

# FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING


## Lecture27


### Lecture 27: Exception Handling - Solved Problems

Welcome to lecture 27 error handling exception. So, in the last lecture, we had seen two examples with respect to C++, that too we had seen the try, throw and catch block, right? And then we have solved and in fact, we have seen how to handle the runtime error, right? So, in this lecture, we will talk about the case study, the first case study, right? So, assume that you are developing a basic calculator in C++.

#### Case Study 1

Imagine you are developing a basic calculator in C++ that performs arithmetic operations like addition, subtraction, multiplication, and division. One common error that can occur is a division by zero, which causes undefined behavior. Additionally, users may input invalid data that the program needs to handle properly.



2

So, that will be performing arithmetic operations like addition, subtraction, multiplication and division right. So, when you do the division you have to be very careful because there will be a divide by 0 error all right. So, this is undefined behavior right. So, this particular problem we had seen in the example. right.

## Case Study 1

Imagine you are developing a basic calculator in C++ that performs arithmetic operations like addition, subtraction, multiplication, and division. One common error that can occur is a division by zero, which causes undefined behavior. Additionally, users may input invalid data that the program needs to handle properly.

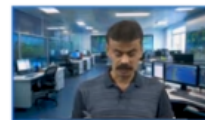


So, since  $a \div b$  when  $b$  is equal to 0, we call this as a undefined behavior. So, how to tackle this undefined behavior, we are going to see. In fact, if you give, let us say  $z$  is equal to  $a \div b$ , if you give  $b$  is equal to 0, so there will be a runtime error. So, how do you handle this? And also, user may input the invalid data.

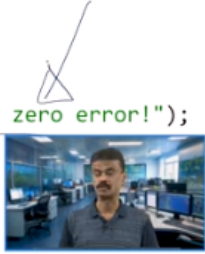
So, your program has to handle everything, right? So, this we will study with the help of C++. So, let us include IO stream and SGD except, all right. So, here you have the function divide.

So, the function divide has A and B as the arguments. If B is equal to 0, you are throwing a runtime error, all right. So, you are throwing a runtime error. So, this is available in standard exception, all right, when you are including the header file called standard exceptions. So, you have this runtime error inbuilt function, and this will throw a division by 0 error, all right. So, this will throw the division by 0 error.

```
1  #include <iostream>
2  #include <stdexcept> // For standard exceptions
3  using namespace std;
4
5  // Function to perform division
6  double divide(int a, int b) {
7      if (b == 0) {
8          // Throw runtime error if b is 0
9          throw runtime_error("Division by zero error!");
10     }
11     return (double) a / b;
12 }
13
```



```
1  #include <iostream>
2  #include <stdexcept> // For standard exceptions
3  using namespace std;
4
5  // Function to perform division
6  double divide(int a, int b) {
7      if (b == 0) {
8          // Throw runtime error if b is 0
9          throw runtime_error("Division by zero error!");
10     }
11     return (double) a / b;
12 }
13
```




If B is not 0, then it has to return A by B. So, the idea is to have the arithmetic calculator. So, now you have two variables, X and Y, whose data type is integer, and you have an operation character. So, now you have a try block. So, as I said in the last class, the try block is nothing but—I mean, you are writing as such, right? You are writing as such, the usual program, all right?

So then you may have, right? You may have to throw the exception. So, now you take the input C in X, right? That means you are taking one input X and another input Y. The C in dot ignore, right? So, this will ignore leftover new line character, right?

So, it will ignore the leftover new line character. So, now it is prompting to enter either the arithmetic operators, right? Either one of the arithmetic operators, maybe plus, minus, star or slash, right? So let us have C in operation. So operation is a character.

So it is expecting one character. So that character should be either plus or minus or star or slash. So now we have one variable result whose data type is double. Now you have switch operation. So this is a character.

```
32     double result;
33
34     switch (operation) {
35         case '+':
36             result = x + y;
37             break;
38         case '-':
39             result = x - y;
40             break;
41         case '*':
42             result = x * y;
43             break;
44         case '/':
45             result = divide(x, y);
46             // Attempt to divide, might throw exception
47             break;
```



If the case is plus, you are doing x plus y. If the operation is minus, you have x minus y. If the operation is star, you are multiplying. But your operation is slash, you are calling the function. Divide x comma y. Divide x comma y. So what will happen? Suppose it is a divide operation, x comma y and assume that y is equal to 0. So, let us pass X sum value and Y is 0 right.

