OOAD – UNIT I

Object-Oriented Analysis & Design

Object-Oriented Design is a software development approach to design and implement software system as a collection of interacting stateful objects with specified structure and behavior.

There are several fundamental concepts defining OOD but there is no agreement on the exact list of the concepts, their definition and taxonomy (classification). We will take a look at some of OOD concepts that seem relevant to the UML:

- class and object,
- message, operation, method,
- encapsulation,
- abstraction,
- inheritance,
- polymorphism.

Class and Object in UML

UML **class** is a classifier which describes a set of objects that share the same

- features,
- constraints,
- **semantics (meaning)**.

Class may be modeled as being **active**, meaning that an instance of the class has some autonomous behavior.

Object is an instance of a class.

An entity with a well defined boundary and identity that encapsulates state and behavior. State is represented by attributes and relationships, behavior is represented by operations, methods, and state machines. An object is an instance of a class.

The Unified Process

- The Unified Process has emerged as a popular and effective software development process.
- In particular, the Rational Unified Process, as modified at Rational Software, is widely practiced and adopted by industry.
- The critical idea in the Rational Unified Process is *Iterative Development*.

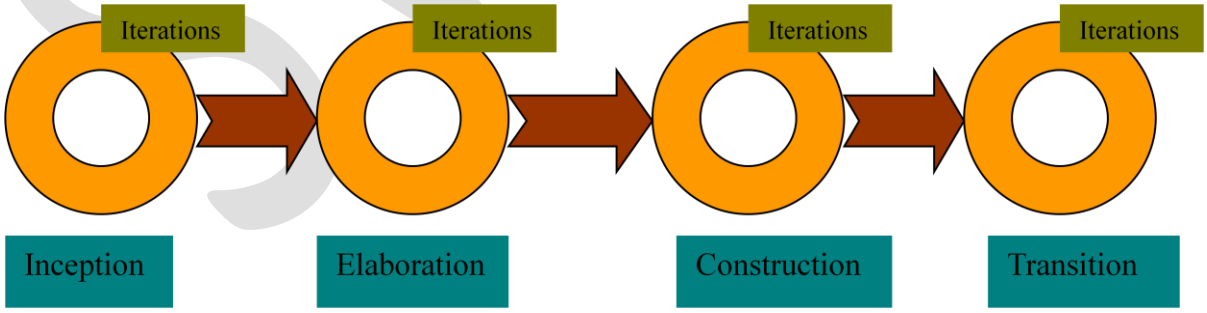
- Iterative Development is successively enlarging and refining a system through multiple iterations, using feedback and adaptation.
- Each iteration will include requirements, analysis, design, and implementation.
- Iterations are *timeboxed*.
- RUP is a complete software-development process framework , developed by Rational Corporation.
- It's an iterative development methodology based upon six industry-proven best practices.
- Processes derived from RUP vary from lightweight—addressing the needs of small projects —to more comprehensive processes addressing the needs of large, possibly distributed project teams.

Phases in RUP

- RUP is divided into four phases, named:
- Inception
- Elaboration
- Construction
- Transition

Iterations

Each phase has iterations, each having the purpose of producing a demonstrable piece of software. The duration of iteration may vary from two weeks or less up to six months.



Inception

- The life-cycle objectives of the project are stated, so that the needs of every stakeholder are considered. Scope and boundary conditions, acceptance criteria and some requirements are established.

Inception - Entry criteria

- The expression of a need, which can take any of the following forms:

- an original vision
- a legacy system
- an RFP (request for proposal)
- the previous generation and a list of enhancements
- some assets (software, know-how, financial assets)
- a conceptual prototype, or a mock-up

Inception – Activities

- **Formulate the scope of the project.**

  Needs of every stakeholder, scope, boundary conditions and acceptance criteria established.

- **Plan and prepare the business case.**

  Define risk mitigation strategy, develop an initial project plan and identify known cost, schedule, and profitability trade-offs.

- **Synthesize candidate architecture.**

  Candidate architecture is picked from various potential architectures

Prepare the project environment.

Inception - Exit criteria

- An initial business case containing at least a clear formulation of the product vision - the core requirements - in terms of functionality, scope, performance, capacity, technology base.
- Success criteria (example: revenue projection).
- An initial risk assessment.

