**Terms to review:**

identifier

variable

constant

declaration

initialization

assignment

argument

data type

**function**

**method**

**invoke / call**

**calling method**

**called method**

**method declaration**

**method body**

**return type**

**parameter**

**parameter list**

**argument**

**local variable**

**nested method call**

**return statement**

overloaded method

value type

reference type

**class**

**object/instance**

**instantiate**

**constructor**

**field (aka. member variable or instance variable)**

**property**

**access modifier**

**public**

**private**

**encapsulation**

**information hiding**

**implementation hiding**

**composition**

**aggregation**

**Homework & Labs**

*// Please name your projects LB1, LB2, LB3, etc*

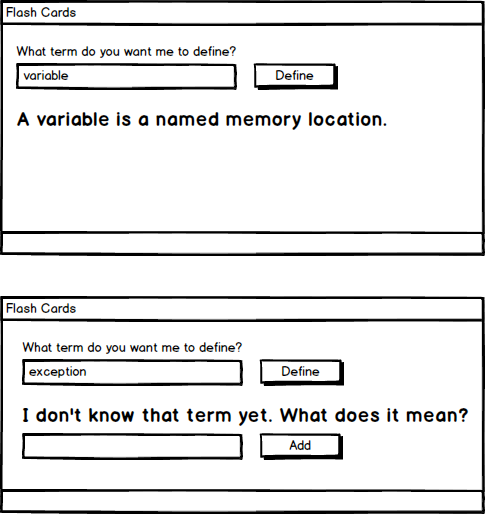
LBI. Complete Naming Conventions Handout

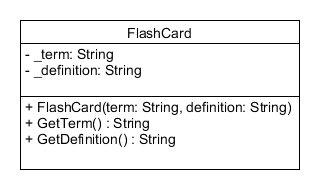
LBII. Complete Data Types Handout

**LB1 FlashCards (Group)**

Create a GUI application to help students remember important terms.

* The program can store definitions for up 20 terms.
* The program comes with 5 definitions included.
* Users can enter a term to get a definition for that term. (case insensitive search)
* If the user enters a term that it doesn't have a definition for, then it will display "I don't know that term yet. What does it mean?" and prompt to enter the definition.

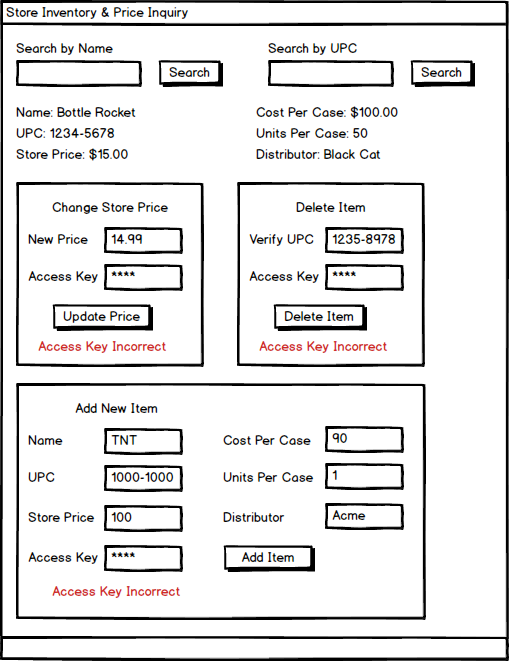




**LB2 StoreInventory (Group)**

Create a GUI application for a warehouse.

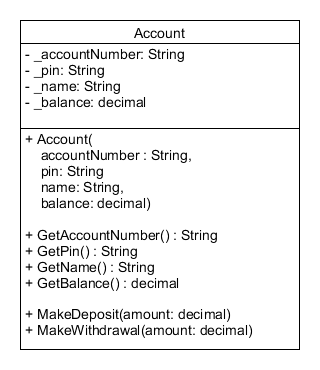
* The program has a pre-populated list of items
* Each item must have a different UPC code
* The program can track up to 20 UPC codes
* The user can search for items
* The user can add new items
* The user can update the price of items
* The user can delete items

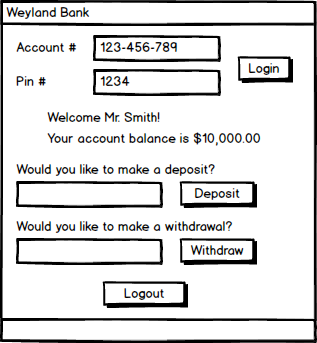
****

**LB3 BankLogin**

Create a prototype for an online bank.

* The program requires an account number and pin number to login.
* Once logged in, the user can deposit or withdraw money.
* When the user is done, they can log out of the system.
* There can only be one user logged in at a time.
* The program should have at least 5 accounts.

**Screen Mockups**



**Methods**

**Login()** accepts an account number and pin number as strings. If valid, logs the user in, displays a welcome message, and displays their balance. If invalid, logs out the previous user and displays an error message.

