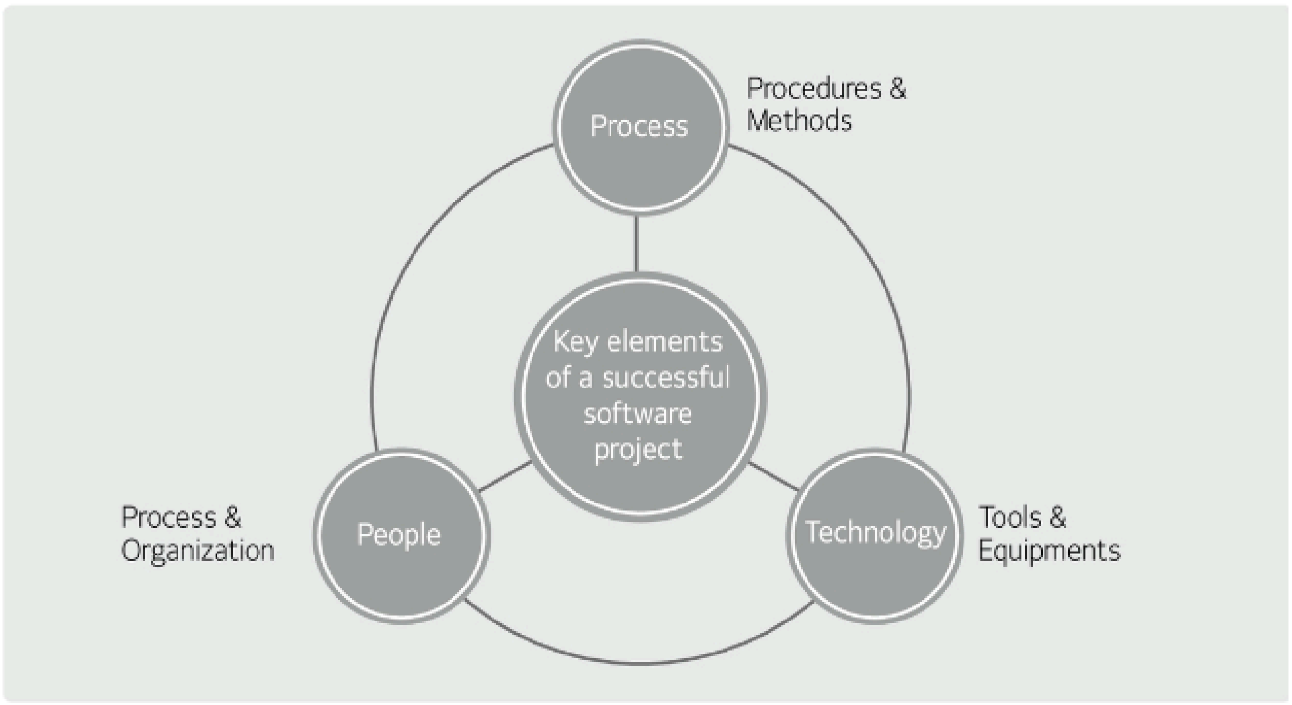
BACKGROUND TO AND PURPOSE OF SOFTWARE ENGINEERING

Applying software engineering technology to successfully develop software that becomes more multifunctional and bigger in scale, which provides technologies that support systematic management to resolve the expected difficulties throughout the entire process, ranging from equipment analysis to maintenance.

3 key elements of a successful Software

1. **Process** - Process of applying definition of systematic business methods and flow
2. **People** - Organization and people equipped with specialized knowledge
3. **Technology** - Infrastructure and technology required for efficient operation of the defined work method s and organizational personnel



1950s - concept of software engineering like hardware engineering began to be introduced to implement software development projects

1960s - software engineering was introduced because of "software crisis" due to the lack of experience, competence, and skilled engineers

1970s- software devs became scarce and demand for software increased,, adopted "coding first correction later" approach to solve this problem,

FOUR KEY ELEMENTS OF SOFTWARE ENGINEERING

- A discipline that studies the overall life cycle of software - such as development, operation, and maintenance - systematically, descriptively and quantitatively.
- These elements make it possible to produce top quality software and deliver it according to a given cost and schedule.

