Checked vs Unchecked Exceptions in Java

In Java, there are two types of exceptions:

1) Checked: 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 *throws* keyword.

For example, consider the following Java program that opens 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*

```
import java.io.*;
class Main {
	public static void main(String[] args) {
		FileReader file = new FileReader("C:\\test\\a.txt");
		BufferedReader fileInput = new BufferedReader(file);

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

		fileInput.close();
	}
}
```

Output:

```
Exception in thread "main" java.lang.RuntimeException: Uncompilable
source code -
unreported exception java.io.FileNotFoundException; must be caught
or declared to be
thrown
    at Main.main(Main.java:5)
```

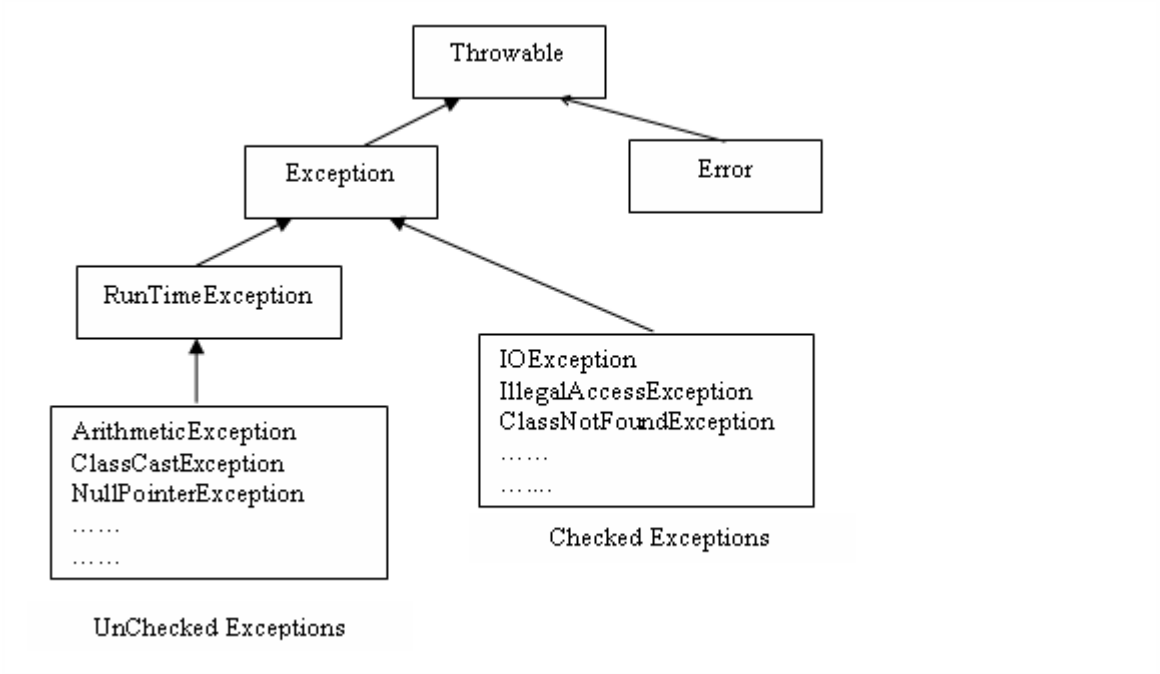
To fix the above program, we either need to specify list of exceptions using throws, or we need to use try-catch block. We have used throws in the below program. Since *FileNotFoundException* is a subclass of *IOException*, we can just specify *IOException* in the throws list and make the above program compiler-error-free.

```
import java.io.*;

class Main {
        public static void main(String[] args) throws IOException {
                FileReader file = new FileReader("C:\\test\\a.txt");
                BufferedReader fileInput = new BufferedReader(file);

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

                fileInput.close();
        }
}
```

Output: First three lines of file “C:\test\a.txt”

2) Unchecked are the exceptions that are not checked at compiled 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.

```
class Main {
public static void main(String args[]) {
        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
```

Throws clause in java – Exception handling

As we know that there are two types of exception checked and unchecked. Checked exception (compile time) force you to handle them, if you don't handle them then the program will not compile.
On the other hand unchecked exception (Runtime) doesn't get checked during compilation. **Throws keyword** is used for handling checked exceptions . By using throws we can declare multiple exceptions in one go.