- An estimate of the resources required to complete the elaboration phase.

Elaboration

An analysis is done to determine the risks, stability of vision of what the product is to become, stability of architecture and expenditure of resources

Elaboration - Entry criteria

- The products and artifacts described in the exit criteria of the previous phase.
- The plan approved by the project management, and funding authority, and the resources required for the elaboration phase have been allocated.

Elaboration – Activities

- **Define the architecture.**

  Project plan is defined. The process, infrastructure and development environment are described.

- **Validate the architecture.**
- **Baseline the architecture.**

  To provide a stable basis for the bulk of the design and implementation effort in the construction phase.

Elaboration - Exit criteria

- A detailed software development plan, with an updated risk assessment, a management plan, a staffing plan, a phase plan showing the number and contents of the iteration , an iteration plan, and a test plan
- The development environment and other tools
- A baseline vision, in the form of a set of evaluation criteria for the final product
- A domain analysis model, sufficient to be able to call the corresponding architecture 'complete'.
- An executable architecture baseline.

Construction

The Construction phase is a manufacturing process. It emphasizes managing resources and controlling operations to optimize costs, schedules and quality. This phase is broken into several iterations.

Construction - Entry criteria

- The product and artifacts of the previous iteration. The iteration plan must state the iteration specific goals
- Risks being mitigated during this iteration.
- Defects being fixed during the iteration.

Construction – Activities

- **Develop and test components**.

  Components required satisfying the use cases, scenarios, and other functionality for the iteration are built. Unit and integration tests are done on Components.

- **Manage resources and control process**.
- **Assess the iteration**

  Satisfaction of the goal of iteration is determined.

Construction - Exit Criteria

- The same products and artifacts, updated, plus:
- A release description document, which captures the results of an iteration
- Test cases and results of the tests conducted on the products,
- An iteration plan, detailing the next iteration

Objective measurable evaluation criteria for assessing the results of the next iteration(s).

Transition

The transition phase is the phase where the product is put in the hands of its end users. It involves issues of marketing, packaging, installing, configuring, supporting the user-community, making corrections, etc.

Transition - Entry criteria

- The product and artifacts of the previous iteration, and in particular a software product sufficiently mature to be put into the hands of its users.

Transition – Activities

- **Test the product deliverable in a customer environment**.

- **Fine tune the product based upon customer feedback**
- **Deliver the final product to the end user**
- **Finalize end-user support material**

Transition - Exit criteria

- An update of some of the previous documents, as necessary, the plan being replaced by a "post-mortem" analysis of the performance of the project relative to its original and revised success criteria;
- A brief inventory of the organization's new assets as a result this cycle.

UML structural diagrams

UML structural diagrams are categorized as follows: class diagram, object diagram, component diagram, and deployment diagram.

