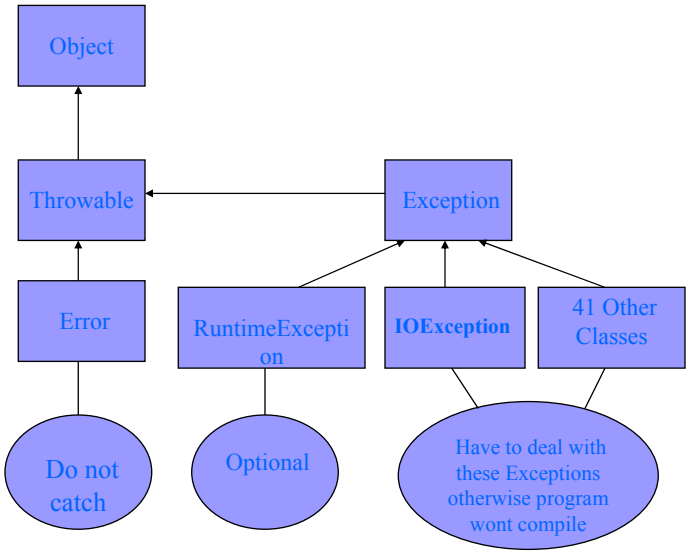
Exceptions

- **Exceptions signal some unusual event in your program (can be errors also) that deserves special attention**
- **Dealing with exceptions involves quite a lot of processing overhead**
- **Not all errors in your programs need to be signaled by exceptions**
- **They provide a way to deal with things gracefully**

Exceptions

- **An exception in Java is an object that is created when an abnormal situation arises in your program**
- **This object has members that stores information about the nature of the problem**
- **An Exception is always an object of some subclass of the standard class Throwable**
- **Java provides a very well defined hierarchy of Exceptions to deal with situations which are unusual.**
- **All Standard exceptions are covered by two direct subclasses of the class Throwable**
  - Class Error
  - Class Exception

Exceptions



Error Exceptions

- **Error Exceptions and its subclasses represent conditions that you are not expected to do anything about.**
- **Direct subclasses are**
  - ThreadDeath
  - LinkageError
  - VirtualMachineError
- **ThreadDeath exception is thrown when an executing thread is deliberately stopped and when it is not caught the thread ends(not your program)**
- **Linkage error arises when you try to create an object of a non-existent class etc**
- **VirtualMachineError is thrown when failure of JVM occurs**

RuntimeException

- **The compiler allows you to ignore the Runtime exceptions**
- **They usually arise because of serious error in your code**
- **Most commonly faced runtime exceptions are**
  - ArithmeticException
  - IndexOutOfBoundException -- ArrayIndexOutOfBoundException
  - NegativeArraySizeException
  - NullPointerException
  - SecurityException
  - IllegalArgumentException --NumberFormatException
  - UnsupportedOperationException

Dealing With Exceptions

- **For all subclasses of Exception Class(except RuntimeException) you must include code to deal with them**
- **If your program has the potential to generate an exception of such a type,you have got two choices**
  - Handle the exception within the method
  - Register that your method may throw such an exception (You are passing the exception on)
- **If you do neither your code wont compile**

Dealing With Exceptions

- **If you have a method which can throw an exception that is neither a subclass of Error nor RuntimeException, and you don't wont to deal with it then you have to at least declare that this exception can be thrown**
- **You can ignore the exception by enabling the method containing the exception throwing code to pass it on.**
- **Add a throws keyword after the parameter list of the method and list the classes for the exceptions that might be thrown separated by comma**
  - **void myMethod ( ) throws EOFException, FileNotFoundException**

    {

    }
- **If another method calls this method now, it must deal with that exception in either of two ways**

Dealing With Exceptions

- If you want to handle the exception where they occur, then you use three kinds of code blocks
- Try block — It encloses the code that may give rise to one or more exceptions. Code that can throw an exception that you want to catch must be in a try block
- Catch block — it contains the code that is handling the exception of a particular type that may be thrown in a try block
- Finally block — contains code that is always executed before the method ends regardless of whether any exceptions are thrown in the try block

Dealing With Exceptions

```
try {
        //code that can throw one or more exceptions
        }
catch(IOException e)
{
//code to handle the exception
}
finally
{
// code to be executed last
}
```

