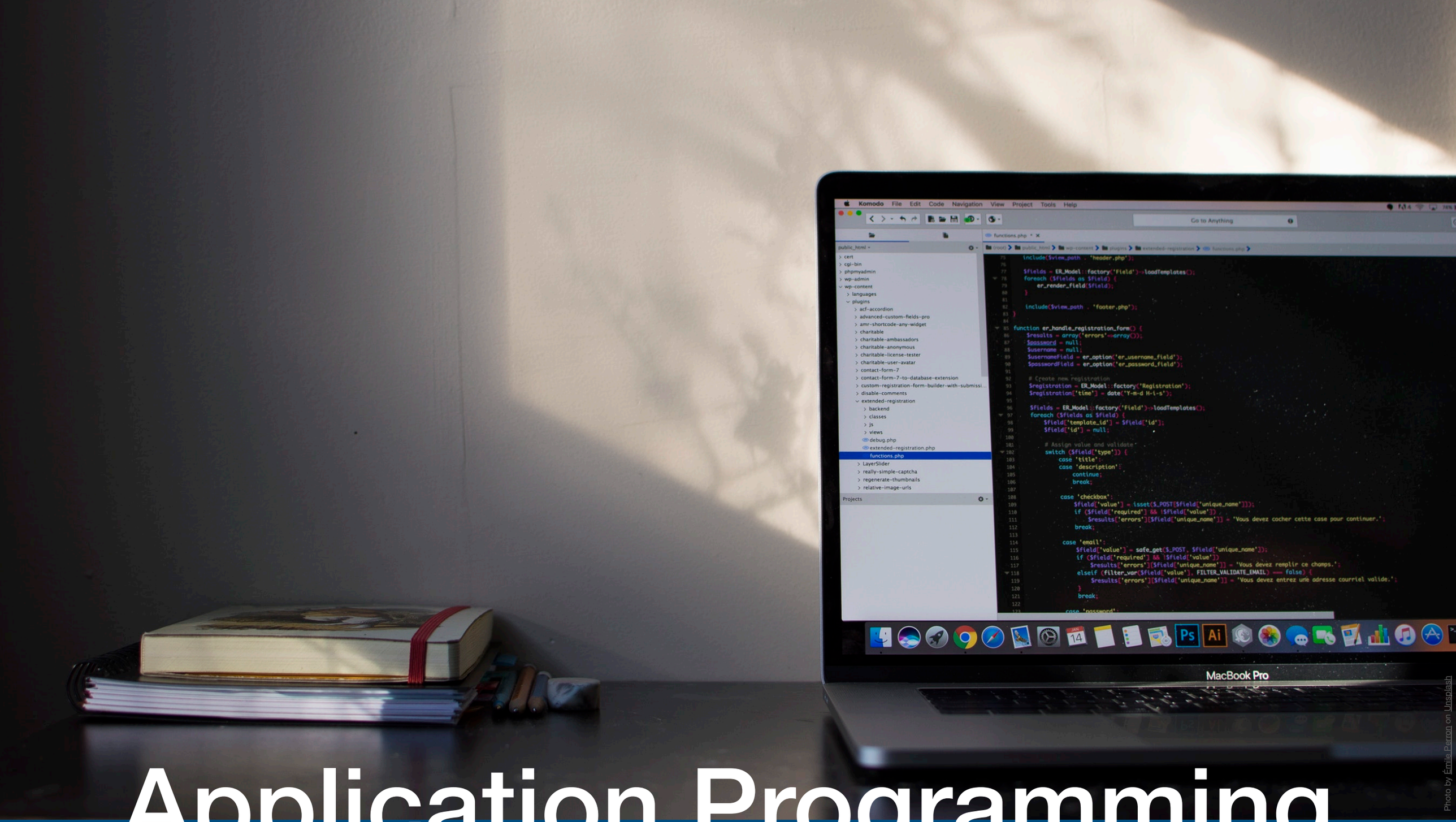


Application Programming

UTSA Computer Science 3443

Fall 2022

Dr. Amanda Fernandez

Week 10

- Exceptions & errors
- Exception handling

Fall 2022*

Week	Dates	Topic	Lab	Quiz	Chapter(s)
1	8/23 - 8/27	Introductions, Syllabus, Java basics, Eclipse			1-6
2	8/30 - 9/3	Java & OOP Concepts, Javadoc	1	1	8,14
3	9/6 - 9/10	ArrayList, Strings			7,15
4	9/13 - 9/17	File I/O, UML, MVC	2	2	12,13
5	9/20 - 9/24	Introduction to JavaFX			
6	9/27 - 10/1	JavaFX, SOLID	3	3	
7	10/4 - 10/8	Introduction to Git, *Review*			
8	10/11 - 10/15	**Midterm Exam**			
9	10/18 - 10/22	JavaFX applications	4	4	13,20
10	10/25 - 10/29	Exception Handling			11
11	11/1 - 11/5	Collections & Generics	5	5	16, 19
12	11/8 - 11/12	JUnit Testing, logging, Application Design			
13	11/15 - 11/19	Concurrency & Multithreading + Lambda	6	6	17
14	11/22 - 11/24*	***Review*** (+*Thanksgiving*)			21
15	11/29 - 12/3	**Team project demos**			
-	12/6 - 12/10	**Final Exams**			

Fred Brooks - The Mythical Man-Month. Required reading for every software engineer & programmer.

“How does a project get to be a year late?”

Fred Brooks - The Mythical Man-Month. Required reading for every software engineer & programmer.

“How does a project get to be a year late? One day at a time.”

Errors & Exceptions

Reading from a File

- When we read from a file, we copy data from disk into memory.
