**Structural Modeling: Class Diagramming**

[PPT 3]

A class diagram in the [Unified Modeling Language (UML)](https://en.wikipedia.org/wiki/Unified_Modeling_Language) is **a type of static structure diagram** that describes the structure of a system by showing the system's classes, their attributes, the operations (or methods), and the relationships among objects. The purpose of Class Diagrams are to shows static structure of classifiers in a system. The diagram provides basic notation for other structure diagrams prescribed by UML, the diagram is helpful for developers and other team members the understand the main objects should be present in the design, and by using the class diagram, then the Business Analysts can use the class diagrams to model systems from business perspective. A UML class diagram is usually made up of a set of classes and a set of relationships between classes.

[PPT4]

An *object* is an instantiation of a class. In other words, an object is a person, place, event, or thing about which we want to capture information. If we were building a sales system for a car dealer, classes might include vehicle, customer, and offer. The specific customers like Mary Jane, Billy Hope, and Nancy Jones are considered instances, or objects, of the customer class. Each object has *attributes* that describe information about the object, such as a customer’s name, address, e-mail, and phone number. The state of an object is defined by the value of its attributes and its relationships with other objects at a particular point in time.

[PPT5]

A *class* is the general template we use to define and create specific *instances*, or objects. Every object is associated with a class. For example, all of the objects that capture information about patients could fall into a class called Patient, because there are attributes (e.g., names, addresses, and birth dates) and methods (e.g., insert new instances, maintain information, and delete entries) that all patients share.

A class is a description of a group of objects in the systems, all with similar roles in the system. An object is any person, place, thing, concept, event, screen, or report applicable to your system. Objects both know things (they have attributes) and they do things (they have methods). A class is a representation of an object and, in many ways, it is simply a template from which objects are created. Classes form the main building blocks of an object-oriented application. Although thousands of students attend the university, you would only model one class, called *Student*, which would represent the entire collection of students.

Class is consists of:

* Structural features (attributes) define what objects of the class "know"
  + It represent the state of an object of the class
  + Are descriptions of the structural or static features of a class
* Behavioral features (operations) define what objects of the class "can do"
  + Define the way in which objects may interact
  + Operations are descriptions of behavioral or dynamic features of a class

[PPT 6]

In modelling, we have Class Notation in the diagram. Classes are typically modeled as rectangles with three sections: the top section for the name of the class, the middle section for the attributes of the class, and the bottom section for the methods of the class. The initial classes of your model can be identified in the same manner as they are when you are using other previous technique, as will the initial responsibilities (its attributes and methods). Attributes are the information stored about an object (or at least information temporarily maintained about an object), while methods are the things that an object or class do. For example, students have student numbers, names, addresses, and phone numbers. Those are all examples of the attributes of a student. Students also enroll in courses, drop courses, and request transcripts, those are all examples of the things a student does, which get implemented (coded) as methods. You should think of methods as the object-oriented equivalent of functions and procedures. In the diagram, a class notation consists of three parts:

1. Class Name
   * The name of the class appears in the first partition.
2. Class Attributes
   * Attributes are shown in the second partition.
   * The attribute type is shown after the colon.
   * Attributes map onto member variables (data members) in code.
3. Class Operations (Methods)
   * Are shown in the third partition. They are services that 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

[PPT 7] Class Relationships

A class may be involved in one or more relationships with other classes. Classes are interrelated to each other in specific ways. In particular, relationships in class diagrams include different types of logical connections. Primary purpose of the class diagram is to show the associations, or relationships, that classes have with one another. These are depicted on the diagram by lines drawn between classes. Associations are maintained by *references*, which are similar to pointers and maintained internally by the system (unlike in the relational models where relationships are maintained by foreign and primary keys).

When multiple classes share an association (or a class shares an association with itself), a line is drawn and labeled with either the name of the association or the roles that the classes play in the association. For example, the two classes *customer* and *offer* are associated with one another whenever a customer makes an offer. Thus, a line labeled *makes* connects *customer* and *offer*, representing exactly how the two classes are associated with each other. We can draw a small solid triangle beside of the name of the association. The triangle allows a direction to be associated with the name of the association. Association includes a triangle notation, indicating that the association is to be read as *customer makes offer.* Inclusion of the triangle simply increases the readability of the diagram. Associations also have *multiplicity*, which shows how an instance of an object can be associated with other instances. Numbers are placed on the association path to denote the minimum and maximum instances that can be related through the association in the format *minimum number maximum number.* Another notation*,Generalization* shows that one class (subclass) inherits from another class (superclass), meaning that the properties and operations of the superclass are also valid for objects of the subclass.

The following are such types of logical connections that are possible in UML:

* [Association](https://creately.com/blog/diagrams/class-diagram-relationships/#Association)
* Multiplicity
* [Aggregation](https://creately.com/blog/diagrams/class-diagram-relationships/#Aggregation)
* [Composition](https://creately.com/blog/diagrams/class-diagram-relationships/#Composition)
* [Realization](https://creately.com/blog/diagrams/class-diagram-relationships/#Realization)
* [Inheritance](https://creately.com/blog/diagrams/class-diagram-relationships/#Inheritance)

[PPT 8]

We will talk them in more detail in your class. Let say, Class Diagrams not only depict entities with their attributes, they also have the power to depict the domains these attributes belong to along with the relationships and other participation constraints, which is helpful when designing physical level schemas. Class Diagrams can also be used to show behavior, in the case of databases, this feature can be used to show low level details about the stored procedures and functions that would have access to the entities or perform operations like insert, update and delete. This means a level of detail which is normally necessary at the physical level of database design. Therefore we could argue that the UML class diagram is more of a logical model (that is implementation specific). ERDs on the other hand, are at a conceptual level of database design and at a higher level of abstraction dealing only with the main items and their relationships. Also, the ER model maps well to a relational database and the mapping procedures are well established.

References:

1. <https://en.wikipedia.org/wiki/Reusability>, (2018)
2. Kenneth S. Rubin , (2014), A Practical Guide To The Most Popular Agile Process, ISBN-10: 0-13-704329-5, Edwards Brothers Malloy, Ann Arbor, Michigan.
3. Nishadha, (2018), UML Class Diagram Relationships in Explained with Examples, <https://creately.com/blog/diagrams/class-diagram-relationships>
4. Roberta M. Roth, Alan Dennis, Barbara H. Wixom, (2013), Systems Analysis and Design, 5th edition, ISBN: 978-1-118-09374-0, John Wiley& Sons, Inc, Singapore
5. Sametinger, Johannes(2018), Software Engineering with Reusable Components, [https://www.springer.com /us/book/9783540626954](https://www.springer.com/us/book/9783540626954) (book preview)
6. Timothy C Lethbridge, Robert Laganiere, (2005), Object-Oriented Software Engineering, ISBN 0-07-70109082, McGraw-Hill Education, Glasgow, UK