CST3613 – Homework #4

Problem Statement & Application Overview

Homework Assignment # 4

Problem Statement

- **UPGRADE** the **`CLIENT/SERVER Console Application`** ***Business Application*** of **HW#3**
- In this version of the application will include the following high-level requirements:

1. **KEEP ALL REQUIREMENTS FROM HW#3:**

   - Security Authentication System
   - Main Welcome Screen and Processing
   - Back-end Management System
   - User Account Management System
   - Retail Management System

2. **UPGRADE – `UserAccount Class`:**
   - A flaw was found in the design of the **`UserAccount Class`** from HW2 & 3.
   - More this in the details requirements section

3. **UPGRADE – `UserAccountList Class`:**
   - To support new requirements, the **`UserAccountList Class`** from HW3 will require an upgrade.
   - More this in the details requirements section

4. **(NEW) – Employee Management System** – Users that enter this section will be presented with the following features to manage Employees of the company:
   1) Search Employee
   2) Add New Employee
   3) Edit Existing Employee
   4) Delete Employee
   5) Print Employee
   6) Print All Employees
   0) Exit (Back to Back-End Management Screen)

5. **NEW/UPGRADE – `Employee Class`:**
   - You are to RE-USE the Employee Class from HW#2 to support the **Employee Management System**
   - Upgrade is required to the **`print() method`**. Details in requirements section of this document.

6. **NEW – `EmployeeList Class`:**
   - Create an **`EmployeeList Class`** to manage all **`Employee Objects`** in memory to support the NEW **Employee Management System**

7. **NEW – Add Exception Handling** – Add Exception handling to all code to handle **RUNTIME ERRORS**:
   - Add Exception handling to all code where specified using **`try-catch-statement`** to handle **RUNTIME ERRORS**

8. **NEW – SAVE all Data To FILE for PERMANENT STORAGE** – Add code using **JAVA FILE API** to implement **FILE ACCESS CODE** to ***SAVE*** & ***LOAD*** **ALL DATA FROM FILE**:
   - ALL DATA is to now be SAVED TO FILE. Add the necessary FILE ACCESS CODE to the **`EmployeeList Class`**.
   - The following text files will be used: **`UserAccountData.txt`** & **`EmployeeData.txt`**

- Details are listed in requirements sections below

Homework Assignment # 4

Requirements #1 – Keep all Functionality of HW#3

- Keep all **Authentication** Features and Functionality of the pervious **HW#3**:
  - **Login Screen**
  - **Authentication process & Main screen functionalities**
  - Only change the implementation if required by this HW.
- Keep all **User Account Management System** Features and Functionality of the pervious **HW#3**:
  - **ADD** User Account Record
  - **SEARCH** for User Account Record
  - **EDIT** User Account Record
  - **DELETE** User Account Record
  - **Change USERNAME**
  - **Change PASSWORD**
  - **Change EMAIL**
  - Only change the implementation if required by this HW.

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)

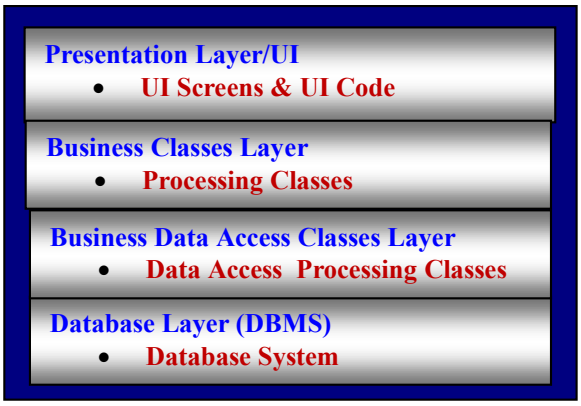
  - **MODULAR** – program should be implemented using tools to make it modular, this works hand in hand with scalability:
    - Program should be written in parts or modular. In other words mayor functionalities and features should be enclosed within block of code that can be called when needed.
    - 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!!!

Homework Assignment # 4

Requirements #3 – UPGRADE the 3-Tiered Client/Server Development Design Pattern by ADDING a FILE Database Layer to TEMPORARY STORE data in TEXT FILES

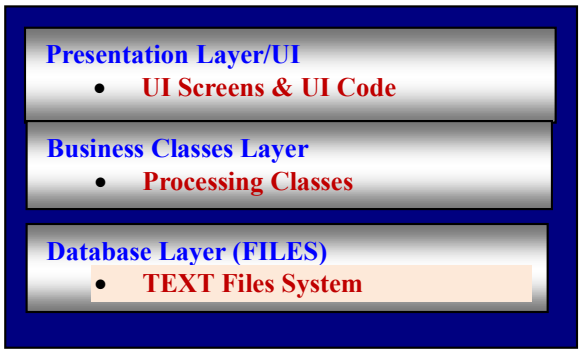
- The goal is to implement this application as a distributed network application. This is accomplished by implementing based on a Client/Server design pattern.
- Objectives are as follows:

  - **MAIN OBJECTIVE** is to take **SECOND 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 BUSINESS DATA ACCESS LAYER OR A TRUE DATABASE LAYER OR DATABASE MANAGEMENT SYSTEM (Oracle, SQL Server etc.) to permanently store our data at this time, we will use a TEMPORARY SOLUTION. WE WILL SAVE OUR DATA TO FILE by IMPLEMENTING A FILE DATABASE LAYER. Therefore you objectives are to create this INTERMEDIATE ARCHITECTURE in this HW:



Limited Client/Server Application Architecture using TEXT FILES

  - Details on how to implement this requirements to follow in sections below:

Homework Assignment # 4

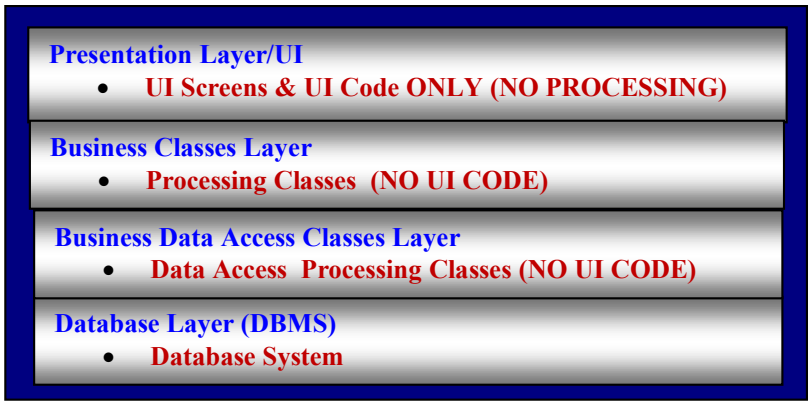
Requirements #4 – NO USER-INTERFACE CODE in Business Class Layer!

- No UI code in Business Class Layer:

  - The Client/Server Application Architecture dictates that there should be **NO USER-INTERFACE CODE** in the **BUSINESS CLASS LAYER**!
  - Nevertheless, in previous HWs & class examples, for teaching purpose we used `System.out.println("string")` statements to print data to the CONSOLE or SCREEN or to DISPLAY MESSAGES FROM THE CLASSES TO THE SCREEN!
  - NO LONGER ARE YOU ALLOW TO DO THIS SINCE IS A VIOLATION TO OUR CLIENT/SERVER ARCHITECTURE.

- Going forward we will enforce the following rules:

  - Presentation/UI Layer – No processing code. UI code only!
  - Business Classes Layer – No UI code. Processing code only!
  - Data Access Business Classes Layer – No UI code. Processing code only!



4 Tiers Windows Client/Server Application Architecture

  - Details on how to implement this requirements to follow in sections below

Update the Business Object Layer

OOP Step 1 – UPGRADE the UserAccount

Homework Assignment # 4

Requirement #5 – UPGRADE the Tested UserAccount Class from HW1 & HW2 to meet the Requirements for a Business Class by adding Data Access Methods. In addition, make other upgrades highlighted in UML diagram below

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

```
UserAccount

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

Public Getter/Setter Methods/Properties:
getUserAccountID():                 String
setUserAccountID():                 void
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
```

Homework Assignment # 4

- **IMPORTANT DISCUSSION OF NEW REQUIREMENTS FOR UserAccount CLASS**:

  - We are now faced with our first **DESIGN CHANGE** to the **UserAccount** Class.
  - Currently the data type of the **userAccountID PRIVATE DATA** in the **UserAccount** Class is **UUID**.
  - This was done by design in previous HWs since we needed a random ID via **UUID**.
  - Nevertheless, as our HW project matures I have realized that we will eventually have to save this to **UUID** value as a STRING datatype to database and load it from database as a STRING data type.
  - Here is where the problem arises. There is **NO SETTER METHOD** for the **userAccountID PRIVATE DATA**. By design we made it **READ ONLY**! So if is read only how are we going to SET IT when LOADING FROM DATABASE?
  - That is the problem, so we are now forced to **UPGRADE** the class **by ADDING A SETTER METHOD TO SET** the **userAccountID PRIVATE DATA**.
  - But this poses another problem that the **userAccountID PRIVATE DATA** is a **UUID OBJECT**. As you know, currently our **GETTER METHOD CONVERTS** the **UUID OBJECT** to a **STRING**. So what we use in the program for the **userAccountID PRIVATE DATA** is a STRING VALUE once we CALL THE GETTER METHOD. If we are to create a **SETTER METHOD** it will require that we **CONVERT** what is **SET** into the **SETTER METHOD** to **STRING** to a **UDID** object.
  - So I have decided that instead of doing SO MANY CONVERSIONS of the many conversion of the **UUID OBJECT** to STRING & back & forth, to simply MAKE THE **userAccountID PRIVATE DATA** A **String** data type INSTEAD OF A **UUID OBJECT**.
  - Finally, we will modify our constructor to CREATE the **UUID OBJECT** as they do now but we WILL ALSO CONVERT IT TO A STRING AT THE SAME TIME, SO THE **userAccountID PRIVATE DATA** WILL ALWAYS STORE A **STRING**!
  - The following requirements will accomplish all changes required in the **UserAccount** Class.

- **REQUIREMENTS 5a – UPGRADE** the data type to the **userAccountID PRIVATE DATA** from **UUID** data type to **String**.

Scope	Data Member Name	Type	Description
private	userAccountID	**String**	▪ MODIFY this PRIVATE DATA by changing the DATA TYPE from **UUID** Pointer/Reference to a TYPE **STRING**! ▪ This POINTER variable will store a UNIQUE user account ID STRING VALUE. We will generate this unique string from a **UUID** object ▪ Note this declaration is a POINTER to a **STRING** object now. ▪ **Theory:** - This String variable will eventually POINT to a **STRING** containing a unique **UUID** UNIQUE RANDOM NUMBER that CANNOT universally be replicated OBJECT. - A **UUID** string looks similar to the following: **7dc53df5-703e-49b3-8670-b1c468f47f1f** ▪ We GENERATE this UNIQUE **UUID STRING** in the **CONSTRUCTORS**!
private	username	**String**	▪ **NO CHANGES REQUIRED!** Stores the username
private	password	**String**	▪ **NO CHANGES REQUIRED!** Stores the password
private	email	**String**	▪ **NO CHANGES REQUIRED!** Stores the employee email

Homework Assignment # 4

- **REQUIREMENTS 5b – MODIFY** the **EXISTING getUserAccountID GETTER METHOD** will NO LONGER NEED TO CONVERT FROM UUID TO STRING SINCE THE **userAccountID PRIVATE DATA** IS NOW A **STRING**! Also, since we will ADD a SETTER THIS PROPERTY (combination of GETTER/SETTER) will no longer be READ-ONLY:

Scope	Name	Return Type	Parameters	Description

public	**getUserAccountID**	**String**	**None**	▪ UPGRADE Objectives – ***MODIFY*** this GETTER to simply return the ***userAccountID*** private data without CONVERTING TO UUID. Remove any conversion code.
				▪ Objectives – ***GETS*** the ***userAccountID*** private **STRING** data
				▪ Here is what the code for this **GETTER METHOD** would look like in your class (I am giving you the code). Is just a regular **GETTER METHOD**:
				
				`public String getUserAccountID() {`
				`  'GET private data`
				`  return userAccountID;`
				
				`}`

- **REQUIREMENTS 5c – ADD** a **SETTER METHOD** to **SET** the **userAccountID PRIVATE DATA** :

Scope	Name	Return Type	Parameters	Description

public	**setUsername**	**void**	**String**	▪ **SETS** the ***userAccountID*** **private data** with a **new VALUE**
				▪ Here is what the code for this **SETTER METHOD** would look like in your class (I am giving you the code). Is just a regular **SETTER METHOD**:
				
				`public void setUserAccountID(String userID) {`
				`  'SET private data`
				`  userAccountID = userID;`
				
				`}`

- **REQUIREMENTS 5d – NO CHANGES** required for other **SETTER/GETTER METHODS**:

Scope	Name	Return Type	Parameters	Description
public	getUsername	**String**	**None**	▪ **NO CHANGES REQUIRED! GETS/RETURNS** the ***username*** **private data**
public	setUsername	**void**	**None**	▪ **NO CHANGES REQUIRED! SETS** the ***username*** **private data** with a **new VALUE**
public	getPassword	**String**	**None**	▪ **NO CHANGES REQUIRED! GETS/RETURNS** the ***password*** **private data**
public	setPassword	**void**	**None**	▪ **NO CHANGES REQUIRED! SETS** the ***m_ Password*** **private data** with a **new VALUE**
public	getEmail	**String**	**None**	▪ **NO CHANGES REQUIRED! GETS/RETURNS** the ***m_ Email*** **private data**
public	setEmail	**void**	**None**	▪ **S NO CHANGES REQUIRED! ETS** & GETS the ***m_ Email*** **private data** with a **new VALUE**

Homework Assignment # 4

- **REQUIREMENTS 5e – MODIFY** the **EXISTING Default Constructor METHOD** to create the a **UUID OBJECT** BUT **CONVERT** IT TO A **STRING** AND ASSIGN TO THE **userAccountID PRIVATE DATA**:

Scope	Name	Return Type	Parameters	Description
private	**UserAccount**	**None**	**None**	▪ **PRIVATE Default Constructor**. ▪ The **UUID OBJECT** is **CREATED** & **CONVERTED** TO A **STRING**, AND ASSIGN TO THE **userAccountID PRIVATE**: **STEPS**: ▪ **(NEW UPGRADE) - SET** the following ***data*** members to **DEFAULT** values as follows: 1. Set ***userAccountID*** to POINT to **UNIQUELY GENERATED UUID** OBJECT we need to call the **UUID** CLASS ***randomUUID()*** & in addition **CONVERT TO A STRING** as follows: `userAccountID = UUID.randomUUID().toString();` ▪ **(NO CHANGES REQUIRED) -** The remaining ***data*** members are **DEFAULTED:** 2. SET ***username*** = "" 3. SET ***password*** ="" 4. SET **email** ="""

- **REQUIREMENTS 5f – MODIFY** the **EXISTING Parameterized Constructor METHOD** to create the a **UUID OBJECT** BUT **CONVERT** IT TO A **STRING** AND ASSIGN TO THE **userAccountID PRIVATE DATA**:

