**Unit 3 - Class Notes**

# UML Diagram

Unified Modeling Language (or UML) is a visual tool to document artifacts of software system. It is a tool that can be used to design software system which will be build using Object Oriented Programming (OOP) language. It is language independent. This means that a system designed with UML can be implemented in any OOP language (like SmallTalk, Java, C , etc.). UML was developed by Grady Booch and Jim Rumbaugh of Rational Software. There are six types of UML diagrams:

* **Use Case Diagram** describes the functionalities of the system and their connections with the users. It contains actors which represent users of the system. Users may be human beings or other software systems. It also contains use cases, representing the functionalities or services provided by the system to its users. It also depicts how users can interact with the system.
* **Class Diagram** describes the static structure of a system. Classes in class diagram represent entities that are part of the system. Attributes (or properties) of the classes and their functionalities are also included in the class diagram. Class diagram also includes relationship that relates two or more classes.
* **Interaction Diagram** can be subdivided into two types:
  + Sequence Diagram describes interaction among classes. It models the sequence in which messages are exchanged between classes to accomplish some desired behavior or task.
  + Collaboration Diagram displays interaction organized around a class and its links to other classes.
* **State Diagram** (or Object Diagram) describes the static structure of objects at a particular time. It contains objects that represent a particular entity at a particular time.
* **Activity Diagram** is a special state diagram where most of the states are action states and most of the transitions are triggered by completion of actions in the source state.
* **Physical Diagram** can be subdivided into two types as follows:
  + Component Diagram displays the high level structure of the system code.
  + Deployment Diagram displays the system configuration of the processing elements and software components at run time.

We will only cover use case diagrams, class diagrams, and sequence diagrams in this course.

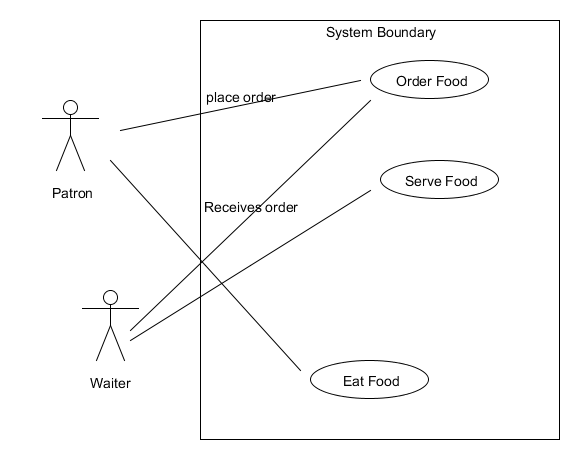
# USE CASES

Use cases are used for identifying the behaviors or functionalities of a system. Use cases can be documented verbally or using UML use case diagrams. It is considered to be the best techniques for capturing user requirements and can be included in the Software Requirements Specification (SRS). They can also be documented using Use Case document. Use cases must be readable by non-engineers and can be used to engage stakeholders and end users in the initial stages of project.

A use case includes actor(s) and the system. Each system will have certain functionalities or behaviors. One or more actors will be associated with each functionality in the system. The association between an actor and a functionality is shown with a straight line and may include a description of the association. A use case describes a potential sequence of interactions between the actor(s) and the system. A use case diagram may contain multiple use cases.

Actors are external to the system being developed and interact directly with the system. Actors may identify human interfaces (like end users), external devices (example: bar code scanner), or another software system (such as billing system interacting with sales system). An actor can also be a “Time” which represents a periodic or a scheduled event.

Example:



This is the use case for a restaurant (system) and shows how the system will be used by actors: patron and waiter. We can conclude the following from this diagram:

* Functionality will be executed in the following order: Order Food, Serve Food, and Eat Food (functionality order is from top to bottom within the system boundary).
* A patron can Order Food by placing order and Eat Food but can't Serve Food. A Waiter can Order Food (by receiving order) and Serve Food but can't eat food.

We can add more actors to the above use case diagram as needed. For example we can add Chef, Cashier, and Bar Tender to the above use case diagram showing the functionalities associated with them. The above use case diagram contains three use cases: Order Food, Serve Food, and Eat Food. We can add other uses cases to the above diagram. Some of the other use cases include: Cook Food, Generate Bill, and Pay Bill.

