Image Credit: Julius Schorzman

Java Lesson 3

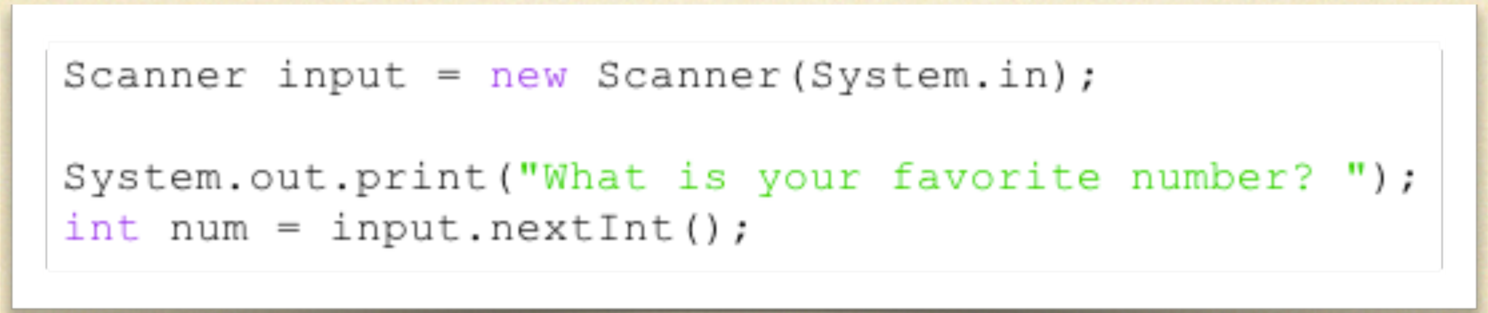
Concept Review

Exception Handling

- Anytime that you write software, chances are that your users will eventually find a way to "accidentally" crash the program that you've written.
- Most of the time, these crashes are unintentionally caused because the user didn't enter the desired information in correctly. (Such as when a user types in "text" instead of a number at a prompt.)
- Luckily, there are ways to catch most of these problems before they crash your application. It's called the try / catch block.

Try / Catch Block

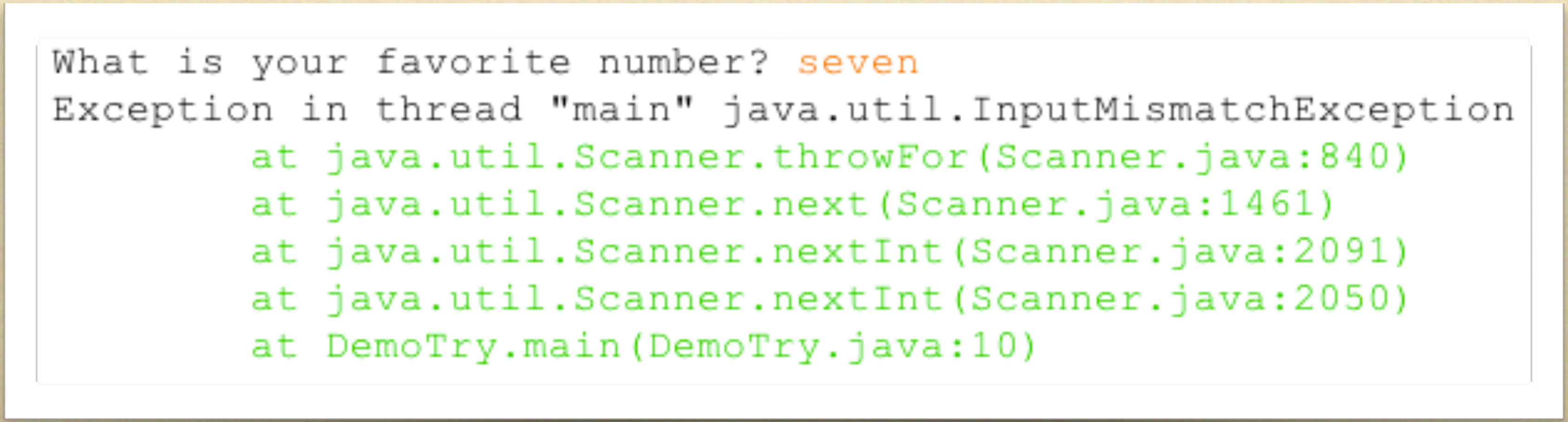
- The Try block is where you place code that *might* potentially crash your program. Consider the following example!



