INCOMPLETE

CST3613 – Homework #3

Problem Statement & Application Overview

Homework Assignment # 3 (Page 01 of xx)

Problem Statement

- **CREATE** a **`CLIENT/SERVER Console Application`** to implement a ***Business Application*** to be used by **Employees** of an organization or enterprise. In this version, the application will only include the following functionalities:

1. **Security Access Authentication** – In order to use the application, employees must login to the system. Only authenticated employees are allow access to the application. All other users must be denied access to the application.

2. **Main/Welcome Screen** – Once and authenticate user is allowed into the system, a MAIN or WELCOME SCREEN should display allowing the user to select from the following application options:

   1) Back-end Management
   2) Retail Point-of-Sales
   0) Exit
   - **Selecting either option 1 will DISPLAY a text-based BACK-END MANAGEMENT SCREEN.**
   - **Selecting option 2 will display a text-based RETAIL- POINT OF SALES SCREEN.**
   - **Selecting option 0 will RETURN BACK TO THE LOGIN SCREEN**

3. **Back-end Management Screen** – Users that enter this section will be presented with a BACK-END MANAGEMENT SCREEN following:

   1) User Account Management
   0) Exit
   - **Selecting either option 1 will DISPLAY a text-based USER ACCOUNT MANAGEMENT SCREEN.**
   - **Selecting option 0 will RETURN THE USER BACK THE BACK-END MANAGEMENT SCREEN**

4. **Retail Point-of-Sales** – Users that enter this section will be presented with the following:

   1) Register New Customer
   0) Exit
   - **Selecting either option 1 will DISPLAY a MESSAGE indicating that this feature is UNDER CONSTRUCTION.**
   - **Selecting option 0 will RETURN THE USER BACK THE MAIN/WELCOME SCREEN**

5. **User Account Management** – Users that enter this section will be presented with the following:

   1) Search
   2) Add
   3) Edit
   4) Delete
   5) Change Username
   6) Change Password
   7) Change Email
   0) Exit

   - **Selecting each option will invoke the necessary operation including returning back to the BACK-END MANAGEMENT SCREEN.**

- This program is to be an ***OBJECT-ORIENTED PROGRAM*** that will leverage ALL THE FEATURES of the **Employee Login Authentication Application** ***IMPLEMENTED*** & ***TESTED*** in **HW#2**.
- Details are listed in requirements sections below

Requirements #1 – Using NetBeans Create a Console Application & Leverage all Authentication functionalities of HW2

- Create a **`Console Application`** using NetBeans & include a Main Class:
  - This application will be a text based application. No graphical user-interface
  - The application will contain all processing & class required to implement the system.

- One important feature of this application is the security access implemented via an authentication system.
- This Employee authentication system has the following requirements:
  - **Security Access Authentication** – In order to use the application, employees must login to the system. Only authenticated employees are allow access to the application. All other users must be denied access to the application.

- These requirements were implemented in HW2, therefore leverage all the code from HW2 including:
  - **UserAccount Class** – represents the user account.
  - **Main() method code** – main method contained code which control the flow of the program. This code will also be leverage here, but modified based on requirements of this application.
  - **Authenticate(U,P) method** – The processing method that authenticated the users. This code will also be leverage here, **but modified based on requirements of this application**.
  - **ARRAY of UserAccount Objects** – The ARRAY which stored the **UserAccount** Objects will be ***REPLACED*** by an OBJECT of a CLASS THAT ENCAPSULATES THE ARRAY.
  - **Any other feature of HW2 that DOES NOT CONFLICT WITH THE REQUIREMENTS OF THIS APPLICATION**.

Requirements #2 – Programming & Algorithm Requirements

- **Requirements 2a – YOU ARE ONLY TO USE THE LANGUAGE COMPONENTS WE HAVE LEARNED UP TO THIS POINT**:

  - Do not use any advanced features or other language components from future lectures or previous programming course you took, **DON'T USE OTHER LANGUAGE STRUCTURES WE HAVE NOT YET COVERED. ONLY WHAT WE HAVE COVERED UP TO THIS POINT** IN CLASS.

- **Requirements 2b – YOU'RE ALGORITHM & APPLICATION WILL BE GRADED ON BEST APPROACH TO IMPLEMENTING THIS PROBLEM BASED ON THE FOLLOWING RULES:**

  - **BEST TOOL** – Selection of best tool for each task:
    - What I mean by best tool is the programming language components or flow chart/algorithm tool (if, if/else, nested if/else, while loop, for loop, data structures such as variable, arrays, etc.)
    - When selecting a tool, keep in mind organization, efficiency etc. (in other words, you will not use a hand saw to cut a tree, if you have a power saw, same apply here, you will NOT use a number of string variables to store a list of items when you have a better tool such as an array of strings)

  - **EFFICIENT/PERFORMANCE** – Program/design should be efficient or with optimal performance :
    - Limit unnecessary processing where possible. Consider CPU & MEMORY usage.
    - Limit any additional or unnecessary steps that would require necessary processing by the CPU.
    - Limit any additional or unnecessary MEMORY Data Structures, example unnecessary variables etc.

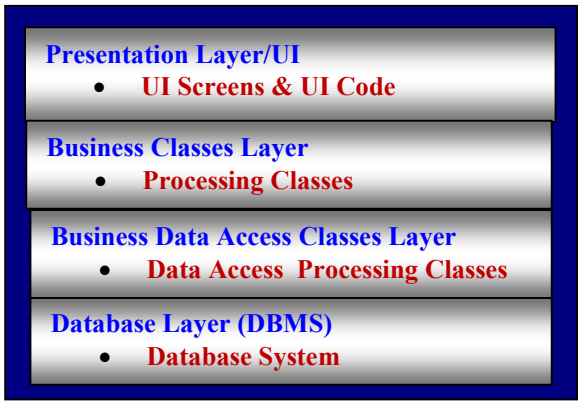
  - **SCALABLE** – design and implement so that program can be SCALED and GROW with FEW LIMITATIONS & CHANGES to other systems:
    - (Don't go crazy here, just make it so that is realistic, for example, only 5 users are now available, but program should provide the flexibility that if more users need to be added we can do it without having to re-engineer the solution)

  - **MODULAR** – program should be implemented using tools to make it modular, this works hand in hand with scalability:
    - Program should be written in parts or modular. In other words mayor functionalities and features should be enclosed within block of code that can be called when needed.
    - When deciding which block of code to modularize, keep in mind **scalability**. Future upgrades of features should only require swapping these modular code segments with a new one and still keep the same structure.
    - Each modular block of code should only perform ONE ATOMIC OPERATION when possible. That is you don't want to print and authenticate at the same time. HINT: For example, the module of code that authenticates a username and password should NOT also display the "Welcome Access Granted" or "**Access Denied**". This block of code should only authenticate!!!

Requirements #3 – Implement Application using a LIMITED Client/Server Development Design Pattern

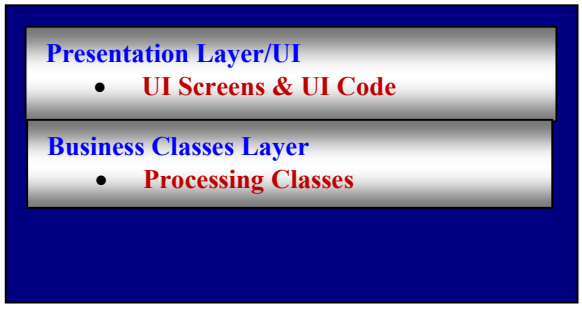
- The goal is to implement this application as a distributed network application. This is accomplished by implementing based on a Client/Server design pattern.
- Objectives are as follows:

  - **MAIN OBJECTIVE** is to take **FIRST STEP** in **CREATING** a **Client/Server Architecture** Application using the following ***4-tiered Client/Server Application Architecture***:



4 Tiers Windows Client/Server Application Architecture

  - BECAUSE WE DON'T HAVE a DATABASE LAYER to permanently store our data at this time, Our FOCUS will be on implementing the ***Presentation/User-Interface Layer*** & ***Business Classes Layer*** FIRST. Therefore you objectives are to create this INTERMEDIATE ARCHITECTURE in **HW3**:

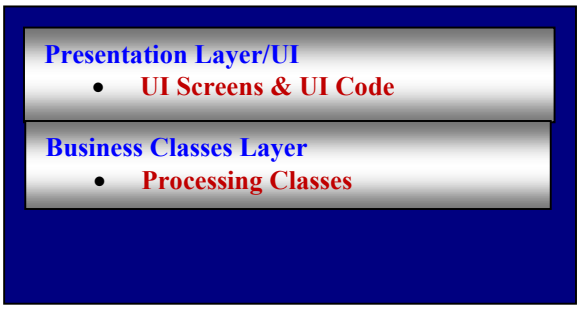


Limited Intermediate Windows Client/Server Application Architecture

Requirements #4 – Create the required Project & Folder Structure to Implement the Client/Server Development Design Pattern

- Create the application using the correct project types and folder structure to meet the Client/Server development requirements.
- Objectives are as follows:

  - **MAIN OBJECTIVE** is to take **FIRST STEP** in **CREATING** a **Client/Server Architecture** Application implementing the ***Presentation/User-Interface Layer*** & ***Business Classes Layer*** FIRST:



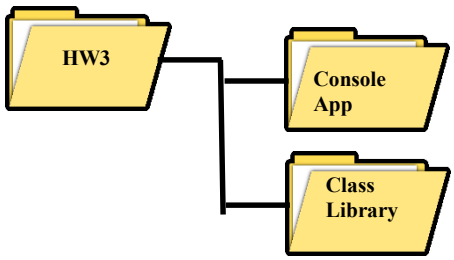
Limited Intermediate Windows Client/Server Application Architecture

- **REQUIREMENTS 4a** – Using your computer FILE MANAGEMENT tool, CREATE an **APPLICATION FOLDER** to ***HOST*** **YOUR APPLICATION PROJECT FILES**.

  - Name this **APPLICATION FOLDER** as you like, for example HW3, etc.

1. Name this **APPLICATION FOLDER** as you like, for example HW3, etc a

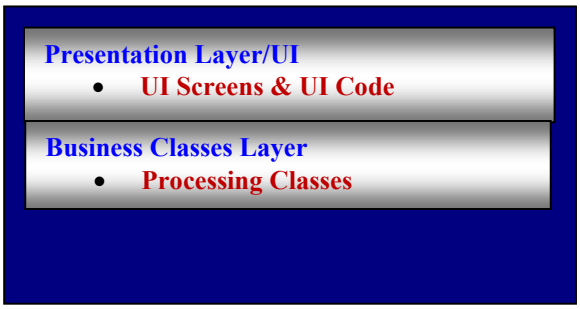
2. After all your **Console Application** & **Class Library Projects** are created, the application folder structure should look as follows where the **APPLICATION FOLDER** is your ROOT directory:



Requirements #4 (Cont.) – Create the required Projects & Folder Structure to Implement the Client/Server Development Design Pattern

- We continue with requirements #4:

  - **AGAIN THE MAIN OBJECTIVE** is to CREATE the following **Client/Server Architecture** Application implementing the ***Presentation/User-Interface Layer*** & ***Business Classes Layer*** FIRST:



Limited Intermediate Windows Client/Server Application Architecture

  - **REQUIREMENTS 4b** – Implement the **`Presentation/UI Layer`** by CREATING a **`Console Application`** as dictated by **Requirements #1** and as dictated by **CLIENT/SERVER IMPLEMENTATION REQUIREMENTS INSIDE THE APPLICATION FOLDER**:

    - The **key technologies** that will allow us to implement ***Presentation/UI Layer*** in Client/Server windows applications are:

    1. **Programming Language** – Java
    2. **NetBeans Project Type – Java Application Project** (Console Application or Windows GUI Application)

  - **REQUIREMENTS 4c** – Implement the **`Business Classes Layer`** by CREATING a **`Class Library Project`** to HOST ALL BUSINESS CLASSES as dictated by **CLIENT/SERVER IMPLEMENTATION REQUIREMENTS INSIDE THE APPLICATION FOLDER**:

    - The **Java key technologies** that will allow us to implement ***Distributed Business Objects*** are:

    1. **Programming Language** – Java
    2. **Java Archive (Jar) – Business Classes need to be packaged as a Class Library or JAR (Java Archive)** in order for them to be portable. **This is the first step in creating distributed objects**. You can package one or several classes into one JAR container
    3. **NetBeans Project Type – Class Library Project** to generate the JAR file **(No main class or main() function)**

OOP Step 1 – UPGRADE the UserAccount

Homework Assignment # 3 (Page 03 of 14)

Requirement #5 – UPGRADE the Tested UserAccount Class from HW1 & HW2 to meet the Requirements for a Business Class by adding Data Access Methods

- The custom UML diagram below illustrates the requirements for the **NEWLY UPGRADED UserAccount Class**:

UserAccount

```
Private Data:
userAccountID:              UUID
username:                   String
password:                   String
email:                      String

Public Getter/Setter Methods/Properties:
Read Only getUserAccountID():       String
getUsername():                      String
setUsername(value):                 void
getPassword():                      String
setPassword(value):                 void
getEmail():                         String
setEmail(value):                    void

Private Constructor Methods:
private UserAccount()
private UserAccount ( U, P, E)

Public Methods:
public static getInstance():        UserAccount
public static getInstance(U,P,E):   UserAccount
public Authenticate(U, P):          boolean

Public Data Access Methods:
public void load(key:               void
public void insert():               void
public void update():               void
public void delete():               void

Public Data Access Methods:
protected void database_Load(key):  void
protected void database_Insert():   void
protected void database_Update():   void
protected void database_Delete():   void
```

How to Re-use a Class from a Previous Project to NEW Project

- ADD the **UserAccount Class** of **HW#1** to this **`Console Application`** PROJECT:

  - To ADD the **UserAccount.java Class** to this **HW#3** Project take the following steps:

    1. Navigate to **HW#1 Project folder & navigate to the SRC folder where the UserAccount.java Class File is located**.
    2. ***Right-Click*** & **COPY** the **UserAccount.java Class File**.
    3. Navigate to this **HW#3 Project folder & navigate to the SRC.** ***Right-Click*** & **PASTE** the **UserAccount.java** Class File to the Folder.
    4. The **UserAccount** Class should automatically appear in the **`Project Windows`** under the **`Package`** for this **HW#3 Project**.

