Q: What is Unified Process

Unified process (UP) is an architecture-centric, use-case driven, iterative and incremental development process that leverages unified modeling language and is compliant with the system process engineering metamodel. Unified process can be applied to different software systems with different levels of technical and managerial complexity across various domains and organizational cultures.

Unified process is a refinement of rational unified process. It is an extensible framework that can be customized for specific projects.   
  
This process divides the development process into four phases:

* Inception
* Elaboration
* Conception
* Transition

Unified Process has the following major characteristics:

* It is use-case driven
* It is architecture-centric
* It is risk focused
* It is iterative and incremental

Q1 What is  Unified Modeling Language (UML)?

[UML](http://www.omg.org/spec/UML/), short for Unified Modeling Language, is a standardized modeling language consisting of an integrated set of diagrams, developed to help system and software developers 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. In this article we will give you detailed ideas about what is UML, the history of UML and a description of each UML diagram type, along with UML examples.

## The Origin of UML

The goal of UML is to provide a standard notation that can be used by all object-oriented methods and to select and integrate the best elements of precursor notations. UML has been designed for a broad range of applications. Hence, it provides constructs for a broad range of systems and activities (e.g., distributed systems, analysis, system design and deployment).

UML is a notation that resulted from the unification of OMT from

1. [Object Modeling Technique OMT](https://en.wikipedia.org/wiki/Object-modeling_technique) [[James Rumbaugh](https://en.wikipedia.org/wiki/James_Rumbaugh) 1991] - was best for analysis and data-intensive information systems.
2. Booch [[Grady Booch](https://en.wikipedia.org/wiki/Grady_Booch) 1994] - was excellent for design and implementation. Grady Booch had worked extensively with the [Ada](https://en.wikipedia.org/wiki/Ada_(programming_language)) language, and had been a major player in the development of Object Oriented techniques for the language. Although the Booch method was strong, the notation was less well received (lots of cloud shapes dominated his models - not very tidy)
3. OOSE (Object-Oriented Software Engineering [[Ivar Jacobson](https://en.wikipedia.org/wiki/Ivar_Jacobson) 1992]) - featured a model known as Use Cases. Use Cases are a powerful technique for understanding the behaviour of an entire system (an area where OO has traditionally been weak).

In 1994, Jim Rumbaugh, the creator of OMT, stunned the software world when he left General Electric and joined Grady Booch at Rational Corp. The aim of the partnership was to merge their ideas into a single, unified method (the working title for the method was indeed the "Unified Method").

By 1995, the creator of OOSE, Ivar Jacobson, had also joined Rational, and his ideas (particularly the concept of "Use Cases") were fed into the new Unified Method - now called the Unified Modelling Language1. The team of Rumbaugh, Booch and Jacobson are affectionately known as the "Three Amigos"

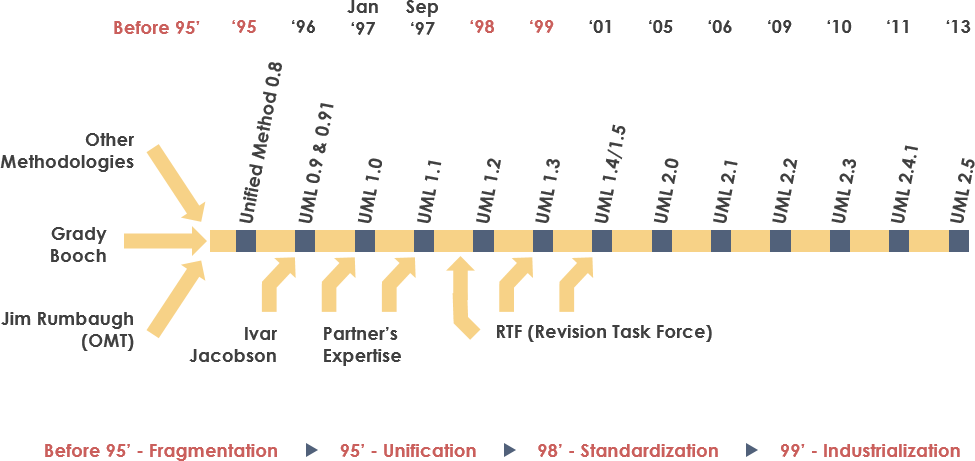
UML has also been influenced by other object-oriented notations:

* Mellor and Shlaer [1998]
* Coad and Yourdon [1995]
* Wirfs-Brock [1990]
* Martin and Odell [1992]

UML also includes new concepts that were not present in other major methods at the time, such as extension mechanisms and a constraint language.