What is the need of having throws keyword when you can handle exception using try-catch?
Well, thats a valid question. We already know we can handle exceptions using try-catch block.
The throws does the same thing that try-catch does but there are some cases where you would prefer throws over try-catch. For example:
Lets say we have a method `myMethod()` that has statements that can throw either ArithmeticException or NullPointerException, in this case you can use try-catch as shown below:

```
public void myMethod()
{
  try {
    // Statements that might throw an exception
  }
  catch (ArithmeticException e) {
    // Exception handling statements
  }
  catch (NullPointerException e) {
    // Exception handling statements
  }
}
```

But suppose you have several such methods that can cause exceptions, in that case it would be tedious to write these try-catch for each method. The code will become unnecessary long and will be less-readable.

One way to overcome this problem is by using throws like this: declare the exceptions in the method signature using throws and handle the exceptions where you are calling this method by using try-catch.
Another advantage of using this approach is that you will be forced to handle the exception when you call this method, all the exceptions that are declared

using throws, must be handled where you are calling this method else you will get compilation error.

```
public void myMethod() throws ArithmeticException,
NullPointerException
{
  // Statements that might throw an exception
}

public static void main(String args[]) {
  try {
    myMethod();
  }
  catch (ArithmeticException e) {
    // Exception handling statements
  }
  catch (NullPointerException e) {
    // Exception handling statements
  }
}
```

Example

```
public class Main {

  static void checkAge(int age) throws ArithmeticException {

    if (age < 18) {

      throw new ArithmeticException("Access denied - You must be at
least 18 years old.");

    }

    else {

      System.out.println("Access granted - You are old enough!");

    }

  }


  public static void main(String[] args) {

    checkAge(15); // Set age to 15 (which is below 18...)

  }

}
```

Example of User defined exception in Java

```
/* This is my Exception class, I have named it MyException
 * you can give any name, just remember that it should
 * extend Exception class
 */
class MyException extends Exception{
   String str1;
   /* Constructor of custom exception class
    * here I am copying the message that we are passing while
    * throwing the exception to a string and then displaying
    * that string along with the message.
    */
   MyException(String str2) {
       str1=str2;
   }
   public String toString(){
       return ("MyException Occurred: "+str1) ;
   }
}

class Example1{
   public static void main(String args[]){
       try{
              System.out.println("Starting of try block");
              // I'm throwing the custom exception using throw
              throw new MyException("This is My error Message");
       }
       catch(MyException exp){
              System.out.println("Catch Block") ;
              System.out.println(exp) ;
       }
   }
}
```

Output:

```
Starting of try block
Catch Block
MyException Occurred: This is My error Message
```

Another Example of Custom Exception

```
class InvalidProductException extends Exception
{
    public InvalidProductException(String s)
    {
        // Call constructor of parent Exception
        super(s);
    }
}

public class Example1
{
   void productCheck(int weight) throws InvalidProductException{
       if(weight<100){
             throw new InvalidProductException("Product
Invalid");
       }
   }

    public static void main(String args[])
    {
       Example1 obj = new Example1();
        try
        {
            obj.productCheck(60);
        }
        catch (InvalidProductException ex)
        {
            System.out.println("Caught the exception");
            System.out.println(ex.getMessage());
        }
    }
}
```

Output:

```
Caught the exception
Product Invalid
```

Exception Handling with Method Overriding in Java

There are many rules if we talk about method overriding with exception handling.

Some of the rules are listed below:

- **If the superclass method does not declare an exception**
  - If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.
- **If the superclass method declares an exception**
  - If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

If the superclass method does not declare an exception

> ***Rule 1: If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception.***

```
1. import java.io.*;
2. class Parent{
3.
4.   // defining the method
5.   void msg() {
6.     System.out.println("parent method");
7.     }
8. }
9.
10. public class TestExceptionChild extends Parent{
11.
12.  // overriding the method in child class
13.  // gives compile time error
14.  void msg() throws IOException {
15.    System.out.println("TestExceptionChild");
16.  }
```

```
17.
18.  public static void main(String args[]) {
19.   Parent p = new TestExceptionChild();
20.   p.msg();
21.  }
22. }
```

Output:

```
C:\Users\Anurati\Desktop\abcDemo>javac TestExceptionChild.java
TestExceptionChild.java:14: error: msg() in TestExceptionChild cannot override msg(
) in Parent
  void msg() throws IOException {
       ^
  overridden method does not throw IOException
1 error
```

Rule 2: If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but can declare unchecked exception.