Homework Assignment # 3 (Page 01 of 14)

- **REQUIREMENTS 5a –** To the **EXISTING UserAccount** Class of **HW#2 & 3 *ADD*** the following **Public** Data Access Methods:

Scope	Name	Return Type	Parameters	Description
public	load	**void**	**String**	▪ Public Data access method that starts the process of FETCHING data from database. ▪ Pass the KEY or Unique ID of the DATABASE RECORD to LOAD and it will perform the database access. ▪ **Process it performs**: Calls `protected database_Load(key)` to do the work
public	insert	**void**	**None**	▪ Public Data access method that ADDS a record to the database. ▪ **Process it performs**: Calls `protected database_Insert()` to do the work
public	update	**void**	**None**	▪ Public Data access method that UPDATES data in the database. ▪ **Process it performs**: Calls `protected database_Update()` to do the work
public	delete	**void**	**None**	▪ Public Data access method that DELETES a record from database. ▪ **Process it performs**: Calls `protected database_Delete(key)` to do the work

- **REQUIREMENTS 5b –** To the **EXISTING UserAccount** Class of **HW#2 & 3 *ADD*** the following **Protected** Data Access Methods:

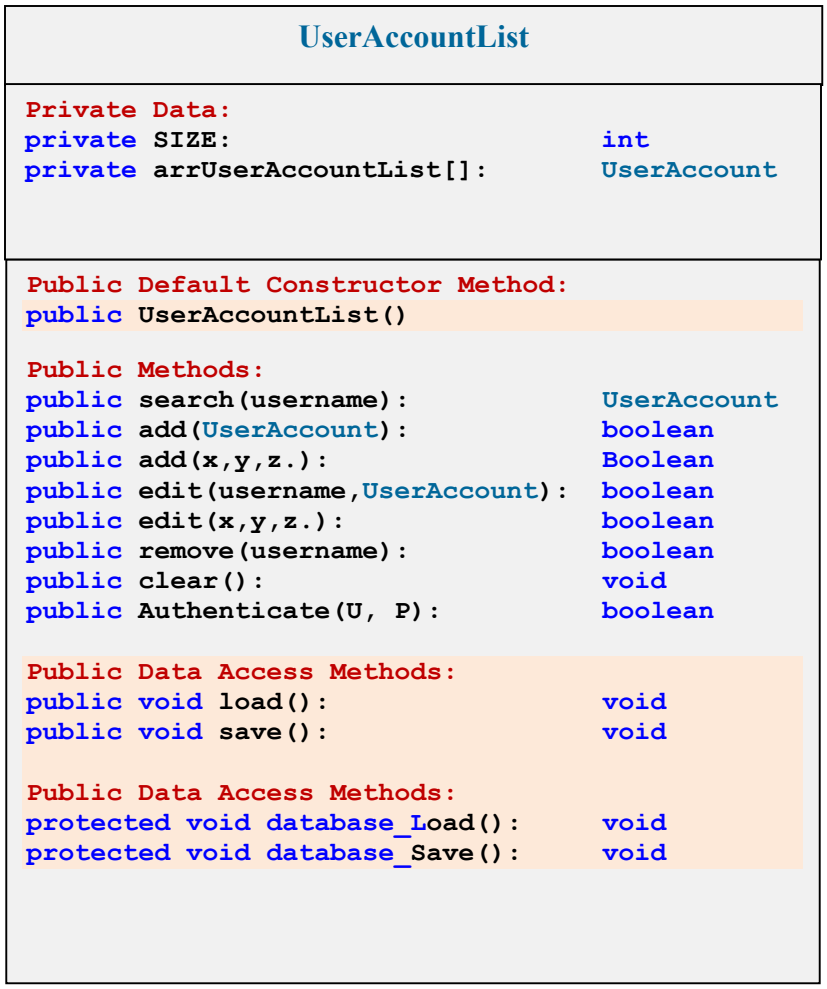
Scope	Name	Return Type	Parameters	Description
protected	Database_Load	**void**	**String**	▪ Protected Data access method that actually performs the FETCHING or RETRIEVAL of data from database & populates the object with data retrieved from database. ▪ **Process it performs**: - **STUB METHOD or NOT IMPLEMENTED**. - Targeted for future implementation (leave empty with required syntax to satisfy the compiler)
protected	Database_Insert	**void**	**None**	▪ Protected Data access method that actually performs the INSERTING/ADDING of a record to database. ▪ **Process it performs**: - **STUB METHOD or NOT IMPLEMENTED**. - Targeted for future implementation (leave empty with required syntax to satisfy the compiler)
protected	Database_Update	**void**	**None**	▪ Protected Data access method that actually performs the UPDATING of a record in the database. ▪ **Process it performs**: - **STUB METHOD or NOT IMPLEMENTED**. - Targeted for future implementation (leave empty with required syntax to satisfy the compiler)
protected	Database_Delete	**void**	**None**	▪ Protected Data access method that actually performs the DELETION of a record from database. ▪ **Process it performs**: - **STUB METHOD or NOT IMPLEMENTED**. - Targeted for future implementation (leave empty with required syntax to satisfy the compiler)

OOP Step 1 – Create a UserAccountList Class

Homework Assignment # 3 (Page 03 of 14)

Requirement #6 – CREATE a UserAccountList Class f Client/Server Business Class to Manage & Encapsulate the ARRAY of UserAccount Objects

- The custom UML diagram below illustrates the requirements for the **NEW UserAccountList Class**:



Homework Assignment # 3 (Page 01 of 14)

- **REQUIREMENTS 6a –** **UserAccountList** Class **PRIVATE DATA**:

Scope	Data Member Name	Type	Description
`private`	`SIZE`	`int`	▪ Stores the SIZE of the ARRAY. MUST BE A CONSTANT variable of size 10
`private`	`arrUserAccountList[]`	`UserAccount`	▪ POINTER DECLARATION of ARRAY of type **UserAccount** Class

- **REQUIREMENTS 6b –** **UserAccountList** Class **DEFAULT CONSTRUCTOR**:

Scope	Name	Return Type	Parameters	Description
`public`	`UserAccountList()`	`NA`	`none`	▪ DEFAULT CONSTRUCTOR – Initializes the `arrCustomerList` **POINTER** by ***CREATING*** an ARRAY OBJECT & assigning to `arrCustomerList[]` POINTER.

Homework Assignment # 3 (Page 01 of 14)

- **REQUIREMENTS 6c –** **UserAccountList** Class **PROCESSING METHODS**:

Scope	Name	Return Type	Parameters	Description
public	search	**UserAccount**	**String username**	▪ ***SEARCH*** **`arrUserAccountList`** **ARRAY** – Method that performs a search of the ARRAY for the **UserAccount object** whose **username** is passed as parameter. ***RETURNS*** a **POINTER** to the object found or returns a NULL if not found.
public	add	**boolean**	**UserAccount POINTER**	▪ ***ADD*** **OBJECT** to **`arrUserAccountList`** **ARRAY** – Method that ***SEARCHES*** the ARRAY for a **NULL POINTER** (empty cell) and ***ADDS*** OBJECT by having it ***POINT*** to the NEW OBJECT passed as parameter. Returns a *TRUE* if empty pointer found and object added, else *FALSE* if no room found in ARRAY.
public	add	**boolean**	Values with appropriate type that make up a **UserAccount** Object: **String username, String *password* & String *email***	▪ ***(OVERLOADED) ADD*** **OBJECT** to **`arrUserAccountList`** **ARRAY** – Method ***CREATES*** a new **UserAccount object**, populates the object with values passed as parameter, then ***SEARCHES*** the ARRAY for a **NULL POINTER** (empty cell) and ***ADDS*** OBJECT by having it ***POINT*** to the NEW OBJECT passed as parameter. Returns a *TRUE* if empty pointer found and object added, else *FALSE* if no room found in ARRAY
public	edit	**boolean**	**String username, UserAccount POINTER**	▪ ***EDITS*** **OBJECT** in **`arrUserAccountList`** **ARRAY** – Method Search ARRAY for OBJECT whose **username** is passed as parameter. If found in ARRAY, object in ARRAY is ***MODIFIED*** by SETTING its PROPERTIES (**username**, **password** & **email**) ***EXCEPT*** the **userAccountID UUID** with VALUES from OBJECT passed as parameter. Returns a **TRUE** if object found and REPLACED. Returns **FALSE** if object not found in ARRAY.
public	edit	**boolean**	Values with appropriate type that make up a **UserAccount** Object: **String username, String *password* & String *email***	▪ ***(OVERLOADED) EDITS*** **OBJECT** in **`arrUserAccountList`** **ARRAY** – Method Search ARRAY for OBJECT whose **username** is passed as parameter. If found in ARRAY, object in ARRAY is ***MODIFIED*** by SETTING its PROPERTIES (**username**, **password** & **email**) ***EXCEPT*** the **userAccountID UUID** with VALUES passed as parameters. Returns a **TRUE** if object found and MODIFIED. Returns **FALSE** if object not found in ARRAY
public	remove	**boolean**	**String username**	▪ ***REMOVES*** **OBJECT** from the **`arrUserAccountList`** **ARRAY** – Method Search ARRAY for OBJECT whose **username** is passed as parameter. REMOVES OBJECT from ARRAY by setting **`arrUserAccountList[i]`** POINTER to NULL. RETURNS a TRUE if found and REMOVED or returns FALSE otherwise

Homework Assignment # 3 (Page 01 of 14)

- **REQUIREMENTS 6c (cont.) – UserAccountList Class PROCESSING METHODS**:

Scope	Name	Return Type	Parameters	Description
public	changeUsername	**boolean**	**String username** **String newUsername**	▪ Method performs the Process of ***CHANGING THE USERNAME*** of a **UserAccount** Object in the **arrUserAccountList ARRAY**. The ***UserAccountID*** of the OBJECT TO BE CHANGED is passed as ***parameter*** in order to SEARCH for the OBJECT in the ARRAY, along with the ***newUsername*** variable containing the **NEW USERNAME** being changed. ▪ **NOTE THAT SEARCH IS BASED ON Username**, but then the **Username** is ***MODIFIED*** and replaced by the ***newUsername*** parameter. ▪ The Method **RETURNS** a **TRUE** if the ***USERNAME*** of the OBJECT in ARRAY is ***CHANGED*** & a **FALSE** if NOT FOUND.
public	changePassword	**boolean**	**String username** **String newPassowrd**	▪ Method performs the Process of ***CHANGING THE PASSWORD*** of a **UserAccount** Object in the **arrUserAccountList ARRAY**. The ***Username*** & ***Password*** of the OBJECT WHOSE ***Password*** TO BE CHANGED are ***BOTH*** passed as ***parameter*** to this method. ▪ **NOTE THAT SEARCH IS BASED ON Username**, but is the **Password** that will be ***MODIFIED***. ▪ The Method **RETURNS** A **TRUE** if the ***PASSWORD*** of the OBJECT in ARRAY is ***CHANGED*** & a **FALSE** if NOT FOUND.
public	changeEmail	**boolean**	**String username** **String newEmail**	▪ Method performs the Process of ***CHANGING THE EMAIL*** of a **UserAccount** Object in the **arrUserAccountList ARRAY**. The ***Username*** & ***Email*** of the OBJECT WHOSE ***Email*** TO BE CHANGED are ***BOTH*** passed as ***parameter*** to this method. ▪ **NOTE THAT SEARCH IS BASED ON Username**, but is the **Email** that will be ***MODIFIED***. ▪ The Method **RETURNS** A **TRUE** if the ***EMAIL*** of the OBJECT in ARRAY is ***CHANGED*** & a **FALSE** if NOT FOUND. ▪ Processing:

Homework Assignment # 3 (Page 01 of 14)

- **REQUIREMENTS 6c (cont.) – UserAccountList Class PROCESSING METHODS**:

Scope	Name	Return Type	Parameters	Description
public	clear	**void**	**none**	▪ ***CLEARS*** the **`arrUserAccountList`** **ARRAY** of **OBJECTS** – Method ***CLEARS*** the ARRAY of Objects by setting all pointers to **NULL**.
public	authenticate	**boolean**	**String username, String password**	▪ Performs the Authentication process of a Username & Password by ***LOADING*** & ***LINER SEARCHING*** the **`arrUserAccountList`** **ARRAY** and calling each **OBJECT'S** **`arrUserAccountList[i].authenticate(U,P)`** method. If any of the OBJECT in the ARRAY returns a TRUE from its **authenticate(U,P)** method call it returns a true. If it searches the entire array and does not get a true, then returns a false. When found ***CLEARS*** the **ARRAY** & returns a **True**. If not found, ***CLEARS*** the **ARRAY** & returns a **False**. ▪ Algorithm is as follows: **1.** Calls the ***Load()*** Method of THIS CLASS to populate the array with **`Objects`** **2.** Liner ***SEARCH*** **`ARRAY`** `(skipping empty cells)` and **Calls** EVERY **`Object's`** **`arrUserAccountList[i].authenticate(U,P)`** method to do the authentication. **3.** If object is found & authentication is true, then Calls ***Clear()*** Method to clear the **`ARRAY`**, **4.** Then **`Return True`** to EXIT the method. **5.** If object is not found by reaching the end of the search, then Calls ***Clear()*** Method to clear the **`ARRAY`** **6.** Then **`Return False`**.

Homework Assignment # 3 (Page 01 of 14)

- **REQUIREMENTS 6d –** UserAccountList Class **PUBLIC DATA ACCESS METHODS**:

Scope	Name	Return Type	Parameters	Description
public	load	**void**	**none**	▪ Public Data access method that starts the process of FETCHING all OBJECTS data from database. ▪ **Process it performs**: Calls `protected database_Load()` to do the work
public	save	**void**	**none**	▪ Public Data access method that SAVES ALL records to the database. ▪ **Process it performs**: Calls `protected database_Save()` to do the work

- **REQUIREMENTS 6e –** UserAccountList Class **PROTECTED DATA ACCESS METHODS**:

Scope	Name	Return Type	Parameters	Description
protected	Database_Load	**void**	**none**	▪ Protected Data access method that actually performs the FETCHING or RETRIEVAL of data from database & populates the object with data retrieved from database. ▪ **Process it performs**: 1) **CREATE 5 OBJECTS** of the UserAccount Class - The 5 UserAccount Class **OBJECTS** created should have the following data

UserAcccountID	Username	Password	Email
Auto Generated GUID	joe	111	jsmith@ comp1.com
Auto Generated GUID	angel	222	arodriguez@comp1.com
Auto Generated GUID	sam	333	speterson@ comp1.com
Auto Generated GUID	mary	444	mjohnson@ comp1.com
Auto Generated GUID	nancy	555	nrivera@ comp1.com

2) **ADD the 5** UserAccount **OBJECTS** to the ***arrUserAccountList*** ARRAY.