1. **Method**
    - project planning and estimation, system and software analysis, data structure, program structure, algorithm, coding, testing, and maintenance tasks
    - Centered on a specific language (OOP), or graphical notation, are introduced from time to time.
    - A series of evaluation standards for software quality
2. **Tool**
    - Automated or semi-automated method used to improve productivity/consistency when doing a task
    - Example tools are: requirements management tools, modeling tools, configuration management tools, change management tools
    - If these are integrated in such a way that the information created by one tool can be used by other tools then it is set as a system that supports software dev.
3. **Procedure**
    - Combines method and tools to develop software in a rational and timely manner
    - Defines the applied method, required deliverables, controls to guarantee quality and assist coordinate adjustment, and the sequence of milestones that enable software managers to evaluate progress
4. **People**
    - performed by employees,, depends more heavily on people
    - Practically impossible to summarize software dez w/out people

LIFECYCLE OF SOFTWARE DEVELOPMENT

- The entire process from understanding the user environment and problems to operation and maintenance.
- Composed of: **Feasibility Review**->**Development Planning->Requirement Analysis**->**Design**->**Implementation**->**Test**->**Operation**->**Maintenance**

Purpose

- Calculate the project costs and draw up a development plan, and to configure the basic framework
- Standardize the terms
- Manage a project

Selecting a software lifecycle

- Important activity for corporations to tailor the development process of a project
- Based on risk and uncertainty of system development and understanding it
- Selected model should be able to minimize the risks and uncertainties
- Waterfall, prototype, evolutionary, and incremental are the most represented life cycle models

Types of software lifecycle models

- V Model
    - clearly shows the activities that should be performed while implementing the project to project managers and developers
    - Helps customers who do not know much about this area to understand the principles of software dev
    - Ideal lifecycle model identifies and clarifies all system reqs
    - Can be used even when the reqs aren't clear.
    - Estimation, planning, and confirmation of a project are limited to only requirement analysis and identification phase
    - If reqs not clear then initial proj plan should specify the start/end dates expected by the customers

    - Example: The implementation of a standard communication protocol

Characteristic:

- Easy to apply; emphasizes project verification and validation (model explains very well)
- Effectively manage the start/end of dev activities and the project can be clearly defined

Prototyping

- Developing a system or part of a system to understand a system or to solve such issues as risks or uncertainties
- Make them understand what is needed and what should be developed
- Independent lifecycle model or can be applied to the dev phase of waterfall or v model