If it is 0 correct if B is equal to 0 it will throw it will call this function runtime error and it will throw this particular message right. So, try so when it is calling this function you have the throw block right. So, throw right the throw statement. So, it is calling this particular function runtime underscore error which is available in standard exceptions. So division by zero error will be thrown and let us see where it will be caught, right?

So you have a catch function here, right? So before that you have one default, right? You have one default case. So the default case is if the user is entering any other operator. So this will tell invalid operation because you are expecting plus minus star or slash.

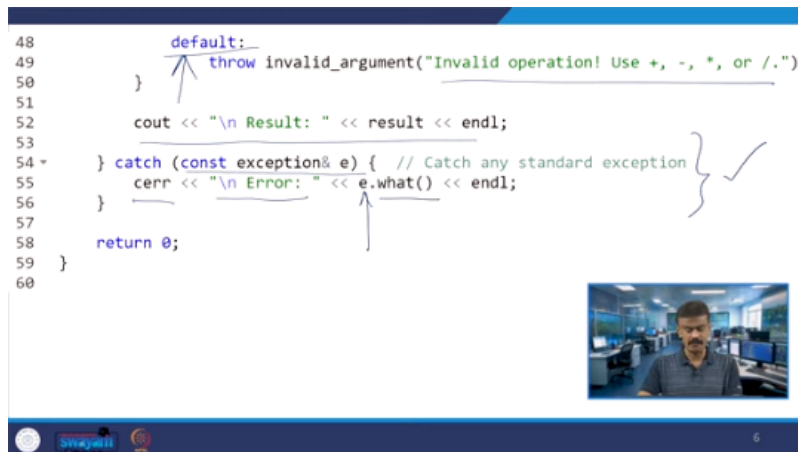
So, otherwise when it is coming out it will display the result all right. So, suppose you have X plus Y all right X minus Y let us say all are Y is a non-zero all right. If Y is a non-zero so it will print all right Y is non-zero it will print the result correct. Otherwise if suppose Y is non-zero right suppose Y is 0 all right. So, in that case if B is equal to 0 it will throw this particular exception.

All right. And this will be caught by the function catch. Right. So you have catch exception E. Right. So you have the object E. So error will be printed.

See error. Error will be printed. And the object E is invoking what? This we had already seen. Right.

When is this invoking what? Assume that my input is, let us say, 10, 0, and slash. That means my X is 10. Y is 0, and the case I am giving is division, right? It is a division operator, right?

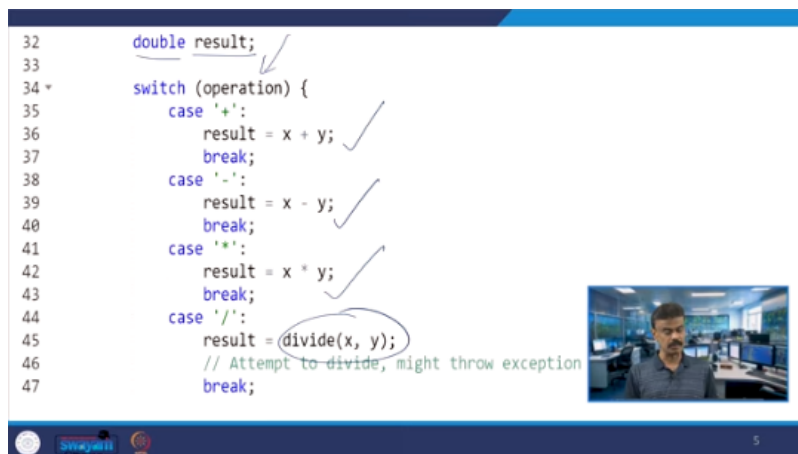
```
48         default:
49             throw invalid_argument("Invalid operation! Use +, -, *, or /.");
50     }
51
52     cout << "\n Result: " << result << endl;
53
54 * } catch (const exception& e) { // Catch any standard exception
55     cerr << "\n Error: " << e.what() << endl;
56 }
57
58 return 0;
59 }
60
```