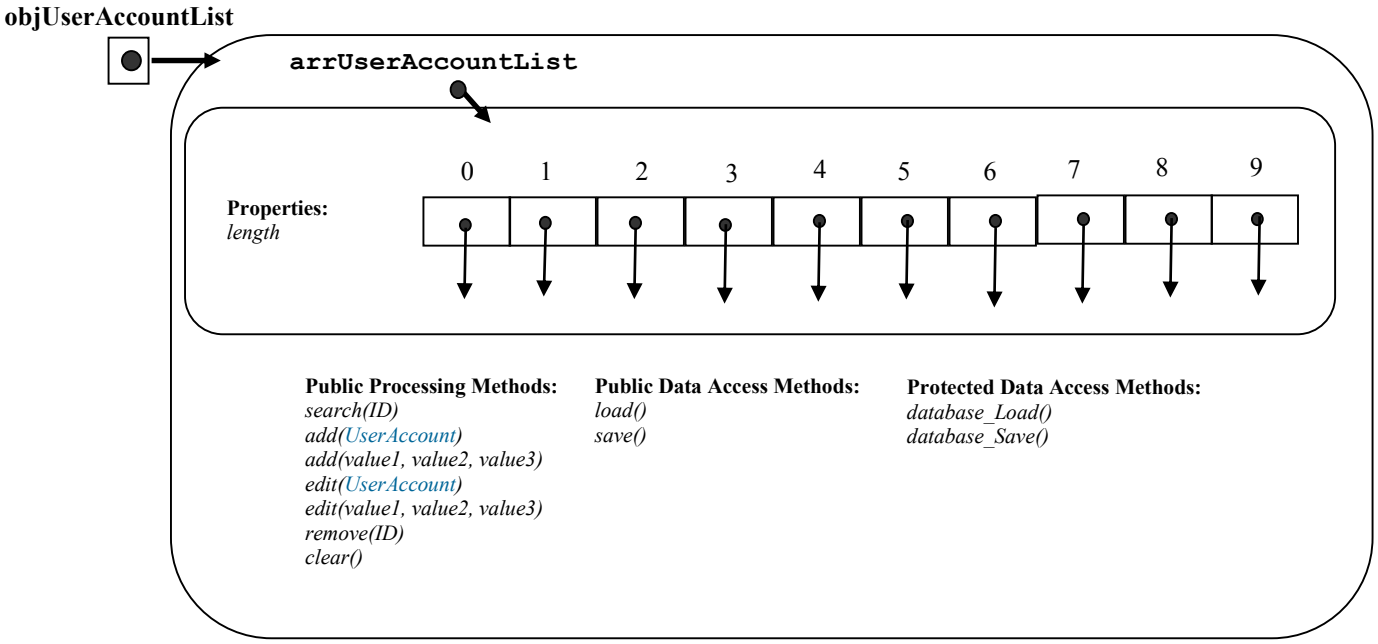
protected	Database_Save	**void**	**none**	▪ Protected Data access method that actually performs the SAVING of ALL RECORDS to database. ▪ **Process it performs**: - **STUB METHOD or NOT IMPLEMENTED**. - Targeted for future implementation (leave empty with required syntax to satisfy the compiler)

OOP Step 2, 3 & 4 (Create Objects, Use Objects & Use Class) – Main Class Requirements

Homework Assignment # 3 (Page 04 of 14)

Requirement #7 – CREATE a GLOBAL STATIC OBJECT objUserAccountList of the UserAccountList Class to manage storage of UserAccount Objects IN-MEMORY

- The company's **User Account Credential** objects will be stored in memory via a **GLOBAL OBJECT** of the **UserAccountList** Class and available to all the program code.
  - **In the MAIN CLASS, *CREATE* a GLOBAL STATIC OBJECT, named `objUserAccountList` *of UserAccountList Class:***



  - This OBJECT of the ***UserAccountList*** Class encapsulates an ARRAY of **10 Cells storing *UserAccount*** Class **POINTERS**. This is how User Accounts and will be managed & populated with ***UserAccount*** **Class OBJECTS** in **MEMORY** throughout the usage of the program.
  - **IMPORTANT! DECLARE the OBJECT OF *UserAccountList* CLASS as GLOBAL or STATIC IN THE PUBLIC DECLARATION SECTION OF THE MAIN CLASS –BEFORE THE DECLARATION OF THE** main() Method:

```
public class BusinessApplication {

//Declare Public STATCI OBJECT of UserAccountList CLASS here IN DECLARATION OF MAIN CLASS

      public static void main(String[] args) {


      }//End of main

}//End of program class
```

Requirement #8 – Main Class Driver Program (main(String[] args)) Requirements

public static void main(String[] args)

- The **main()** function inside the **`Console Application`** **Main Class** is the **DRIVER PROGRAM that CONTROLS** the flow of the application.
- You need to CREATE & design an algorithm that meets the following requirements:

Scope	Method Name	Return Type	Parameters	Processing/Description
public	**static void main ()**	**N/A**	**String[] args**	1. Displays a login menu prompting user to enter username & password & extract the username & password. 2. Once the username & password are captured, the program should call a ***FUNCTION authenticate(U, P)*** which returns a **true** if username & password are valid or a **false** if invalid (*see explanation of method in section below*). 3. From the returned value of the ***authenticate(U, P)*** Function, the program should decide if access is granted or denied. If ***authenticate(U, P)*** returns a TRUE, the ***MAIN/WELCOME SCREEN*** should display 4. If ***authenticate(U, P)*** returns a FALSE, the program should display "***Access Denied***" 5. Whether access is granted & ***MAIN/WELCOME SCREEN*** is displayed or accessed denied & display "***Access Denied***, the process should repeat displaying the login menu for the next user to login: ▪ The program always **repeats**, prompting and displaying the user login screen and the result of the login validation (display welcome screen or Access Denied!) ▪ The loop should run indefinitely until the user enters the magic value of **username = -1 and password = -1**, in which case the loop should terminate and program end. **NOTE! ONLY WHEN BOTH username & password are -1 SHOULD THE PROGRAM END. NOT EITHER ONE BUT BOTH!**

Requirement #9 – Create the STATIC Authenticate(u,p) method to Search arrUserAccountList Array of Objects & Authenticate Users

- **Requirement #6 –** CREATE a function named ***authenticate.(U,P)*** in the MAIN CLASS. **REMEMBER THAT ANY METHODS DECLARED IN THE MAIN CLASS MUST BE STATIC!** The method perform the following process:

Scope	Method Name	Return Type		Parameters	Processing/Description
public static boolean	**Authenticate (U,P)**	**boolean**	▪	▪ A parameter for storing the **username** & a parameter for storing **password**. ▪ Pass-by-Value	▪ Authenticates Username & Password as follows: 1) *Process:* **CALLS** the ***objUserAccountList*** OBJECT'S **objUserAccountList.*authenticate(U, P)*** method ***TO DO THE WORK***! 2) If **objUserAccountList.*authenticate(U, P)*** the return a **True** or is FOUND, then **RETURN** a **True** to the calling program in ***main()*** function. 3) If **objUserAccountList.*authenticate(U, P)*** the return a **False** or is NOT FOUND, then **RETURN** a **False** to the calling program ***main()*** function.

OOP Step 2, 3 & 4 (Create Objects, Use Objects & Use Class) – User-Interface Requirements

Homework Assignment # 3 (Page 09 of 14)

Requirements # 10 – User-Interface Screens Navigation Flow & Control Requirements

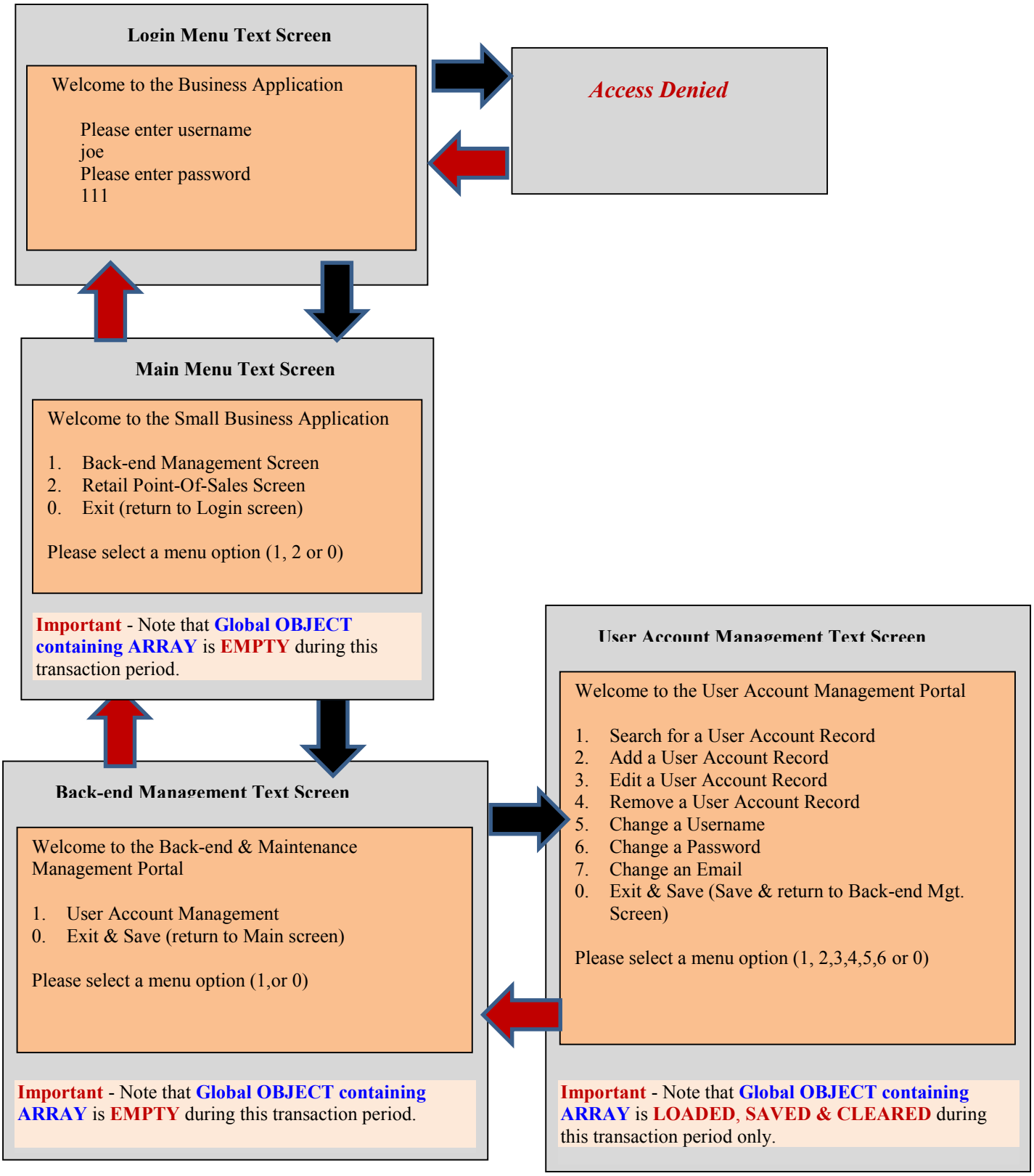
- This application contains the following user-interface screens:
  - `Login Screen`.
  - `Main/Welcome Screen`.
  - `Back-end Management Screen`.
  - `User Account Management Screen`.
  - `Retail Point-of-Sales Screen`.
- Navigation Flow and how these forms interact is described in section below.

Requirements #10a – Navigation Flow #1: Login, Main/Welcome, Back-end Management & User Account Management Screens Navigation Flow

- The **FIRST** navigation flow and control is between the following screens:
  - `Login Screen`.
  - `Main/Welcome Screen`.
  - `Back-end Management Screen`.
  - `User Account Management Screen`.
- **FORWARD & BACKWARDS Navigation Flow #1:**

  `Login Screen <-> Main/Welcome Screen <-> Back-end Management Screen <-> User Account Management Screen`
- A graphical diagram of this flow is shown in figure below:

Requirements #10a (Cont.) – Navigation Flow #1: Login, Main/Welcome, Back-end Management & User Account Management Screens Navigation Flow Diagram



Homework Assignment # 3 (Page 09 of 14)

Requirements #10b – Navigation Flow #2: User Account Management & Associated Screens Navigation Flow

- The **SECOND** navigation flow and control is between the following screens:

  - `User Account Management Screen`.
  - `Search User Account I/O & Results Screen`.
  - `Add User Account I/O & Results Screen`.
  - `Edit User Account I/O & Results Screen`.
  - `Remove User Account I/O & Results Screen.`
  - `Change Username I/O & Results Screen`.
  - `Change Password I/O & Results Screen`.
  - `Change Email I/O & Results Screen`.

- **FORWARD & BACKWARDS Navigation Flow #2 –** **C**omposed of several interactions between the User Account Screen and its Associated Screens:

  `User Account Management <-> Search User Account I/O & Results Screen`

  `User Account Management <-> Add User Account I/O & Results Screen`

  `User Account Management <-> Edit User Account I/O & Results Screen`

  `User Account Management <-> Remove User Account I/O & Results Screen`

  `User Account Management <-> Change Username I/O & Results Screen`

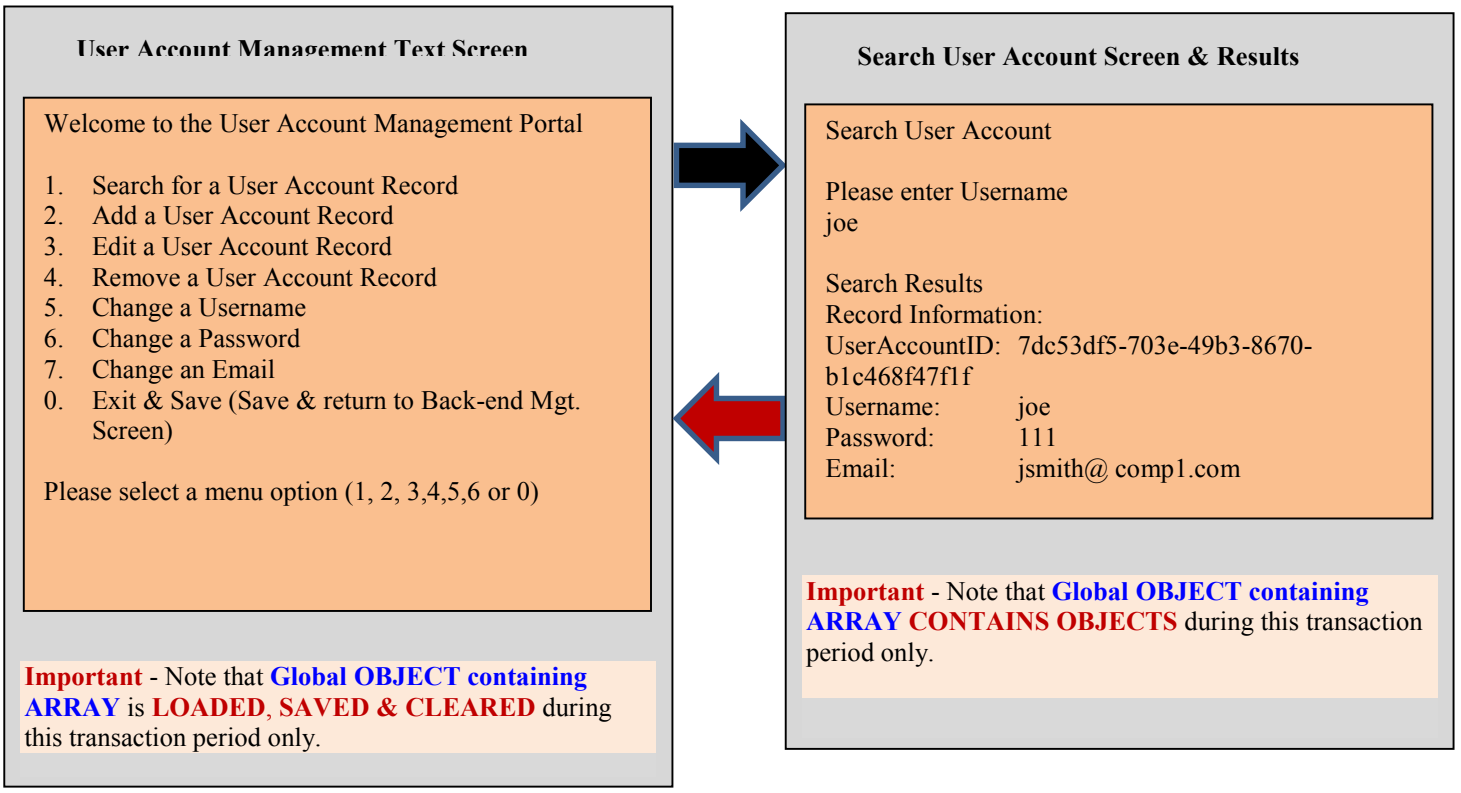
  `User Account Management <-> Change Password I/O & Results Screen`

