Queens College, CUNY, Department of Computer Science

Software Engineering

CSCI 370

Summer 2019

Instructor: Dr. Sateesh Mane

Demo of GUI in class, Monday 8/5/2019 – Thursday 8/8/2019 (see below)

Zip archive due by 11:59 pm on the day before your in-class demo

2 Project 2

- **All students work in teams for this project.**
  1. **Every team member must be able to answer questions on all aspects of the project.**
  2. During a demo, I may call on any team member to answer questions about the project.
  3. Code may be written in any language of your team's choice.
  4. *This project requires a database ("persistence of data across invocations").*
- Every student must submit their solution via email, as a file attachment, to `Sateesh.Mane@qc.cuny.edu`. The file attachment is a zip archive with the following naming format:

  `StudentId_first_last_CS370_project2_Summer2019.zip`

- **Important: Do not encrypt the zip. Other formats such as RAR archive or OneDrive or Google Drive, etc. will not be accepted.**
  1. A cover document explaining what your team has done. It is acceptable if the cover document is the same for all members of the team (and was written jointly), but it is also acceptable to include your personal perspective.
  2. Include screen shots of relevant pages (create a new account, billing statement, etc.).
  3. *Low resolution images (small file size) are acceptable.*
  4. **A separate document with a log detailing the contributions by each team member to the project.**
  5. **The log file must be the same for all members of the team.**
     (a) The log file should be in the form of a timeline.
     (b) Each item in the file should be tagged with a date, the name of the contributor and a summary of the contribution, e.g. made plan for overall architecture, added file for GUI, bug fix (of something), performed unit tests, etc.

Date	Name	Contribution
⋮	⋮	⋮
⋮	⋮	⋮

2.1 Grading policy

- The lectures during the week Monday 8/5/2019 through Thursday 8/8/2019 will be reserved for team demos. During one of the above lectures, your team must present a demo in class, of the team's implementation of the project.

  1. **All team members must be present at the demo.** (Unless a valid medical, etc. reason is supplied to me ahead of time.)
  2. I will ask questions and/or run my own tests of the team demo in class. *Every team member must be able to answer all questions pertaining to the team submission.* **Students who cannot explain the team's work to me personally in class will receive a reduced grade relative to their teammates.**
  3. On each day Monday–Thursday 8/5/2019–8/8/2019, at the start of the lecture I shall take a head count of all the teams who wish to demo their project that day. The time available for each team demo will be equally divided between all the teams presenting on that day. If too many demos are presented on the last day (for example), there may be little time for each demo, which could result in demos missing grade points for parts that were not presented/tested.
  4. The date on which a team presents its demo will be recorded. A team which presents a demo on an earlier day may receive a higher grade than a team which presents a demo of equal quality but on an later day.
  5. Every team member must email their zip archive to me not later than 11:59 pm on the day before your team presents its demo in class. (That is to say, you are not permitted to make changes to your project after the demo in class.) **Students who submit their zip archives after the presentation of the team demo in class will receive a grade cap of C, regardless of their contributions to the project.**
  6. **A team may present its demo only once.** A team is not permitted to present a repeat demo with bug fixes or enhancements, etc. *The project submission is expected to be complete and correct at the time of the team demo in class.*
  7. **Your grade will be determined principally by your team presentation in class.** All team members will receive the same grade for the project, subject to caveats described in the grading policy.
  8. I will use your zip archive to validate the demo in class and I will also check the log file.
  9. **Teams who do not present a demo in class will receive a failing grade, regardless of the quality of the work submitted in the student zip archives.**
     It is not acceptable to submit a zip archive and not present a demo in class.

- **Any attempt to interfere with the presentation of demos by other teams will be treated as a serious offense and will incur a serious grade penalty, including a failing grade.** In this context, "interfere" includes the following:

    1. Attempts to obtain more than your allotted time to present your demo, thereby denying other teams their fair share of time.
    2. Making comments on the work by other teams and/or interrupting conversations between myself and other students when I examine the work of other students.
    3. "Cutting the line" if I call upon another team to present their demo ahead of your team.
    4. Creating a disturbance in the class, which obstructs my ability to examine all team demos fairly.

2.2 Project specification

- This project is to create a simple model of a credit card system.
- The "agents" in the system are (i) cardholders, (ii) shops/vendors, (iii) credit card company.
- Cardholders
    1. Cardholders have accounts and make purchases at the shops ("vendors") and make payments to the credit card company.
    2. Cardholders can view their account (password protected) to see a list of their purchases and payments and amount due.
- Shops/vendors
    1. Shops/vendors have items for sale which cardholders buy using credit cards.
    2. If the credit limit on a credit card account is exceeded, the transaction is denied.
- Credit card company
    1. The credit card company is basically an administrator (password protected).
    2. The credit card company can create and shut down cardholder accounts.
    3. The credit card company can view any cardholder account.
    4. The credit card company can view all the purchases made at a vendor, by the various cardholders.