- Things can go wrong:
  - The file may not exist,
  - The disk may go bad,
  - The file may change while we are reading it.

Reading from a File

- In Java when things go wrong a

  `java.lang.Exception`

  object is created.

Possible Exceptions

- What would happen if we try to read from a file that doesn't exist?
  - `FileNotFoundException`
- What would happen if we try to read past the end of the file?
  - `IOException`
- What would happen if the file changes while we are reading it?
  - `IOException`

Possible Exceptions

- As developers, we must handle any problems that might occur due to input/output.

- Several ways to deal with exceptions:
  - "Handle" the exception with a ***try*** / ***catch*** block
  - ***throw*** the exception

The Call Stack

- When a Java program runs, execution begins in the main method.
    - That method creates objects & invokes methods on them.

- When execution moves to another method an entry is added to the ***call stack***.

The Call Stack

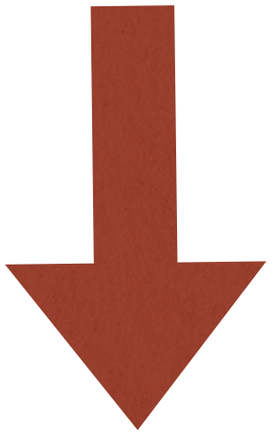
- Among other things, this entry contains:
    - The current method
    - Where the call occurred in that method

- When a method finishes executing…
    1. The entry is removed from the call stack
    2. Execution returns to the next line in that method
        - *This continues until the main method finishes.*

