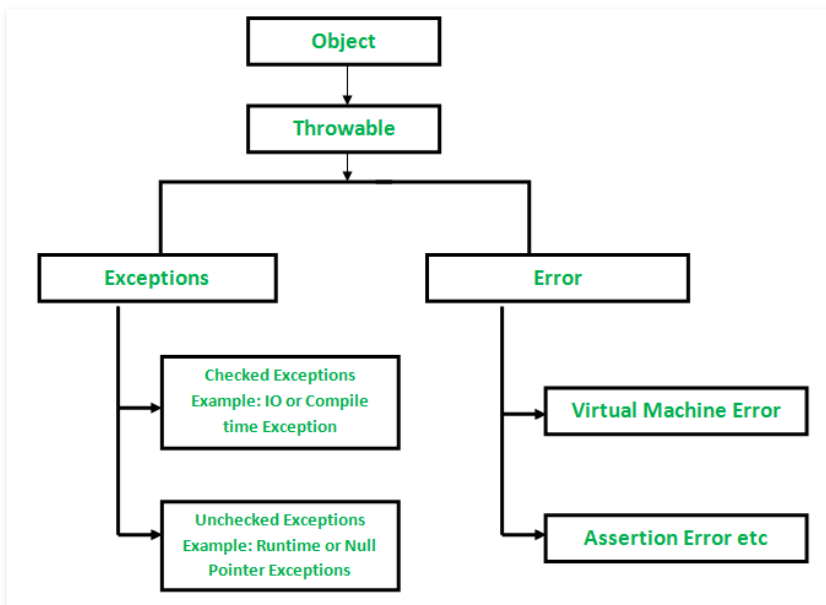


Checked vs Unchecked Exceptions in Java

Difficulty Level : Easy Last Updated : 01 Nov, 2021

An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e at run time, that disrupts the normal flow of the program's instructions. In Java, there are two types of exceptions:

1. Checked exceptions
2. Unchecked exceptions



Checked Exceptions

These are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using the throws keyword.

For example, consider the following Java program that opens the file at location “C:\test\a.txt” and prints the first three lines of it. The program doesn't compile, because the function main() uses FileReader() and FileReader() throws a checked exception *FileNotFoundException*. It also uses readLine() and close() methods, and these methods also throw checked exception *IOException*

Example:

```
// Java Program to Illustrate Checked Exceptions
// Where FileNotFoundException occurred

// Importing I/O classes
import java.io.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {

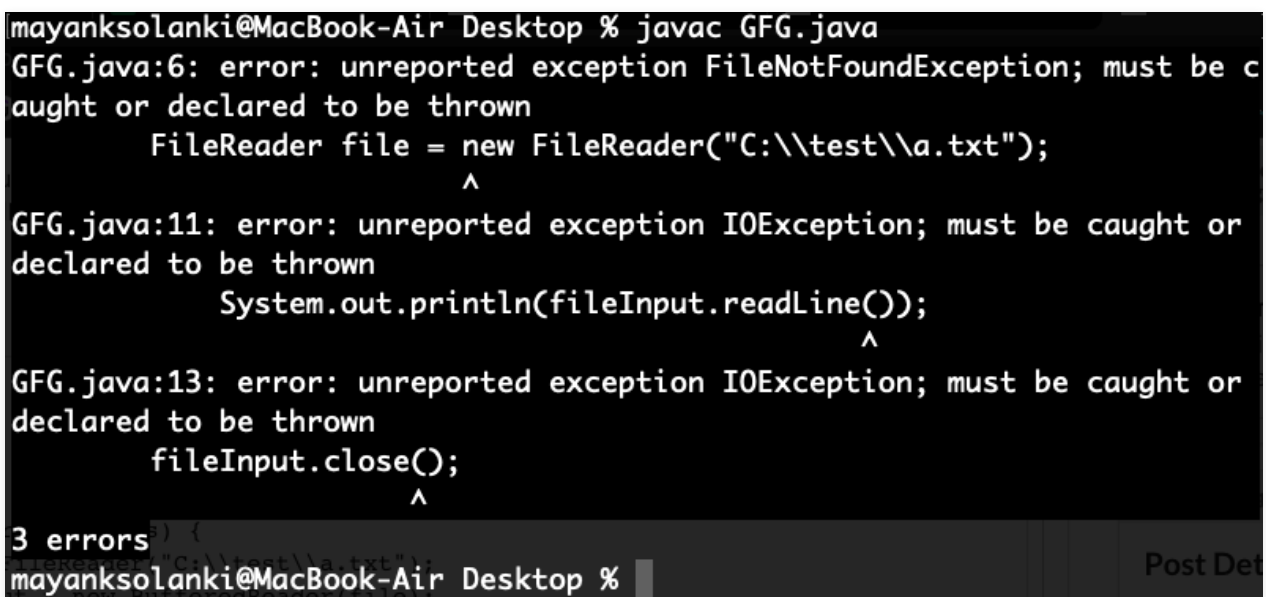
        // Reading file from path in local directory
        FileReader file = new FileReader("C:\\test\\a.txt");

        // Creating object as one of ways of taking input
        BufferedReader fileInput = new BufferedReader(file);

        // Printing first 3 lines of file "C:\\test\\a.txt"
        for (int counter = 0; counter < 3; counter++)
            System.out.println(fileInput.readLine());

        // Closing file connections
        // using close() method
        fileInput.close();
    }
}
```

