# Assignment – 3

Name: Malika Hafiza Pasha
Student ID: 212238171
Course: CSC 581 (Advanced Software Engineering)

1) **What is the focus of control in a sequence diagram? (10 points)**

**Answer:**

The chronological order and flow of interactions or messages between different objects or components within a system are represented by the **focus of control in a sequence diagram**. This visualization shows the flow of events and messages between objects, showing how control and information are transferred between them to carry out a particular task or procedure. Understanding the timing, dependencies, and sequence in which various actions or method calls inside a system are executed is made easier by the control focus.

**Example:** There are two objects - a bank and a customer:

a. The customer requests a withdrawal on their own.
b. To process the withdrawal, this sends a message to the bank.
c. After confirming the request, the Bank proceeds with the withdrawal.
d. The customer receives confirmation regarding the successful withdrawal.

**In this basic order:**

a. **Control Flow:**

The withdrawal is started by the customer and then transferred to the bank for processing.

b. **Message Passing:**

To request and confirm the withdrawal, messages are sent back and forth between the customer and the bank.

2) **What is a “return message” in a sequence diagram? How is it shown? Your answer must include the pairing of messages in a sequence diagram. (10 points)**

**Answer:**

In a sequence diagram, a "**return message**" is the response or transfer of control from the sender object to the recipient object following the completion of an action or message processing. It represents the recognition, outcome, or reaction to the first communication transmitted between objects.

Return messages are shown in a sequence diagram as a dashed arrow, signifying that control is

returned to the sender. To indicate what is being returned, they are frequently tagged with the name of the method, a colon, and perhaps a return value or other information.

**Example: Using message pairing**

For instance, a customer asks a bank for account details.

a. The customer messages the bank to ask for account details.
b. After processing the request, the Bank obtains the account information.
c. The requested account information is included in a message that the Bank sends back to the customer.

**Within the flowchart:**

- A solid arrow labeled with the request (e.g., "getAccountInfo()") represents the first message sent by the customer to the bank requesting account information.
- The response back to the sender is indicated by a dashed arrow in the Bank's return message to the customer. The method name (such as "returnAccountInfo()") and the account information or other required data that is being returned could be labeled with it.
- The sequence diagram's message pairings clarify the communication and interaction between objects by illuminating the flow of control between the sender's request and the recipient's subsequent answer.

3) **Consider a messageXYZ() sent by ObjectA to ObjectB. Which of the two objects will have the code to execute the messageXYZ()? Why? (10 points)**

**Answer:**

The code that runs the messageXYZ() method in object-oriented programming usually lives in ObjectB.

a. **Sender and Receiver:**
By sending ObjectA to ObjectB the message XYZ(), ObjectA is presumably calling or invoking a method on ObjectB.

b. **Encapsulation:**
An object should encapsulate its data and behavior, according to the object-oriented programming concept of encapsulation. This implies that a method's associated behavior (such as messageXYZ()) ought to be defined inside the object to which it makes sense.

c. **Method Location:**
Only ObjectB can use the messageXYZ() function. Consequently, it makes sense and adheres to sound design standards for the code that uses this method to be found inside of ObjectB. The functionality and behavior associated with the messageXYZ() method are stored in ObjectB.

**Summary:**
In accordance with the concepts of encapsulation and object-oriented design, ObjectB will therefore include the code necessary to carry out the messageXYZ() function as that is the location of the method's definition and its corresponding logic.

4) **How can sequence diagrams enrich class diagrams? What are the potential problems with doing so? (10 points)**

**Answer:**

Class diagrams can be enhanced with sequence diagrams, which offer a dynamic perspective of the interactions between class instances across time. By depicting the series of messages sent between objects of different classes, they enhance class diagrams by making the behavior and interactions described in the static structure of class diagrams more visually appealing.