So, this is the one. Assume that I am giving the division operator. The case is slash. So, this will call divide X, Y. That means I am passing X as 10 and Y as 0, right? X is 10 and Y is 0.

```
32 double result;
33
34 * switch (operation) {
35     case '+':
36         result = x + y;
37         break;
38     case '-':
39         result = x - y;
40         break;
41     case '*':
42         result = x * y;
43         break;
44     case '/':
45         result = divide(x, y);
46         // Attempt to divide, might throw exception
47         break;

```



So, in that case, your A is 10 and B is 0. A will be equal to 10, and B will be equal to 0. So, when B is equal to 0, this is throwing an exception. That means it is calling the runtime error function, which is available in standard exceptions. And then the message is 'division by 0 error.'

And this will be caught over here. You have the exception, right? Reference object E. So, error will be printed, right? Error will be printed. And this E is calling this what function.

right. He is calling the what function. In fact, what which is available in exception, this we had seen already, correct. So, then you will get the output like this error division by 0 error. So, this what function is available in standard exceptions, right.


So, in fact, one of the examples we had seen this, right. This will give you as expected error and this is the message division by 0 error. You are passing 10 0 slash You are getting the output like this, alright. So, suppose my input is this, 10, 2, alright.

X is 10, Y is 2 and I am going for division. So, 10 by 2, right, this will be 2, right, passing 10 and 2. When you are passing 10 and 2, B is not 0. So, A by B will be performed, correct. So, 10 by 2, you have to get the answer 5.

So, 5.0 will be returned. Return double of A by B. So, 5.0 will be return and that will be your result here, right. So, it will go here. It will not go to the catch function. So, C out.

```
48         default:
49             throw invalid_argument("Invalid operation! Use +, -, *, or /.");
50     }
51
52     cout << "\n Result: " << result << endl;
53
54     } catch (const exception& e) { // Catch any standard exception } ✓
55         cerr << "\n Error: " << e.what() << endl;
56     }
57
58     return 0;
59 }
60
```

Diagram: A circle with three arrows pointing to 'x', 'y', and 'division'. A checkmark is next to the 'division' label.



So, since there is no catch, it will go to return 0. So, the result. So, 10 by 2, 5 will be printed. So, this is what exactly happened. right.

So, 10 by 2 and I am passing slash and you can see the cursor there, right that is a cursor. So, 10 and 2 I am giving the input one is for x another one is for y. So, slash is the operator. So, it is giving the output 10 by 2 is 5. So, this is one case

study, right. So, another case study is exception handling in bank account management.


So, one is the Easier one we had seen and in fact, we had keep on seeing this divided by 0 error. Maybe we will see if there is anything else, right? So, let us assume a simple banking system where users can perform operations like depositing the money or withdrawing the money or both, all right? So, we will introduce exceptions to handle the following error.

when the withdrawal amount exceeds the account balance all right withdrawal amount exceeds the suppose my balance is 1000 but i want to take 1500 right this is one case and depositing a negative amount right which is also not allowed so other than divide by 0 so this is observed case right in the banking system so when you are doing withdrawing amount exceeds account balance or depositing a negative amount right so which is not allowed so we will see how we are going to handle with respect to error handling exception or exception handling in C++. So, now let us consider include IO stream and include standard exceptions. Class bank account, right. So, here you have private member data.

So, the class is bank account and you have private member data account holder and balance. Account holder is string. And balance is double. So, let us have the constructor, two argument constructor, string name and initial balance. If initial balance less than 0, right?

The negative balance, right? It is throwing, right? The throwing invalid underscore argument, right? Which is the function available in standard exceptions. And this will throw this message.

```
12 public:
13     // Constructor
14     BankAccount(string name, double initialBalance) {
15         if (initialBalance < 0) {
16             throw invalid_argument("Initial balance cannot be negative!");
17         }
18         accountHolder = name;
19         balance = initialBalance;
20     }
21
22     // Function to deposit money
23     void deposit(double amount) {
24         if (amount <= 0) {
25             throw invalid_argument("Deposit amount must be positive!");
26         }
27         balance += amount;
28         cout << "Deposit successful! New balance: Rs." << balance << endl;
29     }
30 }
```



Initial balance cannot be negative. So, this exception is being thrown. So, this exception is being thrown and the name that you are passing it is copied into account holder and initial balance is copied into balance. Next let us see. So, this is the constructor.

