Unit-6 Introduction to Object-Oriented Development

The term object oriented describes the system as a collection of discrete objects that incorporate both data structure and behavior. It is a way of thinking about problems using models organized around real world concepts. Object oriented analysis and design promote the better understanding of requirements, cleaner design and more maintainable system.

Object-oriented analysis is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain. Object-oriented design is a method of design encompassing the process of object-oriented decomposition and a notation for depicting logical and physical as well as static and dynamic models of the system under design.

Object-Oriented Development Life Cycle (OODLC)

In contrast to traditional SDLC, the Object-Oriented System Development Life Cycle is more like an Onion than a waterfall. In the early stages (or core) of development, the model we build is abstract, focusing on the external qualities of application system. As the model evolves, it becomes more and more detailed, shifting the focus on how the system will be build and how it should function. The emphasis in modeling should be an analysis and design, focusing on front-end conceptual issue, rather than back end implementation issues, which unnecessarily restrict design choice.

Object oriented cycle is like an onion, evolving from abstract to detailed, from external qualities to system architecture and algorithms. The object oriented development life cycle consists of progressively developing an object representation through three phases - analysis, design, and implementation.

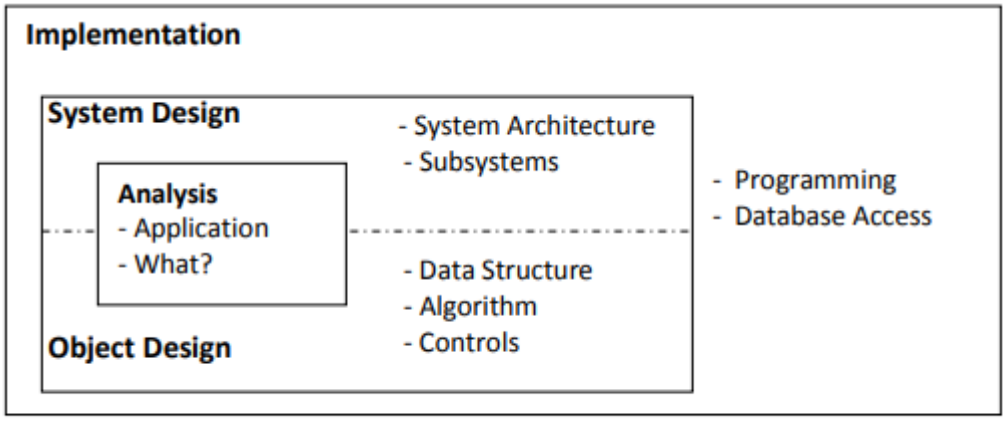


Figure: Phases of Object-Oriented System Development Life Cycle

The Process of Object-Oriented Analysis and Design

Analysis Phase

In the analysis phase, a model of the real world application is developed showing its important properties. It abstracts concepts from the application domain and describes what the intended system must do rather than how it will be done. The model specifies the functional behavior of the system, independent of any implementation details.

Design Phase

The requirement model is translated into a more detailed model that will be essential up to the implementation of the system. In the design phase, we define how the application-oriented analysis model will realized in the implementation environment. All strategic design decision – such as how the DBMS is to be incorporated, how process communication and error handling are to be achieved, what component libraries are to be reused or made. These decisions are incorporated into the design model that adapts to the implementation environment.

- **System Design** During System Design, one proposes an overall system architecture, which organizes the system into components called sub systems and provides the context to make decisions such as identifying concurrency, allocation of sub system to processors and task, handling access to global resources, selecting the implementation of control in software and more.
- **Object Design** During Object Design, one builds a design model by adding implementation details such as restructuring classes for efficiency, internal data structures and algorithms to implement each class, implementation of control, implementation of associations and packaging into physical modules

Implementation Phase

The design phase is followed by the implementation phase. In this phase, one implements the design using a programming language and / or a database management system.

Basic Characteristics of Object-Oriented System

Objects

Objects are the basic run time entities in an object-oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program has to handle. They may also represent user-defined data such as vectors, time and lists. Programming problem is analyzed in term of objects and the nature of communication between them. Program objects should be chosen such that they match closely with the real-world objects. Objects take up space in the memory and have an associated address like a record in Pascal, or a structure in c. When a program is executed, the objects interact by sending messages to one another.

Classes

We just mentioned that objects contain data, and code to manipulate that data. The entire set of data and code of an object can be made a user-defined data type with the help of class. In fact, objects are variables of the type class. Once a class has been defined, we can create any number of objects belonging to that class. Each object is associated with the data of type class with which they are created. A class is thus a collection of objects similar types. For examples, Mango, Apple and orange members of class fruit. Classes are user-defined that types and behave like the built-in types of a programming language. The syntax used to create an object is not different then the syntax used to create an integer object in C. If fruit has been defines as a class, then the statement

Fruit Mango;

Will create an object mango belonging to the class fruit.

