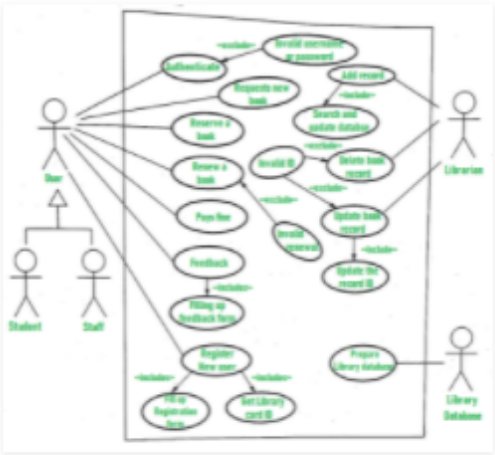
OOP Outlines Solved

1. Use-case Diagram for Library Work

Use Case Diagram for Library Management System

Difficulty Level : Basic • Last Updated : 16 Jun, 2020

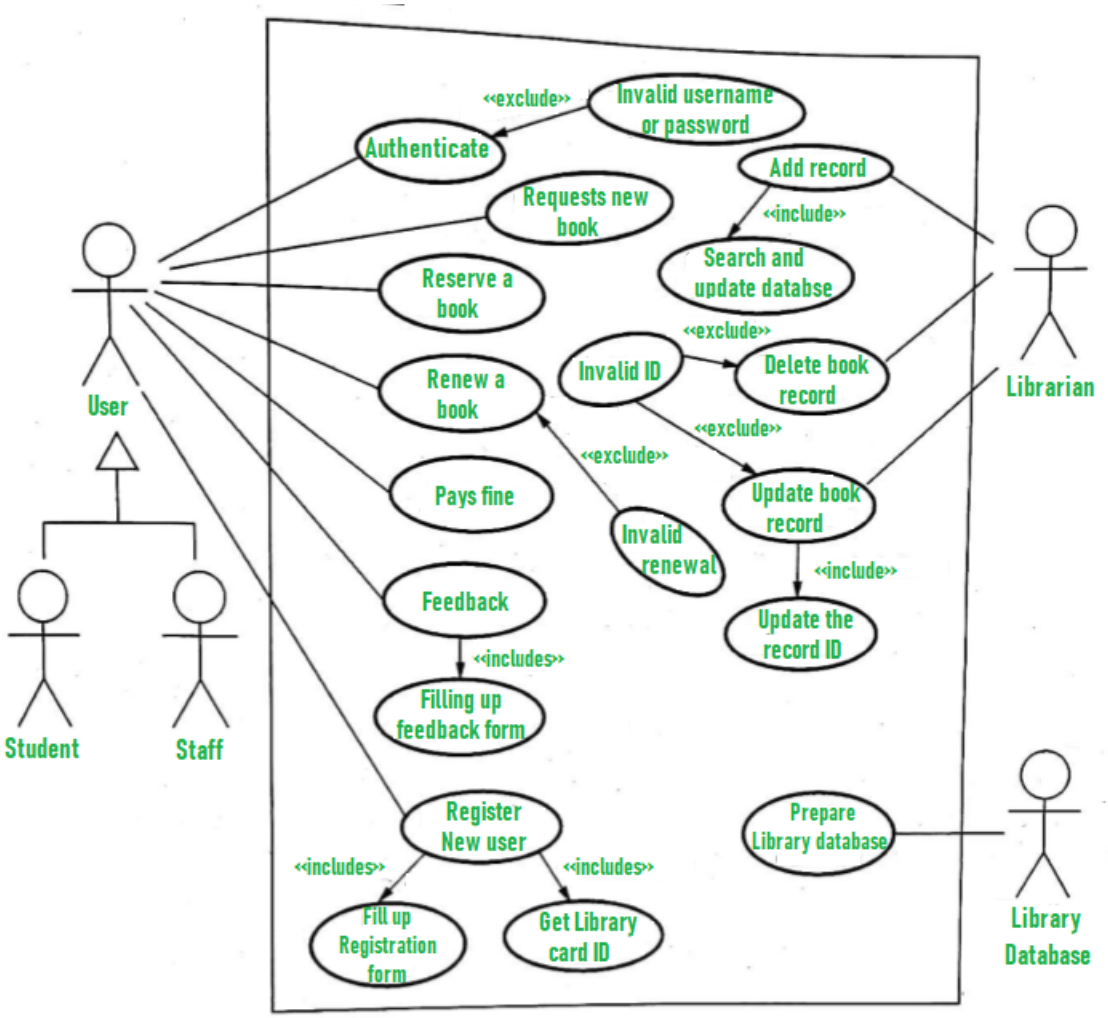
Use case diagrams referred as a Behavior model or diagram. It simply describes and displays the relation or interaction between the users or customers and providers of application service or the system. It describes different actions that a system performs in collaboration to achieve something with one or more users of the system. Use case diagram is used a lot nowadays to manage the system.



