Object-Oriented Programming

Error Handling

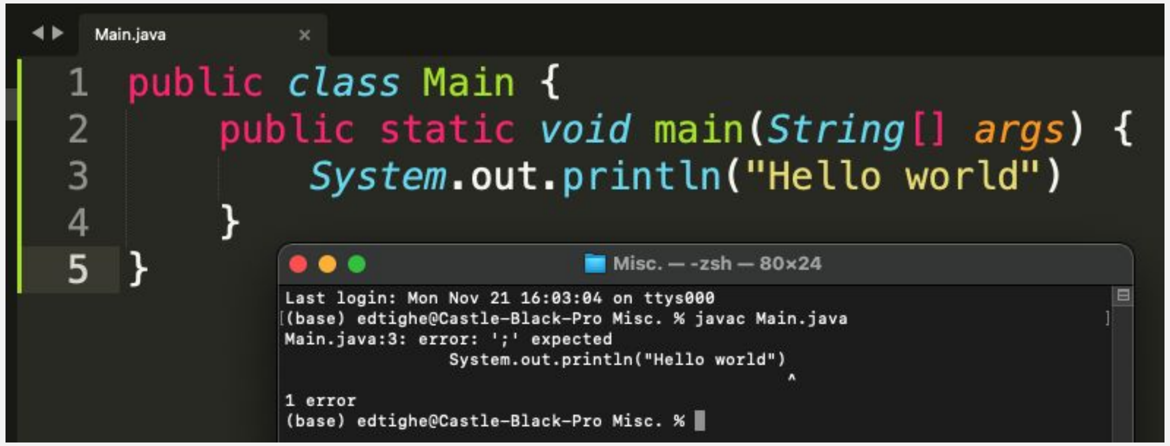
Outline

- Characteristics of Errors/Exceptions
- Handling Exceptions
    - Try/Catch Statement
    - Throw Keyword
- Creating your own Exceptions

What are the common errors that you've encountered so far?

Common Errors

Syntax error



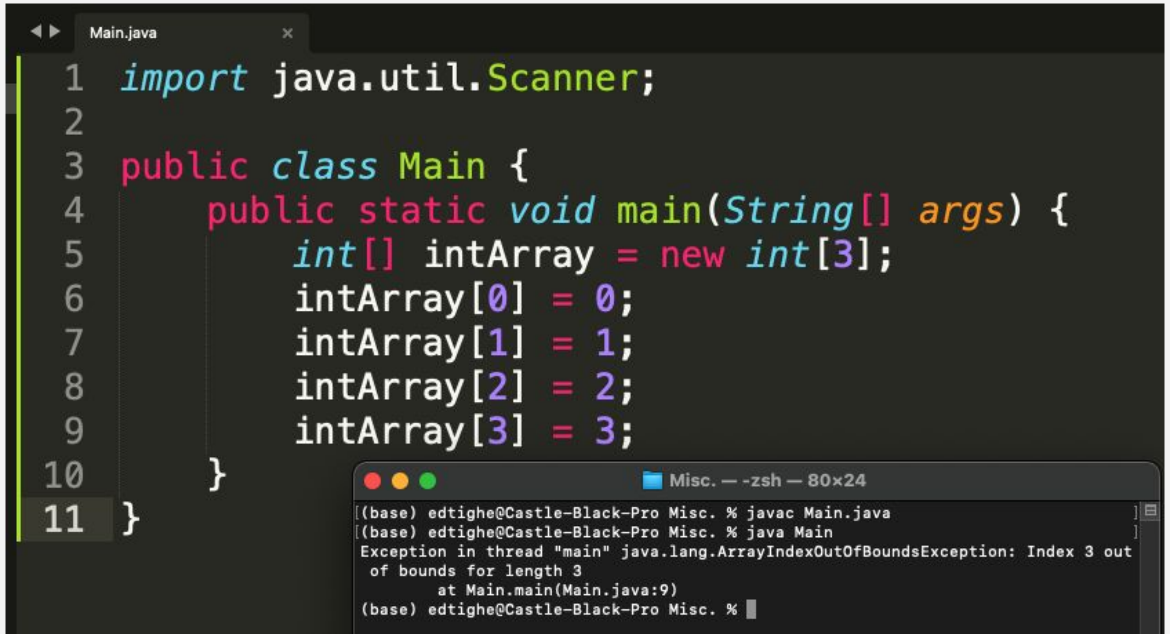
Main.java

```java
1 public class Main {
2     public static void main(String[] args) {
3         System.out.println("Hello world")
4     }
5 }
```

