**Phillip Hoang 1409768**

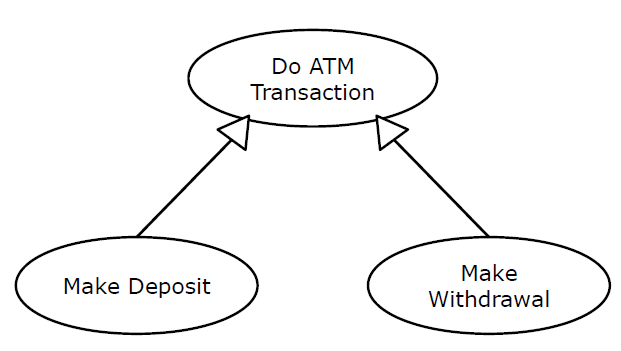
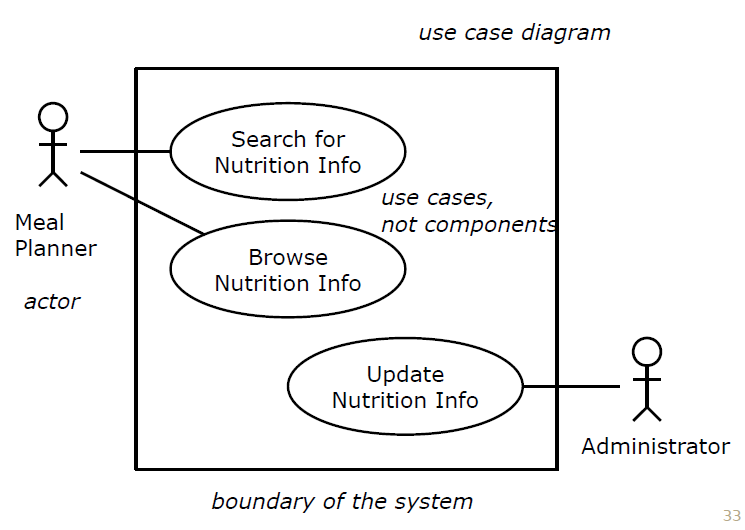
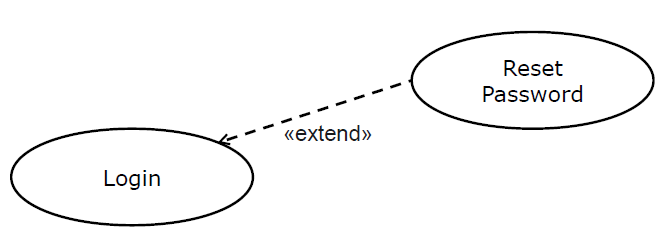
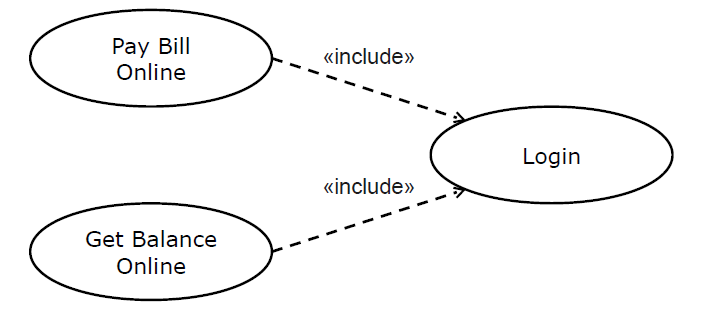
-Listing potential classes, actions/methods/relationships (verbs). And **DRAW** the **UML class diagram** based on them.  
-Converting **Java code to UML class diagrams**