Here, we will understand the designing use case diagram for the library management system. Some scenarios of the system are as follows :

1. User who registers himself as a new user initially is regarded as staff or student for the library system.
   - For the user to get registered as a new user, registration forms are available that is needed to be fulfilled by the user.
   - After registration, a library card is issued to the user by the librarian. On the library card, an ID is assigned to cardholder or user.
2. After getting the library card, a new book is requested by the user as per there requirement.
3. After, requesting, the desired book or the requested book is reserved by the user that means no other user can request for that book.
4. Now, the user can renew a book that means the user can get a new due date for the desired book if the user has renewed them.

5. If the user somehow forgets to return the book before the due date, then the user pays fine. Or if the user forgets to renew the book till the due date, then the book will be overdue and the user pays fine.
6. User can fill the feedback form available if they want to.
7. Librarian has a key role in this system. Librarian adds the records in the library database about each student or user every time issuing the book or returning the book, or paying fine.
8. Librarian also deletes the record of a particular student if the student leaves the college or passed out from the college. If the book no longer exists in the library, then the record of the particular book is also deleted.
9. Updating database is the important role of Librarian.



Class Diagram for Library Management System

Class Diagram for Library Management System

Difficulty Level : Basic • Last Updated : 08 Jul, 2020

In Object-Oriented modeling, the main building block generally represents different objects in a system, their attributes, their different functions, and relationships among objects. These building blocks are known as **Class Diagram**.

Class diagrams are generally used for conceptual modeling of static view of a software application, and for modeling translating models into programming code in a detailed manner. At time of developing or construction software systems, a class diagram is widely used. They are also used for data modeling. It is used to show classes, relationships among them, interface, association, etc. Class in a class diagram simply is a blueprint of an object. It simply describes and explains different type of objects in system, and different types of relationships that exist between them.

Class Diagram for Library Management System :
Aggregation and Multiplicity are two important points that need to take into consideration while designing a Class Diagram. Let us understand in detail.

1. **Aggregation –**
   Aggregation simply shows a relationship where one thing can exist independently of other thing. It means to create or compose different abstractions together in defining a class. Aggregation is represented as a part of relationship in class diagram. In diagram given below, we can see that aggregation is represented by an edge with a diamond end pointing towards superclass. The "Library Management System" is superclass that consists of various classes.
   These classes are User, Book, and Librarian as shown in diagram. Further, for "Account" class, "User" is a superclass. All of these, share a relationship and these relationships are known as aggregate relationships.

2. **Multiplicity –**
   Multiplicity means that number of elements of a class is associated with another class. These relations can be one-to-one, many-to-many, and many-to-one or one-to-many. For denoting one element we use **1**, for zero elements we use **0**, and for many elements we use ***. We can see in diagram; many users are associated with many books denoted by * and this represents a **many-to-many** type of relationship. One user has only one account that is denoted by 1 and this represents a **one-to-one** type of relationship.
   Many books are associated with one librarian and this represents **many-to-one** or **one-to-many** type of relationship. All these relationships are shown in diagram.

Class Diagram for Library Management System simply describes structure of Library Management System class, attributes, methods or operations, relationship among objects.

Classes of Library Management System :

- **Library Management System class –**
  It manages all operations of Library Management System. It is central part of organization for which software is being designed.
- **User Class –**
  It manages all operations of user.
- **Librarian Class –** It manages all operations of Librarian.
- **Book Class –**
  It manages all operations of books. It is basic building block of system.
- **Account Class –**
  It manages all operations of account.
- **Library database Class –**
  It manages all operations of library database.
- **Staff Class –**
  It manages all operations of staff.
- **Student Class –**
  It manages all operations of student.

