**Adding operations**

On a UML class diagram, you can add *operations* to classes and interfaces. An operation is a method or function that can be performed by an instance of a class or interface.

To add an operation, right-click the class or interface, point to **Add**, and then click **Operation**.

If the operations of a class on the diagram are not visible, click the expand chevron at the top of the class or interface. If you can see the **Operation** header, click **[+]** to expand the operations section.

**Signature of an Operation**

The signature of an operation is the line of text that represents it in a class or interface on a UML class diagram. It has the following form:

+ OperationName (parameter1 : Type1 [\*], ...) : ReturnType [\*]

+ denotes public Visibility. The other permitted values are - (private), # (protected), ~ (package).

OperationName is underlined if the **Is Static** property is true, and is italic if the **Is Abstract** property is true.

: ReturnType is omitted if no return type is defined.

[\*] denotes the multiplicity of a parameter or return type. It is omitted if the multiplicity is 1.

## Identifying Concurrency

Concurrency allows more than one objects to receive events at the same time and more than one activity to be executed simultaneously. Concurrency is identified and represented in the dynamic model.

To enable concurrency, each concurrent element is assigned a separate thread of control. If the concurrency is at object level, then two concurrent objects are assigned two different threads of control. If two operations of a single object are concurrent in nature, then that object is split among different threads.

Concurrency is associated with the problems of data integrity, deadlock, and starvation. So a clear strategy needs to be made whenever concurrency is required. Besides, concurrency requires to be identified at the design stage itself, and cannot be left for implementation stage.

UML behavioral diagrams visualize, specify, construct, and document the dynamic aspects of a system. The behavioral diagrams are categorized as follows: use case diagrams, interaction diagrams, state–chart diagrams, and activity diagrams.

## Use Case Model

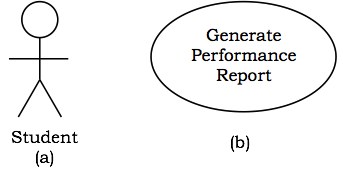
### Use case

A use case describes the sequence of actions a system performs yielding visible results. It shows the interaction of things outside the system with the system itself. Use cases may be applied to the whole system as well as a part of the system.

### Actor

An actor represents the roles that the users of the use cases play. An actor may be a person (e.g. student, customer), a device (e.g. workstation), or another system (e.g. bank, institution).

The following figure shows the notations of an actor named Student and a use case called Generate Performance Report.



## Use case diagrams

Use case diagrams present an outside view of the manner the elements in a system behave and how they can be used in the context.

Use case diagrams comprise of −

* Use cases
* Actors
* Relationships like dependency, generalization, and association

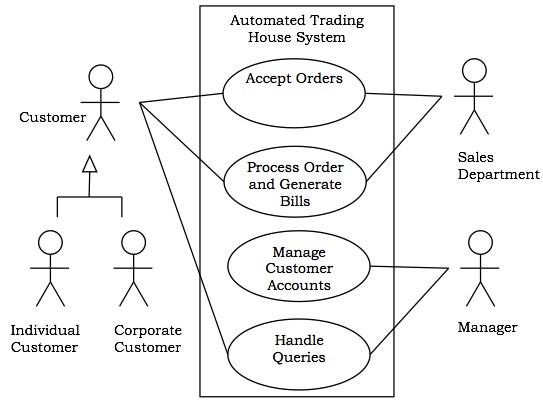
Use case diagrams are used −

* To model the context of a system by enclosing all the activities of a system within a rectangle and focusing on the actors outside the system by interacting with it.
* To model the requirements of a system from the outside point of view.

**Example**

Let us consider an Automated Trading House System. We assume the following features of the system −

* The trading house has transactions with two types of customers, individual customers and corporate customers.
* Once the customer places an order, it is processed by the sales department and the customer is given the bill.
* The system allows the manager to manage customer accounts and answer any queries posted by the customer.



## Interaction Diagrams

Interaction diagrams depict interactions of objects and their relationships. They also include the messages passed between them. There are two types of interaction diagrams −

* Sequence Diagrams
* Collaboration Diagrams

### Interaction diagrams are used for modeling −

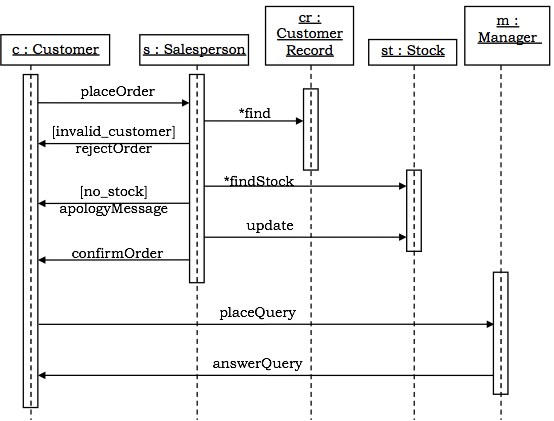
* the control flow by time ordering using sequence diagrams.
* the control flow of organization using collaboration diagrams.

## Sequence Diagrams

Sequence diagrams are interaction diagrams that illustrate the ordering of messages according to time.

**Notations** − These diagrams are in the form of two-dimensional charts. The objects that initiate the interaction are placed on the x–axis. The messages that these objects send and receive are placed along the y–axis, in the order of increasing time from top to bottom.

