Experiment No. 5

SEMESTER: V (2024-2025) **DATE OF DECLARATION:** 1-08-24

SUBJECT: SE **DATE OF SUBMISSION:** 8-08-24

NAME OF THE STUDENT: Shaun Menezes **ROLL NO:** 40

AIM	To Design Use Case Diagram, Activity Diagram and Sequence Diagram for "Monthly Finance Tracker"
LEARNING OBJECTIVE	The student will understand need, notations and steps in the construction of Use Case Diagram, Activity Diagram and Sequence Diagram for "Monthly Finance Tracker"
LEARNING OUTCOME	The student will create Use Case Diagram, Activity Diagram and Sequence Diagram for "Monthly Finance Tracker"
COURSE OUTCOME	**CO501.1:** The student will demonstrate and apply the basic knowledge in software engineering. **CO501.3:** The student will work collaboratively to plan, design, develop, document and validate the software project. **CO501.5:** The Student will have understanding of impact of sound engineering principle.
PROGRAM OUTCOME	**PO1:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems. **PO3:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations. **PO9:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
BLOOM'S TAXONOMY LEVEL	Understand Evaluate

	Apply Create
THEORY	Introduction to UML: The Unified Modeling Language (UML) is "a standard language for writing software blueprints. UML may be used to visualize, specify, construct, and document the artifacts of a software-intensive system" [Boo05]. In other words, just as building architects create blueprints to be used by a construction company, software architects create UML diagrams to help software developers build the software. If you understand the vocabulary of UML (the diagrams' pictorial elements and their meanings), you can much more easily understand and specify a system and explain the design of that system to others. Grady Booch, Jim Rumbaugh, and Ivar Jacobson developed UML in the mid- 1990s with much feedback from the software development community. UML merged a number of competing modeling notations that were in use by the software industry at the time. In 1997, UML 1.0 was submitted to the Object Management Group, a nonprofit consortium involved in maintaining specifications for use by the computer industry. UML 1.0 was revised to UML 1.1 and adopted later that year. The current standard is UML 2.0 and is now an ISO standard. Because this standard is so new, many older references, such as [Gam95] do not use UML notation. UML 2.0 provides 13 different diagrams for use in software modeling. You should note that there are many optional features in UML diagrams. The UML language provides these (sometimes arcane) options so that you can express all the important aspects of a system. At the same time, you have the flexibility to suppress those parts of the diagram that are not relevant to the aspect being modeled in order to avoid cluttering the diagram with irrelevant details. Therefore, the omission of a particular feature does not mean that the feature is absent; it may mean that the feature was suppressed. **Use case Diagram:** Use cases and the UML *use-case diagram* help you determine the functionality and features of the software from the user's perspective. To give you a feeling for how use cases and use-case diagrams work, I'll create some for a software application for managing digital music files, similar to Apple's iTunes software.

	Some of the things the software might do include: • Download an MP3 music file and store it in the application's library. • Capture streaming music and store it in the application's library. • Manage the application's library (e.g., delete songs or organize them in • playlists). • Burn a list of the songs in the library onto a CD. • Load a list of the songs in the library onto an iPod or MP3 player. • Convert a song from MP3 format to AAC format and vice versa. This is not an exhaustive list, but it is sufficient to understand the role of use cases and use-case diagrams. A use case describes how a user interacts with the system by defining the steps required to accomplish a specific goal (e.g., burning a list of songs onto a CD). Variations in the sequence of steps describe various scenarios (e.g., what if all the songs in the list don't fit on one CD?). A UML use-case diagram is an overview of all the use cases and how they are related. It provides a big picture of the functionality of the system. A use-case diagram for the digital music application is shown Figure 1.

