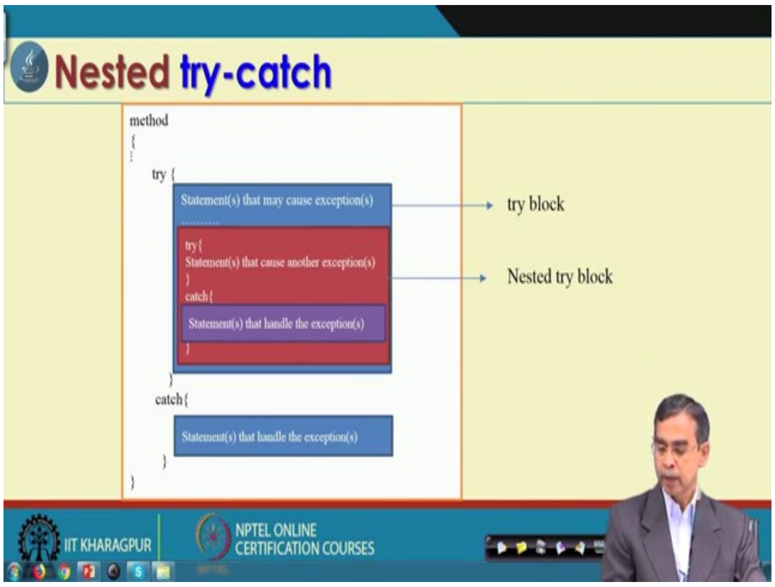
Programming in Java
Prof. Debasis Samanta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 25
Exception Handling – III

So, let us learn few more things regarding exception handling in java programming. We have discussed about the exception handling features mainly the try-catch finally, throw and throws.

(Refer Slide Time: 00:32)



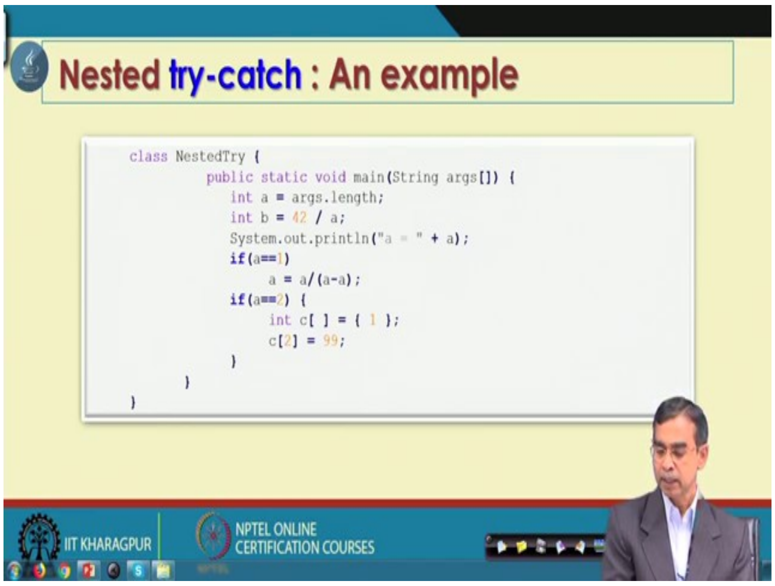
Nested try-catch

```
method
{
  :
    try {
        Statement(s) that may cause exception(s)
        ..........
        try{
        Statement(s) that cause another exception(s)
        }
        catch{
            Statement(s) that handle the exception(s)
        }
    }
    catch{
        Statement(s) that handle the exception(s)
    }
}
```

Now, in this module, we will discuss about the built in the different packages. And before that we should discuss few more things they are the so that try-catch. So, for try-catch concept as we have learned about, there is a scope that the nested try-catch can be implemented. Now, here is an example to include this concept called the nested try-catch. It is a nested means the try-catch block within another try this is the concepts. So, and there is no limit of nesting, try-catch block within try, within that try-catch block another try-catch like this, this. So, and then obviously that what is the reason, what is the purpose; why you should do that there obviously, it is the questions.

Now, the basic concepts, so for the nested try-catch can be understood here from this. So, suppose this is a method which is under the investigation of handling exception. And here if you see there is one what is called the inner try-catch, sorry outer try-catch block, within this try-catch we can define another try-catch is called the inner, within this also another try-catch block can be define like this. So, if we define a try-catch block within another try-catch block, then search a try (Refer Time: 01:49) is called nested try-catch block.

(Refer Slide Time: 01:57)



Nested try-catch : An example

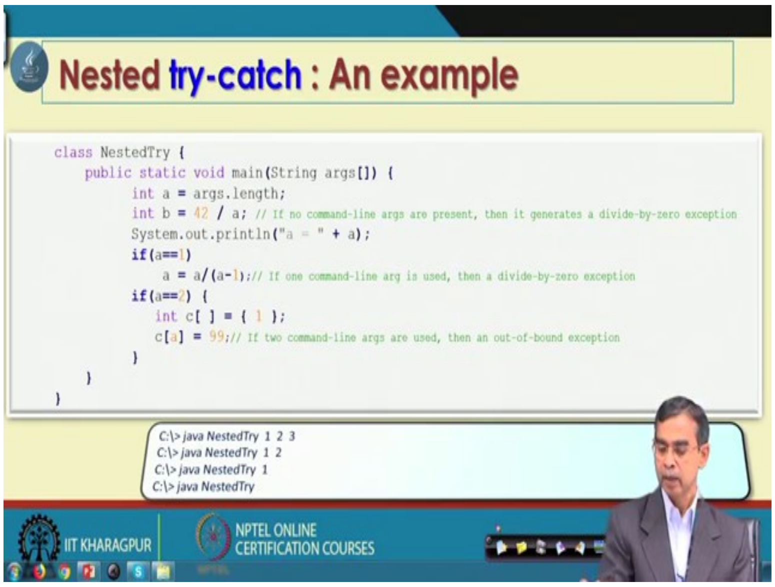
```
class NestedTry {
        public static void main(String args[]) {
          int a = args.length;
          int b = 42 / a;
          System.out.println("a = " + a);
          if(a==1)
               a = a/(a-a);
          if(a==2) {
               int c[ ] = { 1 };
               c[2] = 99;
          }
        }
}
```

Now, let us see one example where it can be applicable. This is again simple example that we can thing about ok. These are very simple program, it has only main method. So, you want to handle the exception in this method only. So, here is basically a args dot length, for the args dot length means whatever the input that user past according the number of input, this value a will be decide. So, if user do not pass any input, a will be 0; and accordingly the value of a will be there. Now, here if user do not pass any input, then a is 0; and as we see there is an exception call divide by 0 or arithmetic exception.