  `User Account Management <-> Change Email I/O & Results Screen`

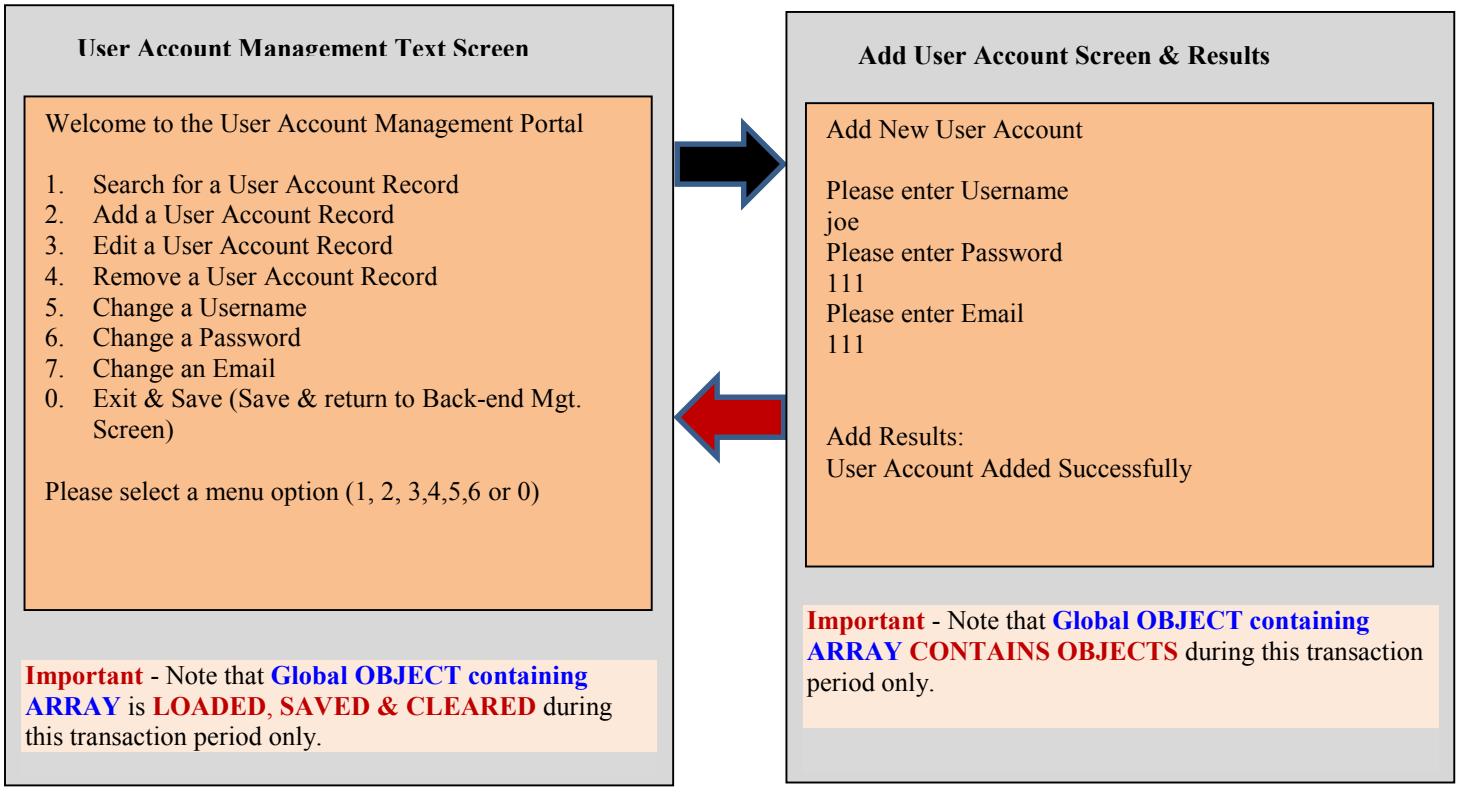
- A graphical diagram of this flow is shown in figure below:

Requirements #10b – Navigation Flow #2: User Account Management & Associated Screens Navigation Flow Diagram

Navigation Flow:
User Account Management <-> Search User Account I/O & Results Screen

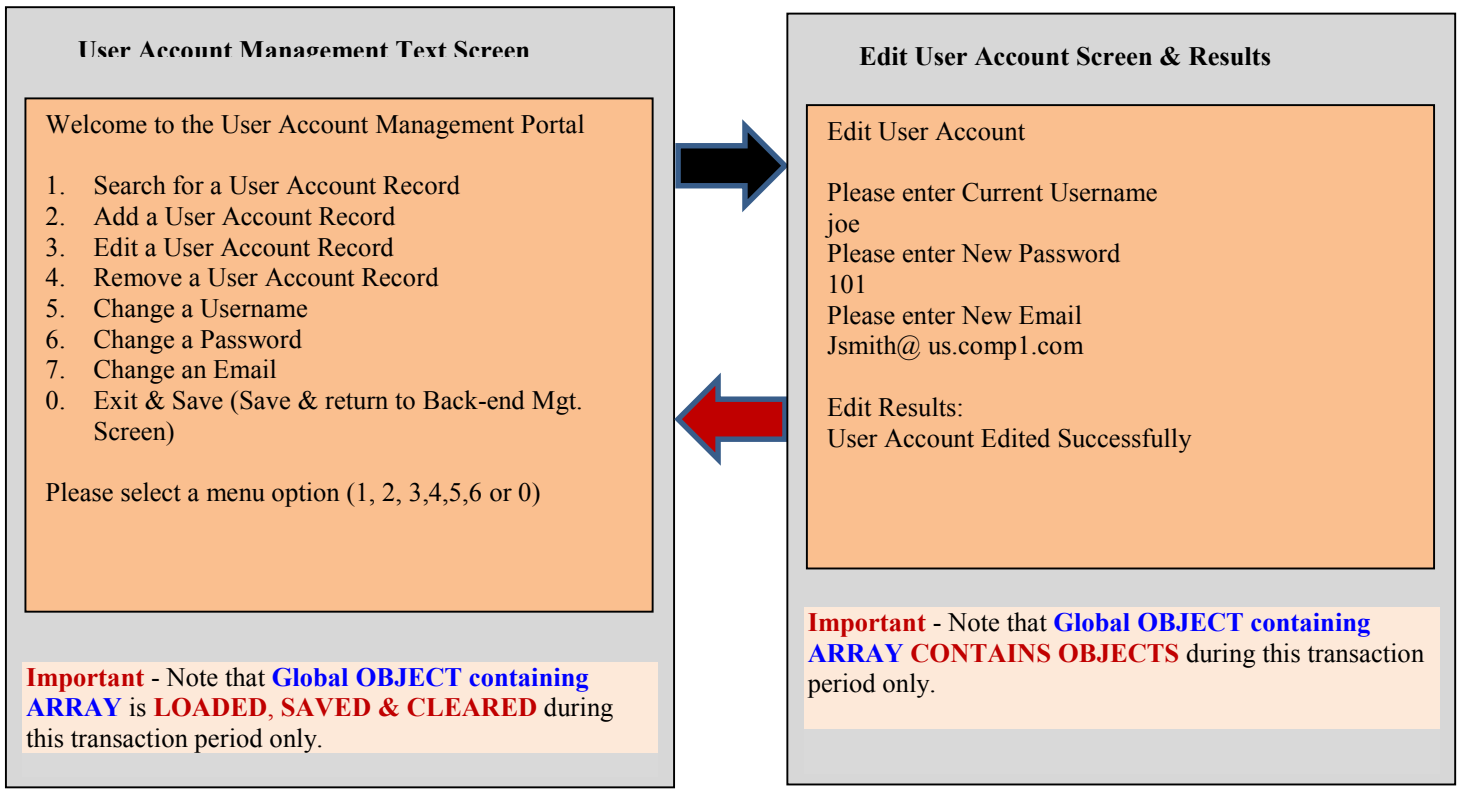


Navigation Flow:
User Account Management <-> Add User Account I/O & Results Screen



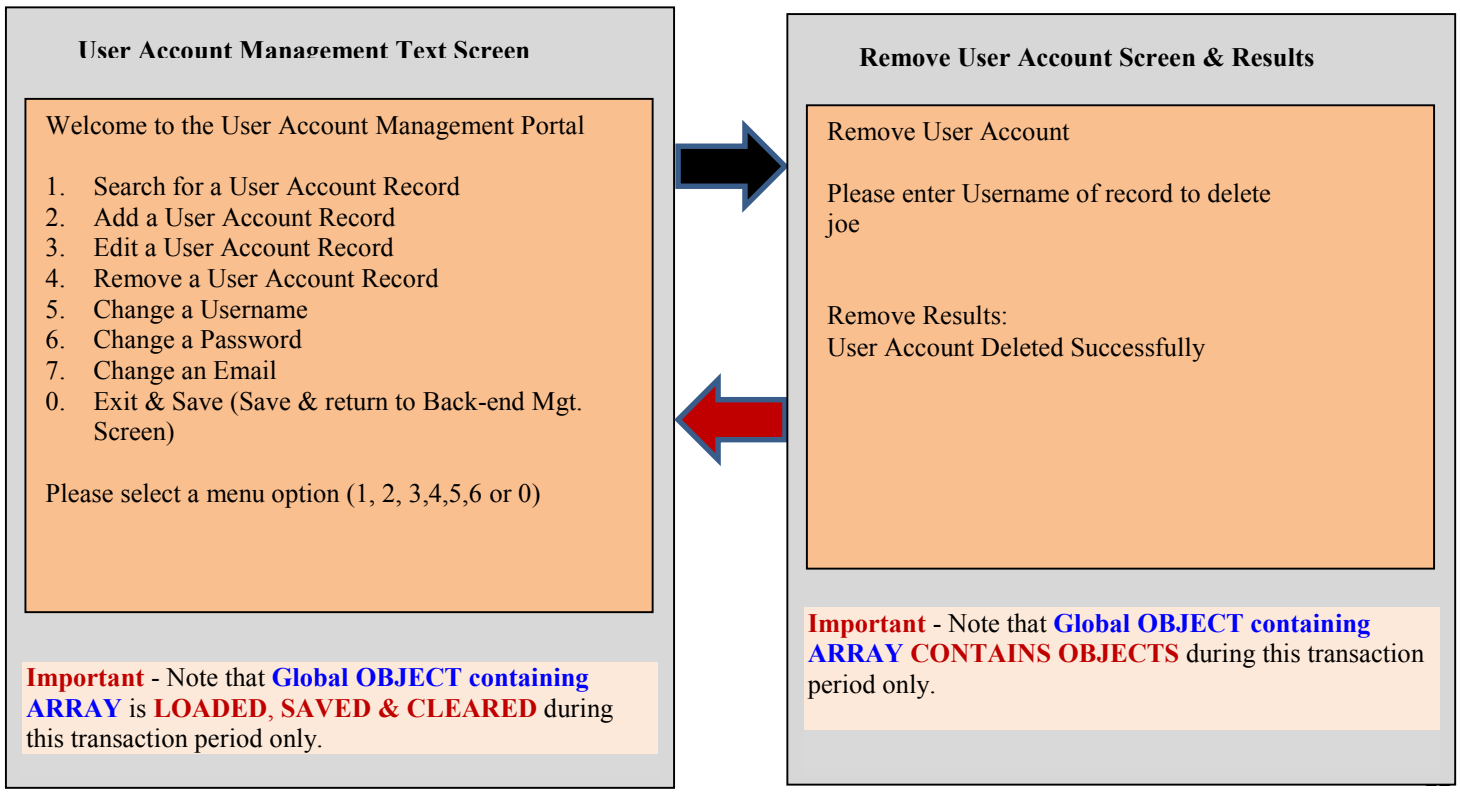
Navigation Flow:

User Account Management <-> Edit User Account I/O & Results Screen



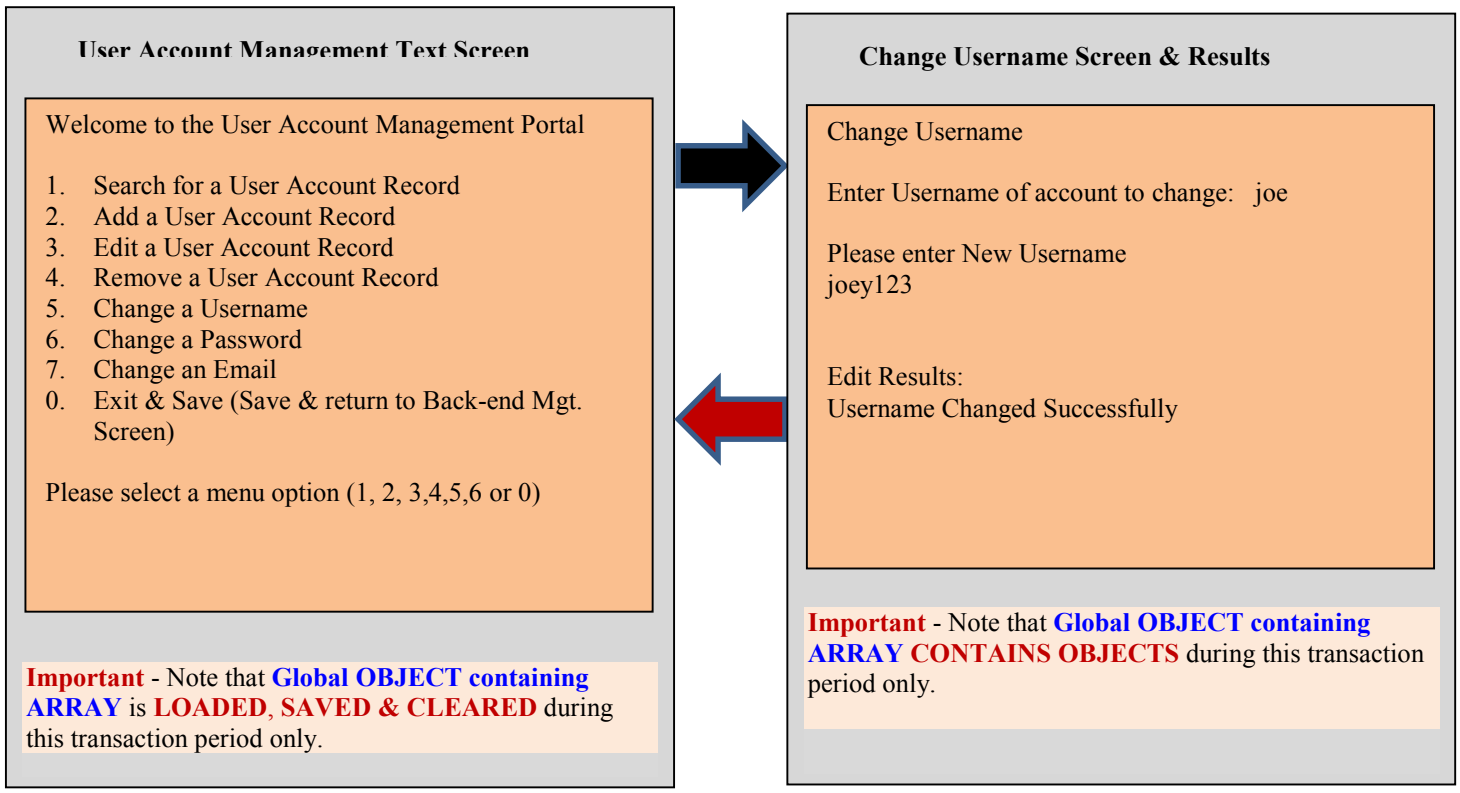
Navigation Flow:

User Account Management <-> Remove User Account I/O & Results Screen



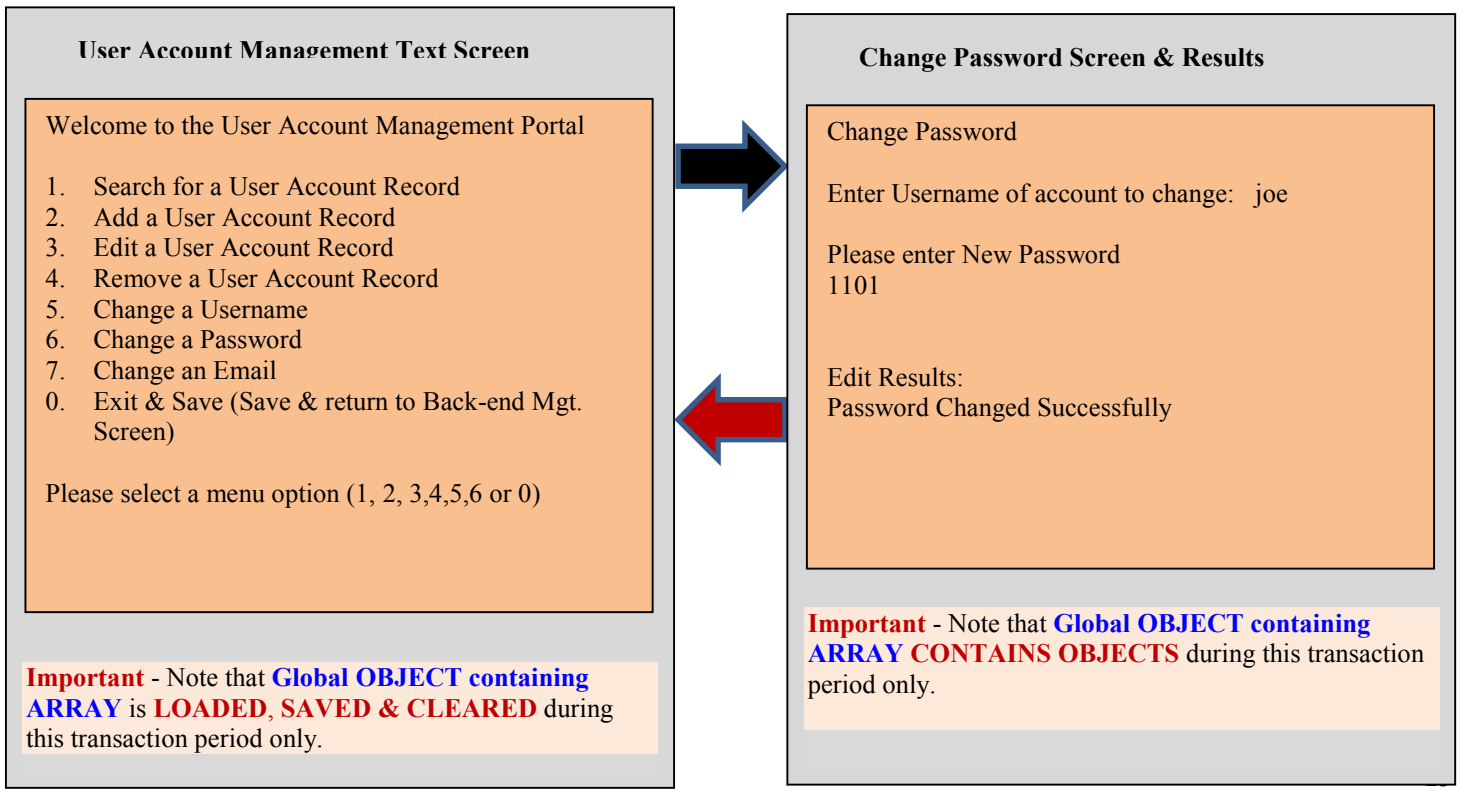
Navigation Flow:

User Account Management <-> Change Username I/O & Results Screen



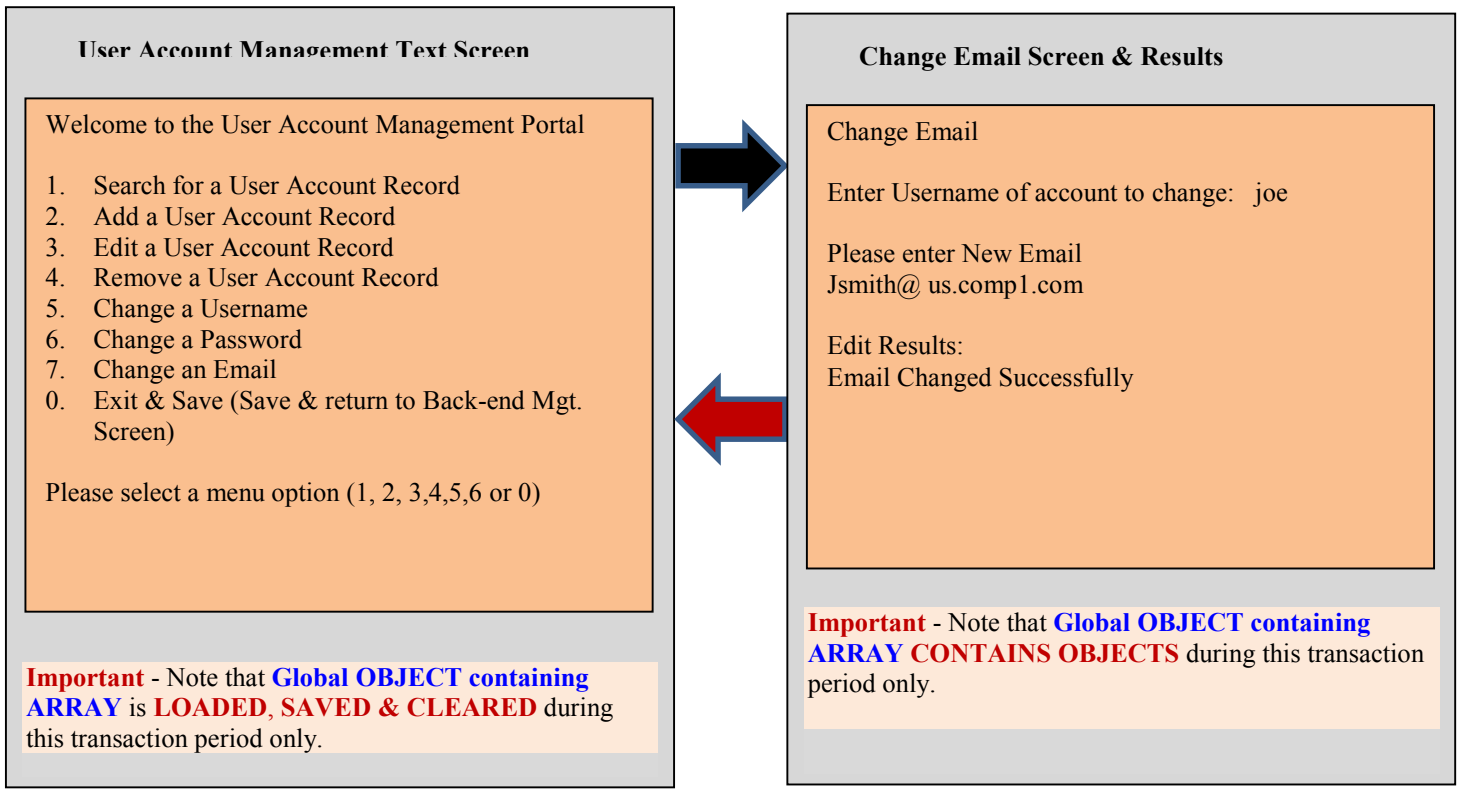
Navigation Flow:

User Account Management <-> Change Password I/O & Results Screen



Navigation Flow:

User Account Management <-> Change Email I/O & Results Screen



Homework Assignment # 3 (Page 09 of 14)

Requirements #10b – Navigation Flow #3: Login, Main/Welcome & Retail Point-of-Sales Screens Navigation Flow

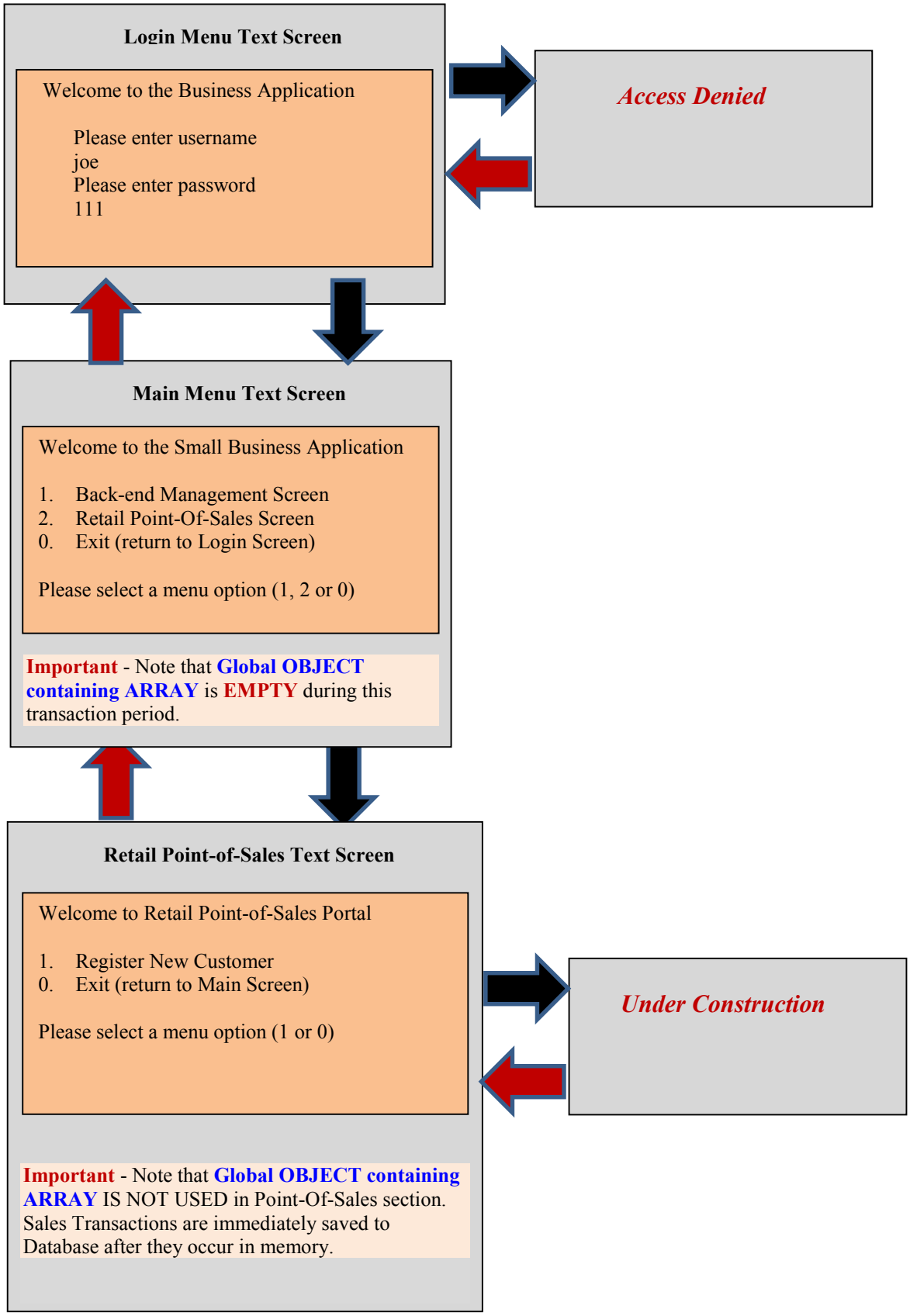
- The **SECOND** navigation flow and control is between the following screens:
  - `Login Screen`.
  - `Main/Welcome Screen`.
  - `Retail Point-of-Sales Screen`.

- **FORWARD & BACKWARDS Navigation Flow #3:**

  `Login Screen <-> Main/Welcome Screen <-> Retail Point-of-Sales Screen <-> msg(under construction)`

- A graphical diagram of this flow is shown in figure below:

Requirements #10b – Navigation Flow #3: Login, Main/Welcome & Retail Point-of-Sales Screens Navigation Flow Diagram



Homework Assignment # 3 (Page 09 of 14)

Requirements #10c – Code that Controls the Navigation from screen to screen

- We listed 2 separate navigation flows:

- **FORWARD & BACKWARDS Navigation Flow #1:**
  `Login Screen <-> Main/Welcome Screen <-> Back-end Management Screen <-> User Account Management Screen`

- **FORWARD & BACKWARDS Navigation Flow #2 –** Composed of several interactions between the User Account Screen and its Associated Screens:

  `User Account Management <-> Search User Account I/O & Results Screen`

  `User Account Management <-> Add User Account I/O & Results Screen`

  `User Account Management <-> Edit User Account I/O & Results Screen`

  `User Account Management <-> Remove User Account I/O & Results Screen`

  `User Account Management <-> Change Username I/O & Results Screen`

  `User Account Management <-> Change Password I/O & Results Screen`

  `User Account Management <-> Change Email I/O & Results Screen`

- **FORWARD & BACKWARDS Navigation Flow #3:**
  `Login Screen <-> Main/Welcome Screen <-> Retail Point-of-Sales Screen <-> msg(under construction)`

What Code Controls the Form Flow?

- What user-interface code controls each navigation?

  - **This is a tough question to answer since it all depends on how you design your program & code.**
  - **IT IS UP TO YOU & HOW YOU WILL PROGRAM THE APPLICATION that determines WHICH CODE & WHERE CONTROLS THE FLOW.**
  - **You will need to THINK! & PLAN YOUR CODE! Some control is done by the main() method requirements, but other YOU WILL HAVE TO DESIGN AND IMPLEMENT YOURSELF**
  - **The example in Lecture 3B is of great help here since the professor has provided an example of how he designed the Small Business Application. FEEL FREE TO ANALYZE & LEVERAGE THAT EXAMPLE & EMULATE ITS PROCESS/FLOW or COME UP WITH YOUR OWN SCHEME!**

Homework Assignment # 3 (Page 09 of 14)

Requirements #10c (Cont.) – Code that Controls the Navigation from screen to screen

- **The table below breaks down which flow is handle by the requirements of this document & which you will have to create:**

- **FORWARD & BACKWARDS Navigation Flow #1:**

`Login Screen <-> Main/Welcome Screen`	▪ Handled by **main() method** requirements. ▪ Implement the **main() method** as required and this flow is taken care of automatically.
`Main/Welcome Screen <-> Back-end Management Screen <->`	▪ **YOU WILL NEED TO DESIGN AND FIGURE THIS OUT.** ▪ See **Lecture 3B Small Business Application** for guidance to emulate or DEVELOP YOUR OWN.
`Back-end Management Screen <-> User Account Management Screen`	▪ **YOU WILL NEED TO DESIGN AND FIGURE THIS OUT.** ▪ See **Lecture 3B Small Business Application** for guidance to emulate or DEVELOP YOUR OWN.

- **FORWARD & BACKWARDS Navigation Flow #2:**

`User Account Management <-> Search User Account I/O & Results Screen`	▪ **YOU WILL NEED TO DESIGN AND FIGURE THIS OUT.** ▪ See **Lecture 3B Small Business Application** for guidance to emulate or DEVELOP YOUR OWN.
`User Account Management <-> Add User Account I/O & Results Screen`	▪ **YOU WILL NEED TO DESIGN AND FIGURE THIS OUT.** ▪ See **Lecture 3B Small Business Application** for guidance to emulate or DEVELOP YOUR OWN.
`User Account Management <-> Edit User Account I/O & Results Screen`	▪ **YOU WILL NEED TO DESIGN AND FIGURE THIS OUT.** ▪ See **Lecture 3B Small Business Application** for guidance to emulate or DEVELOP YOUR OWN.
`Other Associated Screens Flow`	▪ **YOU WILL NEED TO DESIGN AND FIGURE THIS OUT.** ▪ See **Lecture 3B Small Business Application** for guidance to emulate or DEVELOP YOUR OWN.

