AC51003 – Assignment 2 – Submission

Please use this document as a template for submitting your assignment. It provides space for you to insert the content that is required for this assessment. It also provides space for me to provide you with feedback and a mark before returning the document back to you. The document page size is intentionally set to be A3 to ensure that there is a larger space for you to insert the content required. In some cases, the content will be in the form of UML diagrams, and these may require additional viewing space to be legible. NOTE: feel free to change the page orientation from portrait to landscape and/or a mixture of the two throughout - whatever seems sensible to be able to present your diagram content in the most legible way.

There are two main sections to this document that correspond to the two main components of the assessment: sequence diagrams and design patterns. You can complete each of these sections in whatever order you prefer. Prior to this, you will see below the summary of the assessment information along with an area where your overall mark will be placed.

The first thing you should do is write your name in the space below. Later, when you submit this document to My Dundee, please remember to ensure that you name the document file as follows: *surname_firstname.docx*. Example: *ramsay_craig.docx*. **Please also remember to submit the document as a Microsoft Word file**. I need it to be in Word format to be able to mark your assignment effectively and efficiently.

Please enter your name here:	Ananya Das

YOUR GRADE

Component	*Weighting*	*Grade*
Sequence diagrams	80%	xx
Design patterns task	20%	xx
	COMBINED GRADE BASED ON WEIGHTED ELEMENTS:	xx
	NUMBER OF DAYS LATE:	0
	ANY OTHER ADJUSTMENTS, IF APPLICABLE	N/A
	FINAL GRADE:	**xx**

ASSIGNMENT 2 -DESIGN MODELS

Assessment Summary:

Deadline:	Sunday 31st March 2024, 23.59pm
Hand-in method:	Upload as a Microsoft Word file to My Dundee
Date for feedback:	Sunday 21st April 2024
Late penalties:	One grade point per day late (meaning if a submission is one day late and marked as a C2 it will receive a C3 grade). A day is defined as each 24-hour period following the submission deadline including weekends and holidays. Assignments submitted more than 5 days after the agreed deadline will receive a zero mark (AB).
Percentage of module:	This assessment is worth 40% of the module grade.
What to do:	Prepare and submit sequence diagrams for the Vending Machine system (to be embedded / inserted below). Prepare and submit a report for the Design Patterns task (to be embedded / inserted below).
Learning outcomes:	This assessment addresses the following learning outcomes of the module: • To comprehend and apply software design principles, practices, and notations. • To comprehend and selectively utilize visual notations that pertain to the analysis and design of software systems. • To employ design techniques: to formulate and convey behavioural designs for a software system. • To demonstrate awareness of, and to apply software design principles and software quality characteristics. • To critically evaluate the quality of a software design against known design principles and quality characteristics. • To develop and apply problem-solving, reporting, time-management, and independent study skills.
Academic conduct:	By submitting this file, you agree that the work that it contains is entirely your own work, that relevant sources have been referenced and made clear where applicable, and that your work has not been intentionally copied from or shared with others.

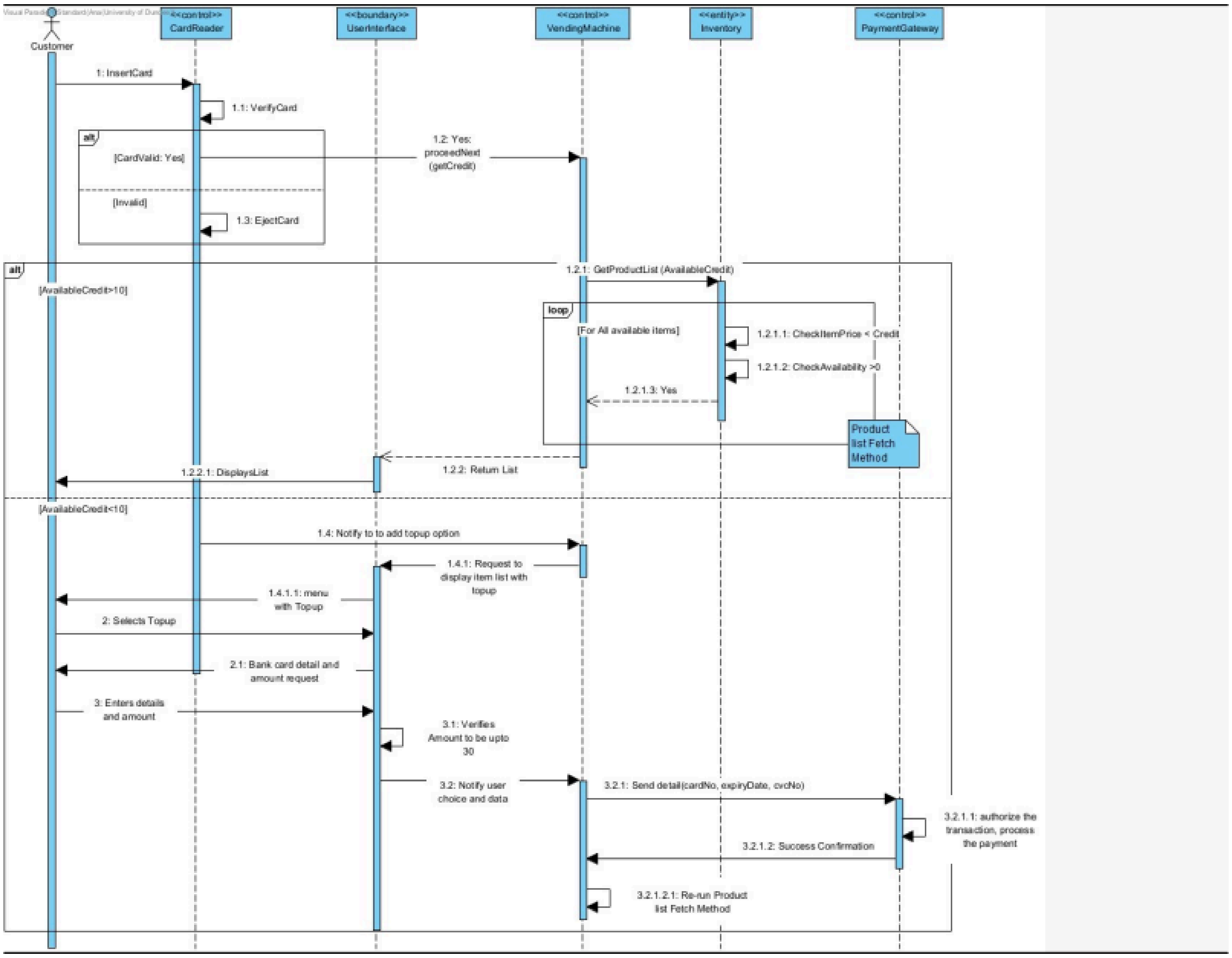
SECTION ONE – SEQUENCE DIAGRAMS (80%)