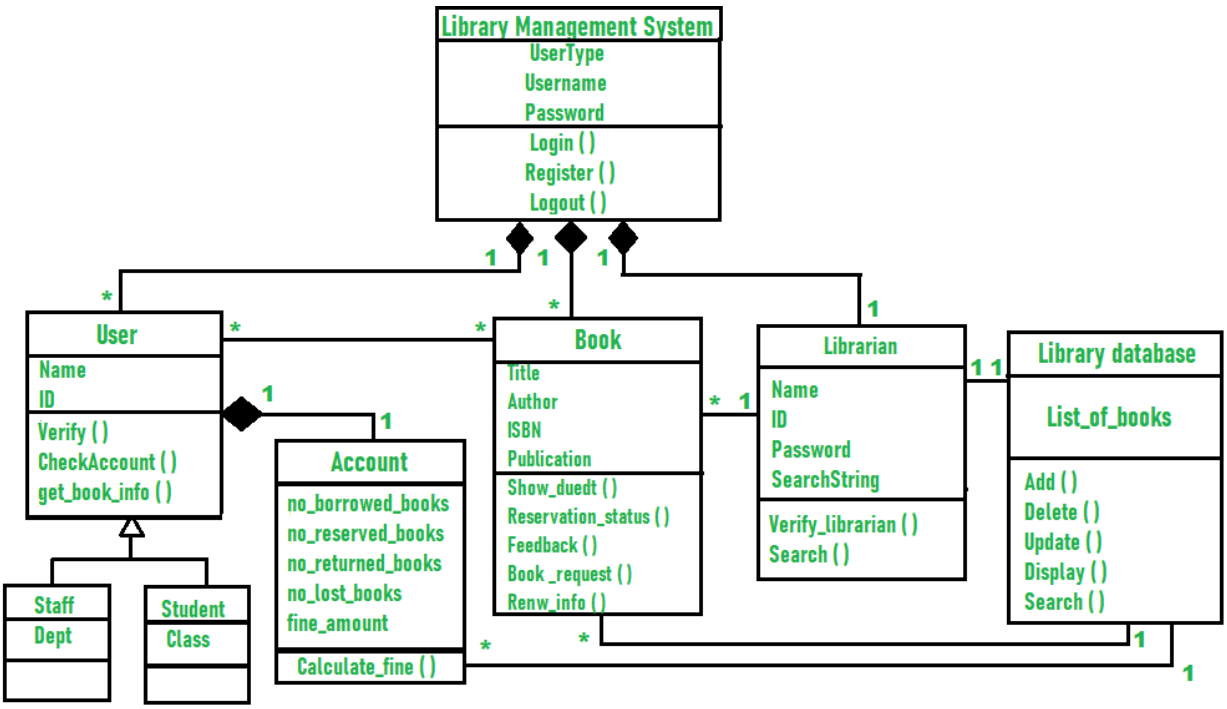
Attributes of Library Management System :

- **Library Management System Attributes –**
  UserType, Username, Password
- **User Attributes –**
  Name, Id
- **Librarian Attributes –**
  Name, Id, Password, SearchString
- **Book Attributes –**
  Title, Author, ISBN, Publication
- **Account Attributes –**
  no_borrowed_books, no_reserved_books, no_returned_books, no_lost_books fine_amount
- **Library database Attributes –**
  List_of_books
- **Staff Class Attributes –**
  Dept
- **Student Class Attributes –**
  Class

Methods of Library Management System :

- **Library Management System Methods –**
  Login(), Register(), Logout()
- **User Methods –**
  Verify(), CheckAccount(), get_book_info()
- **Librarian Methods –**
  Verify_librarian(), Search()
- **Book Methods –**
  Show_duedt(), Reservation_status(), Feedback(), Book_request(), Renew_info()
- **Account Methods –**
  Calculate_fine()
- **Library database Methods –**
  Add(), Delete(), Update(), Display(), Search()

Class Diagram of Library Management System :



CLASS DIAGRAM FOR LIBRARY MANAGEMENT SYSTEM

2. DFD Diagram and Mechanics

What is DFD(Data Flow Diagram)?