Now, again let us see here, suppose user enter only one input, so in that case a equals to 1. Now, in that case also it will give an error. So, it is also the divide by 0 error. Now, another situation so it is not the 0 input it is not the single input. So, suppose user enter two input, the in that case if you see if a equals to 2, then there is again possibility of error, because here we declare an array of integer c with only one element c need, that means, size of the array at represent is one, but we want to store a value at the second, that means, this is array index out of bound error is there.

Now, so here basically two type of errors. In the first two cases as we see here and here, this arithmetic exception error; and in the second case is there is called the array index out of bound exception occurred, so two type of error. Now, if two type of errors, then we can think for only two catch block. Now, here the idea is that we can write one try block here. So, this catch block, this try-catch block will handle these are the arithmetic exception error. Then there is another error that may occur within this, so inside this another try-catch block we can put, so that it can handle this exception. So, this is a concept, this is a concept of nesting. So, this is a simple example.

(Refer Slide Time: 04:11)



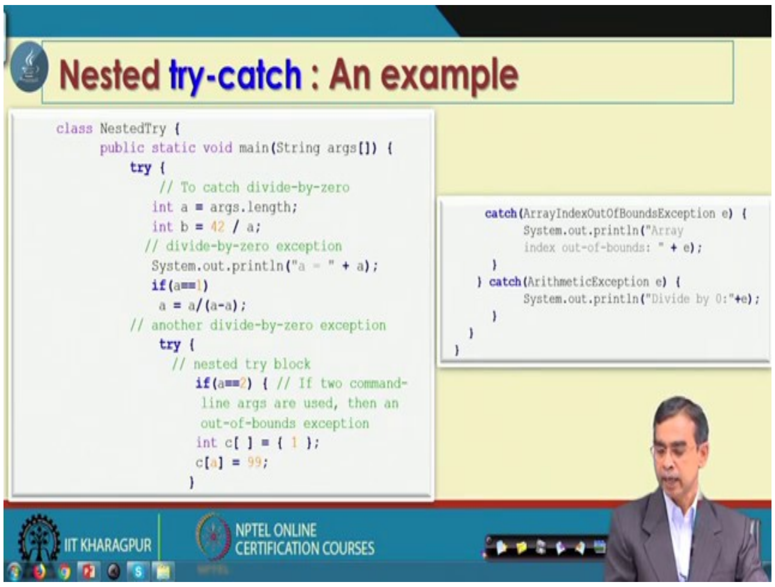
Nested try-catch : An example

```
class NestedTry {
    public static void main(String args[]) {
          int a = args.length;
          int b = 42 / a; // If no command-line args are present, then it generates a divide-by-zero exception
          System.out.println("a = " + a);
          if(a==1)
              a = a/(a-1);// If one command-line arg is used, then a divide-by-zero exception
          if(a==2) {
             int c[ ] = { 1 };
             c[a] = 99;// If two command-line args are used, then an out-of-bound exception
          }
    }
}
```

```
C:\> java NestedTry 1 2 3
C:\> java NestedTry 1 2
C:\> java NestedTry 1
C:\> java NestedTry
```

So, here is a solution for the nesting case is there. As we see this program will not run for all the input because it will give an error whatever it is there. It will run for this one no error, but for this one it will error.

(Refer Slide Time: 04:26)



Nested try-catch : An example

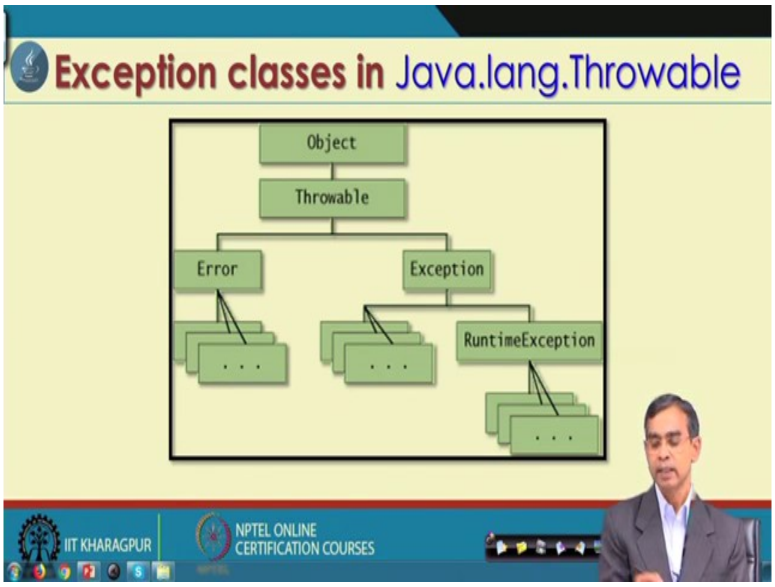
```
class NestedTry {
    public static void main(String args[]) {
        try {
            // To catch divide-by-zero
            int a = args.length;
            int b = 42 / a;
           // divide-by-zero exception
            System.out.println("a = " + a);
            if(a==1)
             a = a/(a-a);
         // another divide-by-zero exception
            try {
              // nested try block
                if(a==2) { // If two command-
                 line args are used, then an
                 out-of-bounds exception
                int c[ ] = { 1 };
                c[a] = 99;
               }
```

```
            catch(ArrayIndexOutOfBoundsException e) {
                System.out.println("Array
                index out-of-bounds: " + e);
            }
          } catch(ArithmeticException e) {
                System.out.println("Divide by 0:"+e);
            }
        }
    }
```

Now, so the remedy here we have checked it here. So, let us see this program as we have see this is the try and corresponding to this. So, this is there is a two try-catch block. As we see in this program this is the one try and this is the corresponding catch, so try-catch and this is a inner try-catch. And outside this, this is another try and this is another catch. So, so it is called the nested try-catch. Now, if we see this inner try-catch, we will handle the exception, this is the array index out of bond. Whereas, this try will handle the case this one, and then finally, this is work. So, this is idea about the nested try-catch block.