Scope	Name	Return Type	Parameters	Description
private	**UserAccount**	**None**	▪ A parameter to be used to SET PRIVATE DATA via **SETTER METHODS** to SET the private data: - **setUsername(par1)** - **setPassword(par2)** - **setEmail(par3)** ▪ Should have a total of **3** parameters: - **par1 to Par3** ▪ **All parameters are Pass-by-Value** ▪ **IMPORTANT! Name parameters as you see appropriate. Try to keep name short**	▪ **PRIVATE Parameterized Constructor**. ▪ The **UUID OBJECT** is **CREATED** & **CONVERTED** TO A **STRING**, AND ASSIGN TO THE **userAccountID PRIVATE**. Note **userAccountID** NOT PART OF THE PARAMETER LIST & needs to be DEFAULTED: **STEPS**: ▪ **(NEW UPGRADE) - SET** the following ***data*** members to **DEFAULT** values as follows: 1. Set ***userAccountID*** to POINT to **UNIQUELY GENERATED UUID** OBJECT we need to call the **UUID** CLASS ***randomUUID()*** & in addition **CONVERT TO A STRING** as follows: `userAccountID = UUID.randomUUID().toString();` ▪ **(NO CHANGES REQUIRED) -** The remaining ***data*** members are SET the data from PARAMETER LIST by calling **SETTER METHOD** with the required PARAMETER VALUES to SET the private data as follows**:** 2. **setUsername(par1)** 3. **setPassword(par2)** 4. **setEmail(par3)**

Homework Assignment # 4

- **REQUIREMENTS 5g – NO CHANGES REQUIRED** to **Public** Data Access Methods:

Scope	Name	Return Type	Parameters	Description
public	load	**void**	**String**	▪ ***NO CHANGES REQUIRED***
public	insert	**void**	**None**	▪ ***NO CHANGES REQUIRED***
public	update	**void**	**None**	▪ ***NO CHANGES REQUIRED***
public	delete	**void**	**None**	▪ ***NO CHANGES REQUIRED***

- **REQUIREMENTS 5h – NO CHANGES REQUIRED** to **Protected** Data Access Methods:

Scope	Name	Return Type	Parameters	Description
protected	Database_Load	**void**	**String**	▪ ***NO CHANGES REQUIRED***
protected	Database_Insert	**void**	**None**	▪ ***NO CHANGES REQUIRED***
protected	Database_Update	**void**	**None**	▪ ***NO CHANGES REQUIRED***
protected	Database_Delete	**void**	**None**	▪ ***NO CHANGES REQUIRED***

OOP Step 1 – UPGRADE the UserAccountList Class

Homework Assignment # 4

Requirement #6 – UPGRADE the UserAccountList Class to SAVE & LOAD data from a TEXT FILE `UserAccount.txt`

- The custom UML diagram below illustrates the CURRENT IMPLEMENTATION of the **UserAccountList Class**:

```
UserAccountList
---------------------------------------------------------------
Private Data:
private SIZE:                                        int
private arrUserAccountList[]:                        UserAccount

---------------------------------------------------------------
Public Default Constructor Method:
public UserAccountList()

Public Methods:
public search(username):                             UserAccount
public add(UserAccount):                             boolean
public add(x,y,z.):                                  boolean
public edit(username,UserAccount):                   boolean
public edit(x,y,z.):                                 boolean
public remove(username):                             boolean
public clear():                                      void
public authenticate(U, P):                           boolean
public changeUsername(username, newUseranme):        boolean
public changePassword(username, newPassword):        boolean
public changePassword(username, newEmail):           boolean

Public Data Access Methods:
public void load():                                  void
public void save():                                  void

Public Data Access Methods:
protected void database_Load():                      void
protected void database_Save():                      void
```

- The following UPGRADE IS REQUIRED to the **UserAccountList Class**:

  1. **Exception Handling** using `Try-Catch-Finally` to on all methods where required.
  2. Using **FILE ACCESS CODE**, Upgrade the `database_Load()` method to ***LOAD*** data from the `UserAccountData.txt` file
  3. Using **FILE ACCESS CODE**, Upgrade the `database_Save()` method to ***SAVE*** data to the `UserAccountData.txt` file

- Details on each of these methods is shown in the following sections.
- Also I will relist all method & data for this class just to validate the implementation requirements from HW3.

Homework Assignment # 4

- **REQUIREMENTS 6a –** **UserAccountList** Class **PRIVATE DATA** ***(NO CHANGES REQUIRED!)***:

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** ***(NO CHANGES REQUIRED!)***:

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 # 4

- **REQUIREMENTS 6c –** **UserAccountList** Class **PROCESSING METHODS** ***(UPGRADE)!***:

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. ▪ Algorithm: **1.** ***SEARCH*** **`ARRAY`** for **`OBJECT whose KEY/username`** is passed as parameter. **2.** **IF FOUND**, ***RETURNS*** the **`POINTER`** to the **`OBJECT`** in **`ARRAY`** & **Exits** the **`Method`**. **3.** **ELSE IF NOT FOUND**, ***RETURNS*** a **`null`** indicating object was not found. **4.** Add **Error-Handling code** using **`Try-Catch-Finally`** to handle all required ***Unique OR/AND GENERAL*** Exceptions & **`Re-Throw`** the Exception.
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. ▪ Algorithm: **1.** ***SEARCH*** **`ARRAY`** for **`Empty`** cell. **2.** **IF FOUND**, ***ADDS*** the **`OBJECT`** to the **`ARRAY`** & **`Return True`** to **Exit** the **`Method`**. **3.** **ELSE IF** **`Empty`** cell **NOT FOUND**, **`Return false`** to **Exit** the **`Method`** indicating **ARRAY IS FULL**. **4.** Add **Error-Handling code** using **`Try-Catch-Finally`** to handle all required ***Unique OR/AND GENERAL*** Exceptions **`Re-Throw`** the Exception.
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 ▪ Algorithm: **1.** ***CREATE*** a Temp **OBJECT** **2.** populates it with **VALUES** from parameters **3.** ***SEARCH*** **`ARRAY`** for **`Empty`** cell. **4.** **IF FOUND**, ***ADD*** the **`OBJECT`** to the **`ARRAY`** & **Exit** the **`Method`**. **5.** **ELSE IF** **`Empty`** cell **NOT FOUND**, **`Return false`** to **Exit** the **`Method`** indicating **ARRAY IS FULL**. **6.** Add **Error-Handling code** using **`Try-Catch-Finally`** to handle all required ***Unique OR/AND GENERAL*** Exceptions **`Re-Throw`** the Exception.

Homework Assignment # 4

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

Scope	Name	Return Type	Parameters	Description
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** the following **PROPERTIES**: **password** & **email** with VALUES from **OBJECT** passed as parameter. NOTE that the ***EXCEPTION*** are the **username & userAccountID UUID** WHICH ARE NOT MODIFIED!!! ONLY THE **password** & **email** ARE MODIFIED. ▪ Returns a **TRUE** if object found and EDITED. Returns **FALSE** if object not found in ARRAY. ▪ Algorithm: **1.** ***SEARCH* `ARRAY` `(skipping empty cells)`** for **`OBJECT`** who's **KEY/USERNAME** is passed as parameter. **2.** If found, ***EDIT*** the **`OBJECT`** in the **`ARRAY`** by ***SETTIN/CALLING* SETTER METHODS** with data from **`UserAccount Object Pointer`** passed as parameter**`. In other words, you are GETTING the PROPERTIES of obUserAccount Object Parameter and SETTING the PROPERTIES of OBJECT found in ARRAY a location arrUserAccountList(Index)`**. **3.** **`Return True`** to **Exit** the **`Method`** ❖ **DO NOT SET the username or userAccountID properties** since is the **KEY** or **ID** and should NOT be tampered with. So **SET all properties EXCEPT the username or userAccountID** **4.** If not found, returns a **`False`**. **5.** Add **Error-Handling code** using **`Try-Catch-Finally`** to handle all required ***Unique OR/AND GENERAL*** Exceptions & **`Re-Throw`** the Exception.
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** the following **PROPERTIES**: **password** & **email** with **VALUES** from VALUES passed as parameter. NOTE that the ***EXCEPTION*** are the **username & userAccountID UUID** WHICH ARE NOT MODIFIED!!! ONLY THE **password** & **email** ARE MODIFIED. ▪ Returns a **TRUE** if object found and EDITED. Returns **FALSE** if object not found in ARRAY ▪ Algorithm: **1.** ***SEARCH* `ARRAY`** for **`OBJECT`** who's **KEY/USERNAME** is passed as parameter. **2.** If found, ***EDIT*** the **`OBJECT`** in the **`ARRAY`** by ***SETTING/CALLING* SETTER METHODS** with data from **`VALUES`** passed as parameters**`. In other words, you are SETTING the PROPERTIES of OBJECT found in ARRAY a location arrUserAccountList(Index)`** with **`VALUES`** from the parameter list. **3.** **`Return True`** to **Exit** the **`Method`** ❖ **DO NOT SET the username or userAccountID properties** since is the **KEY** or **ID** and should NOT be tampered with. So **SET all properties EXCEPT the username or userAccountID** **4.** If nt found, returns a **`False`**. **5.** Add **Error-Handling code** using **`Try-Catch-Finally`** to handle all required ***Unique OR/AND GENERAL*** Exceptions & **`Re-Throw`** the Exception.

Homework Assignment # 4

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

Scope	Name	Return Type	Parameters	Description
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 ▪ Algorithm: **1.** ***SEARCH*** **`ARRAY`** `(skipping empty cells)` for **`OBJECT`** who's **KEY/USERNAME** is passed as parameter. **2.** If found, **DELETE** the **`OBJECT`** from **`ARRAY`** & **`Return True`** to **Exit** the **`Method`**. **3.** If not found, returns a **`False`**. **4.** Add **Error-Handling code** using **`Try-Catch-Finally`** to handle all required ***Unique OR/AND GENERAL*** Exceptions & **`Re-Throw`** the Exception
public	changeUsername	**boolean**	**String** **username** **String** **newUsername**	▪ Method performs the Process of ***CHANGING THE USERNAME*** of a **UserAccount** Object in the **arrUserAccountList** **ARRAY**. The ***username*** 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. ▪ Algorithm: **5.** ***SEARCH*** **`ARRAY`** `(skipping empty cells)` for **`OBJECT`** who's **username** is passed as parameter. **6.** If found, **Calls** the **`Object's`** ***`setUsername(NewUsername)`*** method passing the new **newUsername** parameter to do the work & **`Return True`** to **Exit** the **`Method`**. **7.** If not found, returns a **`False`**. **8.** Add **Error-Handling code** using **`Try-Catch-Finally`** to handle all required ***Unique OR/AND GENERAL*** Exceptions & **`Re-Throw`** the Exception

Homework Assignment # 4

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

Scope	Name	Return Type	Parameters	Description
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. ▪ Algorithm: **1.** ***SEARCH*** `ARRAY` `(skipping empty cells)` for `OBJECT` who's **username** is passed as parameter. **2.** If found, **Calls** the `Object's` `setPassword(NewPassword)` method passing the **newPassowrd** parameter to do the work & `Return True` to **Exit** the `Method`. **3.** If not found, returns a `False`. **4.** Add **Error-Handling code** using `Try-Catch-Finally` to handle all required ***Unique OR/AND GENERAL*** Exceptions & `Re-Throw` the Exception
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. ▪ Algorithm: **1.** ***SEARCH*** `ARRAY` `(skipping empty cells)` for `OBJECT` who's **username** is passed as parameter. **2.** If found, **Calls** the `Object's` `setUsername(NewEmail)` method passing the **newEmail** parameter to do the work & `Return True` to **Exit** the `Method`. **3.** If not found, returns a `False`. **4.** Add **Error-Handling code** using `Try-Catch-Finally` to handle all required ***Unique OR/AND GENERAL*** Exceptions & `Re-Throw` the Exception
public	clear	**void**	**none**	▪ ***CLEARS*** the `arrUserAccountList` **ARRAY** of **OBJECTS** – Method ***CLEARS*** the ARRAY of Objects by setting all pointers to **NULL**. ▪ Algorithm: **1.** ***SEARCH*** `ARRAY` and **DELETE EVERY** `Objects` from Array using ANY METHOD YOU DESIRE, BUT FOLLOW BEST PRACTICE IF POSSIBLE. **2.** Add **Error-Handling code** using `Try-Catch-Finally` to handle all required ***Unique OR/AND GENERAL*** Exceptions & `Re-Throw` the Exception

Homework Assignment # 4

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

Scope	Name	Return Type	Parameters	Description
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`**. **7.** Add **Error-Handling code** using **`Try-Catch-Finally`** to handle all required ***Unique OR/AND GENERAL*** Exceptions & **`Re-Throw`** the Exception

- **REQUIREMENTS 6d – UserAccountList** Class **PUBLIC DATA ACCESS METHODS *(NO CHANGES REQUIRED!)***:

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

Homework Assignment # 4

- **REQUIREMENTS 6e – UPGRADE** the **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 TEXT FILE & populates the object with data retrieved from database. ▪ ***LOAD*** the **OBJECTS** from the `UserAccountData.txt` file and ***ADDS*** them to the `arrUserAccountList` **ARRAY**. ▪ Algorithm: **1. VERIFY FILE EXISTS before READING.** **2. If FILE DOES NOT EXISTS IT CREATES IT**. **3.** ***CREATE*** `BufferredReader` **&** `FileReader` **OBJECTS** to ***OPEN*** `UserAccountData.txt` File for **READING**. **4. Read a LINE** from file & ***PARSE*** each ***comma-delimited line***. **5.** ***CREATE*** a **NEW** temporary **objUserAccount OBJECT** of the **UserAccount CLASS**. **6.** ***SET*** **OBJECT WITH VALUES** from **PARSED LINE READ FROM FILE**. **7.** ***ADD*** **OBJECT** TO **ARRAY** by ***CALLING*** the **Add(objUserAccount) METHOD OF THIS CLASS, TO DO THE WORK**. **8.** ***REPEAT*** STEP 4 through 7 process until **EOF**. **9.** Add **Error-Handling code** using `try-catch-finally` to handle all required ***I/O & GENERAL*** Exceptions & `Re-Throw` the Exception.
protected	Database_Save	**void**	**none**	▪ Protected Data access method that actually performs the SAVING of ALL RECORDS to FILE. ▪ ***SAVES*** **ALL** the **OBJECTS** IN THE **ARRAY** to the `UserAccountData.txt` file. ▪ Algorithm: **1.** ***CREATE*** `PrintWriter`, `BufferredWriter` **&** `FileWriter` **OBJECTS** to ***OPEN*** `UserAccountData.txt` File for **WRITING (NOT APPEND)**. **2.** ***SEARCH/LOOP*** though **ARRAY** `(skipping empty cells)` **3. FOR EVERY OBJECT IN ARRAY do the following:** **a)** ***GET ALL*** THE **PROPERTIES** by ***CALLING*** **GETTER METHODS FOR EACH OBJECT** IN **ARRAY** **b)** AND ***CREATE*** A ***Comma-delimited STRING*** from **ALL THE PROPERTIES OF THE OBJECT IN ARRAY**. **c)** ***CALL*** THE `PrintWriter` OBJECT `println()` Method ***TO WRITE THE Comma-delimited STRING*** **LINE** to the **FILE**. **4.** REPEAT this process until **ALL OBJECTS IN ARRAY HAVE BEEN** ***VISITED*** **AND ITS PROPERTIES** ***WRITTEN*** **TO THE FILE AS A** ***Comma-delimited string***. **5.** Add **Error-Handling code** using `try-catch-finally` to handle all required ***I/O & GENERAL*** Exceptions & `Re-Throw` the Exception.

OOP Step 1 – ADD & UPGRADE an Employee Class

Homework Assignment # 4

Requirement #7 – ADD the Tested Employee Class from HW2 to the application to support the Employee Management features. In addition UPGRADE the print() method to print/write to a file & add Business Class requirements by adding Data Access Methods.

