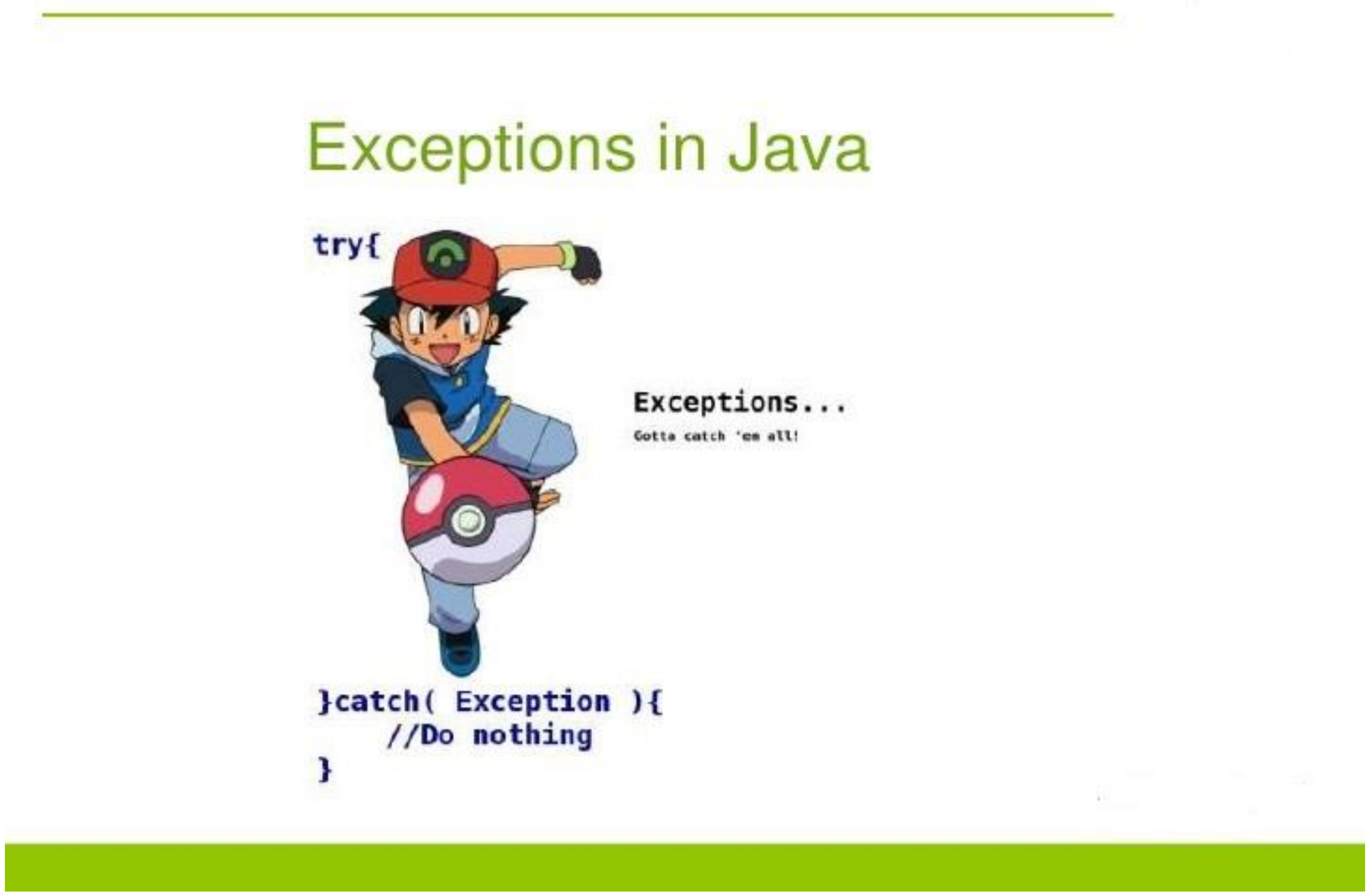
Java – Exceptions Handling



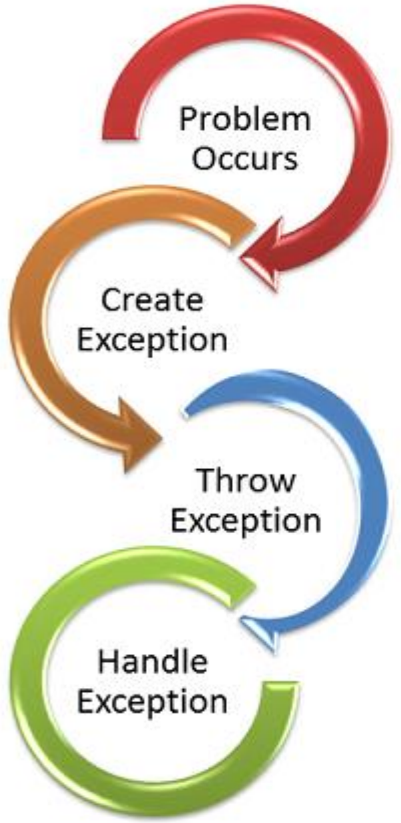
```
try{

}catch( Exception ){
    //Do nothing
}
```

Java – Exceptions Handling

- Exception is an abnormal condition.
- An exception (or exceptional event) is a problem that arises during the **execution** of a program.
- In java, exception is an event that disrupts the normal flow of the program.
- It is an object which is thrown at runtime.

What is exception handling

- Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.



Java – Exceptions

An exception can occur for many different reasons, below given are some scenarios where exception occurs.

- A user has entered 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.

Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

Types of Exception

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

- Checked Exception