Misc. — -zsh — 80×24

```
Last login: Mon Nov 21 16:03:04 on ttys000
(base) edtighe@Castle-Black-Pro Misc. % javac Main.java
Main.java:3: error: ';' expected
                System.out.println("Hello world")
                                                 ^
1 error
(base) edtighe@Castle-Black-Pro Misc. %
```

Array index out of bounds exception



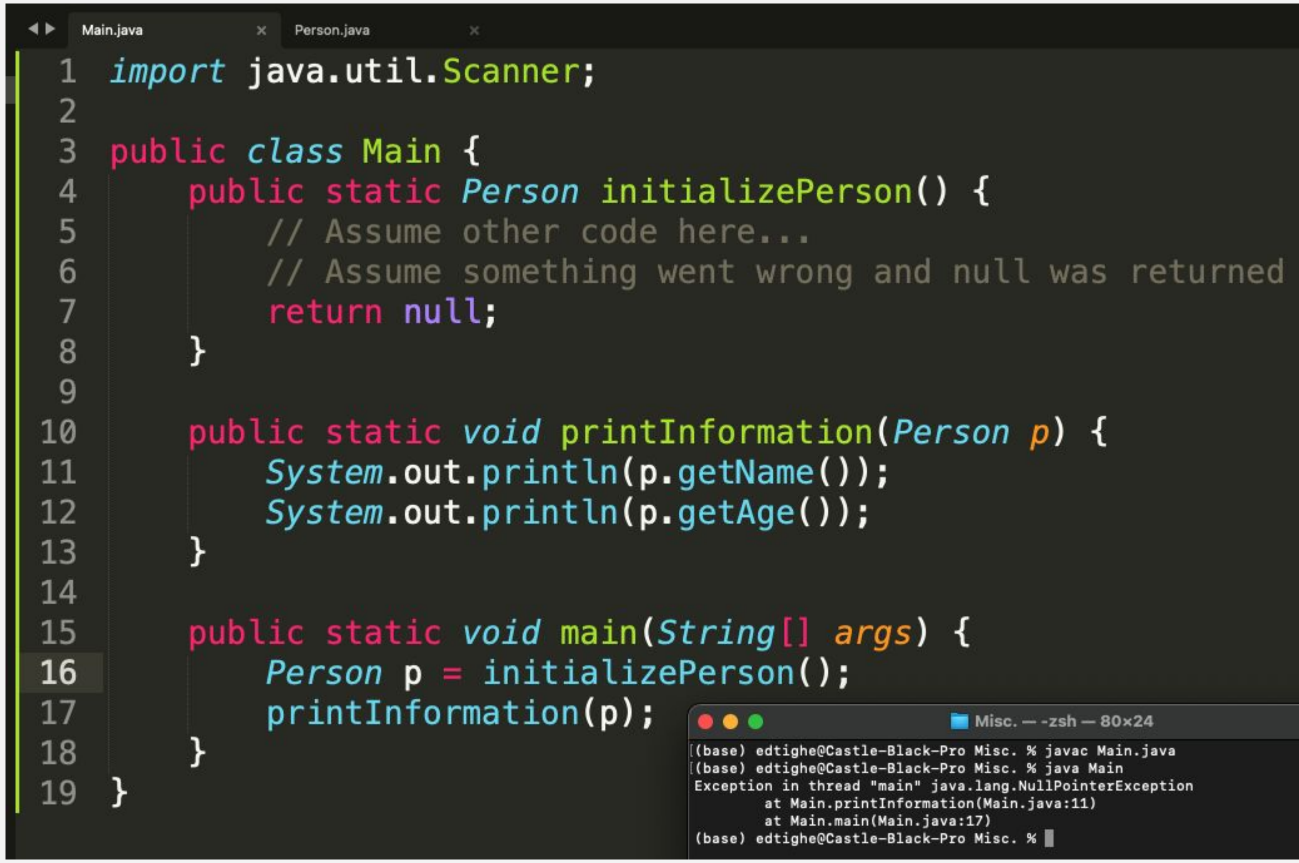
Main.java

```java
1  import java.util.Scanner;
2
3  public class Main {
4      public static void main(String[] args) {
5          int[] intArray = new int[3];
6          intArray[0] = 0;
7          intArray[1] = 1;
8          intArray[2] = 2;
9          intArray[3] = 3;
10     }
11 }
```

