How to Turn in Your CRUD

For this application, you are going to create a WORD document with an entire run through of your program…..YOUR FILE NAME MUST BE CALLED **Crud_Output_Jon_Street**.(Unless you are not Jon Street)

Make sure you have at least 4 records saved to your disk file.

Your word document will look like this…..

Name: Jon Street AKA StreetFighter. Period: 7 APCS APCS Java CRUD RunThrough

Purpose:
This program's purpose is to allow the user to manage a list of characters from an adventure game.
Each character has stats that are kept track of;
name, class, gender, level (start at level 1 and go up) and experience (0-99).
Once a character reaches 100 experience points, they go up to the next level and experience goes back to 0.

Characters will be loaded in from the local file system.
The user can display them, edit any of the fields in them, add new characters and remove characters from the list.
Saving to disk and reading in from disk as well as archiving to a backup is supported.

compareTo
My compareTo method allows two characters to be compared mathematically.
The way is works is that levels are compared.
Higher levels will get moved to the front of the ArrayList when the Collections.sort logic is executed.
If the levels are the same, the compareTo resorts to comparing names alphabetically by ASCII code.

My demonstration below will show that newly added characters add by default at the end but that when Collections.sort executes, characters with higher levels will move to the top.

Test Sequence for CRUD Application		
1. Open your Application		
2. Read in text file	MUST HAVE 4 records already out there	
3. Display	See the 4 records	
4. Delete an item	Need to present the user with a list so they can see what to choose. I prefer to see a message like "Jeff has been deleted."	
5. Display	One less should be there.	
6. Edit an item	Allow user to choose which field to edit	
7. Display	Should show altered record.	
8. Add a new item	Type in new data MAKE SURE IT BELONGS HIGHER UP WHEN SORTED (don't make it just garbage input) Add at end	
9. Display	Show new record at bottom	
10. Sort	Make sure the order is CHANGED after sort (I prefer some sort of message like "**List is now sorted by level then alphabetical by name**"	
11. Optional 2nd Sort If you used a Comparator and sort on a separate criteria	Run your next sort and then display your output, add a comment in your word doc describing the new order of the list.	
12. Display	Should show updated positions.	
13. Save	Either save to both normal file and archive at once or make separate menu options.	
14. Optional Separate Archive		
15. Display		
16. Change a record somehow (either delete, add a new or edit a record)		
17. Try to quit the program	The program should warn you that you are about to lose unsaved data…choose that you want to save	
18. Exit java program		
19. Restart and Read in		
20. Display	Should have records that were updated as of the last run of the program.	

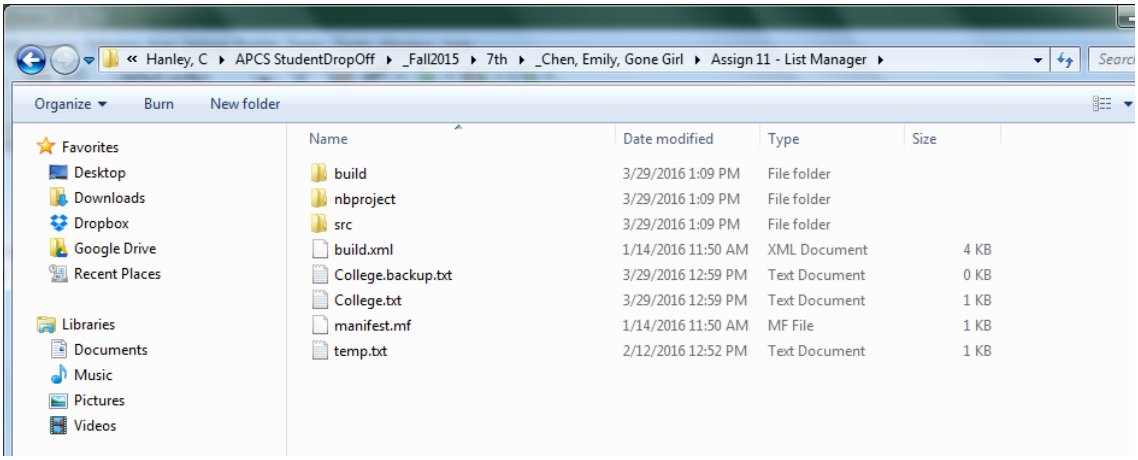
1st Run Through of CRUD
(must be red in font size 28 font)

<Perform Steps 1-18 and paste into word, each section must be preceded by a red font 28 heading>

2nd Run Through of CRUD

<Perform Steps 19-20 and paste into word>

Take a screen shot of your directory in your project showing your .txt file and backup file names
Hit printscreen and paste it into MS Word so I can see both file names




```
//Copy Some of your comments to earn your 10 points for comments at the bottom
String result = "";
// this will reference one line at a time
String line = null;
try {
   // FileReader reads text files in the default encoding.
   FileReader fileReader = new FileReader(fileName);
   // always wrap FileReader in BufferedReader.
   BufferedReader bufferedReader = new BufferedReader(fileReader);
   line = bufferedReader.readLine();
   while (line != null) {
      result += line + "\n";
      line = bufferedReader.readLine();
   }
   // always close files.
   bufferedReader.close();
   // exception handling
```

Paste in your compareTo method in here: (or each of your Comparator classes)

```
public int compareTo(Object o) { //Getting an object to compare said object to

   Character other = (Character) o; //Casting the character

   //First check levels....
   if(level == other.level){
      //if levels same, then go to name as secondary ordering
      return name.compareTo(other.getCharacterName());
   }
```

```
    else return level - other.level; //Checking to see which level is greater and if it needs sorted
}
```