Essentials of Information Technology

PC-CS-305

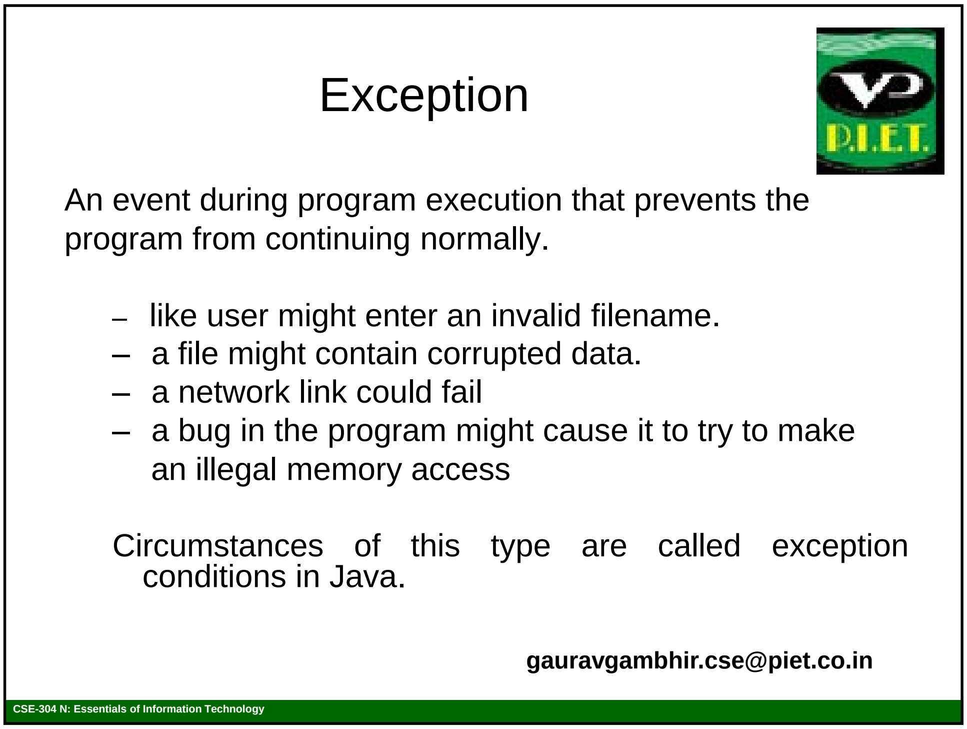
Exception Handling

Objectives

In this lecture, we will

- Define exceptions
- Use try, catch, and finally statements
- Describe exception categories
- Identify common exceptions
- Develop programs to handle your own exceptions

gauravgambhir.cse@piet.co.in



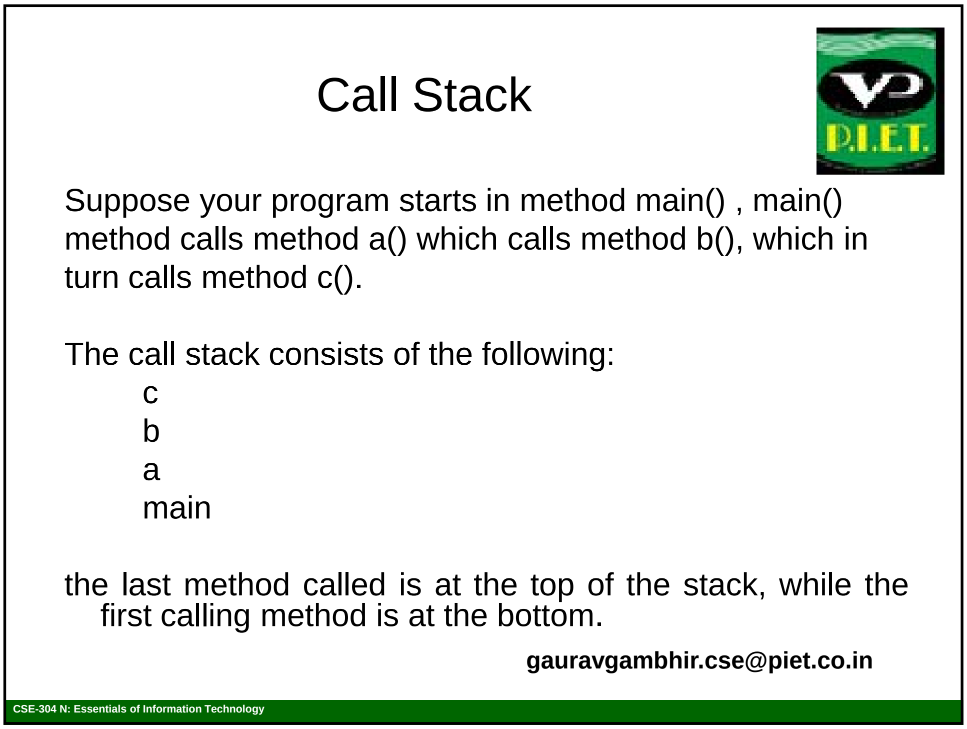
Exception

An event during program execution that prevents the program from continuing normally.

- like user might enter an invalid filename.
- a file might contain corrupted data.
- a network link could fail
- a bug in the program might cause it to try to make an illegal memory access

Circumstances of this type are called exception conditions in Java.

gauravgambhir.cse@piet.co.in



Call Stack

Suppose your program starts in method main() , main() method calls method a() which calls method b(), which in turn calls method c().

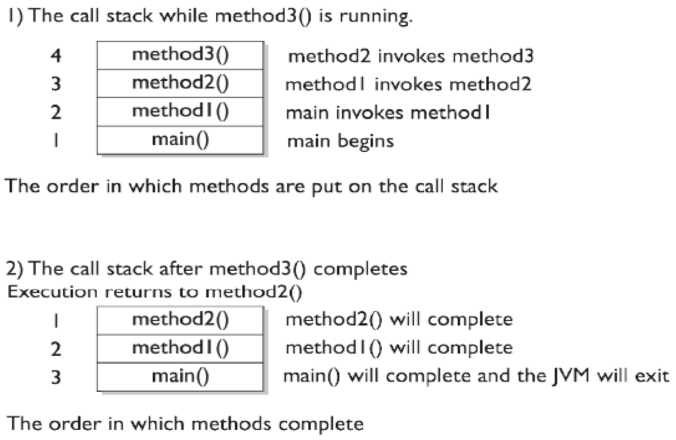
The call stack consists of the following:

c
b
a
main

the last method called is at the top of the stack, while the first calling method is at the bottom.

gauravgambhir.cse@piet.co.in

Call Stack



1) The call stack while method3() is running.

4	method3()	method2 invokes method3
3	method2()	method1 invokes method2
2	method1()	main invokes method1
1	main()	main begins

The order in which methods are put on the call stack

2) The call stack after method3() completes
Execution returns to method2()

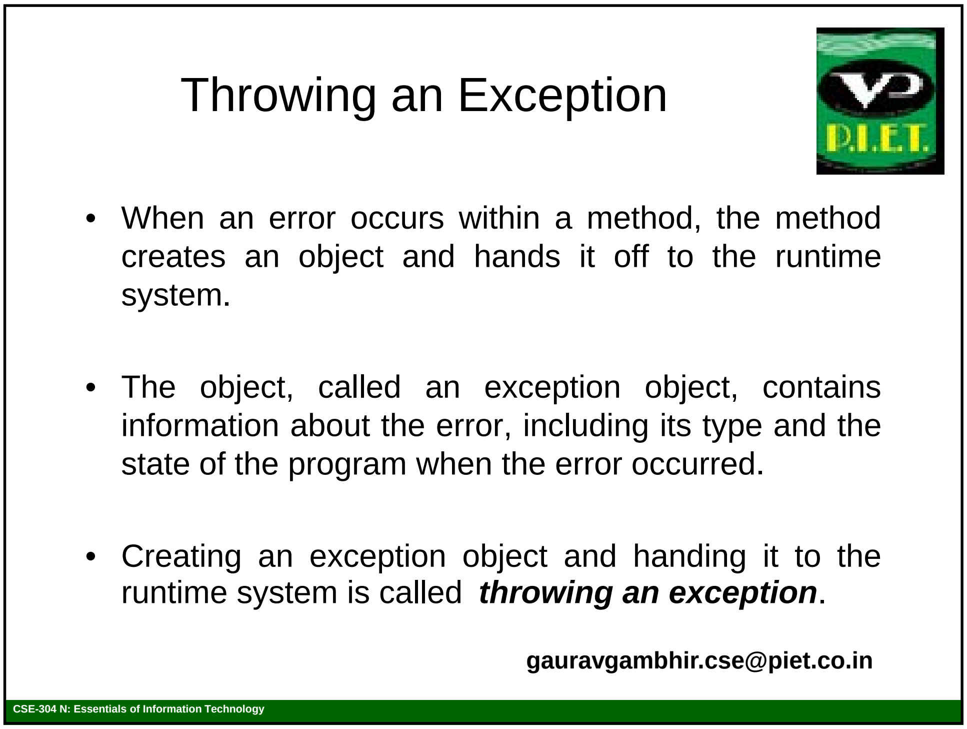
1	method2()	method2() will complete
2	method1()	method1() will complete
3	main()	main() will complete and the JVM will exit

The order in which methods complete

Exception Propagation

- An exception is first thrown from the top of the stack (in other words, the person on the roof)
- If it isn't caught by the same method who threw it, it drops down the call stack to the previous method.
- If not caught there, the exception again drops down to the previous method and so on until it is caught or until it reaches the very bottom of the call stack.
- This is called exception propagation.