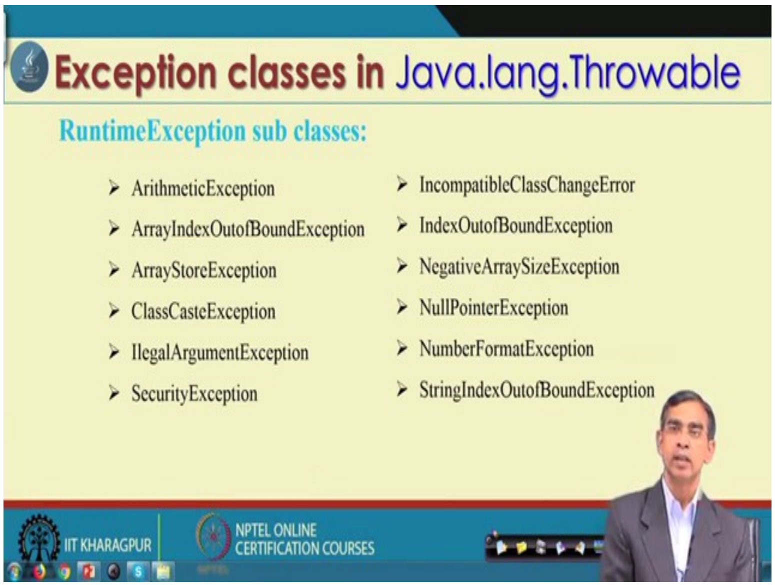
So, java is basically is a very versatile programming language in the sense that whatever the possible exception and errors may occur in a program, it can handle. So, regarding this things, the java developer or jdk package includes a very nice one package java dot lang package. This package has been planned to, so that it can take care many errors and exception.

(Refer Slide Time: 05:46)


Exception classes in Java.lang.Throwable

So, this is the class hierarchy that this java lang package has regarding that exception handling. So, here basically there is a class called object is defined in this java dot lang. Throwable class, it is called the class throwable. Under this Throwable, these are the basically hierarchy of the different sub classes of the class Throwable. So, object is the class super class under the super class Throwable is basically another inter. And this throwable has many at so this basically Throwable class as the two sub classes Error and Exception. Now, the simple exception is there, and then runtime exception is there now. So, so these are the different what is called the class hierarchy that is there in the java dot lang dot Throwable classes are there. Now, for each classes there are different what is called the exceptions that may occurs under this classes, so that they can handle it.

(Refer Slide Time: 07:03)



Exception classes in Java.lang.Throwable

RuntimeException sub classes:

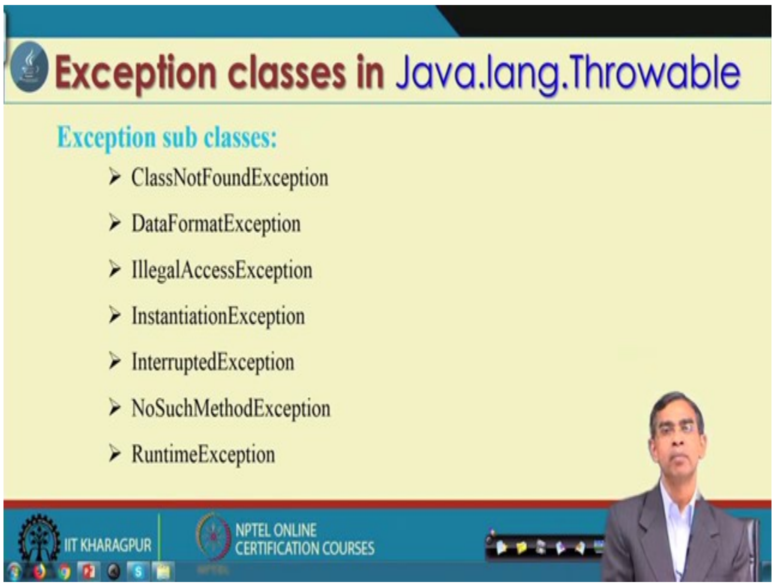
- ArithmeticException
- ArrayIndexOutofBoundException
- ArrayStoreException
- ClassCasteException
- IlegalArgumentException
- SecurityException
- IncompatibleClassChangeError
- IndexOutofBoundException
- NegativeArraySizeException
- NullPointerException
- NumberFormatException
- StringIndexOutofBoundException

So, here is basically let us first discuss about the runtime exceptions sub classes which basically causes many exception. So, I exceptions are discussed here. As we see here that five around ok, so around 12, 18th, 18th, 6, 6, 12 ok. So, 12 different type of exception that may occur during runtime exception sub classes which it can cause.

For example, we have already discussed about array arithmetic exception, array index out of bound exception, and then there is a many more array this illegal argument exception, security exception, incompatible class change error index out of bound exception negative array size exception, null pointer exception, number format exception, string index out of bound exception, few exception already we have discussed in our discussion and few more exception that we will learn about while we have a demonstration on this concept, so that it will clear our idea.

Now, so these are the different type of exceptions that the java developer automatically these are built in exception types which is already there in the runtime exception category of the Throwable class. Now, so these are the frequently occur exception related to the different situations it is there.

(Refer Slide Time: 08:27)


Exception classes in Java.lang.Throwable

Exception sub classes:

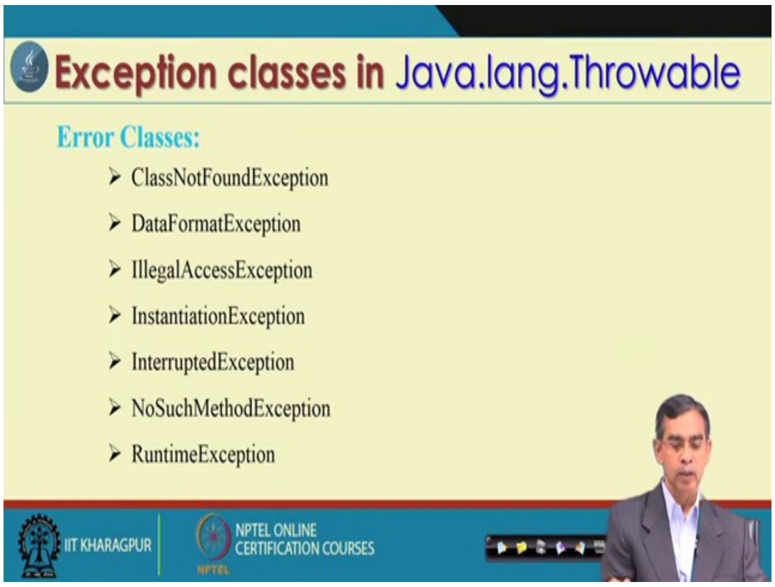
- ClassNotFoundException
- DataFormatException
- IllegalAccessException
- InstantiationException
- InterruptedException
- NoSuchMethodException
- RuntimeException

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Now, other also there is another class exception sub class also there. It is basically in this exception sub class, there are few more exception that it may handle is a ClassNotFound Exception, and then DataFormatException, IllegalAccessException, Instantiation Exception, InterruptedException, NoSuchMethodException and RuntimeException.

So, these are the different type of exception it is usually if you are an advance java programmer, then he may phase this kind of exception when you are developing applications. So, in that case all the coming to the picture, but it is very difficult to illustrate all this exception one by one, because it is too more I mean it is also not possible in this shortest time anyway. So, these are the different exception. So, for RuntimeException and then exception sub classes is concern and there is one is called the Error sub class.