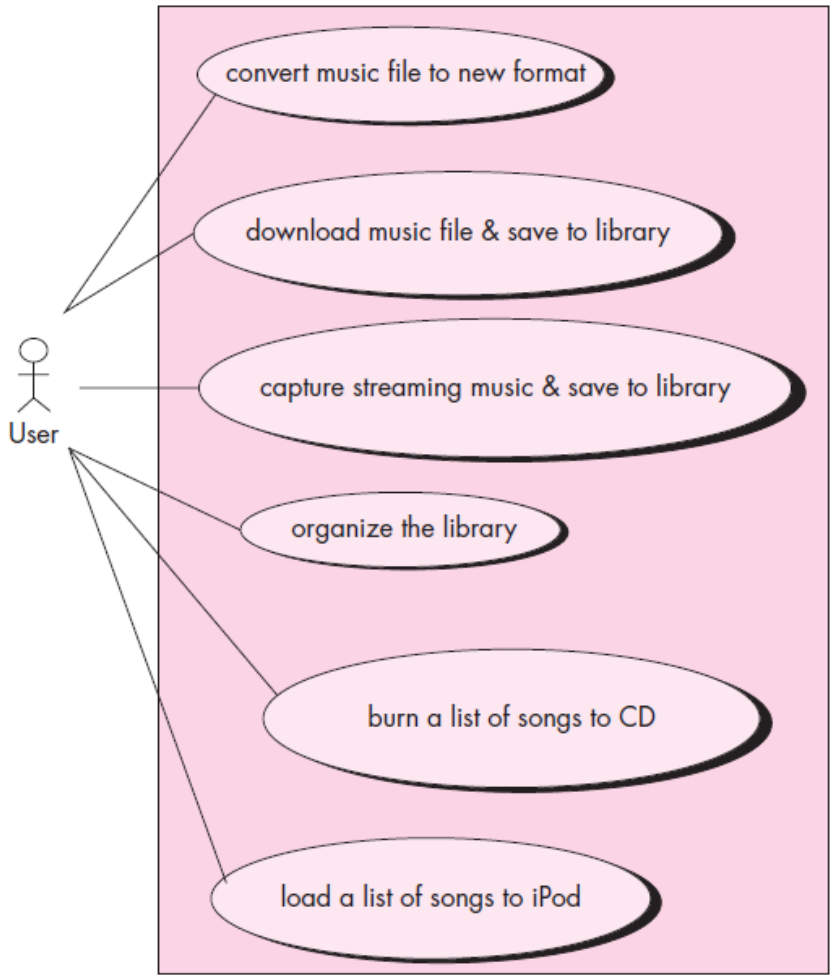


Figure 1: A use case diagram for the music system

In this diagram, the stick figure represents an actor (Chapter 5) that is associated with one category of user (or other interaction element). Complex systems typically have more than one actor. For example, a vending machine application
might have three actors representing customers, repair personnel, and vendors
who refill the machine. In the use-case diagram, the use cases are displayed as ovals. The actors are connected by lines to the use cases that they carry out. Note that none of the details of the use cases are included in the diagram and instead need to be stored separately.

Note also that the use cases are placed in a rectangle but the actors are not. This rectangle is a visual reminder of the system boundaries and that the actors are outside the system.

Activity Diagram:

A UML activity diagram depicts the dynamic behavior of a system or part

	of a system through the flow of control between actions that the system performs. It is similar to a flowchart except that an activity diagram can show concurrent flows. The main component of an activity diagram is an action node, represented by a rounded rectangle, which corresponds to a task performed by the software system. Arrows from one action node to another indicate the flow of control. That is, an arrow between two action nodes means that after the first action is complete the second action begins. A solid black dot forms the initial node that indicates the starting point of the activity. A black dot surrounded by a black circle is the final node indicating the end of the activity. A fork represents the separation of activities into two or more concurrent activities. It is drawn as a horizontal black bar with one arrow pointing to it and two or more arrows pointing out from it. Each outgoing arrow represents a flow of control that can be executed concurrently with the flows corresponding to the other outgoing arrows. These concurrent activities can be performed on a computer using different threads or even using different computers. Figure 2 shows a sample activity diagram involving baking a cake. The first step is finding the recipe. Once the recipe has been found, the dry ingredients and wet ingredients can be measured and mixed and the oven can be preheated. The mixing of the dry ingredients can be done in parallel with the mixing of the wet ingredients and the preheating of the oven.