Difficulty Level : Expert • Last Updated : 26 May, 2020

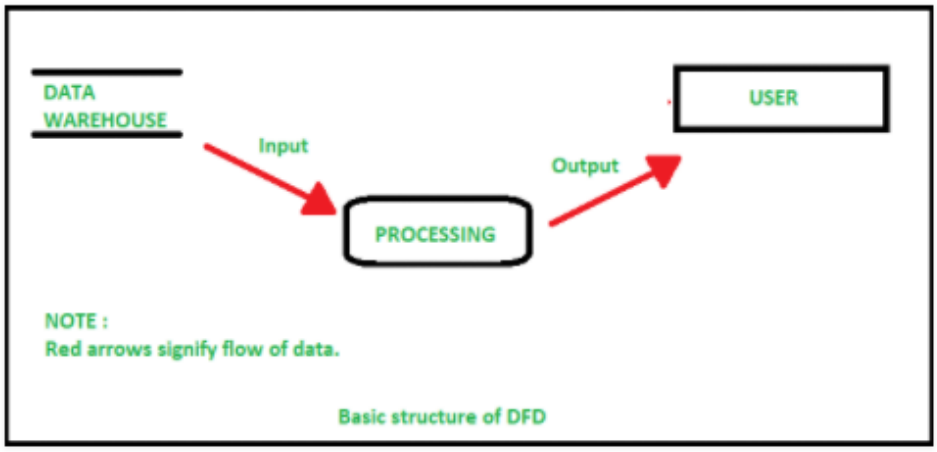
DFD is the abbreviation for **Data Flow Diagram**. The flow of data of a system or a process is represented by DFD. It also gives insight into the inputs and outputs of each entity and the process itself. DFD does not have control flow and no loops or decision rules are present. Specific operations depending on the type of data can be explained by a flowchart. Data Flow Diagram can be represented in several ways. The DFD belongs to structured-analysis modeling tools. Data Flow diagrams are very popular because they help us to visualize the major steps and data involved in software-system processes.

Components of DFD

The Data Flow Diagram has 4 components:

- **Process**
  Input to output transformation in a system takes place because of process function. The symbols of a process are rectangular with rounded corners, oval, rectangle or a circle. The process is named a short sentence, in one word or a phrase to express its essence

- **Data Flow**
  Data flow describes the information transferring between different parts of the systems. The arrow symbol is the symbol of data flow. A relatable name should be given to the flow to determine the information which is being moved. Data flow also represents material along with information that is being moved. Material shifts are modeled in systems that are not merely informative. A given flow should only transfer a single type of information. The direction of flow is represented by the arrow which can also be bi-directional.
- **Warehouse**
  The data is stored in the warehouse for later use. Two horizontal lines represent the symbol of the store. The warehouse is simply not restricted to being a data file rather it can be anything like a folder with documents, an optical disc, a filing cabinet. The data warehouse can be viewed independent of its implementation. When the data flow from the warehouse it is considered as data reading and when data flows to the warehouse it is called data entry or data updation.
- **Terminator**
  The Terminator is an external entity that stands outside of the system and communicates with the system. It can be, for example, organizations like banks, groups of people like customers or different departments of the same organization, which is not a part of the model system and is an external entity. Modeled systems also communicate with terminator.



Basic structure of DFD

Rules for creating DFD

- The name of the entity should be easy and understandable without any extra assistance(like comments).
- The processes should be numbered or put in ordered list to be referred easily.
- The DFD should maintain consistency across all the DFD levels.
- A single DFD can have maximum processes upto 9 and minimum 3 processes.

Levels of DFD

DFD uses hierarchy to maintain transparency thus multilevel DFD's can be created. Levels of DFD are as follows:

- 0-level DFD
- 1-level DFD:
- 2-level DFD:

Advantages of DFD

- It helps us to understand the functioning and the limits of a system.
- It is a graphical representation which is very easy to understand as it helps visualize contents.
- Data Flow Diagram represent detailed and well explained diagram of system components.
- It is used as the part of system documentation file.
- Data Flow Diagrams can be understood by both technical or nontechnical person because they are very easy to understand.

Disadvantages of DFD

- At times DFD can confuse the programmers regarding the system.
- Data Flow Diagram takes long time to be generated, and many times due to this reasons analysts are denied permission to work on it.

Levels in Data Flow Diagrams (DFD)

Difficulty Level : Medium • Last Updated : 28 Oct, 2020

In Software engineering DFD(data flow diagram) can be drawn to represent the system of different levels of abstraction. Higher-level DFDs are partitioned into low levels-hacking more information and functional elements. Levels in DFD are numbered 0, 1, 2 or beyond. Here, we will see mainly 3 levels in the data flow diagram, which are: 0-level DFD, 1-level DFD, and 2-level DFD.

0-level DFD:

It is also known as a context diagram. It's designed to be an abstraction view, showing the system as a single process with its relationship to external entities. It represents the entire system as a single bubble with input and output data indicated by incoming/outgoing arrows.