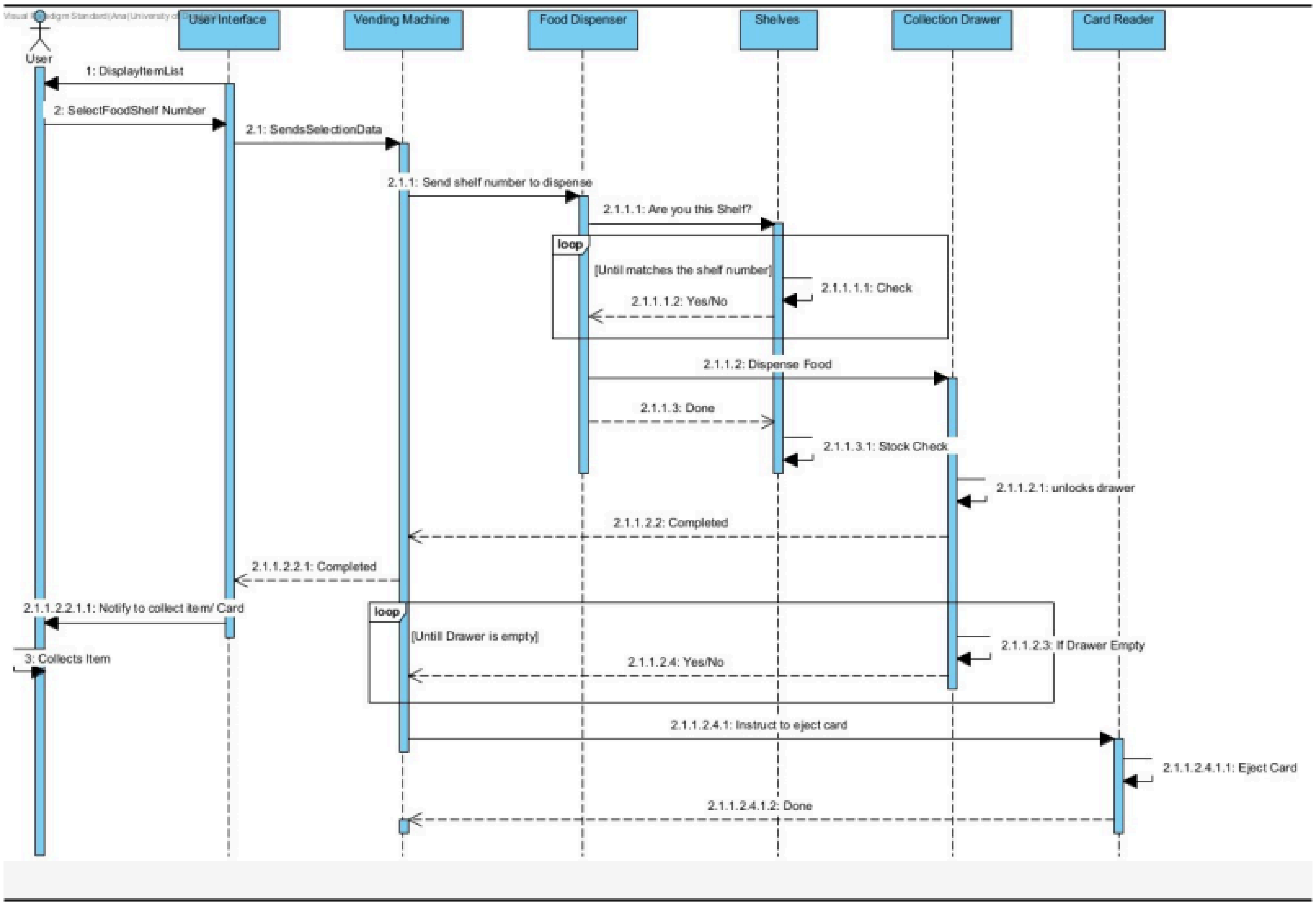
You are being asked to create sequence diagrams for different areas of the system functionality. These different areas of functionality are outlined below, one after another. For each area of functionality, a space is provided for you to insert your sequence diagram solution. Underneath this is a box into which I will be adding any feedback pertaining to your diagram(s) as they are being assessed. For convenience, each diagram is started on a new page. Make sure that you peruse the entire document to ensure that you do not miss anything out that occurs nearer the end, including the Design Patterns section. For each diagram, a summary of the relevant functionality is provided. Please note that it really is just a summary, and you can refer to your system use cases or requirements for a reminder of more details where required. Whilst the summaries of the functionality are brief, remember that the actual logic required to convey them in the sequence diagrams, as an interaction between objects in the system and using the relevant UML notation, may be more complex. You may also need to consider the content of the diagrams holistically. If some of the messages in one diagram are already covered by another diagram, it is not usually necessary to duplicate these same messages again, you can cross-reference them instead, whilst making sure each diagram is also complete with the messaging it requires.