So, this is the constructor and next one is the member function. Member function is void deposit. So, you are passing one argument amount. If amount is less than or equal to 0, right? So, you cannot deposit the money 0, right?

It has to throw the invalid argument, correct? Or the amount cannot be negative. So, I put because minus 15 is an integer, right? I want to deposit rupees minus 15 which is observed, right? So, in that case it has to throw the invalid argument.


That means it has to throw the exception. The exception here is deposit amount must be positive, right? So, the second one, right? Otherwise, if it is greater than 0, balance equal to balance plus amount and the deposit is successful, right? The new balance rupees is balance that you are entering.

Whatever you have deposited, the balance will be displayed, rupees, whatever be the balance. So, now another function is void withdraw, all right? Void withdraw, you have the argument amount right so the parameter amount if the amount is greater than balance all right that means the withdrawal amount if it is greater than balance then it will throw the runtime error this is what we had seen in fact all right so few examples we had seen in the last class all right so throw runtime error so it will throw the runtime error with the message insufficient funds cannot withdraw that amount right, insufficient funds cannot withdraw that amount.

If amount is less than or equal to 0, right, if amount is less than or equal to 0, throw invalid argument, so withdraw. So, you are trying to withdraw, let us say less than or equal to 0. So, then it is a, it is also observed. So, then it has to throw the exception invalid underscore argument, withdrawal amount must be positive. All right.



```
31 // Function to withdraw money
32 void withdraw(double amount) {
33     if (amount > balance) {
34         throw runtime_error("Insufficient funds! Cannot withdraw that amount.");
35     }
36     if (amount <= 0) {
37         throw invalid_argument("Withdrawal amount must be positive!");
38     }
39     balance -= amount;
40     cout << "Withdrawal successful! New balance: Rs." << balance << endl;
41 }
42
43 // Function to display account information
44 void display() {
45     cout << "Account Holder: " << accountHolder << endl;
46     cout << "Current Balance: Rs." << balance << endl;
47 }
48 };
```



So here it is a runtime error. And here it is an invalid argument. Otherwise, balance equal to balance minus amount. Suppose I have 2000 rupees and I'm withdrawing only 1000 rupees. Yeah, my remaining will be 1000.

So, what you are doing in line number 39. And then you are printing what the balance is. Withdrawal successful and new balance is what. So, and then void display account holder name. Right?

Account holder and current balance, balance. Fine? So, this is how the class is being ended here. So, now go to the main program. In main, you start with try.

Right? Bank account, you have an object account passing Pradeep and 1000.0. Right? You have an object account. That means it will call the constructor.

So, the constructor is passing the name and initial balance. The name is Pradeep. And the initial balance is 1000 rupees, right? So, the initial balance is greater than 0. Therefore, it comes here: the account holder is now Pradeep, and the balance is 1000, okay?

And then you are displaying this account object, which is invoking the display member function. So, the display member function is here. So, the account holder is Pradeep, and the current balance is 1000 rupees. Now, the account is invoking a deposit of 500 rupees, right? So, if 500 rupees is being deposited, you see what will happen: deposit will call deposit. 500 rupees is being deposited, right? The amount is 500, which is greater than 0.

So, it will come here: balance equals balance plus amount. 1000 plus 500, so you will have 1500. So, initially you had 1000, now it is 1500. The account

holder's name is Pradeep, and the balance is now 1500, right? So, now the account is withdrawing 2000 rupees.

```

12 public:
13     // Constructor
14     BankAccount(string name, double initialBalance) {
15         if (initialBalance < 0) {
16             throw invalid_argument("Initial balance cannot be negative!");
17         }
18         accountHolder = name;
19         balance = initialBalance;
20     }
21
22     // Function to deposit money
23     void deposit(double amount) {
24         if (amount <= 0) {
25             throw invalid_argument("Deposit amount must be positive!");
26         }
27         balance += amount;
28         cout << "Deposit successful! New balance: Rs." << balance << endl;
29     }
30 }

```

Account is invoking the withdraw function, and 2000 is being passed. So now your balance is 1500. 2000 is the right balance, but 1500 is your actual balance. What is the withdrawal? The withdrawal is 2000. So, you know what will happen, right? So now, it is not possible.