Data Abstraction and Encapsulation

The wrapping up of data and function into a single unit (called class) is known as encapsulation. Data and encapsulation is the most striking feature of a class. The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it. These functions provide the interface between the object's data and the program. This insulation of the data from direct access by the program is called data hiding or information hiding. Abstraction

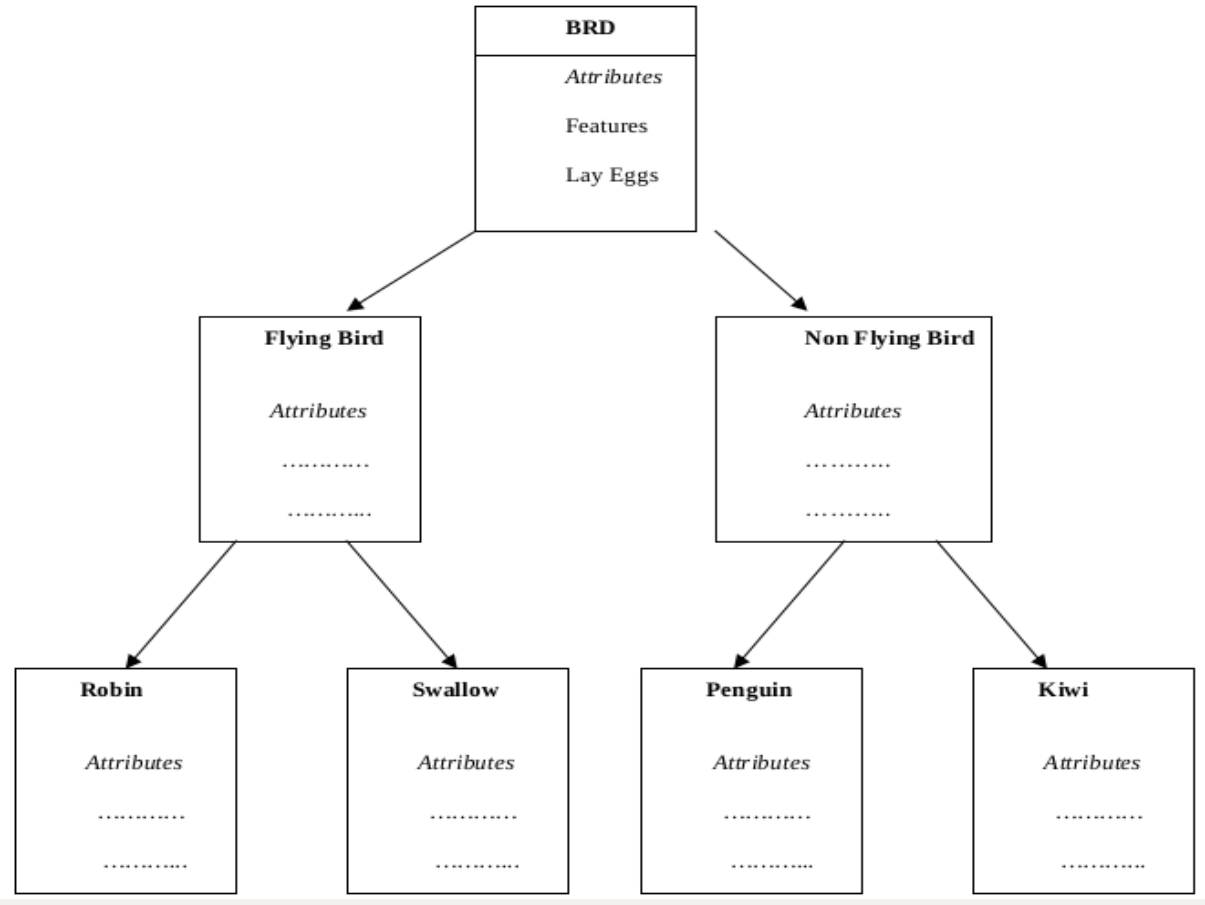
refers to the act of representing essential features without including the background details or explanation. Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, wait, and cost, and function operate on these attributes. They encapsulate all the essential properties of the object that are to be created. The attributes are some time called data members because they hold information. The functions that operate on these data are sometimes called methods or member function.

