Topic 4.1: Strings & File I/O

Learning Goals (Week 4):

- **Uses, throw, and catch built-in exceptions.**
- **Understand and use Finally block when handling exceptions.**
- **Order of operations in a try/catch/finally block**
- Write code that can read and write text files
- Write code that uses data from a file to instantiate objects
- trim() & split() functions
- charAt() & substring()
- Additional helpful String manipulation expressions

Exceptions

- Whenever anything goes wrong in Java, an Exception is generated:
- Maybe you’ve seen some of these
    - 5/0 gives an **ArithmeticException**
    - array[-1] = 0 gives an **ArrayIndexOutOfBoundsException**
    - Integer.parseInt(“Not a number") gives a **NumberFormatException**
    - Person x; x.age = 4; gives a **NullPointerException**

Exceptions

- Whenever anything goes wrong in Java, an Exception is generated:
- Maybe you've seen some of these
    - 5/0 gives an **ArithmeticException**
    - array[-1] = 0 gives an **ArrayIndexOutOfBoundsException**
    - Integer.parseInt("Not a number") gives a **NumberFormatException**
    - Person x; x.age = 4; gives a **NullPointerException**

- Each of these **Exceptions** are actually **Objects**! (Each of those exceptions is a **data type** just like String, or our Person)
    - They have attributes and methods like any object
    - These attributes/methods give us the information we need to figure out what went wrong, where, and sometimes why?
    - You can even define your own (more on this next year!)
- You can even make variables of these types!

```
ArithmeticException ae = new ArithmeticException();
```

Exception Methods

- Remember how there is an Object umbrella class with methods like toString()?
- There is an Exception umbrella class too! This umbrella class gives all these different exceptions access to certain methods:
    - String getMessage()
        - Find out more about what happened
    - void printStackTrace()
        - Print out where the program was at the time this Exception object was created
    - String toString()
        - Yes, just like the Object one (Exceptions actually fall under the Object umbrella too)
        - The name of the exception and its message
- **Exceptions have constructors too!**

Exception Methods

```
Exception oops = new Exception("You really messed up!");
```

- The “stack trace” is set automatically
  - i.e. all the information you see pour out when an exception in your code goes off

Exceptions vs Errors

- There’s another class of objects called “**Error**” in Java

- As opposed to Exceptions, Errors are more serious problems that a program is not supposed to manage / handle
  - when it happens, the program will terminate with a description of the error

- Example:
  - OutOfMemoryError (when the virtual machine is out of memory)
  - StackOverflowError (we will undoubtedly see this in a few weeks time)

More Exception-al information

- Even though they are variables
  - we don’t use them the same way:
  - we don’t pass them around
  - we don’t ‘return’ them
  - We want our code to be able to return it’s normal values **AND** tell us if something goes wrong
- While our code attempts to execute normally it might ***“throw”*** an exception

Throwing Exceptions

- So when something nasty happens, you can “throw” an exception, using the throw keyword:

```
if(something nasty happens) {
     throw new Exception("AHHH RUN AWAY!");
}
```