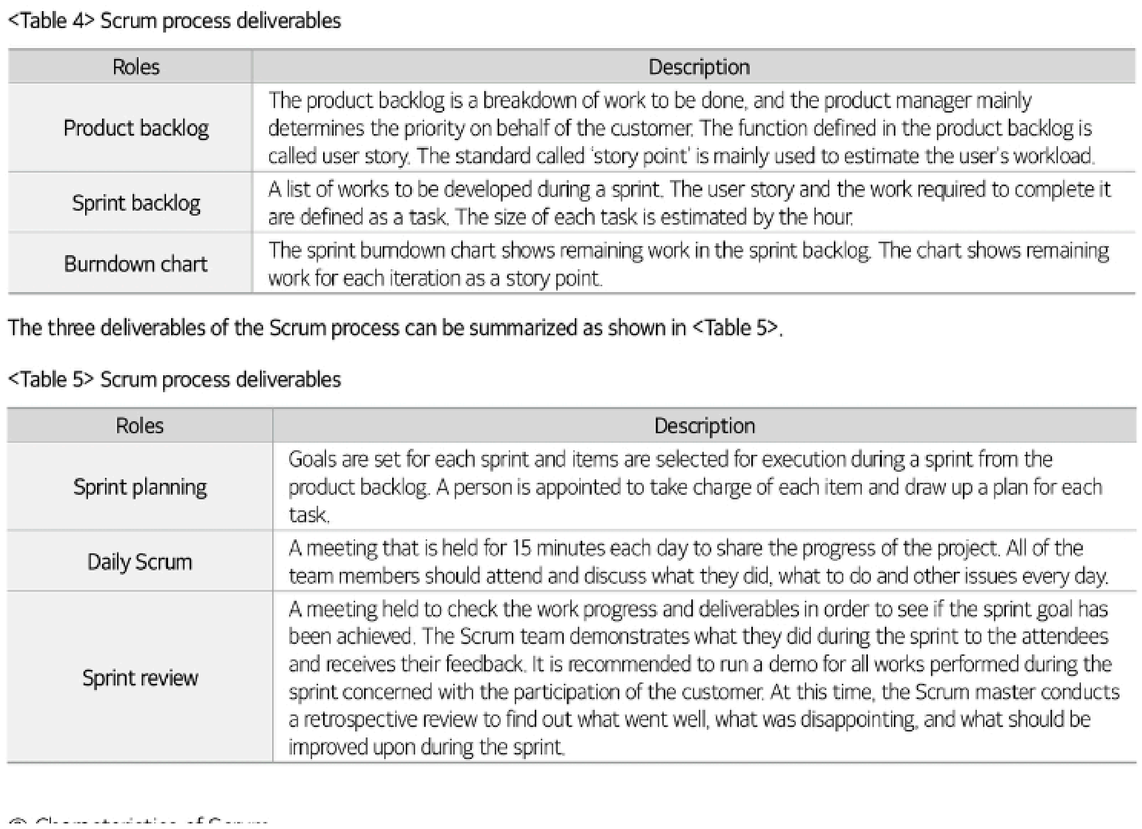
1. Throw away - rejected which means ilisan throw dat shit away
2. Revolutionary - guds and magamit na siya(feasible and usable)

Types of Software development Methodologies

- **Incremental** - mura siyag i++
    - Useful when the system dev time needs to be reduced
    - Useful when most of the reqs are defined, but improvements can be made as time goes by
    - Reduce risk of external interfaces in the early phase
- ➢ **Evolutionary model** - same with incremental but the dev phase for the entire system is reiterated several times. Each system provides all of the functions to the user
    - Changes are identified when a system is developed and used.(Kung mag change kay machange sad tanan functions etc)
    - Suitable when overall system specification is not certain or product should be continually improved
- **Spiral** - done together with incremental(iterative process; loop kunohay)
- **Agile -** Scrum and xp are generally used together**;** Xp used in early days**;** Lean is mainly used with scrum
    - comprehensively adopted to effectively respond to such rapid changes

Types of Agile Methodologies

- **Scrum , Ken Schwaber/ Jeff Sutherland**
    - Designed for project management
    - A representative type of empirical management technique based on estimation and adjustment
    - Modelled after “the new Product development Game” by hirotaka Takeuchi and Ikujuro Nonaka published in the Harvard Business Review in 1986
    - Scrum consist of Three fundamental units:
        - Product Owner - manages the creation product backlog
        - Scrum Master - removes any factors that disturb the team’s work
        - Scrum Team - identifies functions during a sprint using user stories (Composed of 5-9 members)
    - Scrum Process
        - Sprint: refers to repetitive development in units of 1-4 weeks
        - Three meetings: Daily Scrum, Sprint Plan, Sprint Review
        - Three deliverables: Product backlog, Sprint backlog Burndown Chart



<Table 4> Scrum process deliverables

Roles	Description
Product backlog	The product backlog is a breakdown of work to be done, and the product manager mainly determines the priority on behalf of the customer. The function defined in the product backlog is called user story. The standard called 'story point' is mainly used to estimate the user's workload.
Sprint backlog	A list of works to be developed during a sprint. The user story and the work required to complete it are defined as a task. The size of each task is estimated by the hour.
Burndown chart	The sprint burndown chart shows remaining work in the sprint backlog. The chart shows remaining work for each iteration as a story point.

