UNIFIED MODELING LANGUAGE AND ITS RELATED DIAGRAMS

Definition - *is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems.*

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing object oriented software and the software development process. The UML uses **mostly graphical notations to express the design of software projects**. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

Each UML diagram is designed to let developers and customers view a software system from a different perspective and in varying degrees of abstraction. UML diagrams commonly created in visual modeling tools include:

Use Case Diagram displays the relationship among actors and use cases.

Class Diagram models class structure and contents using design elements such as classes, packages and objects. It also displays relationships such as containment, inheritance, associations and others.

Interaction Diagrams

- Sequence Diagram displays the time sequence of the objects participating in the interaction. This consists of the vertical dimension (time) and horizontal dimension (different objects).

  - Collaboration Diagram displays an interaction organized around the objects and their links to one another. Numbers are used to show the sequence of messages.

State Diagram; displays the sequences of states that an object of an interaction goes through during its life in response to received stimuli, together with its responses and actions.

Activity Diagram; displays a special state diagram where most of the states are action states and most of the transitions are triggered by

completion of the actions in the source states. This diagram focuses on flows driven by internal processing.

Physical Diagrams

- **Component Diagram** displays the high level packaged structure of the code itself. Dependencies among components are shown, including source code components, binary code components, and executable components. Some components exist at compile time, at link time, at run times well as at more than one time.
- **Deployment Diagram** displays the configuration of run-time processing elements and the software components, processes, and objects that live on them. Software component instances represent run-time manifestations of code units.

Use Case Diagrams

Bank ATM Use Cases: An automated teller machine (**ATM**) or the automatic banking machine (**ABM**) is banking subsystem (subject) that provides bank **customers** with access to financial transactions in a public space without the need for a cashier, clerk or bank teller.

Customer (actor) uses bank ATM to **check balances** of his/her bank accounts, **deposit funds**, **withdraw cash** and/or **transfer funds (use cases). ATM Technician** provides **maintenance** and **repairs**. All these use cases also involve **Bank** actor whether it is related to customer transactions or to the ATM servicing.
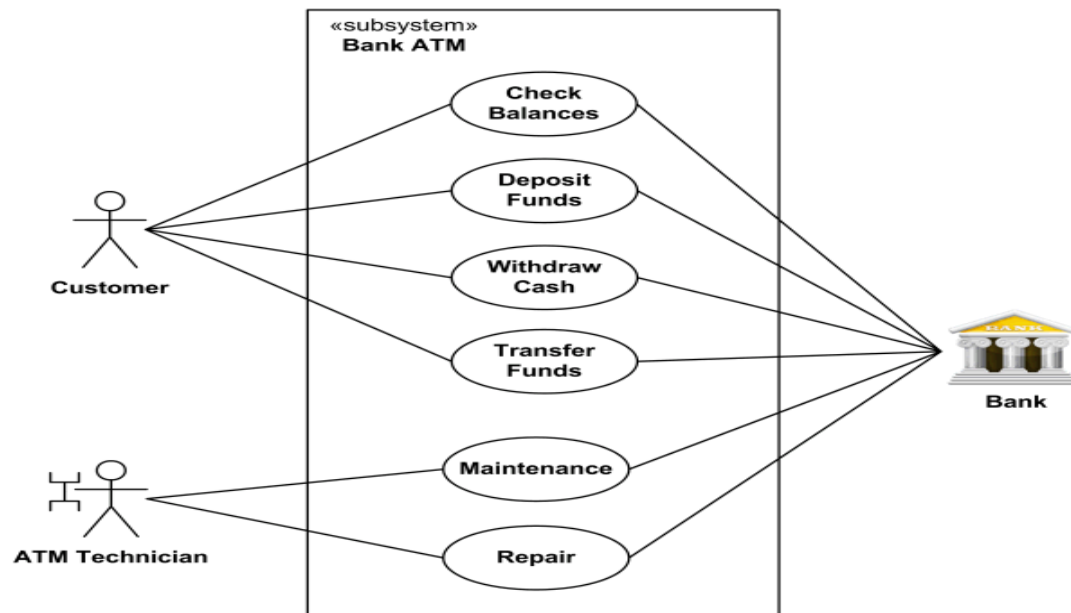
Hospital Use Cases

Hospital Management System is a large system including several subsystems or modules providing variety of functions.

Hospital Reception subsystem or module supports some of the many job duties of hospital receptionist. Receptionist schedules patient's appointments and admission to the hospital, collects information from patient upon patient's arrival and/or by phone. For the patient that will stay in the hospital (inpatient) s/he should have a bed allotted in a ward. Receptionists might also receive patient's payments, record them in a database and provide receipts, file insurance claims and medical reports.
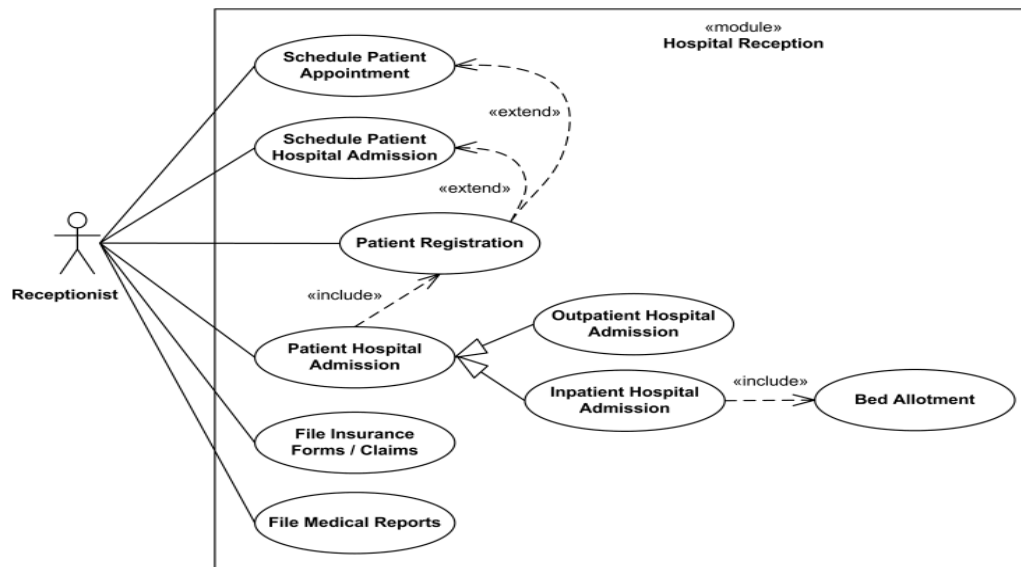
Class Diagrams