- **FORWARD & BACKWARDS Navigation Flow #3:**

`Login Screen <-> Main/Welcome Screen`	▪ Handled by **main() method** requirements. ▪ Implement the **main() method** as required and this flow is taken care of automatically.
`Main/Welcome Screen <-> Retail Point-of-Sales Screen`	▪ **YOU WILL NEED TO DESIGN AND FIGURE THIS OUT.** ▪ See **Lecture 3B Small Business Application** for guidance to emulate or DEVELOP YOUR OWN.
`Retail Point-of-Sales Screen <-> msg(under construction)`	▪ **YOU WILL NEED TO DESIGN AND FIGURE THIS OUT.** ▪ See **Lecture 3B Small Business Application** for guidance to emulate or DEVELOP YOUR OWN.

Requirements # 11 – The Login Screen: Security Access Authentication User Interface & Processing Requirements

- The Security Access Authentication system controls access to the application.
- **IMPORTANT!** The login screen is ***DISPLAYED*** & ***CONTROLLED*** by the **main()** method driver program algorithm code

Requirements #11a – The Login Screen User-interface Design

- The following user-interface text screen & I/O functionality should display prompting the user for their username & password:

  - **Login Menu Screen**:
    - ***It is the FIRST screen displayed by the application.***
    - ***Prompts the user for username and password***
    - ***If AUTHENTICATION IS VALID it DISPLAYS the MAIN/WELCOME SCREEN or else a message stating ACCESS DENIED***

Login Menu Text Screen

```
Welcome to the Business Application

    Please enter username
    joe
    Please enter password
    111
```

User-Interface Design Approach

- The following rules apply to how you design this screen :

  - **You have the freedom to design this screen as you see fit**.
  - **The layout & how is displayed is up to you and your imagination**.
  - **Nevertheless, the processing behind this screen must be followed as described in the next section of this requirement**.

Requirements # 11 (Cont.) – The Login Screen: Security Access Authentication User Interface & Processing Requirements

Requirements #11b – The Login Screen User-interface Code Implementation Requirements

Actions Taken by this Screen

- The **`Login Screen`** displays & does the following:

    1) Prompt for username & password.
    2) GETS the username & password from user.

- The **`main()Method`** requirements controls the flow & processing of this screen and does the following:

    1) If authenticate is valid, displays the **MAIN/WELCOME SCREEN**.
    2) The program will allow the user to continue to work within the functionalities provide by the **MAIN/WELCOME SCREEN** until the user ***EXITS*** the **MAIN/WELCOME SCREEN** screen (details to follow)
    3) If authenticate is invalid, it displays the TEXT message ***ACCESS DENIED!*** and the Login Screen is displayed again to allow the user to login again or the next user to login.

Code that Manages & Controls this Screen

- The **`Login Screen`** is managed and controlled as follows:

    - **FORWARD & BACKWARDS Navigation Flow #1 & #2:**

`Login Screen <-> Main/Welcome Screen`	▪ Handled by **main() method** requirements. ▪ Implement the **main() method** as required and this flow is taken care of automatically.

Requirements # 12 – The Main /Welcome Screen: Main/Welcome Screen User Interface & Processing Requirements

- **`Main/Welcome Screen`** – Once and authenticate user is allowed into the system, a MAIN or WELCOME SCREEN should display allowing the user to select from a list of options.

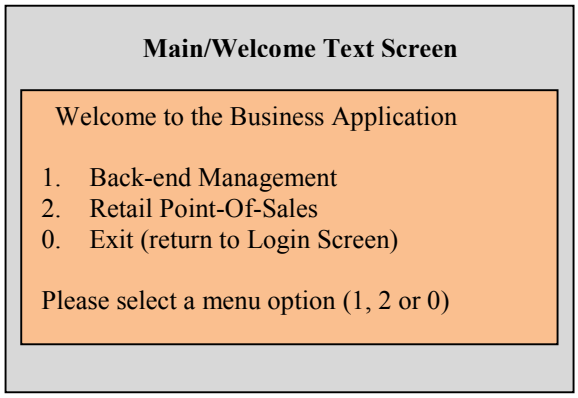
  - **`Main/Welcome Screen`** **Menu**:
    - ***It is the SECOND screen displayed by the application if the user AUTHENTICATION is valid:***

      `Login Screen <-> Main/Welcome Screen`

    - ***Lets the user select options to use the application to perform Back-end Management Tasks, Retail Point-of-Sales or Exit back to the Login Screen.***

Requirements #12a – The Main/Welcome Screen User-interface Design Requirements

- The following user-interface text screen & I/O functionality should be displayed by this screen:



User-Interface Design Approach

- The following rules apply to how you design this screen :

  - **You have the freedom to design this screen as you see fit**.
  - **The layout & how is displayed is up to you and your imagination**.
  - **Nevertheless, the processing behind this screen must be followed as described in the next section of this requirement**.

Requirements # 12 (Cont.) – The Main /Welcome Screen: Main/Welcome Screen User Interface & Processing Requirements

Requirements #12b – The Main/Welcome Screen User-interface Code Implementation Requirements

Actions Taken by this Screen

- The `Main/Welcome Screen` displays the following:

    1) Back-end Management
    2) Retail Point-of-Sales
    0) Exit

    - The following action is taken based on user selection:

        - **Selecting either option 1 – *DISPLAYS* a text-based `Back-end Management Screen`.**
        - **Selecting option 2 will display a text-based `Retail Point-of-Sales Screen`.**
        - **Selecting option 0 will *RETURN* BACK TO THE `Login Screen`.**

Code that Manages & Controls this Screen

- The `Main/Welcome Screen` is managed and controlled as follows:

    - **FORWARD & BACKWARDS Navigation Flow #1:**

`Login Screen <-> Main/Welcome Screen`	▪ Handled by **main() method** requirements. ▪ Implement the **main() method** as required and this flow is taken care of automatically.
`Main/Welcome Screen <-> Back-end Management Screen <->`	▪ **YOU WILL NEED TO DESIGN AND FIGURE THIS OUT.** ▪ See **Lecture 3B Small Business Application** for guidance to emulate or DEVELOP YOUR OWN.

    - **FORWARD & BACKWARDS Navigation Flow #2:**

`Login Screen <-> Main/Welcome Screen`	▪ Handled by **main() method** requirements. ▪ Implement the **main() method** as required and this flow is taken care of automatically.
`Main/Welcome Screen <-> Retail Point-of-Sales Screen`	▪ **YOU WILL NEED TO DESIGN AND FIGURE THIS OUT.** ▪ See **Lecture 3B Small Business Application** for guidance to emulate or DEVELOP YOUR OWN.

Requirements # 13 – The Back-end Management Screen: User Interface & Processing Requirements

- `Back-end Management Screen` – Navigation portal to backend & maintenance functionalities.

  - `Back-end Management Screen` **Menu**:

    - ***Part of Navigation Flow #1:***

    `Main/Welcome Screen <-> Back-end Management Screen <-> User Account Management Screen`

    - ***Allows the user to select options to use the application to perform do the following:***

      1) User Account Management
      0) Exit (return back to Main/Welcome Screen)

Requirements #13a – The Back-end Management Screen User-interface Design Requirements

- The following user-interface text screen & I/O functionality should be displayed by this screen:

Back-end Management Text Screen

Welcome to the Back-end & Maintenance Management Portal

1. User Account Management
0. Exit (return to Main Screen)

Please select a menu option (1,or 0)

Important - Note that **Global OBJECT containing ARRAY** is **EMPTY** during this transaction period.

User-Interface Design Approach

- The following rules apply to how you design this screen :

  - **You have the freedom to design this screen as you see fit**.
  - **The layout & how is displayed is up to you and your imagination**.
  - **Nevertheless, the processing behind this screen must be followed as described in the next section of this requirement**.

Requirements # 13 (Cont.) – The Back-end Management Screen: User Interface & Processing Requirements

Requirements #13b – The Back-end Management Screen User-interface Code Implementation Requirements

Actions Taken by this Screen

- The **`Back-end Management Screen`** displays the following:

    1) User Account Management
    1) Exit

    - The following action is taken based on user selection:

        - **Selecting either option 1 – *DISPLAYS* a text-based `User Account Management Screen`.**
        - **Selecting option 0 will *RETURN* BACK TO THE `Back-end Management Screen`.**

Code that Manages & Controls this Screen

- The **`Back-end Management Screen`** is managed and controlled as follows:

    - **FORWARD & BACKWARDS Navigation Flow #1:**

`Main/Welcome Screen <-> Back-end Management Screen <-> User Account Management Screen`	▪ **YOU WILL NEED TO DESIGN AND FIGURE THIS OUT.** ▪ See **Lecture 3B Small Business Application** for guidance to emulate or DEVELOP YOUR OWN.

Requirements # 14 – The User Account Management Screen: User Interface & Processing Requirements

- `User Account Management Screen` – Navigation portal to manage **User Accounts** Records.

  - `User Account Management Screen`:
    - *Part of Navigation Flow #1:*
    `Back-end Management Screen <-> User Account Management Screen`
    - *Allows the user to select options to use the application to perform do the following:*

      1) Search for a User Account Record by ID
      2) Add a User Account Record
      3) Edit a User Account Record
      4) Remove a User Account Record
      5) Change Username
      6) Change Password
      7) Change Email
      0) Exit & Save (Save & return to Back-end Mgt. Screen)

Requirements #14a – The User Account Management Screen UI Design Requirements

- The following user-interface text screen & I/O functionality should be displayed by this screen:

User Account Management Text Screen

Welcome to the User Account Management Portal

1. Search for a User Account Record
2. Add a User Account Record
3. Edit a User Account Record
4. Remove a User Account Record
5. Change a Username
6. Change a Password
7. Change an Email
0. Exit & Save (Save & return to Back-end Mgt. Screen)

Please select a menu option (1, 2, 3,4,5,6 or 0)

Important - Note that **Global OBJECT containing ARRAY** is **LOADED, SAVED & CLEARED** during this transaction period only.

User-Interface Design Approach

- The following rules apply to how you design this screen :

  - **You have the freedom to design this screen as you see fit**.
  - **The layout & how is displayed is up to you and your imagination**.
  - **Nevertheless, the processing behind this screen must be followed as described in the next section of this requirement**.

Requirements # 14 (Cont.) – The User Account Management Screen: User Interface & Processing Requirements

Requirements #14b – The User Account Management Screen User-interface Code Implementation Requirements

Actions Taken by this Screen

- The `User Account Management Screen` displays the following:

1) Search for a User Account Record by ID
2) Add a User Account Record
3) Edit a User Account Record
4) Remove a User Account Record
5) Change Username
6) Change Password
7) Change Email
0) Exit & Save (Save & return to Back-end Mgt. Screen)

- **Part of Navigation Flow #2**
- The following action is taken based on user selection:
  - **Selecting either option 1 – *DISPLAYS* a text-based `Search User Account I/O Screen & Results`.**
  - **Selecting either option 2 – *DISPLAYS* a text-based `Add User Account I/O Screen & Results`.**
  - **Selecting either option 3 – *DISPLAYS* a text-based `Edit User Account I/O Screen & Results`.**
  - **Etc.**
  - **Selecting option 0 will *RETURN* BACK TO THE `Back-end Management Screen`.**

Code that Manages & Controls this Screen

- The `Back-end Management Screen` is managed and controlled as follows:

- **FORWARD & BACKWARDS Navigation Flow #2:**

`User Account Management <-> Search User Account I/O & Results Screen`	▪ **YOU WILL NEED TO DESIGN AND FIGURE THIS OUT.** ▪ See **Lecture 3B Small Business Application** for guidance to emulate or DEVELOP YOUR OWN.
`User Account Management <-> Add User Account I/O & Results Screen`	▪ **YOU WILL NEED TO DESIGN AND FIGURE THIS OUT.** ▪ See **Lecture 3B Small Business Application** for guidance to emulate or DEVELOP YOUR OWN.
`User Account Management <-> Edit User Account I/O & Results Screen`	▪ **YOU WILL NEED TO DESIGN AND FIGURE THIS OUT.** ▪ See **Lecture 3B Small Business Application** for guidance to emulate or DEVELOP YOUR OWN.
`Other Associated Screens Flow`	▪ **YOU WILL NEED TO DESIGN AND FIGURE THIS OUT.** ▪ See **Lecture 3B Small Business Application** for guidance to emulate or DEVELOP YOUR OWN.

Requirements # 15 – The Search, Add, Edit, Remove, Change Username, Change Password, Change Email, Print & Print All: User Interface & Processing Requirements

- The following screens all are results of transactions with the **`User Account Management Screen`**.
  - **`Search User Account I/O & Results Screen`**.
  - **`Add User Account I/O & Results Screen`**.
  - **`Edit User Account I/O & Results Screen`**.
  - **`Remove User Account I/O & Results Screen`**.
  - **`Change Username I/O & Results Screen`**.
  - **`Change Password I/O & Results Screen`**.
  - **`Change Email I/O & Results Screen`**.

Requirements #15a – The Search, & Add UI Design Requirements

- The following user-interface text screen & I/O functionality should be displayed by this screen:

User-Interface Design Approach

- The following rules apply to how you design this screen :
  - **You have the freedom to design this screen as you see fit**.
  - **The layout & how is displayed is up to you and your imagination**.
  - **Nevertheless, the processing behind this screen must be followed as described in the next section of this requirement**.

Search User Account Screen & Results

Search User Account

Please enter Username
joe

Search Results
Record Information:
UserAccountID: 7dc53df5-703e-49b3-8670-b1c468f47f1f
Username: joe
Password: 111
Email: jsmith@ comp1.com

Important - Note that **Global OBJECT containing ARRAY CONTAINS OBJECTS** during this transaction period only.

Add User Account Screen & Results

Add New User Account

Please enter Username
joe
Please enter Password
111
Please enter Email
111

Add Results:
User Account Added Successfully

Important - Note that **Global OBJECT containing ARRAY CONTAINS OBJECTS** during this transaction period only.

Homework Assignment # 3 (Page 09 of 14)

Requirements #15a (Cont.) – The Edit, Remove, Change Username & Change Password UI Design Requirements

- The following user-interface text screen & I/O functionality should be displayed by this screen:

Edit User Account Screen & Results

Edit User Account

Please enter New Username
joe
Please enter New Password
111
Please enter New Email
111

Edit Results:
User Account Edited Successfully

Important - Note that **Global OBJECT containing ARRAY CONTAINS OBJECTS** during this transaction period only.

Remove User Account Screen & Results

Remove User Account

Please enter Username of record to delete
joe

Remove Results:
User Account Deleted Successfully

Important - Note that **Global OBJECT containing ARRAY CONTAINS OBJECTS** during this transaction period only.

Change Username Screen & Results

Change Username

Enter Username of account to change: joe

Please enter New Username
joey123

Edit Results:
Username Changed Successfully

Important - Note that **Global OBJECT containing ARRAY CONTAINS OBJECTS** during this transaction period only.

Change Password Screen & Results

Change Password

Enter Username of account to change: joe

Please enter New Password
1101

Edit Results:
Password Changed Successfully

Important - Note that **Global OBJECT containing ARRAY CONTAINS OBJECTS** during this transaction period only.

Requirements #15a (Cont.) – The Change Email, Print & Print All UI Design Requirements

- The following user-interface text screen & I/O functionality should be displayed by this screen:

Change Email Screen & Results

Change Email

Enter Username of account to change: joe

Please enter New Email
Jsmith@ us.comp1.com

Edit Results:
Email Changed Successfully

Important - Note that **Global OBJECT containing ARRAY CONTAINS OBJECTS** during this transaction period only.