The three deliverables of the Scrum process can be summarized as shown in <Table 5>.

<Table 5> Scrum process deliverables

Roles	Description
Sprint planning	Goals are set for each sprint and items are selected for execution during a sprint from the product backlog. A person is appointed to take charge of each item and draw up a plan for each task.
Daily Scrum	A meeting that is held for 15 minutes each day to share the progress of the project. All of the team members should attend and discuss what they did, what to do and other issues every day.
Sprint review	A meeting held to check the work progress and deliverables in order to see if the sprint goal has been achieved. The Scrum team demonstrates what they did during the sprint to the attendees and receives their feedback. It is recommended to run a demo for all works performed during the sprint concerned with the participation of the customer. At this time, the Scrum master conducts a retrospective review to find out what went well, what was disappointing, and what should be improved upon during the sprint.

- Characteristics of Scrum:
  - Transparency -enables the user to effectively understand the status of a project
  - Timeboxing - time is limited i.e daily scrum is limited to 15 mins each day
  - Communication - makes comms among team members smooth

- **EXtreme Programming(XP), Kent Bech/ Erich Gamma**
  - A lightweight development method for small/medium-sized dev orgs
  - Related to dev techniques such as **Test-Driven Development**(TTD)
  - Applied together with diff agile dev methods such as scrum, rather than a stand alone methodology
  - Composed of **value** and the **practice of achieving the value** when it is applied, a **principle** is needed to balance the two

        - Iterative development methodology

    - **Lean software development**, **Mary Popendic/Tom Poppendieck**
    - **Agile Unified Process(AUP)**, **Scott Ambler**
- **Waterfall**
    - Proceed With analysis design and implementation sequentially
    - Expensive and slow to implement
    - Less usable for general software developers
    - **1980s** - methods of boosting software dev were studied to efficiently develop software with increasing reusability

SOFTWARE DEVELOPMENT METHODOLOGY

- Defines each dev phase, the activities to perform in each phase, deliverables, verification procedure and completion criteria
- Defines standardized method and procedure and support tool

Necessity of SDM

- Improved dev productivity
- Effective proj management
- Providing a means of communication
- Assuring quality at a certain level

Software Development Phases

Requirement Analysis - hardest thing in software dev is deciding exactly what to develop

- The first step and the phase of understanding what the user needs
- Critical phase that can reduce the dev costs
- Conceptual Step

Design - the first step in physical realization

- Directly affects quality
- Stability deteriorates if not designed properly

Implementation - The goal is to program it in a way that the requirements can be satisfied based on the design specification (program should be written according to the description in the detailed design and user's guide)

- One of the most important task is to decide on a **coding standard** and write the code clearly based on that standard

Testing - refers to a series of processes to inspect and evaluate whether the system satisfies the prescribed reqs, etc, using manual or automated method

- the final step in assuring software qual; includes a quality eval

SCT Constraints

S - Scope

C - Cost

T - Time

SMART Criteria

S -Specific

M - Measurable

A - Attainable

R - Realistic

T - time bounded

Software Quality:

- **Usability**
  - Users can learn it fast and get their job done easily
- **Efficiency**
  - Does not waste resource such as CPU time and memory
- **Reliability**
  - It does what is required to do without failing
- **Maintainability**
  - It can be easily changed
- **Reusability**
  - Its parts can be used in other projects,, so reprogramming
    Is not needed