Class diagrams are widely used to describe the types of objects in a system and their relationships. Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagrams can be used in various software development phases.

Class diagrams describe three different perspectives when designing a system: *conceptual, specification, and implementation.*

1. Conceptual perspective: Conceptual diagrams are describing things in the real world. You should draw a diagram that represents the concepts in the domain under study. These concepts related to class and it is always *language-independent*.

2. Specification perspective: Specification perspective describes software abstractions or components with specifications and interfaces. However, it does not give any commitment to specific implementation.

3. Implementation perspective: This type of class diagrams is used for implementations in a specific language or application. Implementation perspective, use for software implementation.

These perspectives become evident as the diagram is created and help solidify the design. Classes are composed of three things: a name, attributes, and operations. Below is an example of a class.

Class diagrams also display **relationships** such as containment, inheritance, associations and others. Below is an example of an *associative relationship:*
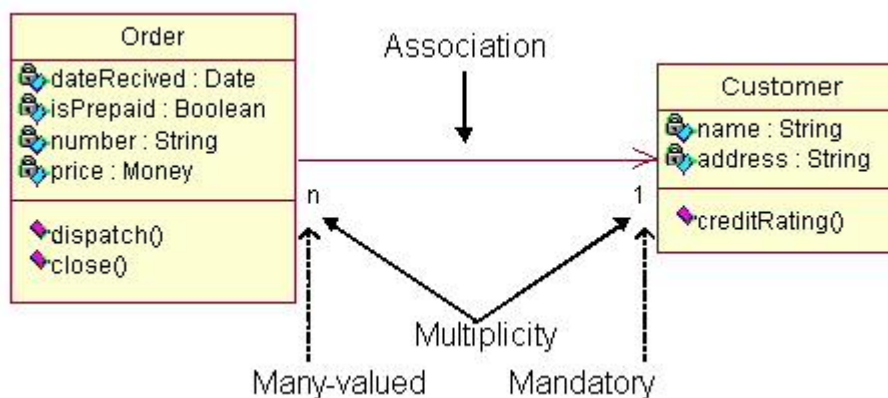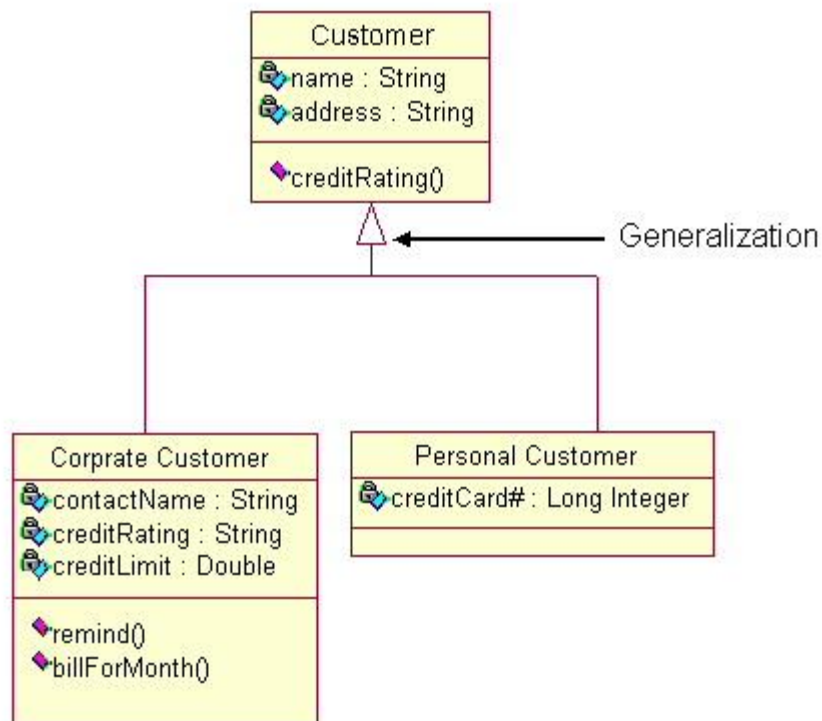

The association relationship is the most common relationship in a class diagram. The association shows the relationship between instances of classes. For example, the class Order is associated with the class Customer.

-The **multiplicity** of the association denotes the number of objects that can participate in the relationship. For example, an Order object can be associated to only one customer, but a customer can be associated to many orders.

Another common relationship in class diagrams is a *generalization.* A generalization is used when two classes are similar, but have some differences.

