Java Exception

https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html

What is an exception?

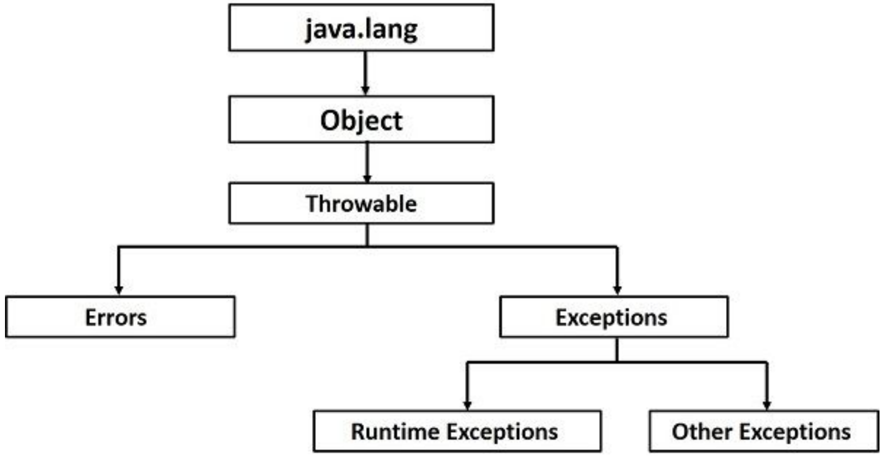
An ***exception*** is an event, **which occurs during the execution of a program**, that **disrupts the normal flow** of the program's instructions. For example,

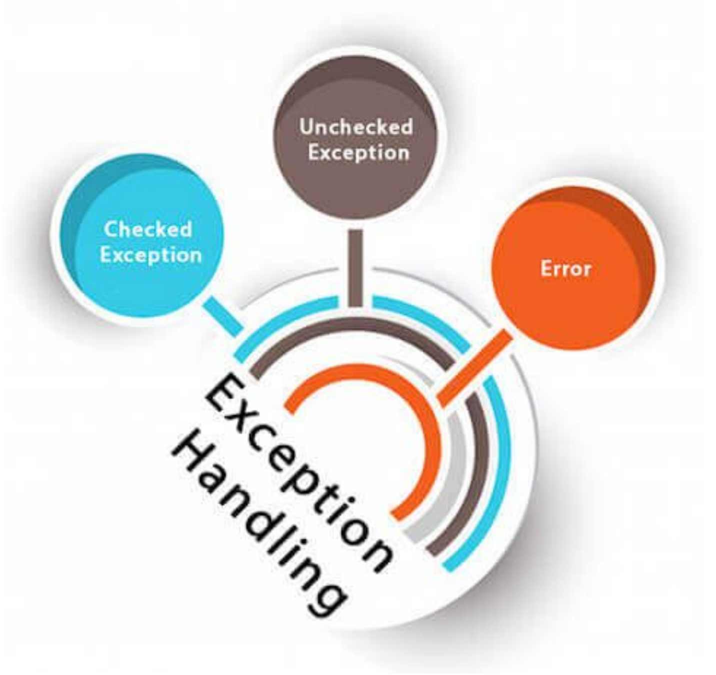
- *A user has **entered an invalid data**.*
- ***A file** that needs to be opened **cannot be found**.*
- ***A network connection has been lost** in the middle of communications or the JVM has run out of memory.*

Advantage of exception handling

- Exception handling **ensures that the flow of the program doesn’t break when an exception occurs**.

