catch block can be put inside another try-catch block, so it is called the nested here now, this is the one program. As you see this program does not have any error but when we want to run it, it may leads to some exception. Here this program if it is run for the three different intention, for example, let us run this program and give the three intentions as we see and when the exception occurs you will see exactly.

(Refer Slide Time: 19:15)



```
1  /* Necessity of nested try-catch example ...*/
2
3  class Demonstration_1011a {
4      public static void main(String args[]) {
5          int a = args.length;
6          int b = 42 / a; // Divide-by-zero exception
7          System.out.println("a = " + a);
8          if(a==1)
9              a = a/(a-1); // Another divide-by-zero exception
10         if(a==2) {
11             int c[] = { 1 };
12             c[2] = 99; // out-of-bound exception, if
13         }
14     }
15 }
16
17 /*Note: Run the program with the following output
18 java Demonstration_1011a 1 2 3
19 java Demonstration_1011a 1 2
20 java Demonstration_1011a 1
21 java Demonstration_1011a 1
22 */
```

The screenshot shows the same Notepad++ window with the Java code. A Command Prompt window is open, showing the execution of the program with three different inputs. The output is as follows:

```
D:\Demonstration\Demonstration-X\Demonstration_1011a>java Demonstration_1011a
1 2 3
a = 3

D:\Demonstration\Demonstration-X\Demonstration_1011a>java Demonstration_1011a
1 2
a = 2
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 2
    at Demonstration_1011a.main(Demonstration_1011a.java:12)

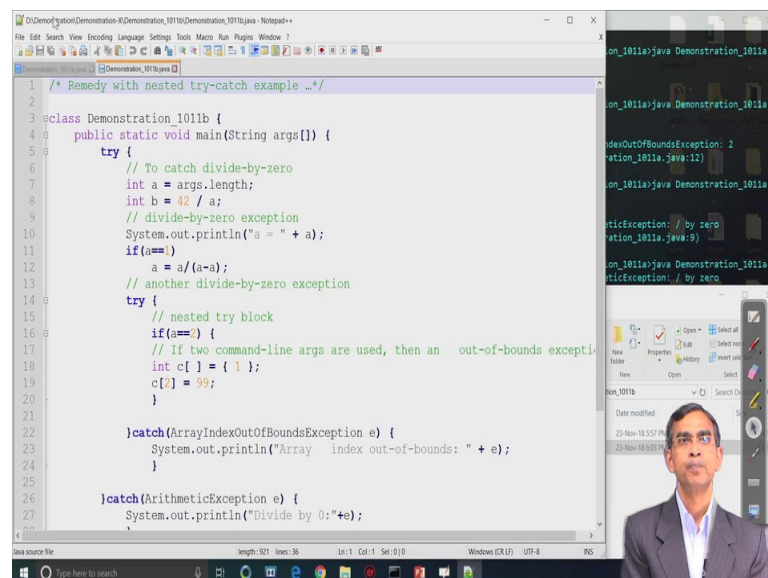
D:\Demonstration\Demonstration-X\Demonstration_1011a>java Demonstration_1011a
1
a = 1
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Demonstration_1011a.main(Demonstration_1011a.java:9)

D:\Demonstration\Demonstration-X\Demonstration_1011a>java Demonstration_1011a
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Demonstration_1011a.main(Demonstration_1011a.java:6)
```


So, without try-catch this is the implementation and run this program first with 1 2 3 as an input yeah, so this program is successful at this stage. Now, 1 2 only two inputs it gives an exception as the array index out of bound exception. Now, again with only single input it also gives an exception it is also an arithmetic exception. Now, no input it also gives an error, it is also given the error that, ok. So, we can see, so this program is vulnerable to three inputs, as we are three cases, as three test data as we have checked it, but it works only for them. So, an exception is there.

Now, let us see our enemy how this can be robust, make a robust program using try-catch, nested try-catch.

(Refer Slide Time: 20:13)



The screenshot shows a Java IDE with a file named 'Demonstration_1011b.java'. The code implements a nested try-catch structure. The outer try block catches `ArithmeticException` (division by zero). Inside it, another try block catches `ArrayIndexOutOfBoundsException` (array index out of bounds). The code includes comments explaining the purpose of each block. The output window on the right shows the execution results for different inputs: '1 2 3' (successful), '1 2' (ArrayIndexOutOfBoundsException), and '1' (ArithmeticException: / by zero).

```
1 /* Remedy with nested try-catch example ... */
2
3 class Demonstration_1011b {
4     public static void main(String args[]) {
5         try {
6             // To catch divide-by-zero
7             int a = args.length;
8             int b = 42 / a;
9             // divide-by-zero exception
10            System.out.println("a = " + a);
11            if(a==1)
12                a = a/(a-a);
13            // another divide-by-zero exception
14            try {
15                // nested try block
16                if(a==1) {
17                    // If two command-line args are used, then an out-of-bounds exception
18                    int c[] = { 1 };
19                    c[2] = 99;
20                }
21            } catch (ArrayIndexOutOfBoundsException e) {
22                System.out.println("Array index out-of-bounds: " + e);
23            }
24        } catch (ArithmeticException e) {
25            System.out.println("Divide by 0: "+e);
26        }
27    }
28 }
```