1-level DFD:

In 1-level DFD, the context diagram is decomposed into multiple bubbles/processes. In this level, we highlight the main functions of the system and breakdown the high-level process of 0-level DFD into subprocesses.

2-level DFD:

2-level DFD goes one step deeper into parts of 1-level DFD. It can be used to plan or record the specific/necessary detail about the system's functioning.

3. Interaction Diagram

Interaction Diagrams

Interaction diagrams are models that describe how a group of objects collaborate in some behavior - typically a single use-case. The diagrams show a number of example objects and the messages that are passed between these objects within the use-case.

I'll illustrate the approach with the following simple use-case. In this behavior the order entry window sends a prepare message to an order. The order then sends prepare to each order line on the order. The order line first checks the stock item, and if the check returns true it removes stock from the stock item. If stock item falls below the reorder level it requests a new delivery.

Interaction diagrams come in two forms, both present in the UML. The first form is the sequence diagram. In this form objects are shown as vertical lines with the messages as horizontal lines between them. This form was first popularized by Jacobson. The diagram below shows this form in its UML notation. The sequence of messages is indicated by reading down the page.

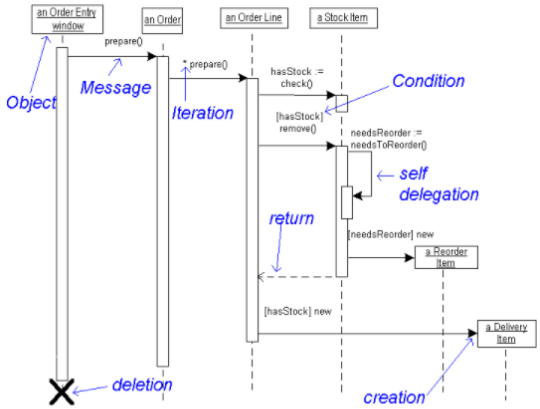


Figure 1: A sequence diagram.

The second form of the interaction diagram is the collaboration diagram. Here the example objects are shown as icons. Again arrows indicate the messages sent in the use case. This time the sequence is indicated by a numbering scheme. Simple collaboration diagrams simply number the messages in sequence. More complex schemes use a decimal numbering approach to indicate if messages are sent as part of the implementation of another message. In addition a letter can be used to show concurrent threads.

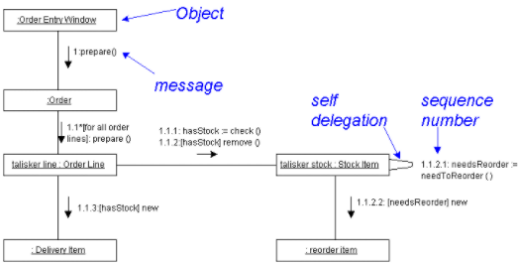


Figure 2: A collaboration diagram.

One of the great strengths of an interaction diagram is its simplicity. It is difficult to write much about interaction diagrams because they are so simple. They do, however, have weaknesses, the principal one is that although they are good at describing behavior: they do not define it. They typically do not show all the iteration and control that is needed to give an computationally complete description. Various things have been done to try and remedy this in the UML.

I have mixed feelings about these trends towards greater executability. To me, the beauty of interaction diagrams is their simplicity, and much of these additional notations lose this in their drive to computational completeness. Thus I would encourage you not to *rush* to the more complex forms of interaction diagrams, you may find that the simpler ones give you the best value.

When to Use Them

Interaction diagrams should be used when you want to look at the behavior of several objects within a single use case. They are good at showing the collaborations between the objects, they are not so good at precise definition of the behavior.

If you want to look at the behavior of a single object across many use-cases, use a state transition diagram. If you want to look at behavior across many use cases or many threads, consider an activity diagram.

Where to Find Out More

Most books on object-oriented modeling discuss some form of interaction diagram. Notations are currently very varied, but they should settle down to the UML in due course. A safe suggestion is that of [Booch] which gives you a simple outline, which is really all you need.