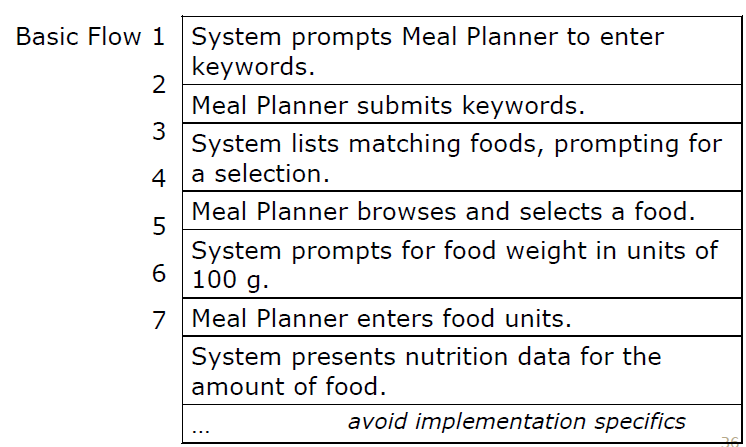
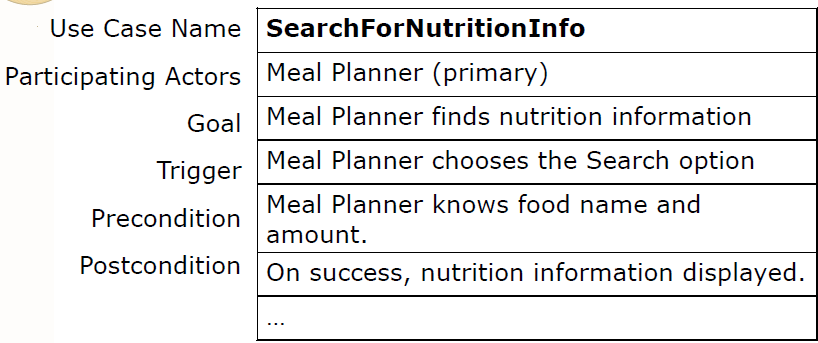
****

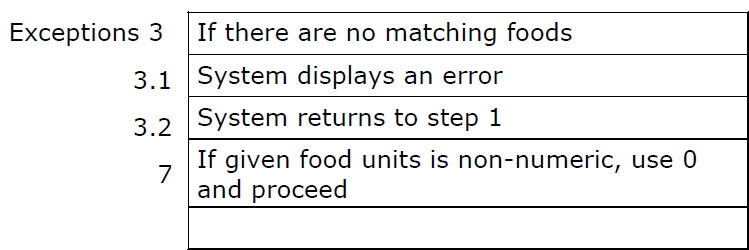
+ is public, - is private, triangle is ISA, dashed lines are for implementation of interface, diamonds for HASA, black diamonds for STRONG HASA

0..\* is 0 or more

**LABEL relationships**

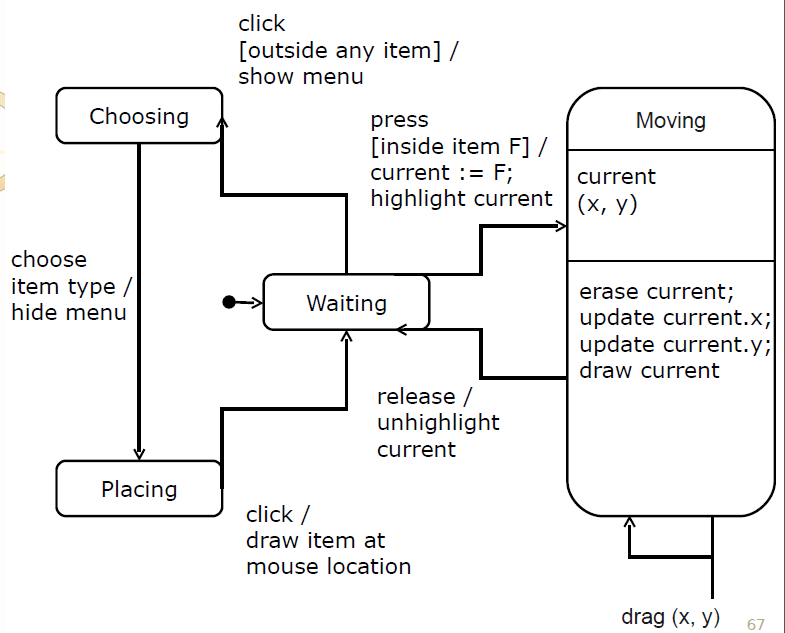
-**Deriving Use Cases** from goals/description, make **UML Use Case diagram** from this including boundary, actors, use case bubbles, and relationships between actor and use case   
  