Different kinds of exceptions

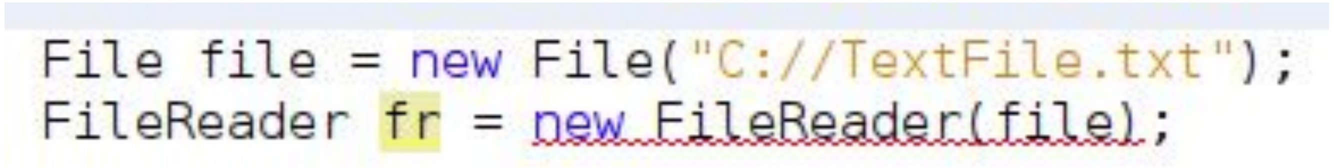




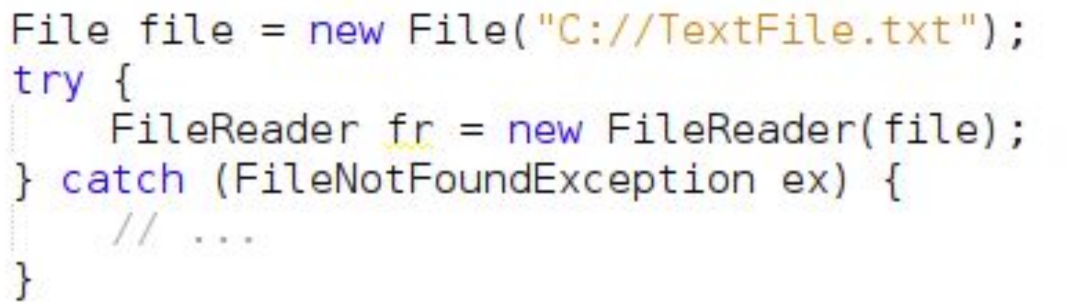
Different kinds of exceptions

Checked Exceptions

Checked exceptions are checked at compile-time. It means if a method is throwing a checked exception then it should handle the exception using **try-catch block** or it should declare the exception using **throws keyword**, otherwise the program will give a **compilation error**.



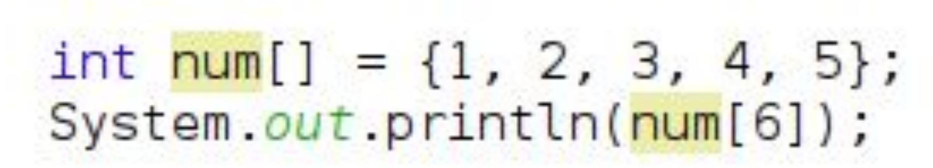
```
File file = new File("C://TextFile.txt");
FileReader fr = new FileReader(file);
```



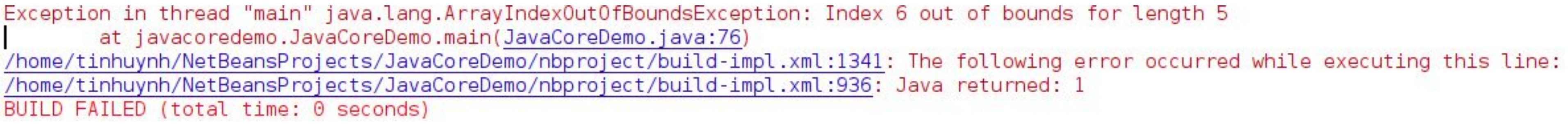
```
File file = new File("C://TextFile.txt");
try {
    FileReader fr = new FileReader(file);
} catch (FileNotFoundException ex) {
    // ...
}
```

Different kinds of exceptions
Unchecked Exceptions

Unchecked exceptions – An unchecked exception is an exception that occurs **at the time of execution**. These are also called as **Runtime Exceptions**.