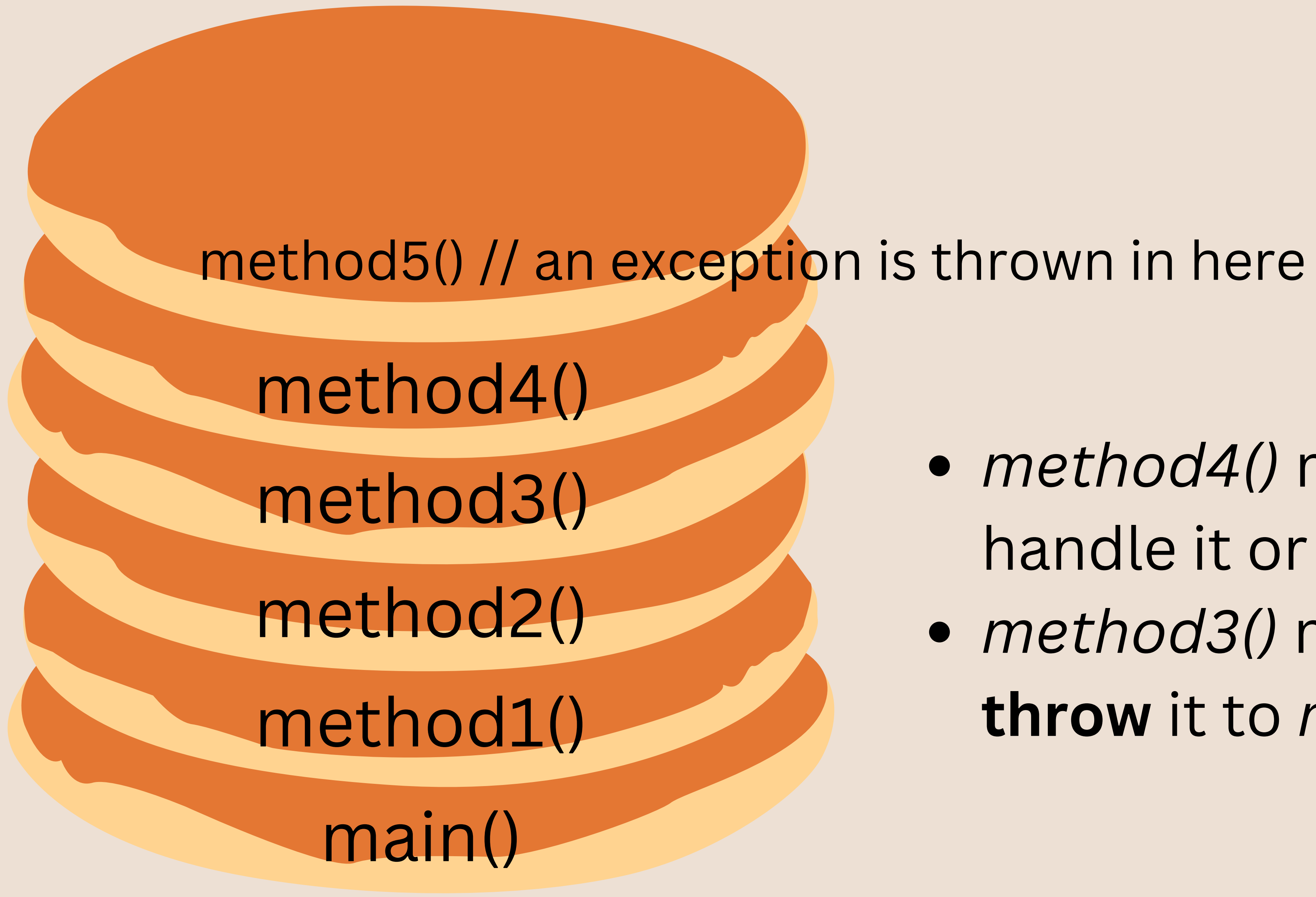
- This object is now automatically thrown back up the “stack” or “call chain” – immediately terminating all of the methods/records it hits as it goes.
    - Until it hits something that “catches” it…

Throwing Exceptions

- All methods that either throw an exception or call a method that throws an exception have this statement at the end of the signature:
    - throws Exception //could be a sub-type of Exception
- This is necessary, unless the method itself “catches” the exception thrown by the callee

A reminder on the runtime stack

- All methods that have been called but not yet completed their execution are on that stack.