Misc. — -zsh — 80×24

```
(base) edtighe@Castle-Black-Pro Misc. % javac Main.java
(base) edtighe@Castle-Black-Pro Misc. % java Main
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 3 out
 of bounds for length 3
        at Main.main(Main.java:9)
(base) edtighe@Castle-Black-Pro Misc. %
```

Null pointer exception



Main.java   Person.java

```java
1  import java.util.Scanner;
2
3  public class Main {
4      public static Person initializePerson() {
5          // Assume other code here...
6          // Assume something went wrong and null was returned
7          return null;
8      }
9
10     public static void printInformation(Person p) {
11         System.out.println(p.getName());
12         System.out.println(p.getAge());
13     }
14
15     public static void main(String[] args) {
16         Person p = initializePerson();
17         printInformation(p);
18     }
19 }
```

Misc. — -zsh — 80×24

```
(base) edtighe@Castle-Black-Pro Misc. % javac Main.java
(base) edtighe@Castle-Black-Pro Misc. % java Main
Exception in thread "main" java.lang.NullPointerException
        at Main.printInformation(Main.java:11)
        at Main.main(Main.java:17)
(base) edtighe@Castle-Black-Pro Misc. %
```

How can we characterize these errors?

What can you observe from these different errors?

Questions to think about…

- What makes an error different from an exception?
- When do errors take place?
  - During compilation?
  - During run time?