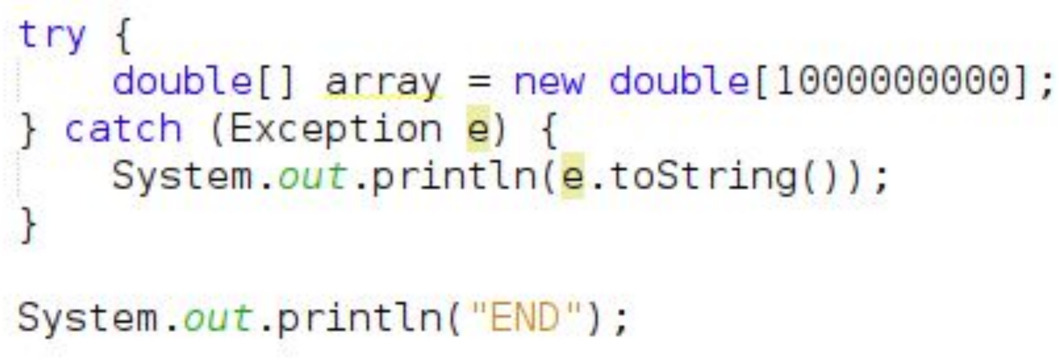
```java
int num[] = {1, 2, 3, 4, 5};
System.out.println(num[6]);
```



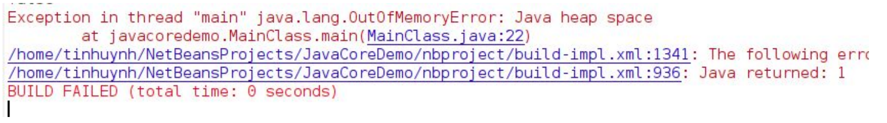
```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 6 out of bounds for length 5
	at javacoredemo.JavaCoreDemo.main(JavaCoreDemo.java:76)
/home/tinhuynh/NetBeansProjects/JavaCoreDemo/nbproject/build-impl.xml:1341: The following error occurred while executing this line:
/home/tinhuynh/NetBeansProjects/JavaCoreDemo/nbproject/build-impl.xml:936: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

Different kinds of exceptions
Errors

- **not handled by the Java programs**.
- generated to indicate **errors generated by the runtime environment**.
- Example: **JVM is out of memory**.
- Normally, **programs cannot recover from errors**.



```java
try {
    double[] array = new double[1000000000];
} catch (Exception e) {
    System.out.println(e.toString());
}

System.out.println("END");
```



```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at javacoredemo.MainClass.main(MainClass.java:22)
/home/tinhuynh/NetBeansProjects/JavaCoreDemo/nbproject/build-impl.xml:1341: The following err
/home/tinhuynh/NetBeansProjects/JavaCoreDemo/nbproject/build-impl.xml:936: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