(Refer Slide Time: 09:24)



Exception classes in Java.lang.Throwable

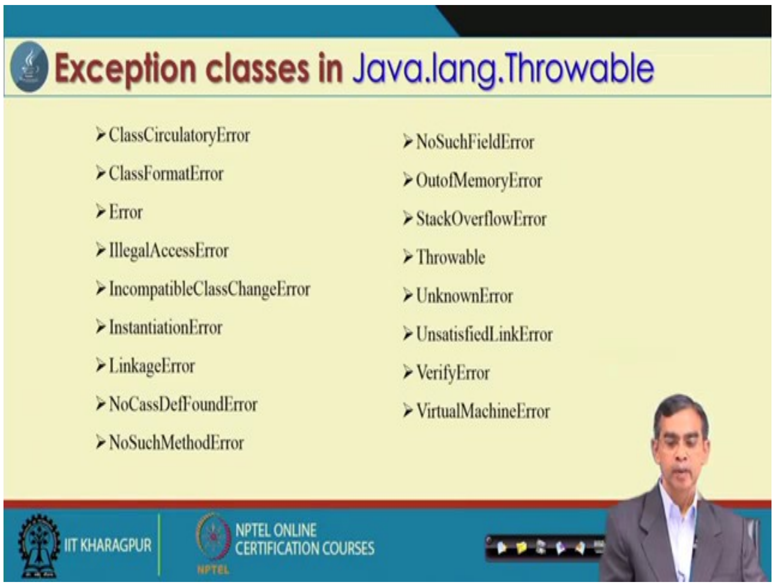
Error Classes:

- ClassNotFoundException
- DataFormatException
- IllegalAccessException
- InstantiationException
- InterruptedException
- NoSuchMethodException
- RuntimeException

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

So, in the error sub class also, you can see these are the different exception that it may occur ok. So, this is the error classes, and this is the exception sub classes.

(Refer Slide Time: 09:44)



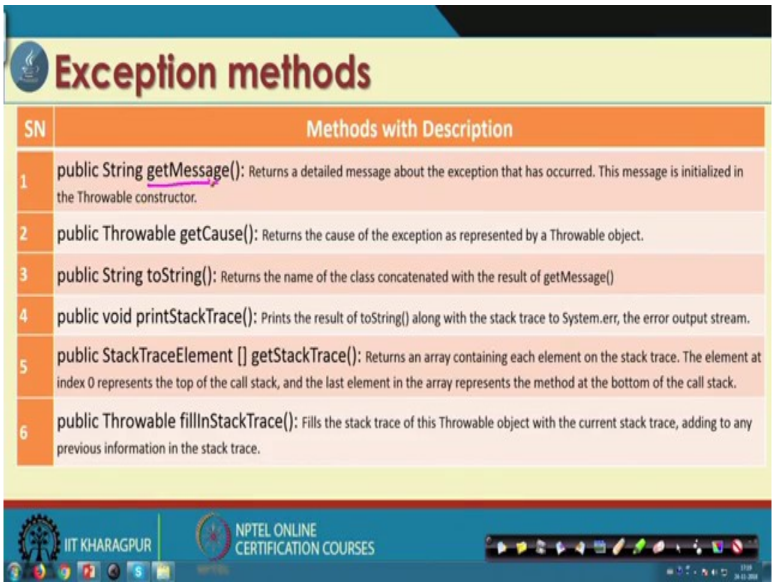
Exception classes in Java.lang.Throwable

- ClassCirculatoryError
- ClassFormatError
- Error
- IllegalAccessError
- IncompatibleClassChangeError
- InstantiationError
- LinkageError
- NoCassDefFoundError
- NoSuchMethodError
- NoSuchFieldError
- OutofMemoryError
- StackOverflowError
- Throwable
- UnknownError
- UnsatisfiedLinkError
- VerifyError
- VirtualMachineError

So, these are the few more errors that is there in the exception classes as we have mentioned here that ClassCirculatoryError,ClassFormatError, Error, simple error, and then LinkageError, InstantiationError, NoClassDefineFound Error, NoSuchMethod Error, NoSuchFieldError, OutofMemoryError, StackoverFlowError, Throwable ,UnknownError, UninsatisfiedLinkError, then VerifyError and

VirtualMachineError. So, there are many errors that may occur in a program which is discussed here.

(Refer Slide Time: 10:26)



Exception methods

SN	Methods with Description
1	public String getMessage(): Returns a detailed message about the exception that has occurred. This message is initialized in the Throwable constructor.
2	public Throwable getCause(): Returns the cause of the exception as represented by a Throwable object.
3	public String toString(): Returns the name of the class concatenated with the result of getMessage()
4	public void printStackTrace(): Prints the result of toString() along with the stack trace to System.err, the error output stream.
5	public StackTraceElement [] getStackTrace(): Returns an array containing each element on the stack trace. The element at index 0 represents the top of the call stack, and the last element in the array represents the method at the bottom of the call stack.
6	public Throwable fillInStackTrace(): Fills the stack trace of this Throwable object with the current stack trace, adding to any previous information in the stack trace.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Now, all these sub classes that we have discussed is associated with many methods. Here we have listed few methods, and with their description. As you see here they get Message probably you have run about the getMessage is basically it will tell about a particular error. So, it is it basically returns a message about a particular exception that usually it over an encounter. So, it is basically this message is initialized through a constructor which is defined in Throwable class; this is the main class, super class.

Now, there is again get cause, it basically a return an object of class Throwable to the caller program if a method in which this exception occur, it will basically throw a throw an exception to the caller, so that is why get cause it basically tell about the reason for the exception that it have. So, it has very useful indication to both the programmer as well as for the user also. Then to string here it basically return the name of the class where this error occurs. It possibly concatenated if this class is under another class or is a nested class or derived class or whatever it is there, it will basically concatenated, all the classes then give a particular location. So, this is particularly very much useful for debugging, debugging the program and testing the program.

