

|  |  |  |  |
| --- | --- | --- | --- |
| Type | How to Recognize | Okay for Programmer to catch | Is programmer required to handle |
| Runtime Exception | Subclass of Runtime Exception | Yes | No |
| Checked Exception | Subclass of Exception but not Runtime Excep. | Yes | Yes |
| Error | Subclass of Throwable and Error | No | No |

Try / catch / finally block rules:

1. Try needs to always have braces.
2. A try expects catch or finally block
3. A try can have multiple catch block
4. **1 try can only have 1 finally block**
5. A finally block can have a try, catch, finally block
6. **A catch block always must come after a try, not finally**
7. If a variable is already caught in the exception i.e Exception e, and you have a nested try and catch again, you cannot redeclare the same exception in the nested blocks catch parameter.
8. **When System.exit is called in the try block, the catch and finally block do not execute**
9. At most one catch block can run so if the superclass is caught before a subclass, then the compiler throws exception with unreachable code.

**This is not allowed:**

**try** {  
  
}**catch**(Exception e){  
  
}*catch(RuntimeException e){  
  
}catch(StringIndexOutOfBoundsException e){  
  
}*

**This is OK:**

**try**{  
  
}**catch**(StringIndexOutOfBoundsException ex){  
  
}**catch**( RuntimeException ex){  
   
}**catch**(Exception e){  
  
}

General rules on handling Exceptions and the compiler:

1. **You have to catch any Checked Exception if the enclosing method block has thrown the checked Exception, this also includes the Exception i.e throw new Exception() , requires that Exception be caught.**
2. **If an enclosing block is surrounded by a try and catch and the caught exception is a checked exception, then the block must throw a checked exception. The only exception is Exception.**

**public void** checkedExceptionConstraint(){ ………..1  
 **try**{ ……….2  
 *//throw new IOException(); ………3* **throw new** Exception(); ……….4  
 }**catch**(IOException e){ ………..5  
 }**catch**(Exception e){ ………….6  
 }  
 }

**If we comment line 4, then line 5 will throw a compile error because the enclosing block needs to throw a checked Exception for it to catch. If we comment line 4 and 5, we will not get a compile error because even though Exception is a checked exception, it is ok for the enclosing block to not throw new Exception, when the catch is catching just Exception. If we coment out line 6, then line 4 will throw an exception because only IOException (which is a subclass of Exception) is caught.**

1. **If a method is calling another method which is throwing any checked method, the caller method needs to handle that exception i.e either have a try and catch for that thrown exception or redeclare the Exception with a throws statement.**

**private int** returnInt(**int** someVal)**throws** Exception{  
 **….**  
}  
  
**private void** intClient() **throws** Exception{  
 returnInt(0);  
}

**If we comment out the throws Exception part from the intClient() method, then the returnInt(0) call in this method will throw a compiler error. We could have caught the checked exception in a try catch block as well.**

1. **public void** testCaughtExceptionsGetOverriden(){ ………………………..1  
    String val = **""**; ………………………..2  
    **try**{ ………………………..3  
    val += **"try"**; ………………………..4  
    System.***out***.println(**"try"**); ………………………..5  
    **throw new** StringIndexOutOfBoundsException();………………………..6  
    }**catch**(StringIndexOutOfBoundsException e){ ………………………..7  
    val += **"catch1"**; ………………………..8  
    System.***out***.println(**"catch1"**); ………………………..9  
    **throw new** ArrayIndexOutOfBoundsException();………………………..10  
    }**catch**(ArrayIndexOutOfBoundsException e){ ………………………..11  
    val += **"catch2"**; ………………………..12  
    System.***out***.println(**"catch2"**); ………………………..13  
    }**finally**{ ………………………..14  
    val += **"finally"**; ………………………..15  
    System.***out***.println(**"finally"**); ………………………..16  
    **throw new** StringIndexOutOfBoundsException(); ………………………..17  
    }  
    }

The output of this method will be:   
try  
catch1  
finally  
java.lang.StringIndexOutOfBoundsException

Two important rules to look for here   
1) **At any given time only one catch block gets executed**. Notice there are two catch blocks, one in line 7 and the other in line 11. Line 10 throws an Array Index Out Of Bound which should technically be caught by the catch in line 11. This does not happen because of the rule at any given time only one catch block gets executed

2) Also notice the Exception thrown in line 10 is completely ignored and a StringIndexOutOfBound Exception is thrown. This is because the finally block throws an exception in line 17, had line 17 been commented out, then an ArrayIndexOutOfBoundException would be thrown.

Note that if an exception is thrown, then a return type is not required

**private int** returnInt(**int** someVal)**throws** Exception{  
 **throw new** Exception();  
}

**Constructors for Exception**

**public Exception()**

Constructs a new exception with null as its detail message. The cause is not initialized, and may subsequently be initialized by a call to [Throwable.initCause(java.lang.Throwable)](https://docs.oracle.com/javase/8/docs/api/java/lang/Throwable.html" \l "initCause-java.lang.Throwable-).

**public Exception([String](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html" \o "class in java.lang) message)**

Constructs a new exception with the specified detail message. The cause is not initialized, and may subsequently be initialized by a call to [Throwable.initCause(java.lang.Throwable)](https://docs.oracle.com/javase/8/docs/api/java/lang/Throwable.html" \l "initCause-java.lang.Throwable-).

**public Exception([Throwable](https://docs.oracle.com/javase/8/docs/api/java/lang/Throwable.html" \o "class in java.lang) cause)**

Constructs a new exception with the specified cause and a detail message of (cause==null ? null : cause.toString()) (which typically contains the class and detail message of cause). This constructor is useful for exceptions that are little more than wrappers for other throwables (for example, [PrivilegedActionException](https://docs.oracle.com/javase/8/docs/api/java/security/PrivilegedActionException.html" \o "class in java.security)).

**protected Exception([String](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html" \o "class in java.lang) message,**

[**Throwable**](https://docs.oracle.com/javase/8/docs/api/java/lang/Throwable.html)**cause,**

**boolean enableSuppression,**

**boolean writableStackTrace)**

Constructs a new exception with the specified detail message, cause, suppression enabled or disabled, and writable stack trace enabled or disabled.

**public Exception([String](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html" \o "class in java.lang) message,**

[**Throwable**](https://docs.oracle.com/javase/8/docs/api/java/lang/Throwable.html)**cause)**

Constructs a new exception with the specified detail message and cause.

Output of below code is, note the print statement in finally clause is displayed first. Also notice that there’s a throw e in the catch block of the testTryCatchFinallyExceptionOutput method but the compiler does not complain that the exception is not declared to be thrown in the method, now if you put throws Exception in the declaration of the testReturnInsideTryAndCatch method, then the compiler complains about the “throw e” statement.

**You got that right**

**Exception caught!**

**java.lang.RuntimeException: Invalid division**

**public int** testReturnInsideTryAndCatch() {  
 **try** {  
 **return** 0 / 0;  
 }**catch**(Exception e){  
 **throw new** RuntimeException(**"Invalid division"**);  
 }**finally**{  
 System.***out***.println(**"You got that right"**);  
 }  
}  
  
@Test  
 **public void** testTryCatchFinallyExceptionOutput() {  
 **try** {  
 testReturnInsideTryAndCatch();  
 }**catch**(Exception e){  
 System.***out***.println(**'A'**);  
 **throw** e;  
 }  
}