How to handle an exception

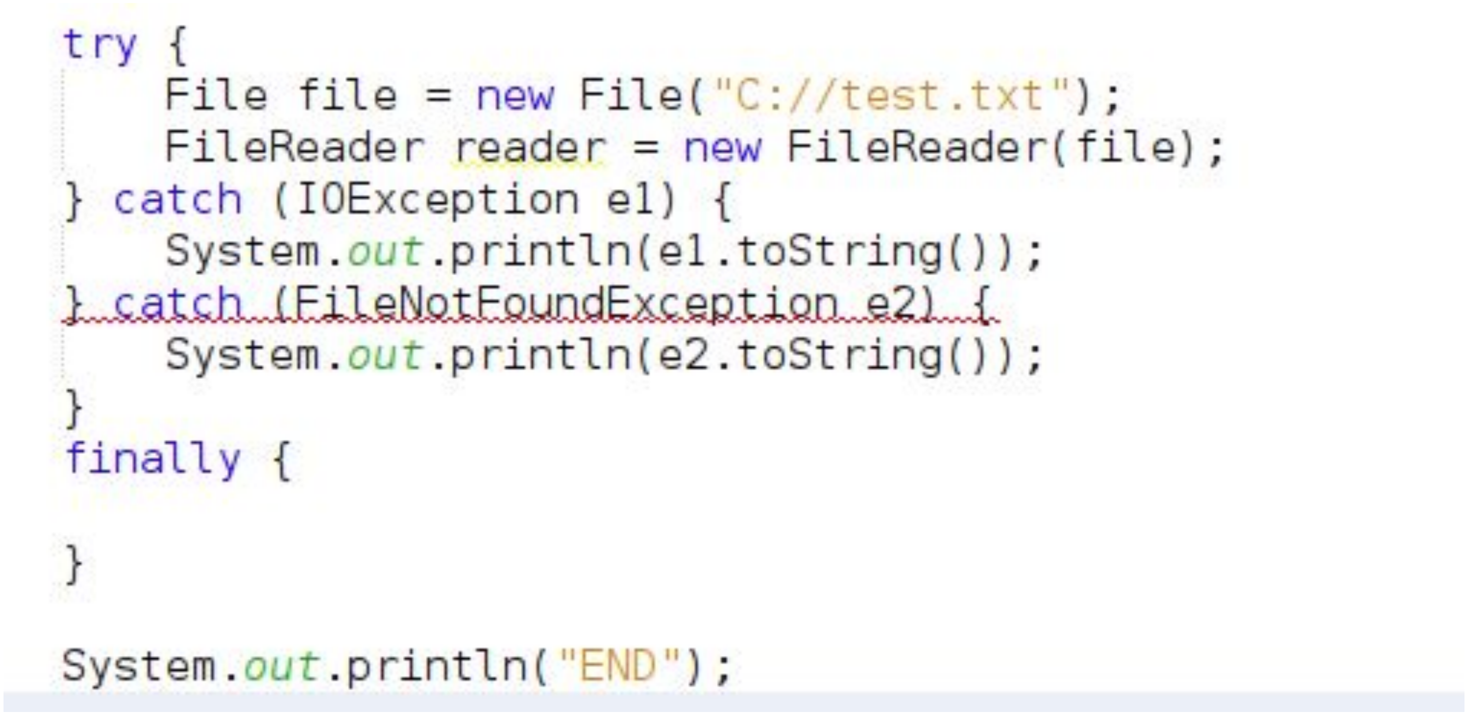
- **Customized Exception Handling :** Java exception handling is managed via five keywords: **try**, **catch**, **throw**, **throws**, and **finally**.
- Using the **try**, **catch**, and **finally** blocks
- Using the **try-with-resources** statement, introduced in Java SE 7

How to handle an exception
Try, catch, finally block

```
try {
   // Protected code
} catch (ExceptionType1 e1) {
   // Catch block
} catch (ExceptionType2 e2) {
   // Catch block
} catch (ExceptionType3 e3) {
   // Catch block
}
```

```
try {
   // Protected code
} catch (ExceptionType1 e1) {
   // Catch block
} catch (ExceptionType2 e2) {
   // Catch block
} catch (ExceptionType3 e3) {
   // Catch block
}finally {
   // The finally block always executes.
}
```

How to handle an exception
Try, catch, finally block

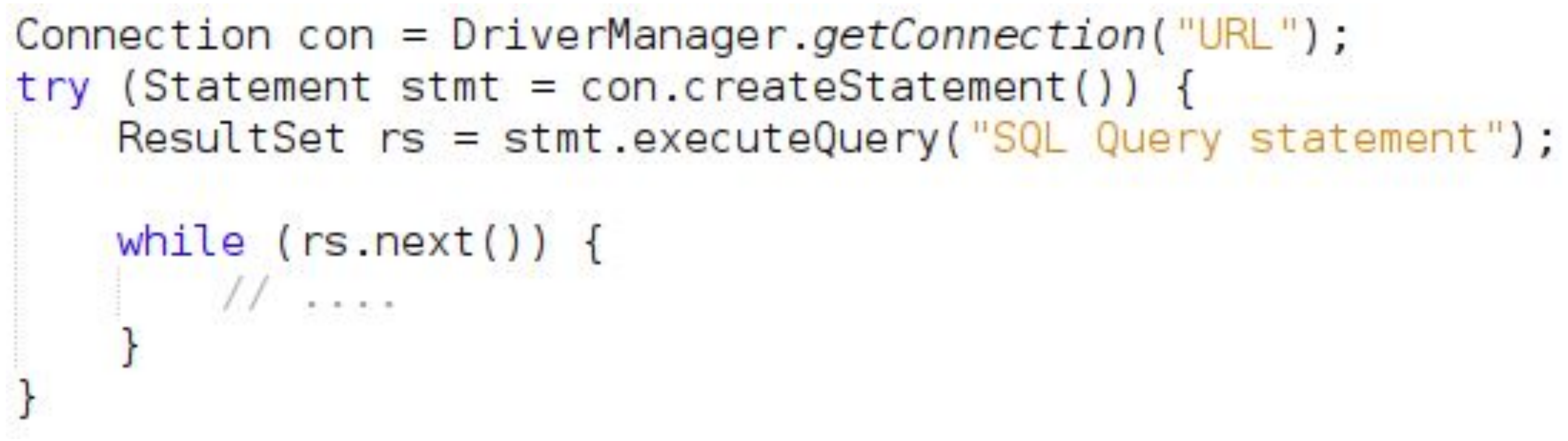


```
try {
    File file = new File("C://test.txt");
    FileReader reader = new FileReader(file);
} catch (IOException e1) {
    System.out.println(e1.toString());
} catch (FileNotFoundException e2) {
    System.out.println(e2.toString());
}
finally {

}

System.out.println("END");
```