Inheritance

Inheritance is the process by which objects of one class acquired the properties of objects of another classes. It supports the concept of hierarchical classification. For example, the bird, 'robin' is a part of class 'flying bird' which is again a part of the class 'bird'. The principal behind this sort of division is that each derived class shares common characteristics with the

inheritance mechanism is that it allows the programmer to reuse a class i.e almost, but not exactly, what he wants, and to tailor the class in such a way that it does not introduced any undesirable side-effects into the rest of classes.

Fig. 1.6 Property inheritances



Polymorphism

Polymorphism is another important OOP concept. Polymorphism, a Greek term, means the ability to take more than on form. An operation may exhibit different behavior is different instances. The behavior depends upon the types of data used in the operation. For example, consider the operation of addition. For two numbers, the operation will generate a sum. If the operands are strings, then the operation would produce a third string by concatenation. The process of making an operator to exhibit different behaviors in different instances is known as operator overloading. Fig. below illustrates that a single function name can be used to handle different number and different types of argument. This is something similar to a particular word having several different meanings depending upon the context. Using a single function name to perform different type of task is known as function overloading.

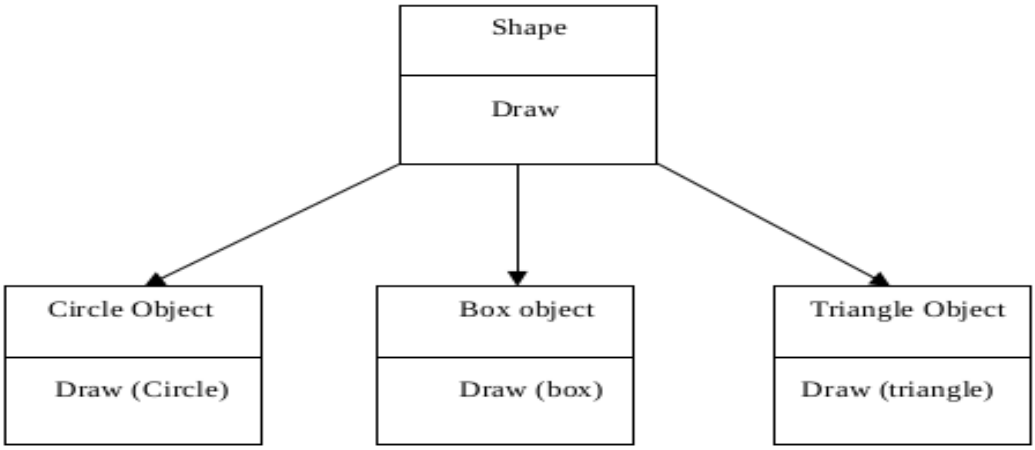


Fig. 1.7 Polymorphism

Polymorphism plays an important role in allowing objects having structures to share the same external interface. This means that a operations may be accessed in the same manner even though specific with each operation may differ. Polymorphism is extensively used inheritance.

Dynamic Binding

Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding means that the code associated with a given procedure call is not known until

the time of the call at run time. It is associated with polymorphism and inheritance. A function call associated with a polymorphic reference depends on the dynamic type of that reference. Consider the procedure "draw" in fig. above by inheritance; every object will have this procedure. Its algorithm is, however, unique to each object and so the draw procedure will be redefined in each class that defines the object. At run-time, the code matching the object under current reference will be called.

Benefits of Object Oriented Modeling

- The ability to tackle more challenging problem domains.
- Improved communication among users, analysts, designers and programmers.
- Increased consistency among analysis, design and programming activities.
- Explicit representation of commonality among system components.
- Robustness of systems.
- Reusability of analysis, design and programming results.
- Increased consistency among all the models developed during Object-Oriented analysis design and programming.

Unified Modeling Language (UML)

UML is a language for specifying, visualizing, constructing and documenting all the artifacts of software systems as well as for business modeling. It is largely based on the object-oriented paradigm and is an essential tool for developing robust and maintainable software systems.

The UML allows representing multiple views of a system by using a variety of graphical diagrams. The underlying model integrates those views so that the system can be analyzed and implemented in a complete and consistent fashion. UML includes many diagrams that represent different perspectives of a system.

The primary goals of the Object-Oriented Design in UML as follows:

- Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.

- Provide extensibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development processes.
- Provide a formal basis for understanding the modeling language.
- Encourage the growth of the OO tools market.
- Support higher-level development concepts such as collaborations, frameworks, patterns and components.
- Integrate best practices.

We prepare UML diagrams to understand the system in a better and simple way, A single diagram is not enough to cover all the aspects of the system. UML defines various kinds of diagrams to cover most of the aspects of a system.

There are two broad categories of diagrams and they are again divided into subcategories-

- Structural Diagrams
- Behavioral Diagrams

Structural Diagrams

The structural diagrams represent the static aspect of the system. These static aspects represent those parts of a diagram, which forms the main structure and are therefore stable. Structural diagram assist in understanding and communicating the elements that make up a system and the functionality the system provides.

