# Unified Modeling Language (UML)

Friday, December 14, 2018 9:51 AM

**What is UML and why use UML?**

- UML is a modern approach to modeling and documenting software.
- It is based on **diagrammatic representations** of software components.
- UML has been used as a general-purpose modeling language in the field of software engineering.
- They provide both a more standardized way of modeling workflows as well as a wider range of features to improve readability and efficacy.

**Sketch:**

UML diagrams, in this case, are used to communicate different aspects and characteristics of a system. However, this is **only a top-level view of the system** and will **most probably not include all the necessary details** to execute the project until the very end.

- *Forward Design* – The design of the sketch is done before coding the application. This is done to get a better view of the system or workflow that you are trying to create. Many design issues or flaws can be revealed, thus improving the overall project health and well-being.
- *Backward Design* – After writing the code, the UML diagrams are drawn as a form of documentation for the different activities, roles, actors, and workflows.

UML is not a stand-alone programming language like Java, C++ or Python, but it can be converted into a pseudo programming language. The whole system will be documented in different diagrams and can be translated directly into code. This approach is only beneficial if the time to create diagrams is less when compared to code development time.

**Types of UML Diagrams:**

There are several types of UML diagrams and each one of them serves a different purpose regardless of whether it is designed before the implementation or after (for documentation).

Two broad categories are:

1. Behavioral UML diagram (Describe the behavior of the system, actor and components)
2. Structural UML diagram (Analyze and depict the structure of a system or process)

**Behavioral UML Diagram**

1. **Activity Diagram**
2. **Use Case Diagram**
3. **Interaction Overview Diagram**
4. **Timing Diagram**
5. **State Machine Diagram**
6. **Communication Diagram**
7. **Sequence Diagram**

**Structural UML Diagram**

1. **Class Diagram**
2. **Object Diagram**
3. **Component Diagram**
4. **Composite Structure Diagram**
5. **Deployment Diagram**
6. **Package Diagram**
7. **Profile Diagram**

The diagrams that are commonly used by the software engineers are **Use Case diagrams, Class diagrams**

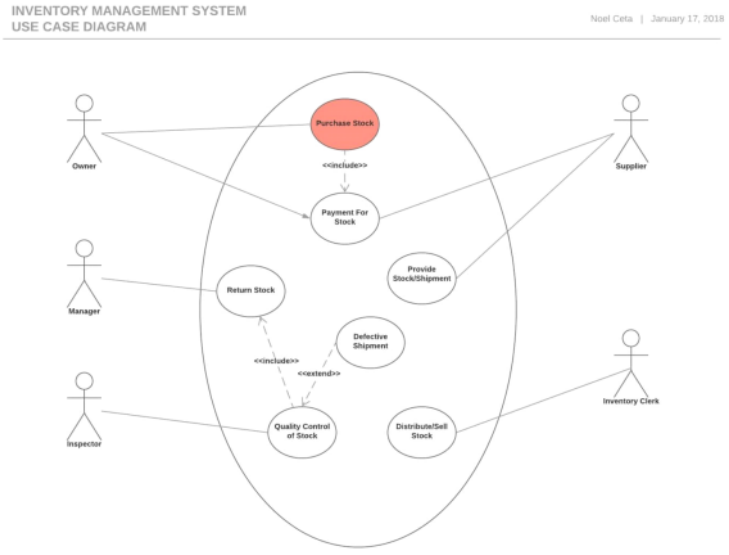
**and Sequence Diagrams**. So this article will concentrate these diagrams.

**Use Case Diagram:**

Use Case diagrams are used to analyze the system's high level requirements. There are three main components present in a Use Case Diagram.

1. Functional requirements: represented as use cases; a verb describing an action
2. Actors : they interact with the system; an actor can be a human being, an organization or an internal or external application
3. Relationships between actors and use cases - represented using straight arrows

The following example depicts the use case UML diagram or an inventory management system. Actors in this example are the owner, the supplier, the manager, the inventory clerk and the inventory inspector. The eclipse container in the middle represents the actions being done by the actors such as purchasing and paying for the stock, checking stock quality, returning the stock or distributing it. From this example we can see that the Use Case UML diagrams are good in showing the dynamic behavior of the actors within the system, by simplifying the view of the system but not giving details about the implementation of the actions/ behaviors.

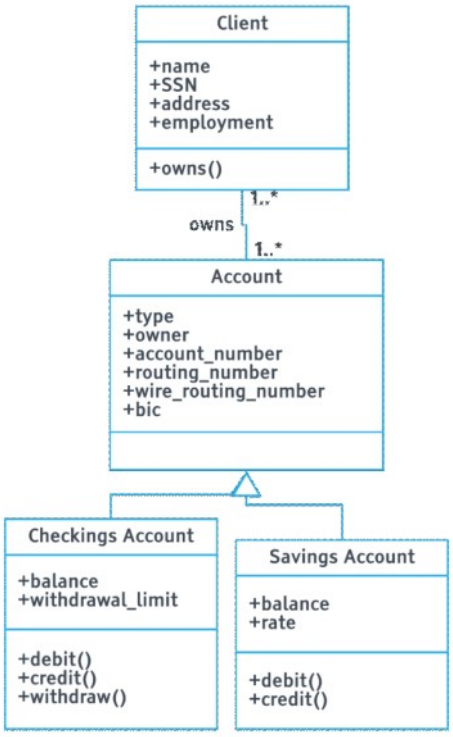



**Class Diagram:**

Class diagram is the most common type of diagram used in software documentation. This is due to the fact that most software being created now-a-days is based on the object oriented paradigm. This is because OOP is based on classes and the relations between them.