Your account has only 1500 rupees, but you are trying to withdraw 2000 rupees. So in that case, here you go—your withdrawal, right? If the amount is greater than the balance, correct? The amount is 2000, and the balance is 1500. So, runtime error.

So this will throw, right? Runtime error. Insufficient funds—cannot withdraw that amount, right? So that means it is calling a runtime error, throwing a runtime error. That means when it is throwing, it will be caught here, right?

Because it is a runtime error. So, error will be printed, alright, C error, error will be printed. So, initially up to balance not a problem. And then when E is invoking the inbuilt function what under standard exceptions, alright, so the following will be printed. So, STD out, name of the account is Pradeep, right.

Up to here, we have to anyway display, right. Based on the display, the amount will be displayed, right. So, because deposit successful, this will also call, right. Account deposit, when you put 500, here you go, deposit. Yes, this will also be printed.

This we discussed already, correct. Up to here will be printed. That is why you are getting in the see out statement. So, Pradeep name of the account holder is Pradeep correct and initially when it is printing up to here right when the account display it will be printing current balance is 1000. Once he deposits 500 the new balance will be rupees 1500 right where is the error coming when he is trying to withdraw 2000 rupees because the amount is 1500.

You are trying to withdraw 2000. So therefore, you will get the error. Right. So this error and E dot what when E is invoking what we are getting the output like this. Right.

So this is your runtime error when it is throwing this exception. This will be caught. Right. This is being thrown. Right.

So this will be caught over here. Error and that particular statement insufficient funds cannot withdraw that amount. So insufficient funds cannot withdraw the amount is being printed. Right. So you are taking care the error handling exceptions.

So now you go through both the case studies. Both the case studies, one is divided by 0, another one, this is a very good example, other than divide by 0. So, this is a practical example. When the banking system, you are writing a quote, when the balance or withdrawal, the withdrawal is greater than balance, what is happening? Or somebody, I mean, simply write minus 15.

As a deposit. Or 0 as a deposit. Which is also not possible. Right. So you have handled this.

With the help of runtime error. As well as invalid arguments. Okay. I hope you understood both the case studies. So now.

On this. Suppose. In the case of Java. Right. So, types of exceptions.


In Java. So, there are two types of exceptions in Java. One is a checked exception. And another one is unchecked. Right?

### Types of Exception in JAVA

- There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception.

#### 1) Checked Exception

- The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions
- e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.



So, whereas the error is considered as an unchecked exception, right? So, checked exceptions, the classes that extend the throwable class except the runtime exception and error, are called checked exceptions. So, for example, input-output exception, SQL exceptions, these are all considered as Checked exceptions, and these are all checked at compile time, right? Checked exceptions are checked at compile time.

Whereas unchecked exceptions occur during runtime, right? So, what are all the unchecked exceptions? The runtime exceptions, right? The classes that extend runtime exception are called unchecked exceptions. So, here you go.

The arithmetic exceptions, null pointer exception, array index out of bound exception. So, these are all considered unchecked exceptions, right? So, in Java, you have checked exceptions and unchecked exceptions. So, the same divide by zero, alright. So, we know that when we try 50 divided by 0.

So, by default, you will get an So, we have seen standard exceptions in C++. In a similar way, in Java, suppose if I run the code, a very simple code, right? So, this is your driver class, correct? And here you have the main, a equals 50 divided by 0, and then you are trying to print a.


All right. So by default, you will get this error. Correct. So that is nothing but an arithmetic exception error. So here you have the arithmetic exception because you have divided by 0.

All right. So this is a scenario where an arithmetic exception occurs. So runtime unchecked exceptions. Correct. And in the second case, you have a null pointer exception.

