Jack Meng

Ms. Zinn

Intro to Programming & Java

11 June 2021

Java Final Project Paper

For my Java Final Project, I have created a simple clicking game. In this game, its main focus is the demonstration of using the Buffered and File classes to store data. Although there are many other ways besides storing data in a simple file, this is most likely the most simple and most efficient way for my project.

The main part of the program is for the user to constantly click a button, and after a certain point, you can then upgrade the multiplier meaning you can then get more clicks per click. However, if I was to just close the program, my data would be lost. To compensate for this, I can use the Buffered and File packages to help me store this data in a '.txt.' or '.save' file. This means that long-term storage of these data can be achieved. It showed also be noted that not only will the program save the clicks and multiplier, it will also be able to retrieve those data for future usage. It can also be noted that there are not just two main packages one can use to fulfill writing and reading tasks to a file.

A Buffered Class/es used throughout the program were the classes BufferedReader and BufferedWriter, both serving the purpose of reading and writing. However, it could be possible for the program to just use the FileWriter and FileReader classes. It is very much possible, but the overall efficiency of Buffered is much better than that of File. On the other hand, File packages, like FileReader are mostly good for small quantities of data. In my project, as I mainly store integer data, it is both ethical and much more efficient to use FileWriter over BufferedWriter.

In my project, I mainly used FileWriter, File, and BufferedReader to retrieve and store data from the main interface. FileWriter is used for the process of writing the data onto the files using the .write() method. The File class as researched was mainly used for the declaration of a file and is mainly used throughout my program to either see if the file exists or not (with .exists()) and to delete files (for the Reset data button). BufferedReader was used mainly for both ways, reading, and writing. I utilized it in storing data to get the old data, while I used it in retrieving data by reading the files (if they exist).

Much of the mentioned packages, reader and writer, use a form of “stream,” and to be more specific, I/O streams. These streams provide input and output functionality, with *InputStream* providing input from an X source, and *OutputStream* providing output from an X source. These streams also need a way to close themselves, this is accomplished by using the *close()* method, when you call the close method, it is then, the stream can pass all of its information to X destination. Similarly, the *flush()* method mainly is used to remove any elements within the X stream and then immediately written to their destination(s). If the user does not close the stream, one can experience memory leaks if this involves This can be easily traced back to the simple Scanner class, which uses *System. in* as its main InputStream (*new Scanner(System.in))*. This also means that one can simply use Scanner to read files. The main difference between using a Buffered stream and one that isn’t, is the fact that one that is buffered can read/write data in sizes that are greater than non-buffered, which must read data byte by byte.

Throughout the research and development phase, I have learned a lot about Java itself and its core mechanics from the internet and my own experience with using it for my final project. Things like *InputStreams* & *OutputStreams* were the main focus of my research and determining if I should use Buffered or File or both. Along with not making the program boring, I also included objectives that will grow and a method to return a random string that will act as the “news”. Much of what I learned from using the packages was that I have gotten a better grasp with error handling (using *Try/Catch* and/or *throws*), it also taught me how to effectively decide which package I should and in which scenario. This new knowledge of reading and writing to files can further my idea of how to store data simply without having to dig deep into data structures and more.

Works Cited & Citations

1. “BufferedInputStream.” *BufferedInputStream (Java Platform SE 8)*, 19 Mar. 2021, docs.oracle.com/javase/8/docs/api/java/io/BufferedInputStream.html
2. Jenkov, Jakob. “Java FileWriter.” *Tutorials*, Jenkov, tutorials.jenkov.com/java-io/filewriter.html
3. “Java.io.OutputStream.close() Method.” *Tutorialspoint*, www.tutorialspoint.com/java/io/outputstream_close.htm.
4. Jumba, Isaac. “Why Use BufferedReader and BufferedWriter classes in Java.” *Medium*, Medium, 23 Sept. 2015, medium.com/@isaacjumba/why-use-bufferedreader-and-bufferedwriter-classses-in-java-39074ee1a966.

5. Baeldung. “A Guide to the Java FileReader Class.” *Baeldung*, 21 May 2021, www.baeldung.com/java-filereader.
6. Chandrakant, Kumar. “Guide to Java OutputStream.” *Baeldung*, 19 May 2021, www.baeldung.com/java-outputstream.
7. *Java Practices->Buffering Usually Appropriate*, www.javapractices.com/topic/TopicAction.do?Id=122#:~:text=Unbuffered%20input%2Foutput%20classes%20operate,disk%20in%20larger%2Dsized%20chunks.&text=With%20small%20files%2C%20buffering%20usually%20makes%20little%20or%20no%20difference.
8.