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

```
Employee

Private Data:
ssNumber:                                   String
firstName:                                  String
lastName:                                   String
dateOfBirth:                                String
age:                                        int
address:                                    String
phone:                                      String
email:                                      String
jobTitle:                                   String
count:                                      int

Public Getter/Setter Methods/Properties:
getSSNumber():                              String
setSSNumber():                              void
getFirstName():                             String
setFirstName(value):                        void
getLastName():                              String
setLastName(value):                         void
getDateOfBirth():                           String
setDateOfBirth(value):                      void
getAge():                                   int
getAddress():                               String
setAddress(value):                          void
getPhone():                                 String
setPhone(value):                            void
getEmail():                                 String
setEmail(value):                            void
getJobTitle():                              String
setJobTitle(value):                         void
getCount():                                 int
setCount(value):                            void

Private Constructor Methods:
private Employee()
private Employee ( S, F, L, DOB, A, P, E, J )

Public Methods:
public static print():                      void

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
```

Homework Assignment # 4

- **How to Re-use the Employee CLASS from a Previous Project to this one**: ADDING the **Employee Class** of previous **HW** to this **`Console Application`** PROJECT

  - To ADD the **UserAccount.java Class** to this **HW#2** Project take the following steps:

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

- **REQUIREMENTS 7a – NO CHANGES REQUIRED** to the data of the existing **`Employee Class`** from previous **HW**.

Scope	Data Member Name	Type	Description
private	ssNumber	String	▪ Stores the Employees Social Security number
private	firstName	String	▪ Stores the Employees first name
private	lastName	String	▪ Stores the Employees last name
private	dateOfBirthte	String	▪ Stores the Employees date of birth ▪ IMPORTANT Use the following FORMAT to store the dates - **MM/DDYYYY** - For example: 01/01/1971, which represents January 1st 1971.
private	age	**Integer**	▪ Stores the Employees age
private	address	**String**	▪ Stores the Employees address
private	phone	**String**	▪ Stores the Employees phone number
private	email	**String**	▪ Stores the Employees email address
private	jobTitle	**String**	▪ Stores the Employees job title
private static	count = 0	**Integer**	▪ **STATIC** class variable, intended to store a count of every employee object created. ▪ count is initialized to 0 at declaration

Homework Assignment # 4

- **REQUIREMENTS 7b – NO CHANGES REQUIRED** for the **EXISTING GETTER/SETTER METHODS** of the existing **`Employee Class`** from previous **HW**:

Scope	Getter/Setter Method/Property Name	Return Type	Parameter Type	Description
public	getSSNumber()	**String**	**None**	▪ **GETS/RETURNS** the ***ssNumber*** **private data**
public	setSSNumber(s)	**void**	**String**	▪ **SETS** the ***ssNumber*** **private data** with a **new VALUE**
public	getFirstName()	**String**	**None**	▪ **GETS/RETURNS** the ***firstName*** **private data**
public	setFirstName(f)	**void**	**String**	▪ **SETS** the ***firstName*** **private data** with a **new VALUE**
public	getLastName()	**String**	**None**	▪ **GETS/RETURNS** the ***larstName*** **private data**
public	setLarstName(l)	**void**	**String**	▪ **SETS** the ***lastName*** **private data** with a **new VALUE**
public	getDateOfBirth()	**String**	**None**	▪ **GETS/RETURNS** the ***dateOfBirth*** **private data**
public	setDateOfBirth(d)	**void**	**String**	1) **SETS** the ***dateOfBirth*** **private data** with a **new VALUE** 2) **Also Calculates the Age based on date of birth value and today's date**. 3) **Assigns** the calculated age from step 2 to the ***age*** **private data**.
public Read Only	getAge()	**Integer**	**None**	▪ **READ-ONLY** - ***GETS*** the ***age*** **private data** ▪ **READ ONLY because there will be NO SETTER METHOD Created.**
public	getAddress()	**String**	**None**	▪ **GETS/RETURNS** the ***address*** **private data**
public	setAddress(a)	**void**	**String**	▪ **SETS** the ***address*** **private data** with a **new VALUE**
public	getPhone()	**String**	**None**	▪ **GETS/RETURNS** the ***phone*** **private data**
public	setPhone(p)	**void**	**String**	▪ **SETS** the ***phone*** **private data** with a **new VALUE**
public	getEmail()	**String**	**None**	▪ **GETS/RETURNS** the ***email*** **private data**
public	setEmail(e)	**void**	**String**	▪ **SETS** the ***email*** **private data** with a **new VALUE**
public	getJobTitle()	**String**	**None**	▪ **GETS/RETURNS** the ***jobTitle*** **private data**
public	setJobTitle(l)	**void**	**String**	▪ **SETS** the ***jobTitle*** **private data** with a **new VALUE**
public static	getCount()	**Integer**	**None**	▪ **STATIC Getter/Setter Method/property** ▪ **GETS/RETURNS** the ***STATIC count*** **private data**.
public static	setCount(c)	**void**	**Integer**	▪ **STATIC Getter/Setter Method/property** ▪ **SETS** the ***count*** **STATIC private data** with a **new VALUE**.

Homework Assignment # 4

- **REQUIREMENTS 7c – – NO CHANGES REQUIRED** for the **EXISTING DEFAULT & PARAMETRIZED CONSTRUCTOR METHODS** of the existing **`Employee Class`** from previous **HW**:

Scope	Method Name	Return Type	Paramters	Processing/Description
	Employee	None	None	▪ **Default Constructor** 1) Sets the following ***data*** members to **DEFAULT VALUES** as follows: - ssSNumber = “” - firstName =”” - lastName =”” - dateOfBirth = “00/00/0000” - age = 0 - address =”” - phone =”” - email =”” - jobTitle =”” 2) **Increment Shared Private Data count by 1**
	Employee	None	▪ A parameter for to Set each of the following **PRIVATE DATA EXCEPT** the **AGE** & ***COUNT***: - ssNumber - firstName - lastName - dateOfBirth - address - phone - email - jobTitle ▪ Should have a total of **8** parameters: - **par1 to Par8** ▪ **All parameters are Pass-by-Value** ▪ **IMPORTANT! Name parameters as you see appropriate. Keep short**	- **Parameterized Constructor** - Sets parameter list to all data members via **PUBLIC SETTER METHODS** EXCEPT the ***AGE*** & ***COUNT***. - Match parameter list appropriately: - ssNumber = **par1** - firstName = **par2** - lastName = **par3** - dateOfBirth = **par4** - address = **par5** - phone = **par6** - email = **par7** - jobTitle = **par8** - **Age Private Data** is handled by ***Birthdate SETTER METHOD,*** no need to pass it as parameter. - **Increment Shared Private Data count by 1**

Homework Assignment # 4

- **REQUIREMENTS 7c – UPGRADE** the **EXISTING `print()` METHOD** of the **`Employee` `Class` to PRINT DATA TO TEXT FILE INSTEAD OF THE CONSOLE SCREEN**:

  - ***REMOVE* the System.out.println()** STATEMENTS in the ***`print()`* method**.
  - We are now enforcing that there should be **NO USER-INTERFACE CODE** in the **BUSINESS CLASS LAYER**!
  - Upgrade the Employee Class ***`print()`* method** to **WRITE & APPEND TO the TEXT FILE `Network_Printer.txt`**.

Scope	Method Name	Return Type	Parameters	Processing/Description
public	**print()**	None	None	▪ **UPGRADE** – The ***print()*** Method WRITES ALL OBJECT'S DATA EXCEPT the STATIC count TO THE ***Network_Printer.txt*** PRINTER FILE as follows: **1.** ***CREATE*** **`PrintWriter`, `BufferredWriter` & `FileWriter` OBJECTS** to ***OPEN*** **`Network_Printer.txt`** TEXT FILE for **APPENDING**. **2.** ***WRITES/APPENDS*** each object's **PROPERTY/GETTER METHODS** (***EXCEPT COUNT***) in the following FORMAT: ***Employee*** information: First Name = ***value*** Last Name = ***value*** Social Security = ***value*** Date Of Birth = ***value*** Address = ***value*** Age = value Email = ***value*** Title = ***value*** **3.** Add **Error-Handling code** using **`Try-Catch-Finally`** to handle all required ***Unique OR/AND GENERAL*** Exceptions & **`Re-Throw`** the Exception

Homework Assignment # 4

- **REQUIREMENTS 7g –** To the **EXISTING** `Employee` `Class` ***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**	**String**	▪ Public Data access method that DELETES a record from database. ▪ **Process it performs**: Calls `protected database_Delete(key)` to do the work

- **REQUIREMENTS 7h –** o the **EXISTING** `Employee` `Class` ***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**	**String**	▪ 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 an EmployeeList Class

Homework Assignment # 4

Requirement #8 – CREATE a EmployeeList Class Business Class to Manage & Encapsulate an ARRAY of Employee Objects

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

```
EmployeeList

Private Data:
private SIZE:                          int
private arrEmployeeList[]:             Employee

Public Default Constructor Method:
public EmployeeList()

Public Methods:
public search(ssNumber):               Employee
public add(Employee):                  boolean
public add(x,y,z.):                    Boolean
public edit(ssNumber, Employee):       boolean
public edit(x,y,z.):                   boolean
public remove(ssNumber):               boolean
public clear():                        void
public print(ssNumber):                Boolean
public printAll():                     void

Public Data Access Methods:
public void load():                    void
public void save():                    void

Public Data Access Methods:
protected void database_Load():        void
protected void database_Save():        void
```

Homework Assignment # 4

- **REQUIREMENTS 8a –** **EmployeeList** 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`	`arrEmployeeList[]`	`Employee`	▪ **POINTER** DECLARATION of ARRAY of type **Employee** Class

- **REQUIREMENTS 8b –** **EmployeeList** Class **DEFAULT CONSTRUCTOR**:

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

Homework Assignment # 4

- **REQUIREMENTS 8c –** **EmployeeList** Class **PROCESSING METHODS** :

Scope	Name	Return Type	Parameters	Description
public	search	**Employee**	**String username**	▪ ***SEARCH*** **`arrEmployeeList`** **ARRAY** – Method that performs a search of the ARRAY for the **Employee object** who's **SSNumber** is passed as parameter. ***RETURNS*** a **POINTER** to the object found or returns a **NULL** if not found. ▪ Algorithm: **1.** ***SEARCH*** **`ARRAY`** for **`OBJECT whose KEY/SSNUMBER`** is passed as parameter. **2.** **IF FOUND**, ***RETURNS*** the **`POINTER`** to the **`OBJECT`** in **`ARRAY`** & **Exits** the **`Method`**. **3.** **ELSE IF NOT FOUND**, ***RETURNS*** a **`null`** indicating object was not found. **4.** Add **Error-Handling code** using **`Try-Catch-Finally`** to handle all required ***Unique OR/AND GENERAL*** Exceptions & **`Re-Throw`** the Exception.
public	add	**boolean**	**Employee POINTER**	▪ ***ADD*** **OBJECT** to **`arrEmployeeList`** **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. ▪ Algorithm: **1.** ***SEARCH*** **`ARRAY`** for **`Empty`** cell. **2.** **IF FOUND**, ***ADDS*** the **`OBJECT`** to the **`ARRAY`** & **`Return True`** to **Exit** the **`Method`**. **3.** **ELSE IF** **`Empty`** cell **NOT FOUND**, **`Return false`** to **Exit** the **`Method`** indicating **ARRAY IS FULL**. **4.** Add **Error-Handling code** using **`Try-Catch-Finally`** to handle all required ***Unique OR/AND GENERAL*** Exceptions **`Re-Throw`** the Exception.
public	add	**boolean**	Values with appropriate type that make up a **Employee** Object: - **SSNumber** - **FirstName** - **LastName** - **Birthdate** - **Address** - **PhoneNumber** - **Email** - **JobTitle**	▪ ***(OVERLOADED)*** ***ADD*** **OBJECT** to **`arrEmployeeList`** **ARRAY** – Method ***CREATES*** a new **Employee 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 ▪ Algorithm: **1.** ***CREATE*** a Temp **OBJECT** **2.** populates it with **VALUES** from parameters **3.** ***SEARCH*** **`ARRAY`** for **`Empty`** cell. **4.** **IF FOUND**, ***ADD*** the **`OBJECT`** to the **`ARRAY`** & **Exit** the **`Method`**. **5.** **ELSE IF** **`Empty`** cell **NOT FOUND**, **`Return false`** to **Exit** the **`Method`** indicating **ARRAY IS FULL**. **6.** Add **Error-Handling code** using **`Try-Catch-Finally`** to handle all required ***Unique OR/AND GENERAL*** Exceptions **`Re-Throw`** the Exception.

Homework Assignment # 4

- **REQUIREMENTS 8c – EmployeeList** Class **PROCESSING METHODS**:

Scope	Name	Return Type	Parameters	Description
public	edit	**boolean**	**String ssNumber,** **Employee POINTER**	▪ ***EDITS*** **OBJECT** in **`arrEmployeeList`** **ARRAY** – Method Search ARRAY for OBJECT whose **SSNumber** is passed as parameter. If found in ARRAY, object in ARRAY is ***MODIFIED*** by **SETTING ALL THE PROPERTIES** with **VALUES** from **OBJECT** passed as parameter. NOTE that the ***EXCEPTION*** is the **SSNumber** WHICH IS NOT MODIFIED!!! ▪ Returns a **TRUE** if object found and EDITED. Returns **FALSE** if object not found in ARRAY. ▪ Algorithm: **1.** ***SEARCH*** **`ARRAY`** `(skipping empty cells)` for **`OBJECT`** who's **KEY/SSNUMBER** is passed as parameter. **2.** If found, ***EDIT*** the **`OBJECT`** in the **`ARRAY`** by ***SETTING/CALLING*** **SETTER METHODS** with data from **`Employee Object Pointer`** passed as parameter**`. In other words, you are`** ***`GETTING`*** **`the PROPERTIES of obEmployee Object Parameter and`** ***`SETTING`*** **`the PROPERTIES of OBJECT found in ARRAY a location arrEmployeeList(Index)`**. **3.** **`Return True`** to **Exit** the **`Method`** ❖ **DO NOT SET the SSNUMBER properties** since is the **KEY** or **ID** and should NOT be tampered with. So **SET all properties EXCEPT the SSNUMBER** **4.** If not found, returns a **`False`**. **5.** Add **Error-Handling code** using **`Try-Catch-Finally`** to handle all required ***Unique OR/AND GENERAL*** Exceptions & **`Re-Throw`** the Exception.
public	edit	**boolean**	Values with appropriate type that make up a **Employee** Object: - **SSNumber** - **FirstName** - **LastName** - **Birthdate** - **Address** - **PhoneNumber** - **Email** - **JobTitle**	▪ ***(OVERLOADED) EDITS*** **OBJECT** in **`arrEmployeeList`** **ARRAY** – Method Search ARRAY for OBJECT whose **SSNUMBER** is passed as parameter. If found in ARRAY, object in ARRAY is ***MODIFIED*** by **SETTING ALL THE PROPERTIES** with **VALUES** passed as **parameter**. NOTE that the ***EXCEPTION*** is the **SSNumber** WHICH IS NOT MODIFIED!!!! ▪ Returns a **TRUE** if object found and MODIFIED. Returns **FALSE** if object not found in ARRAY ▪ Algorithm: **1.** ***SEARCH*** **`ARRAY`** for **`OBJECT`** who's **KEY/SSNUMBER** is passed as parameter. **2.** If found, ***EDIT*** the **`OBJECT`** in the **`ARRAY`** by ***SETTING/CALLING*** **SETTER METHODS** with data from **`VALUES`** passed as parameters**`. In other words, you are`** ***`SETTING`*** **`the PROPERTIES of OBJECT found in ARRAY a location arrEmployeeList(Index)`** with **`VALUES`** from the parameter list. **3.** **`Return True`** to **Exit** the **`Method`** ❖ **DO NOT SET the SSNUMBER properties** since is the **KEY** or **ID** and should NOT be tampered with. So **SET all properties EXCEPT the SSNUMBER** **4.** If not found, returns a **`False`**. **5.** Add **Error-Handling code** using **`Try-Catch-Finally`** to handle all required ***Unique OR/AND GENERAL*** Exceptions & **`Re-Throw`** the Exception.