So, a null pointer exception—you may ask the question whether there is a pointer concept in Java. S, this is an example. String S, which is pointing to null. So, a null pointer exception—string S is pointing to null. And then what are you doing? You are trying to find out the length of this string.

### 2. Scenario where NullPointerException occurs

```
1 public class NULLPtrExc{
2
3     public static void main(String []args){
4         String s=null;
5         System.out.println(s.length());//NullPointerException
6     }
7 }
8
```



S is pointing to null. S is not pointing to anything. And then you are trying to print S.length. So, what will happen? You will get


A null pointer exception. Alright. Null pointer exception. So, this is a null pointer exception. So, S is equal to null.

S is a string which is not pointing to anything. Correct. S is a string and it is not pointing to anything. You call it null. And then you are trying to print.

The string is not pointing to anything. It is an empty string. Correct. Not only an empty string. S is pointing to null.

### 2. Scenario where NullPointerException occurs

```
1 public class NULLPtrExc{
2
3     public static void main(String []args){
4         String s=null;
5         System.out.println(s.length());//NullPointerException
6     }
7 }
8
```



Terminal

```
sh-4.3$ javac NULLPtrExc.java
sh-4.3$ java NULLPtrExc
Exception in thread "main" java.lang.NullPointerException
    at NULLPtrExc.main(NULLPtrExc.java:5)
sh-4.3$
```

Null pointer. Right. And then you are trying to print `S.length`. So, the exception will occur.

The exception is nothing but a null pointer exception. So, when you run the code in the terminal, you get a null pointer exception. And the third scenario is a number format exception. So, this example I talked about in the last class.

So when you are converting let us say string `s` is `abc` right. So let us consider the string `s` is `abc` and here what you do the object `Integer` right which is in the wrapper class which is invoking `parseInt` of `s` that is trying to convert the string into integer. You can convert the character into integer but string into integer which is not possible right. `parseInt` of trying to call this function by passing the string `a`, `b`, `c` you are passing.

So, what is the integer equivalent of `a`, `b`, `c` right. So, this is not possible and when you are trying to print this, this will throw an error line number 5 you will get an error right. So, it is trying `a`, `b`, `c` the input string and then where you see right. the line number 5, right? So, you are getting number format exception, right?

Here you are getting the error number format exception. So, the `Integer.parseInt` So, you are having the problem over here, right? When you are trying to convert the string into integer, it is not possible. And the exception, the name of the exception is number format exception.

And this is the famous one: array index out of bounds exception, all right? So, in Java, you have to be a little careful. So, here you are declaring a dynamic array of length 5, right? Here you are declaring 5. The dynamic array of length 5.

Right. And then you are trying to access the 10th index. 0 is the first index. Trying to access the 10th index. And then you are putting `A[10]` equal to 50.

Right. You try this in C++. This will work. Right. Array of 5 elements.

#### 4. Scenario where `ArrayIndexOutOfBoundsException` occurs

```
1 public class ArrayExcept{
2
3     public static void main(String []args){
4         int a[]=new int[5];
5         a[10]=50; //ArrayIndexOutOfBoundsException
6         System.out.println(a[10]);
7     }
8 }
```



Particularly, a static array you take. A static array you take. A of 5. A dynamic array also should work. Right.

Whereas in Java, you have to be a little careful. Right. When you are trying, let us say, A of 10 equals 50. But you have declared only an array of 5 elements. You have declared an array of 5 elements.

A of 10 is 50. Correct? So, therefore, your `System.out.println A of 10`. So, I can access A0, A1, A2, A3, A4. When I am trying to access A10, therefore I will get an error: array index out of bounds exception, right.