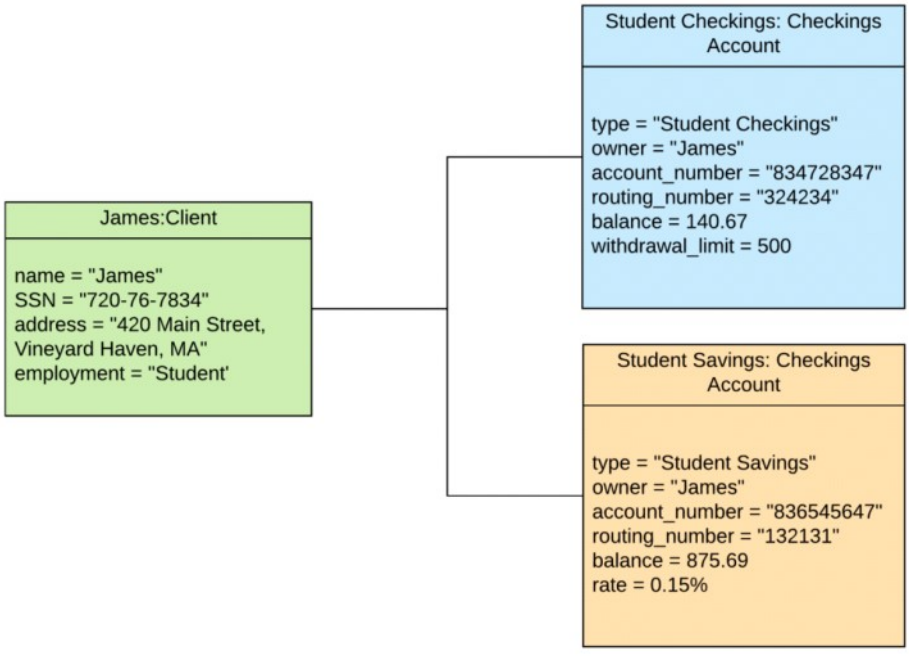
Class diagrams contain classes, alongside with their attributes and their behaviors. So each class has 3 fields: the class name on the top, the class attributes right below the name, the class operations / behaviors at the bottom. The relations between the different classes which is denoted by a connecting line makes up a class diagram.

Example (Class Diagram below):
From the diagram we can infer that the classes Checkings Account and Savings Account both inherit from the more general class, 'Account'. This inheritance is shown using the blank-headed arrow.




**Object Diagram:**

Object UML diagrams help software developers check whether the generic abstract structure that they have created (class diagram), represents a viable structure when put into practice, i.e: when the objects of a class are instantiated. It is considered as a secondary level of accuracy checking.



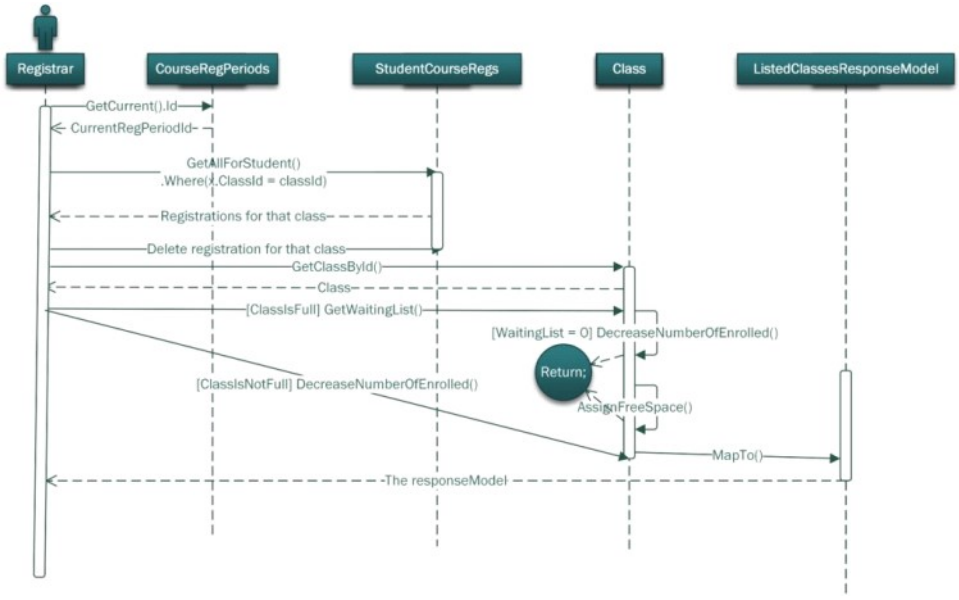

The object diagram mentioned above is based on the class diagram shown earlier. As it can be easily observed, James is an instance of the more generic client class with the same attributes and the values for those attributes. Likewise Student Checkings and Student Savings are the instances of the Checkings Account and Savings Account classes.

**Sequence UML Diagrams:**

These diagrams describe the sequence of messages and interactions that happen between actors and objects. Actors or objects can be active only when needed or when another object wants to communicate with them. All the communication is represented in a chronological manner.

These are more specifically used in software development to represent the architecture of the system and how the different components are interconnected (do not exactly provide information about how they behave or communicate).

The diagram below gives an example of a sequence diagram, which depicts a course registration system.




Reference: https://tallyfy.com/uml-diagram/