Seneca College

Applied Arts & Technology

SCHOOL OF COMPUTER STUDIES

JAC444 Submission date: June 21, 2021

Workshop 4

Description:

The following workshop lets you practice basic java coding techniques, creating classes, methods, using arrays, **Java I/O**, inheritance, polymorphism, Exceptional Handling.

Task 1:

Write a program for *hangman game* that randomly picks a word from a text file named **hangman.txt** (by default you file should have 10 words in it) and prompts the user to guess one letter at a time, as shown in the sample run.

- Each letter in the word is displayed as an asterisk. When the user makes a correct guess, the actual letter is then displayed.
- When the user finishes a word, the following steps should happen
  - Display the number of misses to the user.
  - Ask the user to give you a new word, then add that word to your file **hangman.txt**. *(Optional if you want to check for the duplicate words in the file)*
  - Ask the user whether to continue to play.
- If the user missed the same letter more than one time give extra hint to the player like "You've already tried this letter, try another letter". Don't count twice the same letter which is missed by the user.
- Have minimum 10 words of your choice in the file **hangman.txt**.

```
(Guess) Enter a letter in word ******* > p
(Guess) Enter a letter in word p****** > r
(Guess) Enter a letter in word pr**r** > p
            p is already in the word
(Guess) Enter a letter in word pr**r** > o
(Guess) Enter a letter in word pro*r** > g
(Guess) Enter a letter in word progr** > d
            d is not in the word
(Guess) Enter a letter in word progr** > m
(Guess) Enter a letter in word progr*m > d
```

```
You have already tried d, try a new letter
(Guess) Enter a letter in word progr*m > a      ↵Enter
 The word is program. You missed 1 time
 Enter a new word to be added in the memory> apple
 Do you want to guess another word? Enter y or n>
```

Note: Students can make the output better as well in easier and more readable format.

Task – 2: (Case Study - File Matching)

In commercial data processing, it is common to have several files in each application system. In an accounts receivable system, for example, there's generally a master file containing detailed information about each customer, such as the

- customer's name
- address
- telephone number
- outstanding balance
- credit limit
- discount terms
- contract arrangements
- possibly a condensed history of recent purchases and cash payments.

As transactions occur (i.e., sales are made and payments arrive in the mail), information about them is entered into a file. At the end of each business period (a month for some companies, a week for others, and a day in some cases), the file of transactions (called "**trans.txt**") is applied to the master file (called "**oldmast.txt**") to update each account's purchase and payment record.

During an update, the master file is rewritten as the file "**newmast.txt**", which is then used at the end of the next business period to begin the updating process again.

File-matching programs must deal with certain problems that do not arise in single-file programs.

For example, a match does not always occur. If a customer on the master file has not made any purchases or cash payments in the current business period, no record for this customer will appear on the transaction file. Similarly, a customer who did make some purchases or cash payments could have just moved to this community, and if so, the company may not have had a chance to create a master record for this customer.

Write a complete file-matching accounts receivable program. Use the account number on each file as the record key for matching purposes. Assume that each file is a sequential text file with records stored in increasing account-number order.

a) Define class **TransactionRecord**. Objects of this class contain an account number and amount for the transaction. Provide methods to modify and retrieve these values.
b) Define a class **AccountRecord** object of this class contains account number, first name, last name, account balance. Provide methods to modify and retrieve these values.

c) Include method *combine* in **AccountRecord** class, which takes a **TransactionRecord** object and combines the balance of the Account object and the amount value of the TransactionRecord object.
d) Write a program to create data for testing the program. Use the sample account data in Table 1 and Table 2.
e) Run the program to create the files trans.txt and oldmast.txt to be used by your file-matching program.
f) Create class **FileMatch** to perform the file-matching functionality. The class should contain methods that read *oldmast.txt* and *trans.txt*. When a match occurs (i.e., records with the same account number appear in both the master file and the transaction file), add the dollar amount in the transaction record to the current balance in the master record, and write the "newmast.txt" record. (Assume that purchases are indicated by positive amounts in the transaction file and payments by negative amounts.) When there is a master record for a particular account, but no corresponding transaction record, merely write the master record to "newmast.txt". When there is a transaction record, but no corresponding master record, print to a log file the message "Unmatched transaction record for account number…" (fill in the account number from the transaction record). The log file should be a text file named "log.txt".

Table 1

Master File account number	**Name**	**Balance**
100	Alan Jones	348.17
300	Mary Smith	27.19
500	Sam Sharp	0.00
700	Suzy Green	-14.22

Table 2

Transaction file account number	**Transaction amount**
100	27.14
300	62.11
400	100.56
900	82.17

Workshop Header

/***
Workshop #
Course:*<subject type> - Semester*
Last Name:*<student last name>*
First Name:*<student first name>*
ID:*<student ID>*
Section:*<section name>*
This assignment represents my own work in accordance with Seneca Academic Policy.
Signature
Date:*<submission date>*
***/

Code Submission Criteria:

Please note that you should have:

- Appropriate indentation.
- Proper file structure
- Follow java naming convention.
- Document all the classes properly
- Do Not have any debug/ useless code and/ or files in the assignment

Deliverables and Important Notes:

All these deliverables are supposed to be uploaded on the blackboard once done.

- You are supposed to submit a video/ detailed document of your running solution. **(40%)**
    - Things to be considered if you are uploading a video.
        - A reasonable length of video should be posted. 5%
        - Your video should show the running solution with different inputs. 5%
        - In a minute discuss the design of your solution. 10%
        - In a minute (max 3 to 5) discuss the important functions/ methods in your solution. 20%
        - If you are using in your solution concepts that are not discussed in the class then in a minute or two explain,
            - What is that concept?
            - Why did you use it?

        - How does it benefit your solution?
    - Things to be considered if you are uploaded the detailed document.
        - Should include **screen shots** of your output. 5%
        - Underneath each screen shot explain in 2 to 4 lines what is happening. 10%
        - In 3 to 5 lines explain the design logic of your program. 10%
        - Screen shots of important methods/ functions in your solution and discuss them underneath each screen shot (3 to 6 lines). 15%
        - If you are using in your solution concepts that are not discussed in the class then in a minute or two explain,
            - What is that concept?
            - Why did you use it?
            - How does it benefit your solution?

- A word/ text file which will reflect on learning of your concepts in this workshop. (Also include the instructions on how to run your code, if required) **(30%)**
    - Should state your Full name and Id on the top of the file and save the file with your last name and id, like Ali_123456.txt
- Submission of working code. **(30%)**
    - Make sure your follow the "**Code Submission Criteria"** mentioned above.
    - You should zip your whole working project to a file named after your Last Name followed by the first 3 digits of your student ID. For example, **Ali123.**zip.
- Your marks will be deducted according to what is missing from the above-mentioned submission details.
- Late submissions would result in additional 10% penalties for each day or part of it.
- Remember that you are encouraged to talk to each other, to the instructor, or to anyone else about any of the assignments, but the final solution may not be copied from any source.