Example 1

```
public class TestTryCatch {
  public static void main(String[] args) {
    int i = 1;
    int j = 0;

    try {
      System.out.println("Try block entered " + "i = "+ i + " j =
"+j);
      System.out.println(i/j);    // Divide by 0 - exception thrown
      System.out.println("Ending try block");

    } catch(ArithmeticException e) {        // Catch the exception
      System.out.println("Arithmetic exception caught");
    }

    System.out.println("After try block");
    return;
  }
}
```

Example 2

```
public class TestLoopTryCatch {
  public static void main(String[] args) {
    int i = 12;

    for(int j=3 ;j>=-1 ; j--)
      try {
        System.out.println("Try block entered " + "i = "+ i + " j =
"+j);
        System.out.println(i/j);     // Divide by 0 - exception thrown
       System.out.println("Ending try block");

      } catch(ArithmeticException e) {     // Catch the exception
        System.out.println("Arithmetic exception caught");
      }

    System.out.println("After try block");
    return;
  }
}
```

Dealing With Exceptions

- The catch block must immediately follow the try block which contains the code that may throw that particular exception
- The parameter for the catch block must be of type Throwable or one of the subclasses of it
- If the class you specify as the parameter has subclasses the catch block will be expected to process exceptions of that class plus all the subclasses of that class
- When an exception is thrown the control is transferred immediately to the first statement in the appropriate catch block
- After the catch block has executed, control goes to the statement after the catch block

Dealing With Exceptions

- If a try block can throw different kind of exceptions, you can put several catch blocks after the try block to handle them
- Only one catch block will execute now, depending upon the type of exception thrown
- When you need to catch exceptions of several different types for a try block, the order of the catch blocks is important
- When an exception is thrown, it will be caught by the first catch block that has a parameter type that is the same as that of the exception or a type that is a super class of the type of the exception
- Etc. if you specify the catch block parameter as Exception. This will catch all the exceptions that are its subclasses

Dealing With Exceptions

- Placing a base class exception catch block first causes the starvation problem
- Thus the catch blocks must be in sequence with the most derived type first and the most basic type appearing last otherwise your code would not compile

Dealing With Exceptions

- The finally block must be located immediately following any catch blocks for the try blocks.
- If there are no catch blocks then it must immediately follow the try block
- The finally block provides the means to clean up at the end of executing a try block. You use finally block when you need to be sure that some particular code is run before the method returns, no matter an exception occurs in the try block or not.
- So you can enclose code in try block that would not produce an exception and in this case you don't need a catch block.

Dealing With Exceptions

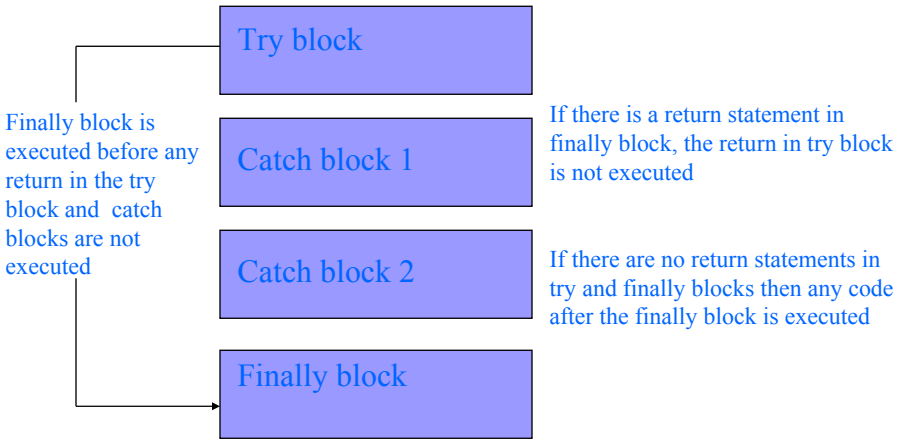
- **You cannot have a try block by itself. Each try block must always be followed by at least one block that is either a catch block or a finally block**
- **You cannot include other code between a try block and its related catch blocks/finally blocks or between the catch blocks and the finally blocks**
- **You can have other code after the finally block**
- **A finally block is optional if there is catch block for the try block**

Execution Sequences

- **For a case when you have a try block, a number of catch blocks and a finally block in a method, there can be a number of execution sequences depending on whether an exception is thrown or not and whether there are any return statements in the try and finally blocks.**