Homework Assignment # 4

- **REQUIREMENTS 8c –** **EmployeeList** Class **PROCESSING METHODS**:

Scope	Name	Return Type	Parameters	Description
public	remove	**boolean**	**String ssNumber**	▪ ***REMOVES*** **OBJECT** from the **`arrEmployeeList`** **ARRAY** – Method Search ARRAY for OBJECT whose **SSNAUMBER** is passed as parameter. REMOVES OBJECT from ARRAY by setting **`arrEmployeeList[i]`** POINTER to NULL. RETURNS a TRUE if found and REMOVED or returns FALSE otherwise ▪ Algorithm: **1.** ***SEARCH*** **`ARRAY`** `(skipping empty cells)` for **`OBJECT`** who's **KEY/SSNUMBER** is passed as parameter. **2.** If found, **DELETE** the **`OBJECT`** from **`ARRAY`** & **`Return True`** to **Exit** the **`Method`**. **3.** If not found, returns a **`False`**. **4.** Add **Error-Handling code** using **`Try-Catch-Finally`** to handle all required ***Unique OR/AND GENERAL*** Exceptions & **`Re-Throw`** the Exception
public	print	**boolean**	**String ssNumber**	▪ Method performs the Process of ***PRINTING THE TARGET*** **Employee** Object in the **`arrEmployeeList`** **ARRAY** TO FILE by SEARCHING & FINDING the **Employee OBJECT** and ***CALLING*** it's **`print() METHOD`** TO DO THE WORK. RETURNS a TRUE if found and PRINTED TO FILE or returns FALSE indicating OBJECT NOT FOUND. ▪ Algorithm: **1.** ***SEARCH*** **`ARRAY`** `(skipping empty cells)` for **`OBJECT`** who's **SSNUMBER** is passed as parameter. **2.** If found, **Calls** the **`Object's`** ***`print()`*** method to do the work & **`Return True`** to **Exit** the **`Method`**. **3.** If not found, returns a **`False`**. **4.** Add **Error-Handling code** using **`Try-Catch-Finally`** to handle all required ***Unique OR/AND GENERAL*** Exceptions & **`Re-Throw`** the Exception
public	printAll	**void**	**None**	▪ Method performs the Process of ***PRINTING ALL OBJECTS IN THE*** **`arrEmployeeList`** **ARRAY** TO FILE by SEARCHING **`arrEmployeeList`** **ARRAY** and ***CALLING*** **EACH OBJECT'S** **`print() METHOD`** TO DO THE WORK. NO RETURN VALUE. ▪ Algorithm: **1.** ***SEARCH*** **`ARRAY`** `(skipping empty cells)`. **2.** FOR EACH **`Object`** in the **`arrEmployeeList`** **ARRAY** **CALLS** **`IT'S`** ***`print()`*** method to do the work. **3.** Add **Error-Handling code** using **`Try-Catch-Finally`** to handle all required ***Unique OR/AND GENERAL*** Exceptions & **`Re-Throw`** the Exception

Homework Assignment # 4

- **REQUIREMENTS 8c (cont.) – EmployeeList** Class **PROCESSING METHODS**:

Scope	Name	Return Type	Parameters	Description
public	clear	**void**	**none**	▪ ***CLEARS*** the `arrEmployeeList` **ARRAY** of **OBJECTS** – Method ***CLEARS*** the ARRAY of Objects by setting all pointers to **NULL**. ▪ Algorithm: **1.** ***SEARCH*** `ARRAY` and **DELETE EVERY** `Objects` from Array using ANY METHOD YOU DESIRE, BUT FOLLOW BEST PRACTICE IF POSSIBLE. **2.** Add **Error-Handling code** using `Try-Catch-Finally` to handle all required ***Unique OR/AND GENERAL*** Exceptions & `Re-Throw` the Exception

- **REQUIREMENTS 8d – EmployeeList** 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

Homework Assignment # 4

- **REQUIREMENTS 8e – EmployeeList** 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 TEXT FILE & populates the object with data retrieved from database. ▪ ***LOAD*** the **OBJECTS** from the **`EmployeeData.txt`** file and ***ADDS*** them to the **`arrEmployeeList`** **ARRAY**. ▪ Algorithm: **1. VERIFY FILE EXISTS before READING.** **2. If FILE DOES NOT EXISTS IT CREATES IT**. **3.** ***CREATE*** **`BufferredReader`** **&** **`FileReader`** **OBJECTS** to ***OPEN*** **`EmployeeData.txt`** File for **READING**. **4. Read a LINE** from file & ***PARSE*** each ***comma-delimited line***. **5.** ***CREATE*** a **NEW** temporary **objEmployee OBJECT** of the **Employee CLASS**. **6.** ***SET*** **OBJECT WITH VALUES** from **PARSED LINE READ FROM FILE**. **7.** ***ADD*** **OBJECT** TO **ARRAY** by ***CALLING*** the **Add(objEmployee) METHOD OF THIS CLASS, TO DO THE WORK**. **8.** ***REPEAT*** STEP 4 through 7 process until **EOF**. **9.** Add **Error-Handling code** using **`try-catch-finally`** to handle all required ***I/O & GENERAL*** Exceptions & **`Re-Throw`** the Exception.
protected	Database_Save	**void**	**none**	▪ Protected Data access method that actually performs the SAVING of ALL RECORDS to FILE. ▪ ***SAVES*** **ALL** the **OBJECTS** IN THE **`arrEmployeeList`** **ARRAY** to the **`EmployeeData.txt`** file. ▪ Algorithm: **1.** ***CREATE*** **`PrintWriter`**, **`BufferredWriter`** **&** **`FileWriter`** **OBJECTS** to ***OPEN*** **`EmployeeData.txt`** File for **WRITING (NOT APPEND)**. **2.** ***SEARCH/LOOP*** though **ARRAY** `(skipping empty cells)` **3. FOR EVERY OBJECT IN ARRAY do the following:** **a)** ***GET ALL*** THE **PROPERTIES** by ***CALLING*** **GETTER METHODS FOR EACH OBJECT** IN **ARRAY** **b)** AND ***CREATE*** A ***Comma-delimited STRING*** from **ALL THE PROPERTIES OF THE OBJECT IN ARRAY**. **c)** ***CALL*** THE **`PrintWriter`** OBJECT **`println()`** Method ***TO WRITE THE Comma-delimited STRING*** **LINE** to the **FILE**. **4.** REPEAT this process until **ALL OBJECTS IN ARRAY HAVE BEEN** ***VISITED*** **AND ITS PROPERTIES** ***WRITTEN*** **TO THE FILE AS A** ***Comma-delimited string***. **5.** Add **Error-Handling code** using **`try-catch-finally`** to handle all required ***I/O & GENERAL*** Exceptions & **`Re-Throw`** the Exception.

Database Layer – Implement Using Text Files

Create Text Files – to be managed by Business Object Layer

Homework Assignment # 4

Requirement #9 – Leverage the EmployeeData.txt File & UserAccountData.txt File to stored Employeee & User Account Records for Permanent Storage

- Note **that the Management of Permanent Storage via e FILES** is managed by the **BUSINESS OBJECT LAYER**. **EmployeeList** & **UserAccountList** **Classes**.
- **THEREFORE THERE IS REALLY NOTHING FOR YOU TO DO IN THIS REQUIREMENT**, **JUST MAKE SURE IS DONE IN THE BUSINESS OBJECTS LAYER EmployeeList** & **UserAccountList CLASSES**.

  - **Employee Management Permanent Storage Support**:

    - The **EmployeeList CLASS** CREATES & MANAGES a **TEXT FILE** named **EmployeeData.txt** to store ALL **Employee Records** as a ***COMMA-DELIMITED LINE*** (***Details in Class Requirements***):

Employee.txt - Notepad

File Edit Format View Help

```
1111,Joe,Smith,01/01/1971,111 Jay St,718-111-1111,Manager,jsmith@mycomp.com
2222,Angel,Rodriguez,02/02/1972,222 Flatbush Ave,718-222-2222,Director,arodriguez@ mycomp.com
3333,Sam,Peterson,03/03/1973,333 Dekalb Ave,718-333-3333,Office Admin,speterson@ mycomp.com
4444,Mary,Johnson,04/04/1974,444 Jay Street,718-444-4444,VP,mjohnson@ mycomp.com
5555,Nancy,Rivera,05/05/1975,555 Flatlands Ave,718-555-5555,Secretary,nrivera@ mycomp.com
```

  - **User Acccount Management Permanent Storage Support**:

    - The **UserAccountList CLASS** CREATES & MANAGES a **TEXT FILE** named **UserAccountData.txt** to store ALL **User Account Records** as a ***COMMA-DELIMITED LINE*** (***Details in Class Requirements***):

UserAccount.txt - Notepad

File Edit Format View Help

```
3e31e4e3-139e-4e83-8beb-c99f39cc080a,joe,111,jsmith@mycomp.com
e1312cc6-66ce-45f9-8b3c-54b5f3ed4961,angel,222,arodriguez@mycomp.com
2ddf6b43-3713-4be1-8425-497c9500f428,sam,333,speterson@mycomp.com
59db2950-3248-4850-882a-b39ef987afc9,mary,444,mjohnson@mycomp.com
960f49fc-0451-40a3-bf4f-1a67b7d73e3a,nancy,555,nrivera@mycomp.com
```

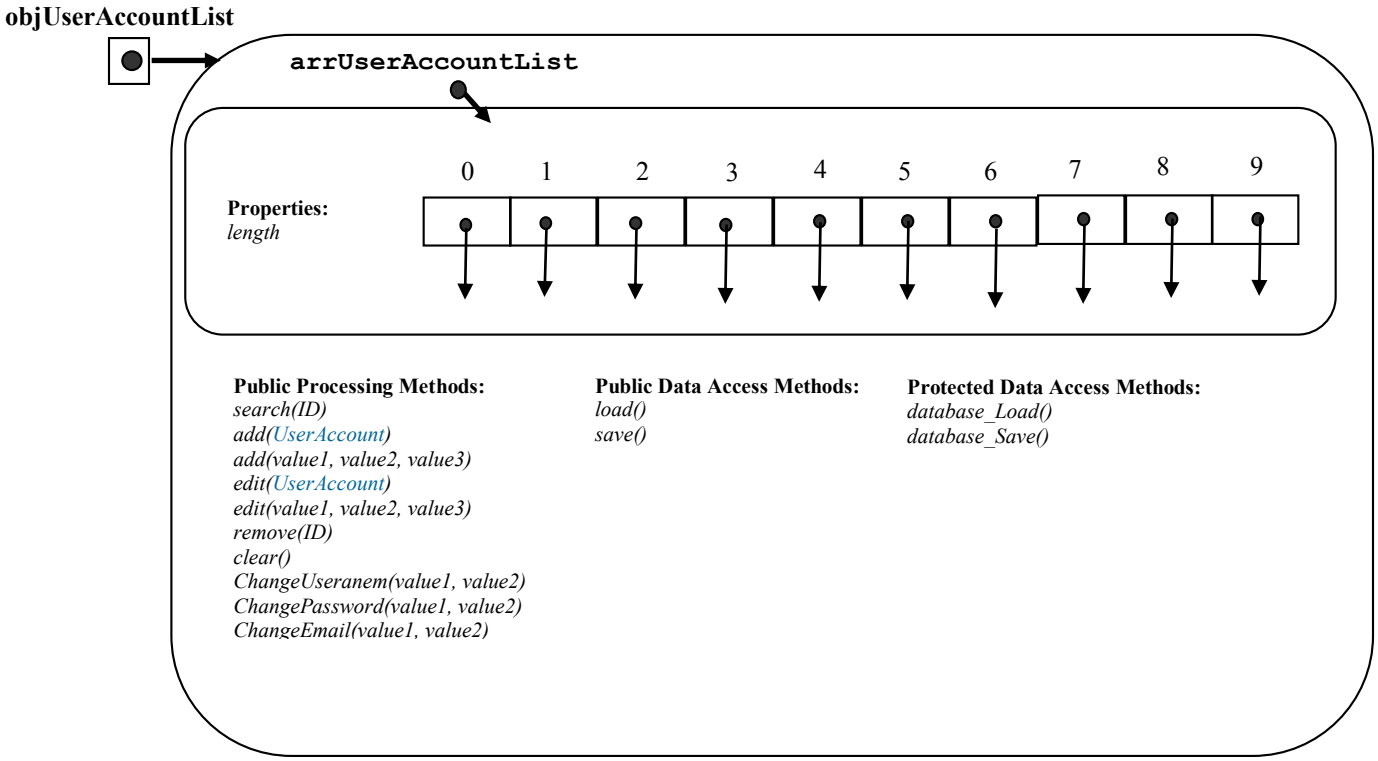
Update the Presentation/User-Interface Layer

Main Class Requirements – UserAccountList Object

Homework Assignment # 4

Requirement #10 – CONTINUE to use the 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. Continue to use this array:



- This OBJECT of the *UserAccountList* Class encapsulates an ARRAY of **10 Cells storing** *UserAccount* Class **POINTERS**.

Main Class Requirements – EmployeeList Object

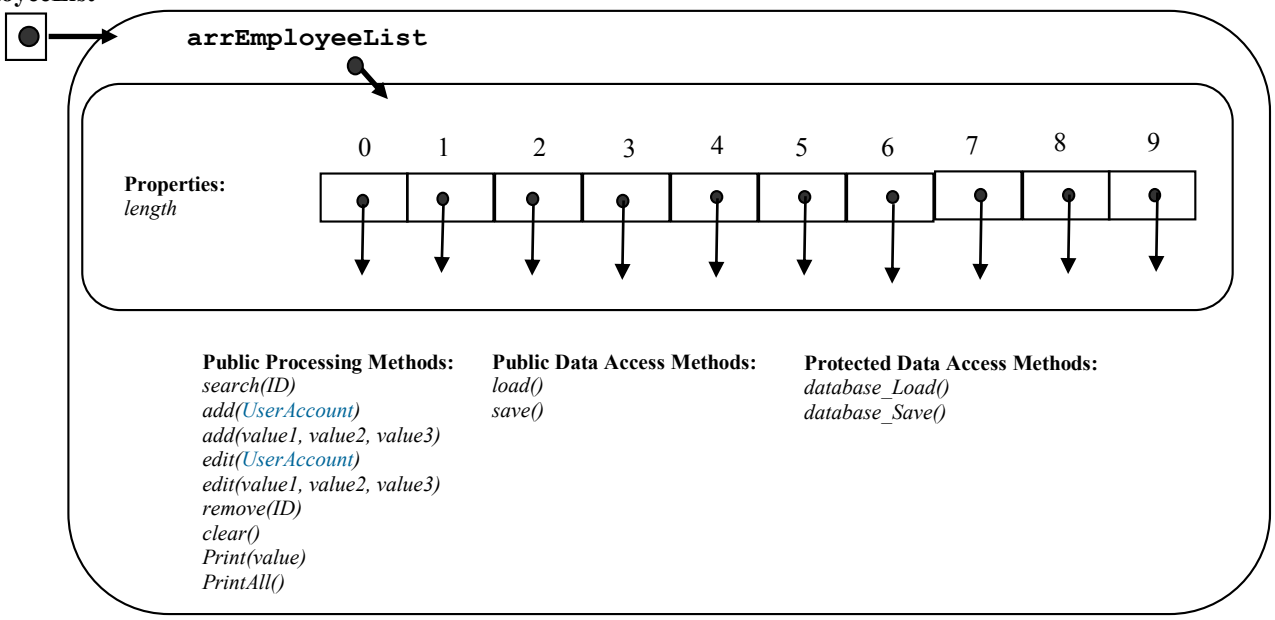
Homework Assignment # 4

Requirement #11 – CREATE a GLOBAL STATIC OBJECT objEmployeeList of the EmployeeList Class to manage storage of Employee Objects IN-MEMORY

- The company's **Employee Credential** objects will be stored in memory via a **GLOBAL OBJECT** of the **EmployeetList** Class and available to all the program code.