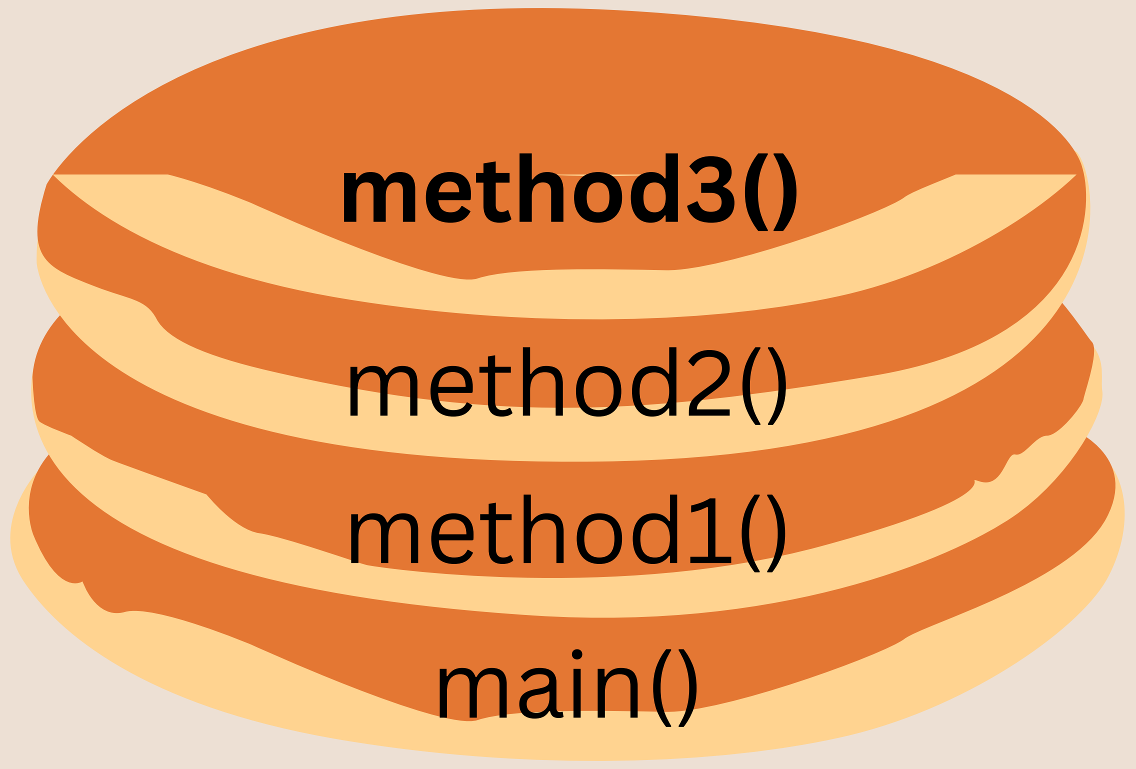


- *method4()* must either **catch** the exception and handle it or **throw** it to *method3()* and then
- *method3()* must **catch** the exception and handle it or **throw** it to *method2()* and so on...

A reminder on the runtime stack

- **Exceptions can be caught and handled without terminating the program**

method5() // an exception was thrown in here and we exit the method
method4() // threw exception and exit the method



- *method4()* **throws** the exception to *method3()* and then *method3()* **catches** and **handles** the exception and the program can continue from there

How can we handle exceptions and keep going?

```
try {
  //  ...statements...
}
catch(TypeOfException e) {
  //  ...do this if one of those Exceptions happens...
}
// …continue here…
// …(unless some un-caught exception occurs)…
```

How can we handle exceptions and keep going?

- You can choose to catch **only** one specific kind of Exception:

```
try {
  // ...statements...
}
catch(ArithmeticException ae) { .... }
//Other kinds will not be caught at all
```

- Or handle different types in different ways:

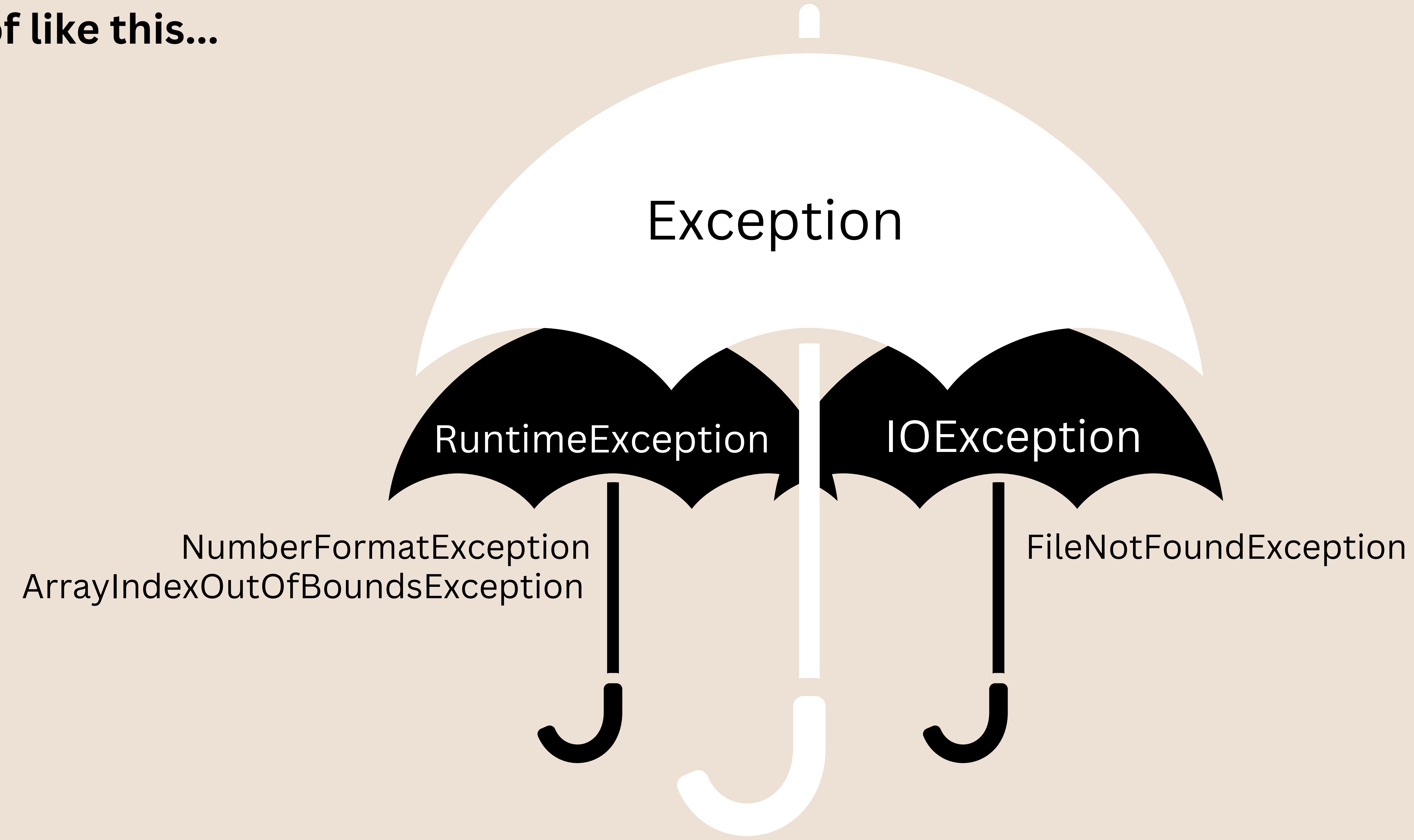
```
try {
    //...statements...
} catch(ArithmeticException ae) {
    // ..handle one way..
} catch(IOException ioe) {
    //..handle some other way..
}
```