**Advantages of enhancing class diagrams with sequence diagrams:**

a. **Behavioral Understanding:**
Sequence diagrams, which supplement the static structure shown in class diagrams, aid in the understanding of the dynamic behavior of the system by illustrating how objects cooperate and communicate during runtime.

b. **Validation and Verification:**
By showing how the methods defined in classes are used and how objects interact in real-time settings, they help validate and verify the design shown in class diagrams.

c. **Communication and Clarity:**
By providing a clear illustration of how various classes work together to accomplish functionalities or processes, sequence diagrams can improve stakeholder communication.

**Disadvantages of adding sequence diagrams to class diagrams:**

a. **Complexity:**
Sequence diagrams can get crowded and complex as systems are bigger or more complex, which makes it difficult to fully depict all interactions. Clarity and readability may be

hampered by this intricacy.

**b. Maintainability:**
When system modifications happen often, it could get difficult to keep sequence diagrams in line with class diagrams. A time-consuming and error-prone process could result from synchronizing changes in both representations.

**c. Abstraction Level:**
Class diagrams give a high-level abstraction, whereas sequence diagrams may go into implementation details. There may occasionally be a lack of consistency or confusion as a result of this mismatch between the two representations.

**d. Scope and Detail:**
It could be difficult to strike a balance between the amount of information provided by sequence diagrams (dynamic behavior) and class diagrams (static structure). Sequence diagrams that with too much detail could overwhelm the observer, while those with too little detail might not adequately depict the interactions.

**Summary:**
Sequence diagrams display dynamic behavior, which complements class diagrams. However, to guarantee that they successfully explain the system's design and behavior without overwhelming or deceiving stakeholders, it is crucial to maintain a balance between the two representations and manage complexity.

5) **When describing a system, explain why you may have to start the design of the system architecture before the requirements specification is complete. (10 points)**

**Answer:**

Prioritizing the design of the system architecture above the completion of the requirements specification is common in complex projects or system development processes for various pragmatic reasons:

**a. Parallel Work Streams:**
Several teams or stakeholders are frequently involved in large-scale projects. One team can begin building the system architecture while the other gathers and refines requirements. This parallel methodology guarantees efficient project progress while saving time.

**b. Iterative Development:**
An emphasis on iterative development is placed by many contemporary development approaches, including Agile. Early system architecture design enables the development of a fundamental structure or framework based on initial requirements. It is possible to gradually

modify the design as requirements change or become more apparent.

**c. Risk Mitigation:**

It's imperative to address important technological risks as early in the project as possible. Potential technological obstacles or restrictions can be identified and mitigated through system architecture design. It aids in determining whether the suggested solution is workable and whether any significant roadblocks need to be removed.

**d. Prototyping and Proof of Concept:**

Prototyping or proof-of-concept models can be made when a preliminary system design is developed. By gathering feedback from stakeholders, showcasing functionality, and validating concepts, these prototypes can help refine requirements throughout the process.

**e. Time Restrictions:**

Holding off on starting development until all criteria are complete might cause delays in fast-paced work settings or projects with strict deadlines. While the requirements are being finalized, the early stages of development can continue while the architecture design is underway.

**f. Adaptability and Flexibility:**

An early architectural design offers a base that is adaptable. An adaptable architecture makes it easier to make adjustments when requirements change or evolve, which lessens the effect of requirement variations on the overall system design.

**g. Client Input and Feedback:**

Working with clients or end users during the architectural design phase may occasionally help to extract more specific or nuanced requirements. It encourages cooperation among stakeholders so they can envision and influence the system's original design.

**Summary:**

There are benefits to beginning system architecture before all requirements are specified, but it's critical to keep the lines of communication open and the architectural choices in line with the requirements as they change. Making sure that the system design successfully fits the changing needs of the project requires flexibility, adaptability, and an iterative process.