TestExceptionChild1.java

```
1.  import java.io.*;
2.  class Parent{
3.    void msg() {
4.      System.out.println("parent method");
5.    }
6.  }
7.
8.  class TestExceptionChild1 extends Parent{
9.    void msg()throws ArithmeticException {
10.     System.out.println("child method");
11.   }
12.
13.   public static void main(String args[]) {
14.    Parent p = new TestExceptionChild1();
15.    p.msg();
```

```
16.  }
17. }
```

Output:

```
C:\Users\Anurati\Desktop\abcDemo>javac TestExceptionChild1.java

C:\Users\Anurati\Desktop\abcDemo>java TestExceptionChild1
child method
```

If the superclass method declares an exception

> ***Rule 1: If the superclass method declares an exception, subclass overridden method can declare the same subclass exception or no exception but cannot declare parent exception.***

Example in case subclass overridden method declares parent exception

TestExceptionChild2.java

```
1.  import java.io.*;
2.  class Parent{
3.    void msg()throws ArithmeticException {
4.      System.out.println("parent method");
5.    }
6.  }
7.
8.  public class TestExceptionChild2 extends Parent{
9.    void msg()throws Exception {
10.     System.out.println("child method");
11.   }
12.
13.   public static void main(String args[]) {
14.    Parent p = new TestExceptionChild2();
15.
16.    try {
```

```
17.    p.msg();
18.    }
19.    catch (Exception e){}
20.
21.   }
22. }
```

Output:

```
C:\Users\Anurati\Desktop\abcDemo>javac TestExceptionChild2.java
TestExceptionChild2.java:9: error: msg() in TestExceptionChild2 cannot override msg
() in Parent
  void msg()throws Exception {
       ^
  overridden method does not throw Exception
1 error
```

Example in case subclass overridden method declares same exception

TestExceptionChild3.java

```
1.  import java.io.*;
2.  class Parent{
3.    void msg() throws Exception {
4.      System.out.println("parent method");
5.    }
6.  }
7.
8.  public class TestExceptionChild3 extends Parent {
9.    void msg()throws Exception {
10.     System.out.println("child method");
11.   }
12.
13.   public static void main(String args[]){
14.    Parent p = new TestExceptionChild3();
15.
16.    try {
17.    p.msg();
```

```
18.   }
19.    catch(Exception e) {}
20.  }
21. }
```

Output:

```
C:\Users\Anurati\Desktop\abcDemo>javac TestExceptionChild3.java

C:\Users\Anurati\Desktop\abcDemo>java TestExceptionChild3
child method
```

Example in case subclass overridden method declares subclass exception

TestExceptionChild4.java

```
1.  import java.io.*;
2.  class Parent{
3.    void msg()throws Exception {
4.      System.out.println("parent method");
5.    }
6.  }
7.
8.  class TestExceptionChild4 extends Parent{
9.    void msg()throws ArithmeticException {
10.     System.out.println("child method");
11.   }
12.
13.   public static void main(String args[]){
14.    Parent p = new TestExceptionChild4();
15.
16.    try {
17.    p.msg();
18.    }
19.    catch(Exception e) {}
20.   }
```

```
21. }
```

Output:

```
C:\Users\Anurati\Desktop\abcDemo>javac TestExceptionChild4.java

C:\Users\Anurati\Desktop\abcDemo>java TestExceptionChild4
child method
```

Example in case subclass overridden method declares no exception

TestExceptionChild5.java

```
1.  import java.io.*;
2.  class Parent {
3.    void msg()throws Exception{
4.      System.out.println("parent method");
5.    }
6.  }
7.
8.  class TestExceptionChild5 extends Parent{
9.    void msg() {
10.     System.out.println("child method");
11.   }
12.
13.   public static void main(String args[]){
14.    Parent p = new TestExceptionChild5();
15.
16.    try {
17.    p.msg();
18.    }
19.    catch(Exception e) {}
20.
21.   }
22. }
```

Output:

```
C:\Users\Anurati\Desktop\abcDemo>javac TestExceptionChild5.java

C:\Users\Anurati\Desktop\abcDemo>java TestExceptionChild5
child method
```

Ref : https://www.javatpoint.com/exception-handling-with-method-overriding