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

System.out.print("What is your favorite number? ");
int num = input.nextInt();
```

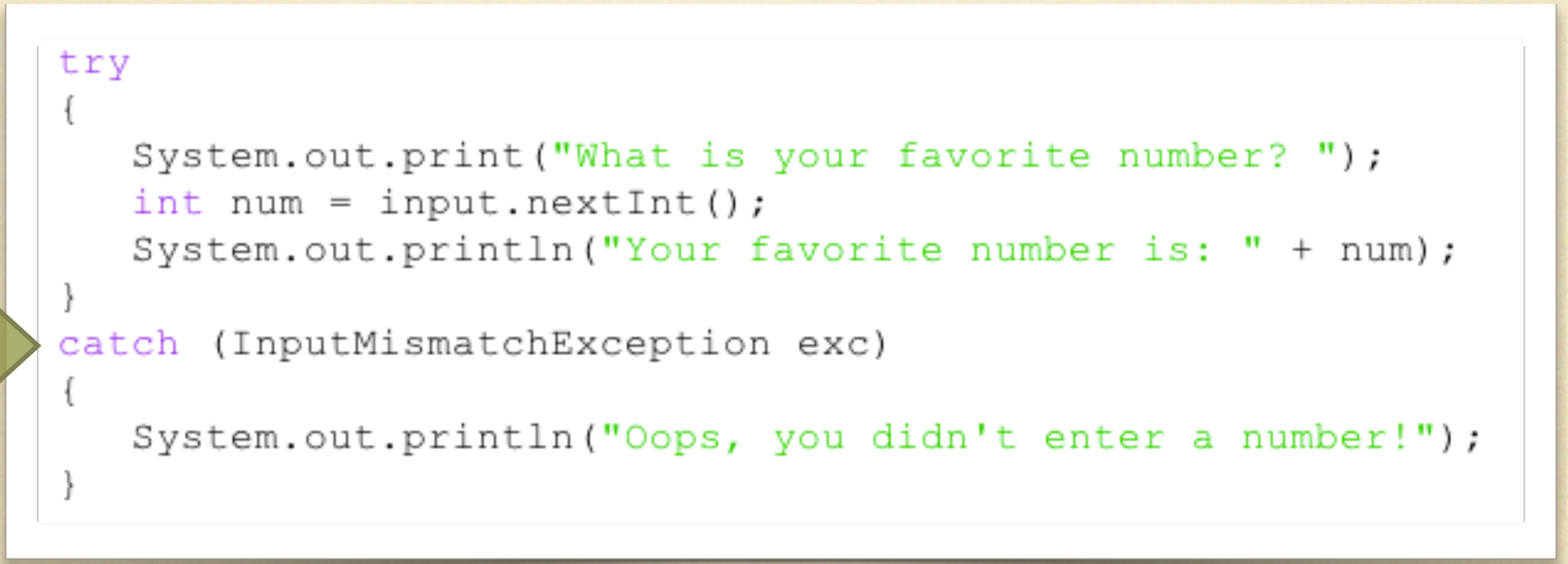
- What would happen if the user was to accidentally enter the word "seven" instead of number 7? It would crash the program with the following error, because the program was expecting an integer and not a String.



```
What is your favorite number? seven
Exception in thread "main" java.util.InputMismatchException
	at java.util.Scanner.throwFor(Scanner.java:840)
	at java.util.Scanner.next(Scanner.java:1461)
	at java.util.Scanner.nextInt(Scanner.java:2091)
	at java.util.Scanner.nextInt(Scanner.java:2050)
	at DemoTry.main(DemoTry.java:10)
```

Try / Catch Block

- To fix this, we can catch that exception! Here's how:



