COP3502 Module 04B Review

So by now, all of you probably know these 2 things:

- Bugs happen
- They suck

What are some things we can do to deal with bugs effectively, or make them less common?

Unit testing- testing very small segments of code (methods or groups of methods) to make sure that they work

```
@Test
public static void testCountRuns1()
{
    byte[] testArray = { 0, 0, 2, 2, 2 };
    assert(RleProgram.countRuns(testArray) == 2);
}
```

```
Scanner input = new Scanner(System.in);

System.out.print ("Please enter a positive integer: ");
int number = input.next();

assert number > 0 : "This should have been positive!"
```

```
Please enter a positive integer: -1
Exception in thread "main" java.lang.AssertionError: This should have been positive!
```

Assertions- statements that test something we expect to be true

Also, use the debugger! (It's pretty awesome)
video link: https://youtu.be/bSL6kCzIOTI

Exceptions:

Exceptions handle errors (bugs) gracefully by providing structured error signaling (They tell the programmer—you—information (the where and why) about a bug that occurred in your code)

```
System.out.print("Enter a number:  ");

int num = scanner.next()

System.out.print("You entered  " + num);
```

Let's say the user is a silly goose and they enter a **string** (say "SillyGoose") instead of a number… what would happen?

Your **output** would be an **exception**

```
Exception in thread "main" java.lang.NumberFormatException: For input string "SillyGoose"
        at … [gives information about location of bug]
```

type

So, what can we do with exceptions…?

We can **throw them** and **catch** them!

(THIS IS REALLY IMPORTANT FOR YOU TO KNOW)

COP3502 Module 04B Review

Catching an exception:

```
Scanner input = new Scanner(System.in);
System.out.print ("Enter a number:  ");
int num = 1;

try{
        num = Integer.parseInt(input.next());
}
catch (NumberFormatException ooopSIIEE){
        System.out.println ("Why did you enter a string bro?");
}

System.out.println("No matter what happens I'm gonna print! :D");
```

Output:
Enter a number: I_am_a_string ← user input
Why did you enter a string bro?
No matter what happens, I'm gonna print! :D

The **try** block should have code that you are unsure about and that you want to "try out". For example, you cannot always be sure that the user will enter an actual number even when prompted to do so…

While your computer runs your **try** block, your catch block will catch any errors that happen and will print out whatever message you tell it to!

After all of this, your computer will keep right on executing!

Throwing an exception:

```
Scanner input = new Scanner(System.in);
System.out.print ("Enter a number:  ");
int num = 1;

try{
        num = Integer.parseInt(input.next());
}
catch (NumberFormatException ooopSIIEE){
        throw new RuntimeException("BRO WHAT DID U DO");
}

System.out.println("Let's see if I print or not…");
```

Output:
Enter a number: I_am_a_string ← user input
Exception in thread "main" java.lang.RuntimeException: BRO WHAT DID YOU DO
        at [some location where this code is]

This is kind of the same thing as before… the **try** block tries some code, and the catch block catches any errors that occur, but then it also **throws** an exception with a message as a parameter.

Once an exception is thrown, program execution is STOPPED.