**Example** − A sequence diagram for the Automated Trading House System is shown in the following figure.

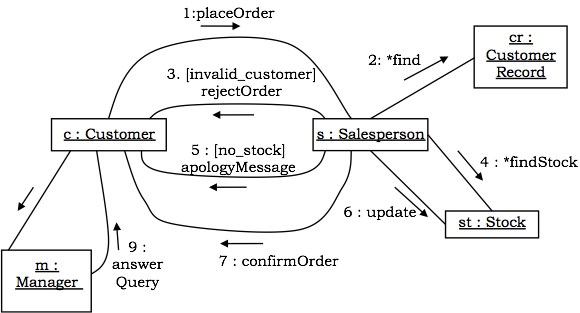


## Collaboration Diagrams

Collaboration diagrams are interaction diagrams that illustrate the structure of the objects that send and receive messages.

**Notations** − In these diagrams, the objects that participate in the interaction are shown using vertices. The links that connect the objects are used to send and receive messages. The message is shown as a labeled arrow.

**Example** − Collaboration diagram for the Automated Trading House System is illustrated in the figure below.



## State–Chart Diagrams

A state–chart diagram shows a state machine that depicts the control flow of an object from one state to another. A state machine portrays the sequences of states which an object undergoes due to events and their responses to events.

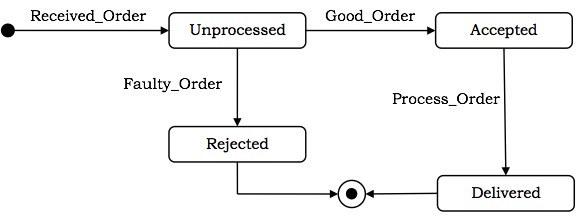
State–Chart Diagrams comprise of −

* States: Simple or Composite
* Transitions between states
* Events causing transitions
* Actions due to the events

State-chart diagrams are used for modeling objects which are reactive in nature.

**Example**

In the Automated Trading House System, let us model Order as an object and trace its sequence. The following figure shows the corresponding state–chart diagram.



## Activity Diagrams

An activity diagram depicts the flow of activities which are ongoing non-atomic operations in a state machine. Activities result in actions which are atomic operations.

Activity diagrams comprise of −

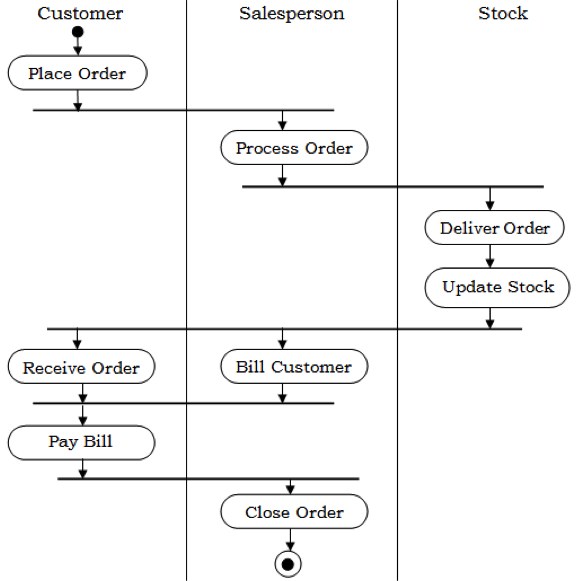
* Activity states and action states
* Transitions
* Objects

Activity diagrams are used for modeling −

* workflows as viewed by actors, interacting with the system.
* details of operations or computations using flowcharts.

**Example**

The following figure shows an activity diagram of a portion of the Automated Trading House System.



# Iteration in Object-Oriented Analysis & Design

Share

Successful software development requires more than powerful concepts and notation; it also requires rigorous software processes.  The software process associated with OO development is inherently different from traditional software life cycles.  For example, whereas the traditional “waterfall” analysis/design/implementation/test life cycle is sequential, the OO software lifecycle is iterative and incremental.  It is sometimes caricatured as “analyze a little, design a little, implement a little, test a little.”

**Why Iterate**

The advantage of an iterative software process is that it allows us, when there is reason to do so, to return to a previous step, introduce a change, and propagate the ramifications of the change forward in the life cycle.  Consequently, as we learn more about the problem from prototypes and user feedback we can continuously refine and extend our design.  The advantage of an incremental software process is that it allows the system to evolve in a series of phased releases, beginning with a skeleton architecture and culminating in a deliverable software product.  As a result, we can systematically extend the architecture as we learn by experience with previous releases.

**Disadvantages**

The main disadvantage of the iterative and incremental lifecycle is that, for large projects, it is frequently a challenge to manage.  Since the various analysis/design/implementation/testing activities typically proceed in parallel, coordinating  activities within and across the releases requires sophisticated project management skills.  To successfully orchestrate these concurrent activities for a large project, a solid understanding of the core OOAD activities and deliverables is essential.

**Iteration in Analysis and Design**

However, unlike in the traditional software life cycle, in the OO life cycle the separation of concerns between analysis and design is not strictly enforced.  Developers are not only allowed, they are encouraged, to move back and forth between analysis and design activities.  The object model facilitates this fluidity, since it is central to both OOA and OOD activities.  For this reason it is sometimes convenient to view the OOA object model as a logical object model, and the OOD object model as a physical object model.