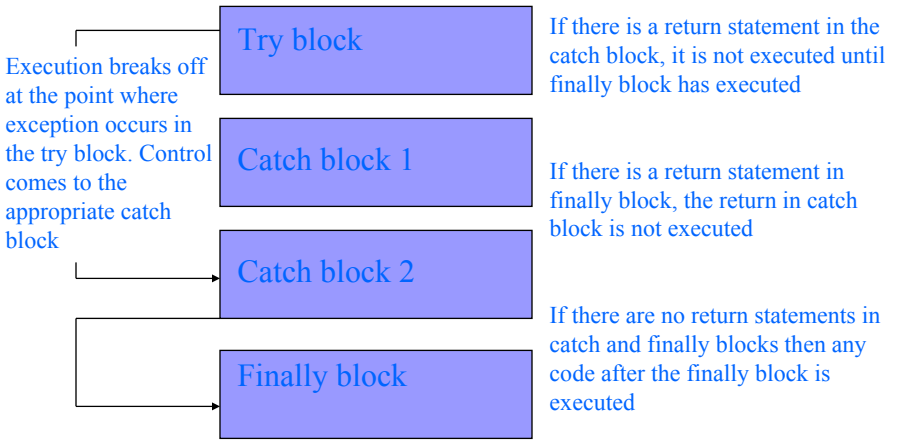
Execution Sequences

- **If no exception occurs in the try block**



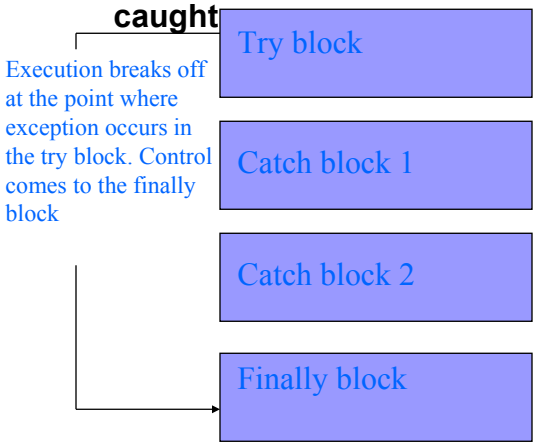
Execution Sequences

- **If an exception occurs in the try block and is caught**



Execution Sequences

- **If an exception occurs in the try block and is not caught**



Rethrowing Exceptions

- **Even after catching an exception inside a catch block, you can pass on that exception to the calling program**
- **You are rethrowing that exception**
- **From within the catch block, use throw statement**
  - catch (IOException e)

    {

    throw e;

    }
- **The keyword throw is followed by the exception object to be thrown**

Exception Objects

- **The exception object that is passed to the catch block can provide information about the nature of the problem that originated that exception**
- **Class throwable is the class from which all java exception classes are derived, thus all exception objects inherit members from the super class Throwable**
  - A message that is initialized by the constructor
  - A record of the execution stack

Exception Objects

- **Members methods of class Throwable are**
  - getMessage( )—returns the content of the message describing the current exception, gives fully qualified name of the exception class and a brief description of the exception
  - printStackTrace( )—Gives the message plus the stack trace
  - fillInStacktrace( )—to update the stack trace to the point where this method is called. Normally used at the point of rethrow of an exception object.
- **The execution stack keeps track of all the methods that are in execution at any given instant. The record of the execution stack that is stored in the exception object consist of line number in the source code where the exception originated followed by a trace of the method calls that immediately preceded the method called plus the line number in the source file where each method call occurred.**

Defining your own Exceptions

- **There are two basic reasons for defining your own exception classes**
  - You want to add information when a standard exception occurs and you can do this by rethrowing an object of your own exception class
  - You want to deal with some specific error in your code
- **Each exception class must always have Throwable as the super class**
- **You can derive from any of the standard exceptions, deriving from Exception class is preferred. Why?**

Defining your own Exceptions

- **By convention, your exception class should include a default constructor and a constructor that accepts a String object as argument**
- **The message stored in the exception object will be initialized with your class name and if a string is passed to the second constructor it will be appended to the name of the class to form the message.**
- **MyException e = new MyException( );**

  throw e;
- **throw new MyException(" Weird Problem ");**