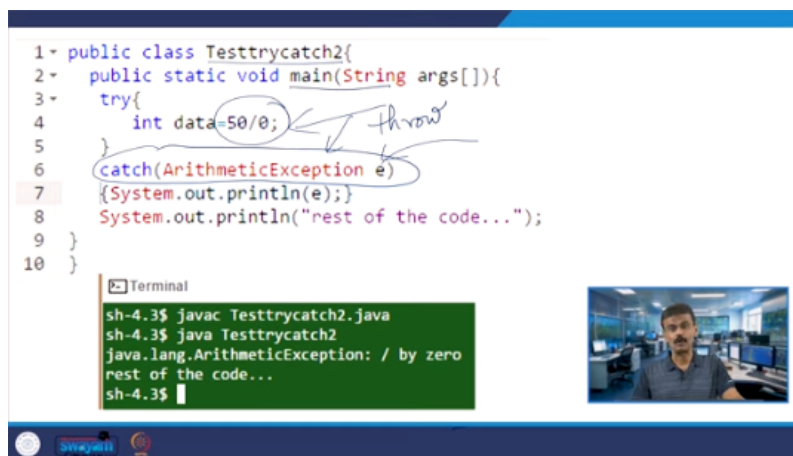
So, you can call you can see here array index out of bound. When you run the code, so the first line is a compilation, second is you are running the code, you are getting the output array index out of bounds exception. So, you cannot even you cannot access A5, you can access only A0, A1, A2, A3 and A4. these five elements because you have declared the dynamic array of five elements in java okay and then you are trying to access here in this particular case 10th index so that is not possible so java very sensitive and it is giving array index out of bounds exception so in c plus plus we have seen how many keywords try catch and throw right so we have seen try catch and throw so here additionally you have finally and throws right.

So, these are all additionally we have. So, now with the help of try and catch right with the help of try and catch in fact it will throw the arithmetic exception correct. So, let us see how this works it is almost like C++ right. Assume that you have a driver class called test try catch 2 right. So, test try catch 2 right and you have this is the test driver class.

You have the main program over here, right? So, here you are trying data equal to 50 by 0, right? When data is equal to 50 by 0, right? So, here it will throw the exception. It will throw the exception.

In C++, we use throw, right? So here, automatically it will throw the exception, right? And this exception will be caught here. This exception will be caught here. That means it will invoke the catch function, right?

So, this is the arithmetic exception as the class, and E is the object, right? And then, what are you doing? You are trying to print E, right? You are trying to print E. What is the exception? What is that exception?



```
1- public class Testtrycatch2{
2-     public static void main(String args[]){
3-         try{
4-             int data=50/0;
5-         }
6-         catch(ArithmeticException e)
7-         {System.out.println(e);}
8-         System.out.println("rest of the code...");
9-     }
10 }
```

Terminal

```
sh-4.3$ javac Testtrycatch2.java
sh-4.3$ java Testtrycatch2
java.lang.ArithmeticException: / by zero
rest of the code...
sh-4.3$
```

We are trying to print that exception. So, this will throw the exception. So, 50 divided by 0 will throw the exception, and here you are printing what that exception is—the first print statement. And the second print statement—the rest of the code. So that means here, you have taken the flow of the code.

Suppose you are not writing this catch statement. Let us say without try-catch, in fact, we have seen 50 by 0. It is calling the standard Java exception, right? And then we had seen what the output we got was. It will give an exception error, right?

So here, you have carefully taken care of 50 by 0, which will be throwing the exception, and the exception you are printing over here. What is the exception? And then one more System.out.println statement, the rest of the code. So, you can see here when I run the code. Compiling and running the code.



So, this is the exception. `Java.lang.ArithmeticException: divided by 0` is the error. Right. In fact, exception. Right.

For this print statement, this is being printed. `Java dot lang dot arithmetic exception`. What is that? Divide by 0. You are printing here.

And then line number 8, rest of the code you are printing, right? So, this is how you can handle the exception in Java. So, I had used, right, the Java exceptions. I can handle this by these keywords. So, in fact, here in this case, we have seen try and catch, right?

So, here we can find it is automatically throwing the exception. In this line, the exception is being thrown. Okay. So, in today's lecture, we talked about two case studies in C++ and I introduced in the case of Java, what are all the exceptions and we have seen four different scenarios and then we talked about this try catch and in fact, line number four, it is throwing the exception. So, we know how try catch throw work in Java as well as in C++.

So, in the next class, we will talk about multiple catches. So, multiple catches. When multiple catches occur in the same program, how does it work? So, which one will be considered? So, that we will see in the next class.

Thank you.