Homework Assignment # 3 (Page 09 of 14)

Requirements #15b – The Search, Add, Edit, Remove, Change Username, Change Password, Change Email, Print & Print All User-interface Code Implementation Requirements

Actions Taken by these Screen

- These `Screen` perm the required processing dictated by the `User Account Management Screen` as follows:

1) Search for a User Account Record
2) Add a User Account Record
3) Edit a User Account Record
4) Remove a User Account Record
5) Change Username
6) Change Password
7) Change Email

  - Each screen performs the selected functionality.

    - `Search User Account I/O Screen & Results –` **Search for a User Account & displays the record**
    - `Add User Account I/O Screen & Results –` **Add new User Account & displays if successful or failed**.
    - `Edit User Account I/O Screen & Results–` **Edit User Account & displays if successful or failed**.
    - `ETC`.

Code that Manages & Controls this Screen

- The `Search, Add, Edit, Remove, Change Username, Change Password, Change Email, Print & Print All` are managed and controlled as follows:

  - **FORWARD & BACKWARDS Navigation Flow #2:**

`User Account Management <-> Search User Account I/O & Results Screen`	▪ **YOU WILL NEED TO DESIGN AND FIGURE THIS OUT**. ▪ See **Lecture 3B Small Business Application** for guidance to emulate or DEVELOP YOUR OWN.
`User Account Management <-> Add User Account I/O & Results Screen`	▪ **YOU WILL NEED TO DESIGN AND FIGURE THIS OUT**. ▪ See **Lecture 3B Small Business Application** for guidance to emulate or DEVELOP YOUR OWN.
`User Account Management <-> Edit User Account I/O & Results Screen`	▪ **YOU WILL NEED TO DESIGN AND FIGURE THIS OUT**. ▪ See **Lecture 3B Small Business Application** for guidance to emulate or DEVELOP YOUR OWN.
`Other Associated Screens Flow`	▪ **YOU WILL NEED TO DESIGN AND FIGURE THIS OUT**. ▪ See **Lecture 3B Small Business Application** for guidance to emulate or DEVELOP YOUR OWN.

Homework Assignment # 3 (Page 09 of 14)

Requirements # 16 – The Retail Point-of-Sales Screen: User Interface & Processing Requirements

- `Retail Point-Of-Sales Screen` – Navigation portal to manage **selling of products**.
  - `Retail Point-Of-Sales Screen`:
    - *Part of Navigation Flow #3:*
      `Back-end Management Screen <-> User Account Management Screen`
    - *Allows the user to select options to use the application to perform do the following:*

      1) Register New Customer
      0) Exit (return to Main Screen)

Requirements #16a – The Retail Point-of-Sales Screen UI Design Requirements

- The following user-interface text screen & I/O functionality should be displayed by this screen:

Retail Point-of-Sales Text Screen

Welcome to Retail Point-of-Sales Portal

1. Register New Customer
0. Exit (return to Main Screen)

Please select a menu option (1 or 0)

Important - Note that **Global OBJECT containing ARRAY** IS NOT USED in Point-Of-Sales section. Sales Transactions are immediately saved to Database after they occur in memory.

User-Interface Design Approach

- The following rules apply to how you design this screen :
  - **You have the freedom to design this screen as you see fit**.
  - **The layout & how is displayed is up to you and your imagination**.
  - **Nevertheless, the processing behind this screen must be followed as described in the next section of this requirement**.

Homework Assignment # 3 (Page 09 of 14)

Requirements #16b – The Retail Point-of-Sales Screen User-interface Code Implementation Requirements

Actions Taken by this Screen

- The `Retail Point-of-Sales Screen` displays the following:

1) Register New Customer
0) Exit (return to Main Screen)

  - The following action is taken based on user selection:
    - **Selecting either option 1 – *DISPLAYS* a text-Message stating `Under Construction!`.**
    - **Selecting option 0 will *RETURN* BACK TO THE `Main/Welcome Screen`.**

Code that Manages & Controls this Screen

- The `Retail Point-of-Sales Screen` is managed and controlled as follows:

  - **FORWARD & BACKWARDS Navigation Flow #3:**

`Main/Welcome Screen <-> Retail Point-of-Sales Screen`	▪ **YOU WILL NEED TO DESIGN AND FIGURE THIS OUT.** ▪ See **Lecture 3B Small Business Application** for guidance to emulate or DEVELOP YOUR OWN.
`Retail Point-of-Sales Screen <-> msg(under construction)`	▪ **YOU WILL NEED TO DESIGN AND FIGURE THIS OUT.** ▪ See **Lecture 3B Small Business Application** for guidance to emulate or DEVELOP YOUR OWN.

Validation Test Script

Test #1 – Login Authentication Test

Homework Assignment 3 (Cont.) – (Page 10 of 14)

Requirements # 17 – Test Algorithm/SCRIPT FOR TESTING IF PROGRAM WORKS

- **Use the following TEST SCRIPT TO TEST YOUR PROGRAM BEFORE SUBMITTING.**
- **ONLY IF YOUR PROGRAM MEETS THE FOLLOWING 2 REQUIREMENTS IS IT CONSIDERED ACCEPTABLE:**
  1. **MEETS ALL 17 REQUIREMENTS listed in this HW3 Requirement document.**
  2. **Passes TEST SCRIPT BELOW BY EXECUTING ALL 9 TEST AND YIELDING THE EXPECTED RESULTS.**

- **Test Script:**

Test		Steps/Action	EXPECTED Results	Explanation
Test 1 – Test END of program **AT THE START OF EXECUTION** using BOTH ***Username = -1*** & ***Password = -1***		**Step 1:** Run the application	***Login Screen*** Should Display	▪ Based on **main() function** algorithm, Login Screen should display as per
		Step 2: In Login Screen **Username = -1** **Password = -1**	Program **ENDS**	▪ Testing if program **ENDS AT THE START OF EXECUTION** ▪ Based on **main()** function algorithm, program ends when both **Username = -1 & Password = -1**. ▪ We are ENDING immediately when the PROGRAM STARTED. There should be no processing or authentication taking place. Proving the design is efficient.
Test 2 – Test Valid ***Username*** & ***Password*** At the **BEGINNING** of the ARRAYS		**Step 1:** Run the application	***Login Screen*** Should Display	▪ Based on **main()** function algorithm, Login Screen should display as per
		Step 2: In Login Screen **Username = joe** **Password = 111**	1) Text screen should displays: "**Access Granted**" 2) Login screen Should Display Again	▪ Testing the Authentication Process **WITH A VALID USERNAME & PASSWORD LOCATED AT THE BEGINNING OF THE ARRAY**. ▪ Based on **main() function** algorithm, U & P values entered are authenticated via MAIN CLASS ***authenticate(U,P) function*** that determines if access is granted all denied. ▪ **"joe"**, "**111**" is within an object in the array, therefore authentication SHOULD PASS & DISPLAY **ACCESS GRANTED**. ▪ Based on the **main() function** algorithm the Login Screen displays after every authentication.

Test Algorithm (Cont.)

- Test Script (Cont.):

Test		Steps/Action	EXPECTED Results	Explanation
Test 3 – Test Valid ***Username*** & ***Password*** At the **END** of the ARRAYS		**Step 1:** In Login Screen **Username** = **nancy** **Password** = **555**	1) Text screen should displays: "**Access Granted**" 2) Login screen Should Display Again	▪ Testing the Authentication Process **WITH A VALID USERNAME & PASSWORD AT THE END OF THE ARRAY** ▪ Based on **main() function** algorithm, U & P values entered are authenticated via MAIN CLASS ***authenticate(U,P) function*** that determines if access is granted all denied. ▪ **"nancy"**, "**555"** is within an object in the array, therefore authentication SHOULD PASS & DISPLAY **ACCESS GRANTED**. ▪ **Also, we actually searched the ENTIRE ARRAY UNTIL THE LAST CELL, verifying that SEARCH is working for VALID USERS IN THE ENTIRE ARRAY.** ▪ Based on the **main() function** algorithm the Login Screen displays after every authentication.

Test Algorithm (Cont.)

- Test Script (Cont.):

Test	Steps/Action	EXPECTED Results	Explanation
Test 4 – Test **Invalid** ***Username*** & ***Password***	**Step 1:** In Login Screen **Username** = **frank** **Password** = **777**	1) Text screen should displays: "**Access Denied**" 2) Login screen Should Display Again	▪ Testing the Authentication Process **WITH AN INVALID USERNAME & PASSWORD SHOULD DENY ACCESS**. ▪ Based on **main() function** algorithm, U & P values entered are authenticated via MAIN CLASS ***authenticate(U,P) function*** that determines if access is granted all denied. ▪ **"frank"**, "**777**" is not an object in the array therefore authentication fails & DISPLAY **ACCESS DENIED**. ▪ Based on the **main() function** algorithm the Login Screen displays after every authentication.
Test 5 – Test **valid** ***Username*** & **Invalid** ***Password combination.***	**Step 1:** In Login Screen **Username** = **joe** **Password** = **555**	1) Text screen should displays: "**Access Denied**" 2) Login screen Should Display Again	▪ Testing the Authentication Process **WITH A VALID USERNAME BUT INVALID PASSWORD FOR ANOTHER USER SHOULD DENY ACCESS**. ▪ Based on **main() function** algorithm, U & P values entered are authenticated via MAIN CLASS ***authenticate(U,P) function*** that determines if access is granted all denied. ▪ **"joe"**, "**555**" IS NOT A CORRECT Username & Password combination found in any OBJECT in the array, therefore authentication fails. ▪ Based on the **main() function** algorithm the Login Screen displays after every authentication ▪ **Proves using the right username but someone else password does not grant access.**
Test 6 – Test **invalid** ***Username*** & **valid** ***Password***	**Step 1:** In Login Screen **Username** = **frank** **Password** = **555**	3) Text screen should displays: "**Access Denied**" 4) Login screen Should Display Again	▪ Testing the Authentication Process **WITH AN INVALID USERNAME BUT VALID PASSWORD SHOULD DENY ACCESS**. ▪ Based on **main() function** algorithm, ***authenticate(U,P)*** determines if access is granted all denied. ▪ **"frank"** Frank is an invalid Username but "**555**" , is a valid password, but having just an existing password does not grant you access. BOTH COMBINATION MUST BE MET FOR ACCESS. ▪ Based on the **main() function** algorithm the Login Screen displays after every authentication.

Test Algorithm (Cont.)

- Test Script (Cont.):

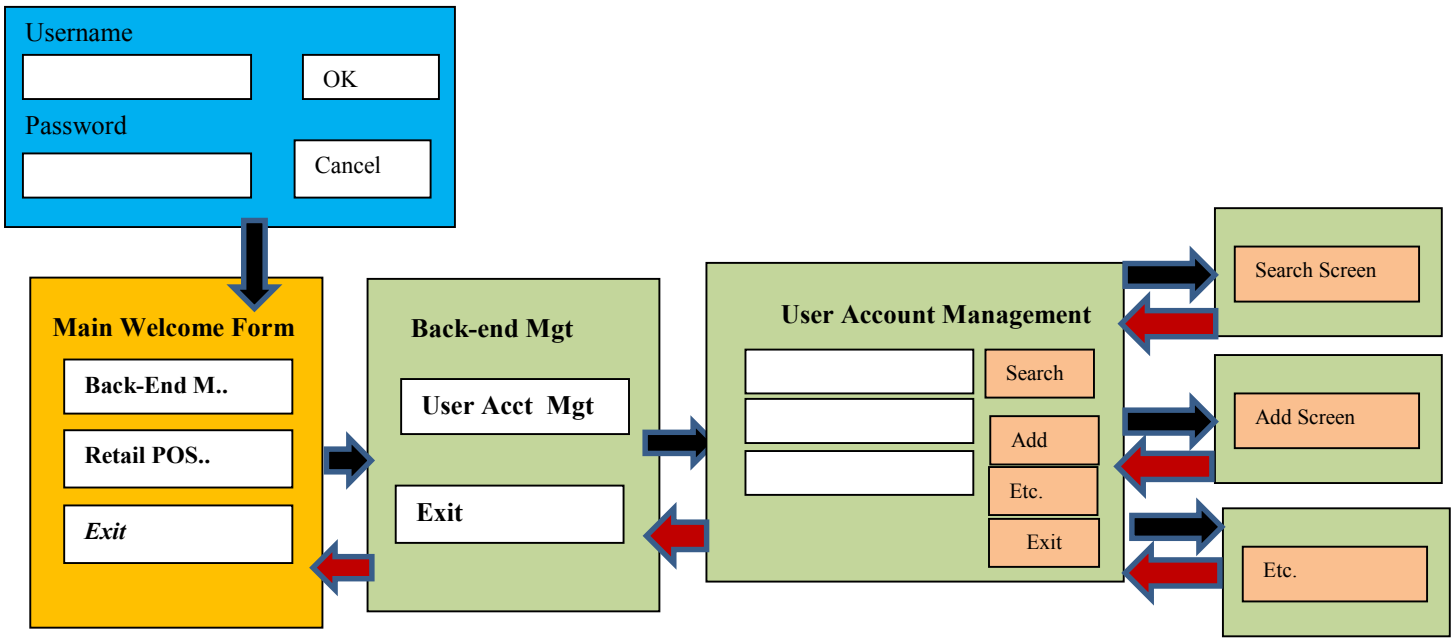
Test	Steps/Action	EXPECTED Results	Explanation
Test 7 – Test "END of program process" using **valid *Username* & *Password* = -1**	**Step 2:** In Login Screen **Username** = **joe** **Password** = **-1**	1) Text screen should displays: "**Access Denied**" 2) Login screen Should Display Again	▪ Testing if program ENDS **WITH ONLY PASSWORD = -1 SHOULD DENY ACCESS & NOT END PROGRAM** ▪ Based on **main() function** algorithm, program ends when both **Username = -1 & Password = -1** ▪ Based on Main algorithm, ***authenticate(U,P)*** determines if access is granted all denied. ▪ **"joe"**, "**-1"** are considered values to authenticate and thus NOT in array of objects therefore authentication fails. ▪ Based on the **main() function** algorithm the Login Screen displays after every authentication.
Test 8 – Test END of program using ***Username* = -1 &** **valid *Password***	**Step 1:** In Login Screen **Username** = **-1** **Password** = **111**	5) Text screen should displays: "**Access Denied**" 6) Login screen Should Display Again	▪ Testing if program ENDS **WITH ONLY USERNAME = -1 SHOULD DENY ACCESS & NOT END PROGRAM** ▪ Based on **main() function** algorithm, program ends when both **Username = -1 & Password = -1** ▪ Based on Main algorithm, ***authenticate(U,P)*** determines if access is granted all denied. ▪ **"-1"**, "**111"** are considered values to authenticate and thus NOT in array of objects therefore authentication fails. ▪ Based on the **main() function** algorithm the Login Screen displays after every authentication
Test 9 – Test END of program using BOTH ***Username* = -1 & *Password* = -1**	**Step 1:** In Login Screen **Username** = **-1** **Password** = **-1**	Program **ENDS**	▪ Testing if program ENDS **WITH BOTH USERNAME = -1 AND PASSWORD = -1 SHOULD END PROGRAM** ▪ Based on **main() function** algorithm, program ends when both **Username = -1 & Password = -1**.

Test #2 – User Account Management Forward & Backwards Navigation Form Flow Test

Homework Assignment # 5 (Page 33 of 39)

Test SCRIPT (Cont.)