SEQUENCE DIAGRAM ONE: CUSTOMER BUYS A FOOD ITEM USING A VENDING CARD

The first diagram that you create should show the main case of a customer wishing to buy an item of food using a vending card. Specifically, the steps which are summarized below:

1. The customer inserts their card.
2. The system should respond in the appropriate way, e.g., determining and showing the items available for the card balance.
3. The customer selects the item they desire.
4. The system goes through the stages of dispensing the item and finalising the purchase.


Customer
<<control>> CardReader
<<boundary>> UserInterface
<<control>> VendingMachine
<<entity>> Inventory
<<control>> PaymentGateway
1: InsertCard
1.1: VerifyCard
alt
[CardValid: Yes]
1.2: Yes: proceedNext (getCredit)
[Invalid]
1.3: EjectCard
alt
[AvailableCredit>10]
1.2.1: GetProductList (AvailableCredit)
loop
[For All available items]
1.2.1.1: CheckItemPrice < Credit
1.2.1.2: CheckAvailability >0
1.2.1.3: Yes
Product list Fetch Method
1.2.2: Return List
1.2.2.1: DisplaysList
[AvailableCredit<10]
1.4: Notify to to add topup option
1.4.1: Request to display item list with topup
1.4.1.1: menu with Topup
2: Selects Topup
2.1: Bank card detail and amount request
3: Enters details and amount
3.1: Verifies Amount to be upto 30
3.2: Notify user choice and data
3.2.1: Send detail(cardNo, expiryDate, cvcNo)
3.2.1.1: authorize the transaction, process the payment
3.2.1.2: Success Confirmation
3.2.1.2.1: Re-run Product list Fetch Method



Feedback
I will be inserting any feedback in here…

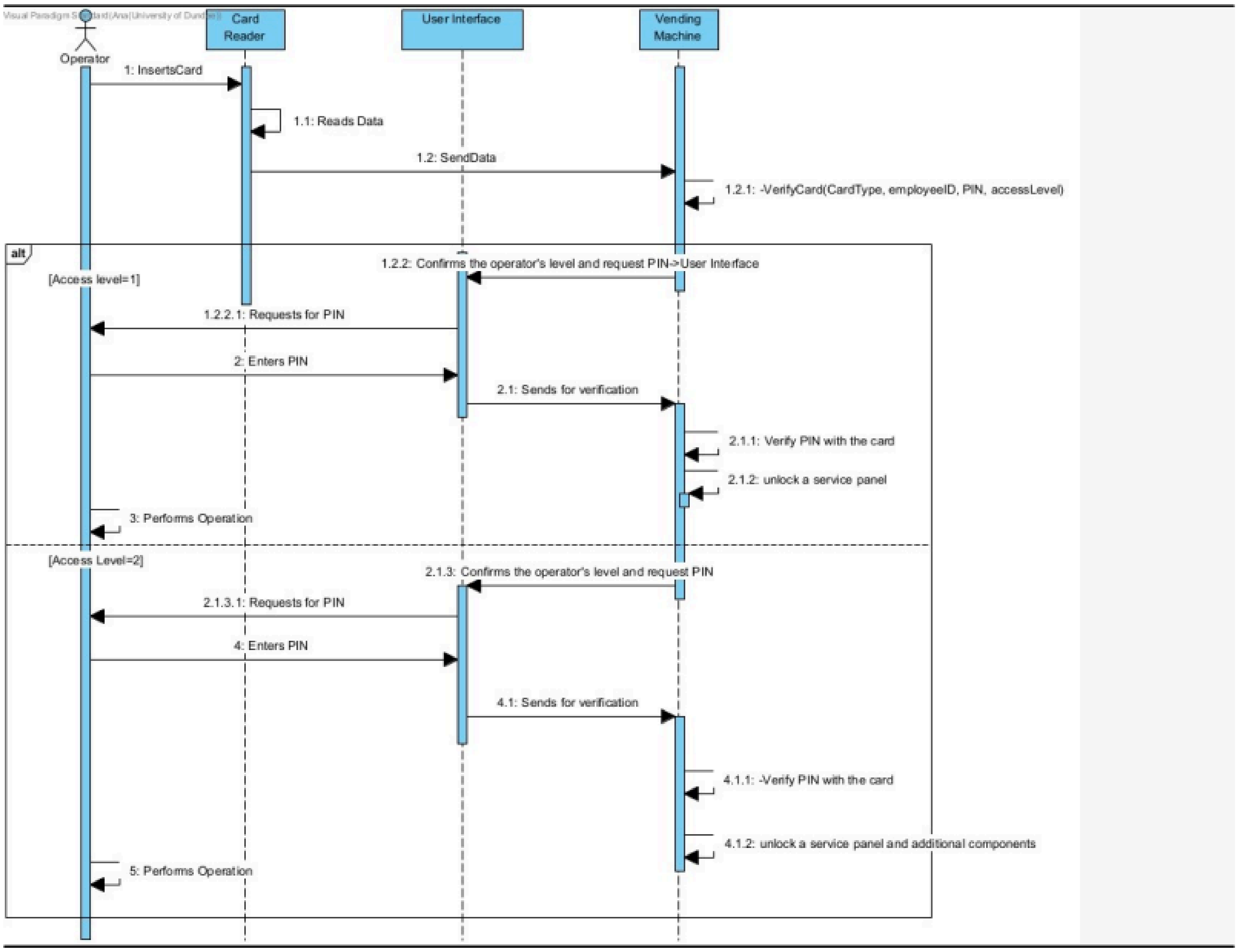
SEQUENCE DIAGRAM TWO: SERVICING THE MACHINE

