**Week 4: UML Diagrams [ Sequence & Component ] and Working with Git**

1. Explanation of sequence diagram and component diagram with ATM as example.
2. Hands on practice with basic git commands like, init, add, commit, diff, log branch etc.
3. Uploading of SRS document with the two UML diagrams discussed as assignment.

**Objective :**

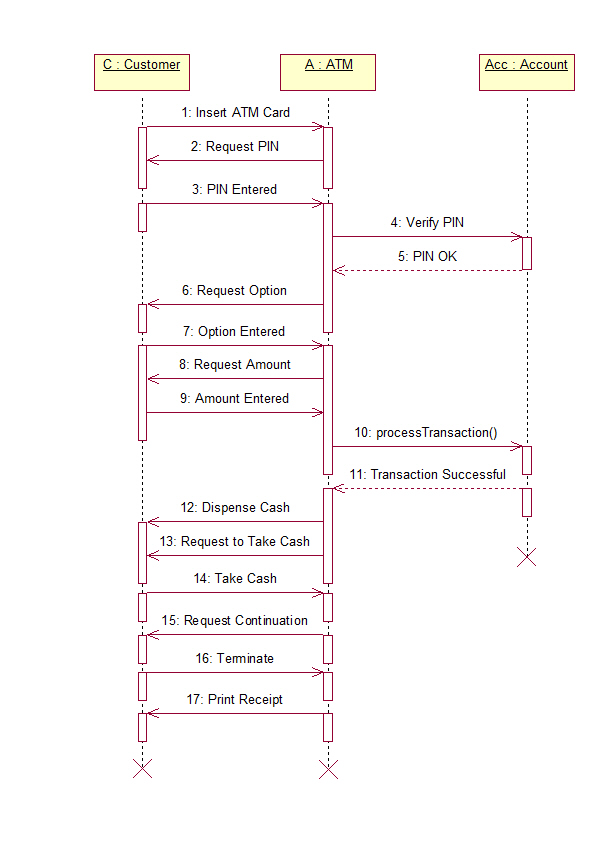
**To understand and apply the concepts of UML by creating a conceptual model, a Sequence and Component diagram for their respective project.**

**Sequence Diagram :**

A sequence diagram is a type of interaction diagram used to represent how objects interact in a particular sequence to accomplish a specific task or process.

**Key Elements of Sequence Diagrams:**

1. **Objects/Participants**: These are entities that interact with each other in the system, represented as rectangles with the object name at the top. Participants can be anything that sends or receives messages, like classes, actors, or components.
2. **Lifelines:** A lifeline represents the existence of an object over time, depicted as a vertical dashed line under each participant.
3. **Activation Bars**: These bars indicate the period during which an object is performing an action or is active, typically shown as a narrow vertical rectangle on a lifeline.
4. **Messages and Return Messages**: Arrows between lifelines represent messages or interactions between objects. Dashed arrows represent return messages that indicate a response to a previously sent message.

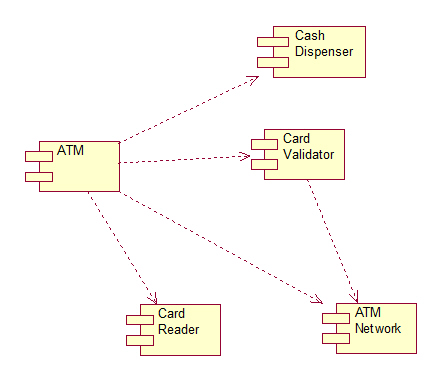


**Component Diagram:**

A component diagram is a type of structural diagram in the Unified Modeling Language (UML) used to represent the high-level architecture of a system. It focuses on the organization and dependencies of components within a system, showing how the system is divided into components and how they interact.

**Key Elements of Component Diagrams:**

1. **Components**:
   * Represented as rectangles with a name and sometimes with a small rectangle (a "lollipop")
   * A component is a modular part of a system, encapsulating a set of related functions or services. For example, components can represent software modules, subsystems, or libraries.
2. **Relationships**:
   * **Dependency**: Shown as a dashed arrow, indicating that one component depends on another.
   * **Association**: Represented by a solid line, showing a structural relationship between components



**Git Commands :**

**1. Setup and Configuration**

* git version : The command git version is used to check the version of git.

**git --version**

* git config: Configures Git settings. Commonly used to set up user information

**git config --global user.name "Your Name"**

**git config --global user.email "youremail@example.com"**

* **git config --list**: Displays all the Git configurations for the current user.

**2. Repository Management**

* **git init**: Initializes a new Git repository in the current directory

git init

* **git clone**: Creates a copy of an existing Git repository from a remote source (e.g., GitHub) to your local machine

git clone https://github.com/username/repository.git

**3. Staging and Committing**

* **git status**: Shows the status of changes in your working directory and staging area. It tells you which files are untracked, modified, or ready to be committed.

**git status**

* **git add**: Adds changes in the working directory to the staging area.

**git add filename.txt # Adds a specific file**

**git add . # Adds all changes in the directory**

* **git commit**: Commits the staged changes to the repository with a descriptive message. The -m option allows you to include a commit message.

**git commit -m "Commit message describing changes"**

**4. Branching and Merging**

* **git branch**: Lists all branches or creates a new branch

**git branch # Lists all branches**

**git branch branch-name # Creates a new branch**

**git branch -d <branch\_name> # Deletes a branch**

* **git checkout**: Switches to a different branch

**git checkout branch-name # Switches to an existing branch**

**git checkout -b new-branch # Creates and switches to a new branch**

* **git merge**: Merges the specified branch into the current branch. This command integrates the changes from the feature branch into the main branch.

**git checkout main # Switch to the main branch**

**git merge branch-name # Merge branch-name into main**

1. **Undoing Changes**

* git reset :Removes the specified file from the staging area but leaves the working directory unchanged. git reset --hard can also reset the working directory and staging area to the last commit.

**git reset <file>:**

* git revert**:** Creates a new commit that undoes the changes from a specified commit, leaving the history intact**.**

**git revert <commit>:**

1. **Viewing History**

* git log**:** Shows a history of commits in the repository, including commit hashes, messages, and timestamps. Use git log --oneline for a more concise view.

**git log:**

* git diff**:** Displays differences between various commits, the working directory, and the staging area. git diff without arguments shows changes not yet staged.

**git diff:**

* git show**:** Shows the details of a specific commit, including the changes made and the commit message.

**git show <commit>**

**Conclusion:**

By following these steps, students will create a conceptual model using sequence and component diagrams for their respective project using **Star UML**

By the end of this session, students are familiarized with the fundamentals of version control tool (git) that helps developers manage and collaborate on code.

**Note: Upload your work in the tesselator.**