An Object-Oriented analysis model is made of a use case model and a class diagram model. We will discuss both models in this study.

**1. Use Case Model**

This section will elaborate on a technique that can be used to specify and model the expected behavior for software. Software engineers refer to this behavioral model as the *use case model*. The use case model defines use cases and flow of events (scenarios) that capture a set of functional requirements and organize the software requirements in an easy-to-manage way. Following the use case model during requirements development can ease the transition from the analysis to the design phase.

Our discussion on use cases will begin with a brief history. The use case was introduced as part of an object-oriented development methodology originated by Ivar Jacobson and described in his book called *Object-Oriented Software Engineering: A Use Case Driven Approach*. Researchers Larry Constantine and others have extended this concept into a general technique for requirements analysis and user interface design. The Unified Modeling Language provides support for modeling a use case.

A use case is defined as sequences of actions a system performs that yield an observable result of value to a particular actor.  A sequence of actions refers to a "set of functions" or an "algorithmic procedure" that is initiated and used by an **actor**. The use case describes the input to the system and the response that is necessary to satisfy the function that produces the intended output.

A use case is best summarized as having the following characteristics:

* provides the foundation for deriving scenarios in which an actor interacts with the system
* encompasses several tasks with a common goal
* achieves a specific goal or accomplishes a particular task or function
* describes an action in terminology relative to the actor's domain and not terminology relative to the computer
* focuses on the simplicity of the use or action being performed without regard to how it is implemented in the software or alternate ways of accomplishing the same task
* provides a means to derive detailed requirements for accomplishing the objective for each scenario by using a software application

When forming a use case, you should think of all the various external entities that interact with the system and the different ways in which they may possibly use the system. Specifying use cases is part of functional modeling of desired system behavior. By modeling the expected behavior with use cases, we can uncover required classes and associations that will be needed during the construction phase of the object model. We emphasize the necessary components for a complete use case in the following list:

* actor, who initiates an event or events
* specific interaction that takes place between this actor and the system (sequence of events or scenarios)
* expected value from the system

When the software engineer meets with the potential end users of the desired system during requirements elicitation, the objective is to get answers to information-seeking questions to form the components of a **use case**. A list of sample questions that the software engineer can use during meetings with the customers and end users includes:

* What is the expected behavior of the system?
* How is the user supposed to use the system?
* What types of services is the system supposed to provide?
* What is the expected result/output of the system?
* Who will be using the system?

The software engineer composes a use case from the information.

A use case **sequence of events or scenario** is created for each use case to depict the expected behavior from the perspective of the actors. These scenarios assist the software engineer with elicitation and analysis of the requirements by describing the functionality that is required in the software. A functional requirement and the corresponding test case can be derived by the action that is modeled in the use case. A test case is used later in development to verify that a functional requirement has been met in the software product

Once all the use cases are developed, they will assist you in identifying the data objects needed to create the proposed software system.

**Use Case and the Unified Modeling Language**

The Unified Modeling Language (UML) provides a user model and notation for the specification of the different ways an end user will interact with the system, the use case.

UML stresses the importance of textual documentation for each use case to accompany each scenario illustration. Following is a list of the items to be included in the use case documentation:

* unique name and number for the use case
* description of the role and purpose of the action
* description of the basic sequence of events (scenario)
* description outlining any special requirements or conditions
* description stating the specific pre-conditions and post-conditions

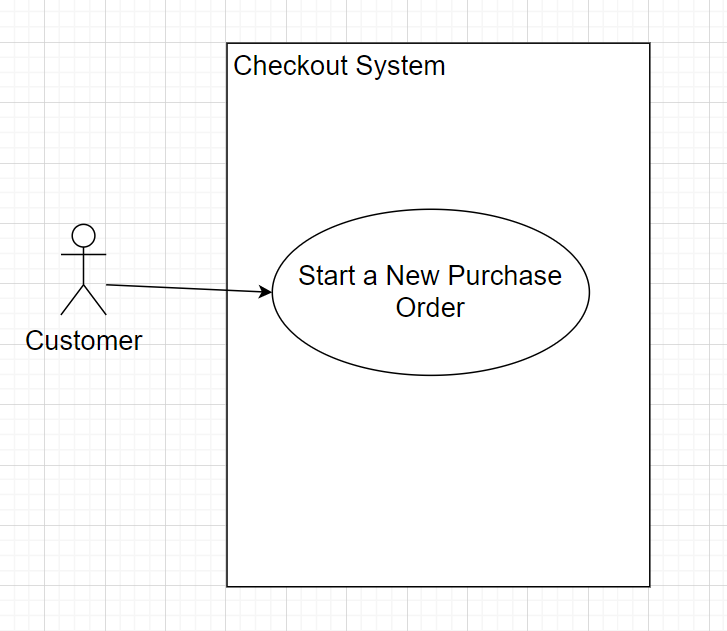
A descriptive name is selected and assigned to each use case that best describes the function required by the software. Nomenclature for use cases is very important in software engineering, as it enhances documentation. When possible, similar functions should be categorized in one use case. A unique number or identifier is assigned to each use case to enhance the traceability to the requirement, the software component, and the test case.