- Unchecked Exception
- Error

Checked exceptions

- All exceptions other than Runtime Exceptions are known as Checked exceptions as the compiler checks them during compilation
- A checked exception is an exception that occurs at the compile time, these are also called as compile time exceptions.
- These exceptions cannot simply be ignored at the time of compilation

Examples of Checked Exceptions :

Some checked exceptions are as follows:

- ClassNotFoundException
- IllegalAccessException
- NotSuchFieldException
- EOFExceptionException

Unchecked Exceptions

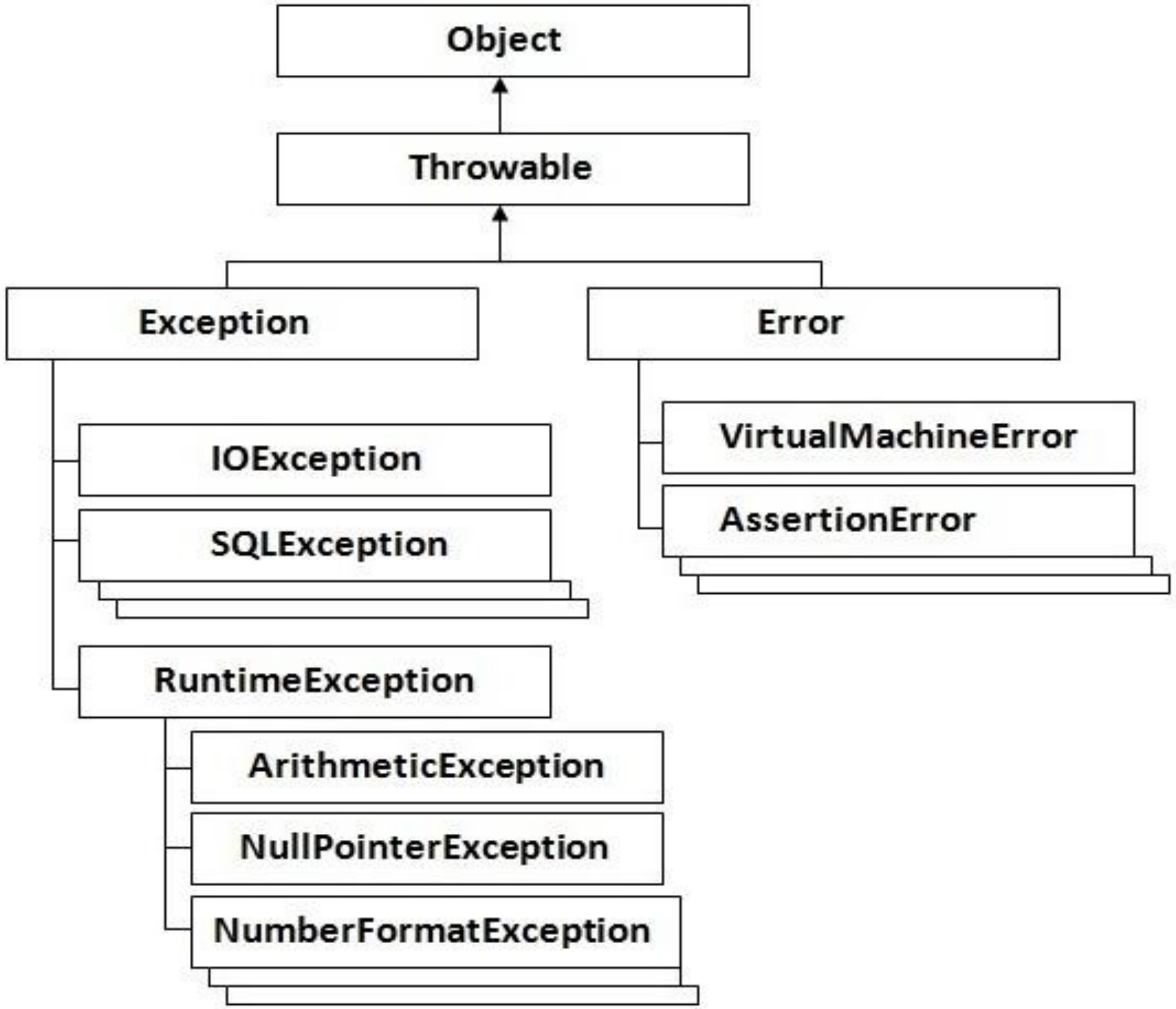
- Runtime Exceptions are also known as Unchecked Exceptions as the compiler do not check whether the programmer has handled them or not
- These exceptions need not be included in any method’s throws list because compiler does not check to see if a method handles or throws these exceptions.