These static parts are represented by classes, interfaces, objects, components, and nodes. The four structural diagrams are

- Class diagram
- Object diagram
- Component diagram
- Deployment diagram

Class Diagram

A Class diagram is used to model the static design view of the system. A class diagram shows the static aspect of a system: classes, their internal structure and the relationship in which they participate. In UML, a class is represented by a rectangle with three compartments separated by horizontal lines. The class name appears in the top compartment, the list of attributes in the middle compartment, and the list of operations in the bottom compartment of the box.

The purpose of class diagram is to model the static view of an application. Class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction.

The purpose of the class diagram can be summarized as:

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.
- Forward and reverse engineering.

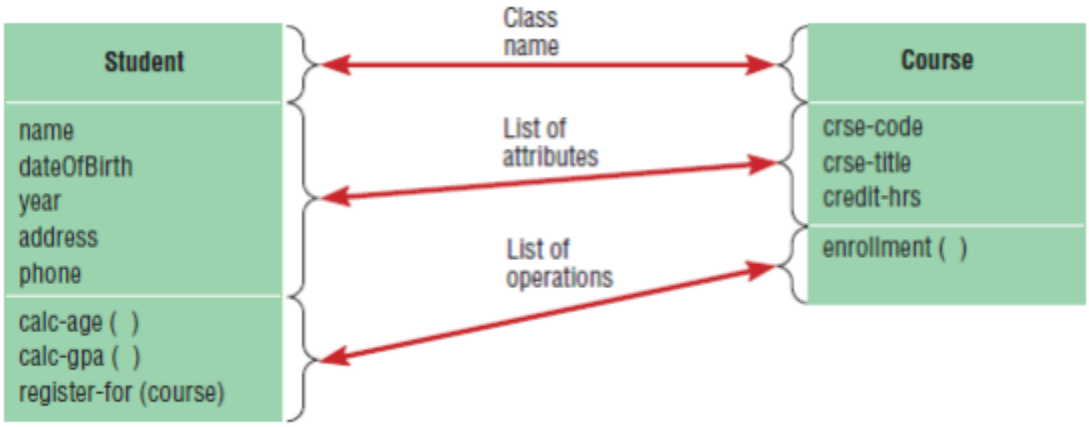


Figure: Class diagram showing two classes

Object diagram

An object diagram shows a set of objects and their relationships. It is also known as instance diagram. It is a graph of instances that are compatible with a given class diagram. It shows the static snapshots of element instances found in class diagrams. In an object diagram, an object is represented as a rectangle with two compartments. The names of the object and its class are underlined and shown on top compartment using the following syntax:

ObjectName: ClassName

Figure: Object diagram with two instances.

The purpose of the object diagram can be summarized as:

- Forward and reverse engineering.
- Object relationships of a system
- Static view of an interaction.
- Understand object behaviour and their relationship from practical perspective

Component diagram

Component diagrams represent a set of components and their relationships. These components consist of classes, interfaces, or collaborations. This diagram is used to represent the how the physical components in a system have been organized. We use them for modeling implementation details. Component Diagrams depict the structural relationship between software system elements and help us in understanding if functional requirements have been covered by planned development. Component Diagrams become essential to use when we design and build complex systems. Interfaces are used by components of the system to communicate with each

other. Component diagrams represent the implementation view of a system. Component diagram also known as implementation diagrams, depict the implementation of a system.