Class Diagram

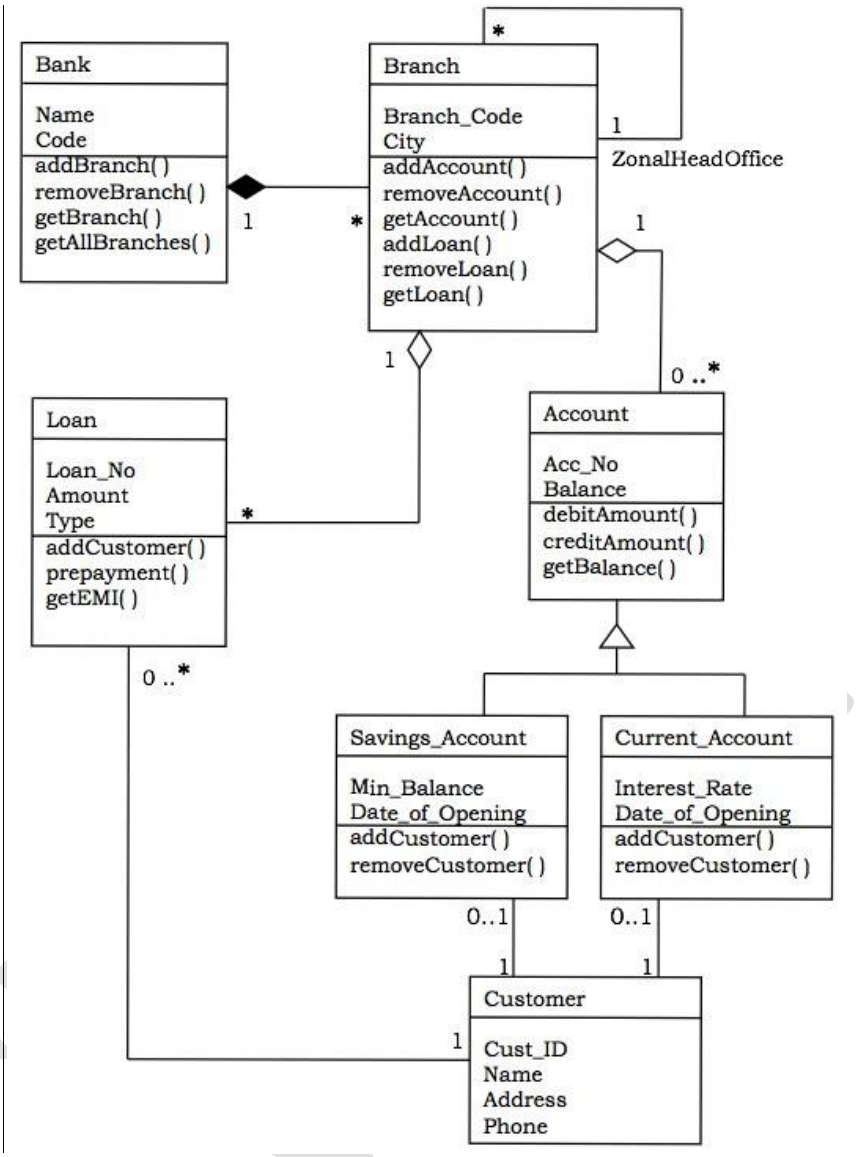
A class diagram models the static view of a system. It comprises of the classes, interfaces, and collaborations of a system; and the relationships between them.

Class Diagram of a System

Let us consider a simplified Banking System.

A bank has many branches. In each zone, one branch is designated as the zonal head office that supervises the other branches in that zone. Each branch can have multiple accounts and loans. An account may be either a savings account or a current account. A customer may open both a savings account and a current account. However, a customer must not have more than one savings account or current account. A customer may also procure loans from the bank.

The following figure shows the corresponding class diagram.



Classes in the system:

Bank, Branch, Account, Savings Account, Current Account, Loan, and Customer.

Relationships:

- **A Bank "has–a" number of Branches** : composition, one–to–many
- **A Branch with role Zonal Head Office supervises other Branches** : unary association, one–to-many
- **A Branch "has–a" number of accounts** : aggregation, one–to–many

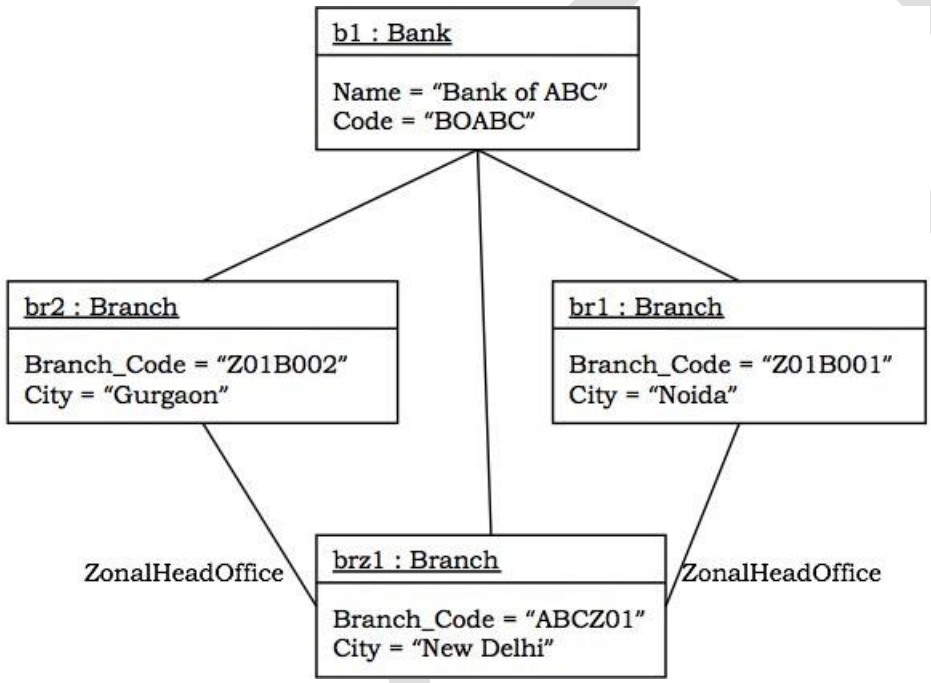
From the class Account, two classes have inherited, namely, Savings Account and Current Account.