  - **In the MAIN CLASS, *CREATE* a GLOBAL STATIC OBJECT, named `objEmployeeList` *of EmployeeList Class:***

objEmployeeList



  - This OBJECT of the ***EmployeeList*** Class encapsulates an ARRAY of **10 Cells storing *Employee*** Class **POINTERS**. This is how Employees will be managed & populated with ***Employee* Class OBJECTS** in **MEMORY** throughout the usage of the program.
  - **IMPORTANT! DECLARE the OBJECT OF *EmployeeList* 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 STATIC OBJECT of EmployeeList CLASS here IN DECLARATION OF MAIN CLASS

      public static void main(String[] args) {


      }//End of main

}//End of program class
```

Main Class Requirements – main() & Authenticate Methods

Homework Assignment # 4

Requirement #12 – NO CHANGES REQUIRED TO 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.
- **NO CHANGES REQUIRED FOR THE DRIVER METHOD**:

Scope	Method Name	Return Type	Parameters	Processing/Description
public	**static void main ()**	**N/A**	**String[] args**	▪ **SAME AS PREVIOUS HW.**

Requirement #13 – NO CHANGES REQUIRED for the STATIC Authenticate(u,p) method

- **NO CHANGES REQUIRED FOR THE METHOD**:

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	▪ **SAME AS PREVIOUS HW.**

Main Class Requirements – User Account Management Screens & UI Processing

Homework Assignment # 4

Requirements # 14 – MAINTAIN ALL User-Interface Screens, Navigation flow & Control Requirements from PREVIOUS HW

- KEEP ALL UI SCREENS, CONTROL & NAVIGATION FLOW from previous HW:

  - `Login Screen`.
  - `Main/Welcome Screen`.
  - `Back-end Management Screen`.
  - `User Account Management Screen`.
  - `Retail Point-of-Sales Screen`.

- **FORWARD & BACKWARDS User Account Management Navigation Flow #1:**

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

- **FORWARD & BACKWARDS User Account Management 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 Retail POS Navigation Flow #3:**

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

Main Class Requirements – New Employee Management Screens & UI Processing

Homework Assignment # 4

Requirements # 15 – New Employee Management User-Interface Screens Navigation Flow & Control Requirements

- We will add several new User-interface screens to support the Employee Management features:

  - `Employee Management Screen.`
  - `Search Employee I/O & Results Screen.`
  - `Add Employee I/O & Results Screen.`
  - `Edit Employee I/O & Results Screen.`
  - `Remove Employee I/O & Results Screen.`
  - `Print Employee I/O & Results Screen.`
  - `Print All Employees I/O & Results Screen.`

- Navigation Flow and how these forms interact is described in section below.

Requirements #15a – Employee Management Navigation Flow #1: Login, Main/Welcome, Back-end Management & Employee Management Screens Navigation Flow

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

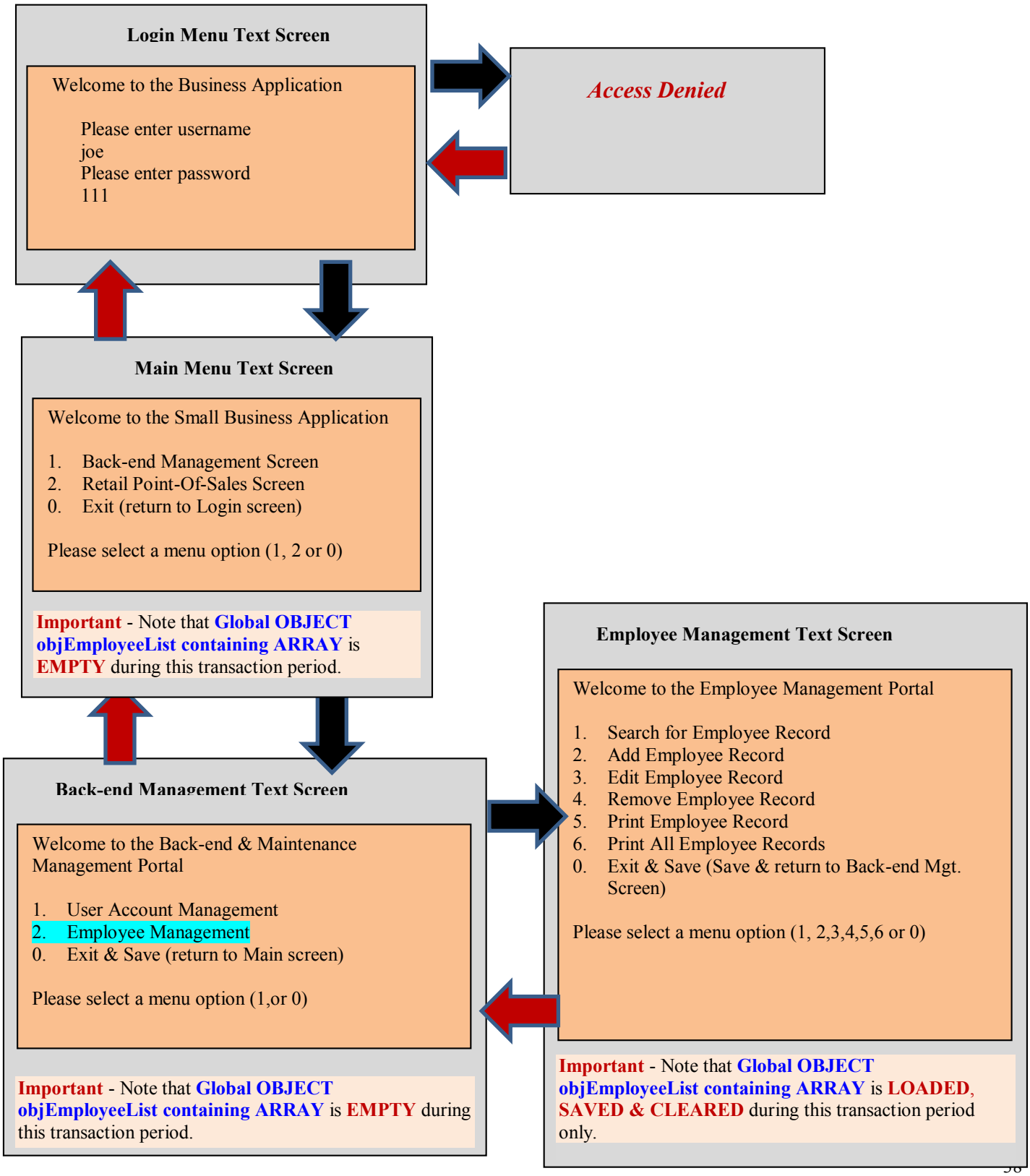
  - `Login Screen.`
  - `Main/Welcome Screen.`
  - `Back-end Management Screen.`
  - `Employee Management Screen.`

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

  `Login Screen <-> Main/Welcome Screen <-> Back-end Management Screen <-> Employee Management Screen`

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

Requirements #15a (Cont.) – Employee Management Navigation Flow #1: Login, Main/Welcome, Back-end Management & Employee Management Screens Navigation Flow Diagram



Homework Assignment # 4

Requirements #15b – Employee Management Navigation Flow #2: User Account Management & Associated Screens Navigation Flow

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

  - **Employee Management Screen.**
  - **Search Employee I/O & Results Screen.**
  - **Add Employee I/O & Results Screen.**
  - **Edit Employee I/O & Results Screen.**
  - **Remove Employee I/O & Results Screen.**
  - **Print Employee I/O & Results Screen.**

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

  Employee Management <-> Search Employee I/O & Results Screen

  Employee Management <-> Add Employee I/O & Results Screen

  Employee Management <-> Edit Employee I/O & Results Screen

  Employee Management <-> Remove Employee I/O & Results Screen

  Employee Management <-> Print Employee I/O & Results Screen

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

Requirements #15b – Employee Management Navigation Flow #2: Employee Management & Associated Screens Navigation Flow Diagram

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

Employee Management Text Screen

Welcome to the Employee Management Portal

1. Search for Employee Record
2. Add Employee Record
3. Edit Employee Record
4. Remove Employee Record
5. Print Employee Record
6. Print All Employee Records
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.

Search Employee Screen & Results

Search Employee Record

Please enter SSNumber
1111

Search Results
Record Information:
First Name = Joe
Last Name = Smith
Social Security = 1111
Date of Birth = 01/01/1971
Address = 111 Jay Street
Age = 44
Email = JSmith@comp1.com
Title = Manager

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

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

Employee Management Text Screen

Welcome to the Employee Management Portal

1. Search for Employee Record
2. Add Employee Record
3. Edit Employee Record
4. Remove Employee Record
5. Print Employee Record
6. Print All Employee Records
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.

Add Employee Screen & Results

Add Employee Record

Please enter NEW SSNumber:
7777
Please enter NEW first name:
Frank
Please enter NEW last name:
Hum
Please enter NEW DOB:
07/07/1977
Please enter NEW address:
777 Madison Avenue
Please enter NEW Email:
FHum@comp1.com
Please enter NEW job title:
Senior VP
Add Results:
Employee Record Added Successfully

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

Navigation Flow:

Employee Management <-> Edit Employee I/O & Results Screen

Employee Management Text Screen

Welcome to the Employee Management Portal

1. Search for Employee Record
2. Add Employee Record
3. Edit Employee Record
4. Remove Employee Record
5. Print Employee Record
6. Print All Employee Records
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.

Edit Employee Screen & Results

Edit Employee Record

Please SSNumber of account to search & Edit
5555
Please enter NEW first name:
Nancy
Please enter NEW last name:
Roberts
Please enter NEW DOB:
05/05/1975
Please enter NEW address:
555 Park Avenue
Please enter NEW Email:
NRoberts@comp1.com
Please enter NEW job title:
Administrative Assistance

Edit Results:
Employee Edited Successfully

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

Navigation Flow:

Employee Management <-> Remove Employee I/O & Results Screen

Employee Management Text Screen

Welcome to the Employee Management Portal

1. Search for Employee Record
2. Add Employee Record
3. Edit Employee Record
4. Remove Employee Record
5. Print Employee Record
6. Print All Employee Records
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.

Remove Employee Screen & Results

Remove Employee Record

Please enter SSNumber of record to delete
3333

Remove Results:
Employee Record Deleted Successfully

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

Navigation Flow:

Employee Management <-> Print Employee I/O & Results Screen

Employee Management Text Screen

Welcome to the Employee Management Portal

1. Search for Employee Record
2. Add Employee Record
3. Edit Employee Record
4. Remove Employee Record
5. Print Employee Record
6. Print All Employee Records
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.

Change Employee e Screen & Results

Print Employee Record

Enter SSNumber of Employee Record to print:
1111

Print Results:
Employee record printed to file successfully

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

Navigation Flow:

Employee Management & Print All Employees I/O (NO RESULTS SCREEN! PRINTS TO FILE)

Employee Management Text Screen

Welcome to the Employee Management Portal

1. Search for Employee Record
2. Add Employee Record
3. Edit Employee Record
4. Remove Employee Record
5. Print Employee Record
6. Print All Employee Records
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.

Homework Assignment # 4

Requirements #15c – 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 <-> Employee Management Screen`

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

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

Employee Management Screen <-> Add Employee I/O & Results Screen

Employee Management Screen <-> Edit Employee I/O & Results Screen

Employee Management Screen <-> Remove Employee I/O & Results Screen

Employee Management Screen <-> Print Employee I/O & Results Screen
```

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 # 4

Requirements #15c (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:**

`Back-end Management Screen <-> Employee 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:**

`Employee Management <-> Search Employee 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.
`Employee Management <-> Add Employee 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.
`Employee Management <-> Edit Employee 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 # 4

Requirements # 16 – UPGRADE The Back-end Management Screen: User Interface & Processing Requirements to Support Employee Management

- **`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 <-> Employee Management Screen`

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

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

Requirements #16a – 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
2. Employee Management
0. Exit (return to Main Screen)

Please select a menu option (1, 2 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**.

Homework Assignment # 4

Requirements # 16 (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
  2) Employee Management
  0) Exit

  - The following action is taken based on user selection:

    - **Selecting either option 1 – *DISPLAYS* a text-based `User Account Management Screen`.**
    - **Selecting either option 2 – *DISPLAYS* a text-based `Employee 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 <-> Employee 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.

INCOMPLETE GOING FORWARD!

Homework Assignment # 4

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.

Homework Assignment # 4

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

Add Results:
User Account Added Successfully

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

Homework Assignment # 4

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 password: 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.

Homework Assignment # 4

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 email: 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 # 4

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 # 4

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 # 4

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

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 #1 – Login Authentication Test

Homework Assignment # 4

Test Script

- **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 Script (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 Script (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 Script (Cont.)