## History of UML

1. During 1996, the first Request for Proposal (RFP) issued by the [Object Management Group (OMG)](http://www.omg.org/)provided the catalyst for these organizations to join forces around producing a joint RFP response.
2. Rational established the UML Partners consortium with several organizations willing to dedicate resources to work toward a strong UML 1.0 definition. Those contributing most to the UML 1.0 definition included:
   * Digital Equipment Corp
   * HP
   * i-Logix
   * IntelliCorp
   * IBM
   * ICON Computing
   * MCI Systemhouse
   * Microsoft
   * Oracle
   * Rational Software
   * TI
   * Unisys
3. This collaboration produced UML 1.0, a modeling language that was well-defined, expressive, powerful, and generally applicable. This was submitted to the OMG in January 1997 as an initial RFP response.1
4. In January 1997 IBM, ObjecTime, Platinum Technology, Ptech, Taskon, Reich Technologies and Softeam also submitted separate RFP responses to the OMG. These companies joined the UML partners to contribute their ideas, and together the partners produced the revised UML 1.1 response. The focus of the UML 1.1 release was to improve the clarity of the UML 1.0 semantics and to incorporate contributions from the new partners. It was submitted to the OMG for their consideration and adopted in the fall of 1997.1 and enhanced 1.1 to 1.5, and subsequently to UML 2.1 from 01 to 06 (now the UML current version is 2.5)



Q2 Why  Unified Modeling Language (UML)?

As the strategic value of software increases for many companies, the industry looks for techniques to automate the production of software and to improve quality and reduce cost and time-to-market. These techniques include component technology, visual programming, patterns and frameworks. Businesses also seek techniques to manage the complexity of systems as they increase in scope and scale. In particular, they recognize the need to solve recurring architectural problems, such as physical distribution, concurrency, replication, security, load balancing and fault tolerance. Additionally, the development for the World Wide Web, while making some things simpler, has exacerbated these architectural problems. The Unified Modeling Language (UML) was designed to respond to these needs. The primary goals in the design of the UML summarize by Page-Jones in Fundamental Object-Oriented Design in UML as follows:

1. Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
2. Provide extensibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development processes.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of the OO tools market.
6. Support higher-level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

## UML - An Overview

Before we begin to look at the theory of the UML, we are going to take a very brief run through some of the major concepts of the UML.

The first thing to notice about the UML is that there are a lot of different diagrams (models) to get used to. The reason for this is that it is possible to look at a system from many different viewpoints. A software development will have many stakeholders playing a part.

For Example:

* Analysts
* Designers
* Coders
* Testers
* QA
* The Customer
* Technical Authors

All of these people are interested in different aspects of the system, and each of them require a different level of detail. For example, a coder needs to understand the design of the system and be able to convert the design to a low level code. By contrast, a technical writer is interested in the behavior of the system as a whole, and needs to understand how the product functions. The UML attempts to provide a language so expressive that all stakeholders can benefit from at least one UML diagram.

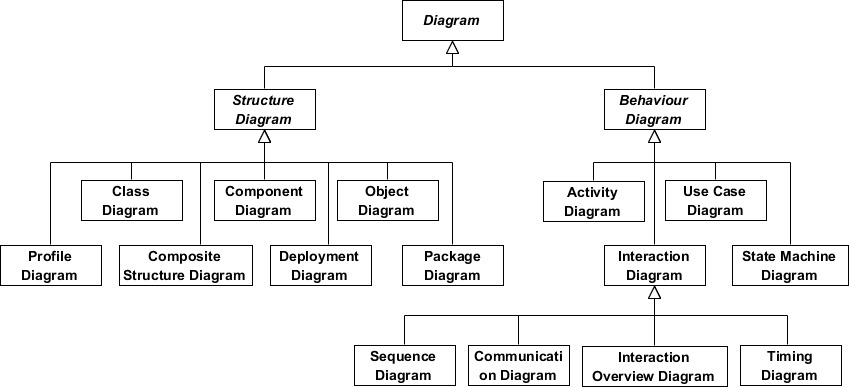
Here's a quick look at each one of these 13 diagrams in as shown in the UML 2 Diagram Structure below:

Structure diagrams show the static structure of the system and its parts on different abstraction and implementation levels and how they are related to each other. The elements in a structure diagram represent the meaningful concepts of a system, and may include abstract, real world and implementation concepts, there are seven types of structure diagram as follows:

* [Class Diagram](https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/#class-diagram)
* [Component Diagram](https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/#component-diagram)
* [Deployment Diagram](https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/#deployment-diagram)
* [Object Diagram](https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/#object-diagram)
* [Package Diagram](https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/#package-diagram)
* [Composite Structure Diagram](https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/#composite-structure-diagram)
* [Profile Diagram](https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/#profile-diagram)

Behavior diagrams show the dynamic behavior of the objects in a system, which can be described as a series of changes to the system over time, there are seven types of behavior diagrams as follows:

* [Use Case Diagram](https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/#use-case-diagram)
* [Activity Diagram](https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/#activity-diagram)
* [State Machine Diagram](https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/#state-machine-diagram)
* [Sequence Diagram](https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/#sequence-diagram)
* [Communication Diagram](https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/#communication-diagram)
* [Interaction Overview Diagram](https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/#interaction-overview-diagram)
* [Timing Diagram](https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/#timing-diagram)



Q3 What is a Class Diagram?

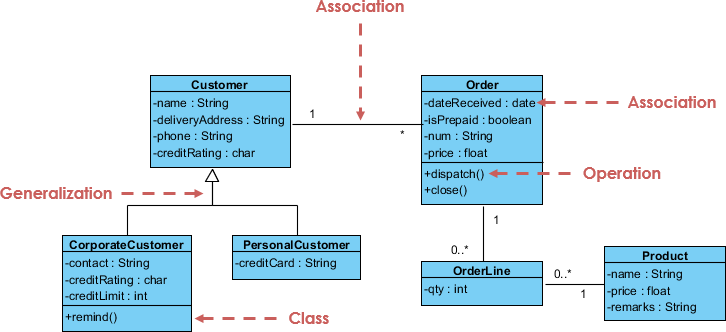
The class diagram is a central modeling technique that runs through nearly all object-oriented methods. This diagram describes the types of objects in the system and various kinds of static relationships which exist between them.

#### Relationships

There are three principal kinds of relationships which are important:

1. Association - represent relationships between instances of types (a person works for a company, a company has a number of offices.
2. Inheritance - the most obvious addition to ER diagrams for use in OO. It has an immediate correspondence to inheritance in OO design.
3. Aggregation - Aggregation, a form of object composition in object-oriented design.

#### Class Diagram Example



Q4: What is an Object Diagram?

An object diagram is a graph of instances, including objects and data values. A static object diagram is an instance of a class diagram; it shows a snapshot of the detailed state of a system at a point in time. The difference is that a class diagram represents an abstract model consisting of classes and their relationships. However, an object diagram represents an instance at a particular moment, which is concrete in nature. The use of object diagrams is fairly limited, namely to show examples of data structure.

Q 5: Give the Difference between Class Diagram vs Object Diagram with an Example

Some people may find it difficult to understand the difference between a UML Class Diagram and a UML Object Diagram as they both comprise of named "rectangle blocks", with attributes in them, and with linkages in between, which make the two UML diagrams look similar. Some people may even think they are the same because in the UML tool they use both the notations for

Class Diagram and Object Diagram are put inside the same diagram editor - Class Diagram.

But in fact, Class Diagram and Object Diagram represent two different aspects of a code base. In this article, we will provide you with some ideas about these two UML diagrams, what they are, what are their differences and when to use each of them.

#### Relationship between Class Diagram and Object Diagram

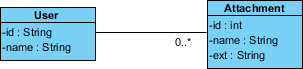
You create "classes" when you are programming. For example, in an online banking system you may create classes like 'User', 'Account', 'Transaction', etc. In a classroom management system you may create classes like 'Teacher', 'Student', 'Assignment', etc. In each class, there are attributes and operations that represent the characteristic and behavior of the class. Class Diagram is a UML diagram where you can visualize those classes, along with their attributes, operations and the inter-relationship.

UML Object Diagram shows how object instances in your system are interacting with each other at a particular state. It also represents the data values of those objects at that state. In other words, a UML Object Diagram can be seen as a representation of how classes (drawn in UML Class Diagram) are utilized at a particular state.