WHY?

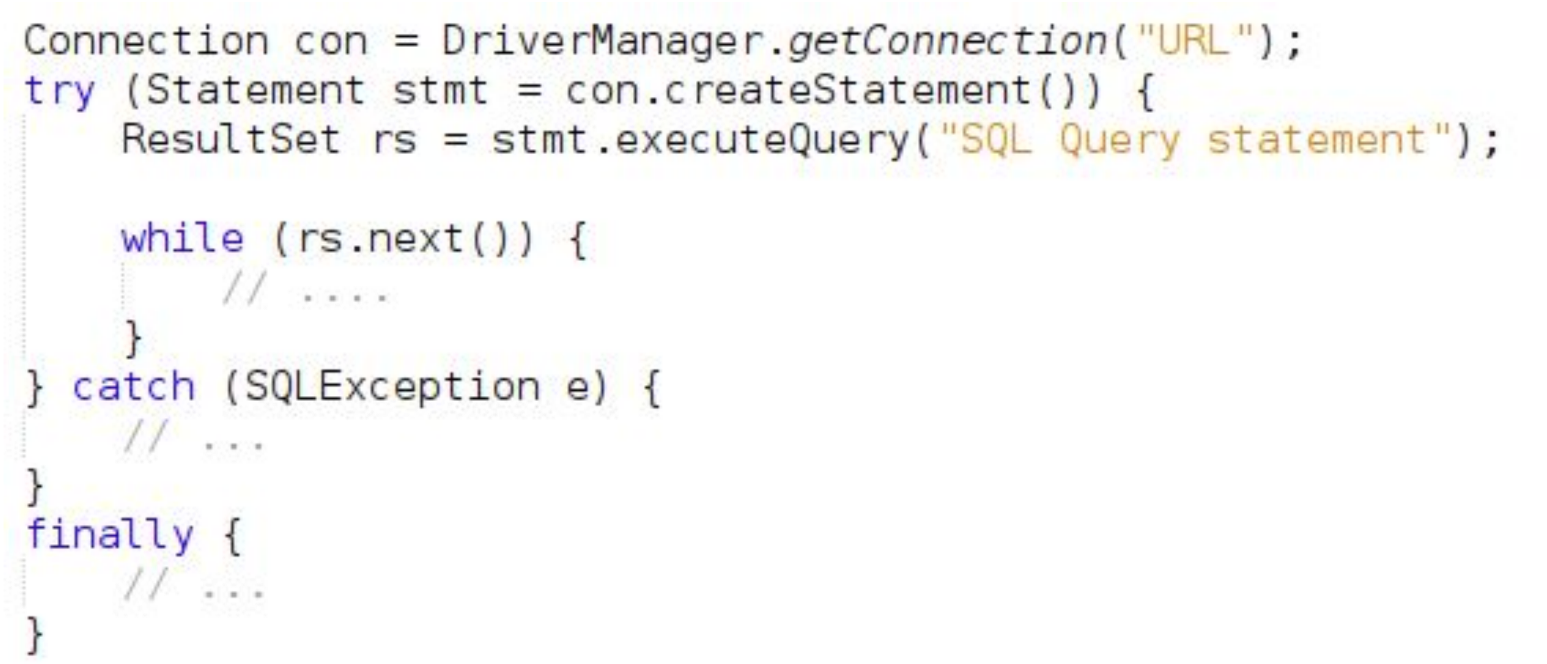
How to handle an exception
Try with resources

