MODULE 1: INTRODUCTION TO PROGRAMMING

Exception Handling and File I/O part 1

TECH ELEVATOR
A Stride Company

```
for(int i = 0; i < array.length; i++)
{
   int sum += array[i+1];
}
```

Exception Handling

- A **compile-time error** occurs when there is a syntactical error or the compiler identifies code that cannot execute (e.g. method does not exist, accessibility not public, redefined constant, type mismatch)
- A **run-time error** occurs while the program is executing. The program often tries to access memory that is inaccessible or may be asked to perform an operation it is incapable of (e.g. access a restricted file, parse a value)

Run Time or Compile Time?

- for(int i = 0; i < array.length; i++)
- {
  int sum += array[i+1];
  }

Compile Time Error

```
double d = 0.0;
int i = d;
```

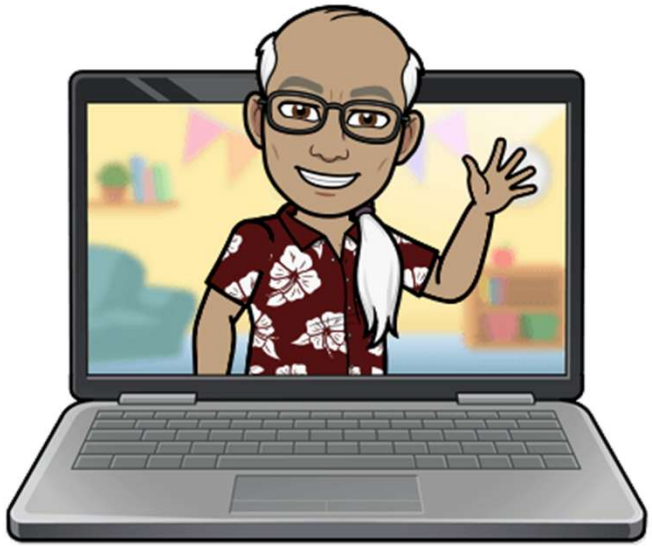
Exception handling

- **Exception handling** is the process of responding to exceptional errors in the programming. This processing often changes the flow of the program so that it can recover.

What is considered an exception?

- A file is deleted while the program is executing
- The network shuts down while calling an API
- Database access is denied
- Trying to access an array outside the bounds
- .....

LET’S CODE!



Checked vs. Unchecked Exceptions

```java
public void outputFile() {

    // Reading file from path in local directory
    Scanner fileScanner = new Scanner("C:\\test\\a.txt");


    // Printing first 3 lines of file "C:\test\a.txt"
    for (int counter = 0; counter < 3; counter++)
        System.out.println(fileScanner.nextLine());

}
```

```java
public void outputFile() throws IOException {

    // Reading file from path in local directory
    Scanner fileScanner = new Scanner("C:\\test\\a.txt");


    // Printing first 3 lines of file "C:\test\a.txt"
    for (int counter = 0; counter < 3; counter++)
        System.out.println(fileScanner.nextLine());

}
```

Checked vs. Unchecked Exceptions

```
public void outputFile() {

    // Reading file from path in local directory
    Scanner fileScanner = new Scanner("C:\\test\\a.txt");


    // Printing first 3 lines of file "C:\test\a.txt"
    for (int counter = 0; counter < 3; counter++)
        System.out.println(fileScanner.nextLine());
}
```

```
public void outputFile()
{
    try{
        // Reading file from path in local directory
        Scanner fileScanner = new Scanner("C:\\test\\a.txt");


        // Printing first 3 lines of file "C:\test\a.txt"
        for (int counter = 0; counter < 3; counter++)
            System.out.println(fileScanner.nextLine());
    } catch (IOException e) {
        System.out.println("Ooops");
    }
}
```

Making your own Checked Exception

Inherit directly from Exception makes your exception checked

Inherit from RuntimeException (which inherits from Exception) makes it unchecked

This is a **checked exception**

```
public class OverdraftException extends Exception {
```

This is an **unchecked exception**

```
public class OverdraftException extends RuntimeException {
```

Important Points

- try/catch/finally structure
- every other Exception class inherits from Exception
- You can make your own exceptions to handle control flow in the application
- stack trace

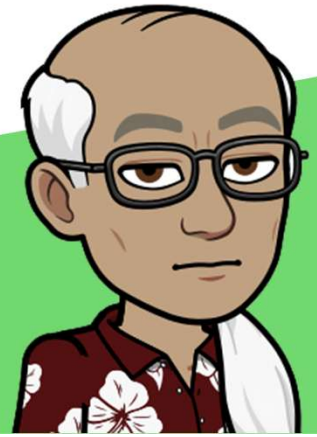
Reading Files

How do you read?

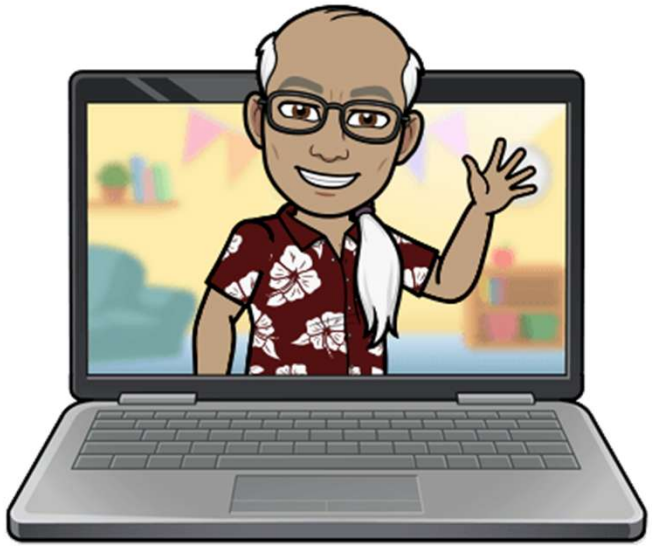
A **stream** refers to a sequence of bytes that can read and write to some sort of backing data store.

> Just like you know when you've come to the end of a book, stream readers know when they've come to the end of file.

Read entire file at a single time



LET’S CODE!



WHAT QUESTIONS DO YOU HAVE?

Reading for tonight:

File I/O and Writing Files