Lecture 19 - 3/26/2024

We began the lecture by discussing the Poker assignment more in-depth and you were provided with some details and hints that are meant to allow you to get a jumpstart on the hardest assignment of the semester.

1. You should ideally have a separate helper method per hand type, with the exception of no pair
2. You should create them with the idea of "at least" in mind, "at least" one pair or "at least" two pairs etc
3. In conjunction with (2) when evaluating the hand, evaluate from the best possible hand to the worst
4. Implement the 1 argument Game constructor with the testHand variable prior to the interactive game.

Reading and writing to files in Java is a process that brings together lots of different concepts from earlier in the semester.

In order to begin handling files in the first place you will need to import the Scanner class (you should know how to do this by now). You will also need to import all the stuff relating to files which are in the java.io package. So you'd need to import java.io.*; Remember that * acts as a wildcard and brings in all classes within the java.io package.

We are introduced to the File object type which takes in a file path in the form of a String as the sole argument to its constructor. Here is an example:

```
File myFile = new File("<pathToFile>");
```

Coupling this with a Scanner we now have a means to parse a file:

```
Scanner sc = new Scanner(myFile);
```

In order to write to a file we need to use an object known as a PrintWriter:

```
PrintWriter output = new PrintWriter("<pathToFile>");
```

We can iterate through the contents of the file that have been loaded into the Scanner using the .next() method on the Scanner object, combining this with the hasNext() method we can form a loop to continually iterate through until we can't anymore:

```
while(sc.hasNext()) {
     //some code dealing with the next() method
}
```

To write to the output file we can use the println method from the PrintWriter object so:

```
output.println(<whatYouWantToPrint>);
```

You must close the PrintWriter object after you finish printing for the output to be saved.

At this point we need to discuss exceptions, you've definitely seen them while doing your homework and some have been shown off in lecture as well. There are two kinds of exceptions: checked and unchecked, every single exception you have seen up to this point has been unchecked. When discussing files we are introduced to our first checked exception: FileNotFoundException here is a quick run down of which exception type means:

1. Unchecked: Not checked for acknowledgement or handling via the Java compiler if it occurs it is the fault of the programmer

2. Checked: Checked for acknowledgement or handling via the Java compiler not necessarily the programmer's fault if it occurs.

When I say acknowledgement I mean you need to tell Java that a method may throw an exception and you do this by attaching "throws someException" to the method header and this is shown in ExampleA.java on codio.

Handling an exception comes via a try-catch block in which we put the code that may throw an exception within the try and in the catch we specify an exception to catch and then place code to handle it. Here is an example:

```
try {
  //risky code
} catch (Exception e) {
  //handle exception e
}
```

There is also a finally block that is optional which will always execute regardless of whether an exception was caught or not.

Lecture 20 - 3/28/2024

This lecture we went more in depth on the topic of exceptions, we quickly reviewed the examples from the last lecture and then moved on to further applications of them.

Specifically we discussed writing our code in a defensive manner, if the user doesn't meet the program requirements then we do not continue running the program and only let them pass once their input is in a valid form. This is primarily used when combating Unchecked exceptions, due to their very nature.

I'll take this time to mention that you can catch multiple exceptions using multiple catch blocks:

```
try {
  //some risky code
} catch(Exception a) {
  //handle a
} catch(SomeOtherException b) {
  //handle b
}
```

When you try to catch multiple exceptions you need to make sure you go from specific to general top to bottom if a more general exception comes before a specific exception in your catches, then your code won't compile.