Output:



```
mayanksolanki@MacBook-Air Desktop % javac GFG.java
GFG.java:6: error: unreported exception FileNotFoundException; must be c
aught or declared to be thrown
        FileReader file = new FileReader("C:\\test\\a.txt");
                        ^
GFG.java:11: error: unreported exception IOException; must be caught or
declared to be thrown
        System.out.println(fileInput.readLine());
                        ^
GFG.java:13: error: unreported exception IOException; must be caught or
declared to be thrown
        fileInput.close();
                ^
3 errors
mayanksolanki@MacBook-Air Desktop %
```

To fix the above program, we either need to specify a list of exceptions using throws, or we need to use a try-catch block. We have used throws in the below program.

Since *FileNotFoundException* is a subclass of *IOException*, we can just specify *IOException* in the throws list and make the above program compiler-error-free.

Example:

```
// Java Program to Illustrate Checked Exceptions
// Where FileNotFoundException does not occur

// Importing I/O classes
import java.io.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
        throws IOException
    {

        // Creating a file and reading from local repository
        FileReader file = new FileReader("C:\\test\\a.txt");

        // Reading content inside a file
        BufferedReader fileInput = new BufferedReader(file);

        // Printing first 3 lines of file "C:\\test\\a.txt"
        for (int counter = 0; counter < 3; counter++)
            System.out.println(fileInput.readLine());

        // Closing all file connections
        // using close() method
        // Good practice to avoid any memory leakage
        fileInput.close();
    }
}
```

Output:

First three lines of file "C:\\test\\a.txt"

Unchecked Exceptions

These are the exceptions that are not checked at compile time. In C++, all exceptions are unchecked, so it is not forced by the compiler to either handle or specify the exception. It is up to the programmers to be civilized, and specify or catch the exceptions. In Java exceptions under *Error* and *RuntimeException* classes are unchecked exceptions, everything else under *throwable* is checked.

Consider the following Java program. It compiles fine, but it throws *ArithmeticException* when run. The compiler allows it to compile because *ArithmeticException* is an unchecked exception.

Example:

```
// Java Program to Illustrate Un-checked Exceptions

// Main class
class GFG {

    // Main driver method
    public static void main(String args[])
    {
        // Here we are dividing by 0
        // which will not be caught at compile time
        // as there is no mistake but caught at runtime
        // because it is mathematically incorrect
        int x = 0;
        int y = 10;
        int z = y / x;
    }
}
```

Output:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Main.main(Main.java:5)
Java Result: 1
```

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above