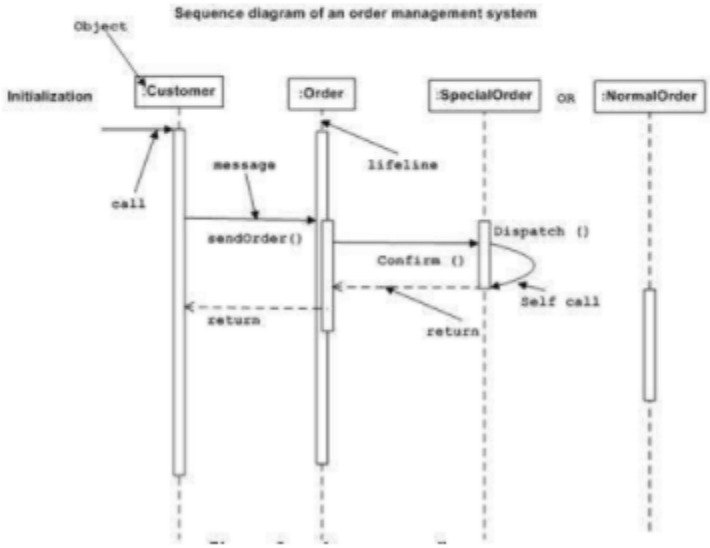
4. Sequence Diagram

The Sequence Diagram

The sequence diagram has four objects (Customer, Order, SpecialOrder and NormalOrder).

The following diagram shows the message sequence for *SpecialOrder* object and the same can be used in case of *NormalOrder* object. It is important to understand the time sequence of message flows. The message flow is nothing but a method call of an object.

The first call is *sendOrder ()* which is a method of *Order object*. The next call is *confirm ()* which is a method of *SpecialOrder* object and the last call is *Dispatch ()* which is a method of *SpecialOrder* object. The following diagram mainly describes the method calls from one object to another, and this is also the actual scenario when the system is running.



5. CRC and State chart Diagram

UML - Statechart Diagrams

The name of the diagram itself clarifies the purpose of the diagram and other details. It describes different states of a component in a system. The states are specific to a component/object of a system.

A Statechart diagram describes a state machine. State machine can be defined as a machine which defines different states of an object and these states are controlled by external or internal events.

Activity diagram explained in the next chapter, is a special kind of a Statechart diagram. As Statechart diagram defines the states, it is used to model the lifetime of an object.

Purpose of Statechart Diagrams

Statechart diagram is one of the five UML diagrams used to model the dynamic nature of a system. They define different states of an object during its lifetime and these states are changed by events. Statechart diagrams are useful to model the reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

Statechart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. The most important purpose of Statechart diagram is to model lifetime of an object from creation to termination.

Statechart diagrams are also used for forward and reverse engineering of a system. However, the main purpose is to model the reactive system.

Following are the main purposes of using Statechart diagrams −

- To model the dynamic aspect of a system.
- To model the life time of a reactive system.
- To describe different states of an object during its life time.
- Define a state machine to model the states of an object.

How to Draw a Statechart Diagram?

Statechart diagram is used to describe the states of different objects in its life cycle. Emphasis is placed on the state changes upon some internal or external events. These states of objects are important to analyze and implement them accurately.

Statechart diagrams are very important for describing the states. States can be identified as the condition of objects when a particular event occurs.

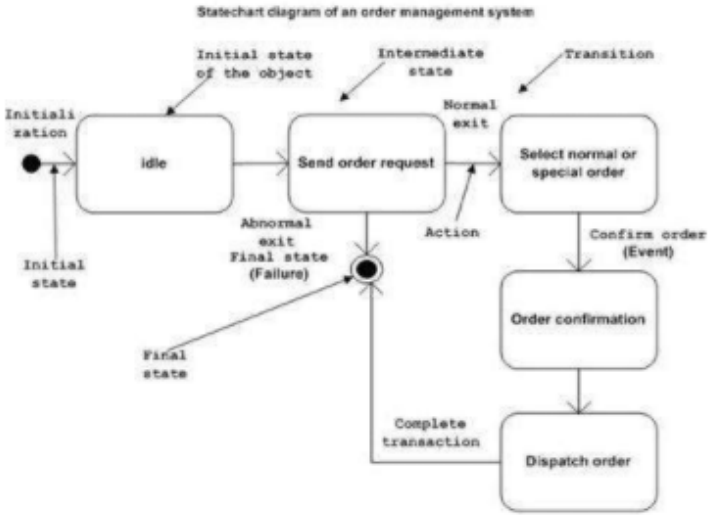
Before drawing a Statechart diagram we should clarify the following points −

- Identify the important objects to be analyzed.
- Identify the states.
- Identify the events.