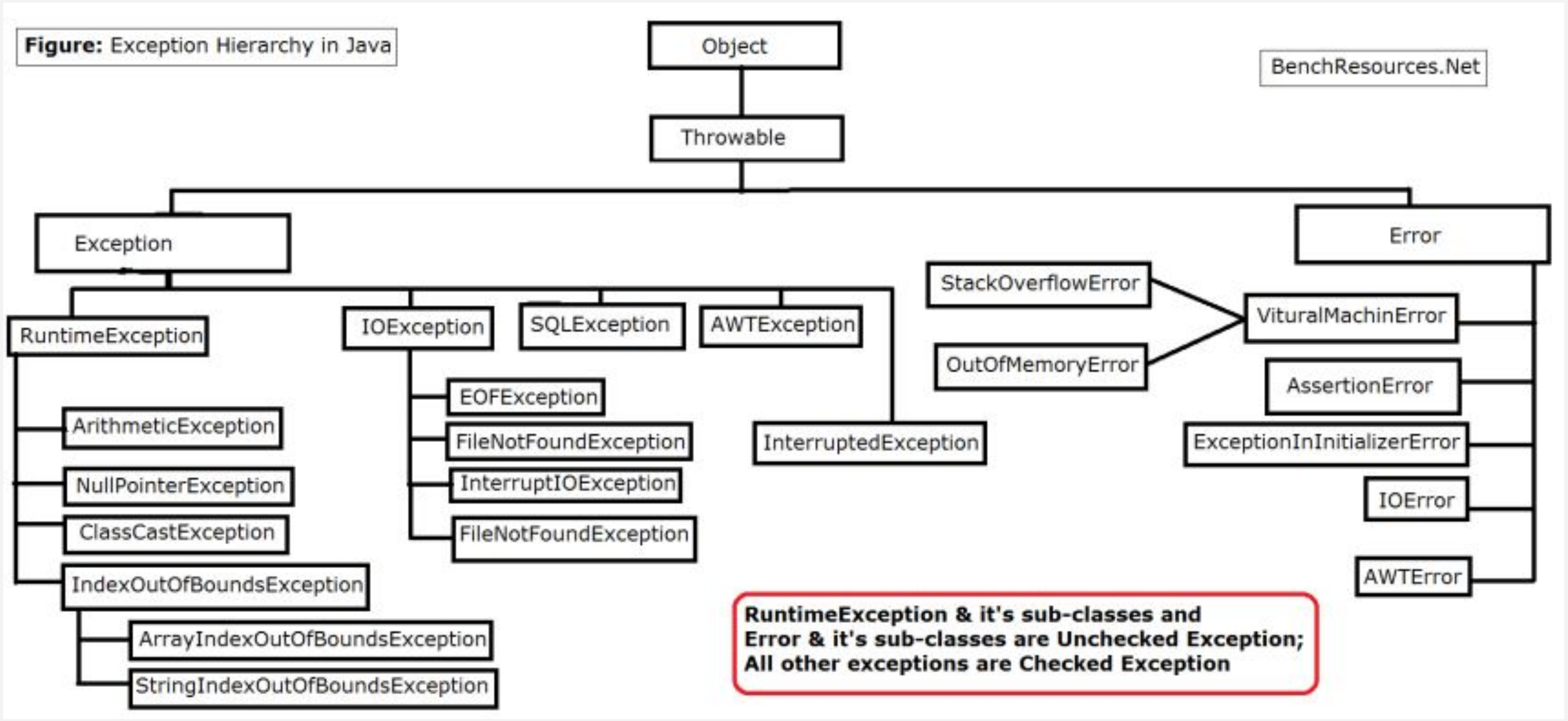
Errors and Exceptions

- In Java, both are implemented as classes
  - Error: https://docs.oracle.com/javase/7/docs/api/java/lang/Error.html
  - Exception: https://docs.oracle.com/javase/7/docs/api/java/lang/Exception.html
- Both extend the class Throwable
  - https://docs.oracle.com/javase/7/docs/api/java/lang/Throwable.html
  - "…contains a snapshot of the execution stack of its thread at the time it was created [+ error message]."