This diagram should show how a service operator services the machine. Specifically, the steps which are summarized below:

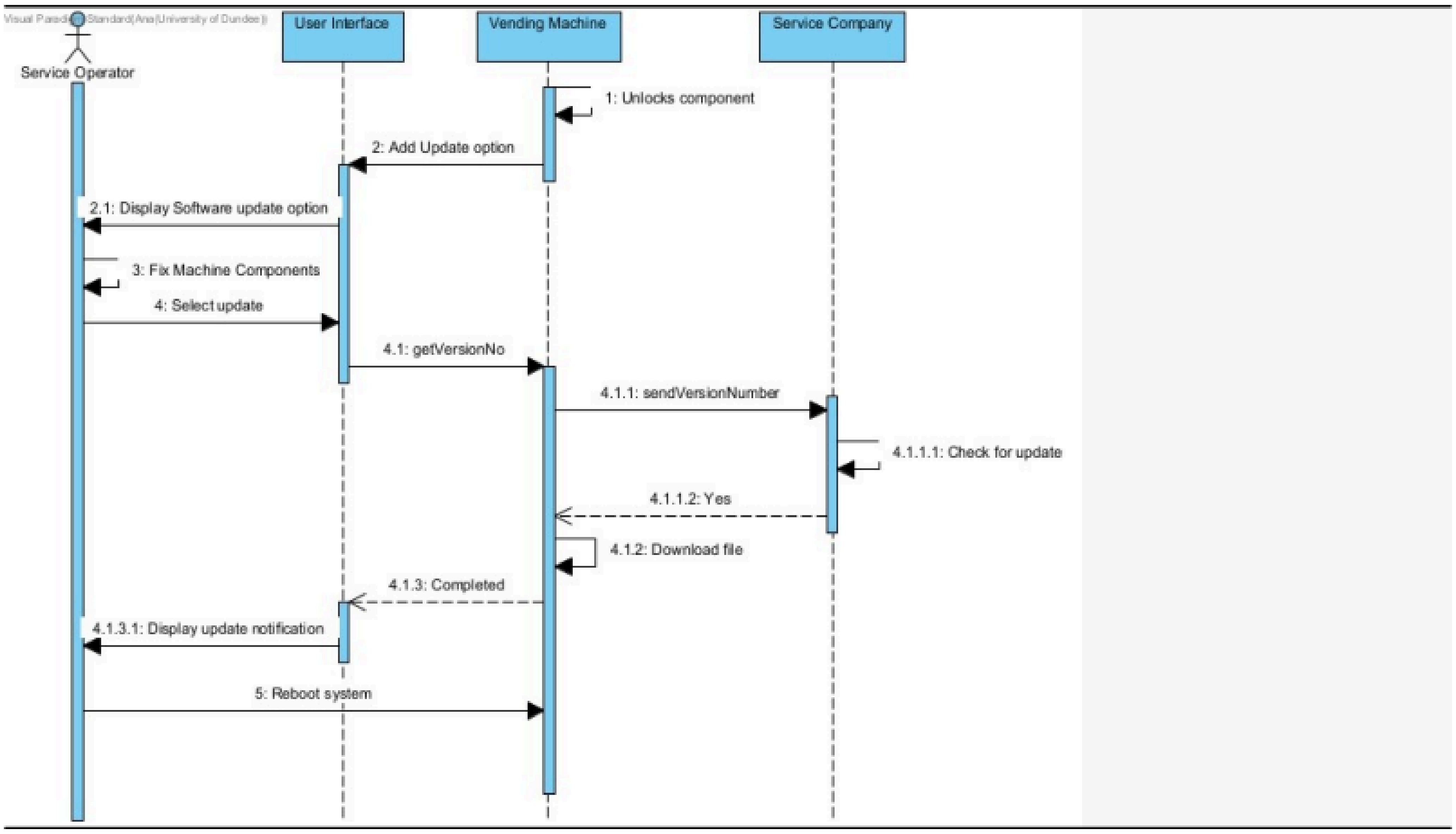
1. A service operator inserts their service card into the machine.
2. The system completes the steps necessary to facilitate the service access, and being mindful of service notifications required, etc.
3. The stages of the service being completed should be conveyed too.

Alternative flows may be considered, and these may attract additional marks.

NOTE: it is not necessary to cover the case of the maintenance operator, just the service operator.