**Logout()** logs out the active user.

**MakeDeposit()** accepts a deposit amount as a decimal and displays the user's updated balance.

**MakeWithdrawal()** accepts a withdrawal amount as a decimal and displays the user's updated balance.

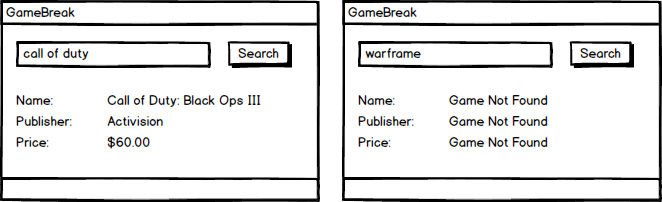
**LB4 GameBreak**

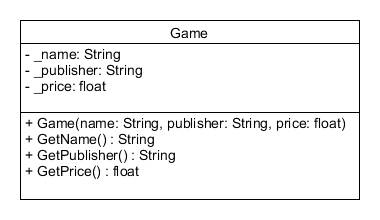
Create a searchable database for a video game store.

* The user can search by either the name of the game or the publisher.
* If a game is found, then the program will display information about the game.
* If a game is not found, then it will display an error message.

| **Name** | **Publisher** | **Price** |
| --- | --- | --- |
| Player Unknown's Battlegrounds (PUBG) | Bluehole | $35 |
| League of Legends | Riot Games | $0 |
| Call of Duty: Black Ops III | Activision | $60 |
| Battlefield 4 | Electronic Arts (EA) | $20 |
| Super Mario Odyssey | Nintendo | $60 |

**Screen Mockups**



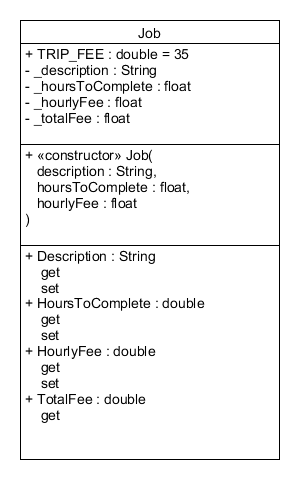
**Methods**

**SearchForGame()** searches for a game by name or publisher, returns the index of the game or -1 if not found.

**ShowGameInfo()** takes the index of a game. If the index is valid, displays the game's information. If the index is invalid, displays error messages.

**LB5 JobEstimate**

Create a GUI application for Harold's Home Services that estimates the cost of various jobs.



The base cost of a job is hours times the hourly rate. Also for each job there is an additional trip fee of $35.

Create a Job class according to the UML diagram below.

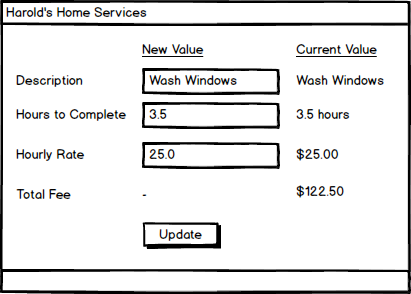
1. The constructor should calculate the total fee as above.
2. The setter for **HoursToComplete** should calculate the total fee as above.
3. The setter for **HourlyFee** should calculate the total fee as above.

Each time the user presses the update button:

1. If a **Job** object has not been created then create one.
2. If a **Job** object has been created then update it's properties.
3. Displays the current values of the **Job's** propertiesbelow as shown

*Program must pass all provided unit tests.*

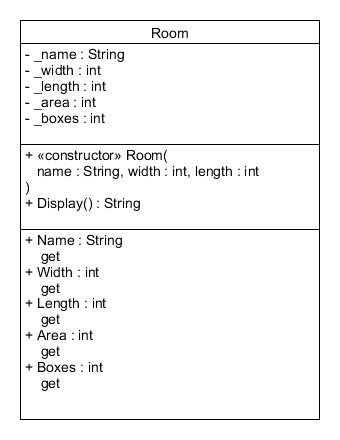
*Please remember to set the tab order.*



**LB6 TileEstimate**

Write a GUI application that calculates how much tile is needed to tile a room. A room requires one box of tile for every 12 full square feet, plus a box for any partial square footage, plus one extra box for waste from irregular cuts.

Create a **Room** class according to the UML diagram below.

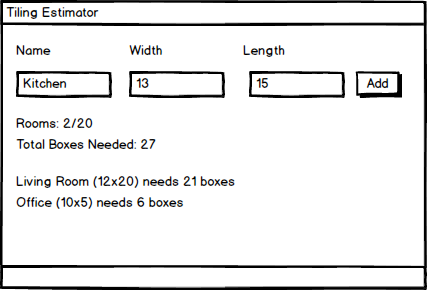
1. Has a field named **\_name** of type **string**
2. Has a field named **\_width** of type **int**
3. Has a field named **\_length** of type **int**
4. Has a field named **\_area** of type **int**
5. Has a field named **\_boxes** of type **int**
6. Has a single constructor that takes a **name, width,** and **length**
7. Has a method that named **Display()** that returns a string in the format **"{name} ({width}x{length}) needs {boxes} boxes"**
8. Has a readonly property named **Name**
9. Has a readonly property named **Width**
10. Has a readonly property named **Length**
11. Has a readonly property named **Area**
12. Has a readonly property named **Boxes**

The user can enter up to 20 rooms.

Each time the user presses the add button:

1. A new **Room** object should be created
2. The **Room** should be added to an array of **Room** objects
3. Displays the **Room** in the output
4. Clears out the text fields and sets the focus to the name field

*Program must pass all provided unit tests. Please remember to set the tab order.*



**LB7 Build-A-Lab**

Create your own lab.

* Submit your design to the bin in class.
* Submit your implementation to GitHub.
* Program must have tab order configured.
* Program must follow naming conventions for all controls, variables, constants, methods, classes, and enumerations.
* Program must have at least two classes.
* Program must demonstrate either composition or aggregation.