6) **A small company has developed a specialized software product that it configures specially for each customer. New customers usually have specific requirements to be incorporated into their system, and they pay for these to be developed and**

**integrated with the product. The software company has an opportunity to bid for a new contract, which would more than double its customer base. The new customer wishes to have some involvement in the con - figuration of the system. Explain why, in these circumstances, it might be a good idea for the company owning the software to make it open source. (10 points)**

**Answer:**

Making software open source could provide multiple strategic benefits for a software company that customizes its product for each customer and has the potential to considerably grow its customer base:

a. **Community Involvement:**

By releasing the software as open source, the business can reach a wider range of programmers and fans who can help to enhance the program. This kind of community-driven cooperation can result in improvements based on a variety of viewpoints, issue solutions, and quicker development.

b. **Quick Customization:**

Open source software makes it easier for users to modify it to meet their own requirements. Customers can engage developers to make changes to the program or customize it themselves if they have access to the source code, which lessens their reliance on the corporation for every customization request.

c. **Increased Adoption:**

More prospective clients that favor customizable solutions may be drawn in by offering an open source version. It can be used as a marketing tactic to draw in clients that respect control and flexibility over their software solutions.

d. **Market Expansion:**

Markets where customization and adaptability are essential tend to favor open source software. Incorporating clients into the configuration process facilitates the software company's entry into new markets and industries where customized solutions are sought for.

e. **Decreased Development Costs:**

The company's development costs can be decreased by utilizing community contributions. For feature development, the open source community can work together to advance the program rather than depending only on internal resources.

**f. Brand Reputation and Trust:**

The software's open source nature shows transparency and faith in the developer's work. It cultivates a favorable perception of the business as one that is open to working together and being honest, which may strengthen ties with clients.