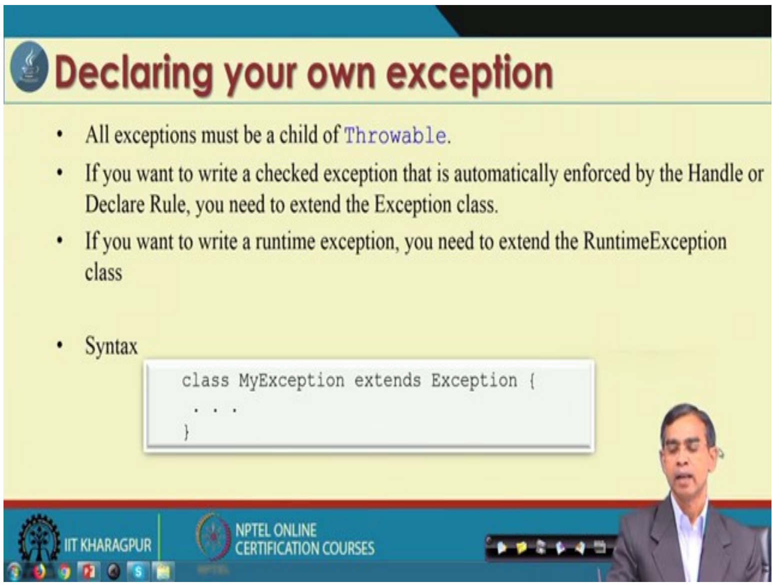
Then printStackTrace it is just like a idea about tracing there that means, if an error occurs in a particular method. If this method is declared in which class if this class is a derived class or not or implements and interfaces like this. So, it will basically give a full trace off the location where this particular exception occurs. Or if there is a chain of exception one exception can propel other exception, then other exception also using this printStackTrace, we will be able to trace the reason or the location of the exception.

Then stack trace element, it is basically and advanced version of this one, it basically it multiple errors are occurs, then it basically put on into the stack whenever we writing same recursive program suppose and there maybe multiple exception occurs in a different points. And then it will basically (Refer Time: 13:11) all the exception into a stack, and that is basically the idea about. And then finally, it will use this one.

So, this get stack trace is basically catch all the exception that it may happen in a method, and it will push on stacking actually it is stacking into a stack. And then fill in StackTrace also it basically this method returns a object returns and exception object of this throwable class. And whenever there is an exception occurs, and then obviously, in the so it basically think that whether stack is able to maintain handle this kind of exception or not. So, this is the (Refer Time: 13:56) is statement like. So, these are the different methods which are there in the exception class.

Now there is a possibility, so that depending on the needs of user, user can define their own exception. So, this concept is basically to make the program, more versatile, more flexible more robust and reliable. So, this is possible by means of user defined exception, in other words user can define their own exception.

(Refer Slide Time: 14:35)



Declaring your own exception

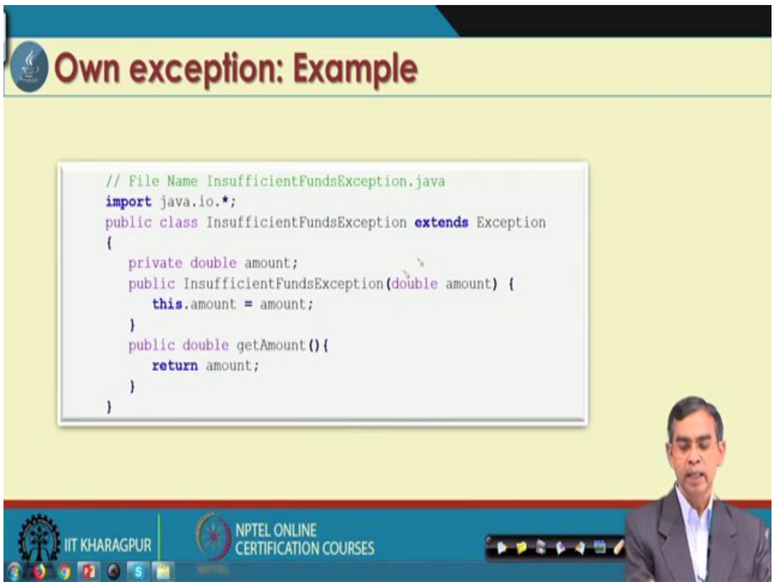
- All exceptions must be a child of Throwable.
- If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.
- If you want to write a runtime exception, you need to extend the RuntimeException class
- Syntax

```
class MyException extends Exception {
  . . .
}
```

So, here is an idea, let us see how user can define their own exception. Now, first of all you have to create a class that class should be a derived class of the super class throwable. So, it is basically all exception must be a child of Throwable class. We have already discussed about this throwable class is defined in java.lang package. Now, so this is the syntax by which we can create an exception which basically type of the runtime exception that can handle it.

So, here as we see class, myException it is basically user defined class. And extend Exception, Exception is basically is a sub class of Throwable, we have already learned about. And sometime the RuntimeException also we can declare, so there are different type ok. So, so this is the way by which user can define their class, and here the code of the exception class. Now, let us have an example, so that we can understand how user can define his own exception.

(Refer Slide Time: 15:46)



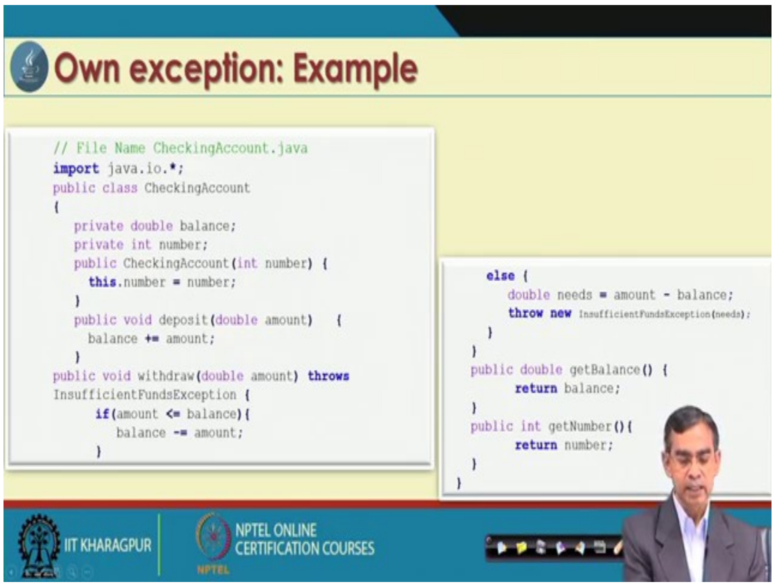
Own exception: Example

```
// File Name InsufficientFundsException.java
import java.io.*;
public class InsufficientFundsException extends Exception
{
   private double amount;
   public InsufficientFundsException(double amount) {
      this.amount = amount;
   }
   public double getAmount(){
      return amount;
   }
}
```

Now, this is the very simple program. And here we can see this is the user defined one class. This is the user defined one class. We define the name of the class that we have define is InsufficientFundsException which basically is a derived class of exception. And these are the a few quotes of this right. So, here private double amount is a one field, and then this is basically the constructor of this class, this will take the amount. And this dot amount it basically initialize. And then there is an another method public double getAmount( ) and returnAmount().