Can have as many "catch" blocks as you want.
When you have multiple catch blocks (as in the above example), only one catch block will handle the exception à the first one that is of the corresponding type (exception type matches thrown exception)

The Exception Umbrella

Remember how I mentioned the Object umbrella? And then the Exception one underneath?

Sort of like this...



The Exception Umbrella

Handling a type of Exception will also cover all other types beneath it in the hierarchy (umbrella)

```
try {
    // ...statements...
} catch(Exception e) {
    // ...will catch absolutely everything...
}
```

Exception is at the top of the hierarchy (umbrella) of exceptions, so it will always catch any type of exception thrown.

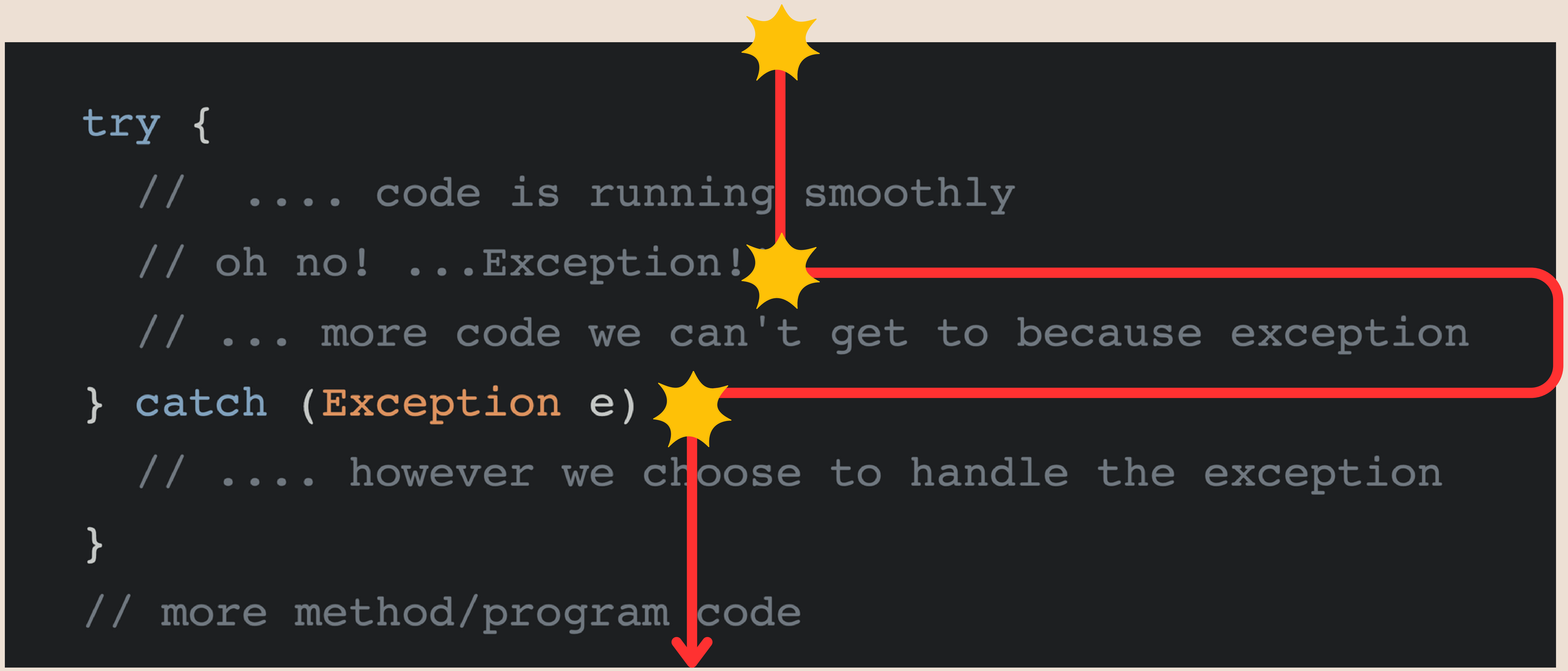
This means that if we added more specific catch blocks below it (with more specific types of exceptions), they would never execute!

Exception Flow: Try & Succeed!

