CS2336 – PROJECT 3 – Cidercade Database

Pseudocode Due: 3/31 by 11:59 PM (No late submission)

Implementation Due: 4/6 by 11:59 PM (No late submission)

Final Code Due: 4/13 by 11:59 PM

KEY ITEMS: Key items are marked in red. Failure to include or complete key items will incur additional deductions as noted beside the item.

Submission and Grading:

- All project source code will be submitted in zyLabs.
    - Projects submitted after the due date are subject to the late penalties described in the syllabus.
- Each submitted program will be graded with the rubric provided in eLearning as well as a set of test cases. These test cases will be posted in eLearning after the due date.
    - zyLabs will provide you with an opportunity to see how well your project works against the test cases. Although you cannot see the actual test cases, a description will be provided for each test case that should help you understand where your program fails.
- **Type your name and netID in the comments at the top of all files submitted. (-5 points)**

Objective: Use object-oriented programming to implement and utilize a binary tree

Problem: Cidercade in Dallas has a primitive flat file database that needs to be updated. This database contains the names, high scores (w/ initials) and the total number of plays of their games. The database also contains the total amount each game would earn if not on free play. You will build a simple interface to interact with the database.

Pseudocode: Your pseudocode should describe the following functions

- For each function, identify the following
    - Determine the parameters
    - Determine the return type
    - Detail the step-by-step logic that the function will perform
- Functions
    - Binary Tree
        - Adding a record
        - Deleting a record
        - Searching for a record
    - Main class
        - Main function
        - Editing a record
        - Printing to the database file
        - Any additional functions that you plan to create

Implementation Phase

- Insert (can be iterative)
- Search (can be iterative)
- Display tree (can be iterative)

Classes

- Use good programming practice for classes – proper variable access, mutators and accessors, proper constructors, etc.
- Remember that classes exist to be used by other people. Just because you don't use it in the program doesn't mean it shouldn't be coded and available for others to use in theirs.
- As with previous projects, you are open to design the classes as you see fit with the minimum requirements listed below
- All classes must be of your own design and implementation.
  - **Do not use the pre-defined Java classes (-20 points if pre-defined classes used)**
- **Requirements**
  - Binary Tree Class
    - Named `BinTree.Java`
    - No reference to anything inside Payload
    - Root pointer
    - Will contain functions for basic functionality of a binary tree
      - Insert
      - Search
      - Delete
    - May contain other functions
    - All traversals of the tree will be done recursively (-10 points if not)
      - This includes functions that add, delete, search and display the tree
      - This **does not** include the breadth-first traversal
  - Node Class
    - Named `Node.java`
    - Generic class (-10 points if not)
    - Must be comparable
    - No reference to anything inside payload
    - Left and right pointers
    - Generic object to hold data
  - Payload Class
    - Named `Payload.java`
    - Must be comparable
    - Name
    - High score
    - Initials
    - Plays

Details:

- Start the program by prompting the user for the following information in the order listed

        - Database filename
            - The contents of the file are stored in the binary tree
            - This file may be empty
        - Batch filename
- All records are stored in memory with a binary tree (-10 points if not)
- The batch file will provide the following options:
    1. Add a record to the database
    2. Search for a record and display it
    3. Edit a record
    4. Delete a record
    5. Sort records
- **Add a record:** Create a new node and add it to the tree
- **Search for a record:** The search term will be a word or phrase. Search through the entries and display the complete record for any game that matches the search term.
- **Edit a record:** With a given game name, the program should update the record and confirm the change by displaying the new record on the screen. The following items can be edited:
    1. High score
    2. Initials
    3. Number of plays
        - If number of plays is edited, the revenue should be recalculated
            - $0.25 per play
- **Delete a record:** User will enter a game name. Delete the record from the tree
- **Sort records:** Display records in ascending or descending order by name.
- The binary tree will be written to a file at the end of the program using a **breadth-first traversal**
    - The new database file will be named `cidercade.dat`
- Revenue is based on a quarter per play
- You can expect all input to be valid.

Database Format: The database will exist in a file. Each record will be on a separate line in the file and each line will have a new line character at the end of the line (except for the last line which may or may not have a newline character). Each record in the database will have the following format. Notice that each field will be separated by a comma and a space.

```
<name>, <high_score>, <initials>, <plays>, <$><revenue><\n>
```

- <name> - may be multiple words
- <high_score> - 1-9 digits
- <initials> - 3 characters – no white space
- <plays> - 1-4 digits
- <revenue> - <1-4 digits><decimal><2 digits>

Input: All input will be file based. Prompt the user to enter the filename for the database file. This would normally be hardcoded in an application, but zyLabs requires a filename for multiple test files. A batch file will be used with the database. Prompt the user to enter the filename for the batch file. Each line in the file will be a command to perform on the database. Each command will be on a separate line in the file and each line will have a new line character at the end of the line (except for the last line which may or may not have a newline character). The format for each option is listed below. There is a single space between fields.

- Add a record
    - `1 ”name” high_score initials plays $revenue`
    - The double quotes surround the name so that you know where the end of the name is
- Search for a record
    - `2 <search term>`
    - Search term may contain spaces
- Edit a record
    - `3 “name” <field number> <new value>`
    - `<field number>`
        - 1 = high score
        - 2 = initials
        - 3 = number of plays
    - The double quotes surround the name so that you know where the end of the name is
- Delete a record
    - `4 <name>`
    - Name may contain spaces
    - Double quotes are not necessary here since there is no data after the name
- Sort records
    - `5 <asc/dec>`
        - A single word will follow the value: `asc` or `dec`

Output: Each command in the input file will generate output to a log file. The log file will be named `cidercade.log`. After each command output, write 2 blank lines to the file. The output for each command is as follows:

- Add a record

    -
      ```
      RECORD ADDED
      Name: <name>
      High Score: <high_score>
      Initials: <initials>
      Plays: <plays>
      Revenue: $<value> - formatted to 2 decimal places
      ```
- Search for a record
    - `<name> FOUND or <name> NOT FOUND`
    - If found

        -
          ```
          High Score: <high_score>
          Initials: <initials>
          Plays: <plays>
          Revenue: $<value> - formatted to 2 decimal places
          ```
- Edit a record
    - `<name> UPDATED`
    - `UPDATE TO <field> - VALUE <value>`
        - Possible fields: `high score, initials, plays`

    - `High Score: <high_score>`
      `Initials: <initials>`
      `Plays: <plays>`
      `Revenue: $<value> - formatted to 2 decimal places`
- Delete a record
    - `RECORD DELETED`
      `Name: <name>`
      `High Score: <high_score>`
      `Initials: <initials>`
      `Plays: <plays>`
      `Revenue: $<value> - formatted to 2 decimal places`
- Sort records
    - `RECORDS SORTED <ASCENDING/DESCENDING>`
    - Display all records (one per line) in the proper order
        - Line format: same as the database format