CRUD - Create, Read, Update ,Delete (HTTP) Post, Get ,Put ,Delete

API - Application Programming interface

Entity - a package that contains classes

-Student entity

-Course Entity

Repository - a package that contains interfaces

-StudentRepo

-CourseRepo

Service- Package that contains CRUD ,business rules, and logics

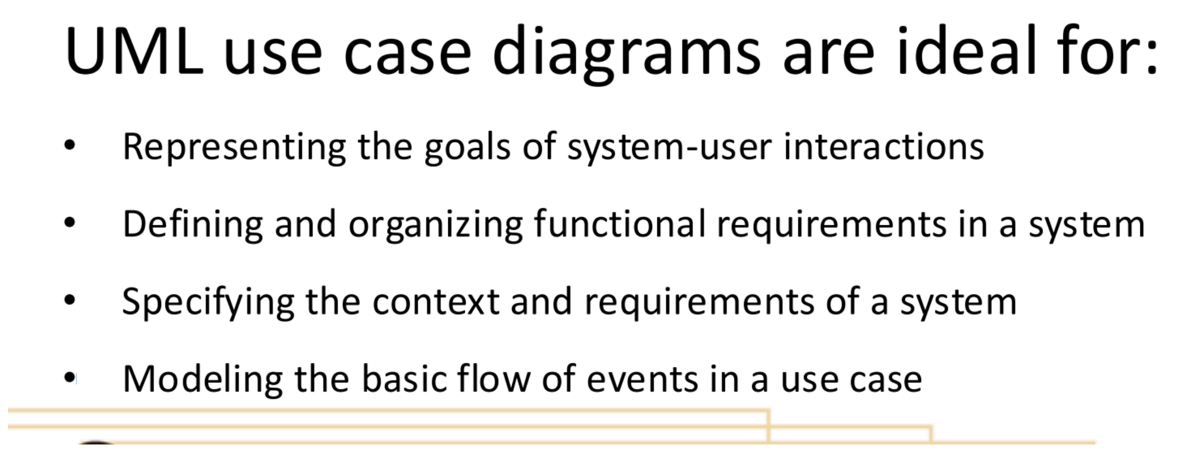
-StudentService

-CourseService

Controller - specific endpoints

-StudentController

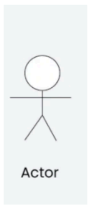
-CourseController



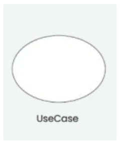
UML use case diagrams are ideal for:

- Representing the goals of system-user interactions
- Defining and organizing functional requirements in a system
- Specifying the context and requirements of a system
- Modeling the basic flow of events in a use case

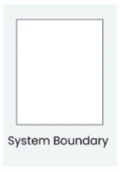
Use Case Diagram Components



Actors Actors are external entities that interact with the system. These can include users, other systems, or hardware devices. In the context of a Use Case Diagram, actors initiate use cases and receive the outcomes. Proper identification and understanding of actors are crucial for accurately modeling system behavior.



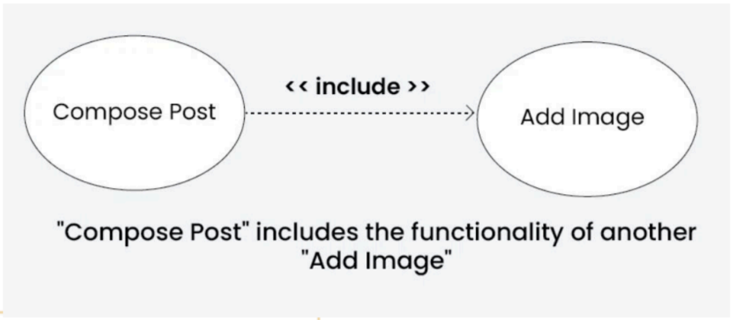
Use cases are like scenes in the play. They represent specific things your system can do. In the online shopping system, examples of use cases could be “Place Order,” “Track Delivery,” or “Update Product Information”. Use cases are represented by ovals.



System boundary defines what is inside the system and what is outside. The boundary helps to establish a clear distinction between the elements that are part of the system and those that are external to it. The system boundary is typically represented by a rectangular box that surrounds all the use cases of the system. Purpose of System Boundary: Scope Definition and Focus on Relevance.