Example:

Customer
name : String
address : String
creditRating()

Generalization

Corprate Customer
contactName : String
creditRating : String
creditLimit : Double

remind()
billForMonth()
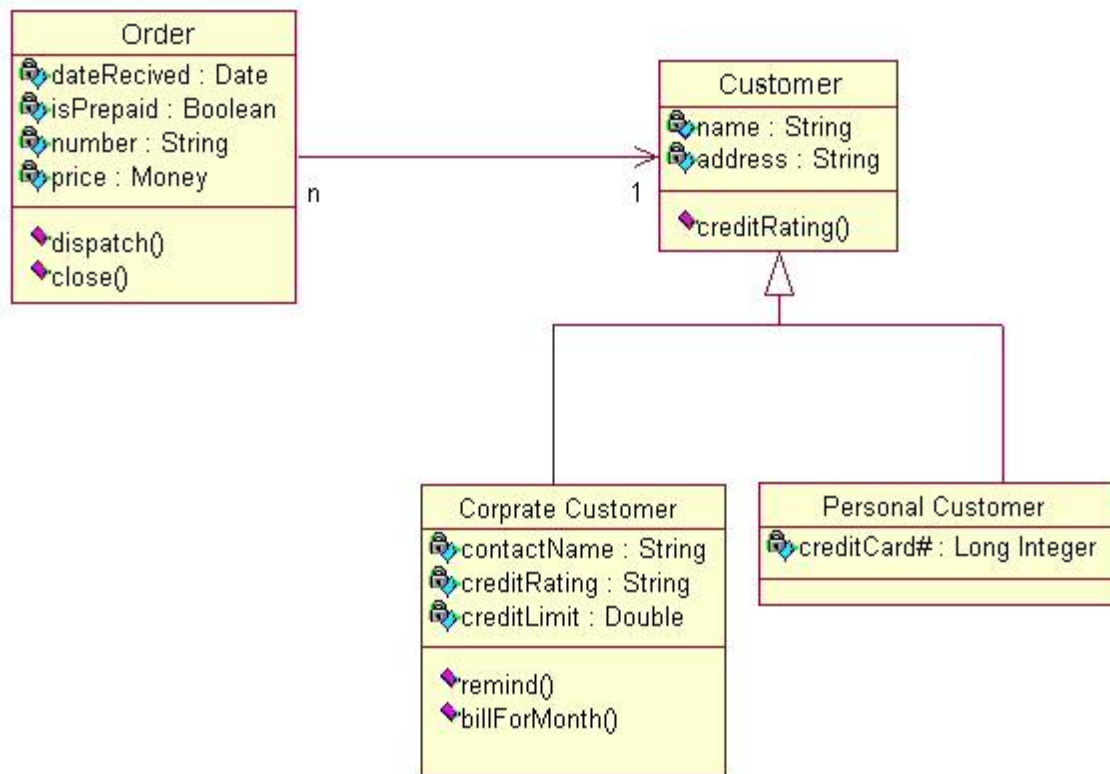
Personal Customer
creditCard# : Long Integer

In this example, the classes Corporate Customer and Personal Customer have some similarities such as name and address, but each class has some of its own attributes and operations. The class Customer is a general form of both the Corporate Customer and Personal Customer classes. This allows the designers to just use the Customer class for modules and do not require in-depth representation of each type of customer.

When to Use: Class Diagrams

Class diagrams are used in nearly all Object Oriented software designs. Use them to describe the Classes of the system and their relationships to each other.

How to Draw: Class Diagrams

When designing classes consider what attributes and operations it will have. Then try to determine how instances of the classes will interact with each other. These are the very first steps of many in developing a class diagram. However, using just these basic techniques one can develop a complete view of the software system.

Order
dateRecived : Date
isPrepaid : Boolean
number : String
price : Money

dispatch()
close()

Customer
name : String
address : String

creditRating()

n   1

Corprate Customer
contactName : String
creditRating : String
creditLimit : Double

remind()
billForMonth()

Personal Customer
creditCard# : Long Integer

Interaction Diagrams

Interaction diagrams model the behavior of use cases by describing the way groups of objects interact to complete the task. The two kinds of interaction diagrams are **sequence** and **collaboration** diagrams.

When to Use: Interaction Diagrams

Interaction diagrams are used when you want to model the behavior of several objects in a use case. They demonstrate how the objects collaborate for the behavior. Interaction diagrams do not give a in depth representation of the behavior.

### How to Draw: Interaction Diagrams

Sequence diagrams, collaboration diagrams, or both diagrams can be used to demonstrate the interaction of objects in a use case. Sequence diagrams generally show the sequence of events that occur. Collaboration diagrams demonstrate how objects are statically connected. Both diagrams are relatively simple to draw and contain similar elements.

Sequence diagrams:

Sequence diagrams demonstrate the behavior of objects in a use case by describing the objects and the messages they pass. The diagrams are read left to right and descending. The example below shows an object of class 1 start the behavior by sending a message to an object of class 2. Messages pass between the different objects until the object of class 1 receives the final message.
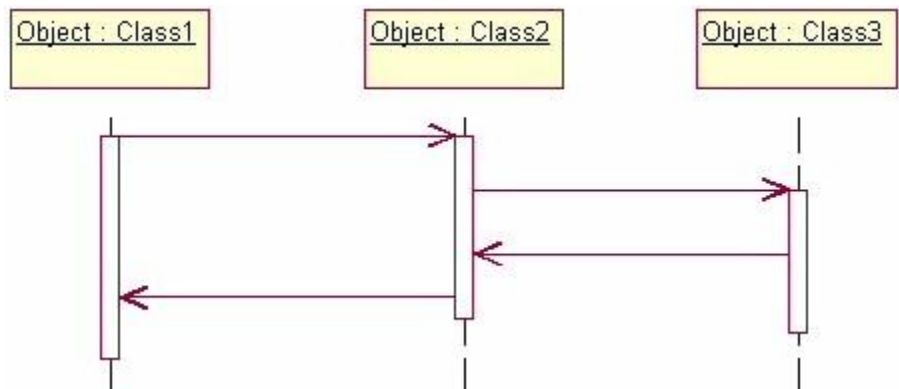