- Test Script (Cont.):

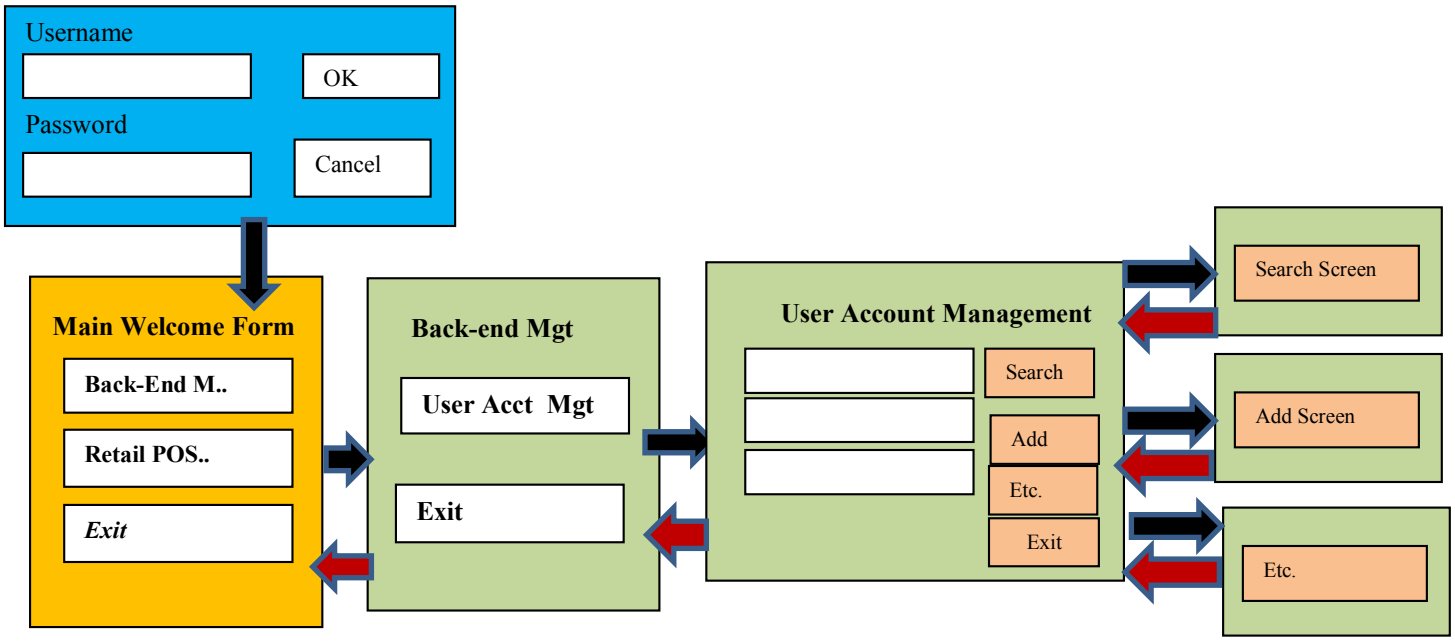
Test	Steps/Action	EXPECTED Results	Explanation
Test 7 – Test "END of program process" using **valid** ***Username*** & ***Password = -1***	**Step 2:** In Login Screen **Username** = **joe** **Password** = **-1**	1) Text screen should displays: "**Access Denied**" 2) Login screen Should Display Again	▪ 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 # 4

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 # 4

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 #3 – Add & Search User Account Test

Homework Assignment # 4

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

TEST 3 – User Account Management Form ADD & SEARCH Test

- In this test we will test **ADD** a ***NEW*** **User Account RECORD** then ***VALIDATING*** by **SEARCHING** for the **RECORD *ADDED* to verify it was ADDED.**

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)

Add User Account Screen & Results

Add New User Account

Please enter NEW Username
frank
Please enter NEW Password
777
Please enter NEW Email
fhum@comp1.com

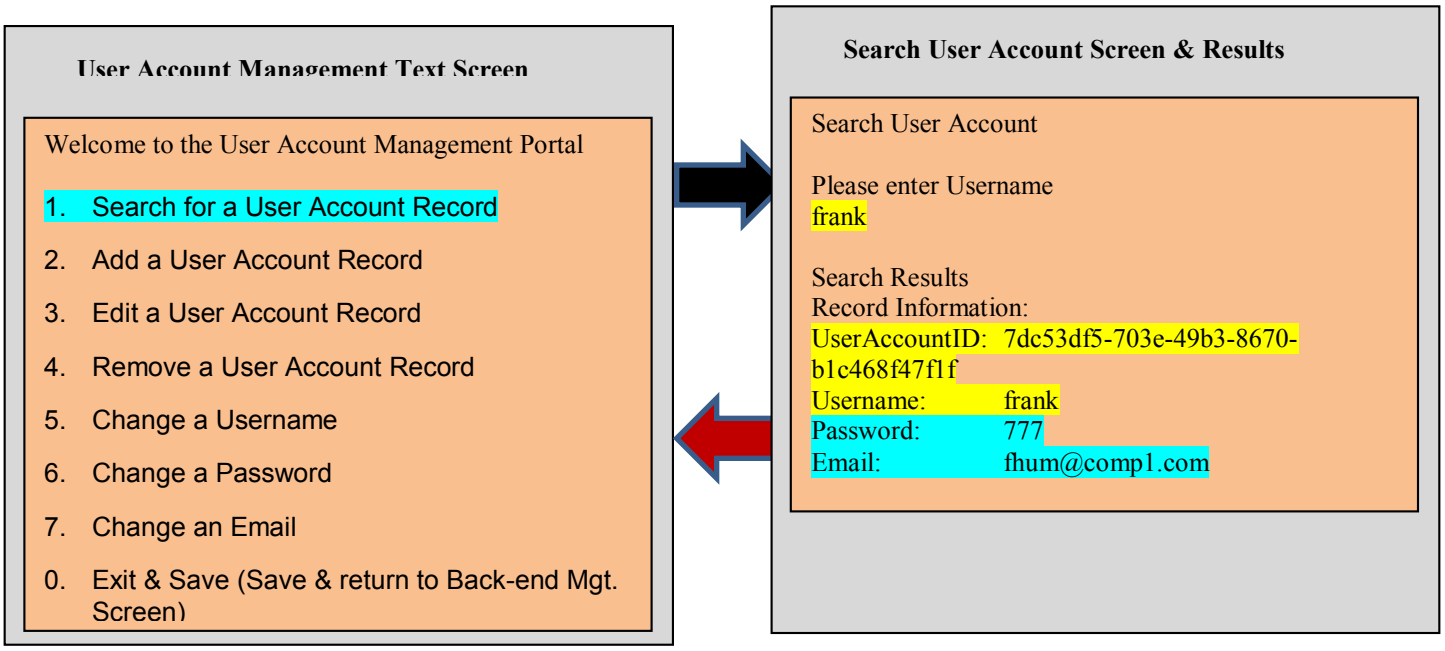
Add Results:
User Account Added Successfully

- **TEST 3a** – Test Script:

Test	Steps/Action	EXPECTED Results	Explanation
Test 3a – Testing the **User Account Management ADD** feature by ***ADDING*** new record & ***SEARCHING*** to verify it was ***ADDED***.	**Step 1:** From the **User Account Management Screen *SELECT*** option #2 **Add User Account Record**	▪ The `Add User Account I/O & Results Screen` **is displayed**.	▪ Screen is displayed.
	Step 2: In the `Add Screen` ***Enter*** the following values as prompted: **Username = frank**, **New Password: 777** **New Email: fhum@comp1.com**	▪ **Message is displayed stating that User Account Record ADDED successfully**.	▪ NEEW **User Account record** for **frank** is ***ADDED*** in the **arrUserAccountList** ARRAY inside the **objUserAccountList** object.

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

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



- **TEST 3b** – Test Script:

Test	Steps/Action	EXPECTED Results	Explanation
Test 3b – **VALIDATING RECORD** was ***ADDED*** by performing a **SEARCH** for the **RECORD** & ***DISPLAYING*** to confirm **ADDITION**.	**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 **Search Screen** ***Enter*** the following **User Account** ***Username*** values when prompted: **Username = frank**,	▪ The record of the **User Account RECORD whose Username = frank** is **DISPLAYING REFLECTING THE ADDITION WAS MADE**	▪ **Username** is needed in order to find the User Account. ▪ The **record** of the user is ***DISPLAYED*** by the **SEARCH** & proving that the **CHANGES** took place.

Test #4 – Edit & Search User Account Test

Homework Assignment # 4

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

TEST 4 – User Account Management Form EDIT & SEARCH Test

- In this test we will test **EDIT an *EXITING* User Account RECORD** then ***VALIDATING*** by **SEARCHING & verifying** the **RECORD WAS *EDITED*. Both the EDIT & SEARCH FEATURES ARE TESTED!**

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)

Edit User Account Screen & Results

Edit User Account

Please enter Username of customer to Edit
nancy
Please enter New Password
515
Please enter New Email
nroberts@comp1.com

Edit Results:
User Account Edited Successfully

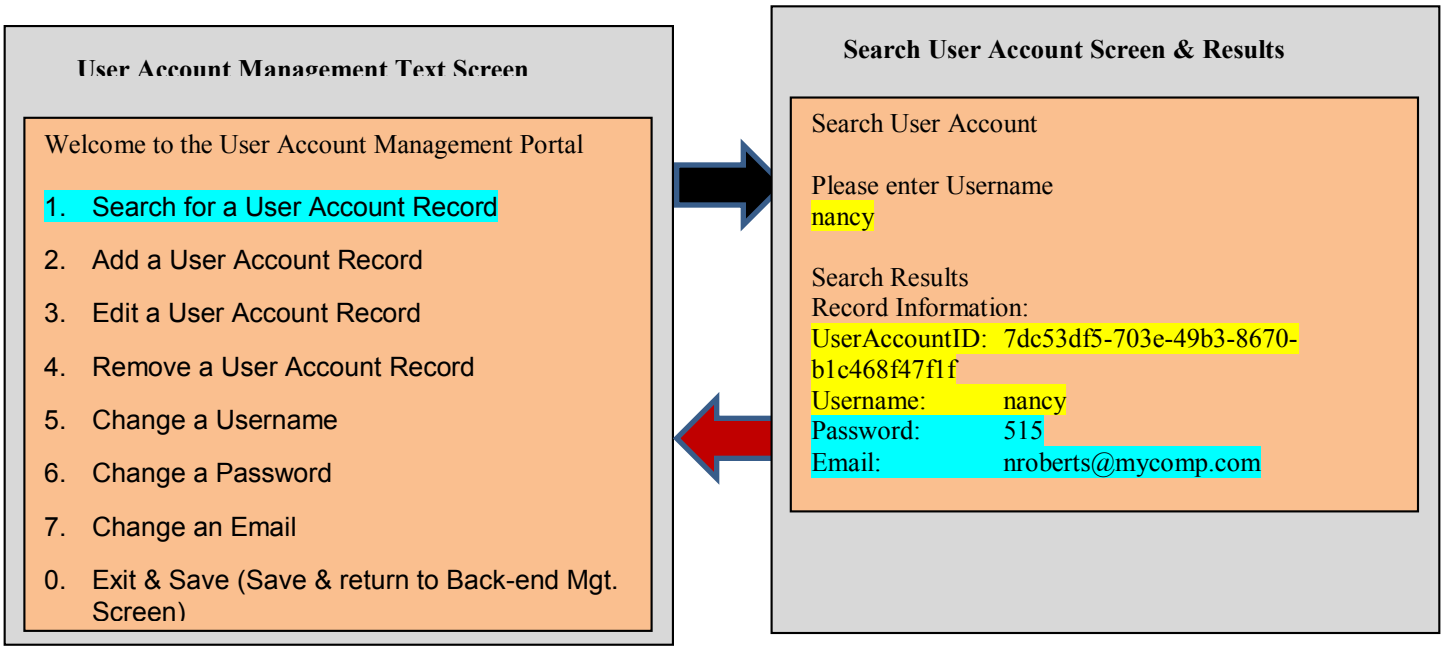
- **TEST 4a** – Test Script:

Test	Steps/Action	EXPECTED Results	Explanation
Test 4a – 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 Screen`** ***Enter*** the following values as prompted: **Username = nancy**, **New Password: 515** **New Email:** nroberts@comp1.com	▪ **Message is displayed stating that User Account Record *MODIFIED* successfully**.	▪ **Username** is needed in order to find the User Account record to modify. ▪ The **User Account record** for **nancy** is ***MODIFIED*** in the **arrUserAccountList** ARRAY inside the **objUserAccountList** object.

Homework Assignment # 4

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

TEST 4 – User Account Management Form EDIT & SEARCH Test (Cont.)



- **TEST 4b** – Test Script:

Test	Steps/Action	EXPECTED Results	Explanation
Test 4b – **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 `Search Screen` ***Enter*** the following **User Account** ***Username*** values when prompted: **Username = nancy**,	▪ 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 **record** of the user is ***DISPLAYED*** by the **SEARCH** & proving that the **CHANGES** took place.

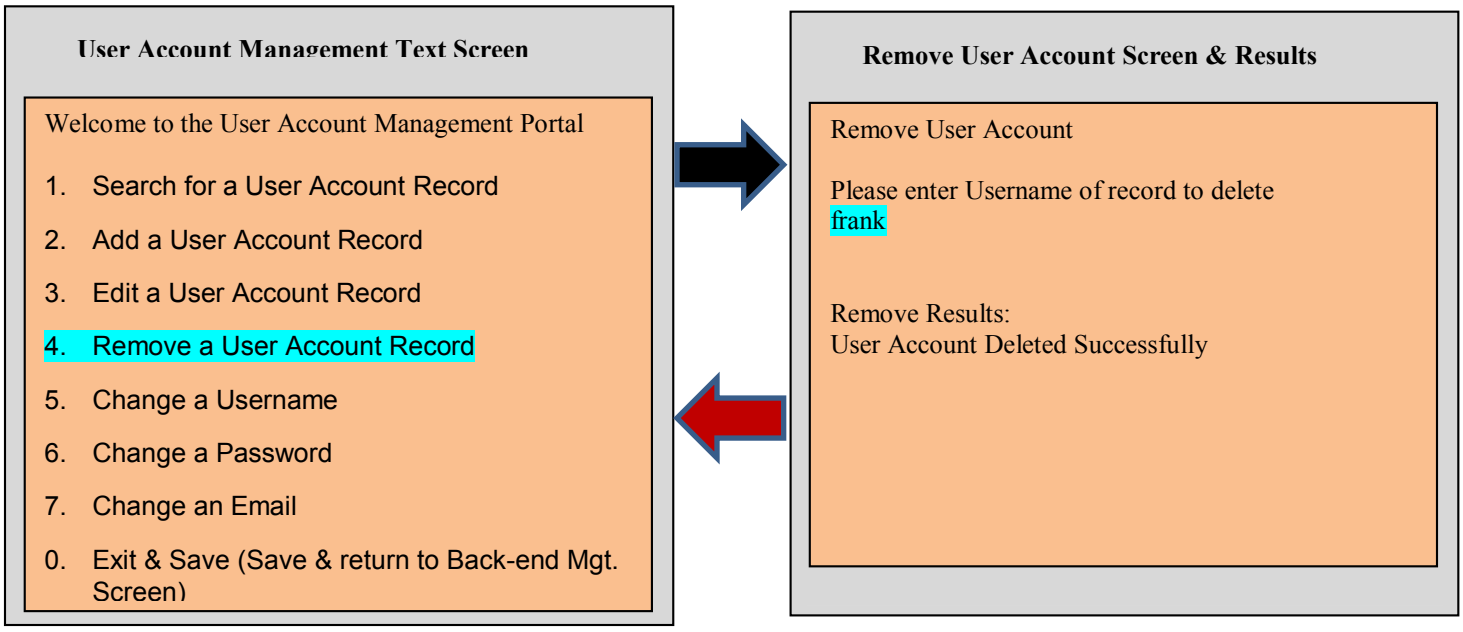
Test #5 – Remove & Search User Account Test

Homework Assignment # 4

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

TEST 5 – User Account Management Form REMOVE & SEARCH Test

- In this test we will test **REMOVE** by ***DELETING*** an ***EXISTING*** **User Account RECORD** then ***VALIDATING*** by **SEARCHING** for the **RECORD *DELETED* to verify it NO LONGER EXIST.**