Include Relationship

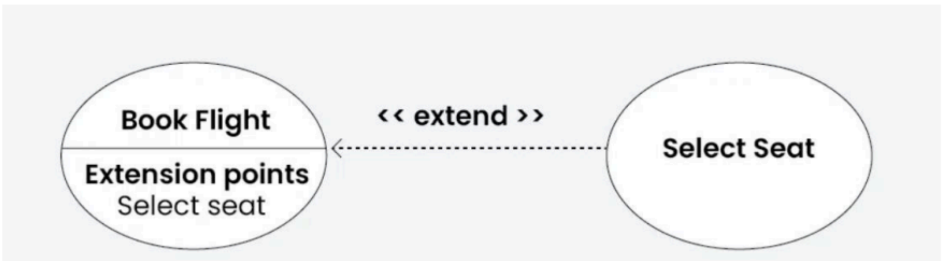
The Include Relationship indicates that a use case includes the functionality of another use case. It is denoted by a dashed arrow pointing from the including use case to the included use case. This relationship promotes modular and reusable design.



"Compose Post" includes the functionality of another "Add Image"

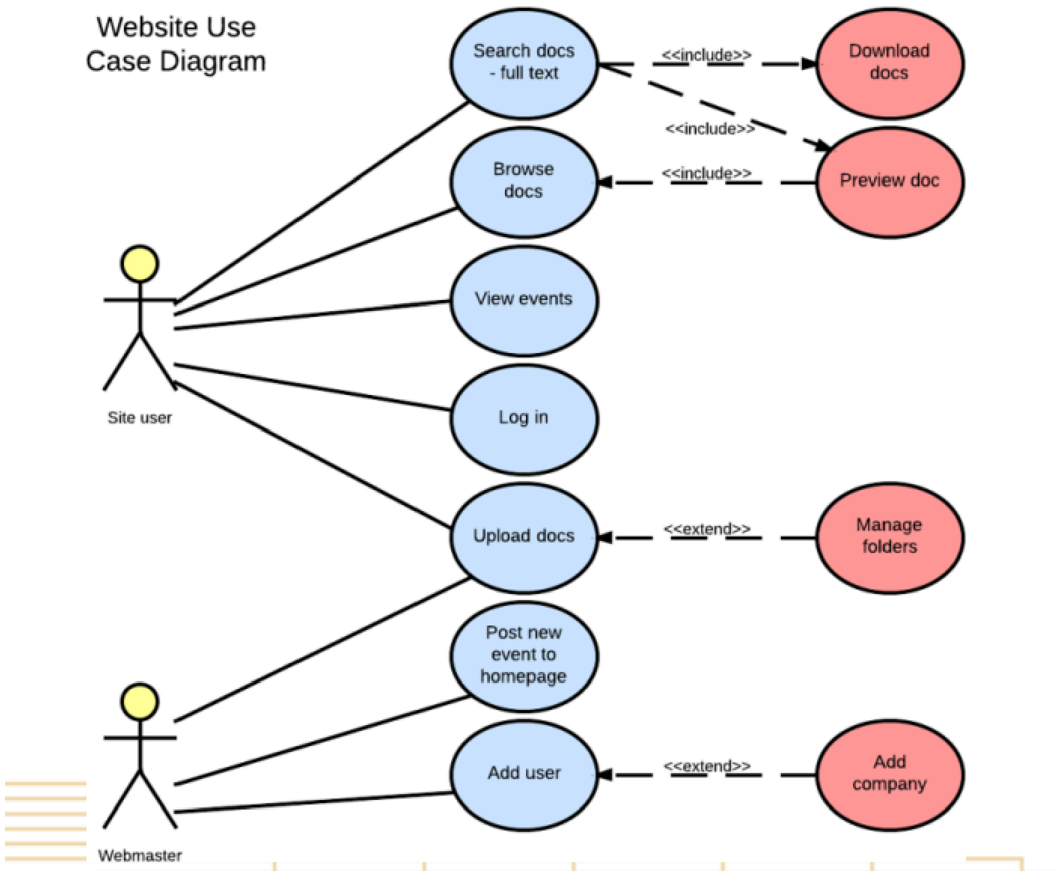
Extend Relationship

The Extend Relationship illustrates that a use case can be extended by another use case under specific conditions. It is represented by a dashed arrow with the keyword “extend.” This relationship is useful for handling optional or exceptional behavior.