Examples of Unchecked Exceptions :

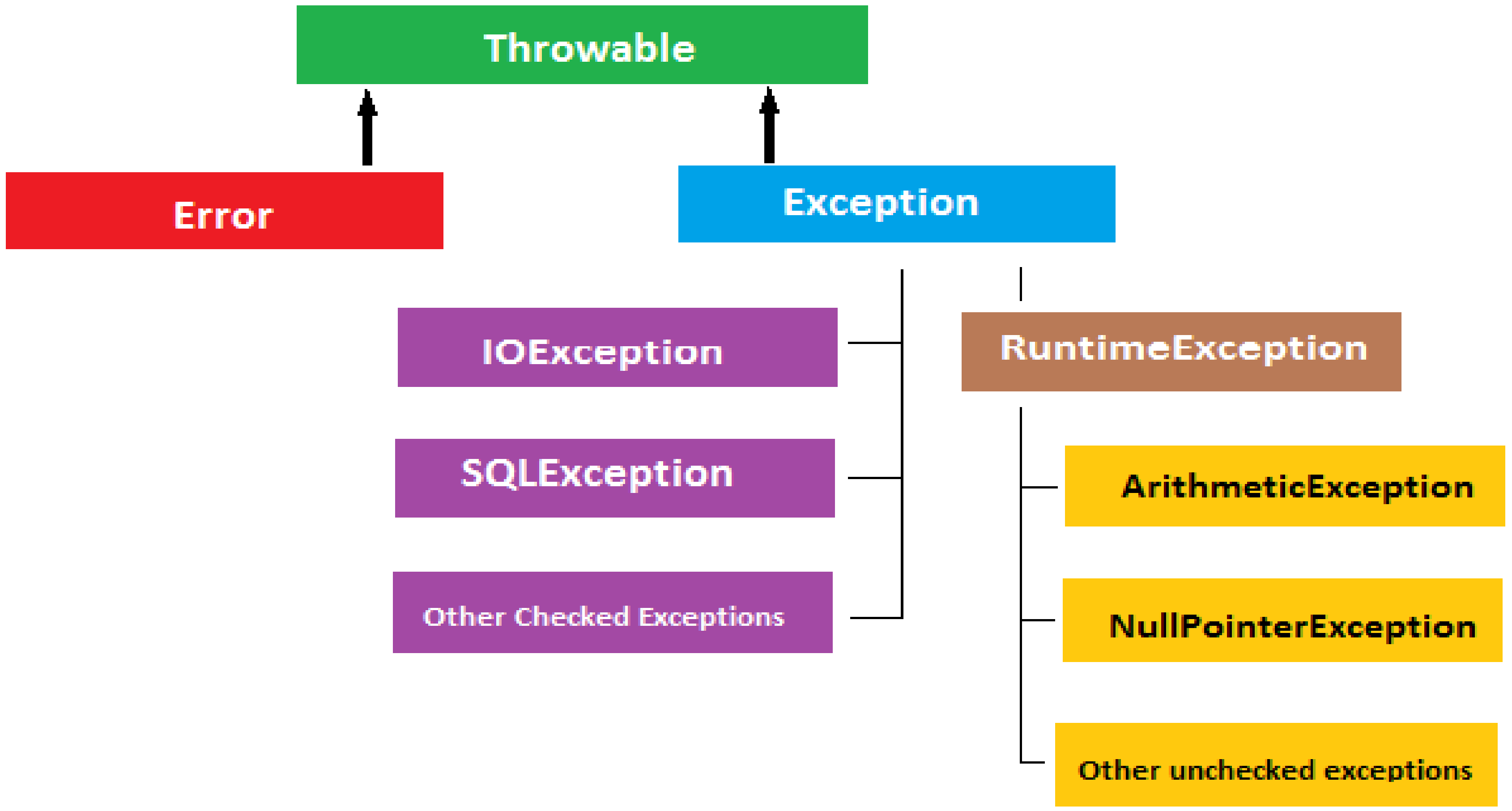
Some unchecked exceptions are as follows:

- ArithmaticException
- ArrayIndexOutOfBoundException
- NullPointerException
- NegativeArraySizeException

Hierarchy of Java Exception classes



Exception classes



Difference between checked and unchecked exceptions

- **Checked Exception**

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

- **Unchecked Exception**

The classes that extend RuntimeException are known as unchecked exceptions e.g.ArithmeticException,NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

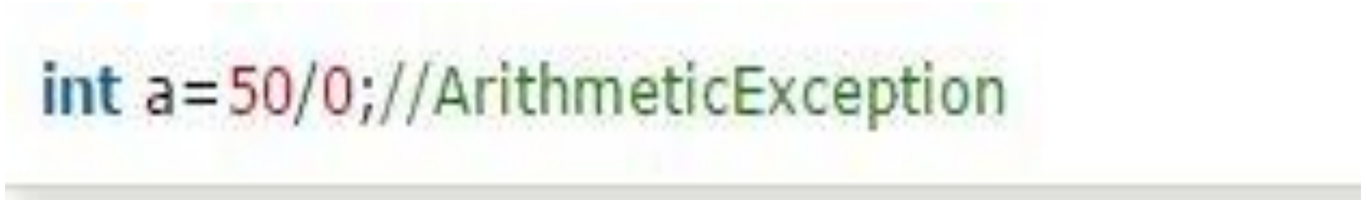
- **Error**

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Common scenarios where exceptions may occur

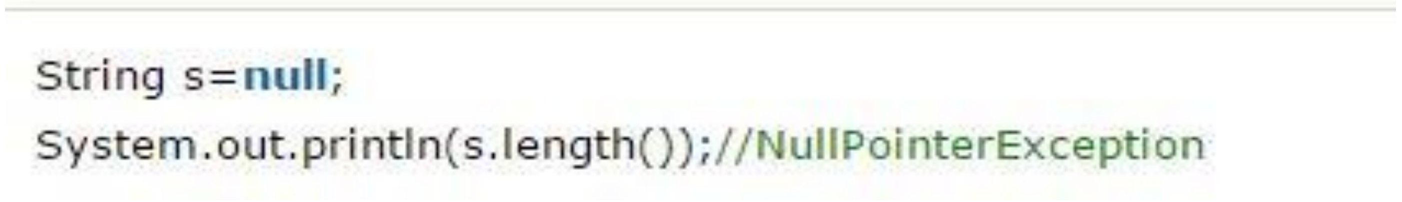
Scenario where ArithmeticException occurs

If we divide any number by zero, there occurs an ArithmeticException.