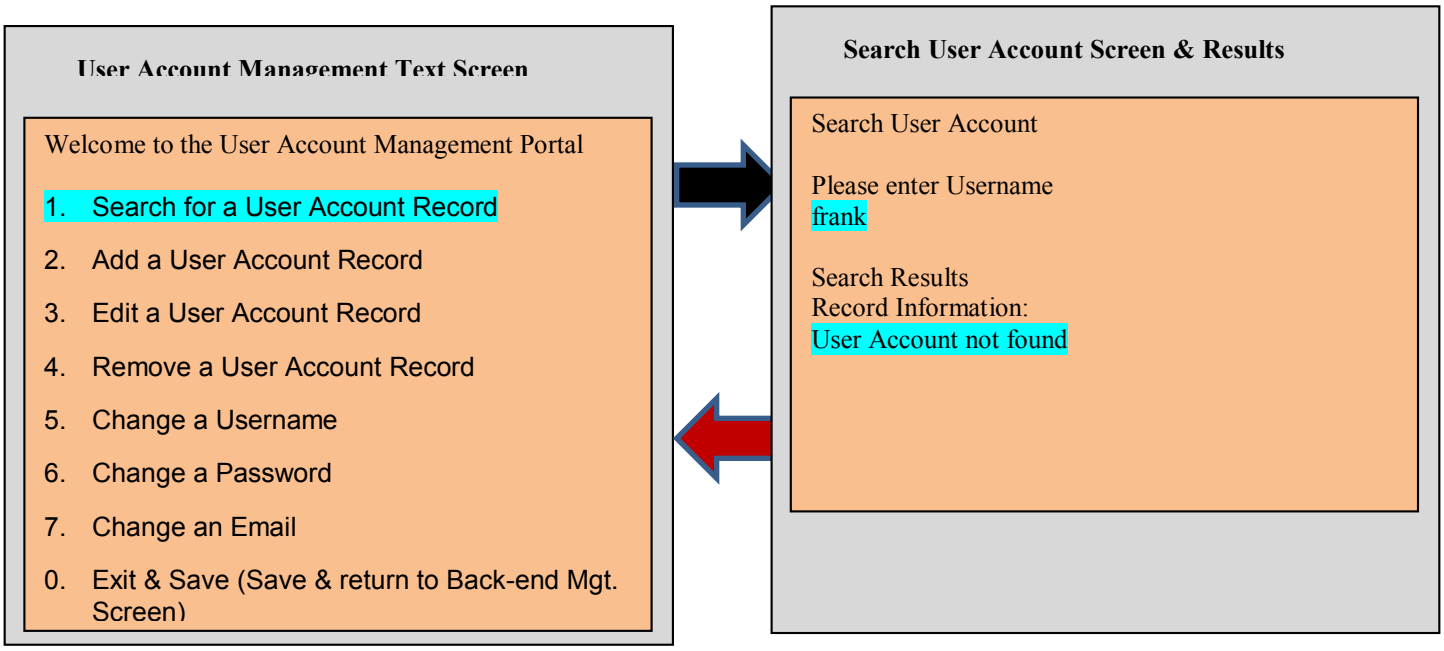
- **TEST 5a** – Test Script:

Test	Steps/Action	EXPECTED Results	Explanation
Test 5a – Testing the **User Account Management REMOVE** feature by ***REMOVING*** an **EXISTING** record & ***SEARCHING*** to verify it was ***REMOVED***.	**Step 1:** From the **User Account Management Screen *SELECT*** option #4 **Remvoe User Account Record**	▪ The `Remove User Account I/O & Results Screen` **is displayed**.	▪ Screen is displayed.
	Step 2: In the `Remove Screen` ***Enter*** the following values as prompted: **Username = frank**,	▪ **Message is displayed stating that User Account Record DELETED successfully**.	▪ **User Account record** for **frank** is ***DELETED*** from the **arrUserAccountList** ARRAY inside the **objUserAccountList** object.

Homework Assignment # 4

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

TEST 5 – User Account Management Form REMOVE & SEARCH Test (Cont.)



- **TEST 5b** – Test Script:

Test	Steps/Action	EXPECTED Results	Explanation
Test 5b – **VALIDATING RECORD** was ***DELETED*** by performing a **SEARCH** for the **RECORD** & ***DISPLAYING*** message confirming **DELETION**.	**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 `Search Screen` ***Enter*** the following **User Account** ***Username*** values when prompted: **Username = frank**,	▪ A message indicating that the **User Account RECORD whose Username = frank WAS NOT FOUND THUS DELETED**	▪ **Username** is needed in order to find the User Account. ▪ The **record** of the user is ***DISPLAYED*** by the **SEARCH** & proving that the **CHANGES** took place.

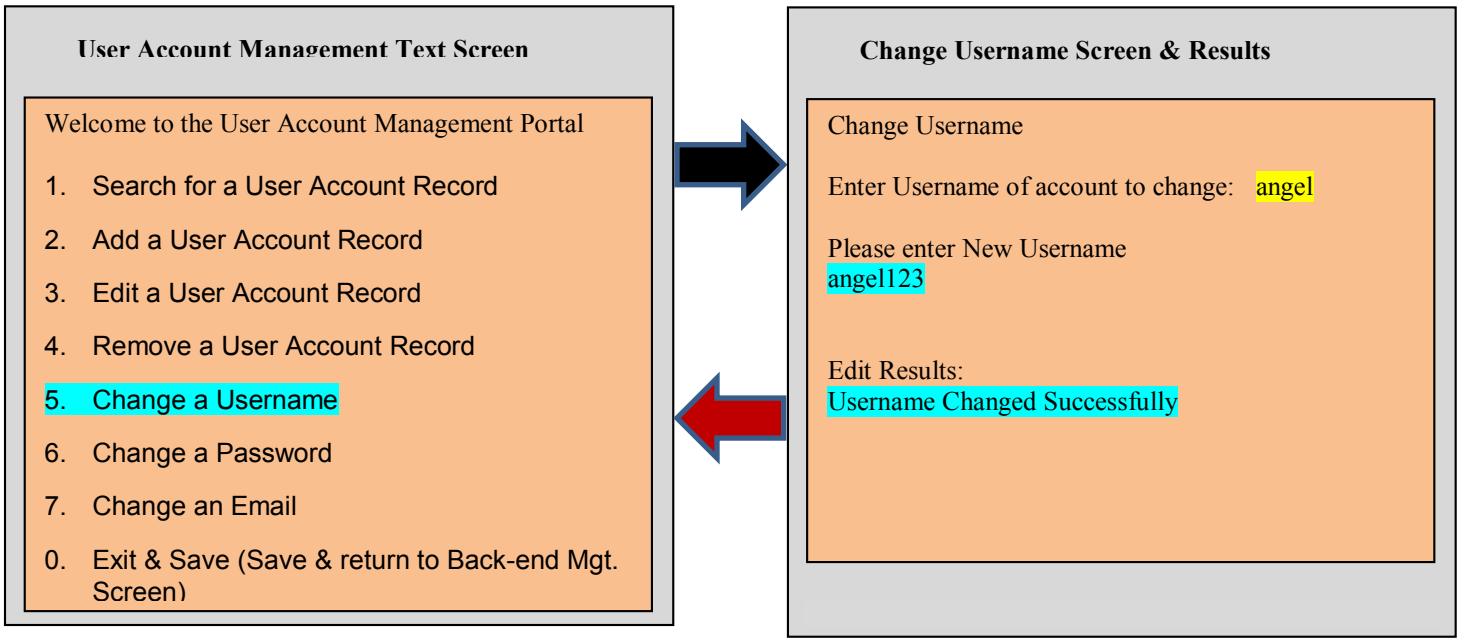
Test #6 – Change Username & Search User Account Test

Homework Assignment # 4

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

TEST 6 – User Account Management Form Change Username & SEARCH Test

- In this test we will test the **CHANGE USERNAME** feature by ***CHANGING THE USERNAME*** of an ***EXISTING* User Account RECORD** then ***VALIDATING*** by **SEARCHING** for the **RECORD *MODIFIED* to verify the Useraccount was *MODIFIED*.**



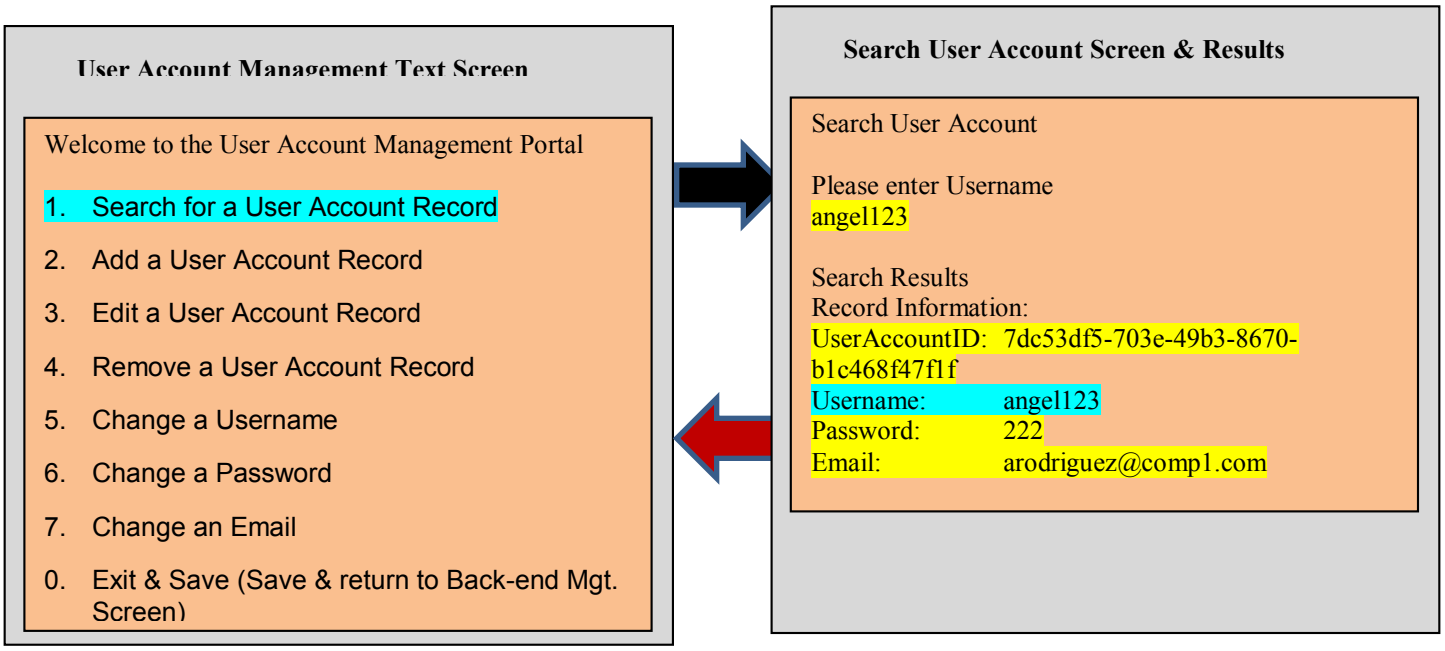
- **TEST 6a** – Test Script:

Test	Steps/Action	EXPECTED Results	Explanation
Test 6a – Testing the **User Account Management CHANGE USERNAME** feature by ***CHANGING*** an existing **Username** & ***SEARCHING*** to verify it was ***CHANGED***.	**Step 1:** From the **User Account Management Screen *SELECT*** option #5 **Change a Username**	▪ The **`Change Username User Account I/O & Results Screen`** **is displayed**.	▪ Screen is displayed.
	Step 2: In the **`Change Username Screen`** ***Enter*** the following values as prompted: **Username = angel** **New Username = angel123**	▪ **Message is displayed stating that User Account Record Username was CHANGED successfully**.	▪ **User Account record** for **username angel** was ***CHANGED*** to **angel123** in the **arrUserAccountList** ARRAY inside the **objUserAccountList** object.

Homework Assignment # 4

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

TEST 6 – User Account Management Form Change Username & SEARCH Test (Cont.)



- **TEST 6b** – Test Script:

Test	Steps/Action	EXPECTED Results	Explanation
Test 6b – **VALIDATING RECORD USERNAME** was ***MODIFIED*** by performing a **SEARCH** for the **RECORD** & ***DISPLAYING*** to confirm **USERNAME** ***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 **`Search Screen`** ***Enter*** the following **User Account** ***Username*** values when prompted: **Username = angel**	▪ A message indicating that the **User Account RECORD whose Username = angel WAS NOT FOUND!**	▪ **Username** is needed in order to find the User Account. ▪ Record **WAS NOT FOUND** because its username was **CHANGED**.
	Step 2: In the **`Search Screen`** ***Enter*** the following **User Account** ***Username*** values when prompted: **Username = angel123**	▪ The record of the **User Account RECORD whose Username = angel123** is **DISPLAYING REFLECTING THE CHANGE OF USERNAME**	▪ **Username** is needed in order to find the User Account. ▪ The **record** of the user is ***DISPLAYED*** by the **SEARCH** & proving that the **USERNAME CHANGE** took place.

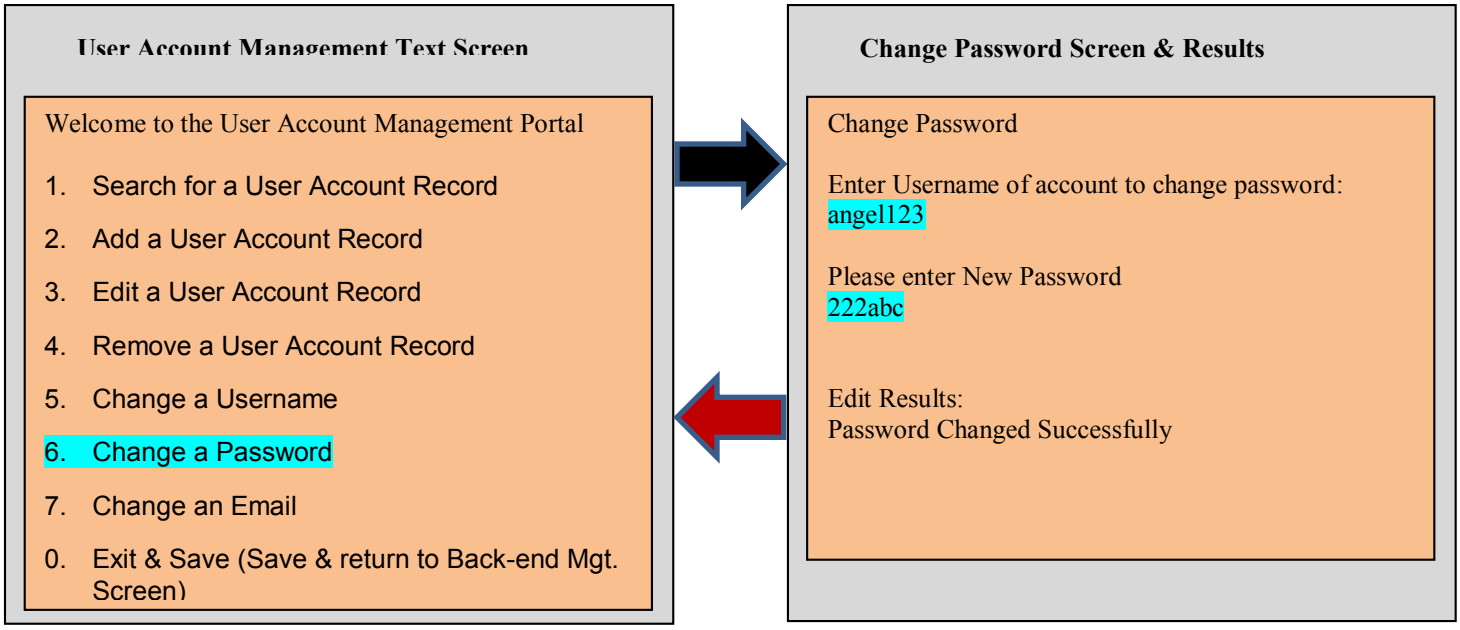
Test #7 – Change Password & Search User Account Test

Homework Assignment # 4

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

TEST 7 – User Account Management Form Change Password & SEARCH Test

- In this test we will test the **CHANGE PASSWORD** feature by ***CHANGING THE PASSWORD*** of an ***EXISTING* User Account RECORD** then ***VALIDATING*** by **SEARCHING** for the **RECORD *MODIFIED* to verify the Password was *MODIFIED*.**