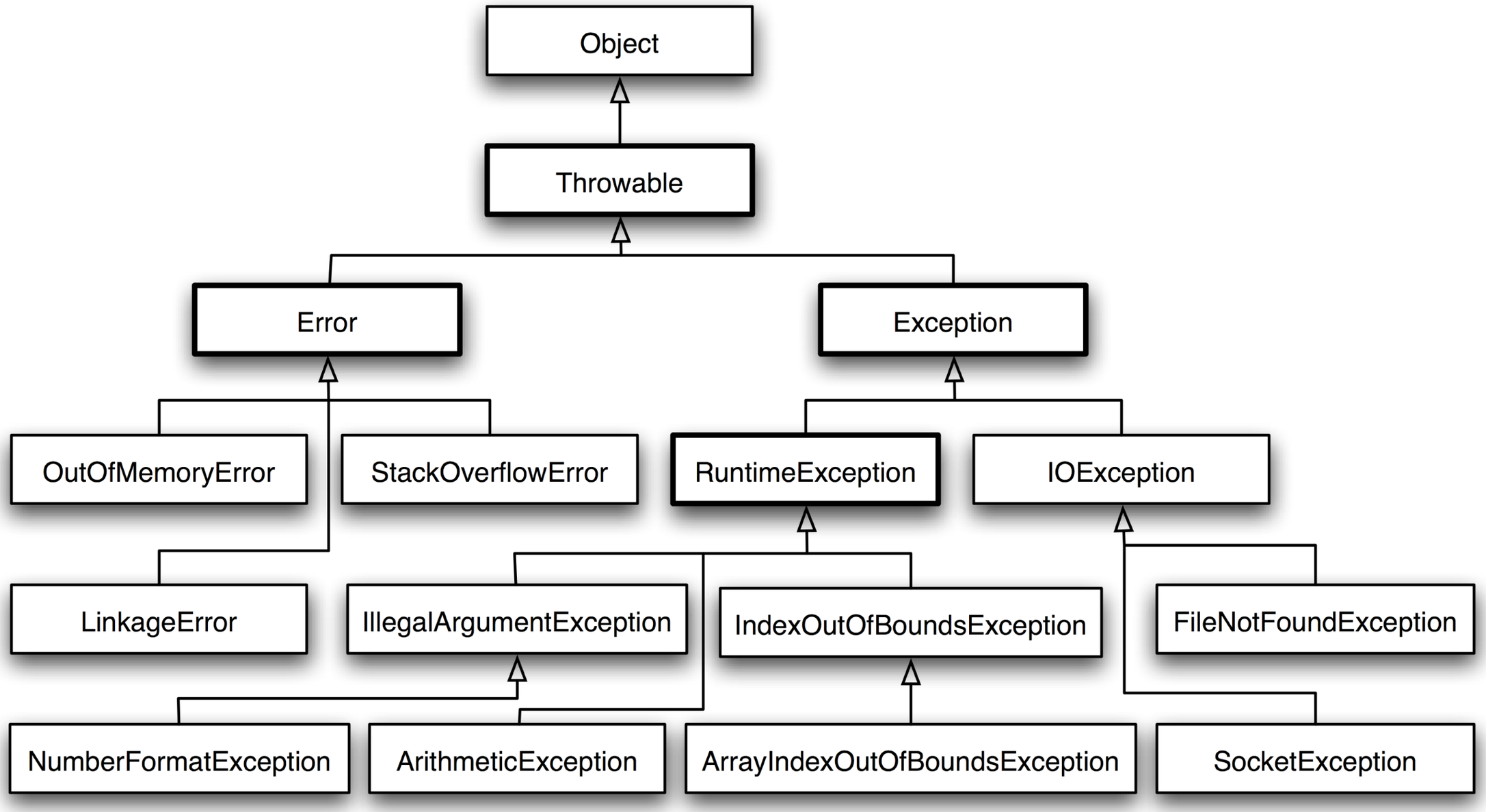
Example Call Stack

```
NullPointerException:
   at Student.getAverage(Student.java:79)
   at Student.toString(Student.java:62)
   at java.lang.String.valueOf(String.java:2615)
   at java.io.PrintStream.print(PrintStream.java:616)
   at java.io.PrintStream.println(PrintStream.java:753)
   at Student.main(Student.java:120)
   at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
   at
    sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.j
    ava:39)
   at
    sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccess
    orImpl.java:25)
   at java.lang.reflect.Method.invoke(Method.java:585)
```

These are here because
we ran our program on the JVM,
and have nothing
to do with our program.



Exception Inheritance Tree



- All classes inherit from **Object**
- All Exception classes inherit from **Exception**

Exceptions

There are two types of exceptions:

- **Checked exceptions** must be *caught* or *thrown, e.g.:*
  - `IOException`
  - `FileNotFoundException`
- **Unchecked exceptions** should never be caught or thrown, *e.g.*:
  - `NullPointerException`
  - `ArrayIndexOutOfBoundsException`

Using Try, Catch, and Finally Blocks

- Wrap all code that can cause a checked exception in **try**, **catch** (and optionally **finally**) blocks

```
try {
    //
    // code that can cause an exception goes here
    //
} catch(ExceptionClass1 ex1) {
   // handle this exception
} catch(ExceptionClass2 ex2) {
    // handle this exception
} finally {  // optional
    // do any required clean up
}
```

Exception Handling example

```
try {
    //
    // code that reads in from a file
    //
} catch(FileNotFoundException e) {
  // handle the case where the file isn't found
} catch(IOException e) {
   // handle the general issue I/O exception
} finally {
   // close the file (if it isn't null)
}
```

Throwing an Exception

```
public void methodA(){
        try {
            dangerZone();

        } catch( IOException e ){
            // handle the exception here!
        }
}


public void dangerZone() throws IOException{
        // code that reads in data from file
}
```

EBookReader

Exceptions & User Error

Errors & Exceptions

- Consider..
    - Where can errors happen caused by our **logic**?
    - Where can **exceptions** happen?
    - Where can **user error** occur?

- For each, *how can we **prevent** or **reduce** these?*
    - *What would the user expect?*