When no exception is thrown in the try block, the program skips over the catch blocks and continues the execution of the code below the try-catch

Just like an if/else statement

Exception Flow: Try & Exception



```
try {
   //  .... code is running smoothly
   // oh no! ...Exception!
   // ... more code we can't get to because exception
} catch (Exception e)
   // .... however we choose to handle the exception
}
// more method/program code
```

Exception Flow: Finally

You can add a **finally** block at the end of your try-catch & this block is always executed, no matter what happens

```
try {
   // ...statements...
} catch(Exception e) {
        // ...will catch absolutely everything...
} finally{
        // … do this every time, whether try successfully completed or an exception was thrown and caught…
}
```

Some Exceptions MUST be handled (but not all)

- You have probably noticed that some exceptions are popping up from time to time when you run your code

- You have probably seen by now:
  - NullPointerException
  - IndexOutOfBoundException
  - IllegalArgumentException
  - ArithmeticException

- Those exceptions fall under the **RuntimeException** umbrella (hierarchy)

- RuntimeExceptions are **unchecked**:
  - You don't need to catch them

- You don't need to declare in your method signature that it could throw them

- **RuntimeExceptions are thrown** when the programmer makes a mistake and calls a method incorrectly (NullPointerException is a good example of that...)
- We ***usually do not*** want to catch these

Some Exceptions MUST be handled (but not all)

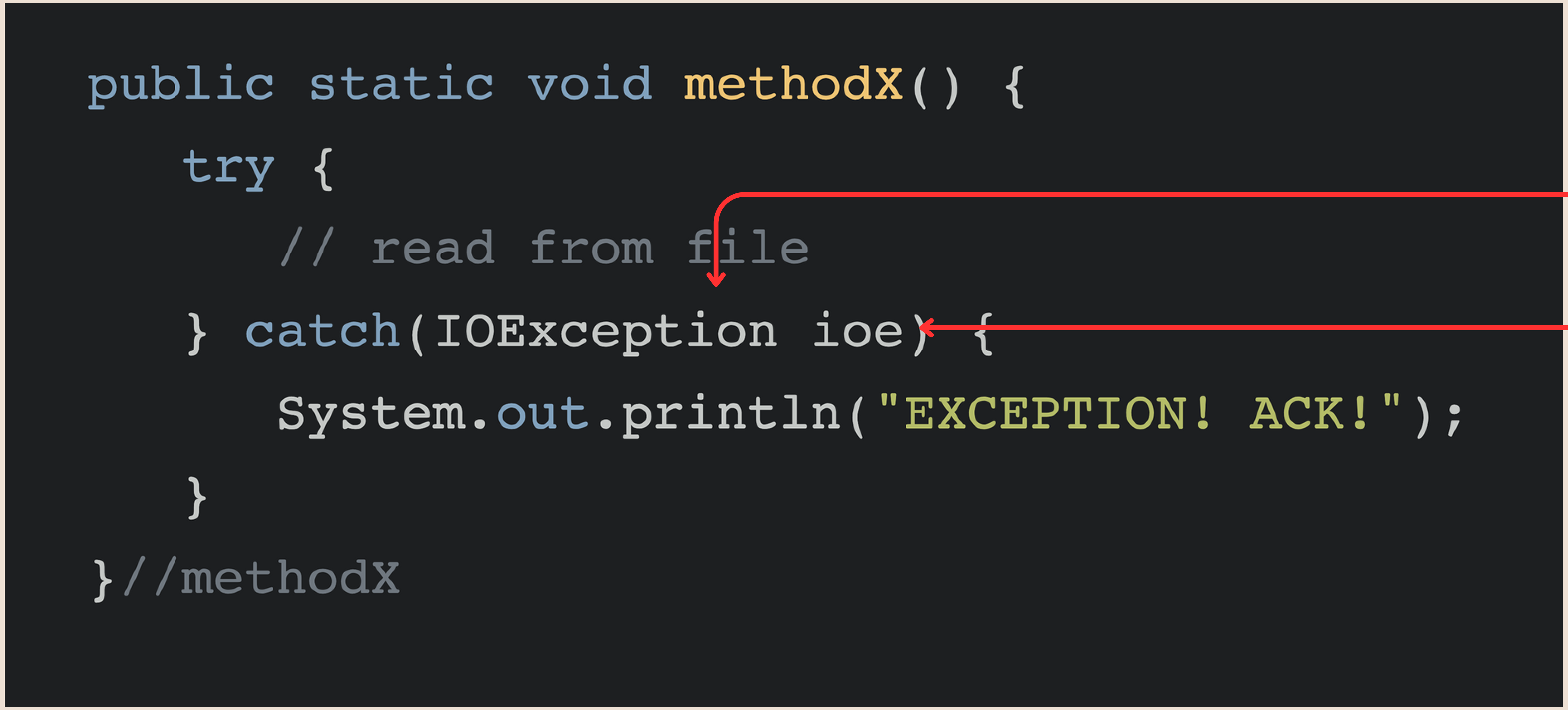
- **Checked** Exceptions must be handled
- Everything under Exception (in the hierarchy) that is **not** a **RuntimeException** is a checked Exception