If you are not a fan of those definition stuff, take a look at the following UML diagram examples. I believe that you will understand their differences in seconds.

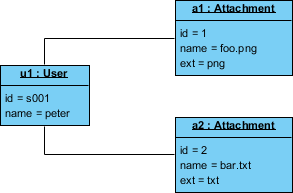
#### Class Diagram Example

The following Class Diagram example represents two classes - User and Attachment. A user can upload multiple attachment so the two classes are connected with an association, with 0..\* as multiplicity on the Attachment side.



#### Object Diagram Example

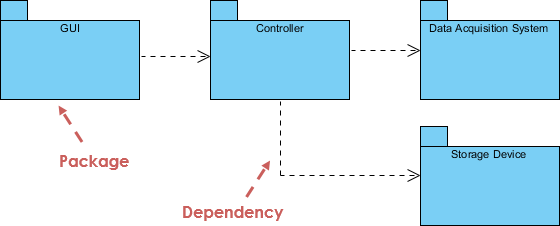
The following Object Diagram example shows you how the object instances of User and Attachment class "look like" at the moment Peter (i.e. the user) is trying to upload two attachments. So there are two Instance Specification for the two attachment objects to be uploaded.



Q6:What is a Package Diagram?

Package diagram is UML structure diagram which shows packages and dependencies between the packages. Model diagrams allow to show different views of a system, for example, as multi-layered (aka multi-tiered) application - multi-layered application model.

#### Package Diagram Example



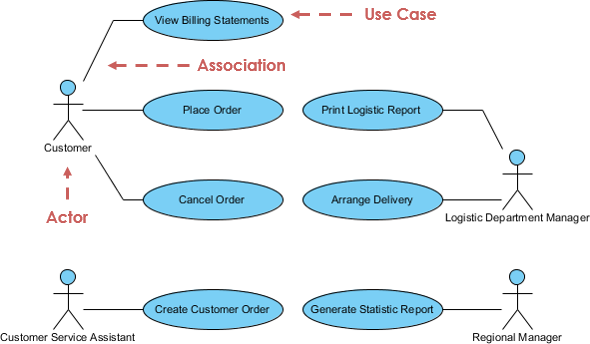
Q7: What is a Use Case Diagram?

A use-case model describes a system's functional requirements in terms of use cases. It is a model of the system's intended functionality (use cases) and its environment (actors). Use cases enable you to relate what you need from a system to how the system delivers on those needs.

Think of a use-case model as a menu, much like the menu you'd find in a restaurant. By looking at the menu, you know what's available to you, the individual dishes as well as their prices. You also know what kind of cuisine the restaurant serves: Italian, Mexican, Chinese, and so on. By looking at the menu, you get an overall impression of the dining experience that awaits you in that restaurant. The menu, in effect, "models" the restaurant's behavior.

Because it is a very powerful planning instrument, the use-case model is generally used in all phases of the development cycle by all team members.

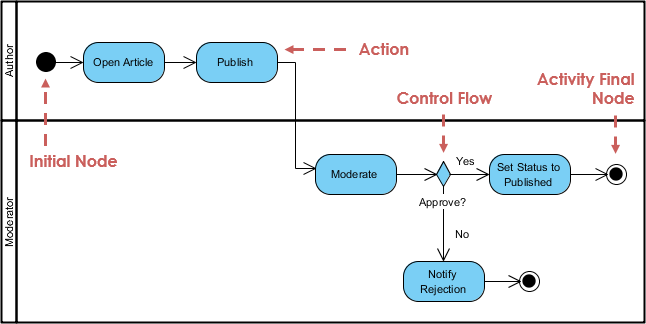
#### Use Case Diagram Example



Q8: What is an Activity Diagram?

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. It describes the flow of control of the target system, such as the exploring complex business rules and operations, describing the use case also the business process. In the Unified Modeling Language, activity diagrams are intended to model both computational and organizational processes (i.e. workflows).