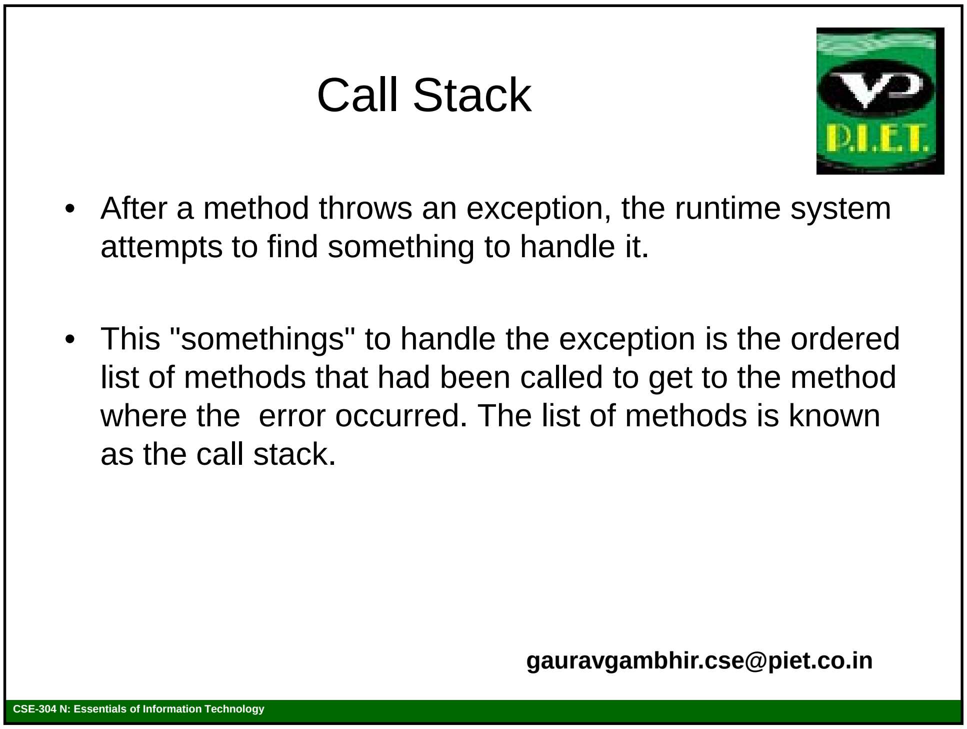
gauravgambhir.cse@piet.co.in



Throwing an Exception

- When an error occurs within a method, the method creates an object and hands it off to the runtime system.
- The object, called an exception object, contains information about the error, including its type and the state of the program when the error occurred.
- Creating an exception object and handing it to the runtime system is called ***throwing an exception***.

gauravgambhir.cse@piet.co.in

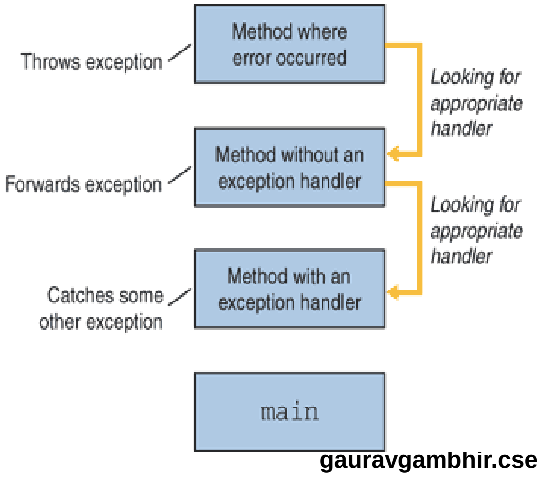


Call Stack

- After a method throws an exception, the runtime system attempts to find something to handle it.
- This "somethings" to handle the exception is the ordered list of methods that had been called to get to the method where the error occurred. The list of methods is known as the call stack.

gauravgambhir.cse@piet.co.in

Call Stack


Method where error occurred
Throws exception
Looking for appropriate handler
Method without an exception handler
Forwards exception
Looking for appropriate handler
Method with an exception handler
Catches some other exception
main

Exception Handler

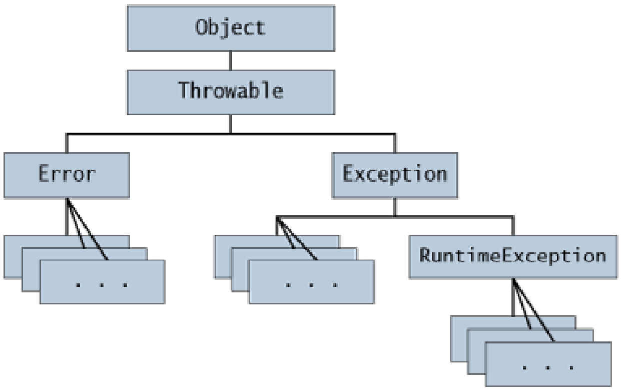
The runtime system searches the call stack for a method that contains a block of code that can handle the exception. This block of code is called an exception handler.

The search begins with the method in which the error occurred and proceeds through the call stack in the reverse order in which the methods were called.

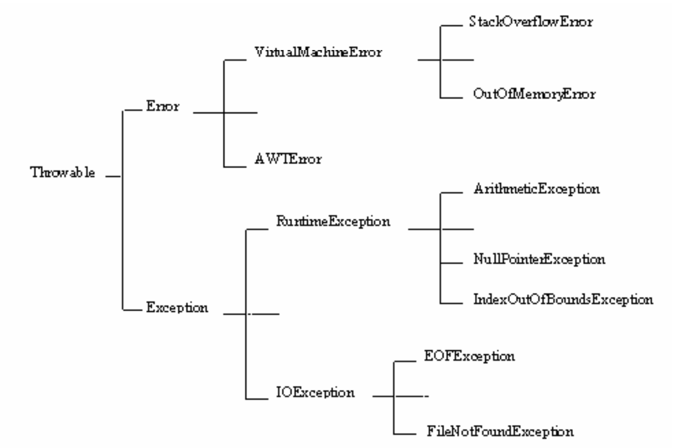
Exception Handler

- When an appropriate handler is found, the runtime system passes the exception to the handler.
- Example DodgeException.java

Exception Categories



Exception Categories



gauravgambhir.cse@piet.co.in

Checked Exception

- These are the exceptional condition that a well written application should anticipate and recover from.
- This is the category of exceptions for which the compiler checks to ensure that your code is prepared for them: prepare for unwelcome but expected guests.

gauravgambhir.cse@piet.co.in

Checked Exception

- For example suppose an application prompts a user for an input filename, the user supplies the name of a nonexistent file, and the constructor throws java.io.FileNotFound Exception.

- A well-written program will catch this exception and notify the user of the mistake.

Checked Exception

- Java compiler check programmer has stated what is to be done when they arise and because of this checking they are called checked exception.

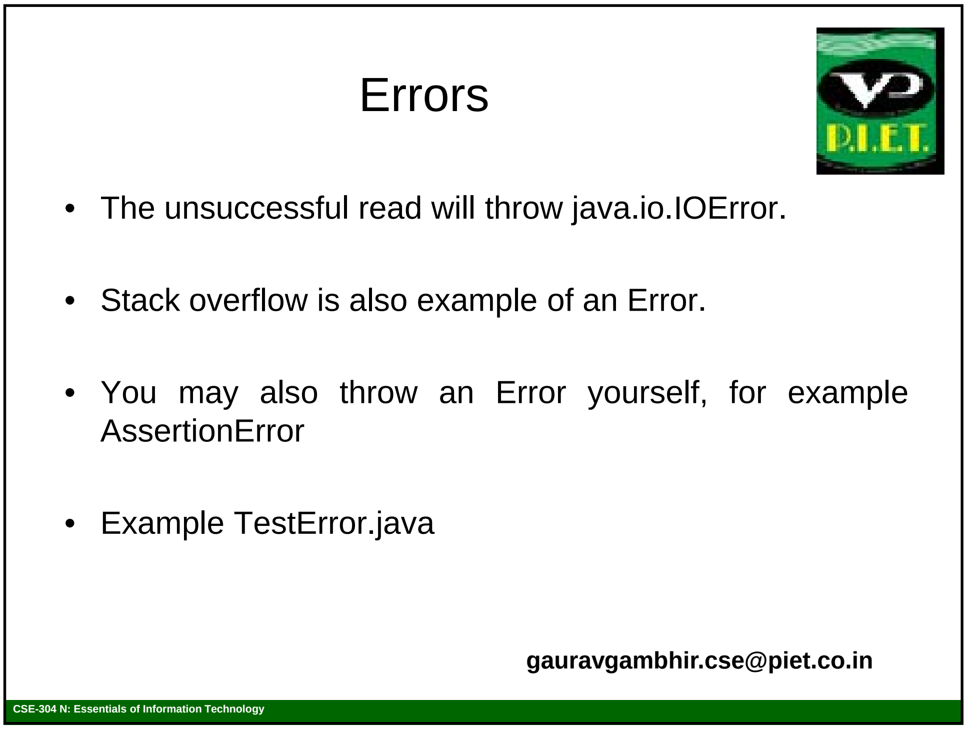
Example TestCheckException.java

Runtime Exceptions

- These are exceptional conditions that are internal to the application, and that the application usually cannot anticipate or recover from.
- These usually indicate programming bugs, such as divison by zero and invalid array indexing.
- An object of type RuntimeException may be thrown from any method without being specified as part of the method's public interface