-Converting Use Case to **Sequence Diagram (UML Sequence Diagram?)**

-Identify Design Patterns based on picture (draw design the design patterns)

-Create a **UML State Diagram  
**

**-**Convert scenario to **use case (cover common exceptions)**

-Git  
 \*-How to track staged delivery process where clients might be using older (stabler) versions

-In **daily scrum meeting** why would you use **git log**? *To show commit history to ...*

-Describe requirements, design, and testing stages of the **waterfall model**, describe how you would use Git to enable and track work of these 3 stages:

-Using git repos, how would you enable and track an iterative software dev process?

-Human Error and Usability   
 -Name of the law that describes the **speed of choosing from a list of choices**

-Name of the law that defines **speed of clicking on a target**

-Which target is the **fastest to click**?

-Why does it take longer to click on the other targets?

-Why is the **difference in time of choosing 2 and 8** choices greater than the difference between 80 and 100

-What is **Saccadic Masking** and how does it affect software, affect use scrolling through a large and long webpage, how to design against it

-Use **Fitt's law** and **Hick's law** to explain which array of buttons would be faster (ex list vs pie)

-Why is there a difference in time between choosing 1 item from 80 unordered, or 1 item from 80 ordered. **How much** is the approximate difference in time?

-[toggle between edit and delete] what kind of common error will the UI cause the user (name of error)? How to fix it?

-What is a mode error? How does one prevent mode errors?

-User Interfaces  
 -What is one UI method that aids usability but also reduces human error?

-Why must we be careful about **colours** we use in UI? *Colour blindness* -How to design light switches that aren't red and green

-Give 2 examples (or instances) of interface metaphors.

-Name of the law that estimates average time to make a simple decision, *n* choices vs *t* time

-GOOD UI Design

-Software Process  
 -Explain what a software development process is

-Relationship between **iterative model** and **waterfall model**. What is their primary difference? Why would you choose waterfall over iterative? Give 1 example where you would use waterfall over iterative, and 1 example of iterative over waterfall

-Provide an example of 2 different software development processes and how they differ from eachother

-Give 1 reason why the **Unified Process** is similar to the **waterfall process,** 1 reason why different: *both have principle-bound stage such as requirements and implementation, unified iterative, waterfall goes sequentially*

*-*In what parts of waterfall model would you use refactoring?

-In SCRUM, what is a daily standup meeting and what are the questions asked? *What did you do, what are you going to do, what is blocking you*

-How does version control like git relate to the notion of **courage** in agile software development? *Branching allows programmers to try new things with no fear of messing up the current product.*

-How is **Test Driven Development** employed in the design of APIs?

-How does **test first development** work? How does it affect software design?

-How do agile processes provide feedback to stakeholders?

-How would one use continuous integration in staged delivery process?

-List 2 tools that promote courage (agile). Explain why

-Identify Design Patterns appropriate for (and explain why):  
 -Want to implement macros learned from the user. These macros can be stored and replayed later.

-Event-based system where users can add plugins at run-time. These plugins can agree to handle some events but might only do so conditionally

-Making a program that procedurally details a universe lazily. Can go down from galaxies to solar systems to planets to countries to people to cells to atoms etc

-Have an algorithm for recognizing different kinds of minerals from photos. The algorithm needs specialized logic for each different mineral, but general control flow and logic can be shared

-Building gravity sim for planets. 3D view is hard to control and configure so want a 2D view and textview that shows planet state. Also want to be able to add or delete bodies in sim as it is running.

-Making mass photo editor where operations can be repeated across entire director of photos.

-Making city sim. People change over time but their identity does not. How do you model difference in behavior between a child and an old man who were the same person

-Making web interface to eclipse. Want to send requests to eclipse to open, view, modify, etc your projects via a web browser and have it executed in your eclipse IDE. Eclipse will run a webserver to do this.

-Making an OS abstraction layer to port apps between different platforms. You have defined interfaces, but the client code needs to get concrete instances of them. How to build appropriate concrete instances for the clients?