Another good example of use case can be found at: [http://www.math-](http://www.math-cs.gordon.edu/courses/cs211/ATMExample/UseCases.html) [cs.gordon.edu/courses/cs211/ATMExample/UseCases.html](http://www.math-cs.gordon.edu/courses/cs211/ATMExample/UseCases.html)

Also see: <http://www.agilemodeling.com/artifacts/useCaseDiagram.htm>

Use cases are documented using a table which will generally contain the following components: Use Case ID, Use Case Name, Actors, Description, Preconditions, Postconditions, Normal Flow, Exceptions, Priority, Special Requirements, and Assumptions.

Example:

|  |  |
| --- | --- |
| Use Case ID: | 1 |
| Use Case Name: | Order Food |
| Actors: | Patron, Waiter |
| Description: | A Patron places an order for food using a Waiter. |
| Preconditions: | 1. Patron is in the restaurant. 2. Waiter is available to take the order. |
| Postconditions: | 1. Food is served to the Patron. |
| Normal Flow: | * 1. **Order Food**      1. Patron arrives in a restaurant      2. Patron views menu      3. Patron selects one or more items from menu      4. Waiter notes the item selected by Patron      5. Patron indicates that Food Order is complete |

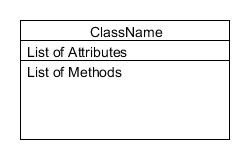
|  |  |
| --- | --- |
|  | 1. Waiter reads back the order to confirm selected items 2. Patron confirms the list of items |
| Exceptions: | * + - 1. Patron arrives after restaurant closing time.       2. Restaurant can't fulfill specific item ordered by Patron       3. Patron cancels the order       4. Restaurant is full       5. Waiter is not available to take order. |
| Priority: | High |
| Special Requirements: | 1. Patron shall be able to cancel the meal order at any time prior to  being served. |
| Assumptions: | 1. Assume that 30 percent of Patrons will order the daily special  (source: previous six months of cafeteria data). |

A good example of Use Case document is given in the document COS\_use\_cases.doc

# Class Diagrams

Class Diagram describes the static structure of a system. Classes in a class diagram represent entities that are part of the system. Attributes (or properties) of the classes and their functionalities are also included in the class diagram. Class diagram also includes relationship that relates two or more classes. Relationships between classes are based on common characteristics or features. There are five types of relationship that may exist between classes: Association, Aggregation (or collection), Composition, Inheritance, and Dependency.

A class in a class diagram is shown with a rectangle box which is divided into three parts as follows:



The top part contains the name of the class. The middle part contains the list of attributes, and the bottom part contains list of methods or functionalities.

Generally name of a class begins with uppercase letter and uses camel casing (each word in the name starts with upper case letter and all other letters are in lower case). Example: Student, Teacher, Course, TimeMachine, GraduateStudent, UndergraduateStudent, SalesTaxCalculator, etc.

Attributes are listed (one per line) using the following syntax:

visibility modifier attribute name : attribute type

Visibility modifier can be private (-), public ( ), or protected (#). Attributes uses the private visibility modifier to hide implementation details (encapsulation). Name starts with lower case letter and uses camel casing (example: date, studentName, teacherName, courseNumber, courseName, dateOfBirth, etc). Attribute type represent the type of data that can be stored in that attribute. A short list of possible types is:

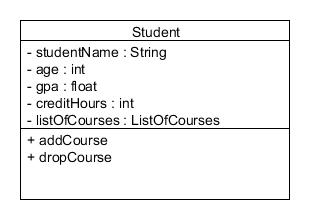
* int (integer number)
* float (floating point number)
* string (alphanumerical data and is enclosed in " ")
* character (one single alphanumerical character and is enclosed in ' ')
* boolean (true or false)
* objectType (a class attribute can be another class)

List of methods will contain the list of functionalities implemented for the class. For our course we will use the following syntax:

visibility modifier method name

Method name starts with lower case letter and uses camel casing. Example: registerForClass, dropClass, eat, study, assignGrade, etc.

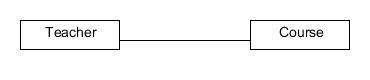
Example:



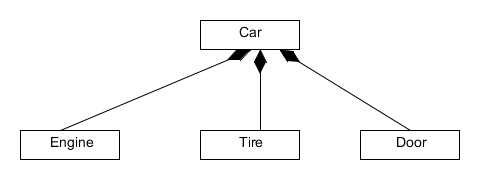
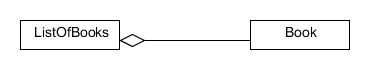
This is the class diagram for Student class. Student class contains five attributes: studentName, age, gpa, creditHours, and listOfCourses. listOfCourses is of type ListOfCourses. Note: listOfCourses with lower case l is attributeName and ListOfCourses with upper case L is class name.