Below is a slightly more complex example. The light blue vertical rectangles the objects activation while the vertical dashed lines represent the life of the object. The green vertical rectangles represent when a particular object has control. The ▣ represents when the object is destroyed. This diagrams also shows conditions for messages to be sent to other object. The condition is listed between brackets next to the message. For example, a [condition] has to be met before the object of class 2 can send a message() to the object of class 3.
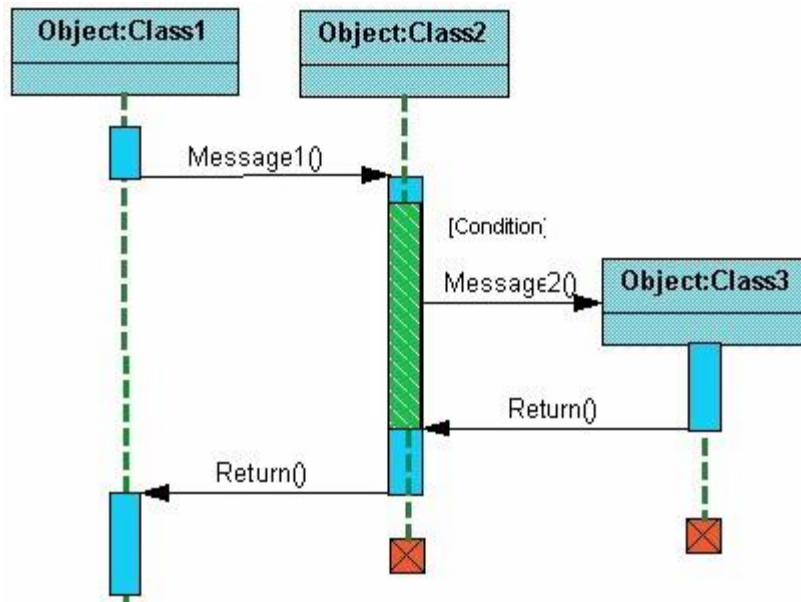
The next diagram shows the beginning of a sequence diagram for placing an order. The object an Order Entry Window is created and sends a message to an Order object to prepare the order. Notice the the names of the objects are followed by a colon. The names of the classes the objects belong to do not have to be listed. However the colon is required to denote that it is the name of an object following the objectName:className naming system.

Next the Order object checks to see if the item is in stock and if the [InStock] condition is met it sends a message to create a new Delivery Item object.
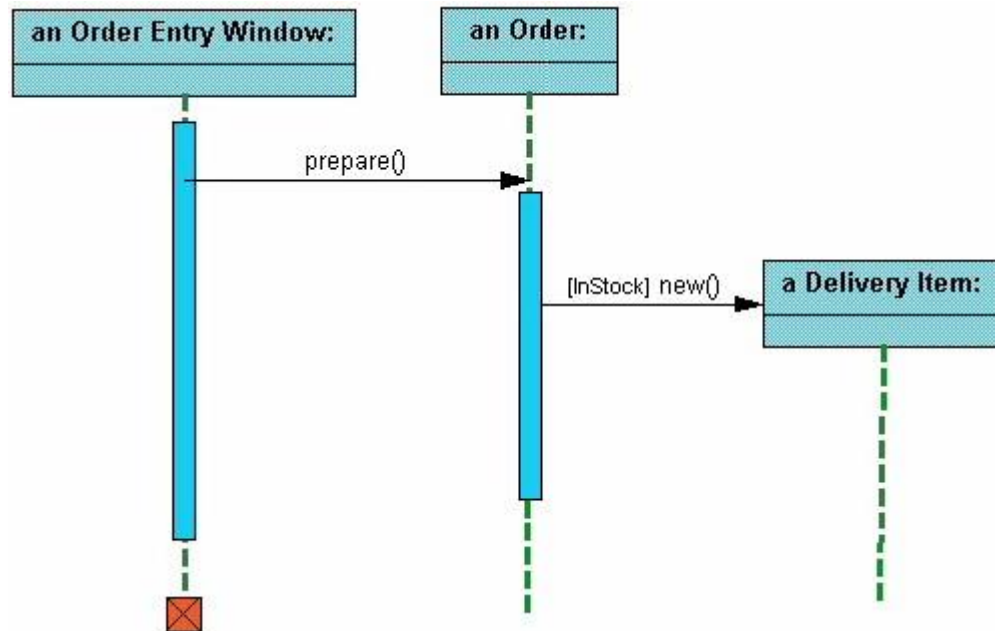
The next diagrams adds another conditional message to the Order object. If the item is [OutOfStock] it sends a message back to the Order Entry Window object stating that the object is out of stack.
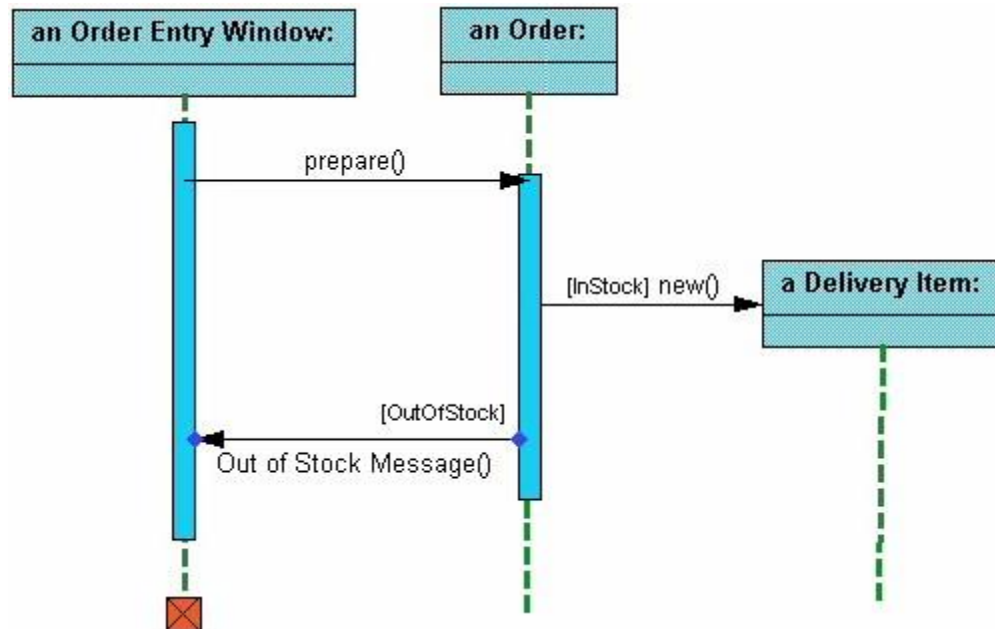

This simple diagram shows the sequence that messages are passed between objects to complete a use case for ordering an item.

