Design Model

**Topics**

1. [Object-Oriented Class Diagram Design](https://leocontent.umgc.edu/content/umuc/tus/cmis/cmis330/2232/modules/m4-module-4/week-4-study.html?ou=941453#Topic1)
2. [Example Class Design](https://leocontent.umgc.edu/content/umuc/tus/cmis/cmis330/2232/modules/m4-module-4/week-4-study.html?ou=941453#Topic2)

The transition from analysis to design can be difficult. Typically, there are multiple implementations for the same requirement. Which is the best implementation? How will the separate implementations for each of the software requirements interact with each other? How will the overall system behave? These are difficult questions for the software engineer to answer. Some of them cannot be fully understood until the software and the system are complete.

Models used during the analysis phase build the foundation for transitioning elements to the design model. Once the analysis phase is complete, the next task for the software engineer is to represent the software requirements by providing the details of the data structures, architecture, human-computer interface, and all the components necessary to implement a working system. The analysis model provides the foundation for the design model. The scenario-based, flow-oriented, class-based, and behavioral elements are translated into the data and class architecture, interface, and component-level designs that comprise the elements of the design model.

Object-Oriented Class Diagram Design

In requirements analysis we applied processes the object-oriented approach. Now we can extend these processes in the design phase. Such a relationship is to be expected because OO development, typically, is iterative and involves many cycles around the requirements-design-code-test loop. In OO design, the analysis models that were created are evolved into design models.

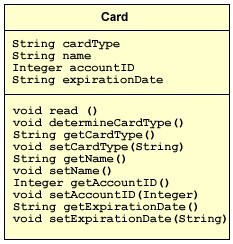
In OO design the software engineer refines the classes and objects that were defined in the class diagrams during the analysis phase. It may be necessary for the software engineer to add more detail to the classes and create new classes to provide the desired functionality. Once all the classes are defined, the relationship between the classes is modeled to show the hierarchy structure and to identify the superclasses and subclasses.

Identifying the set of classes for instantiating all the objects needed in a software design is not an easy task. Considerable research has been conducted on how to identify object classes. Some of the best practices for identifying object classes are:

* Perform an analysis of the grammatical content of the textual descriptions (problem statements, use cases) for the proposed system. Identify the nouns in the descriptions that are good candidates for objects and attributes that can be used in defining classes.
* Identify the tangible entities and any data storage structure needed in the application domain.
* Review the behavioral diagrams that were created during analysis and identify any participant who plays an important role in initiating or participating in the behavior.
* Review the scenario diagrams created during analysis to identify data objects, attributes, and participants.

**Example Class Design**

When designing a class, the software engineer needs to specify the interface, the signature, and semantics of the services provided by the object or groups of objects. These descriptions can be included in the class diagram used to model the design, as shown in Figure 1.

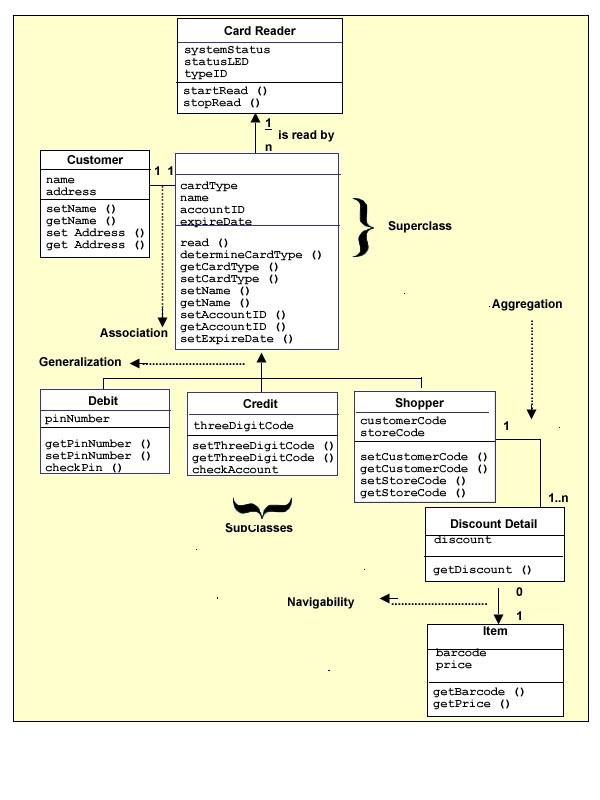


**Figure 1  
Example Class Design**

There are three types of relationships that can be modeled using a class diagram; we have listed them below:

* **Association:** depicts a necessary relationship with another class in order to perform its work
* **Aggregation:** links a class with a collection
* **Generalization:** links inheritance of a class with its superclass

In Figure 2, the extended class diagram for a self-service checkout system shows the possible relationships that can exist between classes.



**Figure 2**

Additionally, as seen in Figure 2, associations can have a **navigability arrow** to indicate the direction and the ownership of the association. For example, a frequent shopper card can be associated with many discount detail objects. However, a discount detail can be queried for an item, but an item cannot query for a discount detail. The multiplicity of an association that shows the possible instances of the classes.