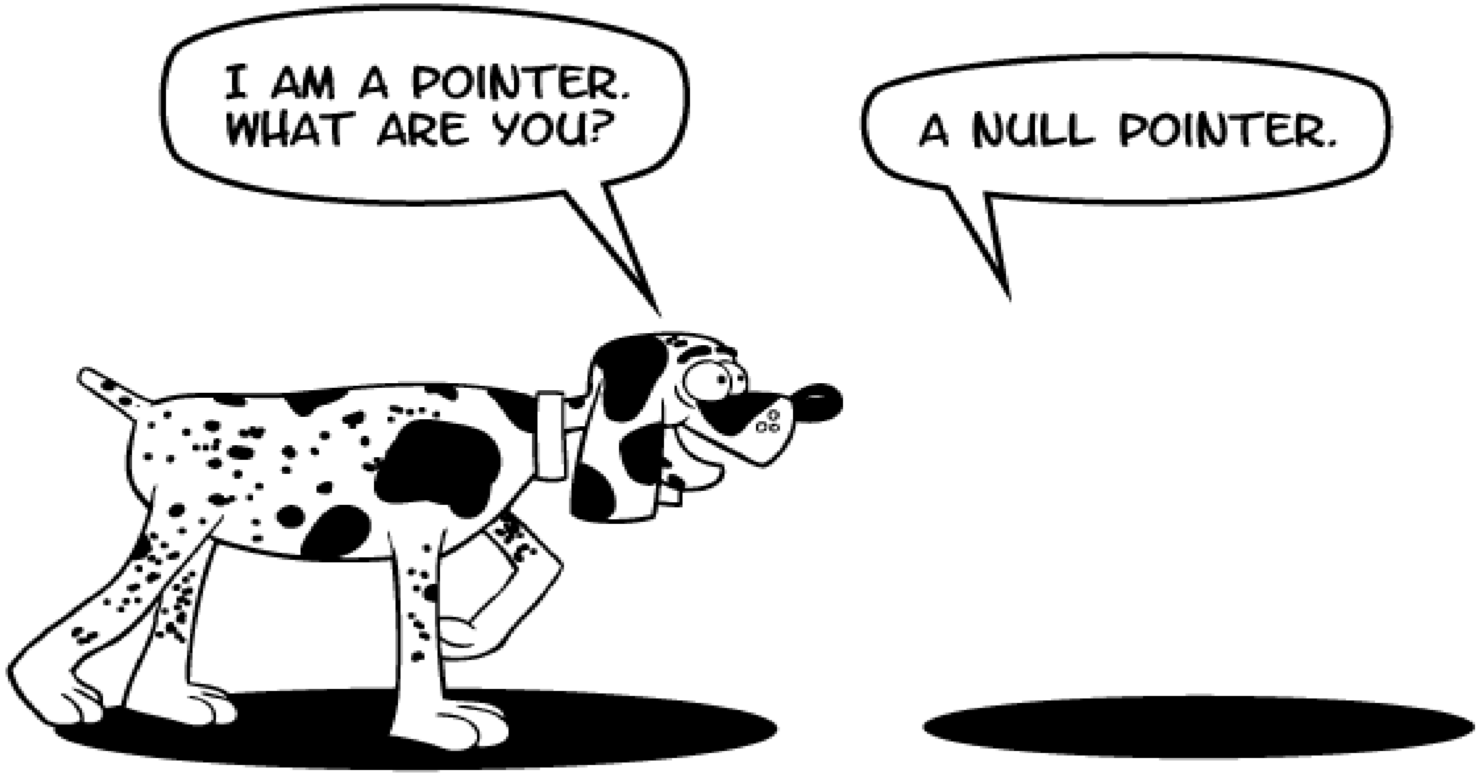
```
int a=50/0;//ArithmeticException
```