Collaboration diagrams:

Collaboration diagrams are also relatively easy to draw. They show the relationship between objects and the order of messages passed between them. The objects are listed as icons and arrows indicate the messages being passed between them. The numbers next to the messages are called sequence numbers. As the name suggests, they show the sequence of the messages as they are passed between the objects. There are many acceptable sequence numbering schemes in UML. A simple 1, 2, 3... format can be used, as the example below shows, or for more detailed and complex diagrams a 1, 1.1 ,1.2, 1.2.1... scheme can be used.
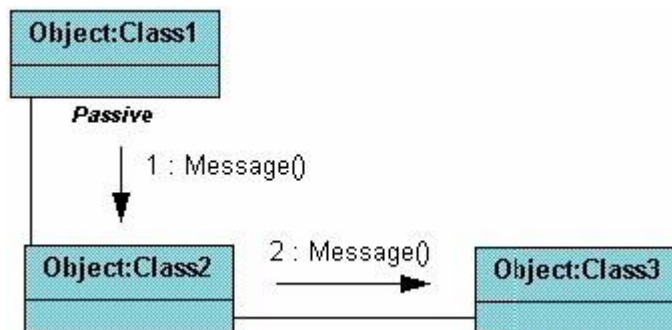
The example below shows a simple collaboration diagram for the placing an order use case. This time the names of the objects appear after the colon, such as :Order Entry Window following the objectName:className naming convention. This time the class name is shown to demonstrate that all of objects of that class will behave the same way.
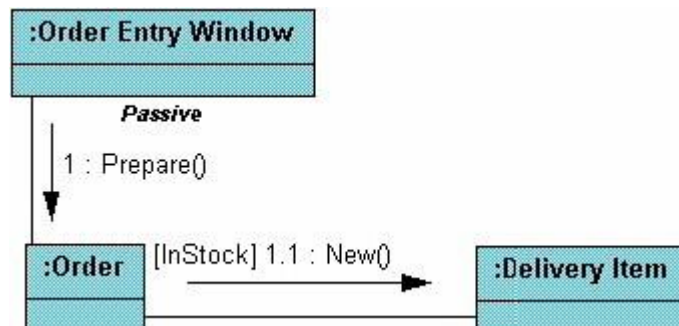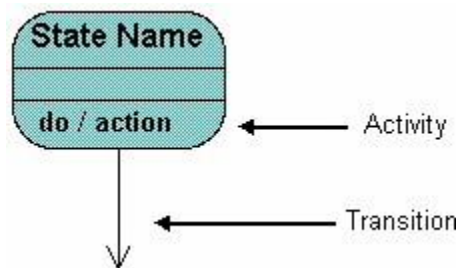
State Diagrams

State diagrams are used to describe the behavior of a system. State diagrams describe all of the possible states of an object as events occur. Each diagram usually represents objects of a single class and track the different states of its objects through the system.

When to Use: State Diagrams

Use state diagrams to demonstrate the behavior of an object through many use cases of the system. Only use state diagrams for classes where it is necessary to understand the behavior of the object through the entire system.

How to Draw: State Diagrams

State diagrams have very few elements. The basic elements are rounded boxes representing the state of the object and arrows indicting the transition to the next state. The activity section of the state symbol depicts what activities the object will be doing while it is in that state.

All state diagrams being with an initial state of the object. This is the state of the object when it is created. After the initial state the object begins changing states. Conditions based on the activities can determine what the next state the object transitions to.
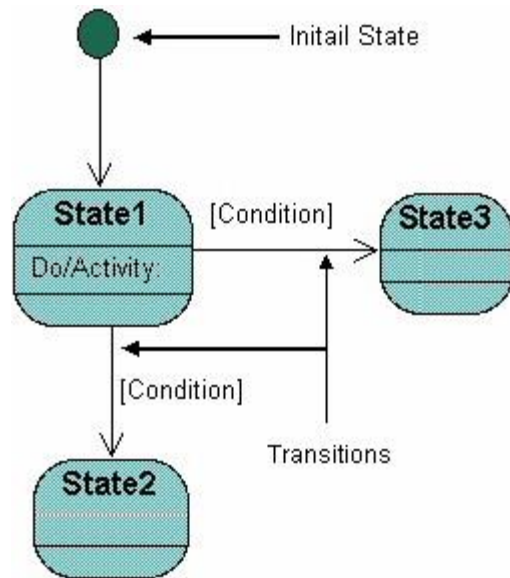
Below is an example of a state diagram might look like for an Order object. When the object enters the Checking state it performs the activity "check items." After the activity is completed the object transitions to the next state based on the conditions [all items available] or [an item is not available]. If an item is not available the order is canceled. If all items are available then the order is dispatched. When the object transitions to the Dispatching state the activity "initiate delivery" is performed. After this activity is complete the object transitions again to the Delivered state.
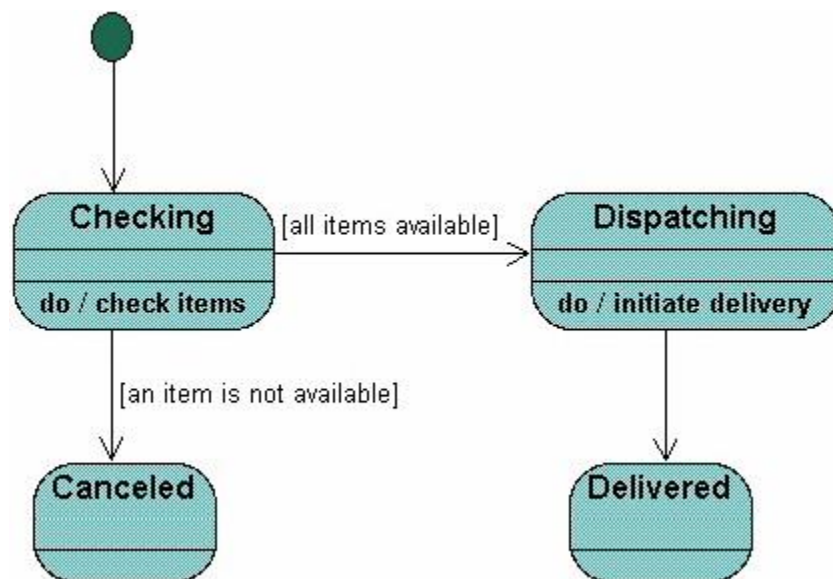
State diagrams can also show a super-state for the object. A super-state is used when many transitions lead to the a certain state. Instead of showing all of the transitions from each state to the redundant state a super-state can be used to show that all of the states inside of the super-state can transition to the redundant state. This helps make the state diagram easier to read.

