COP 3330 – Object Oriented Programming in Java
Writing Output to a File

PrintWriter, FileWriter classes

A PrintWriter object allows us to easily create an output file and write to it. Here is the link to the Java API documentation for the class:

https://docs.oracle.com/javase/8/docs/api/java/io/PrintWriter.html

We can create an output file by passing the PrintWriter constructor a FileWriter object. Here is a call that creates a PrintWriter reference called out which points to PrintWriter object set to write to the file, "output.txt":

```
PrintWriter out = new PrintWriter(new FileWriter("output.txt"));
```

If the file output.txt previously existed in the same file as the .class file that is getting interpreted, the file gets erased upon execution of this line. Then, future lines of code which write to the file will build the file up.

For simplicity, one could simply just use the print and println methods in the PrintWriter class, which work the exact same as the corresponding methods for standard output:

```
// Prints s to this PrintWriter object.
void print(String s);

// Prints s followed by a new line character to this PrintWriter object.
void println(String s);
```

Finally, after you have completed writing to a file, you should close the file via the close method:

```
// Closes the stream and releases any system resource associated with it.
void close();
```

Also, on occasion, it's useful or necessary to flush the stream (same for reading input):

```
// Flushes this stream.
void flush()
```

When you print/println, by default this doesn't automatically put output in the file. Rather, it places the output in a buffer and then the contents of that buffer get flushed periodically into the file (written to the file). If you need that buffer flushed at a fixed point in time, then it's best to call this flush method.

Since you know how standard output works already, it's fairly easy to make the transition to writing to files. The only typical changes are:

1) Creating a PrintWriter object.
2) Closing the PrintWriter object after you are done.

Sorting Student Presentations

Let's just do a quick edit to the Sorting Student Presentations problem from the previous lecture to write the output to a file sorting.out. Since the only printing in the original solution is done in the last four lines of the case loop, we can confine our edits to that portion of the code. Here is the old code:

```
System.out.println("Class #"+loop+" ordering");
for (int i=0; i<n; i++)
    System.out.println(group[i]);
System.out.println();
```

To write to a file, before the main class loop, add this line:

```
PrintWriter out = new PrintWriter(new FileWriter("sorting.out"));
```

Then, change the four lines above as follows:

```
out.println("Class #"+loop+" ordering");
for (int i=0; i<n; i++)
    out.println(group[i]);
out.println();
```

Finally, after the case loop, where we close the input file, also close the output file:

```
out.close();
```

Fast I/O in Java

Both printing to the screen and reading via a Scanner object are quite slow because they try to check for quite a few specific exceptions. If you have a specific need to read input faster or write output faster, one should not use standard printing or a PrintWriter object to write to a file. (Also, for standard output, one should not call the print or println methods many times.)

This section of the notes will introduce the following classes which allow for faster input and output in Java:

1) BufferedReader (can be used for standard input or File input)
2) StringTokenizer (previously covered)
3) StringBuffer (avoids using many print/println)
4) BufferedWriter (can be used for standard output or File output)

The BufferedReader class does not check for too many exceptions, so it runs faster. Here is how to create a BufferedReader object from standard input:

```
BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
```

Here is the BufferedReader API:

https://docs.oracle.com/en/java/javase/22/docs/api/java.base/java/io/BufferedReader.html

From this list, the most commonly used methods are:

```
void close()
String readLine()
boolean ready()
```

Thus, we can only process input with a BufferedReader reading line by line. As previously mentioned, in order to do this, we typically need to use a StringTokenizer to tokenize the contents of each line.

If we want to do fast output to the standard output stream, one trick is to build a String (so essentially keep your own buffer) and then call print or println once at the end of the program to output the whole large string. The problem is that String objects are immutable, and running lines of code of the form:

```
s = s + t;
```

where s is a String and t is a String that we want to append to s is that the run time of this method is the sum of the lengths of s and t. If we append strings of length 1 into an initially empty string of length 0, and do so 100000 times, our run-time would be roughly 500,000,000 operations (1+2+3+....+100000), when realistically, we should just be doing on the order of 100,000 operations.

The StringBuffer object is mutable, so we can append to the end of a StringBuffer object efficiently. Here is the API for it:

https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuffer.html

The standard method to use a StringBuffer is to create an empty object, and then just append items to the object as needed:

```
StringBuffer sb = new StringBuffer();
for (int i=0; i<n; i++) {

    // Some code here.
    sb.append(res);
}

// Only one inefficient print call.
System.out.print(sb);
```

In the code segment we assume that n is read in before and has a positive value and that res is a variable scoped inside of the for loop that is what we want to append to our overall output.

This same technique will work with output to a file via using a PrintWriter object with a single call to print or println.

BufferedWriter class
Finally, an alternative way to write quickly to a file is to use the BufferedWriter class:

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/BufferedWriter.html

This class is very simple and just allows you to write one character at a time and flush the stream as needed.