- **A Customer can have one Current Account** : association, one–to–one
- **A Customer can have one Savings Account** : association, one–to–one
- **A Branch "has–a" number of Loans** : aggregation, one–to–many
- **A Customer can take many loans** : association, one–to–many

Object Diagram

An object diagram models a group of objects and their links at a point of time. It shows the instances of the things in a class diagram. Object diagram is the static part of an interaction diagram.

Example : The following figure shows an object diagram of a portion of the class diagram of the Banking System.



Component Diagram

Component diagrams show the organization and dependencies among a group of components.

Component diagrams comprise of:

- Components
- Interfaces
- Relationships

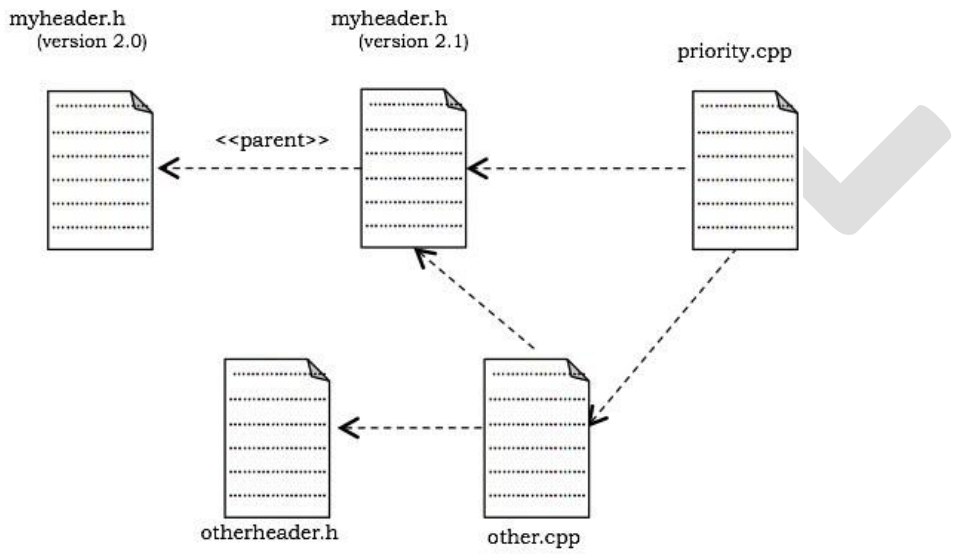
- Packages and Subsystems (optional)

Component diagrams are used for:

- constructing systems through forward and reverse engineering.
- modeling configuration management of source code files while developing a system using an object-oriented programming language.
- representing schemas in modeling databases.
- modeling behaviors of dynamic systems.

Example

The following figure shows a component diagram to model a system's source code that is developed using C++. It shows four source code files, namely, myheader.h, otherheader.h, priority.cpp, and other.cpp. Two versions of myheader.h are shown, tracing from the recent version to its ancestor. The file priority.cpp has compilation dependency on other.cpp. The file other.cpp has compilation dependency on otherheader.h.