**Example Use Case**

Let's take a look at a use case that we can define for the self-service checkout system.  The use case for the customer in a retail store using the self-service checkout system to start a new purchase order is outlined in the table below.

|  |
| --- |
| Use Case to Start a New Purchase Order  **Use Case 1.0:** Start new purchase order  A customer selects one or more items to purchase in a retail store. Once customers have completed selecting all the items, they will walk up to the checkout counter and start a new purchase order to prepare for scanning each item.  **Precondition:** The checkout system needs to be configured properly and have access to the local area network and store database. The system needs to be ready to accept a new purchase order. **Post-condition:** The system is ready to scan items for a new purchase order.  **Actor Profile:** Customer shopping in a retail store who desires to purchase one or more items. A customer can be new to the system or be a frequent shopper familiar with the system. Frequent customers may have a discount card. Many customers will shop in the retail store.  **Sequence of events (scenario) :**  1.  Customer walks up to an available self-service checkout.  2.  Customer reads instructions and selects either Spanish or English language.  3.  Customer selects the option to begin the order.  4.  Customer scans frequent shopper discount card, if one exists.  5.  Customer reads the directions for scanning an item. |

Once the use case scenarios are written for the use case, we can model them with a use case diagram as shown in Figure 1.



**Figure 1**

**Start a New Purchase Order Use Case Diagram**

**Class Diagram Model**

The use case can provide insight into the identification of objects that can be grouped by similarity to form classes for these objects. Objects are easily identifiable because the actors of a system manipulate objects when they use the software to perform a function to accomplish a task. Identifying the commonality among the objects will help software engineers to derive the base classes and establish attributes, needed functionality, and relationships. Software engineers model classes and their relationships with diagrams. UML is the common notation language being used in today's software industry for creating class diagrams.

Once the potential objects are identified, they can be grouped by similarity and used to determine base classes with a set of attributes. A class describes a group of objects with similar properties or attributes, common behaviors, relationships to other objects, and semantics. For example, think of a person as an object. Every person has the same set of common attributes, including a name, gender, height, weight, eye color, nationality, fingerprint, and social security number. These attributes are used to identify an individual and distinguish one person from another. A base class can be designed with these attributes and the class can be used to create instances of different individuals with a unique set of characteristics. Class instances are also referred to as objects. An object is defined as a concept, idea, or thing with crisp boundaries and meaning for the problem at hand.

Once the software engineer establishes all the base classes, the relationships between the classes are modeled using a class diagram. A class diagram is a schema, pattern, or template used for describing the class. Each class description contains a name, a set of attributes, and a set of operations.

Creating a UML class diagram can achieve several goals in modeling expected behavior. The class diagram can be used to:

document classes within a system or subsystem  
describe association, generalization, and aggregation between classes show characteristics of the class  
show the operations and interfaces of objects  
 direct activities throughout the development process

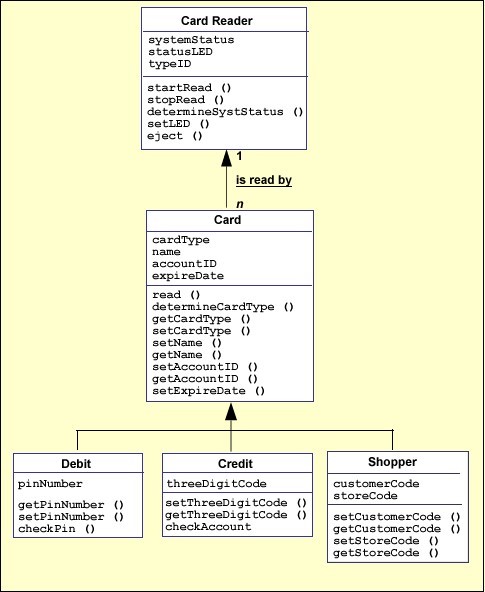
Class instances or objects collaborate with other objects through message passing. An association between two classes is described when a class instance or object passes a message to another object and that message is received by the recipient object. When drawing a class diagram, a connecting line is used to depict the association. The association is assigned a descriptive name. Software engineers can further define the association existing between the two class instances by adding its multiplicity, which indicates the possible number of instances for the class. In Table 1 we summarize the indicators that can be placed at each end of the line in the class diagram to show multiplicity.

Table 1

Multiplicity Indicators for Class Diagrams

|  |  |
| --- | --- |
| **Indicator** | **Meaning** |
| 0..1 | Zero or one |
| 1 | One only |
| 0..\* | Zero or more |
| 1..\* | One or more |
| n | Only *n*, where *n*> 1 |
| 0..n | Zero to *n*, where *n* > 1 |
| 1..n | One to *n*, where *n*> 1 |

Refer to Figure 2 for a class diagram that depicts the card reader and the card objects that are necessary for the payment transaction of the self-service checkout system.



**Figure 2  
Example Class Diagram**

**Data Dictionary**

We will mention the data dictionary during requirements analysis because it can be used to store logical representations of data objects. It is important to adopt a notation for a data dictionary that provides for unambiguous and accurate descriptions of the data. Such notations facilitate interpretation and also allow for the use of CASE tools to assist in developing and maintaining dictionaries.

Modern systems may require extremely large data dictionaries, which may be practical only when a CASE tool is available to reduce the human effort and manage the complexity. Such tools, however, are useful for more than simple management of the data objects. If a formalized notation is used to describe the data, the tool can perform automated analyses of the data to ensure consistency across the system. Such analyses would be quite difficult if developers had to perform them by themselves.