- **The try-with-resources** statement ensures that each resource is closed at the end of the statement.



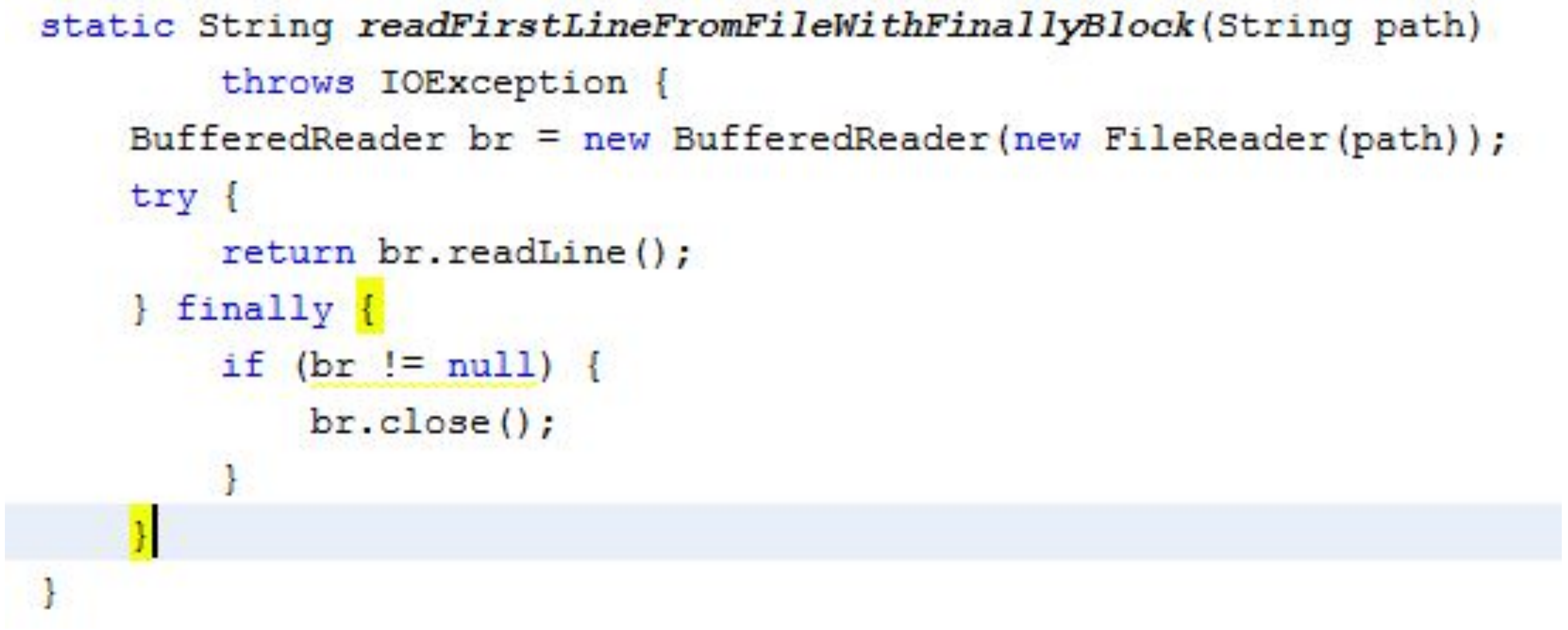
```
Connection con = DriverManager.getConnection("URL");
try (Statement stmt = con.createStatement()) {
    ResultSet rs = stmt.executeQuery("SQL Query statement");

    while (rs.next()) {
        // ....
    }
}
```