Errors

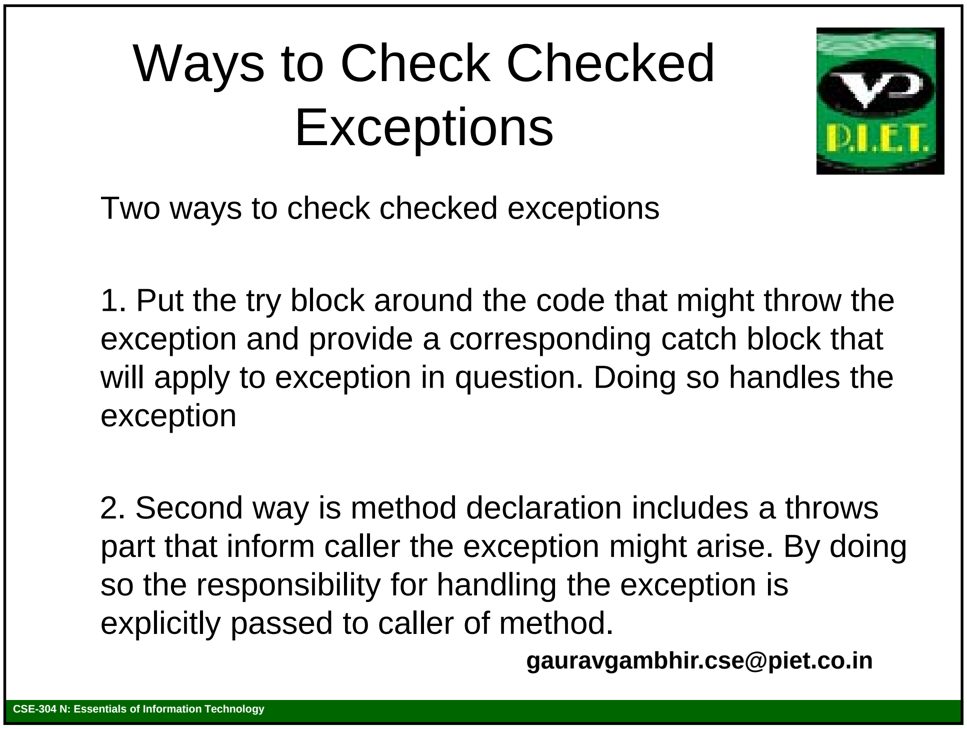
- These are exceptional conditions that are external to the application, and that the application usually cannot anticipate or recover from.
- For example, suppose that an application successfully opens a file for input, but is unable to read the file because of a hardware or system malfunction.



Errors

- The unsuccessful read will throw java.io.IOError.
- Stack overflow is also example of an Error.
- You may also throw an Error yourself, for example AssertionError
- Example TestError.java

gauravgambhir.cse@piet.co.in



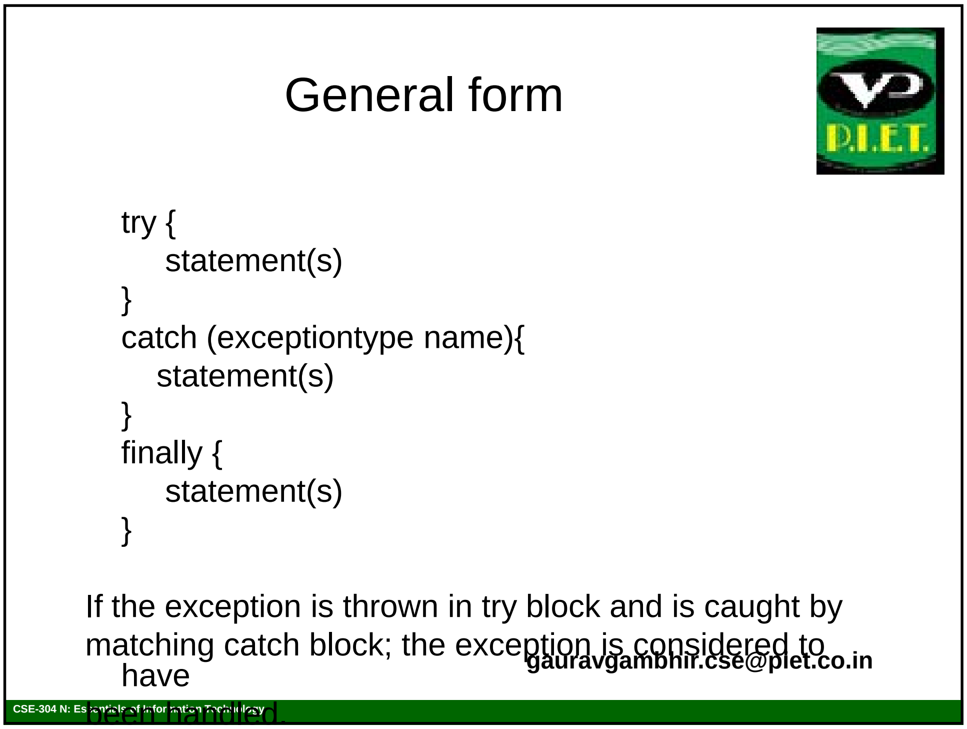
Ways to Check Checked Exceptions

Two ways to check checked exceptions

1. Put the try block around the code that might throw the exception and provide a corresponding catch block that will apply to exception in question. Doing so handles the exception

2. Second way is method declaration includes a throws part that inform caller the exception might arise. By doing so the responsibility for handling the exception is explicitly passed to caller of method.

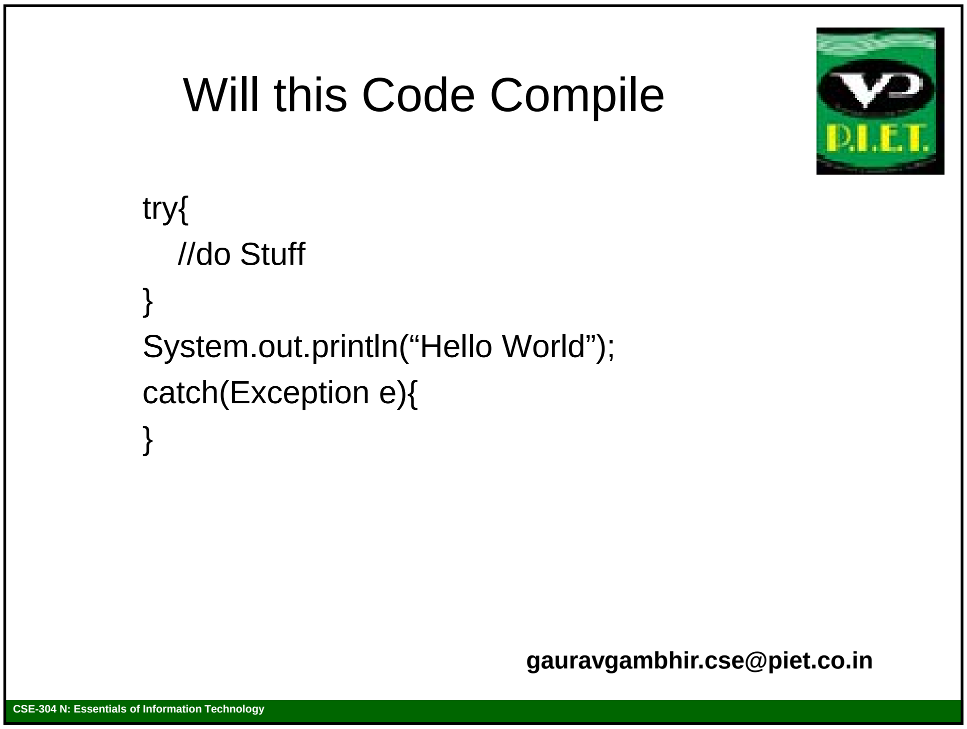
gauravgambhir.cse@piet.co.in



General form

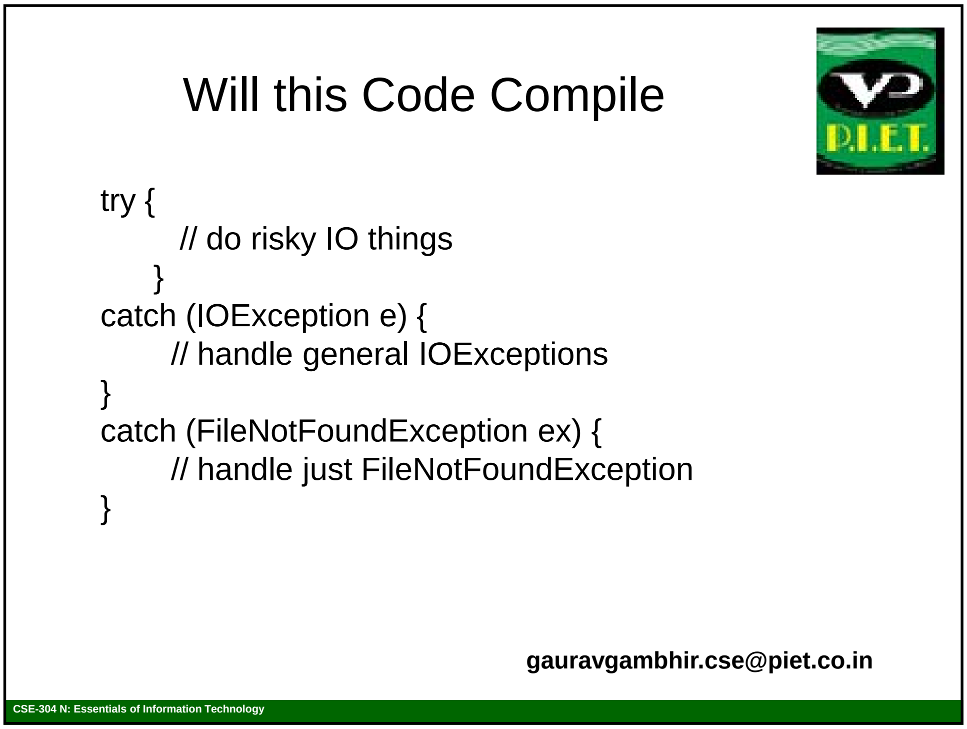
```
try {
     statement(s)
}
catch (exceptiontype name){
    statement(s)
}
finally {
     statement(s)
}
```

If the exception is thrown in try block and is caught by matching catch block; the exception is considered to have been handled.



