**GENERAL TRY – CATCH – FINALLY**

*inherit those methods provided by Throwable*

|  |  |
| --- | --- |
| **package** com.company;  **public class** Main {   **public static void** main(String[] args) {  **int** x = 0;  **int** y = 10;   **try** {  **int** z = y / 0;  System.***out***.println(**"the answer will be : "** + z);  }**catch** (Exception ex){  System.***out***.println(**"message : "** + ex.getMessage());  System.***out***.println(**"description : "** + ex);  System.***out***.println(**"Cause"** + ex.getCause());  System.***out***.println();  } **finally** {  System.***out***.println(**"the finally block is executing"**);  }  } } | message : / by zero  description : java.lang.ArithmeticException: / by zero  Causenull  the finally block is executing |
| **package** com.company;  **public class** Main {   **public static void** main(String[] args) {  **int** x = 0;  **int** y = 10;   System.***out***.println(**"the result will be : "** + *multiply*(y, x));  }   **public static int** multiply(**int** x, **int** y){  **try** {  **int** n = y / x;  System.***out***.println(**"from multiply - try"**);  **return** n;  } **catch** (Exception e){  System.***out***.println(**"from multiply catch"**);  System.***out***.println(**"Message : "** + e.getMessage());  System.***out***.println();  } **finally** {  System.***out***.println(**"from multiply - finally"**);  **return** 1;  }  }   **public class** Duck{  **public void** giveMessage(){  System.***out***.println(**"got a motherfucking exception"**);  }  } } | from multiply - try  from multiply - finally  the result will be : 1 |
|  |  |

**Multiple catch Clauses**

|  |  |
| --- | --- |
| **package** com.company;  **public class** Main {   **public static void** main(String[] args) {  **int** x = 0;  **int** y = 10;   **try** {  **int** z = y / 0;  System.***out***.println(**"the answer will be : "** + z);  }**catch** (Exception ex){  System.***out***.println(**"message : "** + ex.getMessage());  System.***out***.println(**"description : "** + ex);  System.***out***.println(**"Cause"** + ex.getCause());  **try** {  **int** z1 = x \* y;  System.***out***.println(**"result is : "** + z1);  } **catch** (Exception ex1){  System.***out***.println(**"exception occured : "** + ex1);  } **finally** {  System.***out***.println(**"finally from inner try - catch block"**);  }  System.***out***.println();  } **finally** {  System.***out***.println(**"the finally block is executing"**);  }  } } | **message : / by zero**  **description : java.lang.ArithmeticException: / by zero**  **Causenull**  **result is : 0**  **finally from inner try - catch block**  **the finally block is executing** |

**THROW**

***it is possible for your program to throw an exception explicitly, using the throw statement.***

|  |  |  |
| --- | --- | --- |
| **package** com.company;  **public class** Main {   **public static void** main(String[] args) {  **try** {  ThrowDemo.*demoproc*();  } **catch**(NullPointerException e) {  System.***out***.println(**"Recaught: "** + e);  }  } } | **package** com.company;  **public class** ThrowDemo {  **public static void** demoproc() {  **try** {  **throw new** NullPointerException(**"demo"**);  } **catch** (NullPointerException e) {  System.***out***.println(**"Caught inside demoproc."**);  **throw** e; *// rethrow the exception* }  } } | **Caught inside demoproc.**  **Recaught: java.lang.NullPointerException: demo** |

**THROWS**

**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. You do this by including a throws clause in the method’s declaration. A throws clause lists the types of exceptions that a method might throw. This is necessary for all exceptions, except those of type Error or RuntimeException, or any of their subclasses. All other exceptions that a method can throw must be declared in the throws clause. If they are not, a compile-time error will result.

This is the general form of a method declaration that includes a throws clause:

type method-name(parameter-list) throws exception-list

{

// body of method

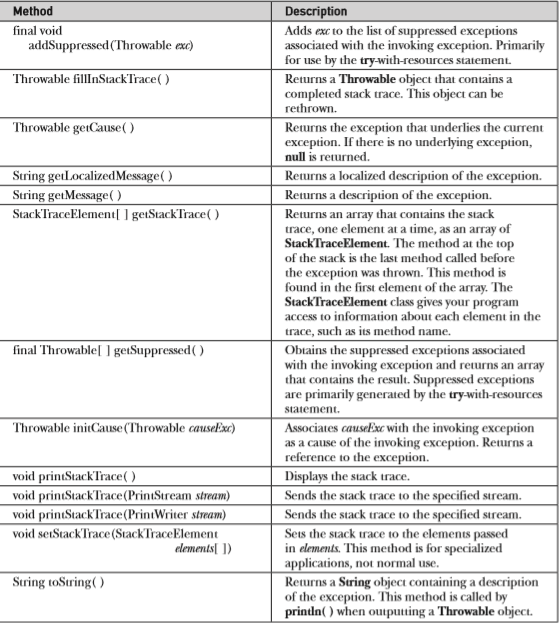
}

Here, exception-list is a comma-separated list of the exceptions that a method can throw.