```
try
{
   System.out.print("What is your favorite number? ");
   int num = input.nextInt();
   System.out.println("Your favorite number is: " + num);
}
catch (InputMismatchException exc)
{
   System.out.println("Oops, you didn't enter a number!");
}
```

- If the user accidentally enters "seven" instead of 7, the catch statement will display a message to the user letting them know that they need to enter a number! It also stops your program from crashing too!

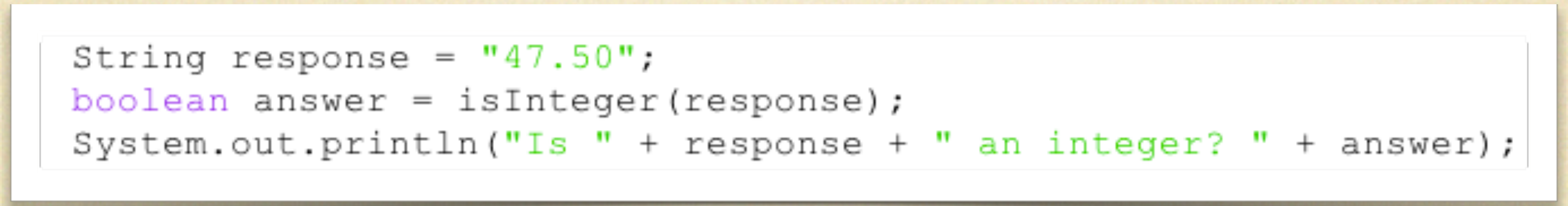
Getting Error Details

- Sometimes, it's handy to know what caused an error to occur. One way to find out is to use the getMessage() method.
    - **Please note:** the getMessage() method is quite helpful most of the time, but it does have its limitations with some of the exceptions out there. One of it's shortcomings is the InputMismatchException error, in which it simply provides a null response to the user -- which doesn't exactly tell you much!

Conversion Testing

- The try / catch block is frequently used to test information for validity!

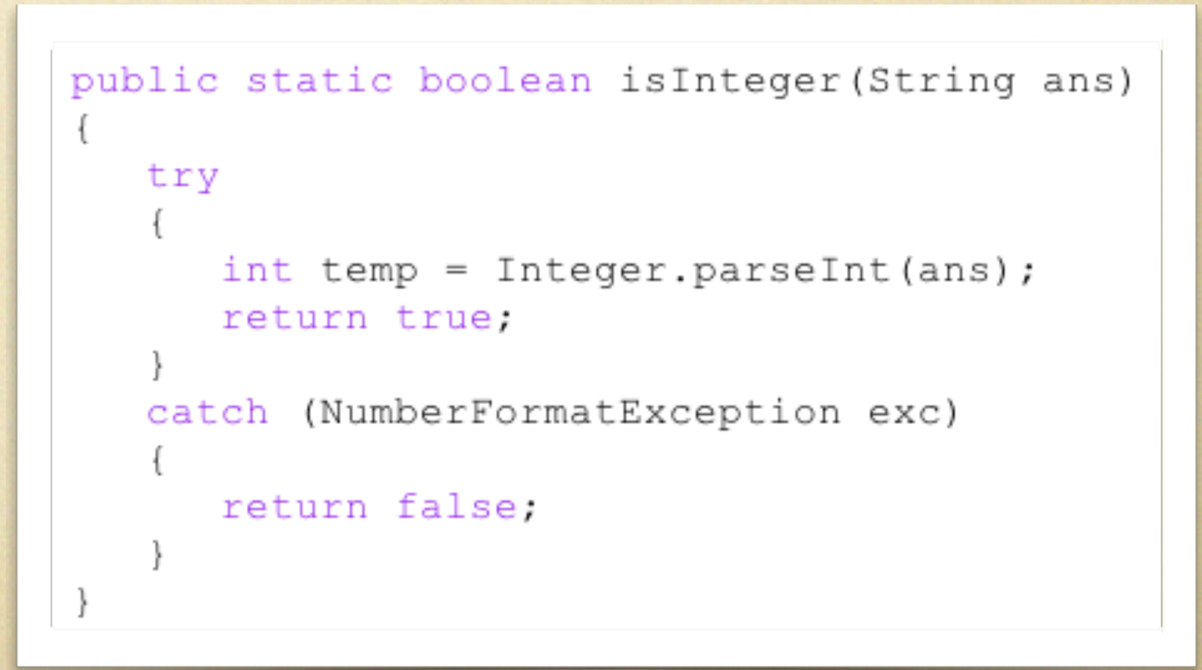
- Say that you need to test if a String contains an integer. How can we determine this? Why not use a try / catch block?!



```
String response = "47.50";
boolean answer = isInteger(response);
System.out.println("Is " + response + " an integer? " + answer);
```

- Notice that the isInteger() method receives a String and then attempts to convert it to an integer. If the conversion succeeds, it returns as true.

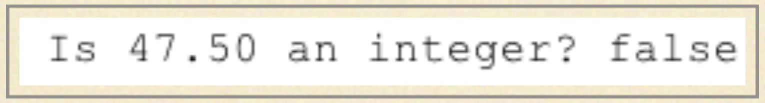
- If not, false is returned.



```
public static boolean isInteger(String ans)
{
    try
    {
        int temp = Integer.parseInt(ans);
        return true;
    }
    catch (NumberFormatException exc)
    {
        return false;
    }
}
```

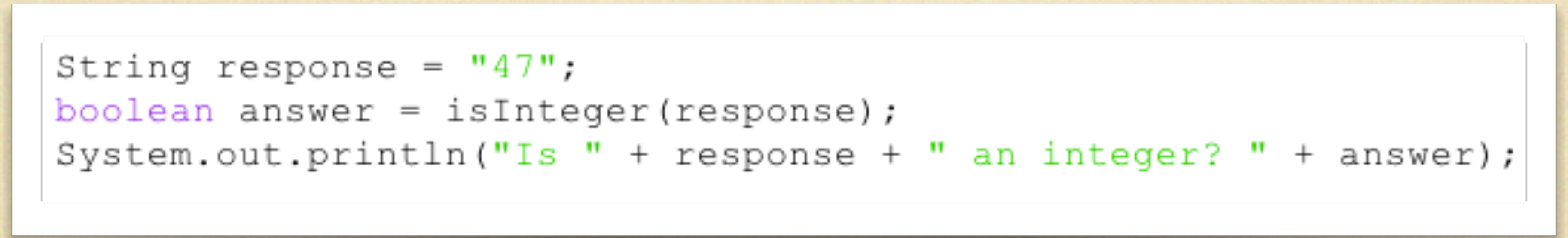
Conversion Testing

- When we run the program, it lets us know that the String does NOT contain an integer:

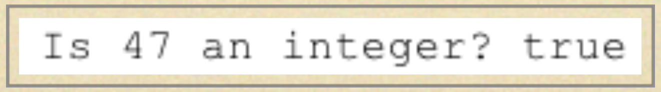


```
Is 47.50 an integer? false
```

- Yet, if we are to change the value of the String to "47" instead of "47.50", the response is different -- it's returns true!



```
String response = "47";
boolean answer = isInteger(response);
System.out.println("Is " + response + " an integer? " + answer);
```



```
Is 47 an integer? true
```