So, so the class is very simple. Without any exception this class can be considered only simple class a and then all this course can also be there. But we have to just make it attend exception class, this one of our own. Now, let us see how we can use it this is a one application that we are going to discuss in the context of banking transaction and then we can understand about it. Let us have the discussion so that we can discuss about it.

(Refer Slide Time: 17:05)



Own exception: Example

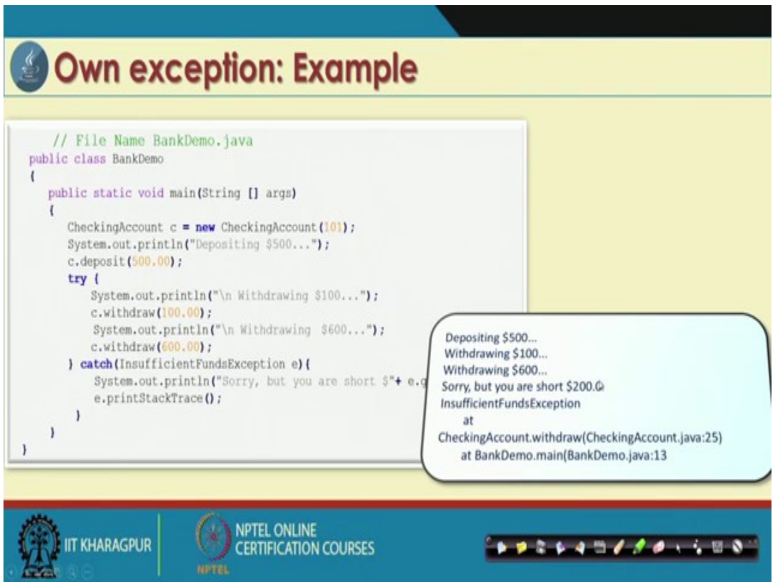
```
// File Name CheckingAccount.java
import java.io.*;
public class CheckingAccount
{
   private double balance;
   private int number;
   public CheckingAccount(int number) {
     this.number = number;
   }
   public void deposit(double amount)   {
     balance += amount;
   }
public void withdraw(double amount) throws
InsufficientFundsException {
      if(amount <= balance){
        balance -= amount;
      }
```

```
      else {
         double needs = amount - balance;
         throw new InsufficientFundsException(needs);
      }
   }
   public double getBalance() {
         return balance;
   }
   public int getNumber(){
         return number;
   }
}
```

Now, so this exception that we have discussed here we want to use it in our application class. Now, so application class is like this. So, there is a user defined class Checking Amount, checkingAccount. So, it will take the balance, then number, and then it is a constructor here. And this is another method deposit amount and the balance will be increased. So, this is a concept it is there. So, it account will be checked, and then the amount that will be deposited will be deposited into the account. And there is another method which is in this class is also there withdraw method, deposit method, withdraw method. And here in this case we have mentioned one exception throws, throws InsufficientFundsException .So, if we can find that if amount less than equals to balance, then balance will be negative or balance will be adjusted, and then if it is not, then it will throw an exception. So, this exception is basically it is there. So, suppose user one user wants to withdraw some amount which is not permissible as per the balance is concern, in that case it will throw this exception insufficient funds exception. So, this is the idea about.

Then once this class is declared, we can use we can define the main class here. Here you can see this is an extension of this one just ok, withdraw has this one else and this class has few more course of course. So, here is few course it is there. And then these are the simple getBalance and then getNumber, these are the complete description of the class has it is there, checking account class. Now, on this class is defined, then we will be able to define our main class.

(Refer Slide Time: 19:00)



Own exception: Example

```
   // File Name BankDemo.java
 public class BankDemo
 {
    public static void main(String [] args)
    {
       CheckingAccount c = new CheckingAccount(101);
       System.out.println("Depositing $500...");
       c.deposit(500.00);
       try {
          System.out.println("\n Withdrawing $100...");
          c.withdraw(100.00);
          System.out.println("\n Withdrawing  $600...");
          c.withdraw(600.00);
       } catch(InsufficientFundsException e){
          System.out.println("Sorry, but you are short $"+ e.g
          e.printStackTrace();
        }
    }
 }
```

```
Depositing $500...
Withdrawing $100...
Withdrawing $600...
Sorry, but you are short $200.0
InsufficientFundsException
    at
CheckingAccount.withdraw(CheckingAccount.java:25)
    at BankDemo.main(BankDemo.java:13
```

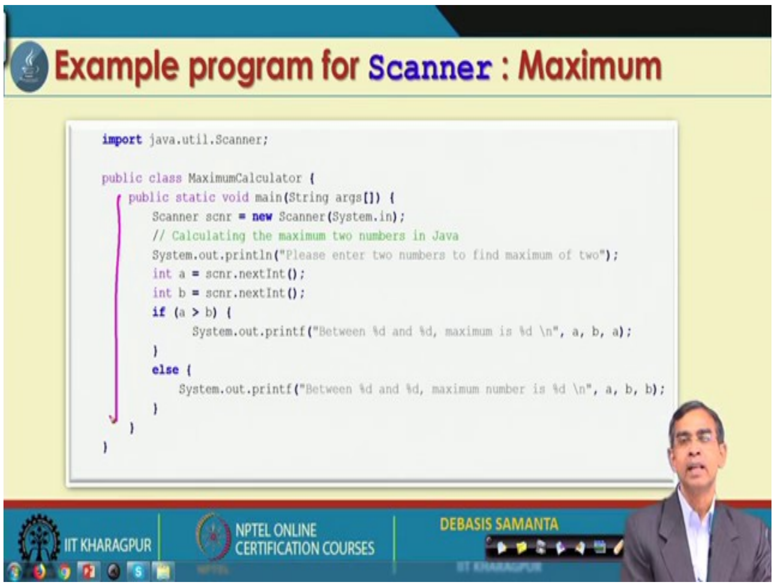
So, the main class it looks like this the name of the main class is BankDemo, and it has the main method. And we can see we have created an object of checkingAccount and then say let there may be 100 plus the account holder is there. So, these are size of the total customer base actually. And then system.out.println depositing dollar 500, this is a one method it is there. And then see deposit 500, so it is basically that ok. This for this object we want to write call the method deposit.

So, now, after this thing try catch is there, and then c.withdraw. So, this is the 100 withdraw one instance, 600 withdraw another instance. Now, in case, suppose there is any exception occur, then this is a catch block which will take care. So, this is our the fund exception, user defined exception as you can note it here

Now, so there are few instances that it may occur. If you run this program the entire class declaration that we have discussed and then run it. And then there are few instances of the input that we can see depositing dollar 500, withdrawing dollar 100, these are the things that we will sorry, but you are short of 200, insufficient funds exception at this one, it basically shows the different situation when this program is executed.