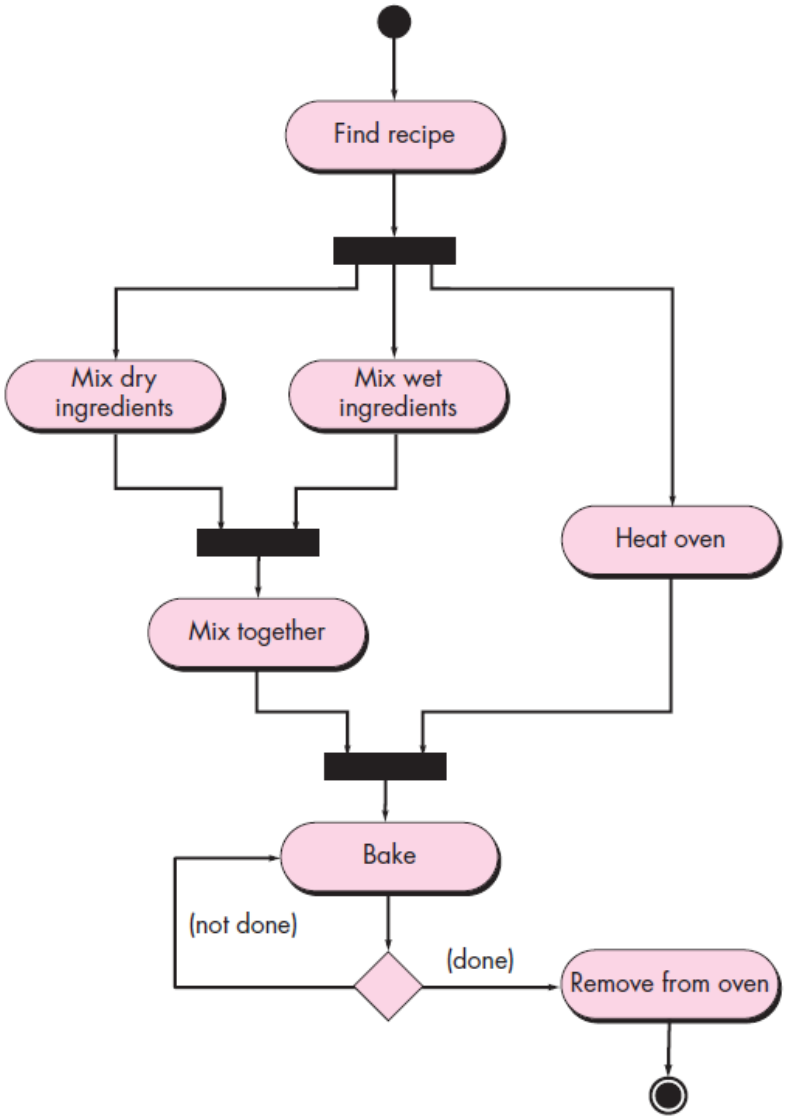


Figure 2 : UML activity diagram showing how to bake a cake

Sequence Diagram:

A join is a way of synchronizing concurrent flows of control. It is represented by a horizontal black bar with two or more incoming arrows and one outgoing arrow. The flow of control represented by the outgoing arrow cannot begin execution until all flows represented by incoming arrows have been completed.

In Figure 2, we have a join before the action of mixing together the wet and dry ingredients. This join indicates that all dry ingredients must be mixed and all wet ingredients must be mixed before the two mixtures can be combined. The second join in the figure indicates that, before the baking of the cake can begin, all ingredients must be mixed together and the oven must be at the right temperature.

Sequence diagram:

A sequence diagram is used to show the dynamic communications between objects during execution of a task. a sequence diagram is used to show the dynamic communications between objects during execution of a task.

In Figure 3, you see a sequence diagram for a drawing program. The diagram shows the steps involved in highlighting a figure in a drawing when it has been clicked. Each box in the row at the top of the diagram usually corresponds to an object, although it is possible to have the boxes model other things, such as classes.

