Prerequisites

It is recommended to have completed Introductory Project 1.

Note that Java uses "streams" to manage I/O to and from things like the console, files, and sockets. Often, stream libraries are meant to wrap around one another, each layer adding more functionality than the one below. Do some research on the following Java classes: File, FileReader, BufferedReader, FileWriter, BufferedWriter. You get/create a File object that you hand to a FileReader object for File I/O capabilities, and then hand the FileReader object to a BufferedReader for ease of access methods – such as `readLine()`. Also, look at String.split().

Requirements

Write a Java program that provides the functionality of a checkbook – including persistent storage.

Specifically,

- Allow the user to select an existing checkbook (file) or to create a new one.
- Use the filename suffix ".cbk" for the checkbook file.
- If a new or empty checkbook is selected, prompt the user to enter a beginning balance.
- If an existing checkbook is selected, always display the current balance at the start or after a transaction is complete.

- Allow the user to enter three types of transactions – CASH, Check, and DEPOSIT. A CASH transaction deducts money from the account with no associated check number. A Check deducts money from the account with an associated check number. Deposit adds money to the account.
- All transactions should use the current date from the system.
- All transactions should allow the user to enter a memo.
- A completed transaction must be recorded to the checkbook file in a comma-delimited format. Each transaction must appear on its own line. The format for each transaction in the file is as follows:
    - CASH:

      [*date*],CASH,[*amount*],CASH,[*memo*],[*new balance*]
    - Check:

      [*date*],[*check #*],[*amount*],[*to field*],[*memo*],[*new balance*]
    - DEPOSIT:

      [*date*],DEPOSIT,[*amount*],DEPOSIT,[*memo*],[*new balance*]
- When the user selects to create a Check transaction, auto-increment the check number from the last used number. Check numbers for new checkbooks should start with 1001.

- Allow the user to select a box to view the completed check. If the box is checked, upon completion of a check transaction, display a completed check. Include check number, the "to" field, memo, amount, etc. – everything from the last assignment.
- Prior to completing a transaction, provide a warning to the user if any transaction would create a negative balance. Allow the user to cancel this transaction and do not record it if it is cancelled.
- Allow the user to repeatedly enter multiple transactions without having to exit and restart the program.
- Data must persist through multiple openings and closing of the program. In other words, the user must be capable of closing the program, starting it again, creating another check transaction, and having the program provide the next valid check number and provide a correct balance based upon all transactions in the checkbook.
- Accept all required user input via graphical user interface.
- Provide all user feedback/output via graphical user interface.
- You may use one additional file for program-specific data.
- Suggestion: Validate your results by importing your checkbook file into Excel and verifying correct entries and a correct balance.
- Using the PSP forms, estimate the effort *prior* to beginning your design or implementation and then carefully denote the time spent on the project so that you can compare to your estimates.

- Provide a class diagram of all Java classes that you created
    - You must **include** class relationships

- Provide a requirements list with your estimate Product Planning Sheet. The requirements List must also specify all functional (behaviors the program must have) and non-functional (project requirements that are not program behaviors) requirements that you perceive that are **not** explicitly stated in this assignment document. Be sure to separate the functional and non-functional requirements.

- Provide a page or two summary of your testing approach and test results. Identify the types of tests that you conducted (from our testing lecture and Schach Ch 6). Be sure to include test case examples.

- At the end of the summary, include a couple of sentences about what worked for your team and what you'd improve upon if you were to do another assignment together or redo the assignment. Discuss any changes (positive or negative) from last assignment (IP01) based upon your reflections there.

Grading

The assignment is worth 50 points. A maximum of 20 points will be given for your team's coded solution. A maximum of 10 points each will be given for your class diagrams and test summary. A maximum of 10 points will be given individually for each team member's PSP forms.

Extra Credit (3 points)

Worth 1.5 points each, grade will not exceed maximum of 50 points with any extra credit points added:

- When opening a checkbook, allow the user to select the directory to save, but limit the Open File dialog to just display ".cbk" files.
- When saving a checkbook file, append ".cbk" to the filename if not done so already by the user.

Submission

Your final submission must include the following:

- Java source code files (.java)
- Java JAR file of working code,
- Class diagrams with relationships,
- Test summary, and
- Completed PSP forms

Files (with the exception JAR files) should be individually named "IP2_NAME_[description]". Place all files into a ZIP archive named "IP2_NAME". Only one team member need upload the team documents. Note that NAME refers to your team name. Individual PSP files should also follow this convention except that NAME would be your last name and first initial (*e.g.*, MayerG).

- All files must be uploaded to Moodle no later than one hour prior to class on Thursday, 13 September 2012. The CATME peer evaluation will be due Saturday, 15 Sep 2012, by 1:00pm.