The diagram below shows a super-state. Both the Checking and Dispatching states can transition into the Canceled state, so a transition is shown from a super-state named Active to the state Cancel. By contrast, the state Dispatching can only transition to the Delivered state, so we show an arrow only from the Dispatching state to the Delivered state.
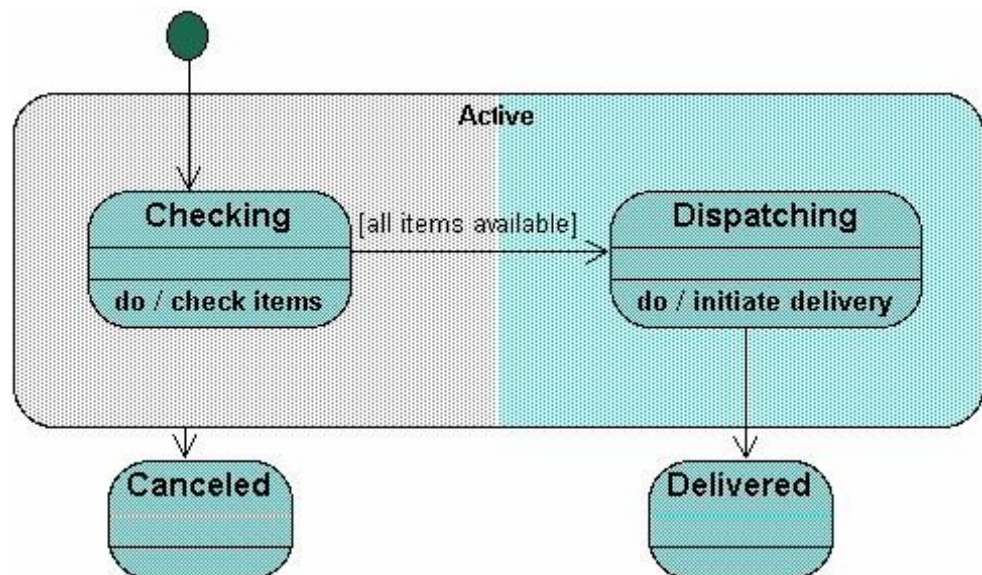

Activity Diagrams

Activity diagrams describe the workflow behavior of a system. Activity diagrams are similar to state diagrams because activities are the state of doing something. The diagrams describe the state of activities by showing the sequence of activities performed. Activity diagrams can show activities that are conditional or parallel.

Activity diagrams should be used in conjunction with other modeling techniques such as interaction diagrams and state diagrams. The main reason to use activity diagrams is to model the workflow behind the system being designed. Activity Diagrams are also useful for: analyzing a use case by describing what actions need to take place and when they should occur; describing a complicated sequential algorithm; and modeling applications with parallel processes.

However, activity diagrams should not take the place of interaction diagrams and state diagrams. Activity diagrams do not give detail about how objects behave or how objects collaborate.
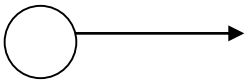
Elements of an activity diagrams
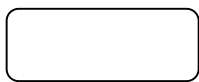
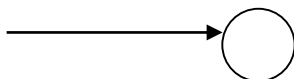Rounded box-contains the name of the activity
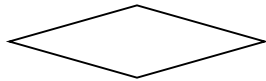
Decision diamond –shows decision