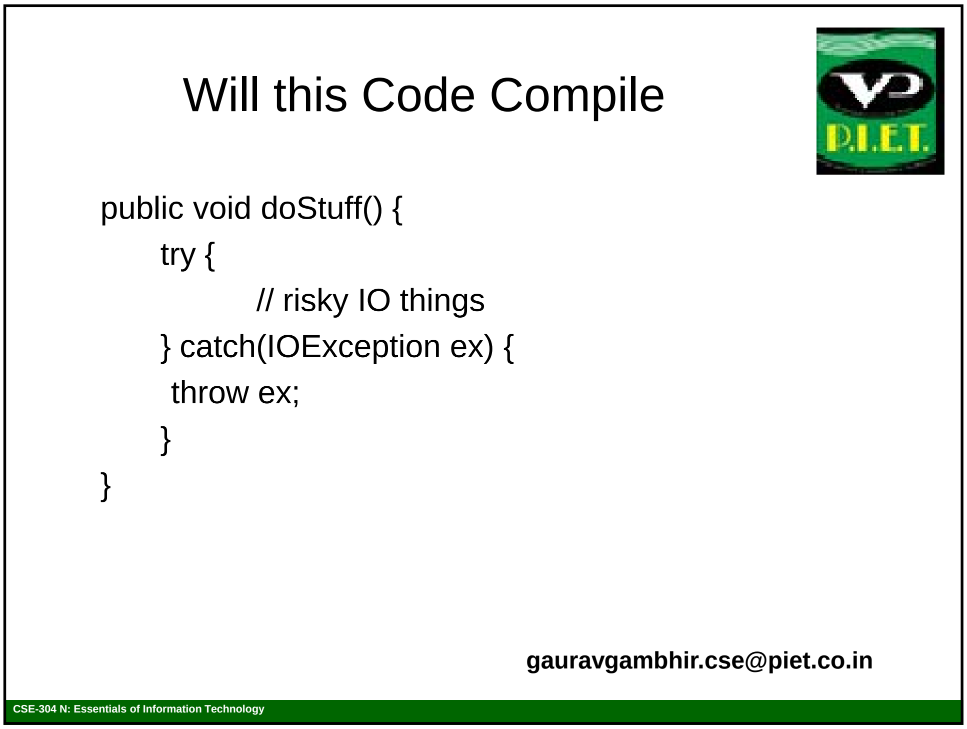
Will this Code Compile

```
try{
    //do Stuff
}
System.out.println("Hello World");
catch(Exception e){
}
```



Will this Code Compile

```
try {
       // do risky IO things
     }
catch (IOException e) {
      // handle general IOExceptions
}
catch (FileNotFoundException ex) {
      // handle just FileNotFoundException
}
```



Will this Code Compile

```
public void doStuff() {
     try {
              // risky IO things
     } catch(IOException ex) {
      throw ex;
     }
}
```

Exception Example

```
class Exc0 {
      public static void main(String args[]) {
                int d = 0;
                int a = 42 / d;
      }
}
```

When the Java run-time system detects the attempt to divide by zero, it constructs a new exception object and then throws this exception.

gauravgambhir.cse@piet.co.in

Exception Example

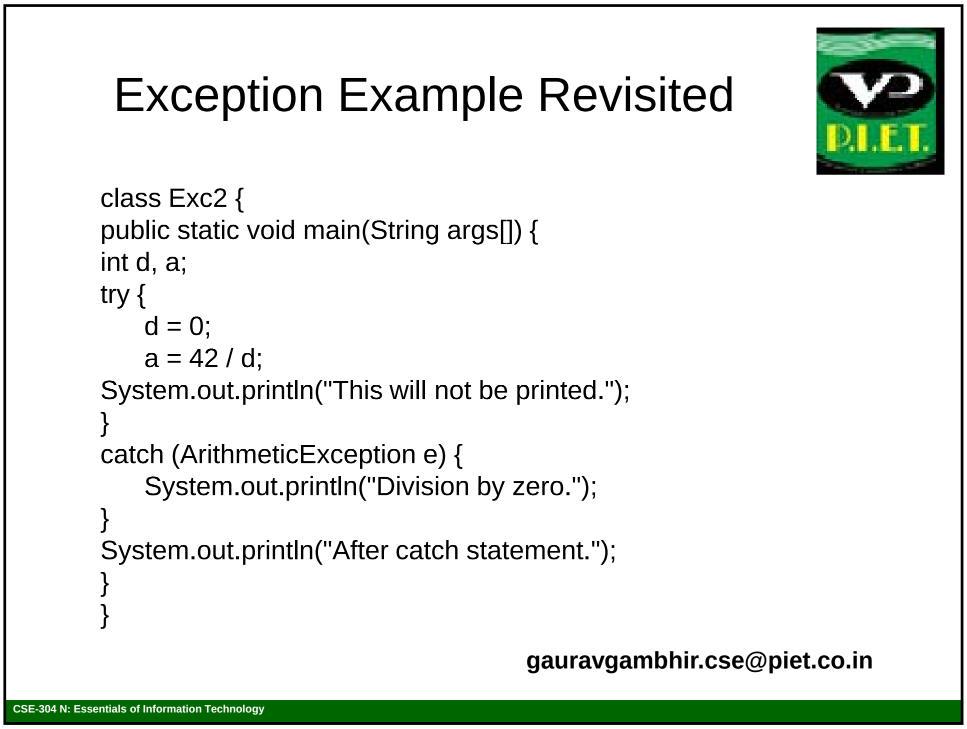
- Exception is caught by the default handler provided by the Java run-time system.
- The resulting stack trace from the default exception handler shows how the entire call stack is displayed:

  Exception in thread "main"

  java.lang.ArithmeticException:

  / by zero at Exc0.main(Exc0.java:4)

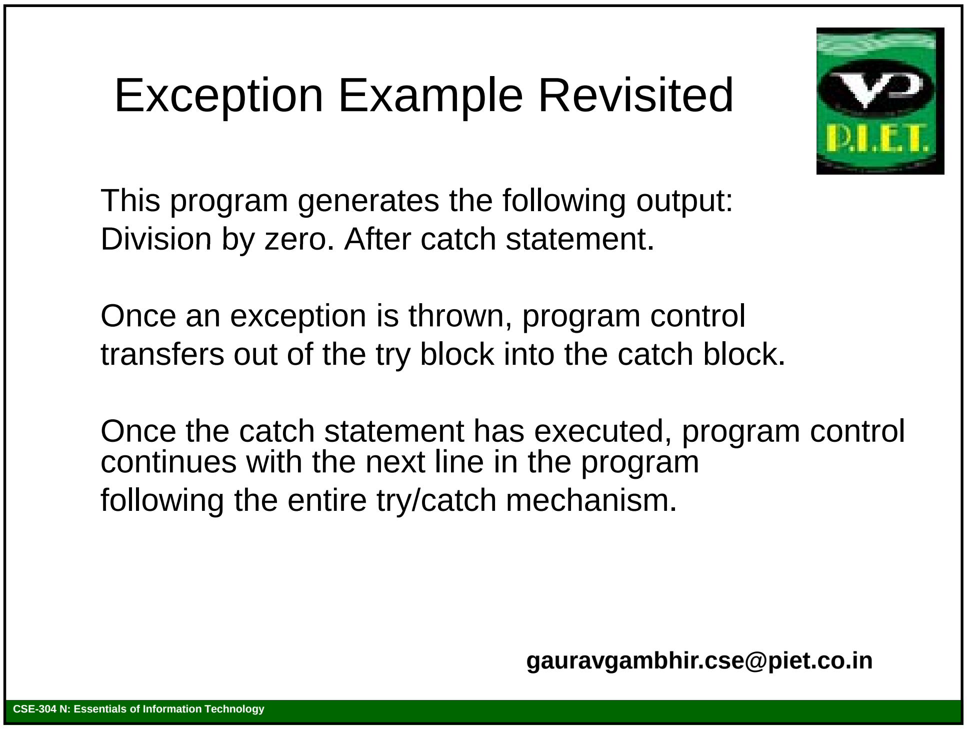
gauravgambhir.cse@piet.co.in



Exception Example Revisited

```
class Exc2 {
public static void main(String args[]) {
int d, a;
try {
    d = 0;
    a = 42 / d;
System.out.println("This will not be printed.");
}
catch (ArithmeticException e) {
    System.out.println("Division by zero.");
}
System.out.println("After catch statement.");
}
}
```

gauravgambhir.cse@piet.co.in



Exception Example Revisited

This program generates the following output:
Division by zero. After catch statement.