Scenario where NullPointerException occurs



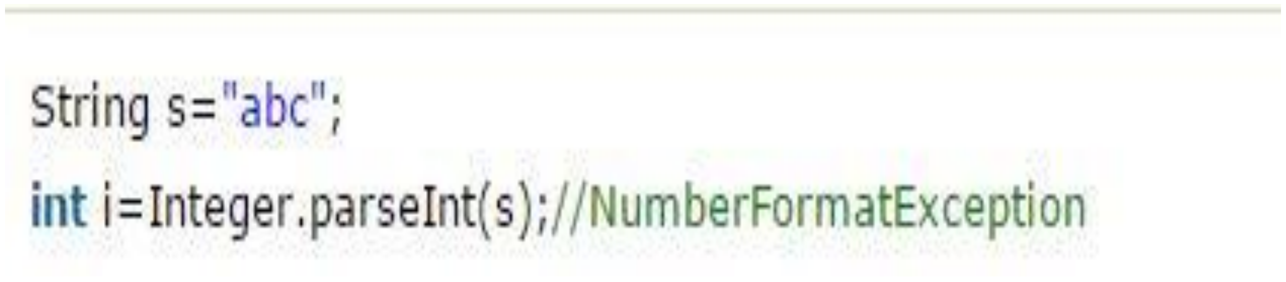
```
String s=null;
System.out.println(s.length());//NullPointerException
```

NullPointerException



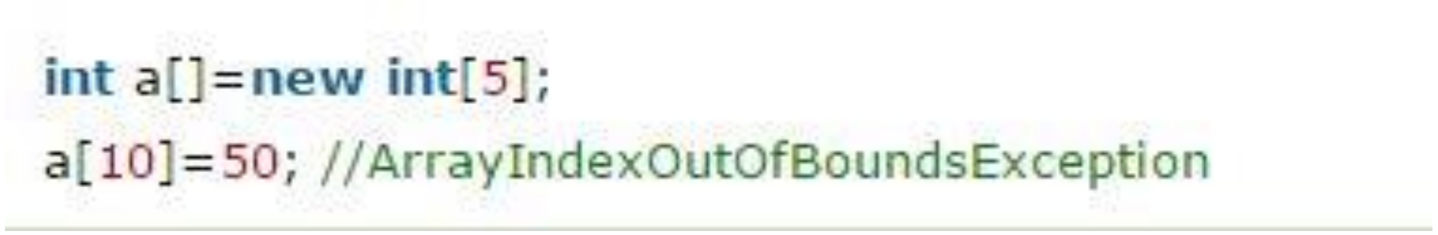
Common scenarios where exceptions may occur

Scenario where NumberFormatException occurs



```java
String s="abc";
int i=Integer.parseInt(s);//NumberFormatException
```

Scenario where ArrayIndexOutOfBoundsException occurs



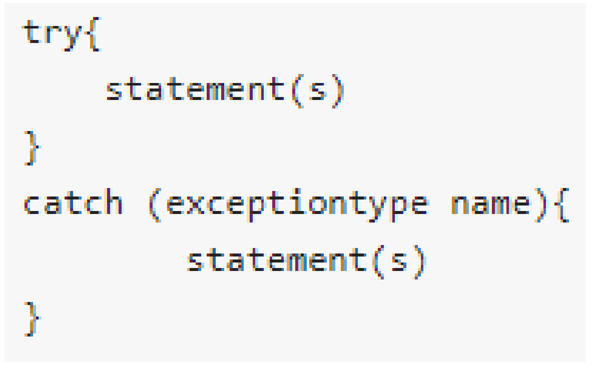
```java
int a[]=new int[5];
a[10]=50; //ArrayIndexOutOfBoundsException
```

Java Exception Handling Keywords

There are 5 keywords used in java exception handling.

- try
- catch
- finally
- throw
- throws

Syntax for using try & catch



```
try{
    statement(s)
}
catch (exceptiontype name){
        statement(s)
}
```

Example

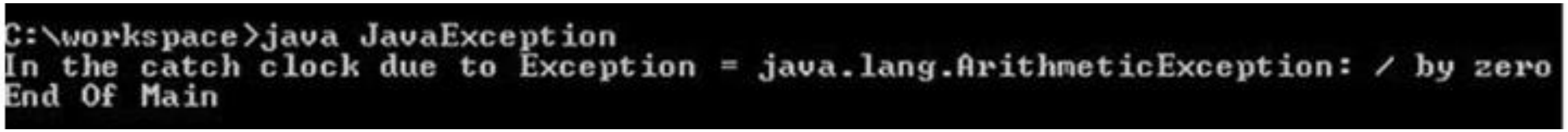
Step 1) Copy the following code into an editor

```
class JavaException {
   public static void main(String args[]){
      int d = 0;
      int n = 20;
      int fraction = n/d;
     System.out.println("End Of Main");
   }
}
```