The purpose of the component diagram can be summarized as

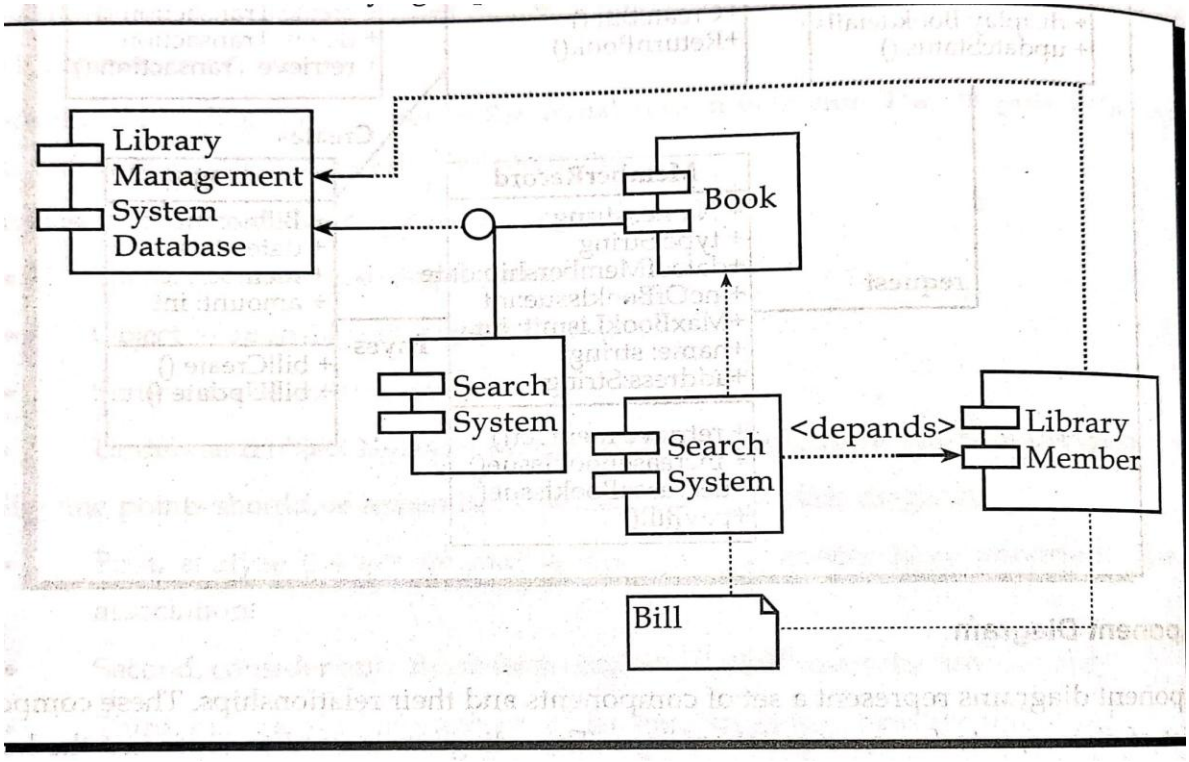
- Visualize the components of a system.
- Construct executables by using forward and reverse engineering.
- Describe the organization and relationships of the components.

Before drawing a component diagram, the following artifacts are to be identified clearly-

- Files used in the system.
- Libraries and other artifacts relevant to the application.
- Relationships among the artifacts.

After identifying the artifacts, the following points need to be kept in mind.

- Use a meaningful name to identify the component for which the diagram is to be drawn.
- Prepare a mental layout before producing the using tools.
- Use notes for clarifying important points.



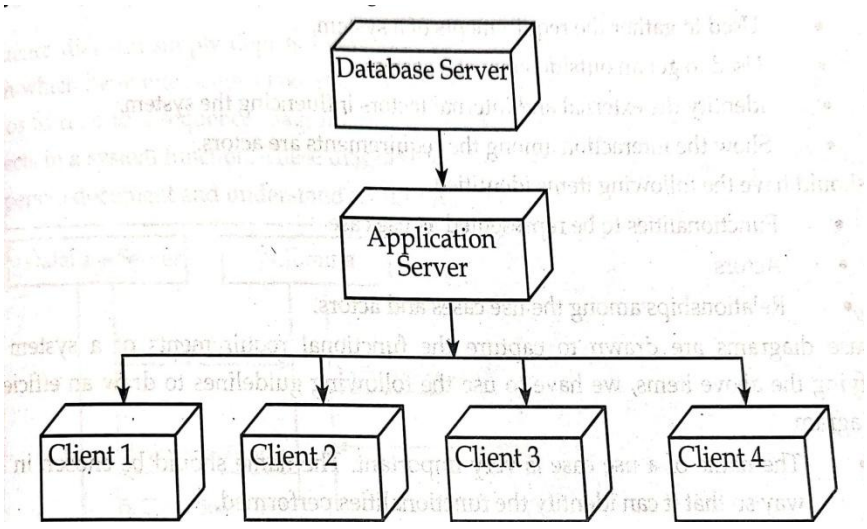
Deployment Diagram

Deployment Diagrams, also known as the implementation diagram, depict the implementation environment of system are used to represent system hardware and its software. It tells us what primarily Deployment Diagrams, also known as the implementation diagram, depict the implementation hardware components exist and what software components run on them. They are used when a software is being used, distributed or deployed over multiple machines with different configurations. Deployment diagrams have set of nodes and their relationships. These nodes are physical entities where the components are deployed. Deployment diagrams are used for visualizing the deployment view of a system. This is generally used by the deployment team.

Deployment diagrams are used to visualize the topology of the physical components of system, where the software components are deployed. Deployment diagrams are used to describe the static deployment view of a system. Deployment diagrams consist of nodes and their relationships.