Transition arrow-shows the direction of the flow of the activities

Start marker –shows the initial activity

End marker – shows the last activity

Synchronization bar- shows activities that takes place   in parrallel

How to Draw: Activity Diagrams

Activity diagrams show the flow of activities through the system. Diagrams are read from top to bottom and have branches and forks to describe conditions and parallel activities. *A fork* is used when multiple activities are occurring at the same time. The diagram below shows a fork after activity1. This indicates that both activity2 and activity3 are occurring at the

same time. After activity2 there is a **branch**. The *branch* describes what activities will take place based on a set of conditions. All branches at some point are followed by a merge to indicate the end of the conditional behavior started by that branch. After the merge all of the parallel activities must be combined by a join before transitioning into the final activity state.
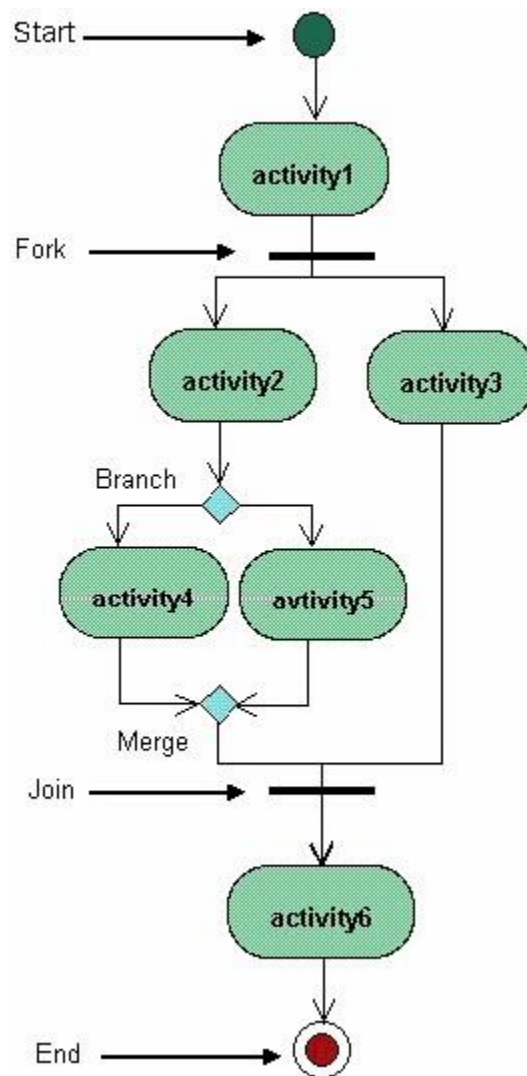
Below is a possible *activity diagram for processing an order.*

The diagram shows the flow of actions in the system's workflow. Once the order is received the activities split into two parallel sets of activities. One side fills and sends the order while

16

the other handles the billing. On the Fill Order side, the method of delivery is decided conditionally. Depending on the condition either the Overnight Delivery activity or the Regular Delivery activity is performed. Finally, the parallel activities combine to close the order.
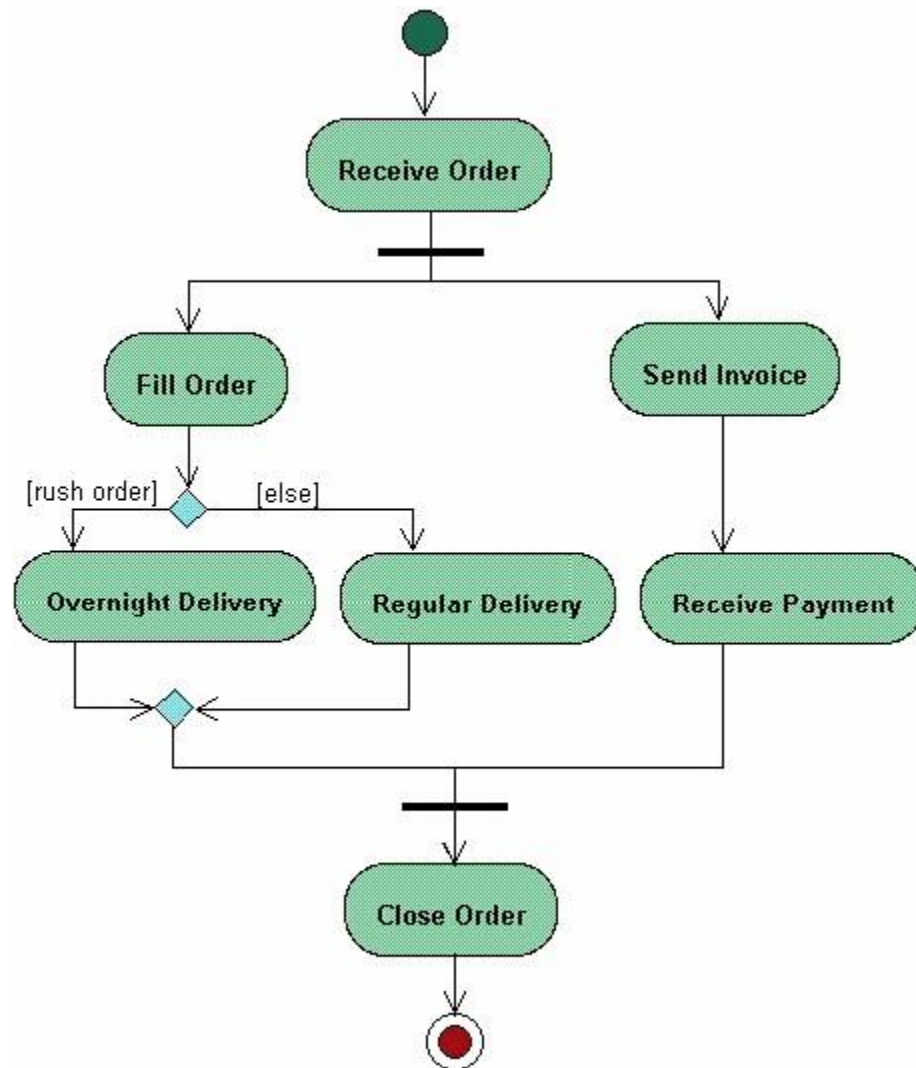
Physical Diagrams

There are two types of physical diagrams: **deployment diagrams** and **component diagrams.** Deployment diagrams show the physical relationship between hardware and software in a system. Component diagrams show the software components of a system and how they are related to each other. These relationships are called *dependencies*.

When to Use: Physical Diagrams

Physical diagrams are used when development of the system is complete. Physical diagrams are used to give descriptions of the physical information about a system.