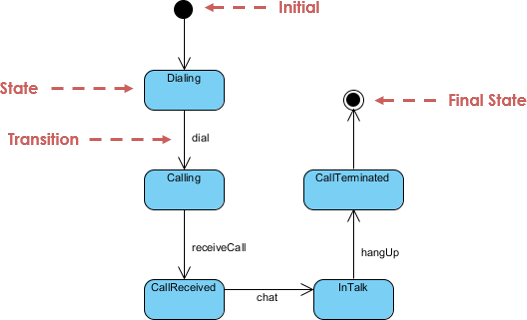
#### Activity Diagram Example



Q9:What is a State Machine Diagram?

A state diagram is a type of diagram used in UML to describe the behavior of systems which is based on the concept of state diagrams by David Harel. State diagrams depict the permitted states and transitions as well as the events that effect these transitions. It helps to visualize the entire lifecycle of objects and thus help to provide a better understanding of state-based systems.

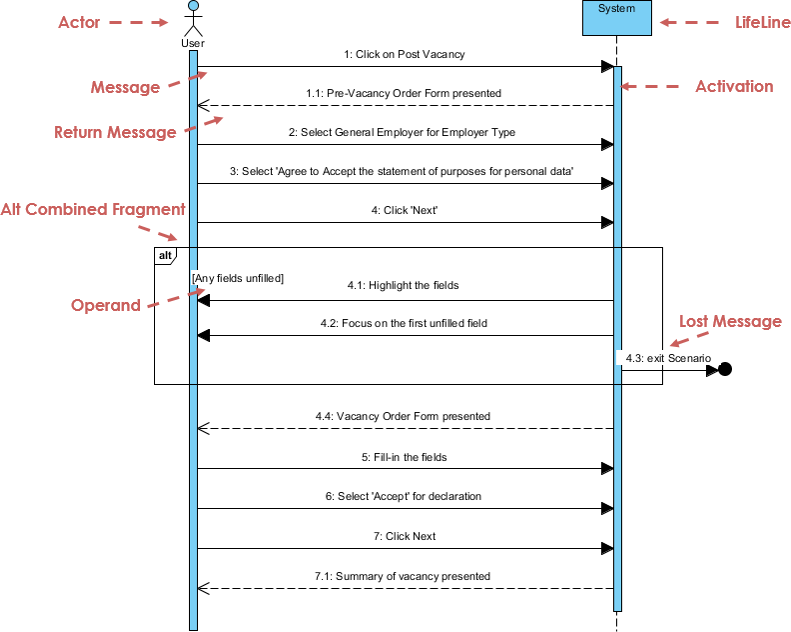
#### State Machine Diagram Example



Q10 :What is a Sequence Diagram?

The Sequence Diagram models the collaboration of objects based on a time sequence. It shows how the objects interact with others in a particular scenario of a use case. With the advanced visual modeling capability, you can create complex sequence diagram in few clicks. Besides, some modeling tool such as Visual Paradigm can generate sequence diagram from the flow of events which you have defined in the use case description.

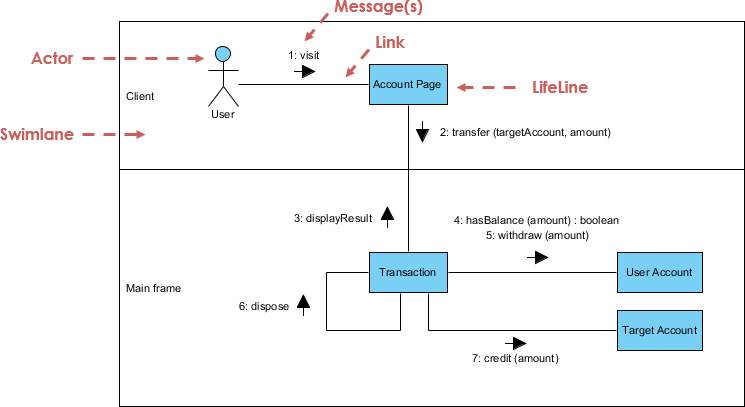
#### Sequence Diagram Example



Q 11: What is a Communication Diagram?

Similar to Sequence Diagram, the Communication Diagram is also used to model the dynamic behavior of the use case. When compare to Sequence Diagram, the Communication Diagram is more focused on showing the collaboration of objects rather than the time sequence. They are actually semantically equivalent, so some of the modeling tool such as, Visual Paradigm allows you to generate it from one to the other.