Once an exception is thrown, program control transfers out of the try block into the catch block.

Once the catch statement has executed, program control continues with the next line in the program following the entire try/catch mechanism.

gauravgambhir.cse@piet.co.in

Flow of Simple Exception Conditions

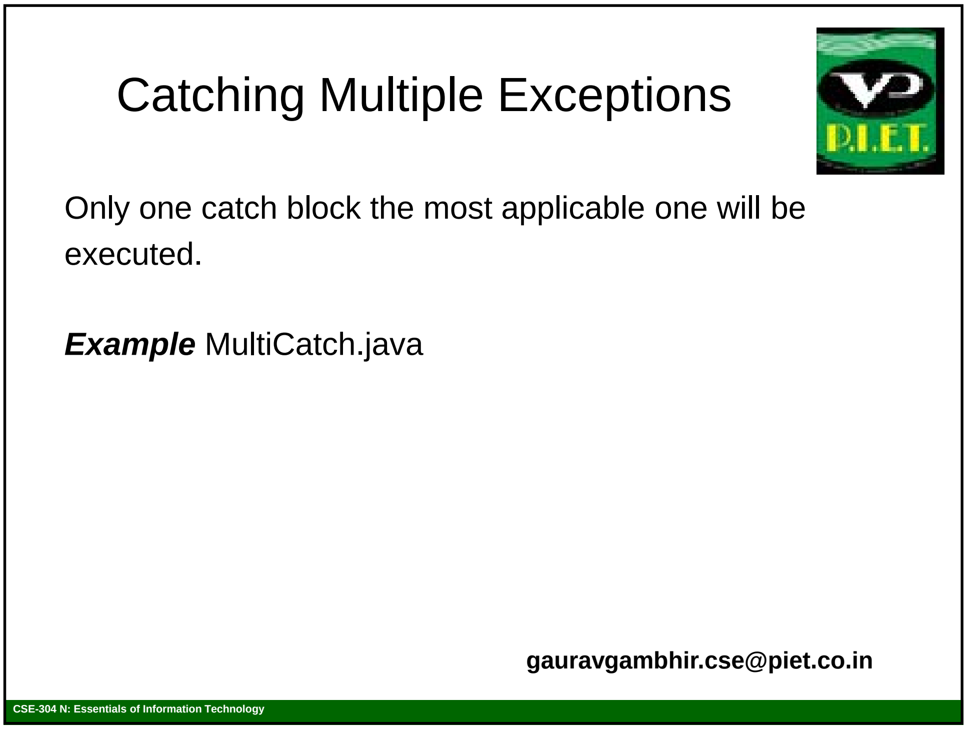
Exception	try()	catch()	Behaviour
No	N/A	N/A	Normal Flow
Yes	No	N/A	Method Termination
Yes	Yes	No	Compile Time Error
Yes matching	Yes	Yes	Terminate try {} block Execute body of Catch block Continue Normal flow after Catch block

Catching Multiple Exceptions

Catch block catch the exceptions of the class specified including any exceptions that are subclass of the one specified.

A more specific catch block must precede a more general one, otherwise compiler error.

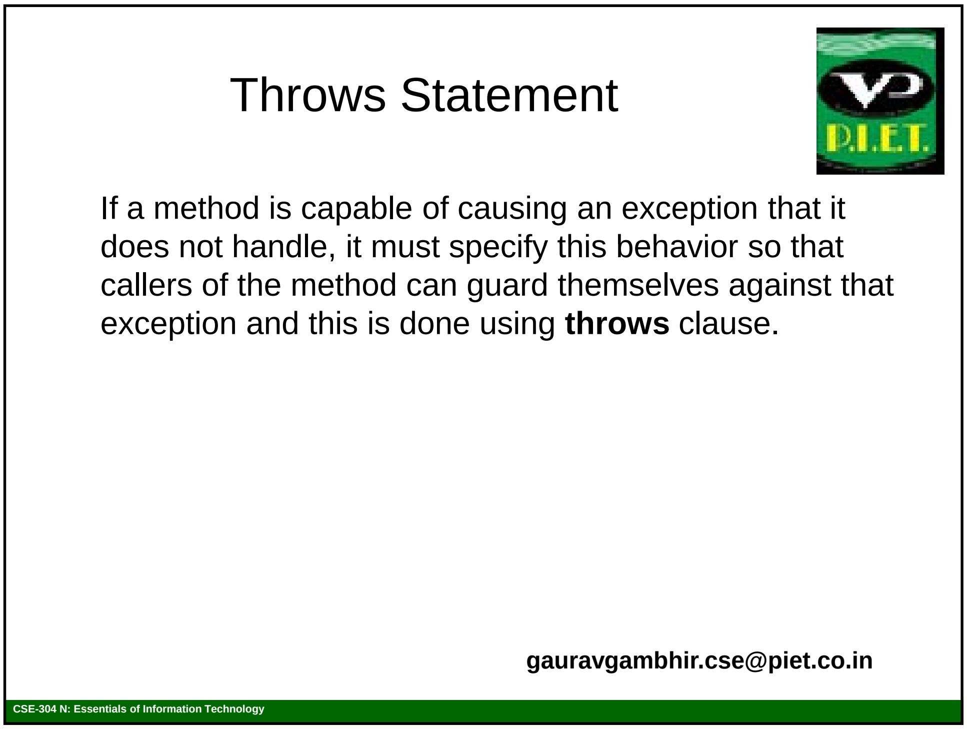
Example SuperSubCatch.java



Catching Multiple Exceptions

Only one catch block the most applicable one will be executed.

Example MultiCatch.java



Throws Statement

If a method is capable of causing an exception that it does not handle, it must specify this behavior so that callers of the method can guard themselves against that exception and this is done using **throws** clause.

What would happen while compiling and running the program

```
class ThrowsDemo {
    static void throwOne() {
        System.out.println("Inside throwOne.");
        throw new IllegalAccessException("demo");
    }
    public static void main(String args[]) {
        throwOne();
    }
}
```

Result

The program will not compile.

To make this example compile, you need to make two changes.

1. First, you need to declare that throwOne( ) throws IllegalAccessException.

2. Second, main( ) must define a try/catch statement that catches this exception.

Correct Version

```
class ThrowsDemo {
    static void throwOne() throws
    IllegalAccessException {
        System.out.println("Inside throwOne.");
        throw new IllegalAccessException("demo");
    }
    public static void main(String args[]) {
        try {
        throwOne();
        } catch (IllegalAccessException e) {
        System.out.println("Caught " + e);
        }
    }
}
```

gauravgambhir.cse@piet.co.in

Throw Statement

Throwing an exception in its most basic form is simple . Two steps involved in throwing an exception are

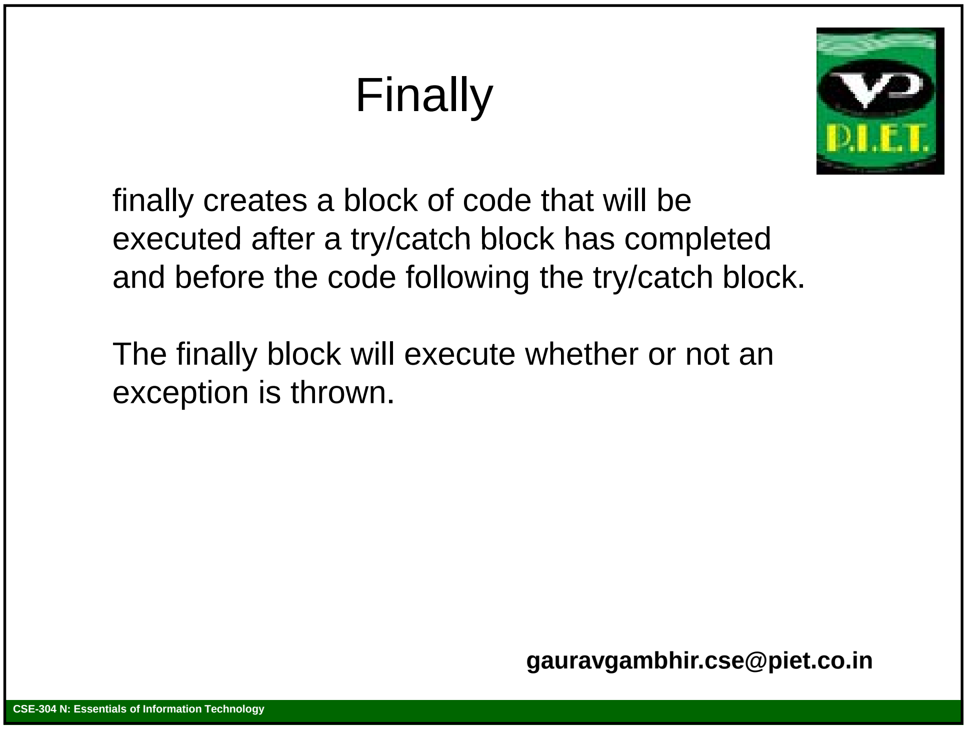
1.Create an instance of an object that is subclass of java.lang.Throwable.

2.Use throw keyword to actually throw the exception.

