Week 9 Seminar & Pre-class activities

Q1. Coupling & Cohesion

Question A *Submitted May 7th 2022 at 11:43:05 am*

Discuss 2 reasons as to why low Coupling and high Cohesion result in systems that are easier to reuse and maintain.

Value: 20%

Because the low-coupling classes can work independently, are quite loose and easy for the system to understand, with only a few ripple effects, without changing all the other classes at maintenance time.

Hight Cohesion reduced the dependency of objects, causing little ripple effect and increasing the ability to reuse parts of the system for easier maintenance.

Q2. Design Class Diagrams

Question A *Submitted May 7th 2022 at 12:16:22 pm*

Describe 3 ways in which a Design Class Diagram differs from a Domain Class Model Diagram?

Value: 30%

The Domain Class Model Diagram only has the Class name and attributes. The Design Class Diagram details the type of the Class and the data type of the attributes.

The Design Class Diagram defines a special symbol for a Class based on four types of stereotypes. The Domain Class Model Diagram has none of this.

The Design Class Diagram is composed of Class name, attributes and method signatures. Domain Class Model Diagram only Class name and attributes.

Q3. Sequence Diagrams

Question A *Submitted May 7th 2022 at 2:33:01 pm*

For the student functionality "Update student information", using the Domain Model Class Diagram and the System Sequence diagram:

- Draw a First-cut Sequence Diagram
- Draw a Final-cut Sequence Diagram

NOTES: See Process Guidelines for how to draw these diagrams

IMPORTANT - Please cut and paste an image of your 2 diagrams in the answer

Value: 50%

Process Guidelines

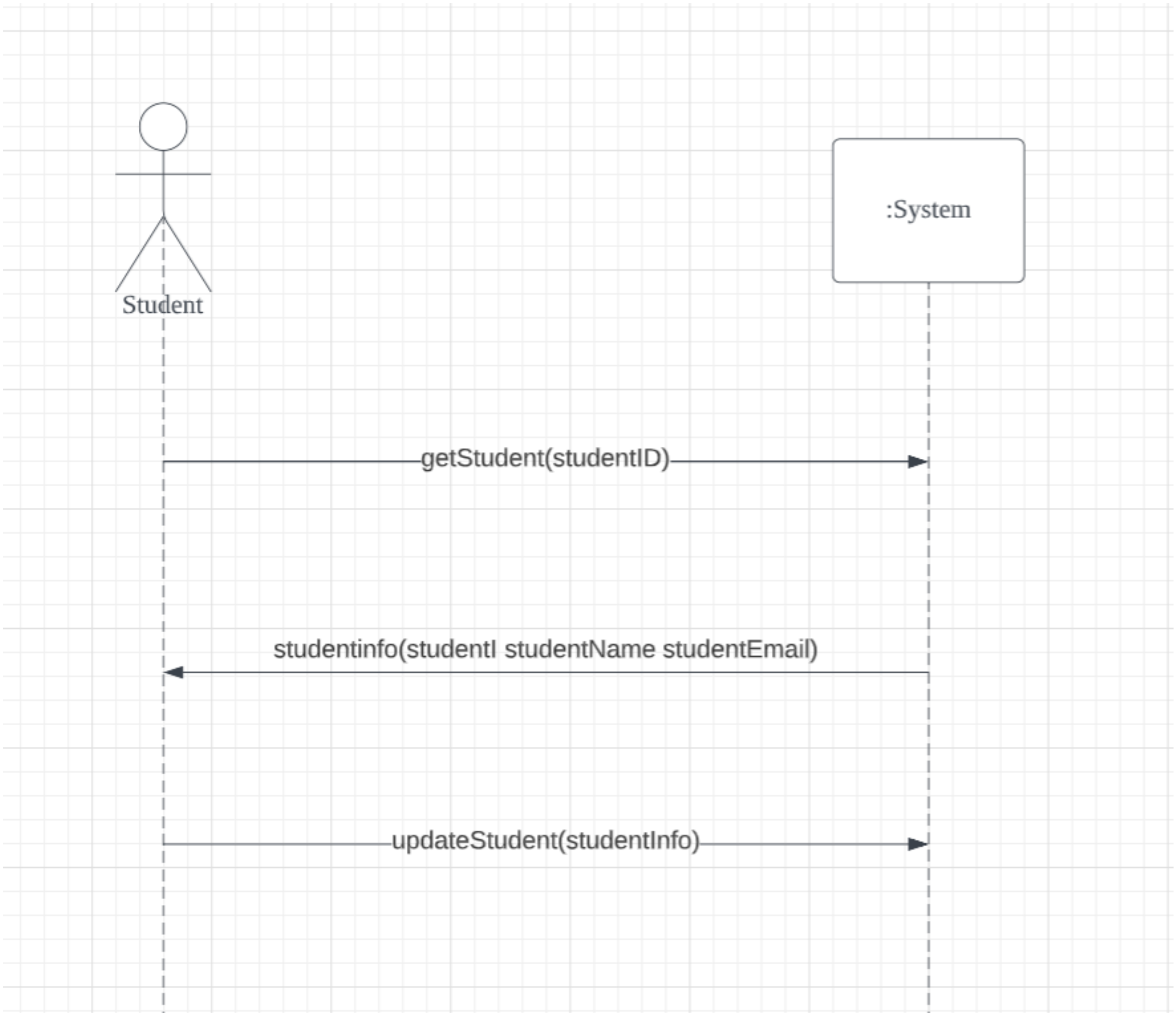
Draw the First-Cut Sequence Diagram

- Start with elements from System Sequence Diagram (SSD) and first-cut Design Class Diagram
  - A Sequence Diagram uses all elements of an SSD .. System object replaced by all internal objects and messages

- Replace the :System object with an appropriately named use case controller
- For each input message
  - Determine all internal messages that result from that input
  - Determine its objective, what information is needed
  - Identify the complete set of classes from the domain model affected by the message
    - What class needs it (destination)
    - What class provides it (source)
    - Whether any objects are created as a result of the input
  - Flesh out the components for each message
    - Iteration, true/false conditions, return values, passed parameters
  - Add the controller - use case controller acts as intermediary between outside world and internal system

Draw the Final-Cut Sequence Diagram

- Add a view layer interface class before the controller either as a single GUI class or as Windows classes
- Add a data access class for each problem domain class
  - Data access layer should only support database CRUD – (Create, Read, Update, Delete) operations so classes maintain a high level of cohesion and are loosely coupled with the business layer


Student
:System
getStudent(studentID)
studentinfo(studentI studentName studentEmail)
updateStudent(studentInfo)