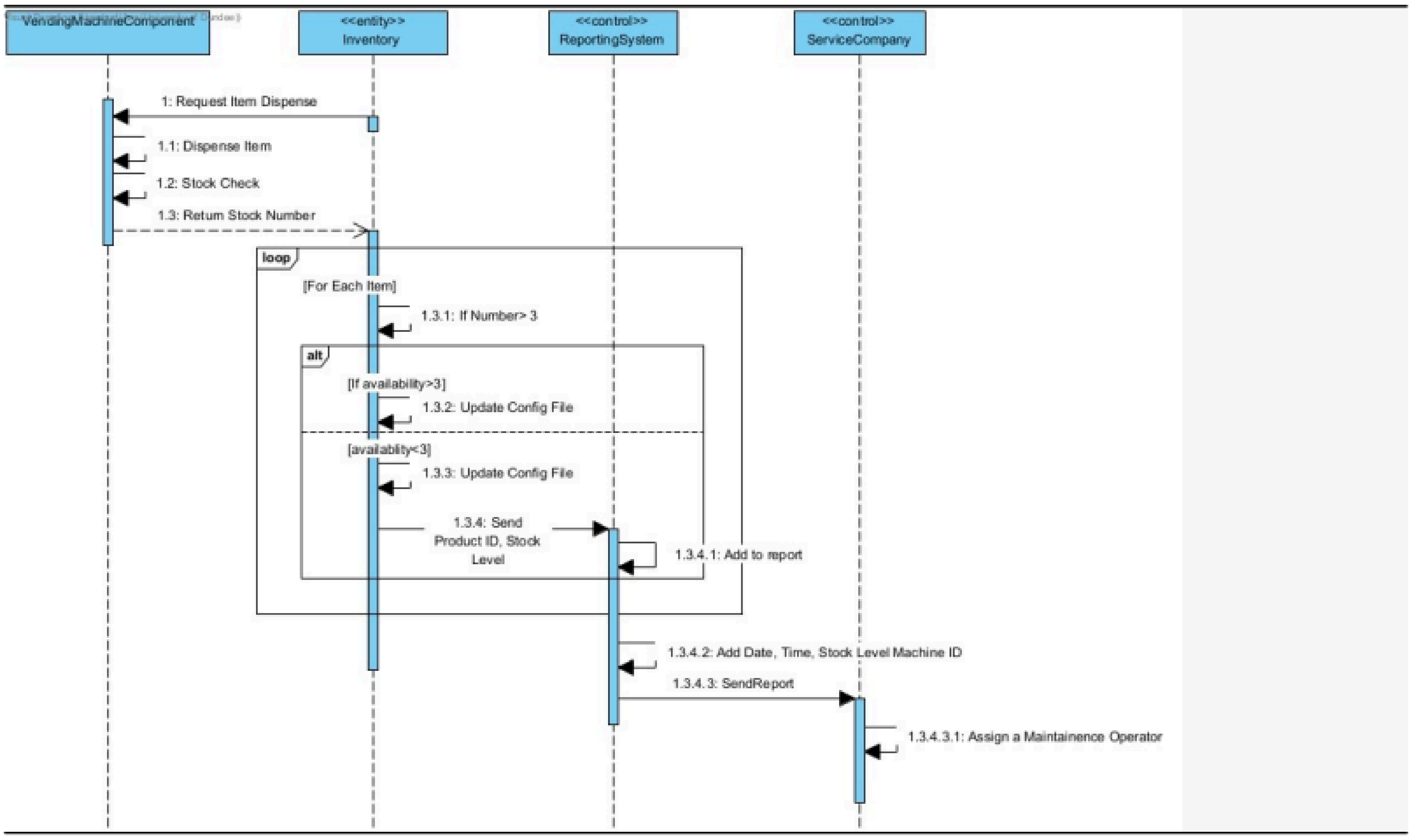
Visual Paradigm Standard(Ana(University of Dundee))
Operator
Card Reader
User Interface
Vending Machine
1: InsertsCard
1.1: Reads Data
1.2: SendData
1.2.1: -VerifyCard(CardType, employeeID, PIN, accessLevel)
alt
[Access level=1]
1.2.2: Confirms the operator's level and request PIN->User Interface
1.2.2.1: Requests for PIN
2: Enters PIN
2.1: Sends for verification
2.1.1: Verify PIN with the card
2.1.2: unlock a service panel
3: Performs Operation
[Access Level=2]
2.1.3: Confirms the operator's level and request PIN
2.1.3.1: Requests for PIN
4: Enters PIN
4.1: Sends for verification
4.1.1: -Verify PIN with the card
4.1.2: unlock a service panel and additional components
5: Performs Operation



Feedback
I will be inserting any feedback in here…

SEQUENCE DIAGRAM THREE: STOCK NOTIFICATIONS

This diagram should show the stage at which the system detects that the stock of a given item has fallen below an acceptable level and so the system must send a notification to the service company. Alternative flows may be considered, and these may attract additional marks.