The purpose of deployment diagrams can be described as:

- Visualize the hardware topology of a system.
- Describe the hardware components used to deploy software components.
- Describe the runtime processing nodes.



Behavior Diagrams

Any system can have two aspects, static and dynamic. So, a model is considered as complete when both the aspects are fully covered. Behavioral diagrams basically capture the dynamic aspect of a system. Dynamic aspect can be further described as the changing/moving parts of a system. Behavioral diagram assist in understanding and communicating how elements interact and collaborate to provide the functionality of a system.

UML has the following five types of behavioral diagrams

- Use case diagram
- Sequence diagram
- Collaboration diagram
- Statechart diagram
- Activity diagram

Use Case Diagram

A use case diagram is a dynamic or behavior diagram in UML (Unified Modeling Language). Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. In this context, a "system" is something being developed or operated, such as a web site. The "actors" are people or entities operating under defined roles within the system.

Basic Use Case Diagram Symbols and Notations

System

Draw your system's boundaries using a rectangle that contains use cases. Place actors outside the system's boundaries.

System

UseCase

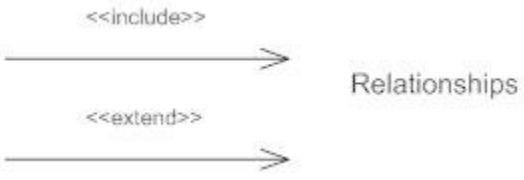
Draw use cases using ovals. Label the ovals with verbs that represent the system's functions.

Actors

Actors are the users of a system. When one system is the actor of another system, label the actor system with the actor stereotype.

Relationships

Illustrate relationships between an actor and a use case with a simple line. For relationships among use cases, use arrows labeled either "uses" or "extends." A "uses" relationship indicates that one use case is needed by another in order to perform a task. An "extends" relationship indicates alternative options under a certain use case.



Example:

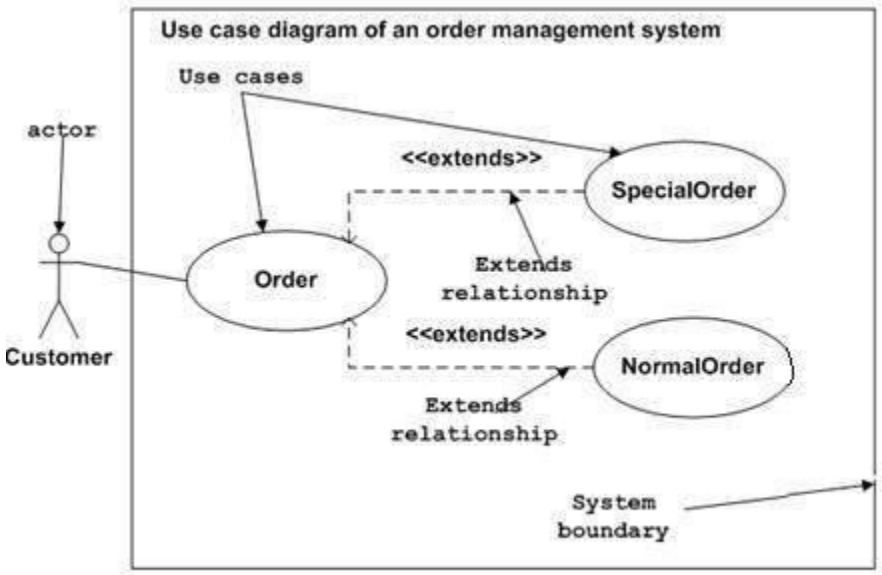


Figure: Sample Use Case diagram

Sequence Diagram

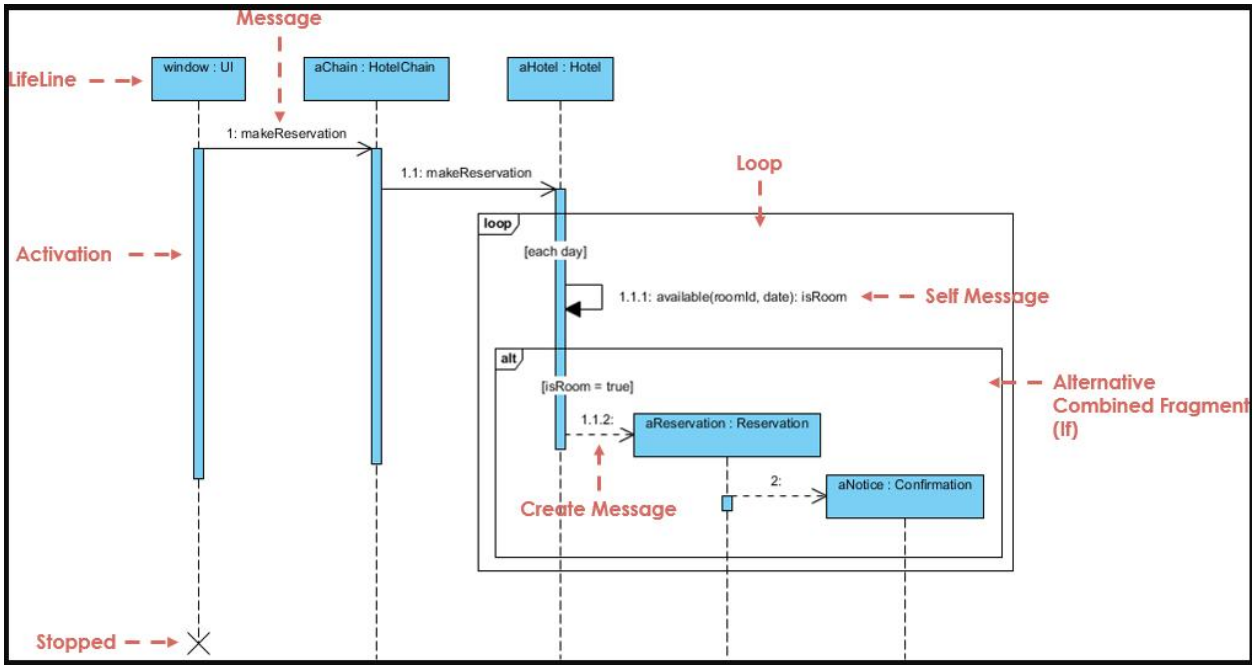
A sequence diagram is a type of interaction diagram because it describes how and in what order a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process. Sequence diagrams are sometimes known as event diagrams or event scenarios.

Benefits of sequence diagrams

Sequence diagrams can be useful references for businesses and other organizations. Try drawing a sequence diagram to:

- Represent the details of a UML use case.
- Model the logic of a sophisticated procedure, function, or operation.
- See how objects and components interact with each other to complete a process.
- Plan and understand the detailed functionality of an existing or future scenario

Below is a sequence diagram for making a **hotel reservation**. The object initiating the sequence of messages is a Reservation window



Sequence Diagram notation descriptions

Lifeline

A lifeline represents an individual participant in the Interaction.

Activations

A thin rectangle on a lifeline represents the period during which an element is performing an operation.

Call Message

A message defines a particular communication between Lifelines of an Interaction. Call message is a kind of message that represents an invocation of operation of target lifeline.

Return Message

A message defines a particular communication between Lifelines of an Interaction. Return message is a kind of message that represents the pass of information back to the caller of a corresponded former message.

Self Message

A message defines a particular communication between Lifelines of an Interaction. Self message is a kind of message that represents the invocation of message of the same lifeline.

Create Message

A message defines a particular communication between Lifelines of an Interaction. Create message is a kind of message that represents the instantiation of (target) lifeline.

Destroy Message

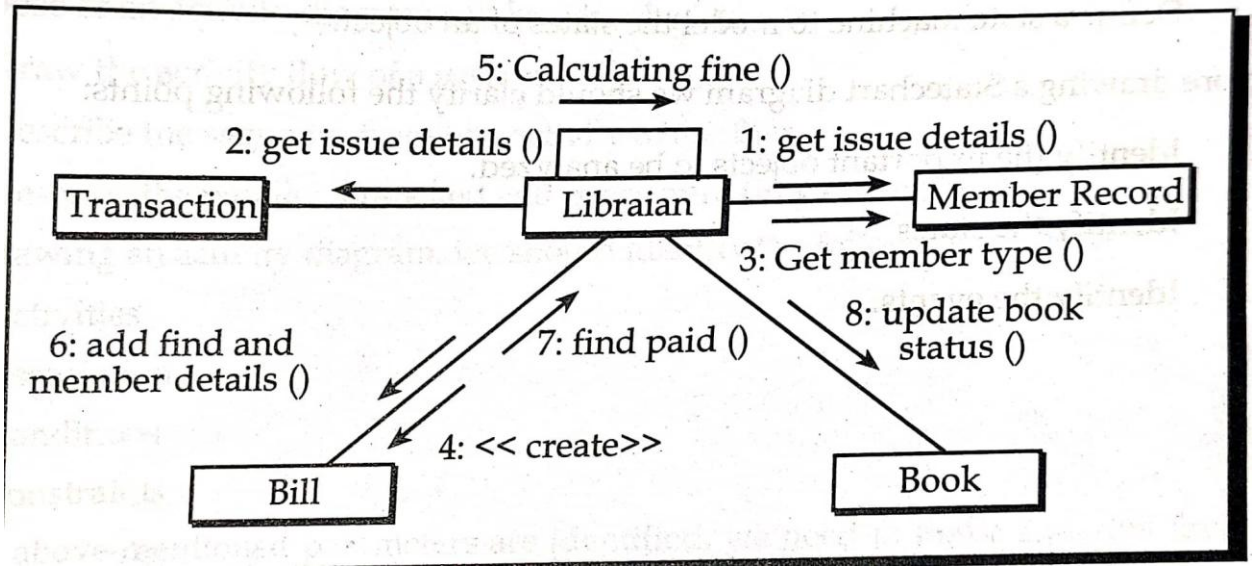
A message defines a particular communication between Lifelines of an Interaction. Destroy message is a kind of message that represents the request of destroying the lifecycle of target lifeline.