-Making a mind-map to model web knowledge where users can make entries that can be related to 0 or more other entries

-Making programmable text editor in the cloud that can be controlled via webpage or API. You have some atomic operations but want to allow automation of these operations by scripts and services. You want to compose operations together.

-Making enemy characters for a vidya. They act differently depending on if they see you, how recently they saw you, or if they unaware of you.

-Have image gallery program with very large images. They take long to load, so you generate thumbnails for them until they are needed or loaded. You might not load all of the images

-You are writing files to disk. Some files should be encrypted, compressed or both.

-Making multiplatform user interface library. Devs should be able to ask for widgets and buttons and get appropriate one for their platform without having to know what they get.

-Making a shared canvas paint program where multiple users draw on the same shared canvas. The users can pain, draw, and erase elements all together on the same canvas

-Making a system that can respond to natural language queries such as "I want some horse radish". System provides responses through a series of dynamically loaded plugins that can be loaded and unloaded by the user at any time

-Making an RPG that has an inventory system where boxes, sacks, chests, and bags can hold other containers. Some of the containers have magical properties that imbue the items contained within with properties like fire or lightning

-Writing DB rows to a disk. Depending on the DB, some columns have private info that needs to be encrypted. Some columns contain a lot of text and should be compressed, some columns have to be encrypted and compressed.

-Making a game where you run over enemies and they become a part of you, then when you press fire, all the ones attached to you shoot off in all directions

-Making instant messaging client. Want user to load 3rd party plugins that can respond to certain requests such as "where are you" automatically

-OO Principles  
 -Explain how the **hide delegate** refactoring applied to the **message chains** bad smell increases or decreases **coupling**

**\***-Explain how coding to the **specification** rather than the **implementation** increases or decreases coupling

-Explain how Java's **dot operator** and **switch statement** increase or decrease coupling

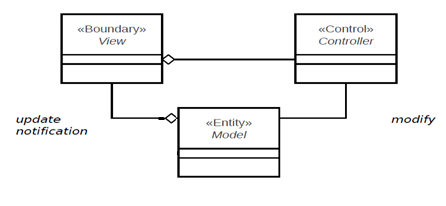
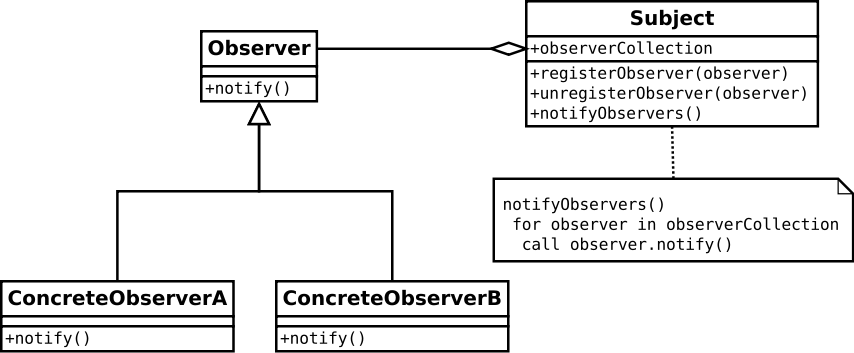
-Explain why using ArrayList as a Stack violates the **Liskov substitution princple** eg class Stack extends ArrayList<Object>: *Arraylist can be substituted anywhere a stack is used, but stack cannot be substituted for arraylists*

-Explain how to **replace conditional with polymorphism** refactoring applied to the **switch statement** bad smell increases or decreases coupling

-How are cohesion and coupling affected by: high dependence on abstractions, low dependence on concrete classes

-How are cohesion and coupling affected by: favor composing objects (delegation) over implementation inheritance

-MVC and Observer Pattern  
 \*-How does observer pattern **decouple** a model from views?