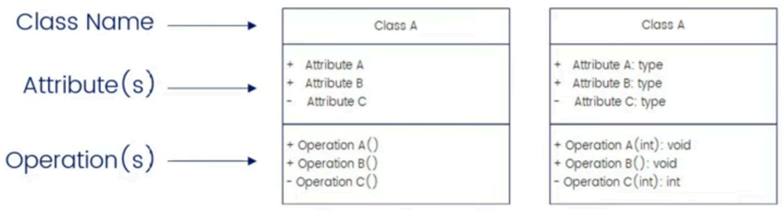
The “Select Seat” use case may extend the “Book Flight” use case when the user wants to choose a specific seat

Use Case Diagram Example 3



Class Notation

A class symbolize a concept which encapsulates state (attributes) and behavior (operations).



Class without signature Class with signature

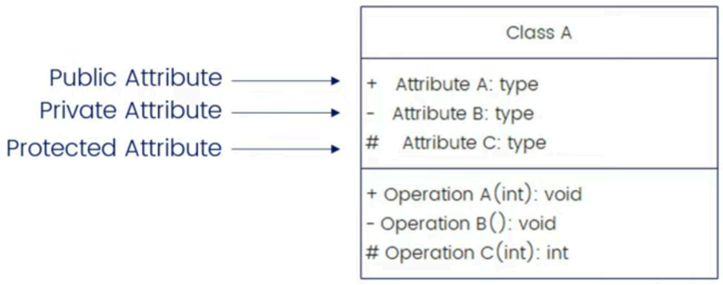
Class Name: The name of the class is always shown in the first section. It is the mandatory information.

Class Attributes:
• Attributes are shown in the second section.
• The attribute type is shown after the colon.
• Attributes are values that define a class.

Class Operations (Methods):
• Operations are shown in the third section. They are services the class provides.
• The return type of a method is shown after the colon at the end of the method signature.
• The return type of method parameters are shown after the colon following the parameter name. Operations map onto class methods in code.

Class Visibility

The symbols before an attribute and operation name in a class denote the visibility of the attribute and operation. Visibility symbols are used to determine the accessibility of the information contained in classes.



Public (+)

Private (-)

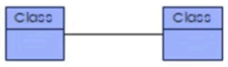
Protected (#)

Package (~)

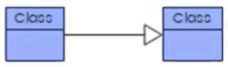
Derived (/)

Static (underlined)

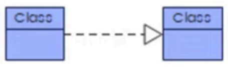
Relationships between classes



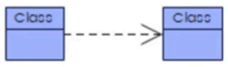
Bidirectional Association - A bilateral association is represented by a straight line connecting two classes. It simply demonstrates that the classes are aware of their relationship with each other.



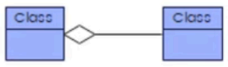
Inheritance - Indicate a “child-parent” relationship between classes. The child class is a specialized, sub-class of the parent.



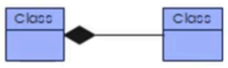
Realization - One class implements the behavior specified by another class.



Dependency - As the name suggests, one class depends on another. A dashed arrow shows this.



Aggregation - This represents a unilateral relationship between classes. One class is part of, or subordinate to, another. In this instance, the child and parent classes can exist independently.



Composition - It is a form of aggregation where one class is dependent on another. One class is a part of the other. In this instance, the child classes and parent classes cannot exist independently.