Duration Message

A message defines a particular communication between Lifelines of an Interaction. Duration message shows the distance between two time instants for a message invocation.

Collaboration Diagram

Collaboration diagram is another form of interaction diagram. It represents the structural organization of a system and the messages sent/received. Structural organization consists of objects and links. The purpose of collaboration diagram is similar to sequence diagram. However, the specific purpose of collaboration diagram is to visualize the organization of objects and their interaction.



Statechart diagram

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. 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.

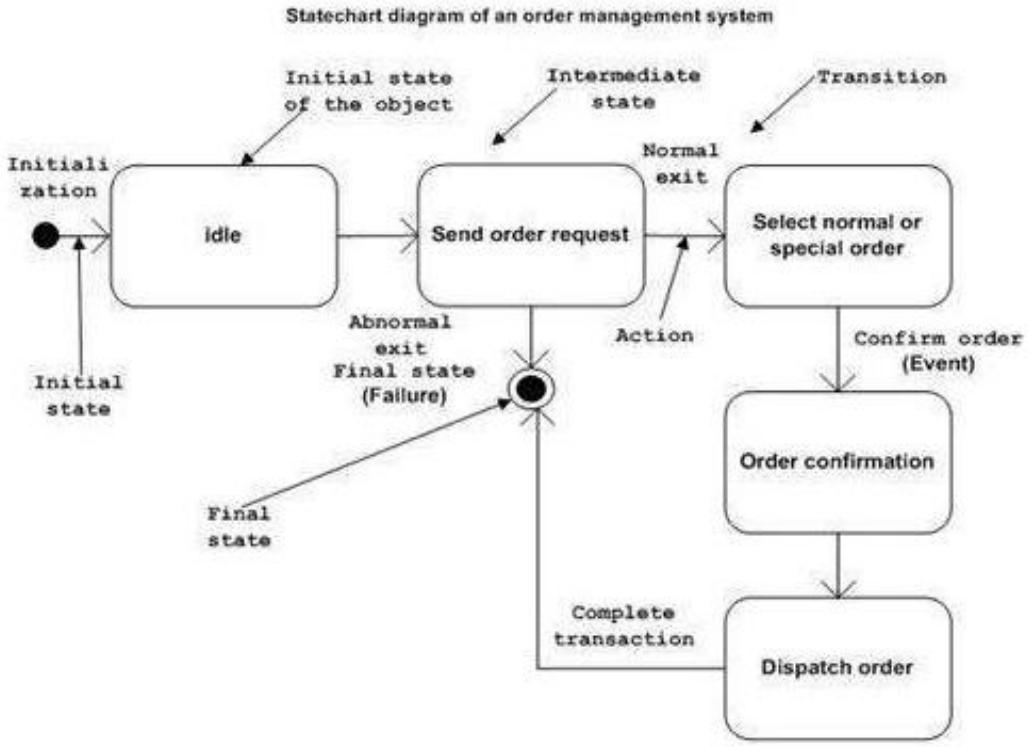
Following are the main purposes of using Statechart diagrams :

- 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

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:



Activity diagram

Activity diagram is another important diagram in UML to describe the dynamic aspects of the system. Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system.

The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc.

The purpose of an activity diagram can be described as:

- Draw the activity flow of a system.
- Describe the sequence from one activity to another.
- Describe the parallel, branched and concurrent flow of the system

Following is an example of an activity diagram for order management system. In the diagram, four activities are identified which are associated with conditions. One important point should be

clearly understood that an activity diagram cannot be exactly matched with the code. The activity diagram is made to understand the flow of activities and is mainly used by the business users

Following diagram is drawn with the four main activities:

- Send order by the customer
- Receipt of the order
- Confirm the order
- Dispatch the order

After receiving the order request, condition checks are performed to check if it is normal or special order. After the type of order is identified, dispatch activity is performed and that is marked as the termination of the process.