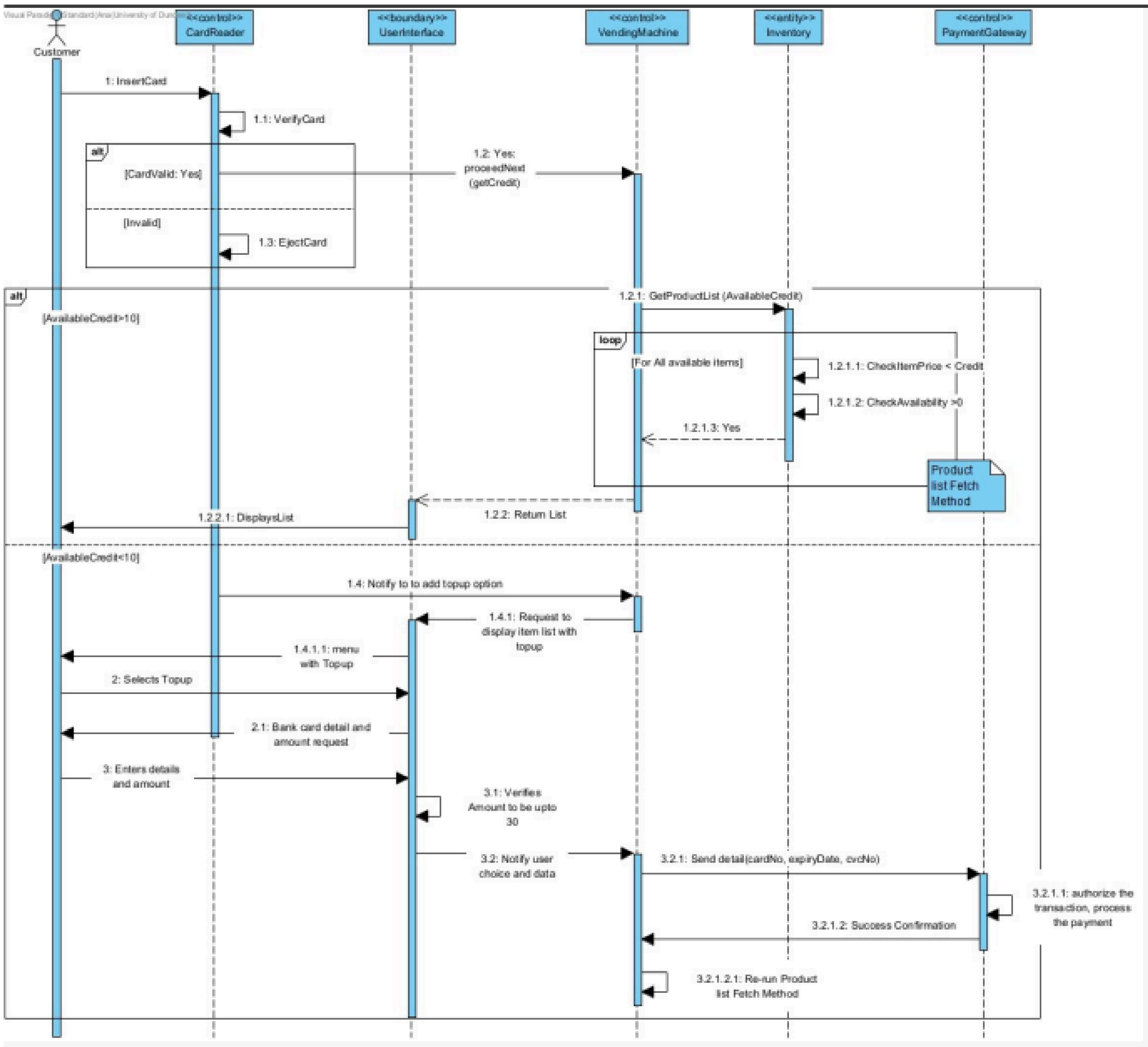
Feedback
I will be inserting any feedback in here…

SEQUENCE DIAGRAM FOUR: TOP-UP CARD

This diagram should show the process for when a customer wishes to top up the amount of credit on their vending card. You can assume that the customer has already inserted their vending card into the machine. A summary of the process to convey is as follows:

1. The customer selects the top-up option.
2. The customer must enter the details of their bank / credit card and the amount of top-up they require.
3. The request must be authorised with the payment system.
4. The customer's card will be topped up.

Alternative flows may be considered, and these may attract additional marks.



Customer
<<control>> CardReader
<<boundary>> UserInterface
<<control>> VendingMachine
<<entity>> Inventory
<<control>> PaymentGateway
1: InsertCard
1.1: VerifyCard
alt
[CardValid: Yes]
1.2: Yes: proceedNext (getCredit)
[Invalid]
1.3: EjectCard
alt
[AvailableCredit>10]
1.2.1: GetProductList (AvailableCredit)
loop
[For All available items]
1.2.1.1: CheckItemPrice < Credit
1.2.1.2: CheckAvailability >0
1.2.1.3: Yes
Product list Fetch Method
1.2.2: Return List
1.2.2.1: DisplaysList
[AvailableCredit<10]
1.4: Notify to to add topup option
1.4.1: Request to display item list with topup
1.4.1.1: menu with Topup
2: Selects Topup
2.1: Bank card detail and amount request
3: Enters details and amount
3.1: Verifies Amount to be upto 30
3.2: Notify user choice and data
3.2.1: Send detail(cardNo, expiryDate, cvcNo)
3.2.1.1: authorize the transaction, process the payment
3.2.1.2: Success Confirmation
3.2.1.2.1: Re-run Product list Fetch Method

Feedback
I will be inserting any feedback in here…

SECTION TWO – DESIGN PATTERNS TASK (20%)

Please see the separate 'Design Pattern Brief' for what you are required to do. Please insert your solution below. You can insert the necessary text for your answer as well as the UML diagram(s) required too. A reminder of what to provide:

(i) The name of the design principle which is being violated by the design presented in Figure 3 in the Design Patterns Brief and explain why it is being violated.
(ii) A brief description of the problem(s) that you have identified with the design presented in Figure 3 in relation to code maintainability, making clear what the undesirable impact of future changes might be.
(iii) A brief explanation of what the Decorator design pattern is and why you think it might be appropriate to consider for this scenario.
(iv) A UML class diagram which shows the re-working of the original design (Figure 3) to incorporate the Decorator design pattern instead, showing the relevant classes, fields, methods, and relationships between them.
(v) A brief explanation of the benefits that your re-worked design provides and, specifically, how it resolves the violation of the design principle that you identified in (i) above, and how it addresses the problem(s) that you identified in (ii) above.
(vi) A UML sequence diagram which provides an example of how the object(s) in your new design could be used to fulfil their intended purpose.