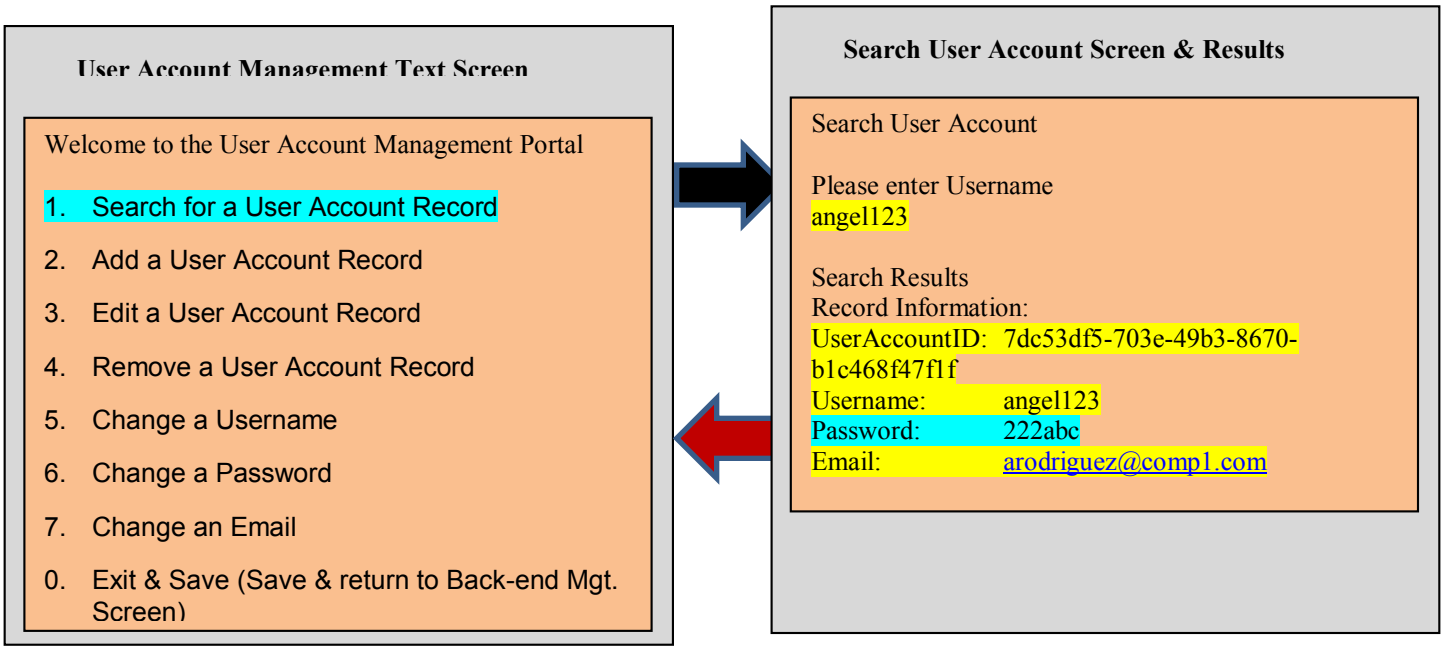
- **TEST 7a** – Test Script:

Test	Steps/Action	EXPECTED Results	Explanation
Test 7a – Testing the **User Account Management CHANGE PASSWORD** feature by ***CHANGING*** an existing **PASSWORD** & ***SEARCHING*** to verify it was ***CHANGED***.	**Step 1:** From the **User Account Management Screen *SELECT*** option #6 **Change a Password**	▪ The `Change Password User Account I/O & Results Screen` **is displayed**.	▪ Screen is displayed.
	Step 2: In the `Change Password Screen` ***Enter*** the following values as prompted: **Username = angel123** **New Password = 222abc**	▪ **Message is displayed stating that User Account Record Password was CHANGED successfully**.	▪ **User Account record** for **password angel123** was ***CHANGED*** to **222abc** in the **arrUserAccountList** ARRAY inside the **objUserAccountList** object.

Homework Assignment # 4

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

TEST 7 – User Account Management Form Change Password & SEARCH Test (Cont.)



- **TEST 7b** – Test Script:

Test	Steps/Action	EXPECTED Results	Explanation
Test 7b – **VALIDATING RECORD PASSWORD** was ***MODIFIED*** by performing a **SEARCH** for the **RECORD** & ***DISPLAYING*** to confirm **PASSWORD *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 **`Search Screen`** ***Enter*** the following **User Account *Username*** values when prompted: **Username = angel123**	▪ The record of the **User Account RECORD whose Username = angel123** is **DISPLAYING REFLECTING THE CHANGE OF PASSWORD**	▪ **Username** is needed in order to find the User Account. ▪ The **record** of the user is ***DISPLAYED*** by the **SEARCH** & proving that the **PASSWORD CHANGE** took place.

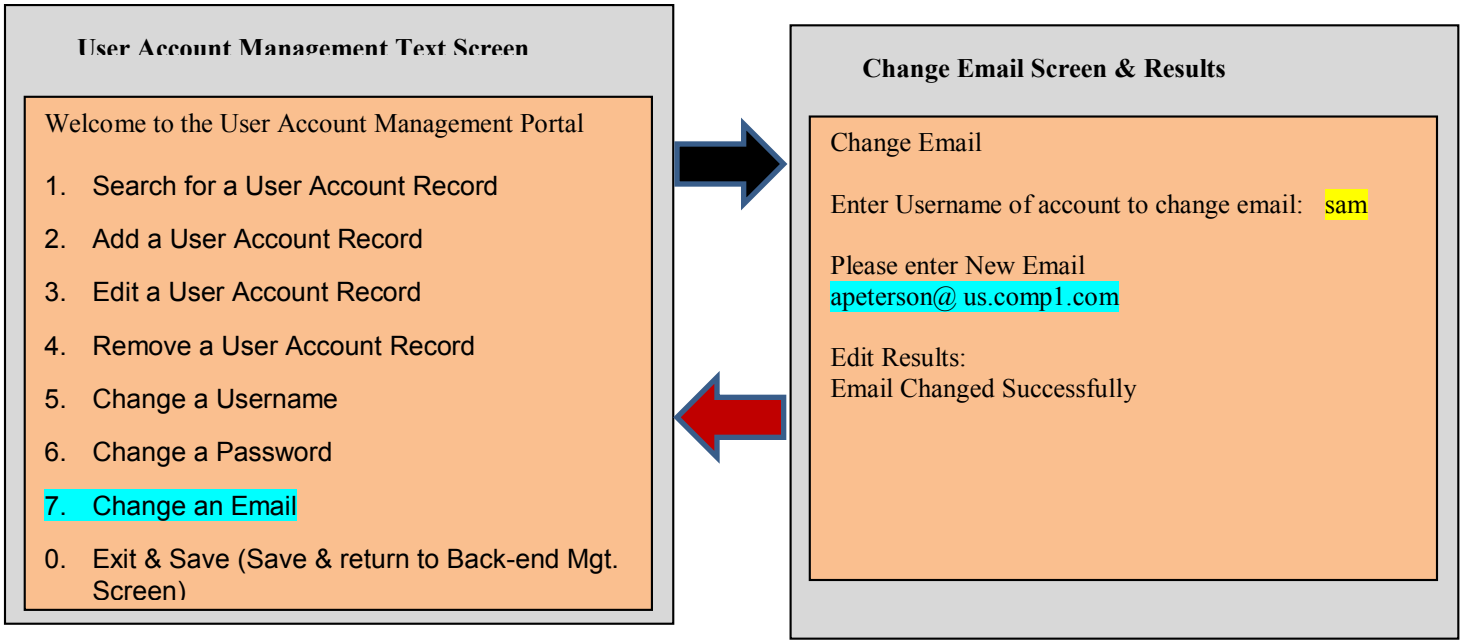
Test #8 – Change Email & Search User Account Test

Homework Assignment # 4

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

TEST 8 – User Account Management Form Change Email & SEARCH Test

- In this test we will test the **CHANGE EMAIL** feature by ***CHANGING THE EMAIL*** of an ***EXISTING*** **User Account RECORD** then ***VALIDATING*** by **SEARCHING** for the **RECORD *MODIFIED* to verify the Email was *MODIFIED*.**



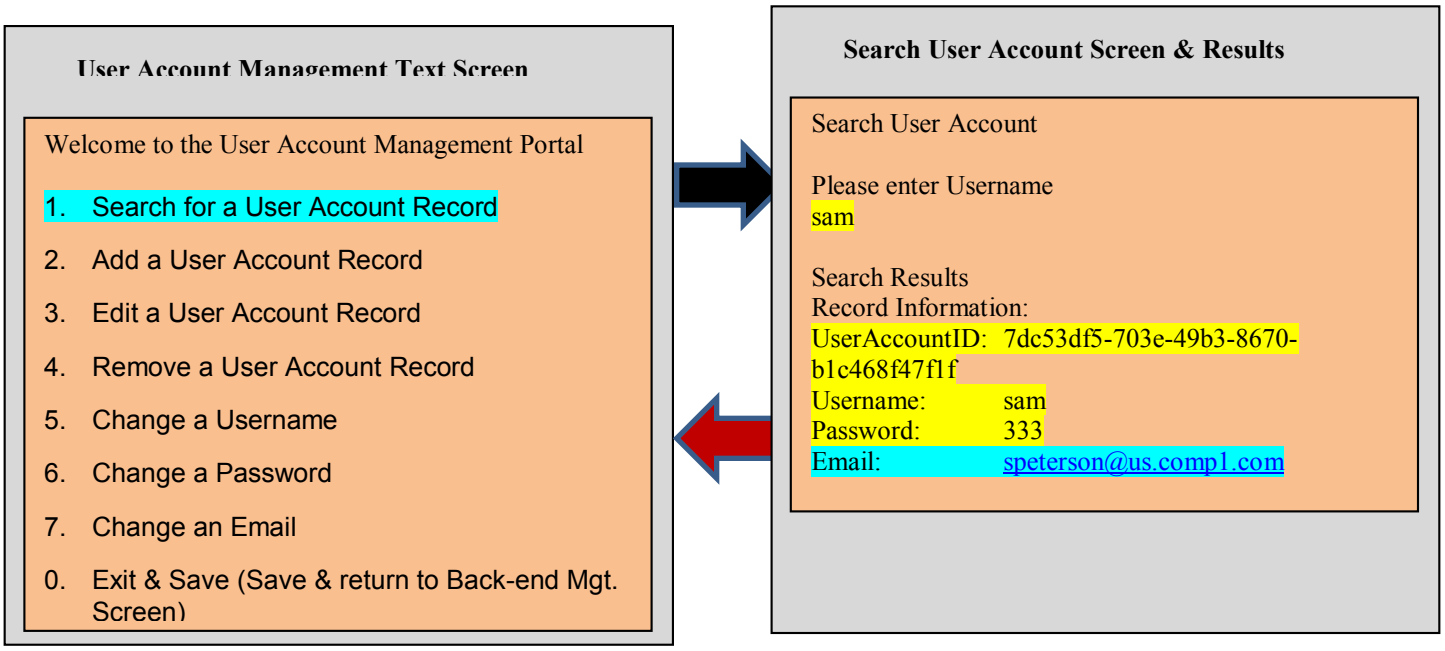
- **TEST 8a** – Test Script:

Test	Steps/Action	EXPECTED Results	Explanation
Test 8a – Testing the **User Account Management CHANGE EMAIL** feature by ***CHANGING*** an existing **EMAIL** & ***SEARCHING*** to verify it was ***CHANGED***.	**Step 1:** From the **User Account Management Screen *SELECT*** option #7 **Change a Email**	▪ The `Change Email User Account I/O & Results Screen` **is displayed**.	▪ Screen is displayed.
	Step 2: In the `Change Email Screen` ***Enter*** the following values as prompted: **Username = sam** **New Email = apeterson@ us.comp1.com**	▪ **Message is displayed stating that User Account Record Email was CHANGED successfully**.	▪ **User Account record** for **email apeterson@comp1.com** was ***CHANGED*** to **apeterson@us.comp1.com** in the **arrUserAccountList** ARRAY inside the **objUserAccountList** object.

Homework Assignment # 4

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

TEST 8 – User Account Management Form Change Email & SEARCH Test (Cont.)



- **TEST 8b** – Test Script:

Test	Steps/Action	EXPECTED Results	Explanation
Test 8b – **VALIDATING RECORD EMAIL** was ***MODIFIED*** by performing a **SEARCH** for the **RECORD** & ***DISPLAYING*** to confirm **EMAIL** ***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 **`Search Screen`** ***Enter*** the following **User Account** ***Username*** values when prompted: **Username = sam**	▪ The record of the **User Account RECORD whose Username = sam** is **DISPLAYING REFLECTING THE CHANGE OF EMAIL**	▪ **Username** is needed in order to find the User Account. ▪ The **record** of the user is ***DISPLAYED*** by the **SEARCH** & proving that the **EMAIL CHANGE** took place.

Expected Deliverables

Homework Assignment # 4

Expected Deliverables

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

Working Application (DUE on Monday May 4)

- **Working Project– Implementation of WORKING Application (No partial credit!!! YOU CAN ONLY SUBMIT IF IT WORKS & BASED ON REQUIREMENTS!!)**
- Important:
  - **Only 1) WORKING TESTED PROGRAM, 2) IMPLEMENTED BASED ON REQUIREMENTS, 3) THAT PASSES THE TEST SCRIPT, 4) SUBMITTED ON DUE DATE, WILL QUALIFY FOR AN "A" GRADE. NO PARTIAL CREDIT!!!!**
  - **IF YOU CANNOT MEET ALL REQUIREMENTS or project is NOT working (DOES NOT MEET ALL 9 REQUIREMENTS OR PASS THE TEST SCRIPT) by due date, DO NOT SUBMIT with ERRORS otherwise is an F!**
  - **Try to get help and submit the next class sessions for a lower grade (B+ (next class after due date) B (Week after due date) C (there after).**
  - If you are having difficulty than get help or contact the professor for help. **No group effort accepted, everyone should work on their own project. Nevertheless, professional courtesy is allowed**. You can ask for advice from other students or suggestion if you are stuck, but no code copying. **Students should not copy each other's code (both students will get an Grade = F if I find you copied code or other deliverables)**

How to Submit Deliverables

- Logistics for sending via email:

Working Application

1) ZIP using **WinZip or Windows compression tool** (**NOT RAR**)., the ENTIRE APPLICATION FOLDER

2) Mail zip file as follows:

   - Mail to **arod1212@outlook.com**.
   - Subject line should indicate **CS3613 – Your Name– HW#**

3) IF YOU NEED HELP or have a question and YOU NEED TO REACH ME DIRECTLY send email to :

   - Mail to **arod@microsoft.com**.
   - Subject line should indicate **CST3613 – Your Name– HW# - HELP!**
   - **DON'T SEND A PROJECT WITH YOUR REQUEST, SIMPLY THE QUESTION.**
   - **YOU CAN ALSO SEND COPY/PASTE CODE IN THE BODY OF EMAIL WITH DESCRIPTION OF YOUR ISSUE.**
   - **IF YOU NEED ME TO SEE YOUR PROJECT AS PART OF THE HELP CONVERSATION, SEND PROJECT TO arod1212@outlook.com AND I WILL REPLY FROM THAT EMAIL, BUT MAKE SURE YOU SEND ME A NOTICE TO arod@microsoft.com. LETTING ME KNOW FIRST.**