- Most IO exceptions are of this type (as we will very very soon see!)
  - Checked Exceptions must either be:
  - Caught by a try-catch block, or
  - Thrown from this method to the calling method (by declaring “throws ExceptionType” in the signature)

```java
public static void main(String[] args) {
    // read from a file
}
```

This is not allowed and the Java compiler will complain. If we are doing File I/O, we need to **explicitly** tell Java how we want to handle the possibility of exceptions

Some Exceptions MUST be handled (but not all)



```
public static void methodX() {
    try {
        // read from file
    } catch(IOException ioe) {
        System.out.println("EXCEPTION! ACK!");
    }
}//methodX
```

Catches all the IOExceptions

This is the Exception variable with the deets

Solution 1: **CATCH IT**

Some Exceptions MUST be handled (but not all)

```
public static void methodX() throws IOException {
      // read from file
}//methodX
```

Solution 2: **THROW IT**

- If any method call or action inside this method causes an IOException of any kind, it's thrown back to the method that called this one.
- This method is immediately terminated at that point.
- Now any other method that uses methodX must also catch or throw IOExceptions.

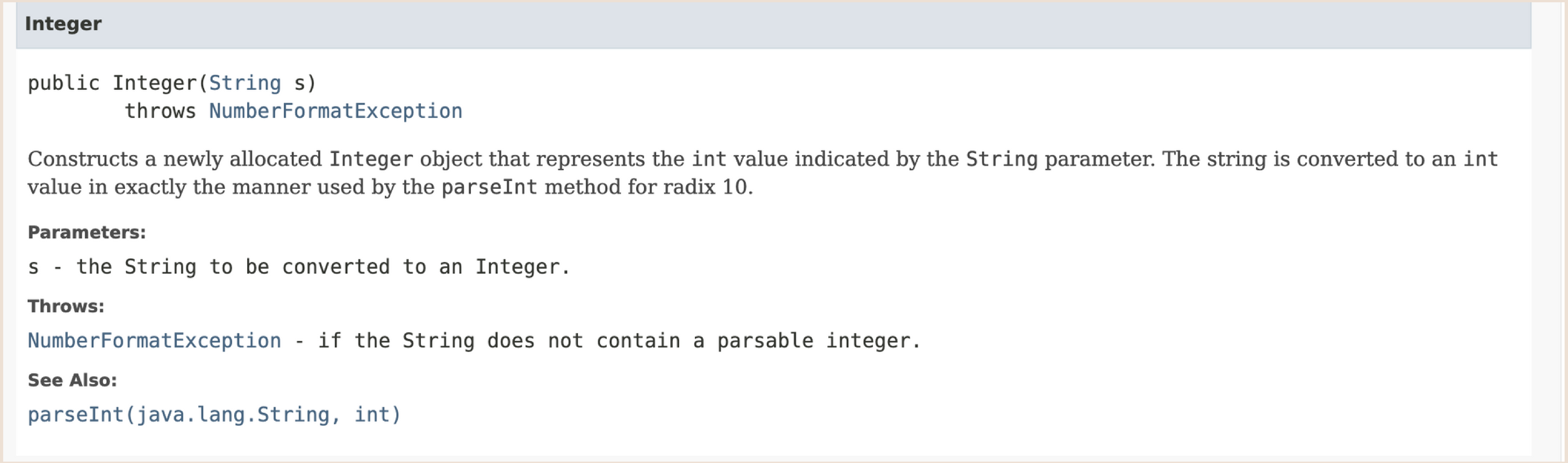
Returning from these methods

- If nothing goes wrong, we return like normal! (If a return type is specified)
- If something goes wrong, the exception gets thrown and since we aren’t catching it, we don’t return anything from here, we just throw the Exception up the call chain

- You have to handle checked exceptions when using a method that throws a checked exception
- So… how do you know if a method throws something?
  - The compiler will get mad if you try to use something that may throw a checked exception and requires a try/catch (if you don’t provide one)
- If you don’t know what throws what and you want to:
  - ORACLE:
    - E.G (Integer Class) https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html

What throws what?

- If you don't know what throws what and you want to:
  - ORACLE:
    - E.G (Integer Class) https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html



Integer

```
public Integer(String s)
        throws NumberFormatException
```

Constructs a newly allocated `Integer` object that represents the `int` value indicated by the `String` parameter. The string is converted to an `int` value in exactly the manner used by the `parseInt` method for radix 10.