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 
Index 3 out of bounds for length 3
   at Main.main(Main.java:9)
```

Errors and Exceptions

- Errors and exceptions are “thrown” when the compiler or program cannot handle the situation, e.g.:
    - Syntax error: *Cannot understand what to do…*
    - Null pointer exception: *No instance to interact with…*
- There are also many types of errors and exceptions…



Source: https://www.benchresources.net/exception-hierarchy-in-java/

Errors

- Are very serious and should not be caught or handled…

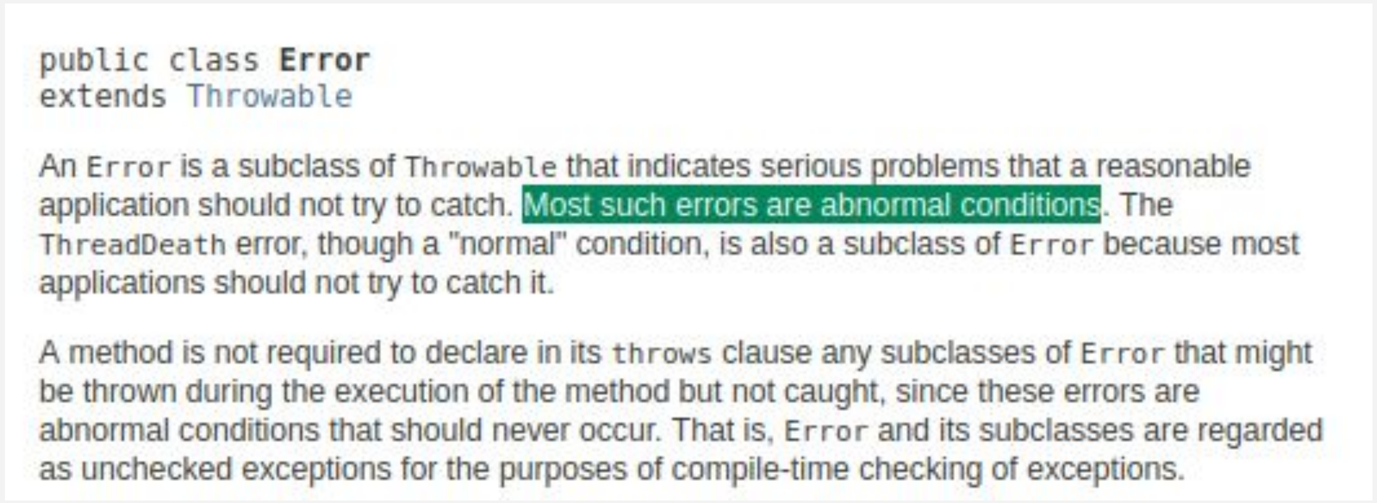
Object-Oriented Programming

Maybe we should update...

~~Error~~Exception Handling

Errors

- Are very serious and should not be caught or handled…
  - These are critical errors (like a memory stack overflow) where the program **should** most likely not continue
  - "should" is used because you can still catch an error, but it is not advised (check Javadoc)



```
public class Error
extends Throwable
```

An `Error` is a subclass of `Throwable` that indicates serious problems that a reasonable application should not try to catch. Most such errors are abnormal conditions. The `ThreadDeath` error, though a "normal" condition, is also a subclass of `Error` because most applications should not try to catch it.

A method is not required to declare in its `throws` clause any subclasses of `Error` that might be thrown during the execution of the method but not caught, since these errors are abnormal conditions that should never occur. That is, `Error` and its subclasses are regarded as unchecked exceptions for the purposes of compile-time checking of exceptions.

Exceptions

- Are errors that are more reasonable to catch and handle
  - E.g. FileNotFoundException or when a file cannot be found should (more often) not cause a program to close vs an OutOfMemoryError
- There are two types to keep in mind:
  - Checked – compile-time; predictable *(FileNotFoundException)*
  - Unchecked – runtime; unanticipated *(IndexOutOfBound)*

Handling Exceptions (let's look at an example)

```
Scanner s = new Scanner(System.in);
System.out.print("Enter a number: ");
int number = s.nextInt();
System.out.println("Number: " + number);
System.out.println("End of program");
```

We're just getting a number and printing it

```
>> java Main
Enter a number: 5
Number: 5
End of program
>> java Main
Enter a number: a
Exception in thread "main" java.util.InputMismatchException
	at java.base/java.util.Scanner.throwFor(Scanner.java:939)
	at java.base/java.util.Scanner.next(Scanner.java:1594)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
	at Main.main(Main.java:7)
```

How can we catch this? …

Using the Try-Catch

```
Scanner s = new Scanner(System.in);
System.out.print("Enter a number: ");
try {
    int number = s.nextInt();
    System.out.println("Number: " + number);
} catch(InputMismatchException e) {
    System.out.println(e);
    e.printStackTrace();
}
System.out.println("End of program");
```

Try some code where you're expecting something might go wrong

If the code block fails, catch the exception and handle accordingly. Otherwise, this is skipped.

We can specify a specific exception we want to catch

If we want to be more general (we might not be sure what exception to throw), then we can specify the class **Exception** (the superclass of all exceptions)

Using the Try-Catch

```java
Scanner s = new Scanner(System.in);
System.out.print("Enter a number: ");
try {
    int number = s.nextInt();
    System.out.println("Number: " + number);
} catch(InputMismatchException e) {
    System.out.println(e);
    e.printStackTrace();
}
System.out.println("End of program");
```

When running the code...