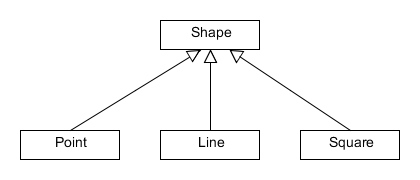
Relationship between classes is shown by connecting the classes with different types of lines. For brevity, I have left out list of attributes and methods in the following examples.

Example of Association:



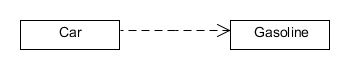
In this example a teacher is associated with a course. Example of Aggregation (or collection):

ListOfBooks aggregate Book class or ListOfBooks is a collection of Book class. Example of Composition:

Car is composed (or made up) of Engine, Tire, Door, etc Example of Inheritance:

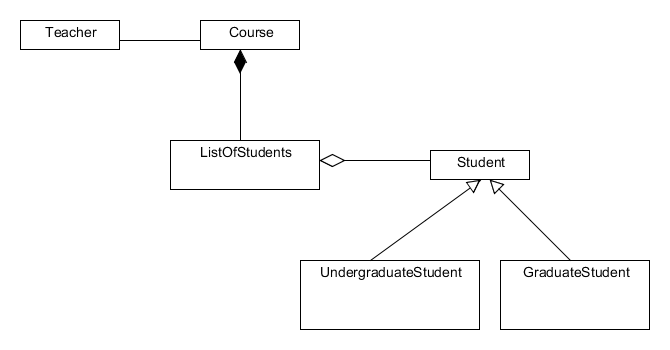
Shape is the general class (or super class) and Point, Line, and Square are specialized class (or sub class). Point, Line, and Square classes inherit from the Shape class.

Example of Dependency:



Car depends on Gasoline or Car uses Gasoline. The following is an example of multiple objects involved in different relationships:

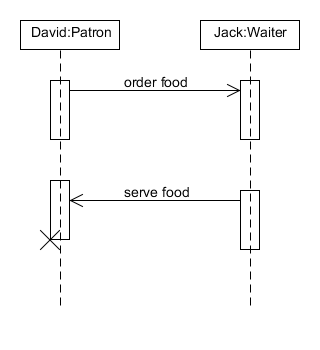
Example of Class Diagram



# Sequence Diagram

Sequence diagram describes interaction among classes and objects. It models the sequence in which messages are exchanged between classes and objects to accomplish some desired behavior or task. Sequence diagram includes objects and classes, lifeline of objects and classes, and message being passed between objects.

Example of Sequence Diagram



This example shows interaction between David (which is an object of class Patron) and Jack (which is an object of class Waiter). The interaction starts with David ordering food by passing message to Jack. Message will include a list of food items that David wants to order. Jack serves the food when it is ready. This diagram also shows that message called "order food" will be passed from David (Patron) to Jack (Waiter) and then the message called "serve food" will be passed from Jack (Waiter) to David (Patron). It does not include information about what David will do with the food as that process is internal to David (Patron) object. The X mark indicates David's lifeline. This sequence diagram can be extended by adding objects of other classes like Chef, BarTender, Cashier, etc.

# Additional Links

See the following web sites for more information:

Practical UML: A Hands-On Introduction for Developers: <http://edn.embarcadero.com/article/31863>

Object Management Group – UML (this is the official site of Object Management Group which defines standards for UML): <http://www.uml.org/>

There are number of tools available that can be used to create UML diagrams. One such tool is UMLET. It is an open source software and can be downloaded from the following website: <http://www.umlet.com/>(Click on the "Download" text button to download the UMLET. A video tutorial is also available on the web site.)

Additionally, almost all of the major Integrated Development Environments (IDEs) also have UML tools that can be used to design software system. These tools can generate part of the software code

based on the UML diagram. These tools can also be used to reverse engineer and create UML diagram based on the software source code. See the following web sites for some tutorial:

Creating UML 2 Diagrams with Eclipse UML2 Tools – Tutorial: <http://www.vogella.de/articles/UML/article.html>

Introduction to UML 2 Class Diagrams: <http://www.agilemodeling.com/artifacts/classDiagram.htm>