Deployment Diagram

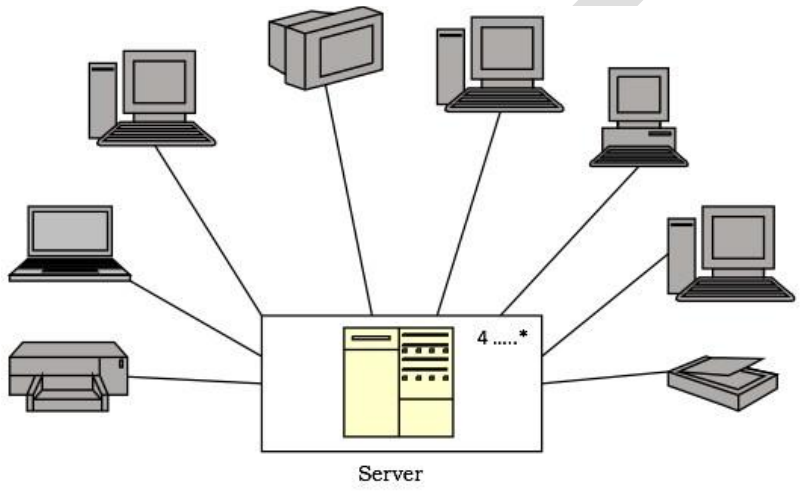
A deployment diagram puts emphasis on the configuration of runtime processing nodes and their components that live on them. They are commonly comprised of nodes and dependencies, or associations between the nodes.

Deployment diagrams are used to:

- model devices in embedded systems that typically comprise of software-intensive collection of hardware.
- represent the topologies of client/server systems.
- model fully distributed systems.

Example

The following figure shows the topology of a computer system that follows client/server architecture. The figure illustrates a node stereotyped as server that comprises of processors. The figure indicates that four or more servers are deployed at the system. Connected to the server are the client nodes, where each node represents a terminal device such as workstation, laptop, scanner, or printer. The nodes are represented using icons that clearly depict the real-world equivalent.



UML behavioral diagrams

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

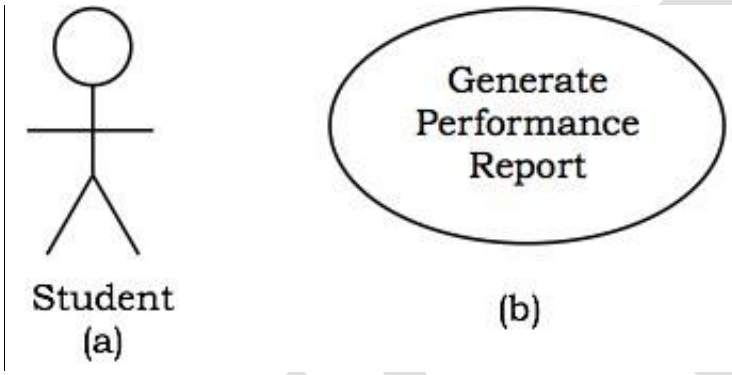
(a) Use case

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.

(b) 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.



(c) 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

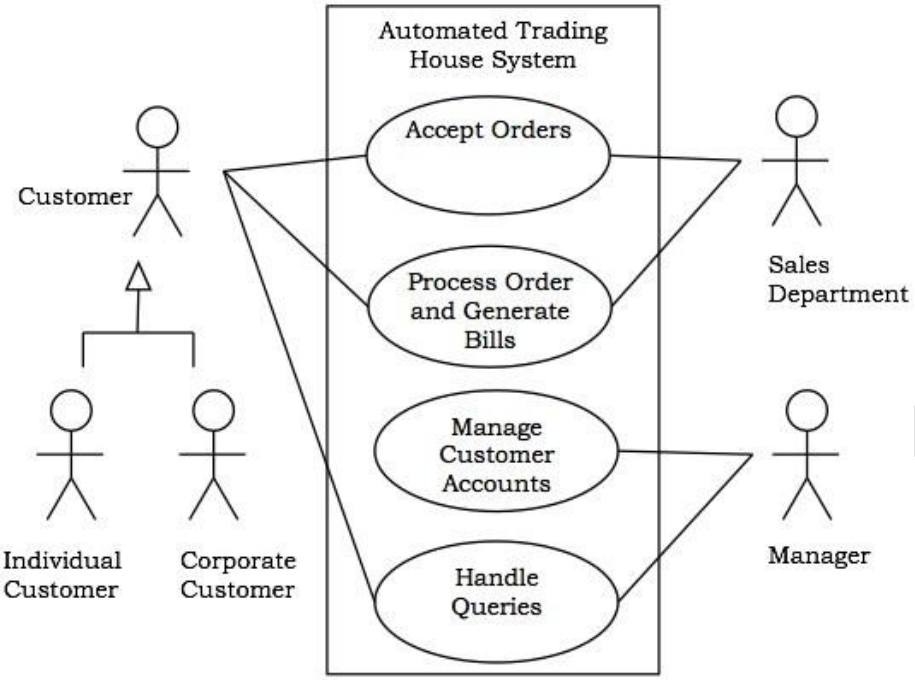
Use case diagrams are used:

- 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:

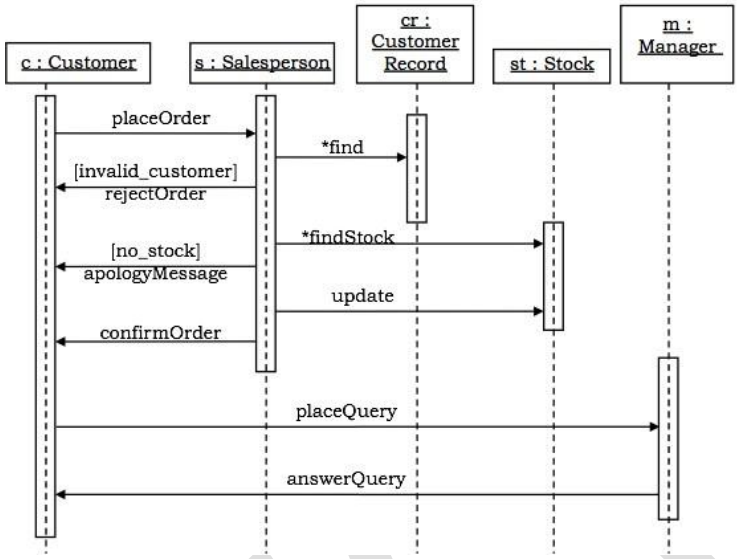
- the control flow by time ordering using sequence diagrams.
- the control flow of organization using collaboration diagrams.

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.

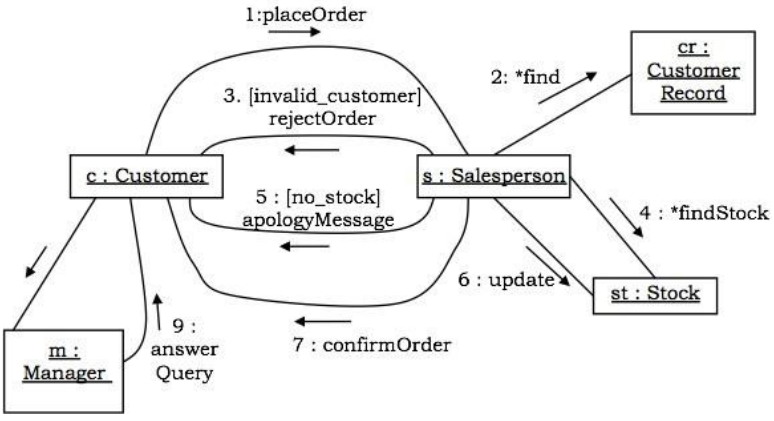


Collaboration Diagrams

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.

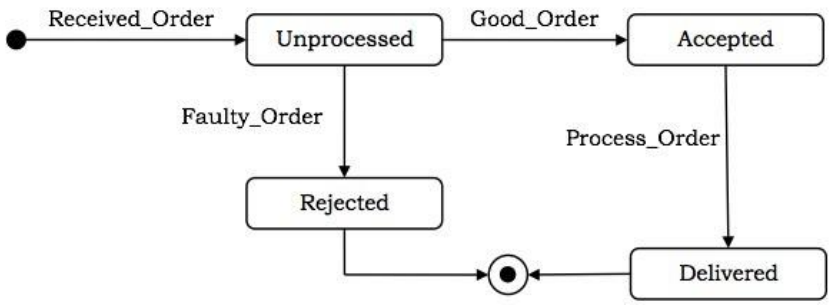
State–Chart Diagrams comprise of:

- 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.