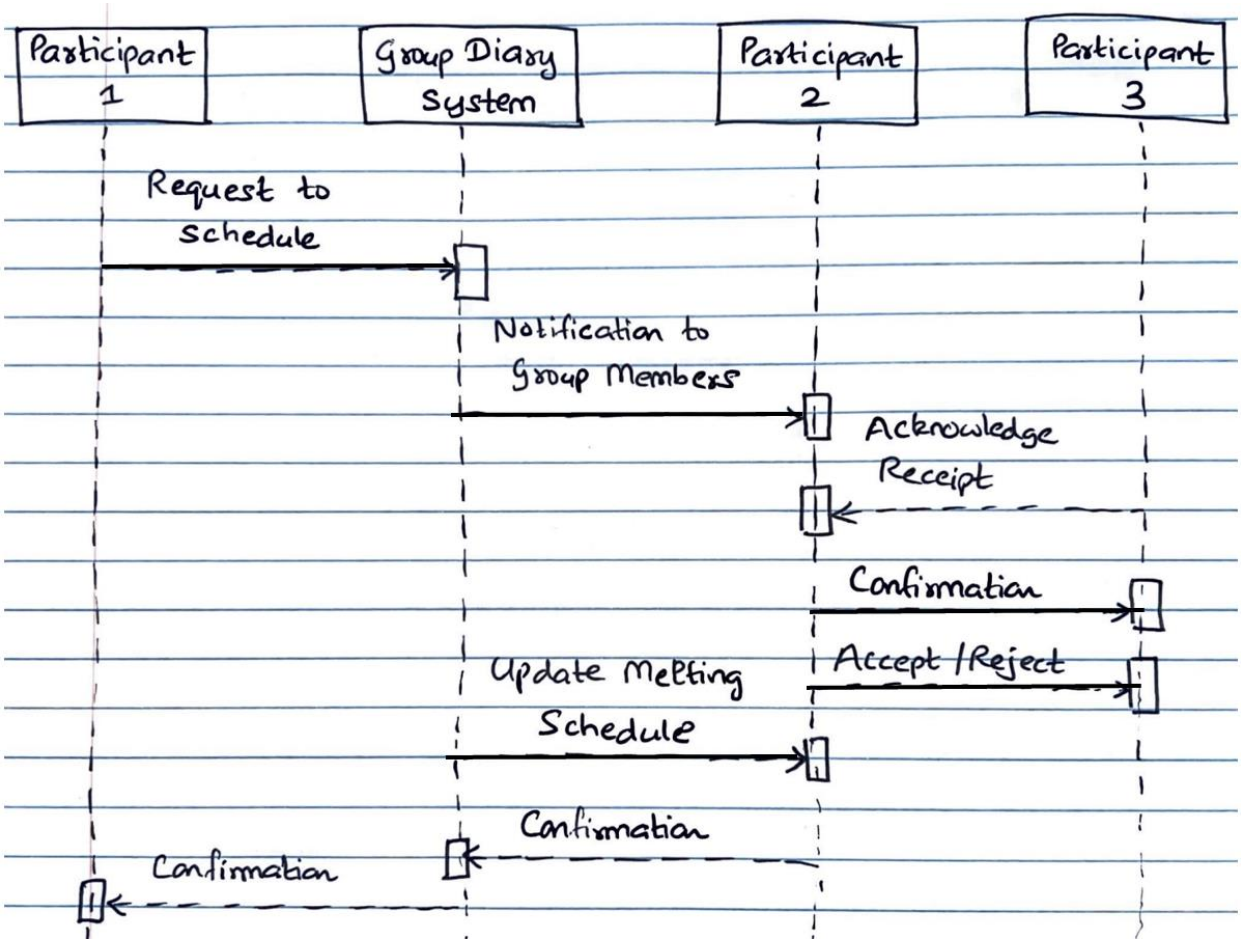
**Summary:**

Even if there are benefits to open sourcing software, it's still important to plan for how the business will monetize its services and set itself apart from competitors. While keeping the basic program open source, the corporation may think about charging for other services like support, advice, or premium features. Success in the open source world requires careful thought out community engagement plans and economic models.

7) **Draw a sequence diagram showing the interactions of objects in a group diary system when a group of people are arranging a meeting. (20 points)**

**Answer:**

Here is a sequence diagram that shows how the various elements interact in a group diary system when individuals are setting up meetings:



The interactions' explanation:

a. **Meeting Requests from Participants:**

Using the Group Diary System, one of the participants makes the initial request to set up a meeting time.

b. **Notification to Group Members:**

All participants are informed about the planned meeting through the Group Diary System.

c. **Recognition of Notification:**

Attendees attest to having received the notice of the meeting.

d. **Acceptance or Rejection:**

Using the Group Diary System, participants react by either accepting or rejecting the meeting invite.

e. **Update Meeting Schedule:**

The Group Diary System modifies the meeting schedule in accordance with participant acceptances or rejections.

f. **Confirmation of modified Schedule:**

Every participant receives a confirmation via the Group Diary System of the modified meeting schedule.

g. **Confirmation Receipt:**

The revised meeting schedule is confirmed to the participants.

8) **Identify possible objects in the following system and develop a class-diagram for them. You may make any reasonable assumptions about the systems when deriving the design. (20 points)**

**A filling station (gas station) is to be set up for fully automated operation. Drivers swipe their credit card through a reader connected to the pump; the card is verified by communication with a credit company computer, and a fuel limit is established. The driver may then take the fuel required. When fuel delivery is complete and the pump hose is returned to its holster, the driver's credit card account is debited with the cost of the fuel taken. The credit card is returned after debiting. If the card is invalid, the pump returns it before fuel is dispensed.**