- Forward Form Flow being Tested:



- Test Script:

Test	Steps/Action	EXPECTED Results	Explanation
Test 1 – User Account Management FORWARD NAVIGATION FORM FLOW	Step 1: Run the application	*Login Form* Should Display	▪ The *main()* Method navigation flow code should handle this process of displaying the Login Screen.
	Step 2: In Login Screen Username = joe Password = 111	▪ "Main Welcome Form" Displays	▪ Navigation flow code should handle this process of displaying the Back-end Management Screen.
	Step 3: In the Welcome Screen ***SELECT*** option #1 Back-end Management Screen	The Back-end Management Screen Should *Display*.	▪ Navigation flow code should handle this process of displaying the Back-end Management Screen.
	Step 4: In the Back-end Management Screen ***SELECT*** User Account Management option #1	The User Account Management Screen Should *Display*.	▪ Navigation flow code should handle this process of displaying the User Account Management Screen.

Homework Assignment # 5 (Page 33 of 39)

Test SCRIPT (Cont.)

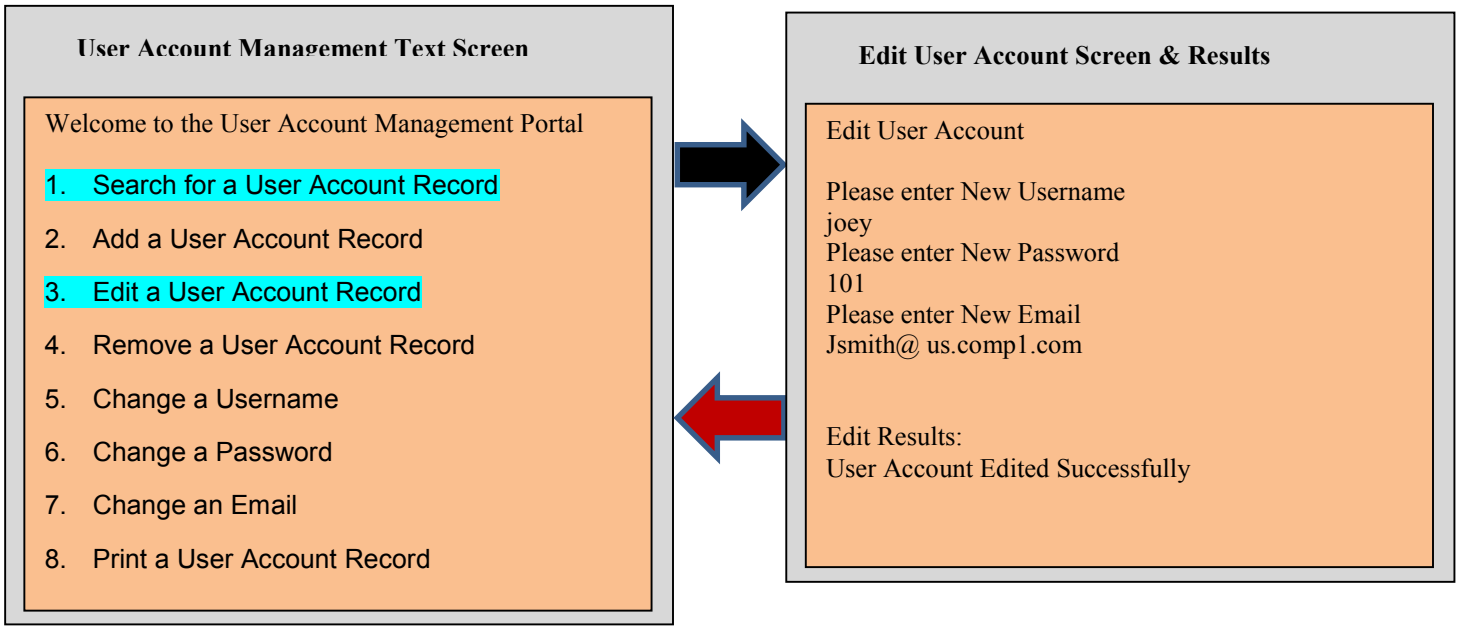
- Backward Form Flow being Tested:

Test	Steps/Action	EXPECTED Results	Explanation
Test 2 – **User Account Management FORWARD NAVIGATION FORM** FLOW	**Step 1:** From the **User Account Management Screen** ***SELECT*** option #0 **EXIT**	**Back-end Management Screen** Should Display	▪ Navigation flow code should handle this process of displaying the **Back-end Management Screen**.
	Step 2: From the **Back-end Management Screen** ***SELECT*** option #0 **EXIT**	**Main Welcome Screen** Displays	▪ Navigation flow code should handle this process of displaying the **Main Welcome Screen**.
	Step 3: In the **Main Welcome Screen** ***SELECT*** option #0 **EXIT**	The **Login Screen** Should ***Display***.	▪ Navigation flow code should handle this process of displaying the **Main Welcome Screen**.

Test SCRIPT – User Account Management Functionality Test (Cont.)

TEST 10 – User Account Management Form EDIT & SEARCH Test

- In this test we will test **SEARCH & EDIT an *EXITING* User Account RECORD** then VALIDATING by **SEARCHING** for the **RECORD *EDITED* & verify.**



- **TEST 3a** – Test Script:

Test	Steps/Action	EXPECTED Results	Explanation
Test 3a – Testing the **User Account Management EDIT** feature by ***SEARCHING*** & ***EDITING*** an **EXISTING RECORD**.	**Step 1:** From the **User Account Management Screen *SELECT*** option #3 **Edit User Account**	▪ The **`Edit User Account I/O & Results Screen`** **is displayed**.	▪ Screen is displayed.
	Step 2: In the **`Edit User Account I/O & Results Screen`** enter **Username** of customer to search: ***Enter*** the following **User Account *Username***: **nancy** Using keyboard ENTER.	▪ The **`username`** is entered.	▪ **Username** is needed in order to find the User Account record to modify.
	Step 3: After entering the **Username** data, For the RECORD whose **Username = nancy**, ***MODIFY*** the following values as prompted: **New Password: 515** **New Email:** nroberts@mycomp.com	▪ **Message is displayed stating that User Account Record Modified successfully**.	▪ **UPDATED User Account record** is ***MODIFIED*** in the **arrUserAccountList** ARRAY inside the **objUserAccountList** object.

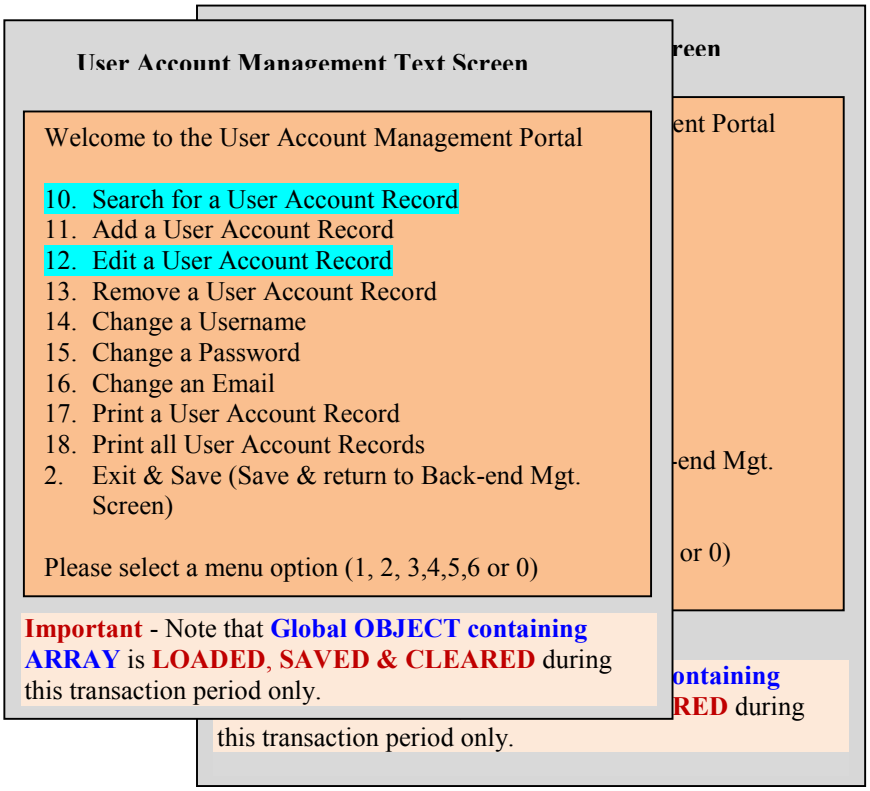
Homework Assignment # 6 (Page 61 of 69)

Test SCRIPT – User Account Management Functionality Test (Cont.)

TEST 3 – User Account Management Form EDIT & SEARCH Test (Cont.)

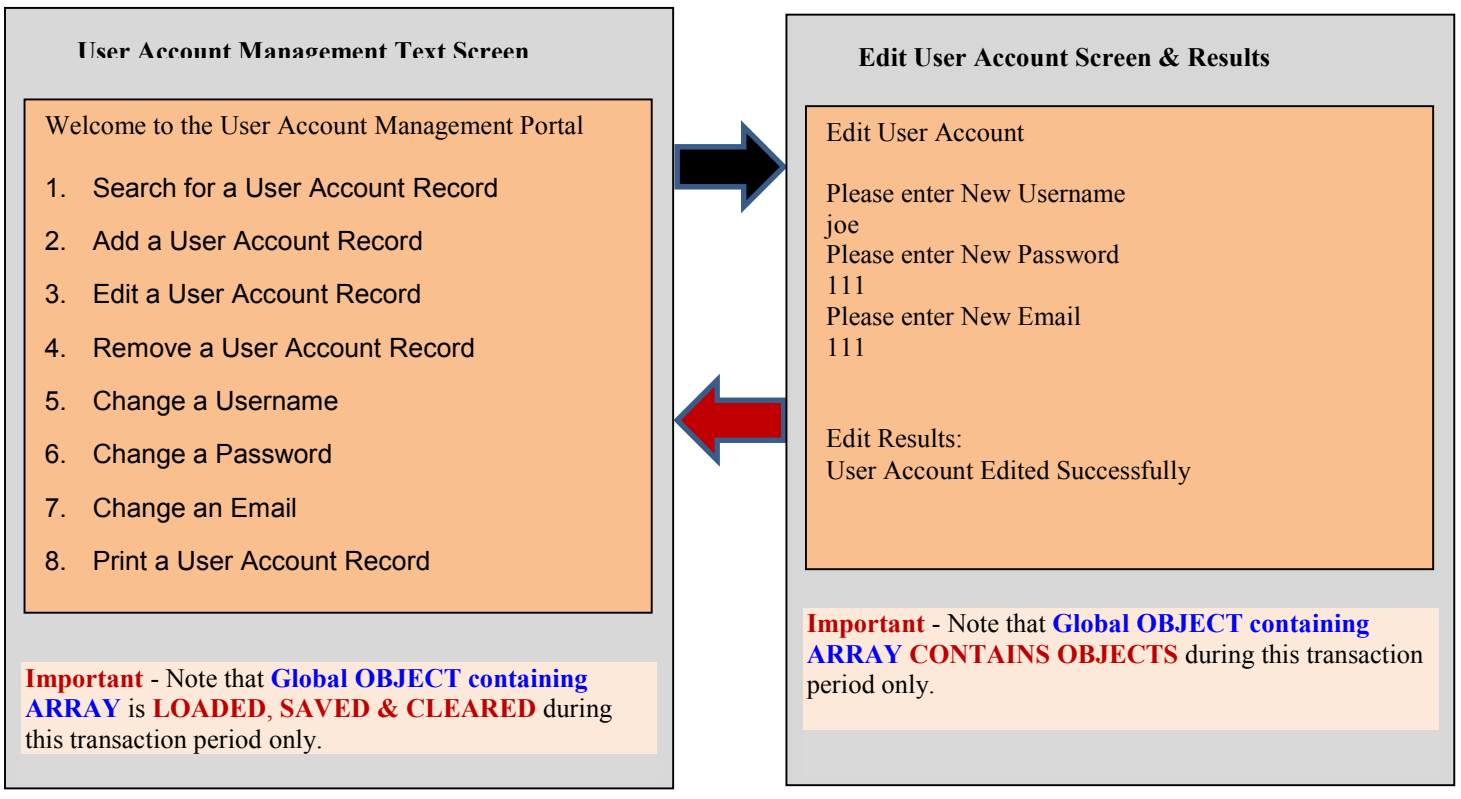
- **TEST 3b** – Test Script:

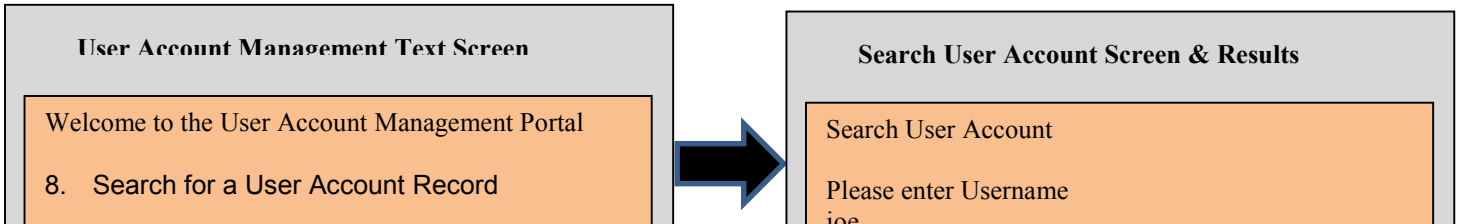
Test	Steps/Action	EXPECTED Results	Explanation
Test 3b – **VALIDATING RECORD** was ***EDITED*** by performing a **SEARCH** for the **RECORD** & ***DISPLAYING*** to confirm **MODIFICATION**.	**Step 1:** From the **User Account Management Screen** ***SELECT*** option #1 **Search User Account**	▪ The **`Search User Account I/O & Results Screen`** **is displayed**.	▪ Screen is displayed.
	Step 2: In the **`Edit User Account I/O & Results Screen`** enter **Username** of customer to search: ***Enter*** the following **User Account** ***Username***: **nancy** Using keyboard ENTER.	▪ The **`username is entered`**. ▪ The record of the **User Account RECORD whose Username = nancy** is DISPLAYING REFLECTING THE EDITS OR CHANGES THAT WERE MADE	▪ **Username** is needed in order to find the User Account. ▪ The recode of the user is DISPLAYED by the SEARCH & proving that the CHANGES took place.



- **Search User Account I/O & Results Screen**.
- **Add User Account I/O & Results Screen**.
- **Edit User Account I/O & Results Screen.**
- **Remove User Account I/O & Results Screen.**
- **Change Username I/O & Results Screen**.
- **Change Password I/O & Results Screen.**
- **Change Email I/O & Results Screen.**
- **Print User Account I/O & Results Screen.**
- **Print All Results Screen**.

UserAcccountID	Username	Password	Email
Auto Generated GUID	joe	111	jsmith@ comp1.com
Auto Generated GUID	angel	222	arodriguez@comp1.com
Auto Generated GUID	sam	333	speterson@ comp1.com
Auto Generated GUID	mary	444	mjohnson@ comp1.com
Auto Generated GUID	nancy	555	nrivera@ comp1.com





Expected Deliverables

Expected Deliverables

- **YOU WILL BE GRADED BASED ON SOLVING THE PROBLEM STATEMENT & MEETING ALL REQUIREMENTS!**
- You need to submit the following:

HW2 PART 2 –Working Application (DUE on Wednesday April 1)