So, this way you the user can define their own exception, and then can develop the program according to their requirement ok. So, this is about the different cases of the exception that it may occur here. And we are almost at the final round of the exception handling discussion. Before going to this things I just want to mention few a simple example, let us see whether you can follow it or not.

(Refer Slide Time: 21:05)



Example program for Scanner : Maximum

```
import java.util.Scanner;

public class MaximumCalculator {
    public static void main(String args[]) {
        Scanner scnr = new Scanner(System.in);
        // Calculating the maximum two numbers in Java
        System.out.println("Please enter two numbers to find maximum of two");
        int a = scnr.nextInt();
        int b = scnr.nextInt();
        if (a > b) {
              System.out.printf("Between %d and %d, maximum is %d \n", a, b, a);
        }
        else {
            System.out.printf("Between %d and %d, maximum number is %d \n", a, b, b);
        }
    }
}
```

Here is the one program and just took a check this program and tell where is the possible scope of exception that it may result. As we see here so this is a simple class only one class which has the main method, this is the main method. Now, let us can one by one this statement. Now, I am telling one very simple rule of thumb is that if we suspect any exception that it may results, then that statement that code can be put under try-catch. So, in that catch, the very simple way that you can handle the exception is that so many try-catch corresponding this one, but this is obviously, makes the program to so many try-catch, but ultimately it is a robust of course.

But there are many way that is the rather efficient way that with minimum number of try-catch how we can handle everything. So, usually the advanced programmer think for this one that with minimum number of try-catch, whether this is try-catch nested, throws-throws you, with your own exception, whatever it is there, anyway. So, for a beginners, the best practice should be that if you see that there is an error so put a try-catch and accordingly handle it, you can use the finally, if you want if you do not want, then final is a default one. So, it is not necessary to use.

Now, so here again here if you see this is the one statement, do you think that there is a possibility of an exception occurred here. Yes, because if the Scanner class is not instantiated, and then create a scanner object, then it may leads to an exception. So, there is a possibility that an exception occurs. So, this is a one possibility that exception occur.

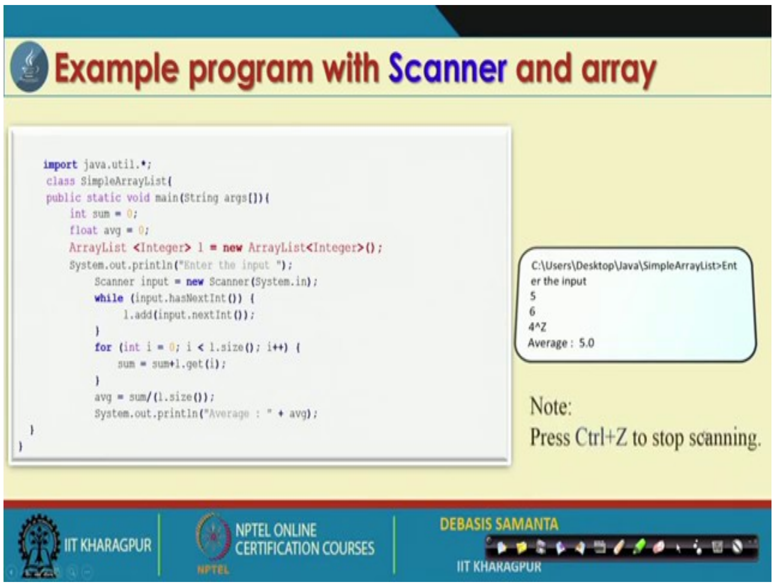
Now, next is here this statement and this statement as well as. So, nextInt is basically it check that in the line of input, whether the next input is there. If some regions are there, whenever this input is not there or not able to read integer rather some string and everything, then this statement also throws exception. And these are the very simple. So, there is no exception.

Now, so these are the point here, here, and here. Now, how many try-catch block that you can think for fine. So, one solution is that, all these things put can be put only try, and then here is the catch or as it is a try this is the one kind of exception and this is the only kind of exception. So, nested try also can be plan, so that with minimum number of try-catch code, we can handle the exception here in this case. So, it is the practice.

So, you can think about how you can put it. It is basically if you put many try-catch block, absolutely no error, no problem, because if the exception occurred then only it will be there. But the problem is that if you do not mention try-catch block properly, so that even in the case of exception occurred you are not able to catch it, then it is a problem, so that is a things are there.

And one more thing, so if you do not put try-catch block explicitly, but there is a possibility of exception occurs. So, during the compilation, the compiler can note it and then compiler can report it. Until you put try-catch in your block, compiler is not compile it successfully. So, this is the one great advantage for the programmer that ok, if you are supposed miss to add try-catch construct in your program, the compiler will help you to do that. So, there are many ways it is possible. So, this is the one example that we have discussed about in a scanner, and this is simple one example.

(Refer Slide Time: 25:07)



Example program with Scanner and array

```
import java.util.*;
 class SimpleArrayList{
 public static void main(String args[]){
     int sum = 0;
     float avg = 0;
     ArrayList <Integer> l = new ArrayList<Integer>();
     System.out.println("Enter the input ");
         Scanner input = new Scanner(System.in);
         while (input.hasNextInt()) {
             l.add(input.nextInt());
         }
         for (int i = 0; i < l.size(); i++) {
             sum = sum+l.get(i);
         }
         avg = sum/(l.size());
         System.out.println("Average : " + avg);
  }
}
```

```
C:\Users\Desktop\Java\SimpleArrayList>Ent
er the input
5
6
4^Z
Average : 5.0
```

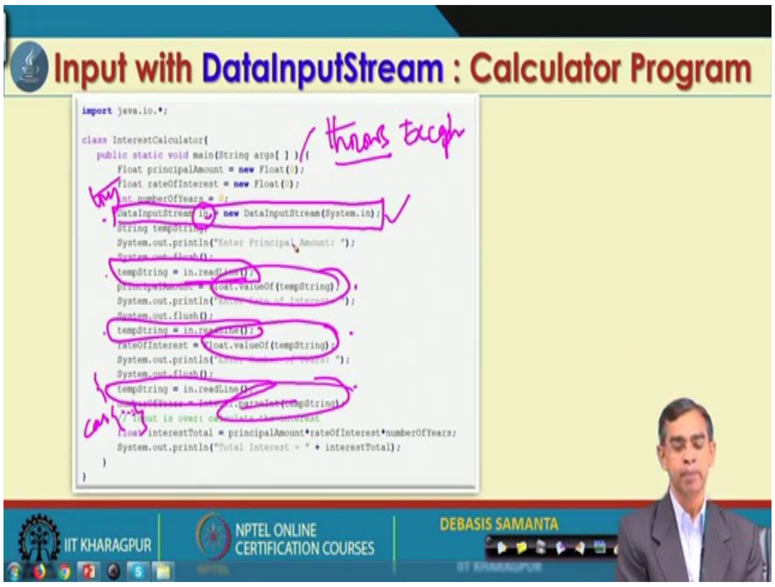
Note:
Press Ctrl+Z to stop scanning.