These two are combined into a single statement like this throw new IOException(“File not found”);

Example ThrowDemo.java

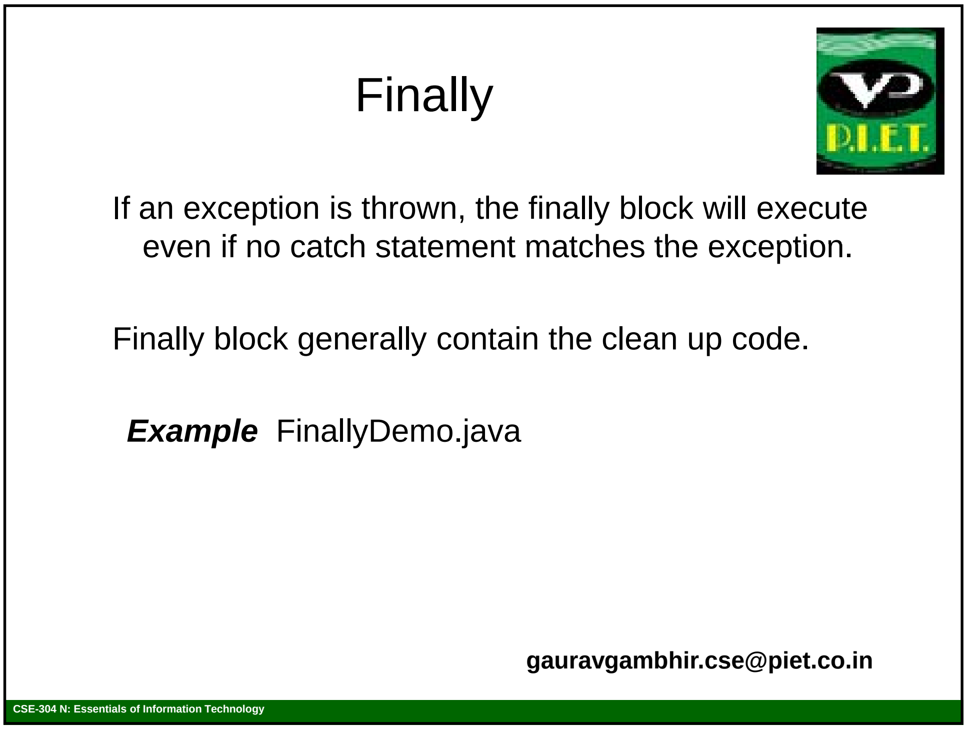
gauravgambhir.cse@piet.co.in



Finally

finally creates a block of code that will be executed after a try/catch block has completed and before the code following the try/catch block.

The finally block will execute whether or not an exception is thrown.

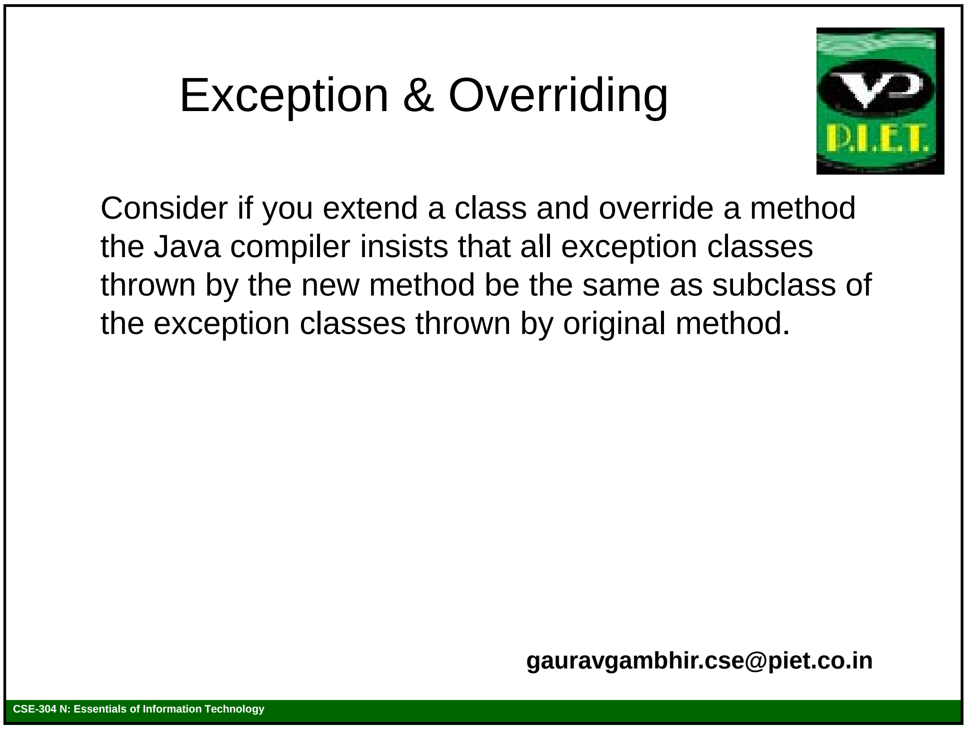


Finally

If an exception is thrown, the finally block will execute even if no catch statement matches the exception.

Finally block generally contain the clean up code.

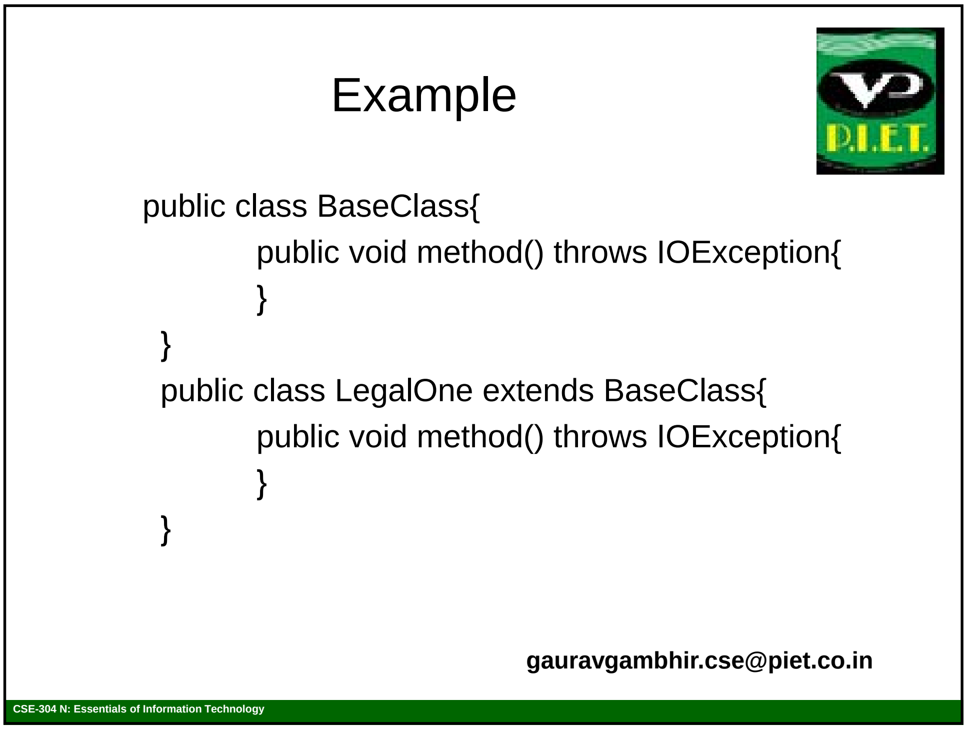
Example FinallyDemo.java



Exception & Overriding

Consider if you extend a class and override a method the Java compiler insists that all exception classes thrown by the new method be the same as subclass of the exception classes thrown by original method.

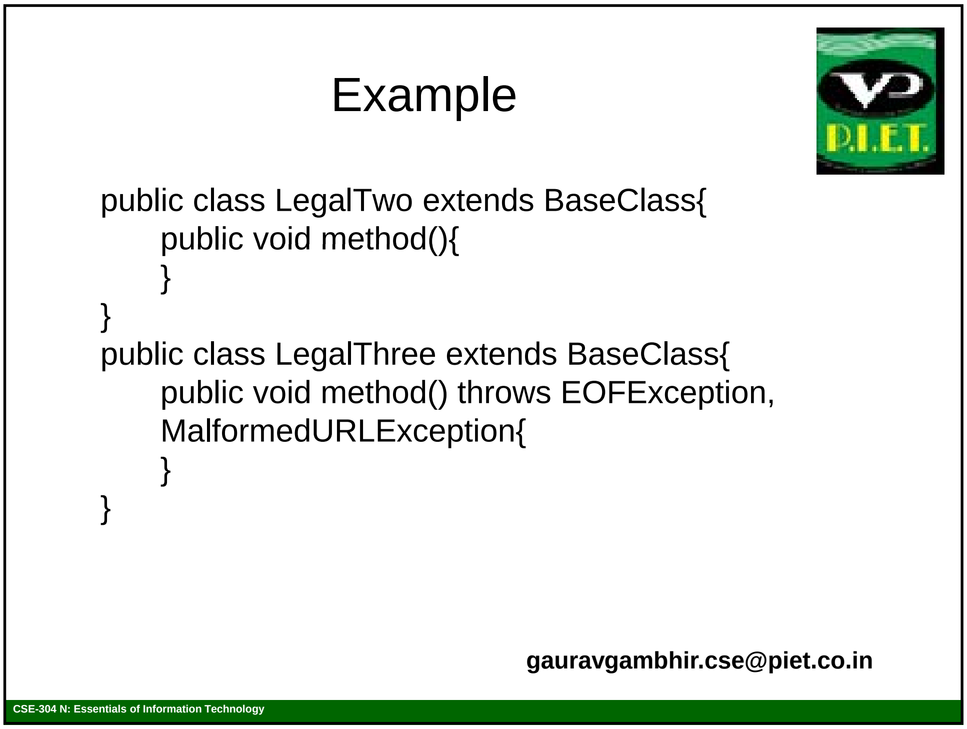
gauravgambhir.cse@piet.co.in



Example

```
public class BaseClass{
            public void method() throws IOException{
            }
  }
  public class LegalOne extends BaseClass{
            public void method() throws IOException{
            }
  }
```