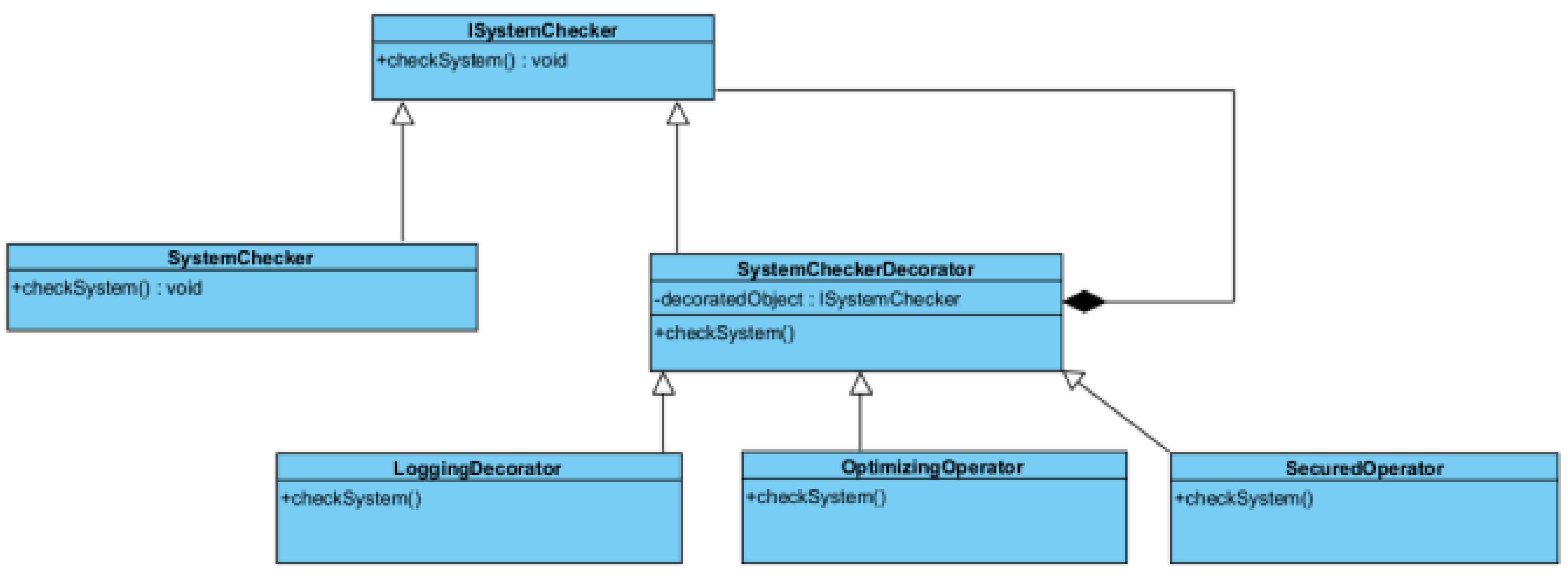
Please provide your answer in the space below. Expand the space to accommodate your answer.

(i) The design principle being violated in this case is the Single Responsibility Principle (SRP). SRP states that a class should have only one reason to change, meaning it should have only one responsibility or job within the system. Initially, when class had clear functionality. However in Figure 3; introducing child classes to represent combinations and permutations of different specialisms, the responsibility of each derived class becomes unclear. Each class potentially encompasses multiple responsibilities that violate the SRP because these classes are now responsible for more than one aspect of system checking.

(ii) Complexity:
Now this design will create a rather complex system and creating the code would be also difficult. If the developer changes, the next developer will have to struggle to understand the relations between the classes. A change of code in the future will introduce multiple errors.
Coupling:
Even if the code is created, as the classes now have multiple responsibilities, cohesion is also decreased, so it would be harder to reuse the code as well. Introducing new functionality or modifying existing functionality will likely require changes to multiple classes. As this design relies on inheritance to create different combinations of functionalities, there may be tight coupling between classes. Changing one chalss will have an impact on the inherited classes, need to modify multiple classes which are related (Ex. optimizedChecker, secureOptimizedChecker, optimizedLoggedChecker, etc).
Duplication:
Two or more classes now have similar functionality, which will lead to code duplication. As similar logic is implemented in multiple places, it will become very hard to maintain consistency. Fixing code will be very time-consuming as well.
Scalability:
As the coupling tends to impossible in this case, reusing the code will be difficult. It will become less scalable now. Adding new functionalities would require creating additional classes.
Testing:
As each class encapsulates multiple functionalities, the responsibilities of each class are very unclear which makes it very hard to isolate and test individual responsibilities

(iii) The Decorator design pattern is a structural pattern that allows behaviour to be added to individual objects, dynamically, without affecting the behaviour of other objects from the same class. It is used to extend or modify the functionality of objects at runtime by wrapping them with one or more decorator objects. In this scenario, the Decorator pattern could be appropriate because it allows for the dynamic addition of functionalities to the system checker classes without the need to create numerous subclasses for every combination of functionalities. Instead of creating separate classes for each combination of logging, optimizing, securing, etc., we can define a set of decorators, each responsible for adding a specific functionality. The Parent system checker class serves as the base component and LoggingDecorator, OptimizingDecorator, SecuringDecorator can be created to add logging, optimizing, securing, or other functionalities as needed. These decorators can be combined and stacked in various ways to achieve different permutations of functionalities.