How to handle an exception
Try with resources



```
Connection con = DriverManager.getConnection("URL");
try (Statement stmt = con.createStatement()) {
    ResultSet rs = stmt.executeQuery("SQL Query statement");

    while (rs.next()) {
        // ....
    }
} catch (SQLException e) {
    // ...
}
finally {
    // ...
}
```

Note: In a **try-with-resources** statement, any catch or finally block is run after the resources declared have been closed.



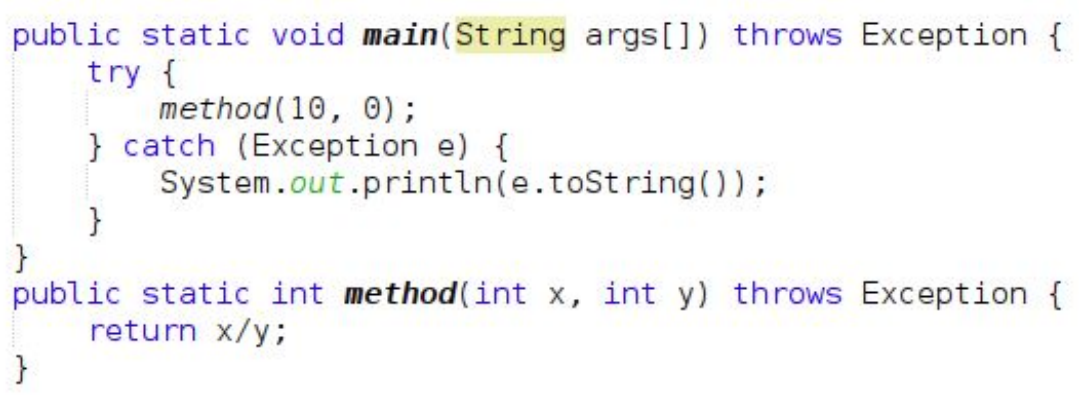
```java
static String readFirstLineFromFileWithFinallyBlock(String path)
        throws IOException {
    BufferedReader br = new BufferedReader(new FileReader(path));
    try {
        return br.readLine();
    } finally {
        if (br != null) {
            br.close();
        }
    }
}
```

- Nếu cả ***readLine()*** và ***close()*** throw các exceptions, thì phương thức ***readFirstLineFromFileWithFinallyBlock*** ném ra Exception mà được ***ném từ khối finally***, còn Exception ném từ ***khối try bị "ỉm đi".*** Còn với ***try-with-resource*** thì Exception trong khối ***try-with-resource sẽ bị "ỉm đi"***.

How to handle an exception

Specifying the exceptions thrown by a method

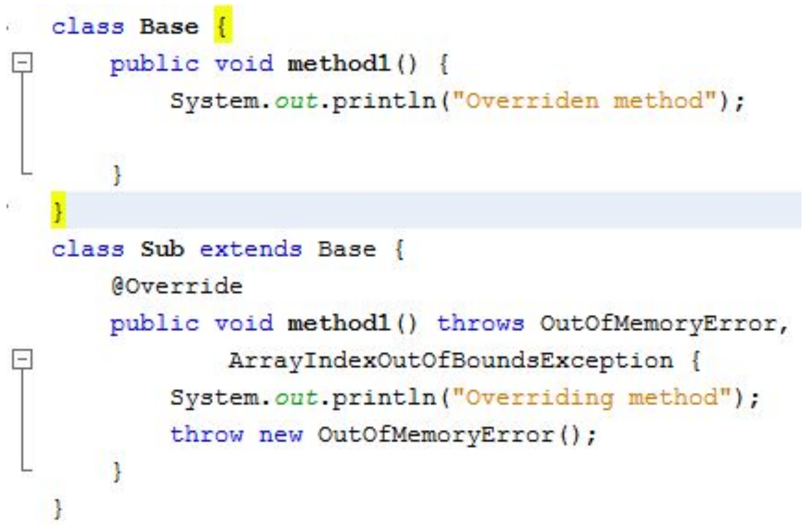
- An automated way of keeping track of methods that might throw an exception.
- The method can "**throw**" one or more types of exceptions **to the calling method instead of handling them**.