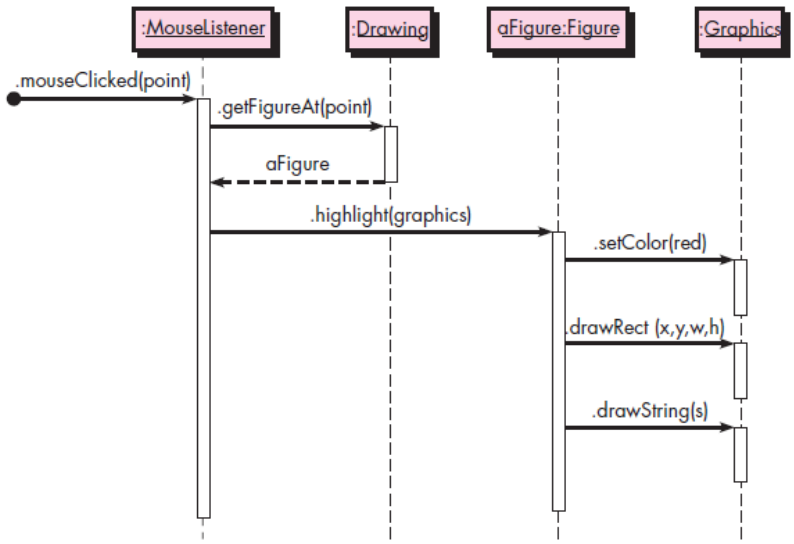
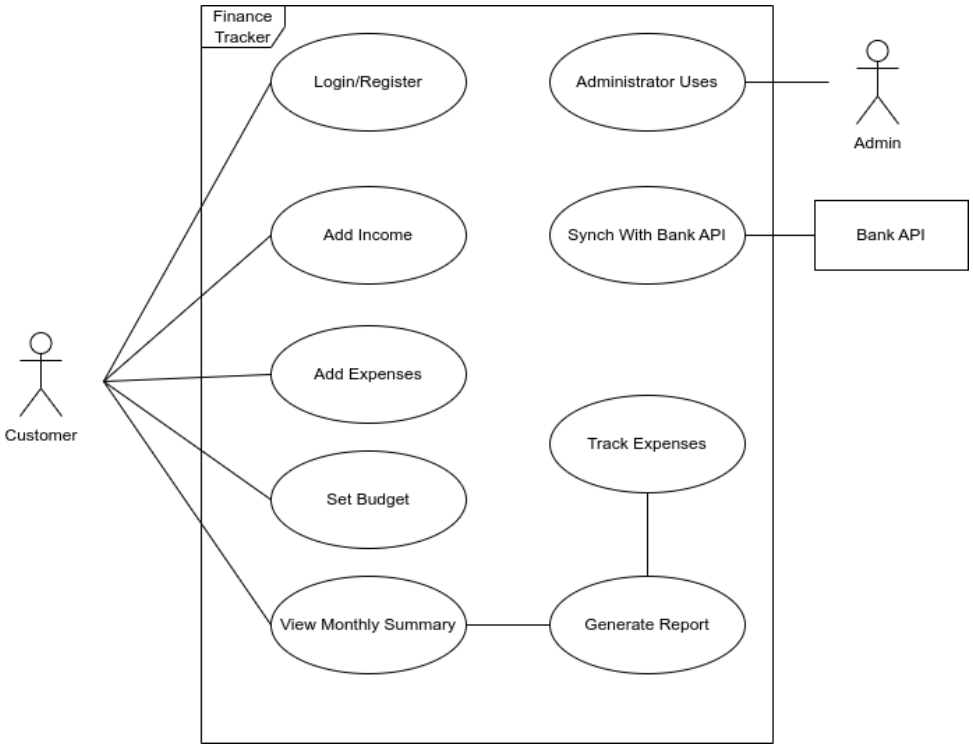