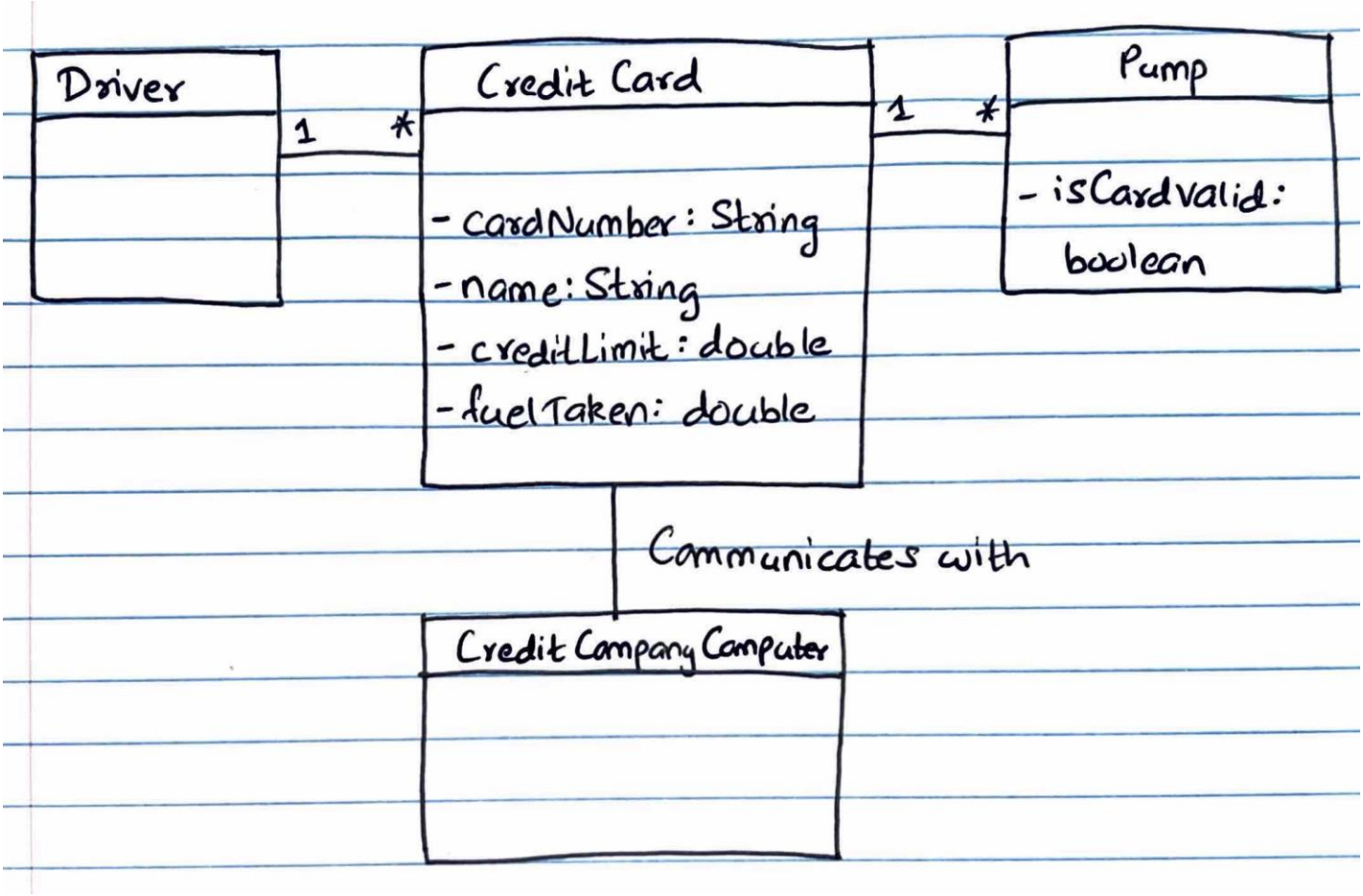
**Answer:**

In order to create a class diagram for the automated gas station system, the potential objects present in this situation are:

Driver/Operator
CreditCard

Pump
CreditCompanyComputer
FuelDeliverySystem

Here is a class diagram that shows the relationships and fundamental functions of these objects:




Relation between two classes:

a. **Driver/Operator:**

Depicts the operator of the gas pump. For the sake of brevity, certain attributes, including a driver's license and name, may be deleted.

b. **CreditCard:**

Provides details on the credit card that was used for the purchase, including the name, credit limit, and quantity of fuel spent. To confirm validity, the card is checked on the CreditCompanyComputer.

c. **Pump**

Controls the fuel delivery mechanism. Before fuel is dispensed, it engages with the CreditCard to confirm its legitimacy. Additionally, it keeps track of the fuel that the driver takes and debits the card when the fuel supply is finished.

**d. CreditCompanyComputer:**

This is a representation of the external system that talks to the CreditCard to establish a fuel limit and confirm its authenticity.

**Summary:**

The fundamental characteristics and relationships between the key components of the automated gas station system are shown in this straightforward class diagram. Additional classes and relationships may be required, depending on the functionality and complexity of the system.