**Explain the use of the throws clause?**

From: [https://docs.oracle.com/javase/tutorial/essential/exceptions/throwing.html (Links to an external site.)](https://docs.oracle.com/javase/tutorial/essential/exceptions/throwing.html)

Any code can throw an exception… your code, package code or someone else’s code.

The exception is thrown with the throw statement.

The Java exception classes are descendants of the Throwable class.

You can also create your own exception classes for problems that can occur within your classes.

All methods use the throw statement to throw an exception.

The throw statement requires a throwable object as an argument.

The Throwable objects are instances of any subclass of the Throwable class.

You can only throw objects that inherit from the java.lang.Throwable class

To use a throws clause you add the throws clause to the method declaration.

The throws clause consists of the throws keyword and a comma separated list of exceptions that can be thrown by the method.

**When is it required?**

If a method doesn’t catch the checked exceptions that can occur within it, it must specify that it can throw the exceptions.

From: [https://docs.oracle.com/javase/tutorial/essential/exceptions/runtime.html (Links to an external site.)](https://docs.oracle.com/javase/tutorial/essential/exceptions/runtime.html)

*Any Exception that can be thrown by a method is part of the method's public programming interface. Those who call a method must know about the exceptions that a method can throw so that they can decide what to do about them. These exceptions are as much a part of that method's programming interface as its parameters and return value…*

*Here's the bottom line guideline: If a client can reasonably be expected to recover from an exception, make it a checked exception. If a client cannot do anything to recover from the exception, make it an unchecked exception.*

I found the following from:  [http://pages.cs.wisc.edu/~hasti/cs368/JavaTutorial/NOTES/Exceptions.html (Links to an external site.)](http://pages.cs.wisc.edu/~hasti/cs368/JavaTutorial/NOTES/Exceptions.html)

* most of the built-in exceptions (e.g., NullPointerException, IndexOutOfBoundsException) are *unchecked*.
* IOExceptions (e.g., FileNotFoundException) are checked
* user-defined exceptions should usually be *checked*, so they should be subclasses of Exception.

For example, the method createNewFile() throws IOException.

If you use createNewFile() in a program and do not include a try block, then you must include the throws clause in the method declaration.

The IOException class extends Exception class and Exception extends Throwable.

IOException Signals that an I/O exception of some sort has occurred. This class is the general class of exceptions produced by failed or interrupted I/O operations.

[https://docs.oracle.com/javase/7/docs/api/java/io/IOException.html (Links to an external site.)](https://docs.oracle.com/javase/7/docs/api/java/io/IOException.html)

The File constructor can throw a NullPointerException but it does not require a throws clause in the example below.

 checked vs unchecked exceptions (from the book):

The Java compiler enforces special conditions for checked exceptions. The type of exception determines if it is checked or unchecked.

An exception is unchecked if it a direct or indirect subclass of RuntimeException (usually from defects in code such as ArrayIndexOutOfBoundsException or ArithemticException.

Also unchecked is any class that inherits from class Error

.

An exception is checked if inherits directly or indirictly from class Exception but not from class RuntimeException. (example file processing or file does not exist).

*"The compiler checks each method call and method declaration to determine whether the method throws a checked exception.*

*If so, the compiler verifies that the checked exception is caught or is declared in a throws clause—this is known as the****catch-or-declare requirement.***

*... To satisfy the catch part the code that generates the exception must be wrapped in a tryblock and must provide a catch handler for the checked-exception type (or one of its superclasses).*

*To satisfy the declare part of the catch-or-declare requirement, the method containing the code that generates the exception must provide a throws clause containing the checked-exception type after its parameter list and before its method body.*

*If the catch-or-declare requirement is not satisfied, the compiler will issue an error message. This forces you to think about the problems that may occur when a method that throws checked exceptions is called."*

Also from the book:

"*Although the compiler does not enforce the catch-or-declare requirement for unchecked exceptions, provide appropriate exception-handling code when it’s known that such exceptions might occur.*

*For example, a program should process theNumberFormatException from Integer method parseInt, even though NumberFormatException is an indirect subclass ofRuntimeException (and thus an unchecked exception). This makes your programs more robust."*

**Provide a small unique code snippet illustrating the use of the clause.**

import java.io.File;

import java.io.IOException;

import java.util.Scanner;

public class ThrowsClauseExample {

    public static void main(String[] args) throws IOException {

        Scanner keyboard = new Scanner(System.in);

        boolean result = false;

        //createNewFile without try/catch - ioexception

        System.out.println("createNewFile Test 1: Creating a new file with an invalid path");

        System.out.printf("The next test throws an exception & causes the program to halt. "

                + "Do you want to continue? y n : ");

        String runTest = keyboard.nextLine();

        if(runTest.equals("y"))

        {

            System.out.println("Test creating a file with an invalid path: ");

            String filename2 = "ThisisanInvalidPath\\x\\xxx\\x.file";

            // this can throw a NullPointerException but it does not require a throws clause

            File f2 = new File(filename2);

            //this can throw an IOException and requires a throws clause

            result = f2.createNewFile();

            if(result)

                System.out.println("createNewFile Test 1: Failed " + filename2 + " created" );

            else

                System.out.println("createNewFile Test 1: Success " + filename2 + " not created");

        }

        else

            System.out.println("createNewFile Test 1: did not run");

        System.out.println();

    }

}