2.3 Cardholder

- A cardholder owns one or more credit cards.
- A credit card account has a name and number.
- **There must be at least two cardholders.**
  At least one cardholder must have two or more credit cards.
- The name is an alphanumeric string, no minimum length but cannot be blank.
- **The credit card number must be unique (let us say 4 digits for simplicity).**
- **The cardholder accesses a GUI to set the account password.**
  1. The password is alphanumeric, minimum 6 characters, but let us not bother with "strong password" and special characters, etc.
  2. Purchases cannot be made until the password is set.
  3. Effectively, the account is "activated" by setting the password.
- The GUI for an account statement displays a list of purchases (sorted by date, with the most recent at the top) and a list of payments.
- **Access to the account statement is password protected.**
- At the top of the GUI, display the account name, card number, credit limit, amount due.
- By default, the GUI should only display the most recent payment and the list of new purchases made after the latest payment.
- The GUI should have a button "View all" to display all payments and the list of all purchases (sorted by date with the most recent at the top).

Account Statement		
Account name:		
Card number:		
Credit limit:		
Amount due:		
Recent purchases:		
date	amount	description
⋮	⋮	⋮
⋮	⋮	⋮
Payments:		
date	amount	"thank you"

- **There must also be a GUI (or some mechanism) for a cardholder to make payments for an account.** This need not be password protected: anyone can "write a check" to pay money into an account.

2.4 Shop/vendor

- I employ the term "vendor" because some companies are online businesses only, without physical buildings for customers to visit.
- **The project implementation must have at least three vendors:**
  1. Supermarket
  2. Restaurant
  3. Department store, e.g. general store such as Walmart, else a hardware store, etc.
  4. *You may add other vendors, e.g. online bookstore, etc.*
- **A vendor's name must be unique.**
- **To avoid copyright problems, do not use names of real companies.**
  Also do not have vendors which sell military items or other restricted material.
- The implementation for a vendor is essentially a single GUI.
- No password is required to access a vendor (no password is required to enter a shop).
- The vendor GUI displays a selection of items to purchase.
  There should be a minimum of five items to buy.
- The cardholder selects items to purchase ("shopping cart"), default quantity = 1.
- The cardholder clicks a "Buy" or "Proceed to checkout" button.
- The total amount to pay is displayed in the GUI.
- The cardholder enters a credit card number and clicks a "Pay" button.
- If the credit limit on the account is exceeded, the transaction is denied (error message).
- If the transaction is successful, a suitable message is displayed. The credit card account is updated with the date, amount and description of the purchase.
- **After a successful transaction, clear the credit card information, etc. from the GUI.**
- **There must be a "Cancel" button or equivalent in case the cardholder decides not to proceed to the checkout.**
- **It is acceptable to implement all the vendor functionality in a single GUI.**
- Neglect "cash back" hence the amount charged is the total cost of the purchases.
- Assume the vendor never runs out of stock so all items are always available to buy.
- We assume the credit card company pays the vendor in full immediately for all successful transactions, hence we do not require functionality to display any money owed to the vendor.
- We assume there are no "returns" of purchased items, hence there are no refunds to cardholder accounts.

2.5 Credit card company

- In this project, there is only one credit card company.
- **An administrator manages the company and access is password protected.**
- The administrator does not need to have a name, just a password.
- **The password is "admin" and can be hardcoded in the database.**
- The administrator performs multiple functions (see below).
- It is your responsibility to implement one or more GUIs to support the required functionality.

2.5.1 Create/delete account

- In real life, people apply to a company to obtain a credit card.
- In this project, the administrator creates an account with (name, number, credit limit).
- The administrator can change the credit limit of an account.
- The administrator can close an account.

2.5.2 View cardholder accounts

- The administrator can view a list of cardholder accounts (name & number).
- The list is sorted lexicographically by the names on the cardholder accounts.
- The administrator can click on any account and view all the transactions in that account.
- The transaction GUI should display the account name and number, credit limit and a list of all purchases (sorted by date, most recent at the top), also all payments (sorted by date, most recent at the top).

2.5.3 View vendors

- The administrator can view a list of vendors (sorted lexicographically by the vendor names).
- The administrator can click on any vendor and view all the transactions for that vendor, by all the cardholders.
- The transaction GUI should display the vendor name and a list (date, amount, cardholder name, cardholder number) sorted by date, most recent at the top.

2.6 Persistence across invocations

- I will enter data during the team demo in class.
- Every team will be asked to shut down and restart its project during the demo in class.
- **When the project is shut down and restarted, all saved data must be present on the restart.**

2.7 Dates of transactions

- I will test your demo by entering data and navigating the various GUIs.
- However, because the demo will be presented in class on a single day, logically the dates of all the transactions (purchases and payments) entered in class will all be on the same day.
- This does not provide a good test of the software.
- **Find a way to configure your project so that transactions entered at different times (separated by a few minutes) are assigned to different dates.**

  1. Sort order: if transactions are entered at real-world times $(t_1, t_2, t_3, \dots)$ and $t_1 < t_2 < t_3 < \dots$, then the displayed dates $(d_1, d_2, d_3, \dots)$ should be sorted in the same order $d_1 < d_2 < d_3 < \dots$.
  2. The real-world time span for a demo might be approximately 20 minutes.
  3. It will be good if the total time span of the posted dates $(d_1, d_2, d_3, \dots)$ is more than one month, i.e. the demo displays multiple monthly billing cycles.
     Projects which achieve this feature will score a higher grade.

- **Project submissions where all the transactions I enter during the demo in class all have the same date will score a reduced grade.**