gauravgambhir.cse@piet.co.in

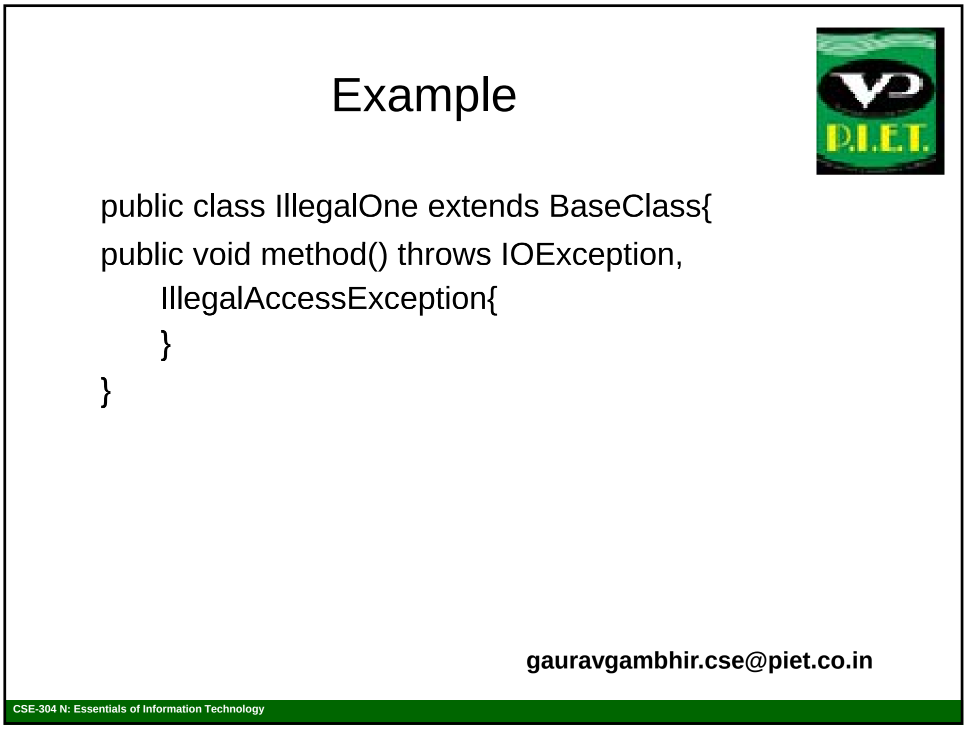


Example

```
public class LegalTwo extends BaseClass{
      public void method(){
      }
}
public class LegalThree extends BaseClass{
      public void method() throws EOFException,
      MalformedURLException{
      }
}
```

gauravgambhir.cse@piet.co.in

CSE-304 N: Essentials of Information Technology

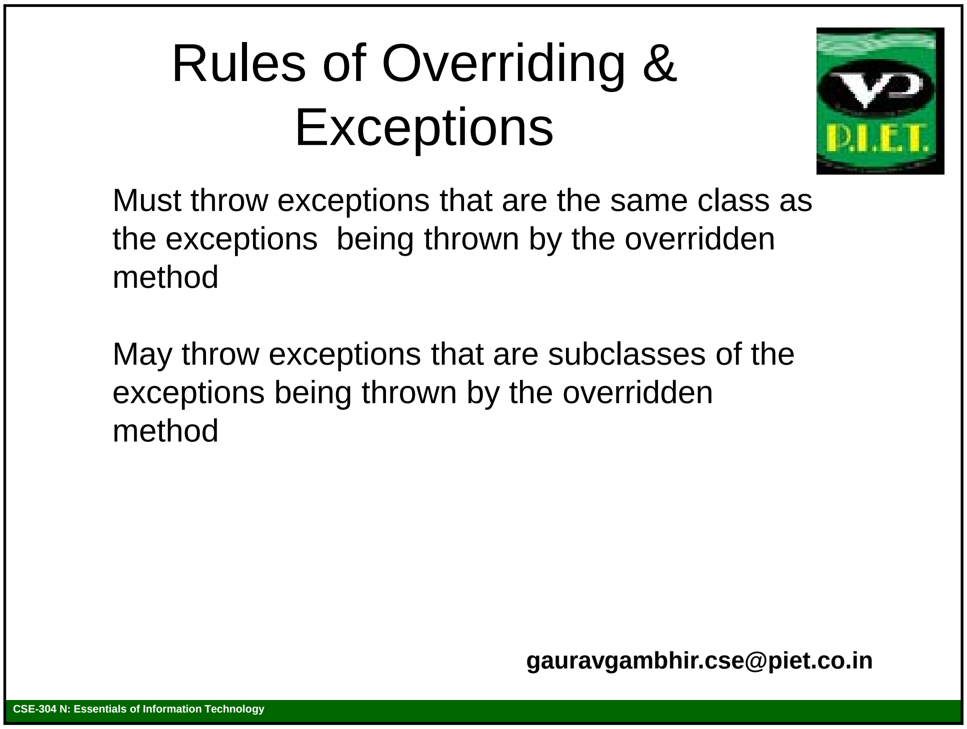


Example

```
public class IllegalOne extends BaseClass{
public void method() throws IOException,
      IllegalAccessException{
      }
}
```

gauravgambhir.cse@piet.co.in

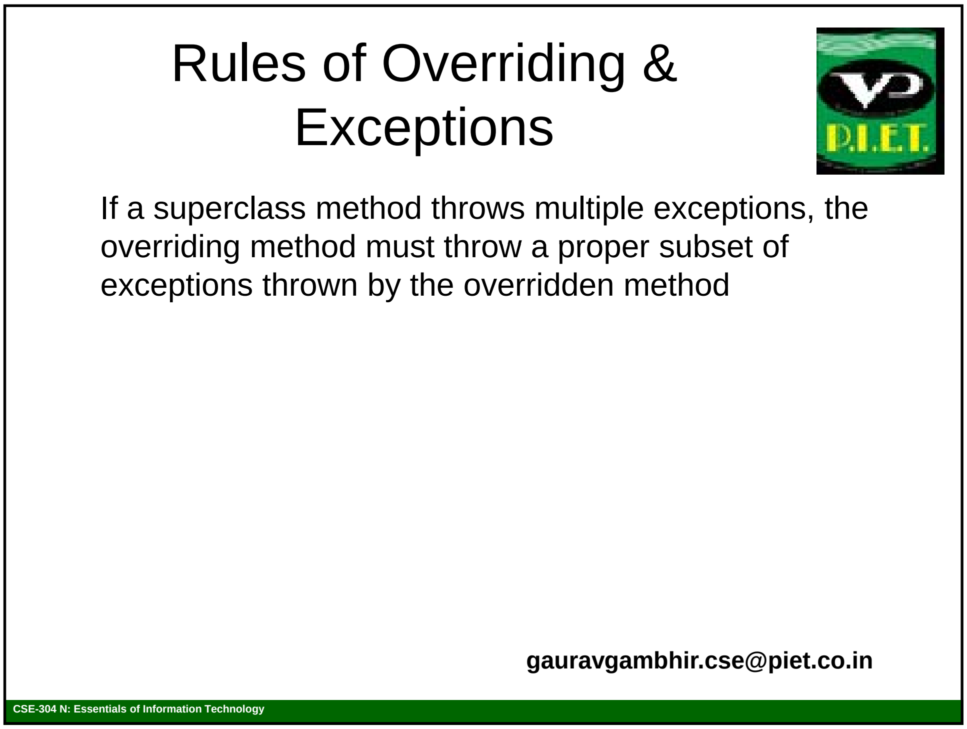
CSE-304 N: Essentials of Information Technology