- **Step 2)** Save the file & compile the code. Run the program using command, java JavaException
- **Step 3)** An Arithmetic Exception - divide by zero is shown as below for line # 5 and line # 6 is never executed
- **Step 4)** Now let's see examine how try and catch will help us to handle this exception. We will put the exception causing the line of code into a **try** block, followed by a **catch** block. Copy the following code into the editor.

```
1.  class JavaException {
2.   public static void main(String args[]) {
3.    int d = 0;
4.    int n = 20;
5.    try {
6.     int fraction = n / d;
7.     System.out.println("This line will not be Executed");
8.    } catch (ArithmeticException e) {
9.     System.out.println("In the catch Block due to Exception = " + e);
10.   }
11.   System.out.println("End Of Main");
12.  }
13. }
```

Step 5) Save, Compile & Run the code.You will get the following output



```
C:\workspace>java JavaException
In the catch clock due to Exception = java.lang.ArithmeticException: / by zero
End Of Main
```

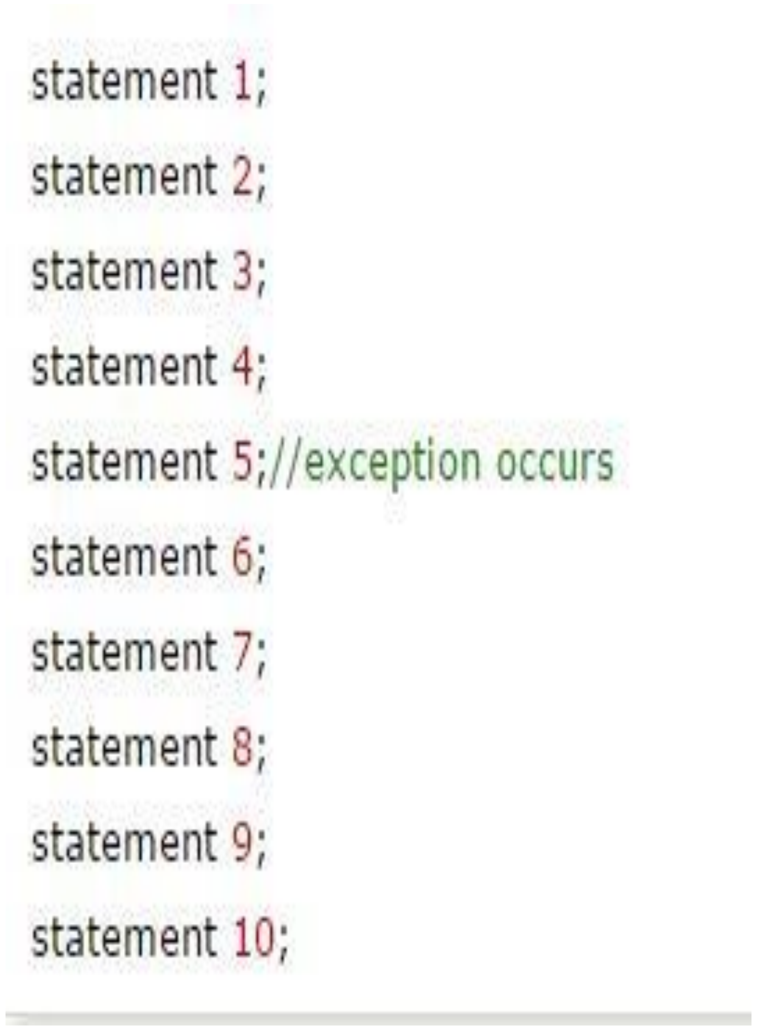
- As you observe, the exception is handled, and the last line of code is also executed. Also, note that Line #7 will not be executed because **as soon as an exception is raised the flow of control jumps to the catch block.**
- **Note:** The AritmeticException Object "e" carries information about the exception that has occurred which can be useful in taking recovery actions.

Advantage of Exception Handling

The core advantage of exception handling is **to maintain the normal flow of the application**.

Let's take a scenario:

Suppose there is 10 statements in your program and there occurs an exception at statement 5, rest of the code will not be executed i.e. statement 6 to 10 will not run. If we perform exception handling, rest of the statement will be executed. That is why we use exception handling in java.



```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5;//exception occurs
statement 6;
statement 7;
statement 8;
statement 9;
statement 10;
```