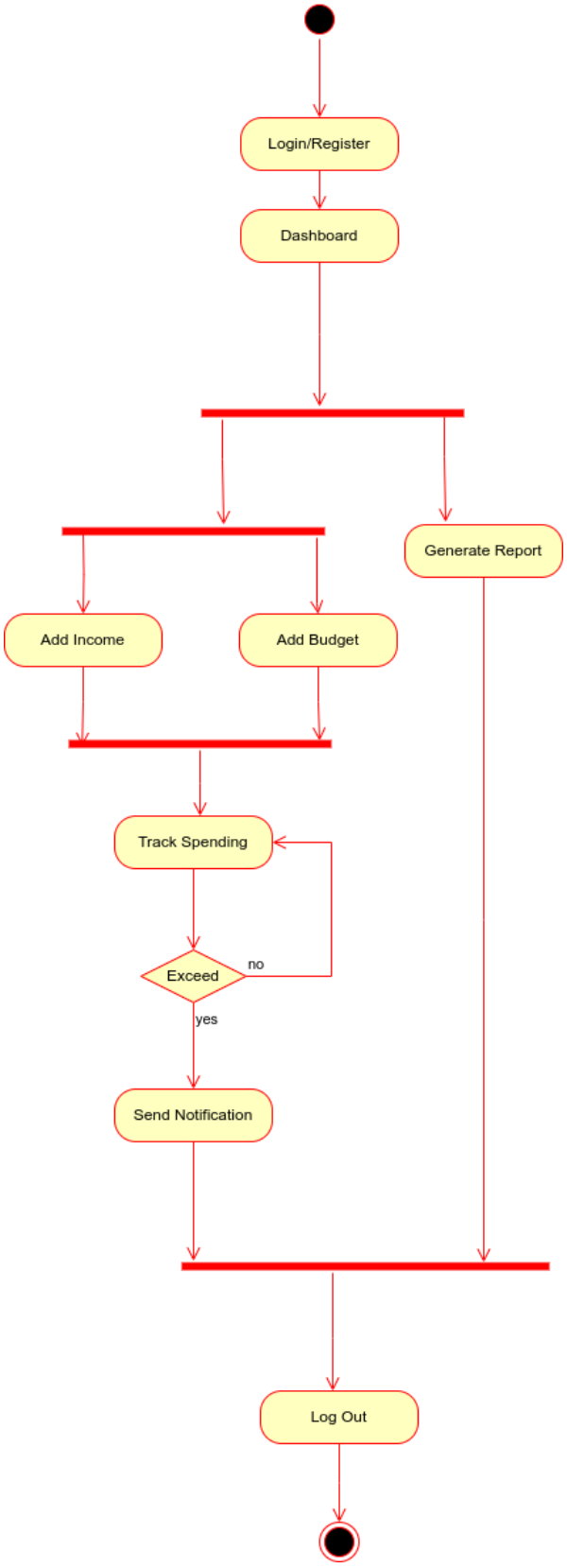
Figure 3: Sample Sequence Diagram

If the box represents an object (as is the case in all our examples), then inside the box you can optionally state the type of the object preceded by the colon. You can also precede the colon and type by a name for the object, as shown in the third box in Figure 3. Below each box there is a dashed line called the lifeline of the object. The vertical axis in the sequence diagram corresponds to time, with time increasing as you move downward.

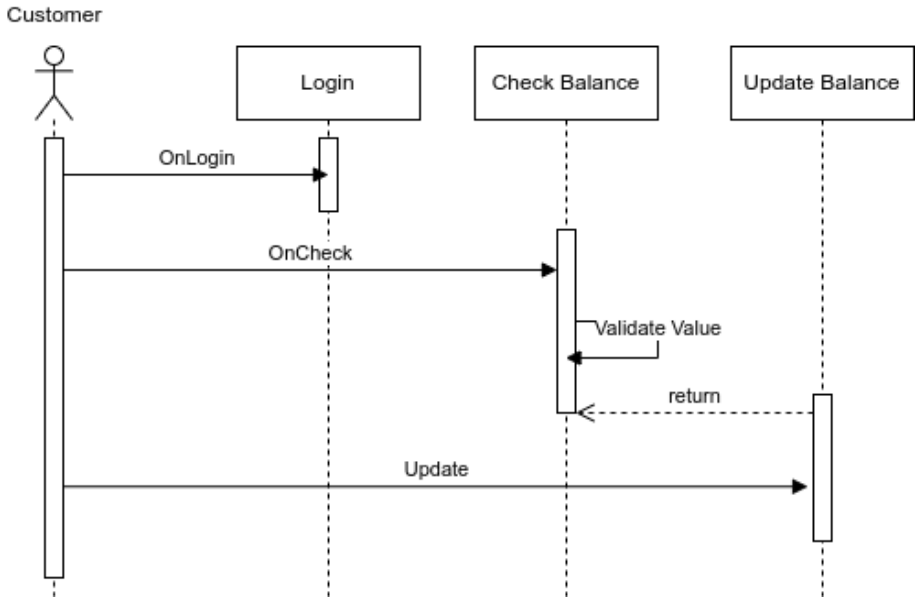
A sequence diagram shows method calls using horizontal arrows from the caller to the callee, labeled with the method name and optionally including its parameters, their types, and the return type. For example, in Figure 3, the MouseListener calls the Drawing's getFigureAt() method. When an object is executing a method (that is, when it has an activation frame on the stack), you can optionally display a white bar, called an activation bar,

	down the object's lifeline. In Figure 3, activation bars are drawn for all method calls. The diagram can also optionally show the return from a method call with a dashed arrow and an optional label. In Figure 3. the getFigureAt() method call's return is shown labeled with the name of the object that was returned. A common practice, as we have done in Figure 3, is to leave off the return arrow when a void method has been called, since it clutters up the diagram while providing little information of importance. A black circle with an arrow coming from it indicates a found message whose source is unknown or irrelevant. You should now be able to understand the task that Figure 3 is displaying. An unknown source calls the mouseClicked() method of a MouseListener, passing in the point where the click occurred as the argument. The MouseListener in turn calls the getFigureAt() method of a Drawing, which returns a Figure. The MouseListener then calls the highlight method of Figure, passing in a Graphics object as an argument. In response, Figure calls three methods of the Graphics object to draw the figure in red.
LAB EXERCISE	**Use Case Diagram** 

Activity Diagram



Sequence Diagram



Group Details

Sr. No.	Roll No.	Name of the Batch	Name of the Student
1.	34	B	Aibal Biju
2.	35	B	Ramya Kulkarni
3.	40	B	Shaun Menezes

REFERENCES

1. Roger Pressman, Software Engineering: A Practitioners Approach, (7th Edition), McGraw Hill
2. http://www.uml-diagrams.org/uml-24-diagrams.html
3. http://creately.com/blog/diagrams/uml-diagram-types-examples/
4. http://www.tutorialspoint.com/uml/uml_tutorial.pdf
5. https://www.smartdraw.com/data-flow-diagram/