```
>> java Main
Enter a number: e
java.util.InputMismatchException
java.util.InputMismatchException
        at java.base/java.util.Scanner.throwFor(Scanner.java:939)
        at java.base/java.util.Scanner.next(Scanner.java:1594)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
        at Main.main(Main.java:10)
End of program
```

Notice that the program continues

Handling Exceptions

- Usage of some classes or methods require exception handling

Here we are writing to a file…

```
public static void writeToFile() {
    BufferedWriter bw = new BufferedWriter(new FileWriter("myFile.txt"));
    bw.write("Test");
    bw.close();
}
```

If you run this as is…

```
>> javac Main.java
Main.java:5: error: unreported exception IOException; must be caught or declared to be thrown
   BufferedWriter bw = new BufferedWriter(new FileWriter("myFile.txt"));
                                          ^
Main.java:6: error: unreported exception IOException; must be caught or declared to be thrown
   bw.write("Test");
           ^
Main.java:7: error: unreported exception IOException; must be caught or declared to be thrown
   bw.close();
```

Source: https://rollbar.com/blog/how-to-use-the-throws-keyword-in-java-and-when-to-use-throw/#

Handling Exceptions

- First way to solve the problem is using a try-catch

```
public static void writeToFile() {
    try {
        BufferedWriter bw = new BufferedWriter(new FileWriter("myFile.txt"));
        bw.write("Test");
        bw.close();
    } catch(IOException e) {
        // handle exception
    }
}
```

Source: https://rollbar.com/blog/how-to-use-the-throws-keyword-in-java-and-when-to-use-throw/#

Handling Exceptions

- Another way is to declare the exception can be thrown using the `throws` keyword

Declare the exception to be thrown in the signature of the method

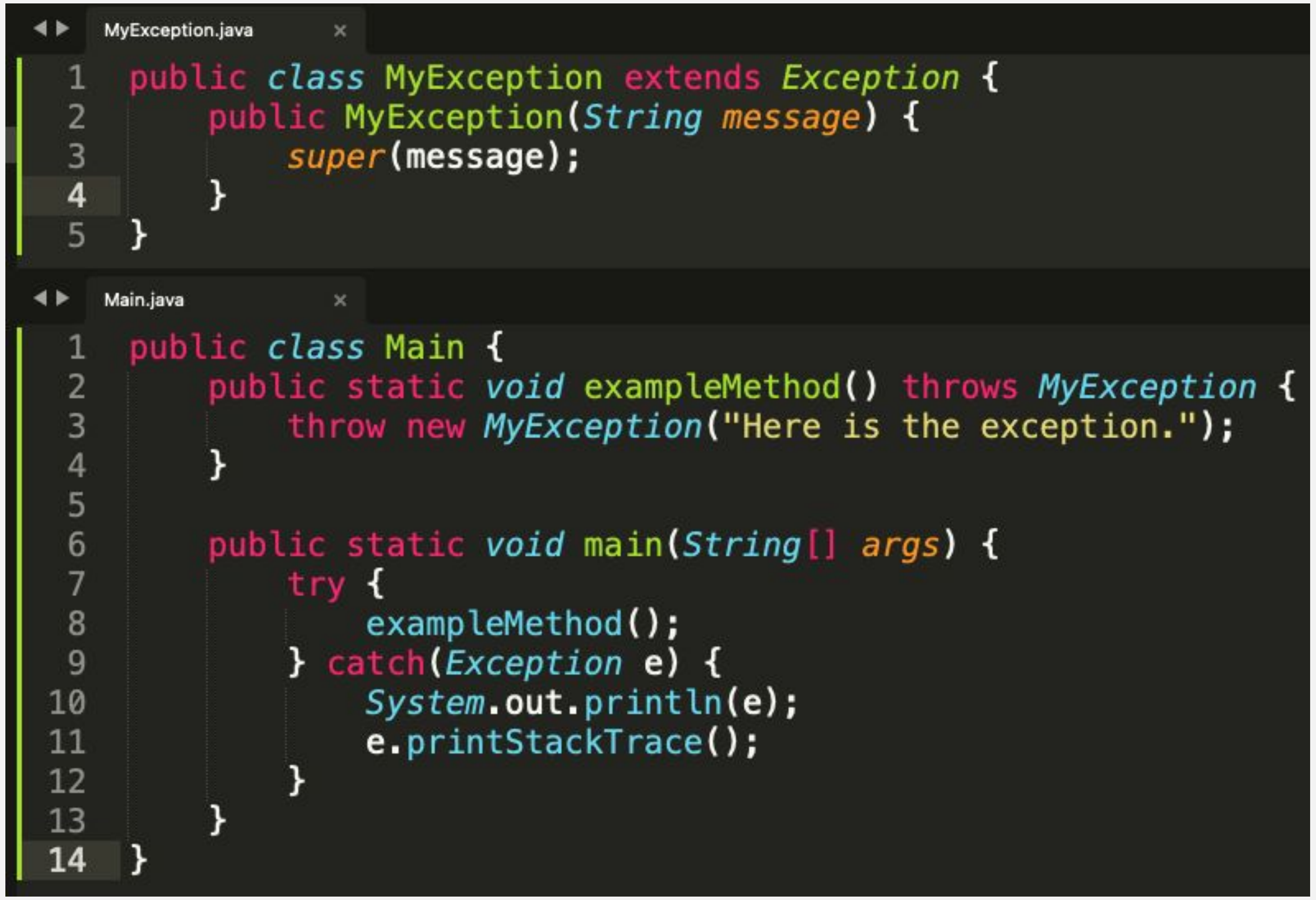
```java
public static void writeToFile() throws IOException {
    BufferedWriter bw = new BufferedWriter(new FileWriter("myFile.txt"));
    bw.write("Test");
    bw.close();
}
public static void main(String[] args) {
    try {
        writeToFile();
    } catch(IOException e) {
        // Handle it here
    }
}
```

However, there should still have an exception handling in place wherever this method is used

Source: https://rollbar.com/blog/how-to-use-the-throws-keyword-in-java-and-when-to-use-throw/#

Customized Exceptions

- If you have a specific example, you can also create and throw your own exception
- You can extend a specific exception class or extend from the general Exception class


MyException.java

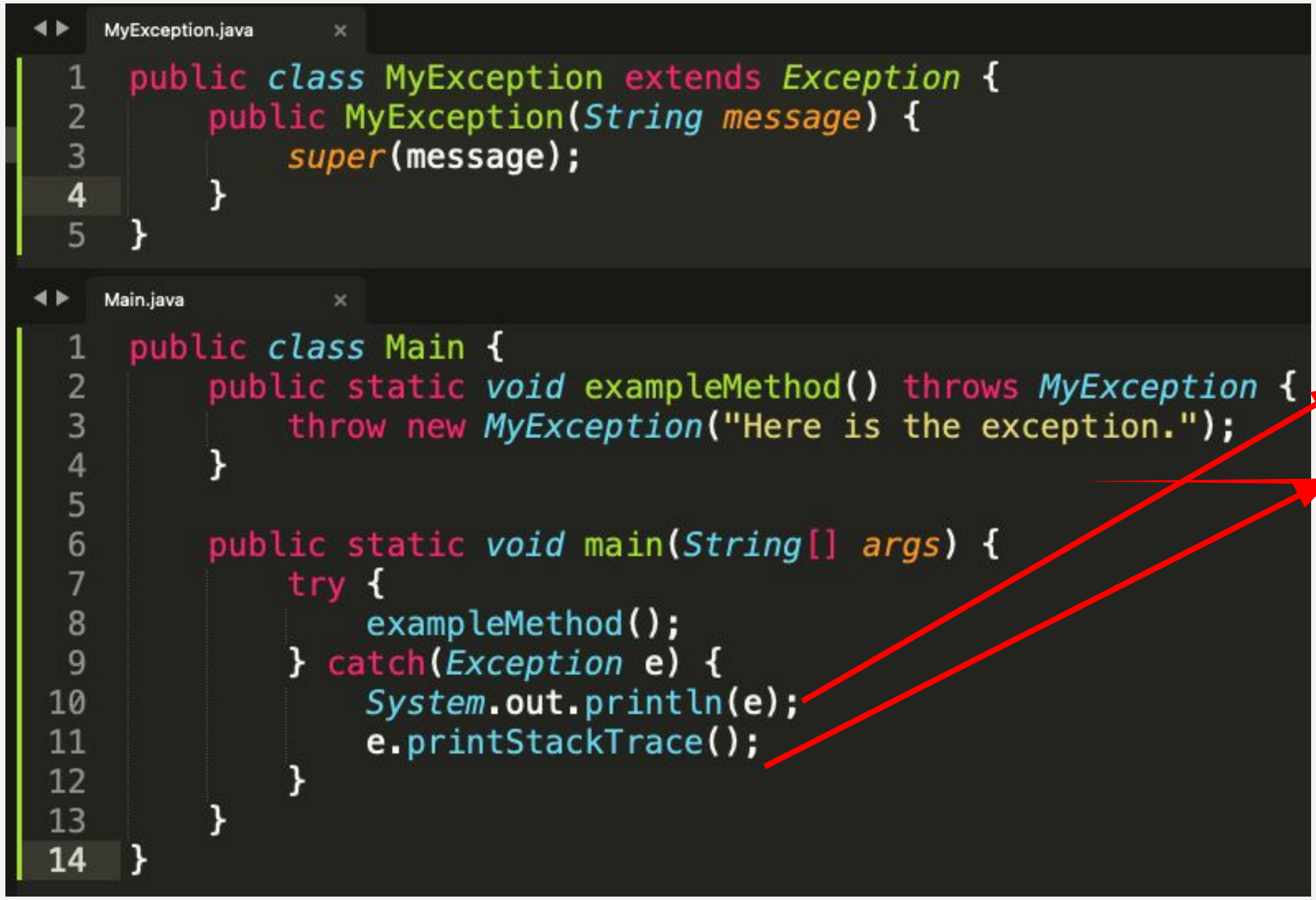
```java
public class MyException extends Exception {
    public MyException(String message) {
        super(message);
    }
}
```

Main.java