Parameters:

`s - the String to be converted to an Integer.`

Throws:

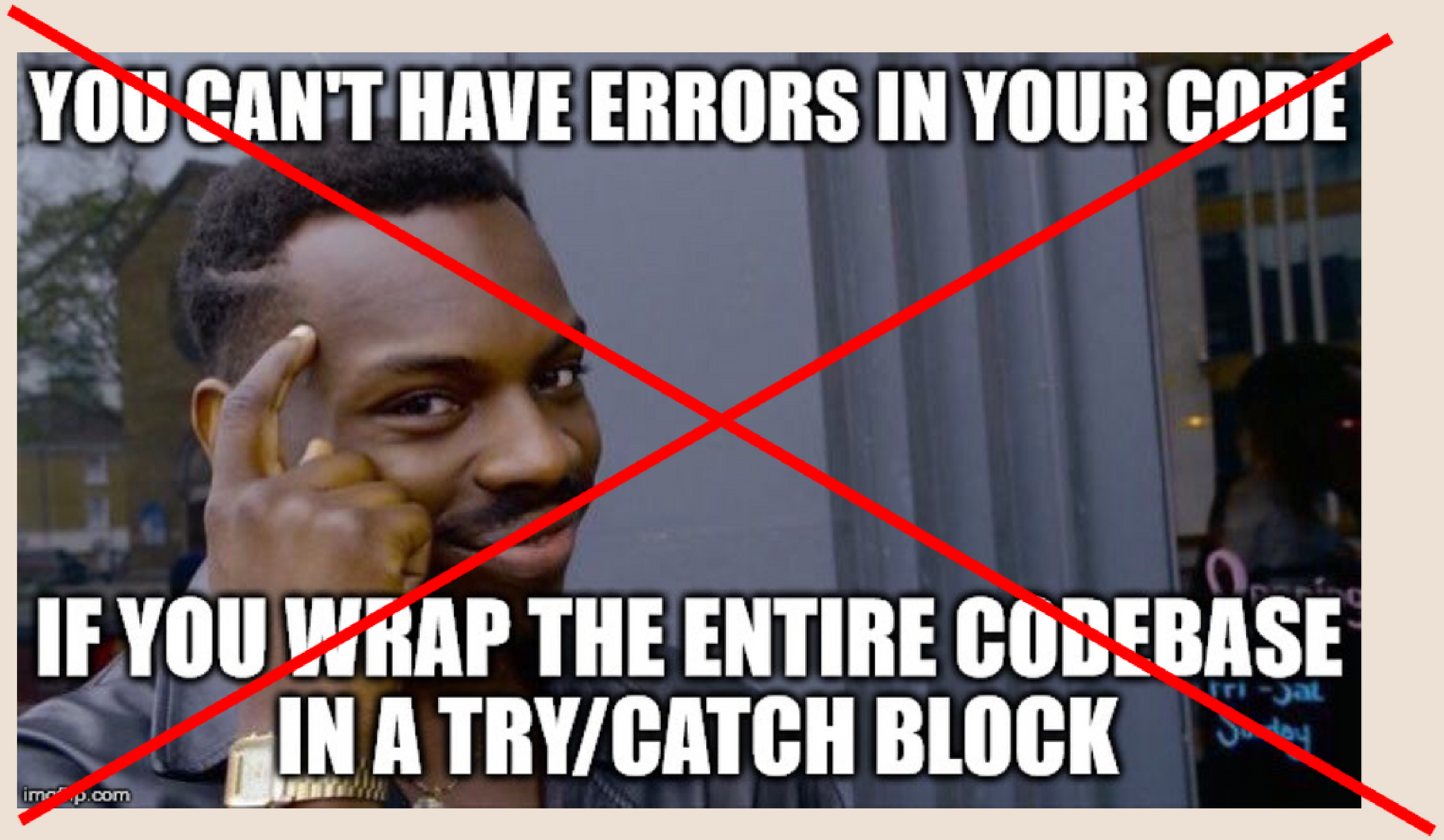
`NumberFormatException - if the String does not contain a parsable integer.`

See Also:

`parseInt(java.lang.String, int)`

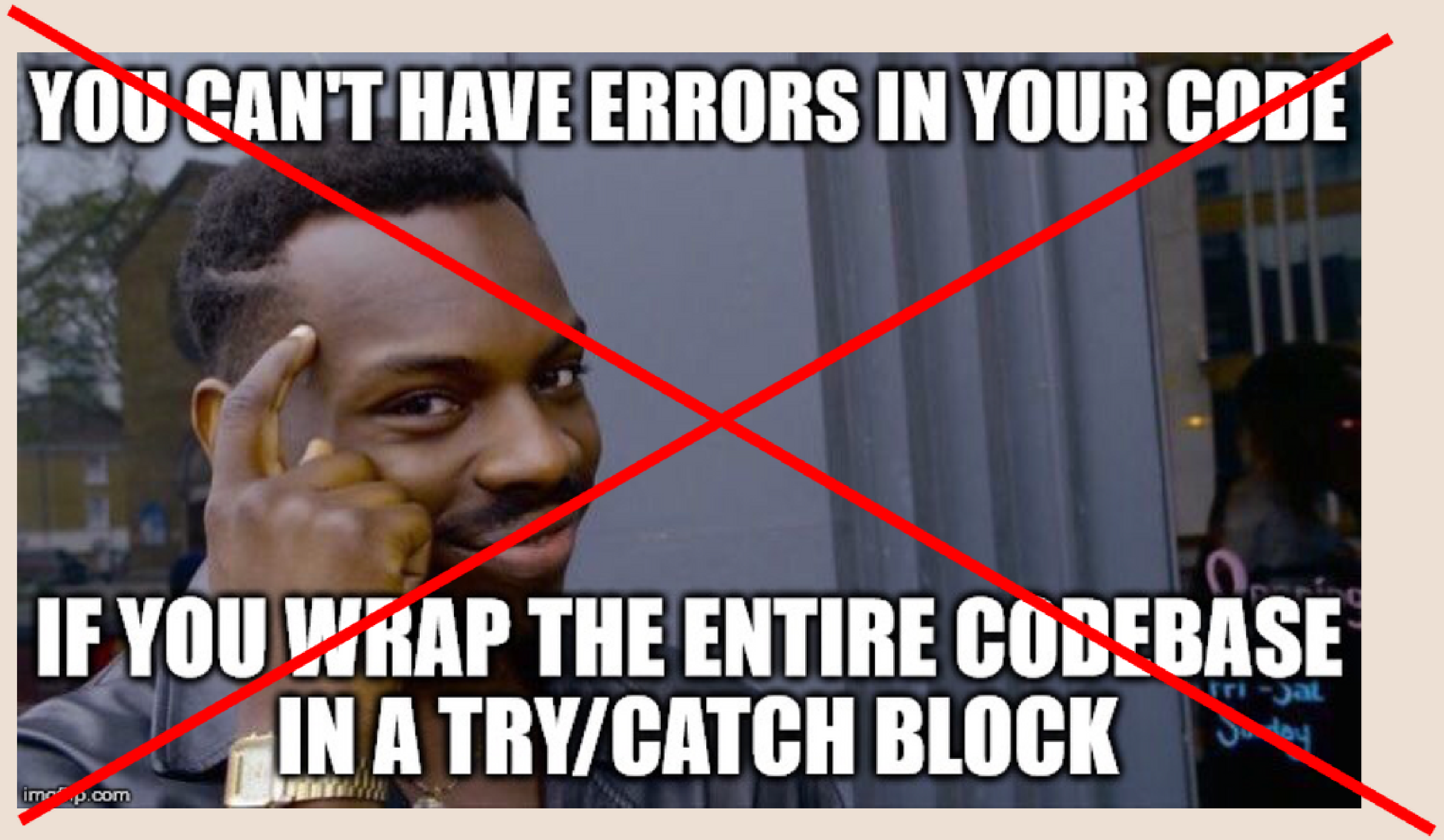
Final Notes on Responsible Exception Use

- DO NOT use as if/else statements
- DO NOT use to catch RunTime Exceptions (unless you have a very good reason to)
    - don’t be lazy and make the appropriate checks instead of using try-catch for those
    - check if your reference is null before calling a method on it, instead of catching the NullPointerException


YOU CAN'T HAVE ERRORS IN YOUR CODE
IF YOU WRAP THE ENTIRE CODEBASE IN A TRY/CATCH BLOCK
imgflip.com

Final Notes on Responsible Exception Use

- DO NOT use as if/else statements
- DO NOT use to catch RunTime Exceptions (unless you have a very good reason to)
  - don't be lazy and make the appropriate checks instead of using try-catch for those
  - check if your reference is null before calling a method on it, instead of catching the NullPointerException


YOU CAN'T HAVE ERRORS IN YOUR CODE
IF YOU WRAP THE ENTIRE CODEBASE IN A TRY/CATCH BLOCK
imgflip.com

- These Exceptions are just types of objects like any other.
  - Some are automatically available
  - Some need the correct "import" at the top
- Example: IOException is really java.io.IOException so
  - Use **import java.io.*;**
  - Or use **import java.io.IOException;**

Pause & Practice

- We will do LOTS of try/catch practice in 4.2, when we start File Reading/Writing
- For now:
  - try to make your code throw **runtime** exceptions
  - How many can you force your code to throw?
  - Write down the Exceptions and the code your wrote that made the exception get thrown
  - Consider ways you could avoid running into these exceptions ‘in the wild’
    - i.e what type of checks would you need to avoid these exceptions
- Need help getting started? Can you make your code throw:
  - NullPointerException
  - ArrayIndexOutOfBoundsException
  - NumberFormatException
  - InputMismatchException
  - ArithmaticException
  - NegativeArraySizeException