Rules of Overriding & Exceptions

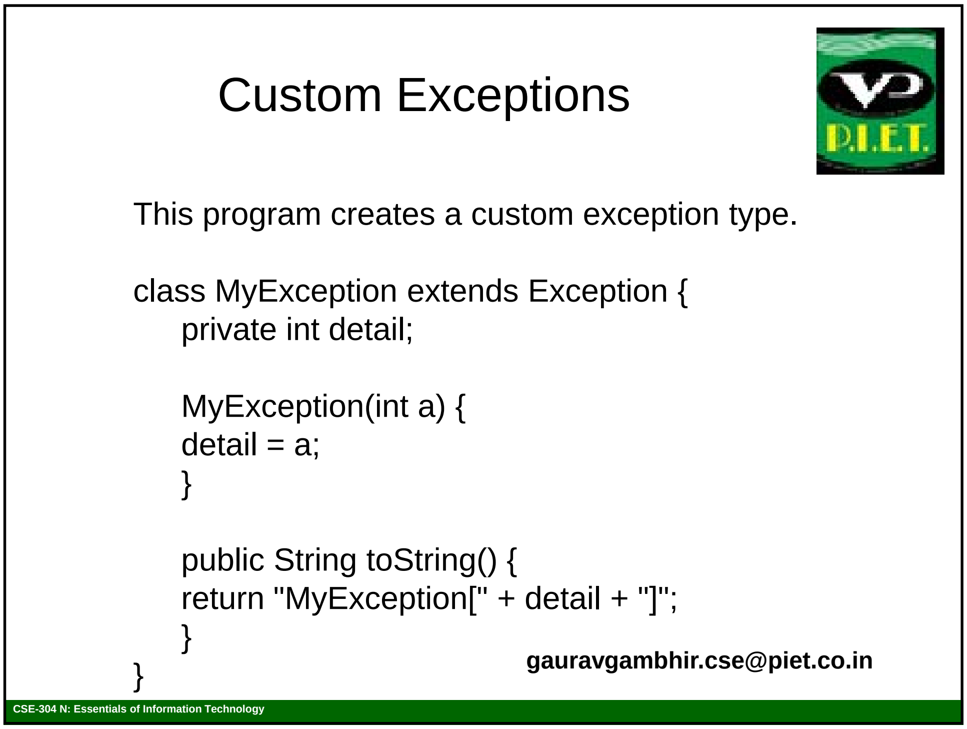
Must throw exceptions that are the same class as the exceptions being thrown by the overridden method

May throw exceptions that are subclasses of the exceptions being thrown by the overridden method



Rules of Overriding & Exceptions

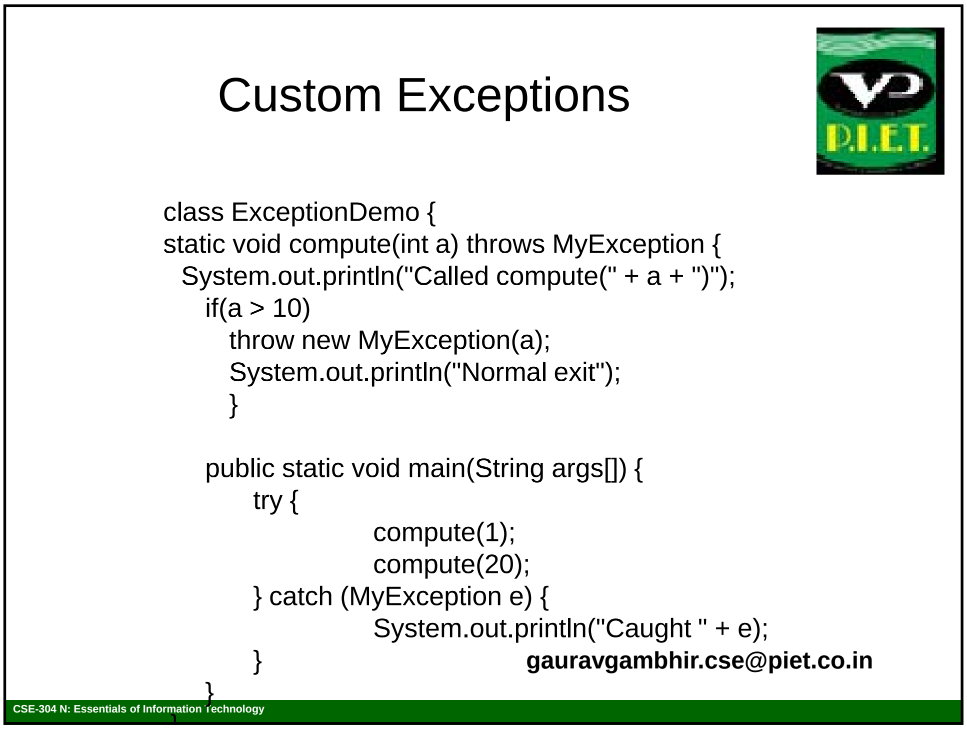
If a superclass method throws multiple exceptions, the overriding method must throw a proper subset of exceptions thrown by the overridden method



Custom Exceptions

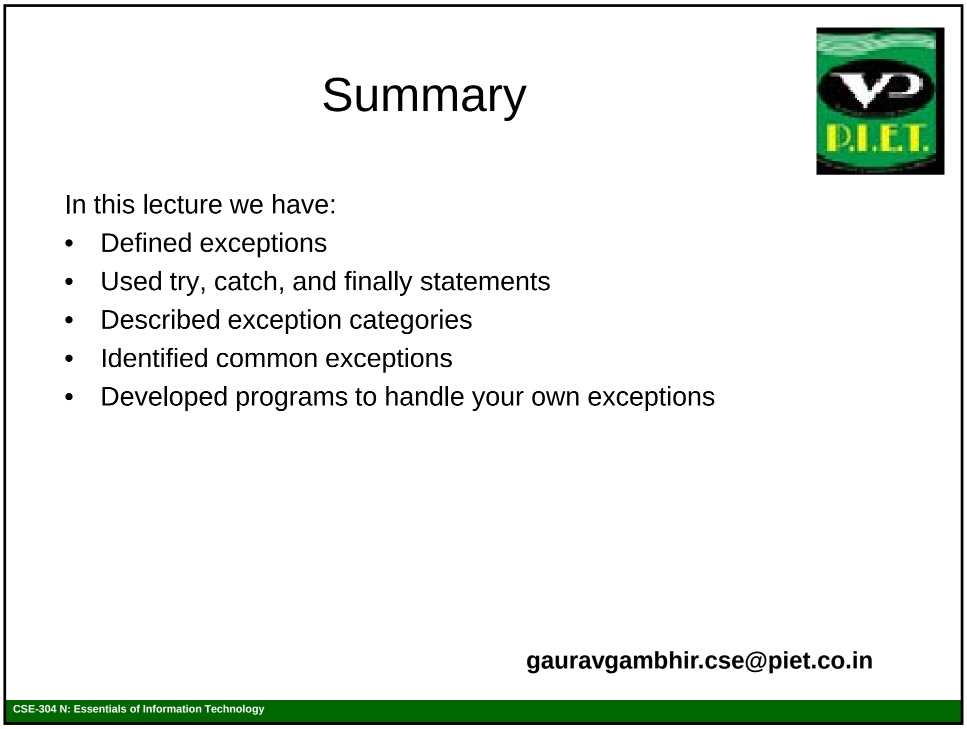
This program creates a custom exception type.

```
class MyException extends Exception {
    private int detail;

    MyException(int a) {
    detail = a;
    }

    public String toString() {
    return "MyException[" + detail + "]";
    }
}
```



Custom Exceptions

```
class ExceptionDemo {
static void compute(int a) throws MyException {
  System.out.println("Called compute(" + a + ")");
     if(a > 10)
        throw new MyException(a);
        System.out.println("Normal exit");
        }

     public static void main(String args[]) {
           try {
                          compute(1);
                          compute(20);
           } catch (MyException e) {
                          System.out.println("Caught " + e);
           }
     }
  }
```

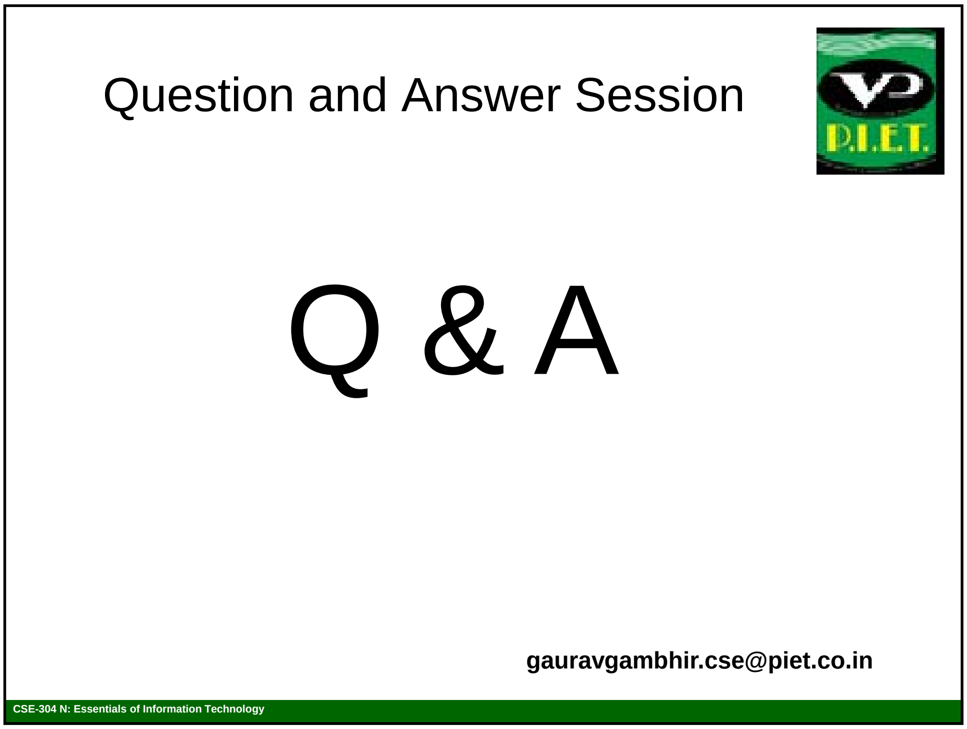


Summary

In this lecture we have:

- Defined exceptions
- Used try, catch, and finally statements
- Described exception categories
- Identified common exceptions
- Developed programs to handle your own exceptions

gauravgambhir.cse@piet.co.in



Question and Answer Session

Q & A

gauravgambhir.cse@piet.co.in