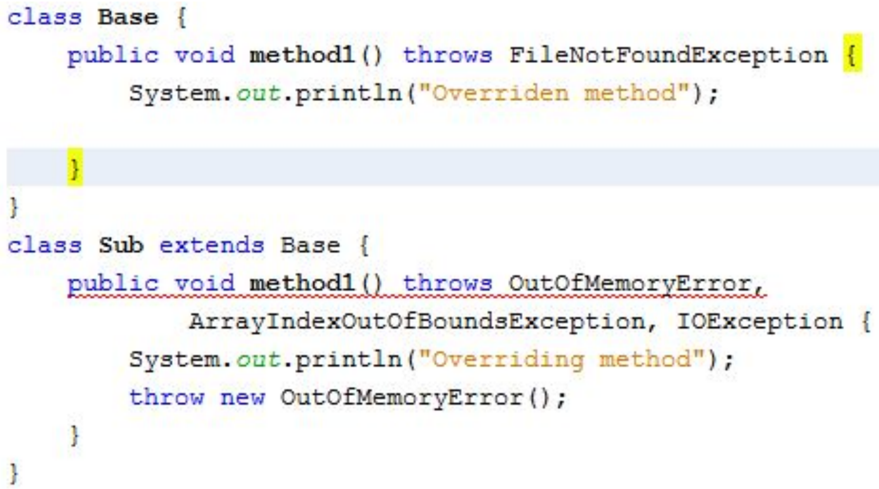
```
public static void main(String args[]) throws Exception {
    try {
        method(10, 0);
    } catch (Exception e) {
        System.out.println(e.toString());
    }
}
public static int method(int x, int y) throws Exception {
    return x/y;
}
```

```
method1() {
    try {
        call method2;
    } catch (exception e) {
        doErrorProcessing;
    }
}

method2() throws exception {
    call method3;
}

method3() throws exception {
    call readFile;
}
```

- Overriding method (trong lớp con) ***có thể throw/throws*** unchecked exception (RuntimeException hoặc Error), bất kể overridden method (trong lớp cha) có mô tả Exception hay không.
- Overriding method ***không thể throw/throws*** những checked exception "mới" hay "rộng hơn" các Exception mô tả trong overridden method.



```
class Base {
    public void method1() {
        System.out.println("Overriden method");

    }
}
class Sub extends Base {
    @Override
    public void method1() throws OutOfMemoryError,
            ArrayIndexOutOfBoundsException {
        System.out.println("Overriding method");
        throw new OutOfMemoryError();
    }
}
```



```
class Base {
    public void method1() throws FileNotFoundException {
        System.out.println("Overriden method");

    }
}
class Sub extends Base {
    public void method1() throws OutOfMemoryError,
            ArrayIndexOutOfBoundsException, IOException {
        System.out.println("Overriding method");
        throw new OutOfMemoryError();
    }
}
```

References

https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html
https://www.javatpoint.com/exception-handling-in-java
https://www.w3schools.com/java/java_try_catch.asp
https://www.geeksforgeeks.org/exceptions-in-java/
https://www.tutorialspoint.com/java/java_exceptions.htm

References

https://www.tutorialspoint.com/java/java_files_io.htm

https://docs.oracle.com/javase/tutorial/essential/io/streams.html

https://www.javatpoint.com/java-io