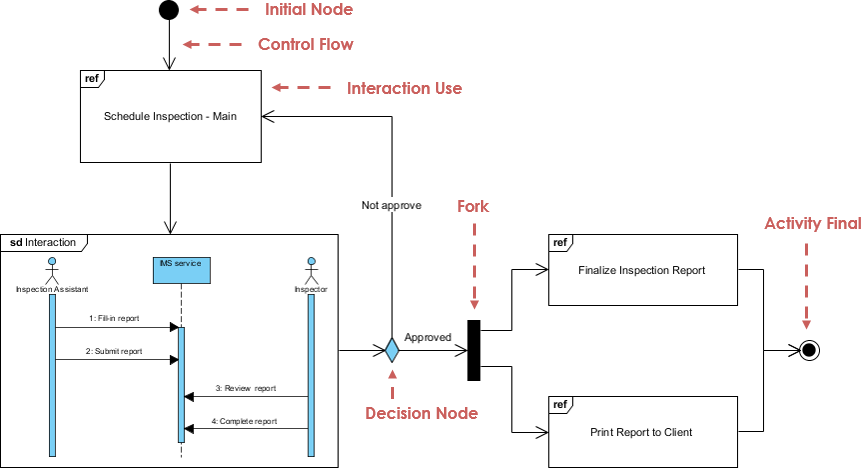
#### Communication Diagram Example



Q 12: What is Interaction Overview Diagram?

The Interaction Overview Diagram focuses on the overview of the flow of control of the interactions. It is a variant of the Activity Diagram where the nodes are the interactions or interaction occurrences. The Interaction Overview Diagram describes the interactions where messages and lifelines are hidden. You can link up the "real" diagrams and achieve high degree navigability between diagrams inside the Interaction Overview Diagram.

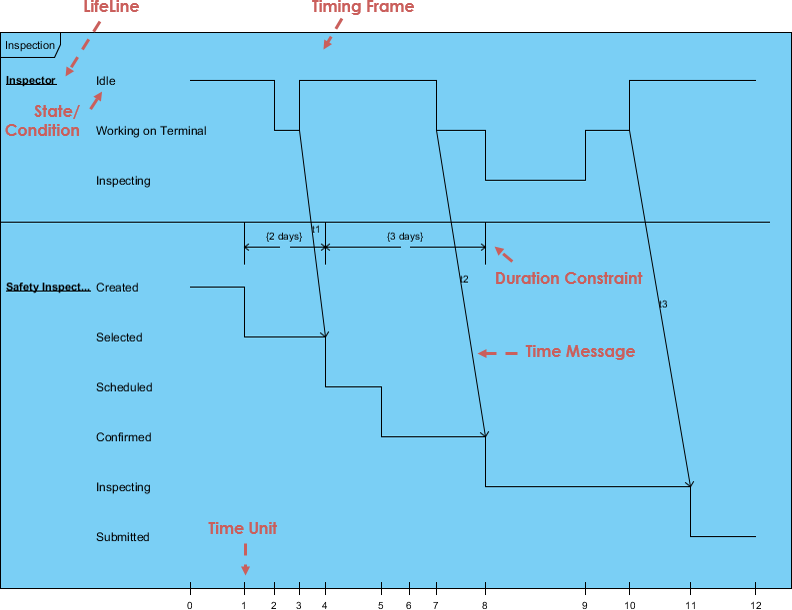
#### Interaction Overview Diagram Example



Q 13: What is Timing Diagram?

Timing Diagram shows the behavior of the object(s) in a given period of time. Timing diagram is a special form of a sequence diagram. The differences between timing diagram and sequence diagram are the axes are reversed so that the time are increase from left to right and the lifelines are shown in separate compartments arranged vertically.

#### Timing Diagram Example



Q:What is a use case diagram?

In the Unified Modeling Language (UML), a use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use a set of specialized symbols and connectors. An effective use case diagram can help your team discuss and represent:

* Scenarios in which your system or application interacts with people, organizations, or external systems
* Goals that your system or application helps those entities (known as actors) achieve
* The scope of your system