-Draw **UML Class Diagram of MVC, of observer pattern  
**

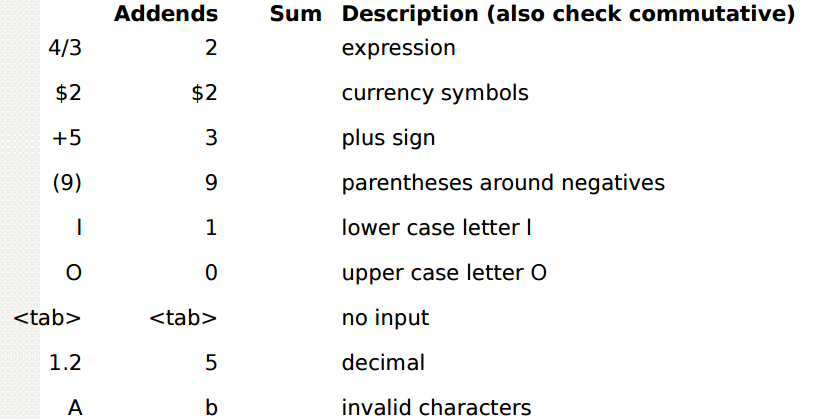
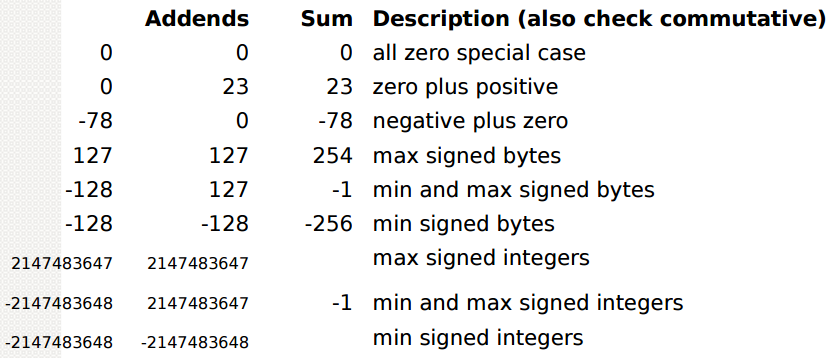
**-**Draw **UML Sequence diagram** for update of model in observer pattern

**--**Model: entity layer, complete, self-contained representation of data (back end)  
 --View: boundary layer, what the user sees (front end)  
 --Controller: control layer, handles events and uses elements from the UI to modify the model

-Refactoring (and Decorator) [and Template Method] {and Factory Method}  
 -Find at least 3 bad smells, and at least 2 refactoring that could be applied to this code snippet. Then **DRAW** the UML class diagram of the code after refactoring

-Provide UML class diagram for xClass after refactoring xMethod() using **Decorator Pattern  
 -**Provide UML class diagram for xClass after refactoring xMethod() using **Template Method Pattern  
 -**Provide UML class diagram for xClass after refactoring xMethod() using **Template and Factory Method Pattern  
 -**Provide UML class diagram for xClass after refactoring xMethod() using **Factory Method Pattern**

-Testing  
 -Write a class for a **mock object** that will allow for testing of line x of xClass

-Provide 5 good test cases for a function. (Max of 1 test per equivalence)  


-Write test-cases using jUnit style unit tests for xMethod()

-Theory

-Waterfall: Requirements Specification Architectural Design Detailed Design Integration and Testing Delivery and Operation Maintenance and Support  
 -verification at every phase, sequential, easily understood, enforces discipline, manufacturing view of software

-Prototyping: iterative design cycling through more than 1 design, improving with each iteration  
 -throwaway: build and test prototype to be thrown away to build the real product  
 -incremental: have "must do", "should do", and "could do"  
 -evolutionary: features are refined over time ex: key for cut undoable cut drag and drop cut & paste  
 -hand drawn sketches, storyboards,

-XP: 40 hour work week, metaphor, simple design, collective ownership, coding standards, small releases, continuous integration, refactoring, planning game, testing, on-site customer, pair programming

-Decomposition: dividing whole things into parts

-Dynamic Binding: selection of method to be run is made at run time depending on input parameters

-Requirements  
 -User requirements: what tasks the user can do with the system  
 -Functional requirements (features): what behaviors does system support  
 -Non-functional requirements (qualities): how well the system should do what it does (fast? mem usage)