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

Now, let us consider another example. It basically is similar to the Scanner. This Scanner means it is basically read the input from the keyboard, and then store into an array. Here is an example as we see. And array list is a one structure data structure defined in java.util package. Now, let us see here. So, so this is a simple again code, the main method. And obviously, there is no point of exception, but here is a one point of exception that if the array list is not this l not built in properly or not created successfully. If it is not created successfully, then here is an error occurs, here is an error occurs, and then here is an error occurs. And also here another error occurs that l.size if you have created an array list, but this size of the array list is very small rather is a 0, then also it can created, it can creates errors.

Now, we have pointed out few things are there, again another also there is also possibility that one error is occur here. Now, again if I ask you that ok, how you can insert the try-catch block in this program, so that this program will be robust or if you can think about of your own throw, and then accordingly user-defined exception you can. So, if you want to have your own user-defined one, you should defined a exception classes which is a child class derived class of throwable or exception, and then according to that you can write the try, here the try, here the catch or here again try another catch, so that all the exception it can be handled in this program. So, the many ways actually that you can do. So, this is the one idea about.

(Refer Slide Time: 27:09)



Input with DataInputStream : Calculator Program

```
import java.io.*;

class InterestCalculator{
   public static void main(String args[ ] ){
      Float principalAmount = new Float(0);
      Float rateOfInterest = new Float(0);
      int numberOfYears = 0;
      DataInputStream in = new DataInputStream(System.in);
      String tempString;
      System.out.println("Enter Principal Amount: ");
      System.out.flush();
      tempString = in.readLine();
      principalAmount = Float.valueOf(tempString);
      System.out.println("Enter Rate of Interest: ");
      System.out.flush();
      tempString = in.readLine();
      rateOfInterest = Float.valueOf(tempString);
      System.out.println("Enter Number of Years: ");
      System.out.flush();
      tempString = in.readLine();
      numberOfYears = Integer.parseInt(tempString);
      // Input is over: calculate the interest
      Float interestTotal = principalAmount*rateOfInterest*numberOfYears;
      System.out.println("Total Interest = " + interestTotal);
   }
}
```

Now, let us have another example ah. So, here is another example. And this example we have discussed while we are considering input and output to the system. This example as we see here we have fine, this is again simple method main method it is there. In this main method, these are the simple declaration of the two objects. So, this is also another declaration. So, these are declaration. There is no point of any exception occurs here just a matter of discussion a declaration only.

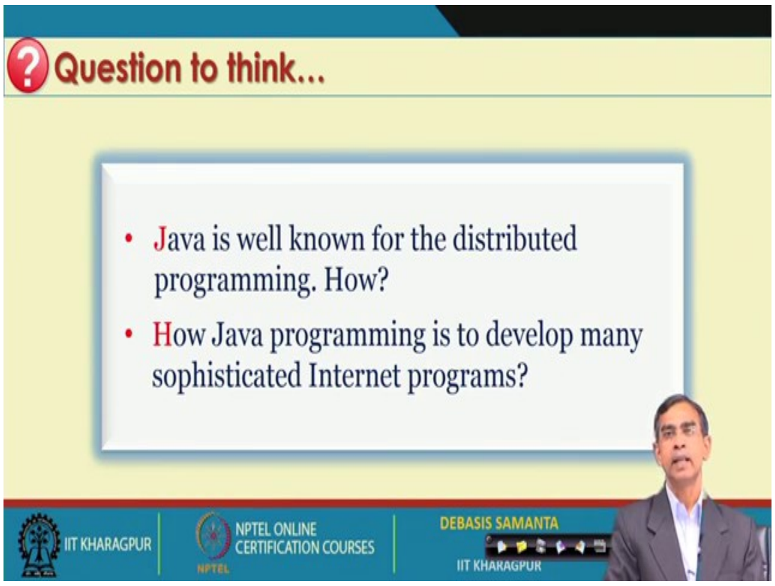
Now, here if we see here again this one this basically is the one very critical code here, because we have to create an object of data input stream and sometimes this may have problem. Sometimes I told you, not always, and exception is like that it can happen in some special situation like this. Now, in that case, so this statement should be covered under try-catch block. Now, if there is a problem and subsequently, they are also the possibility that there is a problem here either it is a readLine problem from the object in. There is also a readLine from the object in; there is also there and like this ok. So, one, two, three, four, the four points where we can see the exception occurs here..

And also here, so this is basically this is the one value of changing conversation string to float value, string to integer to parse in there is also possible scope of errors there. Now, as we see as they these are the possible, because you have entered some value which is not possible to convert an integer say 2 0 20.55, then in that case it basically not able to parse into integer, so that error will be there. So, as you see, so these are the four cases and this also other three cases that error may occur.

Now, once you identify this is a possible location that the exception can be thrown or exception can occur, then accordingly we have to take care about that try-catch. There are again many ways are there. First of all we can use the throw statement. So, what we can do is that, we can write here the throws, throws and exception. Then what about the causes are there, we can put try and then here catch, so try and then catch, then exception occur (Refer Time: 29:57). So, this is the one way. So, throws means it will basically throw all the exception that it may happens and then using this try-catch they can be exception.

Now, other than using the throw statement itself simply try and then the catch also can be put here exception. If you can know that this is the exception due to this, this is the exception due to this, then accordingly several catch block several catch block you can put it for each exception. So, this is a case of single try with multiple catch. And also one try with all exception in multiple exceptions by one catch also in that case similar to the throws also we can use it. So, in the many ways, many ways that you can do only if you know the mechanism, then all those ways that can be incorporated in your program, and thus you can make your program very robust.

(Refer Slide Time: 30:56)



Question to think...

- Java is well known for the distributed programming. How?
- How Java programming is to develop many sophisticated Internet programs?

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
DEBASIS SAMANTA
IIT KHARAGPUR

So, this is all about the exception handling concept in java programming. Now, after knowing the exception mechanism; our next tasks to learn about how the distributed programming can be done because java is very famous for distributed computing. So, in our next model, we will learn about the distributed programming more specifically the multithreading concept. And then, we will discussed the how java right language can helps a programmer to develop their internet programs. So, these are the concepts these are, I mean questions will be answer in our next module (Refer Time: 31:45).

Thank you very much.