```java
public class Main {
    public static void exampleMethod() throws MyException {
        throw new MyException("Here is the exception.");
    }

    public static void main(String[] args) {
        try {
            exampleMethod();
        } catch(Exception e) {
            System.out.println(e);
            e.printStackTrace();
        }
    }
}
```

Customized Exceptions



```java
public class MyException extends Exception {
    public MyException(String message) {
        super(message);
    }
}
```

```java
public class Main {
    public static void exampleMethod() throws MyException {
        throw new MyException("Here is the exception.");
    }

    public static void main(String[] args) {
        try {
            exampleMethod();
        } catch(Exception e) {
            System.out.println(e);
            e.printStackTrace();
        }
    }
}
```

Output

```
>> java Main
MyException: Here is the exception.
MyException: Here is the exception.
        at Main.exampleMethod(Main.java:3)
        at Main.main(Main.java:8)
```

Questions?

Why Handle Exceptions?

- There are some errors which we can anticipate that should not cause a program to halt
- Exception handling allows programs to continue and provide feedback to the user to recover from the error
- Most of what was shown here is what the developer would see (via console); however, you can easily show error messages in layman's term in GUI components

Summary

- Errors should generally not be handled because they’re serious problems
- Exceptions are easier to deal with and can be thrown either at compile or runtime
- Exception handling involves the knowledge of existing Exception classes and usage of try-catch statement or throw keyword
    - You can also create your own Exceptions

Keep learning…