Following is an example of a Statechart diagram where the state of Order object is analyzed

The first state is an idle state from where the process starts. The next states are arrived for events like send request, confirm request, and dispatch order. These events are responsible for the state changes of order object.

During the life cycle of an object (here order object) it goes through the following states and there may be some abnormal exits. This abnormal exit may occur due to some problem in the system. When the entire life cycle is complete, it is considered as a complete transaction as shown in the following figure. The initial and final state of an object is also shown in the following figure.



Where to Use Statechart Diagrams?

From the above discussion, we can define the practical applications of a Statechart diagram. Statechart diagrams are used to model the dynamic aspect of a system like other four diagrams discussed in this tutorial. However, it has some distinguishing characteristics for modeling the dynamic nature.

Statechart diagram defines the states of a component and these state changes are dynamic in nature. Its specific purpose is to define the state changes triggered by events. Events are internal or external factors influencing the system.

Statechart diagrams are used to model the states and also the events operating on the system. When implementing a system, it is very important to clarify different states of an object during its life time and Statechart diagrams are used for this purpose. When these states and events are identified, they are used to model it and these models are used during the implementation of the system.

If we look into the practical implementation of Statechart diagram, then it is mainly used to analyze the object states influenced by events. This analysis is helpful to understand the system behavior during its execution.

The main usage can be described as −

- To model the object states of a system.
- To model the reactive system. Reactive system consists of reactive objects.
- To identify the events responsible for state changes.
- Forward and reverse engineering.

Class Responsibility Collaborator (CRC) Models: An Agile Introduction

A Class Responsibility Collaborator (CRC) model (Beck & Cunningham 1989; Wilkinson 1995; Ambler 1995) is a collection of standard index cards that have been divided into three sections, as depicted in Figure 1. A class represents a collection of similar objects, a responsibility is something that a class knows or does, and a collaborator is another class that a class interacts with to fulfill its responsibilities. Figure 2 presents an example of two hand-drawn CRC cards.

Figure 1. CRC Card Layout.