How to Draw: Physical Diagrams

Many times the deployment and component diagrams are combined into one physical diagram. A combined deployment and component diagram combines the features of both diagrams into one diagram.

The deployment diagram contains nodes and connections. A node usually represents a piece of hardware in the system. A connection depicts the communication path used by the hardware to communicate and usually indicates a method such as TCP/IP.
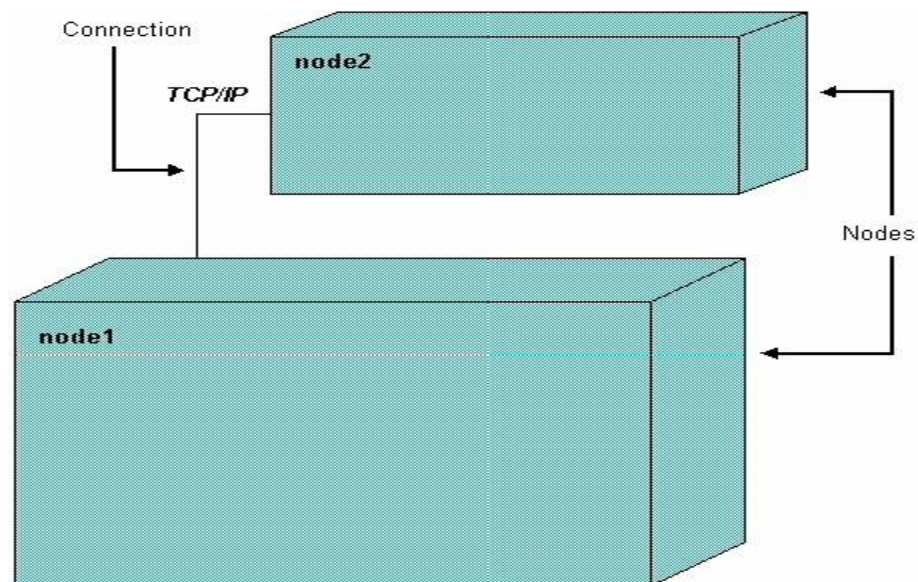
The component diagram contains components and dependencies. Components represent the physical packaging of a module of code. The dependencies between the components show how changes made to one component may affect the other components in the system.

- Dependencies in a component diagram are represented by a dashed line between two or more components. Component diagrams can also show the interfaces used by the components to communicate to each other.

The combined deployment and component diagram below gives a high level physical description of the completed system. The diagram shows two nodes that represent two machines communicating through TCP/IP. Component2 is dependant on component1, so changes to component 2 could affect component1. The diagram also depicts component3 interfacing with component1. This diagram gives the reader a quick overall view of the entire system.
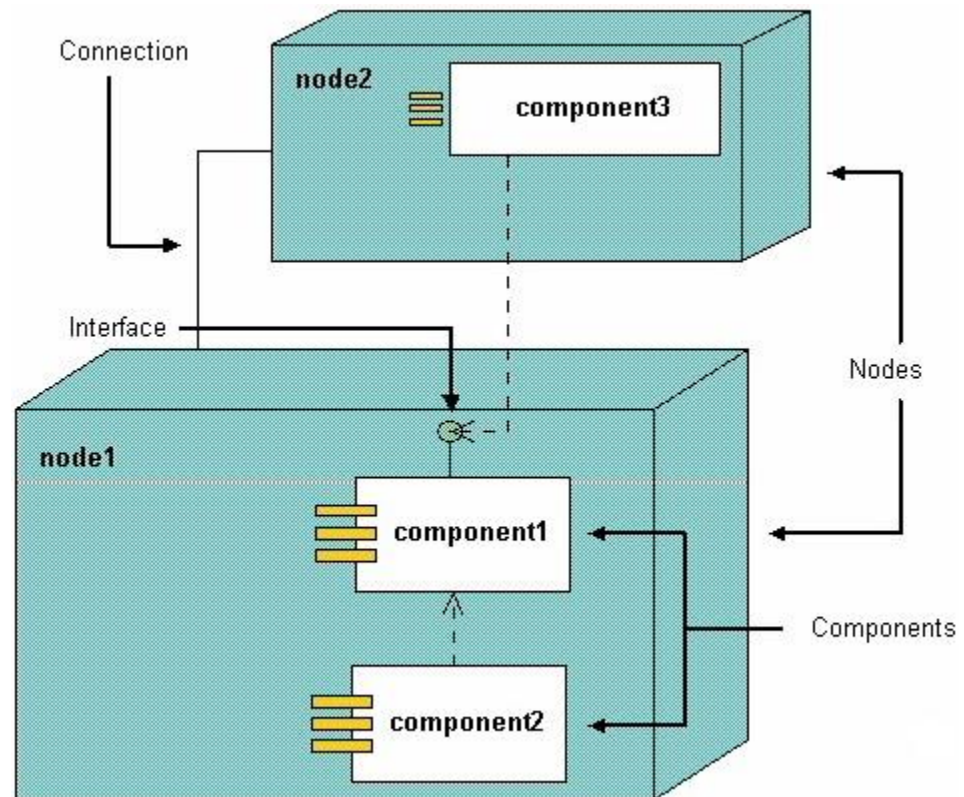
Summary

Object oriented development method is architecture centric in that they support at least three separate but interrelated architectural views of the evolving information system i.e.

i) **Functional view**: Describe the external behavior of the system form perspective of the user. Use case and use case diagrams are the primary approach used by UML to depict the functional view. Activity diagrams are used to supplement the use cases.

ii)   **The static view**: Describe the structure of the system in terms of attributes, methods, classes and relationships. Class diagrams are the main models in UML used to portray the static view of an evolving information system.

iii) **Dynamic view**: Describe the internal behavior of the system in terms messages passed among objects and state changes within an object. In UML this view is represented using state and sequence diagram