**Exception FAQ**

**1.What is an exception?**

An *exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

**2.What is error?**

An Error indicates that a non-recoverable condition has occurred that should not be caught. Error, a subclass of Throwable, is intended for drastic problems, such as OutOfMemoryError, which would be reported by the JVM itself.

**3.Which is superclass of Exception?**

**"Throwable"**, the parent class of all exception related classes.

**4.What are the advantages of using exception handling?**

Exception handling provides the following advantages over "traditional" error management techniques:

* Separating Error Handling Code from "Regular" Code.
* Propagating Errors Up the Call Stack.
* Grouping Error Types and Error Differentiation.

**5.What are the types of Exceptions in Java**

There are two types of exceptions in Java, unchecked exceptions and checked exceptions.

* **Checked exceptions**: A checked exception is some subclass of Exception (or Exception itself), excluding class RuntimeException and its subclasses. Each method must either handle all checked exceptions by supplying a catch clause or list each unhandled checked exception as a thrown exception.
* **Unchecked exceptions:** All Exceptions that extend the RuntimeException class are unchecked exceptions. Class Error and its subclasses also are unchecked.

**6.Why Errors are Not Checked?**

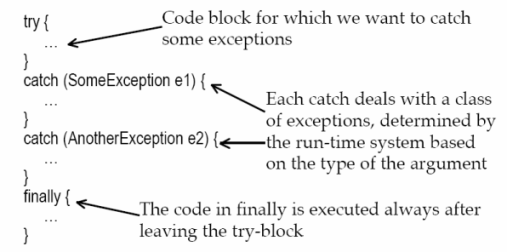
A unchecked exception classes which are the *error* classes (Error and its subclasses) are exempted from compile-time checking because they can occur at many points in the program and recovery from them is difficult or impossible. A program declaring such exceptions would be pointlessly.

**7.Why Runtime Exceptions are Not Checked?**

The *runtime exception* classes (RuntimeException and its subclasses) are exempted from compile-time checking because, in the judgment of the designers of the Java programming language, having to declare such exceptions would not aid significantly in establishing the correctness of programs. Many of the operations and constructs of the Java programming language can result in runtime exceptions. The information available to a compiler, and the level of analysis the compiler performs, are usually not sufficient to establish that such run-time exceptions cannot occur, even though this may be obvious to the programmer. Requiring such exception classes to be declared would simply be an irritation to programmers.

**8.Explain the significance of try-catch blocks?**

Whenever the exception occurs in Java, we need a way to tell the JVM what code to execute. To do this, we use the try and catch keywords. The try is used to define a block of code in which exceptions may occur. One or more catch clauses match a specific exception to a block of code that handles it.

  
  
**9.What is the use of finally block?**

|  |
| --- |
|  |

The finally block encloses code that is always executed at some point after the try block, whether an exception was thrown or not. This is right place to close files, release your network sockets, connections, and perform any other cleanup your code requires.

**Note**: If the try block executes with no exceptions, the finally block is executed immediately after the try block completes. It there was an exception thrown, the finally block executes immediately after the proper catch block completes

**10.What if there is a break or return statement in try block followed by finally block?**

If there is a return statement in the try block, the finally block executes right after the return statement encountered, and before the return executes.

**11.Can we have the try block without catch block?**

Yes, we can have the try block without catch block, but finally block should follow the try block.  
**Note:** It is not valid to use a try clause without either a catch clause or a finally clause.

**12.What is the difference throw and throws?**

**throws:** Used in a method's signature if a method is capable of causing an exception that it does not handle, so that callers of the method can guard themselves against that exception. If a method is declared as throwing a particular class of exceptions, then any other method that calls it must either have a try-catch clause to handle that exception or must be declared to throw that exception (or its superclass) itself.

A method that does not handle an exception it throws has to announce this:

public void myfunc(int arg) **throws** MyException {  
        …  
    }

**throw:** Used to trigger an exception. The exception will be caught by the nearest try-catch clause that can catch that type of exception. The flow of execution stops immediately after the throw statement; any subsequent statements are not executed.

To throw an user-defined exception within a block, we use the throw command:

**throw** new MyException("I always wanted to throw an exception!");

**13.How to create custom exceptions?**

A. By extending the Exception class or one of its subclasses.

**Example:**

class MyException extends Exception {  
  public MyException() { super(); }  
  public MyException(String s) { super(s); }  
  }

**14.What are the different ways to handle exceptions?**

|  |
| --- |
|  |

There are two ways to handle exceptions:

* Wrapping the desired code in a try block followed by a catch block to catch the exceptions.
* List the desired exceptions in the throws clause of the method and let the caller of the method handle those exceptions.