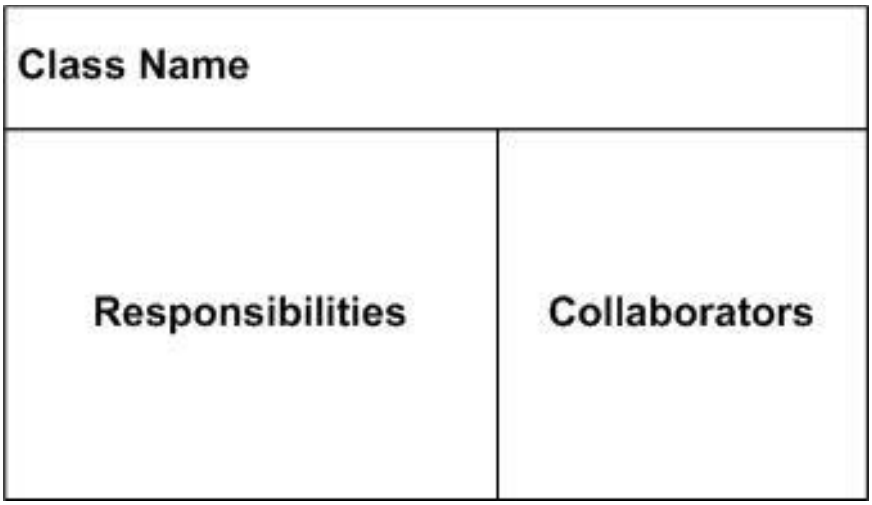
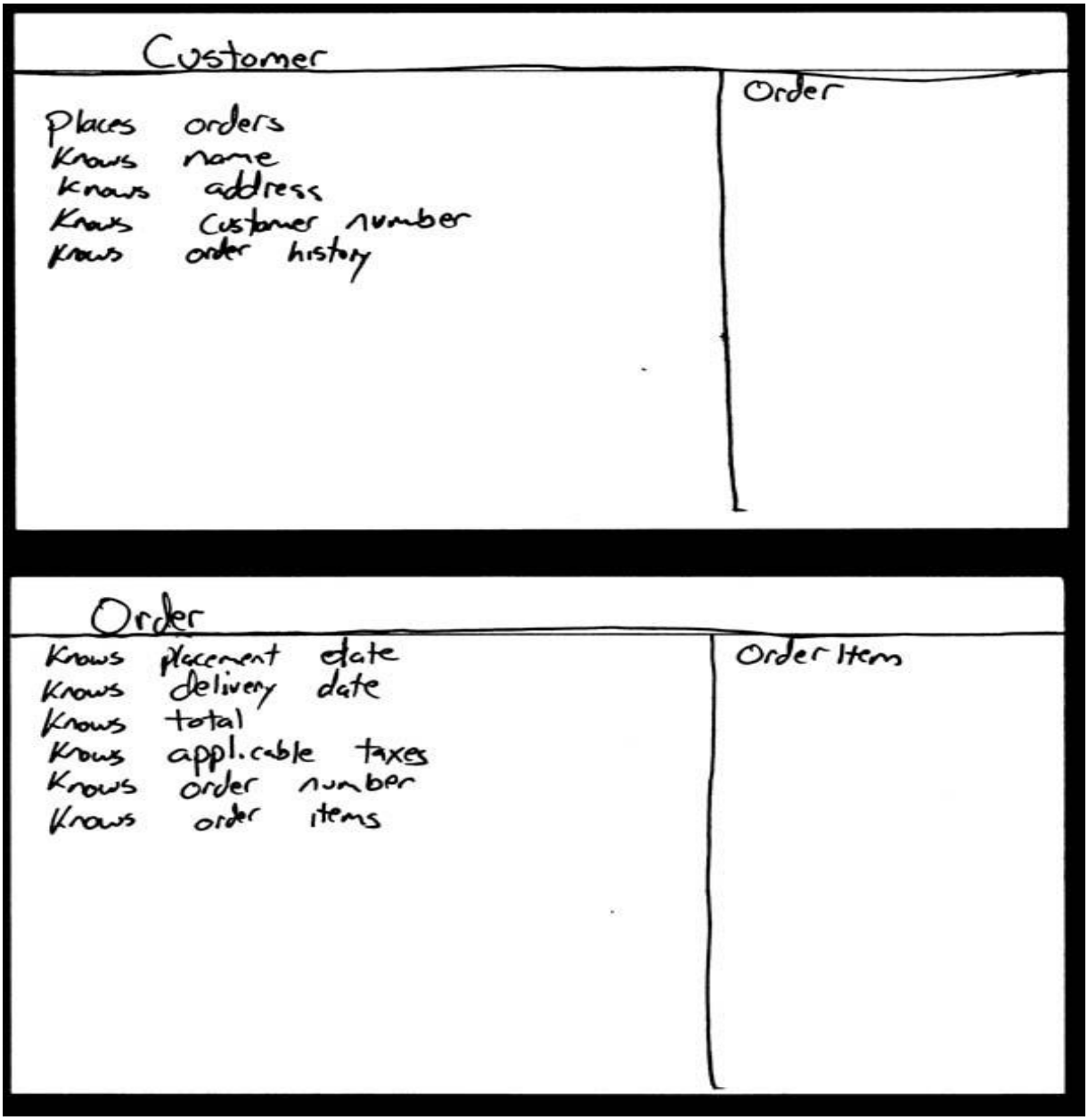


Figure 2. Hand-drawn CRC Cards.



Although CRC cards were originally introduced as a technique for teaching object-oriented concepts, they have also been successfully used as a full-fledged modelling technique. My experience is that CRC models are an incredibly effective tool for conceptual modelling as well as for detailed design. CRC cards feature prominently in extreme Programming (XP) (Beck 2000) as a design technique. My focus here is on applying CRC cards for conceptual modelling with your stakeholders.

A class represents a collection of similar objects. An object is a person, place, thing, event, or concept that is relevant to the system at hand. For example, in a university system, classes would represent students, tenured professors, and seminars. The name of the class appears across the top of a CRC card and is typically a singular noun or singular noun phrase, such as *Student*, *Professor*, and *Seminar*. You use singular names because each class represents a generalized version of a singular object. Although there may be the student John O'Brien, you would model the class *Student*. The information about a student describes a single person, not a group of people. Therefore, it makes sense to use the name *Student* and not *Students*. Class names should also be simple. For example, which name is better: *Student* or *Person who takes seminars*?

A responsibility is anything that a class knows or does. For example, students have names, addresses, and phone numbers. These are the things a student knows. Students also Enroll in seminars, drop seminars, and request transcripts. These are the things a student does. The things a class knows and does constitute its responsibilities. Important: A class is able to change the values of the things it knows, but it is unable to change the values of what other classes know.

THE END!