Following is an example of an incorrect program that tries to throw an exception that it does not catch. Because the program does not specify a throws clause to declare this fact, the program will not compile.

|  |  |
| --- | --- |
| package com.company;  public class Main {  static void throwOne() {  System.*out*.println("Inside throwOne.");  throw new IllegalAccessException("demo");  }   public static void main(String args[]) {  *throwOne*();  } } | **ERROR** |
| **CORRECT VERSION** | |
| **package** com.company;  **public class** Main {  **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);   }   } } | **Inside throwOne.**  **Caught java.lang.IllegalAccessException: demo** |

**The Methods Defined by Throwable**

****

**USE DEFINED THRADES**

|  |  |  |
| --- | --- | --- |
| **package** com.company;  **public class** Main {  **public static void** main(String args[]) {  **try** {  ThrowDemo.*computevals*(1);  ThrowDemo.*computevals*(15);  } **catch** (Exception ex){  System.***out***.println(**"Caught in : "** + ex.getMessage());  System.***out***.println(**"Cause is : "** + ex.getCause());  } **finally** {  System.***out***.println(**"finally block is executing from Main"**);  }  } } | **package** com.company;  **public class** ThrowDemo {  **static void** computevals(**int** x) **throws** UserDefinedException{  System.***out***.println(**"need to compute "** + x);   **if** (x > 10){  **throw new** UserDefinedException(x);  } **else** {  System.***out***.println(**"Normal exit of "** + x + **" from ThroeDemo"**);  }  } } | **package** com.company;  **public class** UserDefinedException **extends** Exception {  **private int details**;   **public** UserDefinedException(**int** a){  **details** = a;  }   @Override  **public** String toString() {  **return** (**"My exception "** + **details**);  } } |
| need to compute 1  Normal exit of 1 from ThroeDemo  need to compute 15  Caught in : null  Cause is : null  finally block is executing from Main | | |

**Chained Exceptions**

Beginning with JDK 1.4, a feature was incorporated into the exception subsystem: chained exceptions. ***The chained exception feature allows you to associate another exception with an exception. This second exception describes the cause of the first exception***. For example, imagine a situation in which a method throws an ArithmeticException because of an attempt to divide by zero. However, the actual cause of the problem was that an I/O error occurred, which caused the divisor to be set improperly. Although the method must certainly throw an ArithmeticException, since that is the error that occurred, you might also want to let the calling code know that the underlying cause was an I/O error. Chained exceptions let you handle this, and any other situation in which layers of exceptions exist.

**To allow chained exceptions, two constructors and two methods were added to Throwable. The constructors are shown here:**

* Throwable(Throwable causeExc) -- causeExc is the exception that causes the current exception.
* Throwable(String msg, Throwable causeExc) -- to specify a description at the same time that you specify a cause exception.

|  |  |
| --- | --- |
| **package** com.company;  **public class** Main {  **public static void** main(String args[]) {  **try** {  ThrowDemo.*computevals*();  } **catch** (Exception e){  System.***out***.println(**"Cause : "** + e.getCause());  System.***out***.println(**"original cause : "** + e);  }  } } | **package** com.company;  **public class** ThrowDemo {  **static void** computevals(){  NullPointerException np =**new** NullPointerException(**"top"**);  np.initCause(**new** Exception(**"fucking cause"**));   **throw** np;  } } |
| **Cause : java.lang.Exception: fucking cause**  **original cause : java.lang.NullPointerException: top** | |

In this example, the top-level exception is NullPointerException. To it is added a cause exception, ArithmeticException. When the exception is thrown out of demoproc( ), it is caught by main( ). There, the top-level exception is displayed, followed by the underlying exception, which is obtained by calling getCause( ).

Chained exceptions can be carried on to whatever depth is necessary. Thus, the cause exception can, itself, have a cause. Be aware that overly long chains of exceptions may indicate poor design.