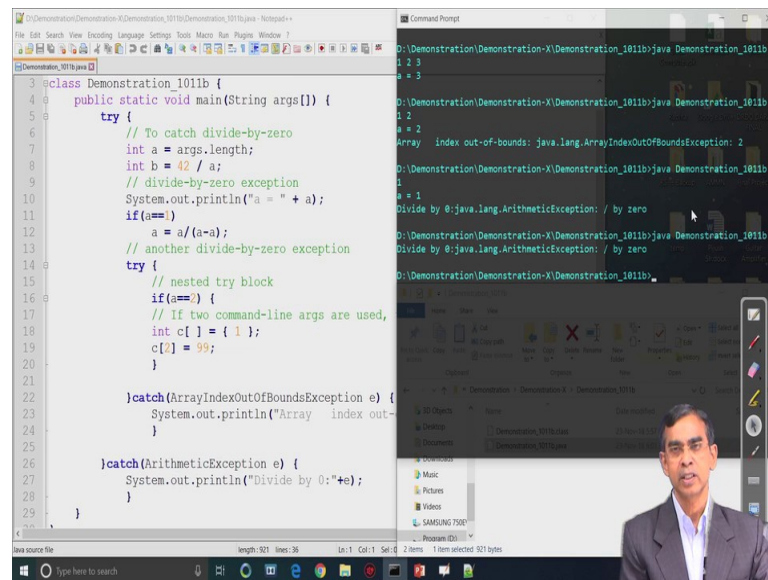
Output:

```
on_1011a>java Demonstration_1011b 1 2 3
a = 3
on_1011a>java Demonstration_1011b 1 2
ArrayIndexOutOfBoundsException: 2
on_1011a>java Demonstration_1011b 1
ArithmeticException: / by zero
on_1011a>java Demonstration_1011b
ArithmeticException: / by zero
```

Now, here if we see the first blocks 42 by a is basically arithmetic exception and if a equals to 1 it is an arithmetic exception. So, it is a very similar kind of exception. So, as it is only one exception, so this can be tried with an only try block. So, this is the outer try block.

Now, inside again this is another exception if a equal to 2 it is basically the exception is called array out of bound exception. So, to take care of this exception we put another try-catch, so inside this one. So, this is an example of a try-catch. As you see the in-out try-catch caught, I mean try for the array index out of bound exception and catch accordingly whereas, the outermost try-catch block caught the arithmetic exception here and cause it. So, this is the idea about the instant try-catch block.

(Refer Slide Time: 21:07)



The screenshot shows a Java IDE on the left and a Command Prompt on the right. The IDE displays the source code for a class named `Demonstration_1011b`. The `main` method takes an array of strings `args` and processes them. It includes several exception handling blocks: a `try` block for `ArrayIndexOutOfBoundsException` (line 14), a nested `try` block for `ArithmeticException` (line 16), and a `catch` block for `ArithmeticException` (line 26). The Command Prompt shows the output of running the program with arguments `1 2 3`. The output is: `1 2 3`, `a = 3`, `a = 2`, `Array index out-of-bounds: java.lang.ArrayIndexOutOfBoundsException: 2`, `a = 1`, `Divide by 0: java.lang.ArithmeticException: / by zero`, `Divide by 0: java.lang.ArithmeticException: / by zero`, and `Divide by 0: java.lang.ArithmeticException: / by zero`. A small video inset in the bottom right corner shows a man speaking.

```
class Demonstration_1011b {  
    public static void main(String args[]) {  
        try {  
            // To catch divide-by-zero  
            int a = args.length;  
            int b = 42 / a;  
            // divide-by-zero exception  
            System.out.println("a = " + a);  
            if(a==1)  
                a = a/(a-a);  
            // another divide-by-zero exception  
            try {  
                // nested try block  
                if(a==1) {  
                    // If two command-line args are used,  
                    int c[] = { 1 };  
                    c[0] = 99;  
                }  
            } catch (ArrayIndexOutOfBoundsException e) {  
                System.out.println("Array index out-  
            }  
        } catch (ArithmeticException e) {  
            System.out.println("Divide by 0:" + e);  
        }  
    }  
}
```

And now if we run this program and again try with all possible output for which it was not working in the previous case let us see what will happen, 1 2 3 it is work. Now java 1 2, it is 1 2 it also works, right. It is showing the report; it basically catches the exception that has been caught. Again 1 it is also seen divide by 0 it is basically the exception has been caught, and then these also by 0 it is caught.

So, in this case, the program is robust in the sense that in the cause of exception the program will completes its execution and finally without any abnormal termination or any causes or loss of data whatever it is there. So, this is a concept of exception handling. In fact, exception handling is a very important feature in java programming language scenario, and more practice is required of course.

(Refer Slide Time: 21:54)

In today's demonstration ...

1. About compile-time error
2. About run-time error
3. Simple Try-Catch block
4. Try with multiple Catch
5. Multiple errors with single catch
6. Finally in try-catch block
7. Exception handling using Throws statement
8. Nested try-catch block

DEBASIS SAMANTA
CSE
IIT KHARAGPUR

Again I advise you to practice all the example that we have used in this demonstration so that you can practice your own. And then this basically hands-on practice is required to understand all these things thoroughly, and in case of any doubt and confusion, you are most welcome to, feel free to ask us. So, thank you for your attention.

Thank you very much and all the best for learning.