(iv)

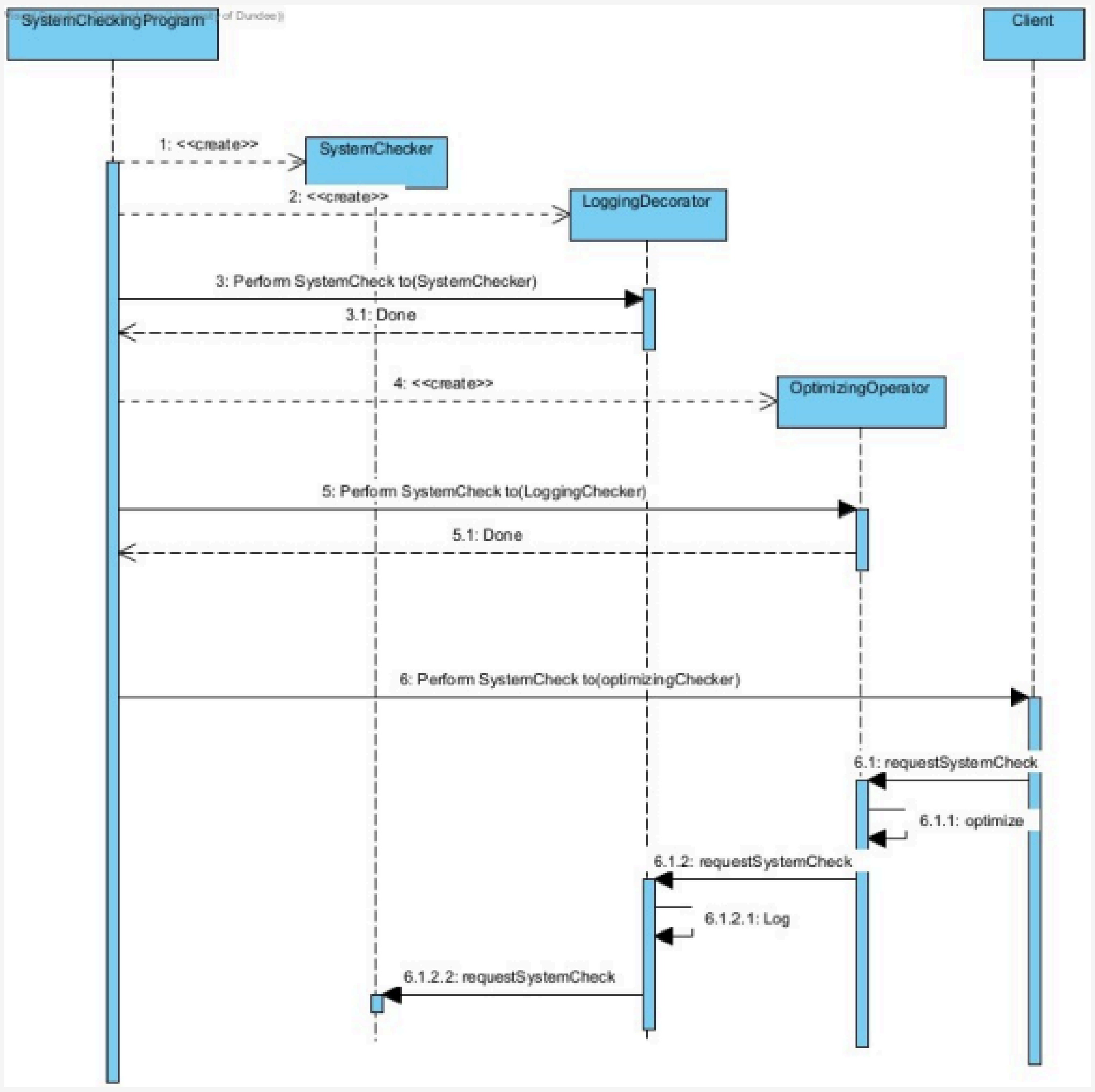


(v)

By using the Decorator pattern, we can achieve the following benefits:
Flexibility:
Decorators can be added or removed dynamically at runtime, allowing for greater flexibility in combining functionalities. This eliminates the need for creating multiple subclasses to represent different combinations of functionalities.
Single Responsibility Principle (SRP):
Each decorator is responsible for a single functionality, adhering to the SRP. This makes the codebase easier to maintain and understand compared to the inheritance-based approach where multiple responsibilities are often bundled within a single subclass.
Code Reusability:
Decorators can be reused across different components within the system, promoting code reusability and reducing duplication.

Scalability:
As new functionalities are added or requirements change, new decorators can be easily created and integrated into the system without the need for extensive modifications to existing code.

ISystemChecker is the interface or abstract class defining the basic behaviour of a system checker. SystemCheckerDecorator is the abstract decorator class implementing the SystemChecker interface and holding a reference to the component. LoggingDecorator, OptimizingDecorator, and SecuringDecorator etc. are concrete decorator classes implementing additional functionalities like logging, optimizing, and securing, respectively. Relationships between classes show how decorators can be applied to the concrete system checker to add or modify its behavior dynamically.



(vi)

Feedback
I will be inserting any feedback in here…