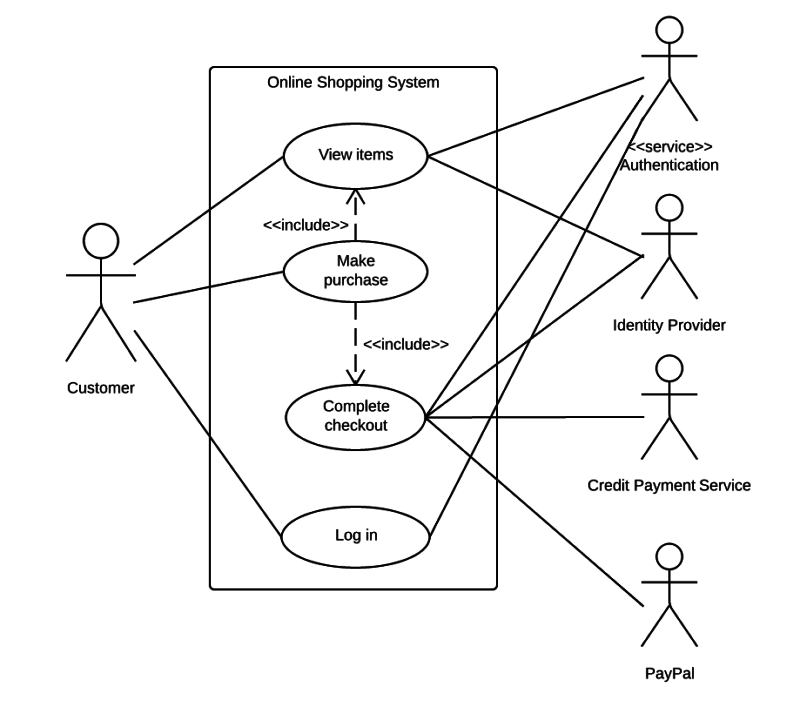
When to apply use case diagrams

A use case diagram doesn't go into a lot of detail—for example, don't expect it to model the order in which steps are performed. Instead, a proper use case diagram depicts a high-level overview of the relationship between use cases, actors, and systems. Experts recommend that use case diagrams be used to supplement a more descriptive textual use case.

UML is the modeling toolkit that you can use to build your diagrams. Use cases are represented with a labeled oval shape. Stick figures represent actors in the process, and the actor's participation in the system is modeled with a line between the actor and use case. To depict the system boundary, draw a box around the use case itself.

UML use case diagrams are ideal for:

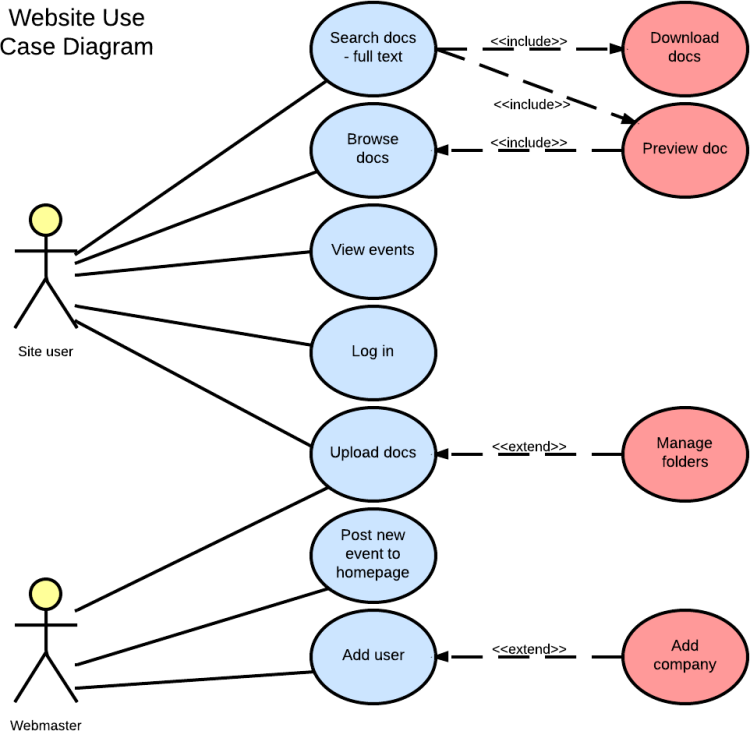
* Representing the goals of system-user interactions
* Defining and organizing functional requirements in a system
* Specifying the context and requirements of a system
* Modeling the basic flow of events in a use case



Use case diagram components

To answer the question, "What is a use case diagram?" you need to first understand its building blocks. Common components include:

* Actors: The users that interact with a system. An actor can be a person, an organization, or an outside system that interacts with your application or system. They must be external objects that produce or consume data.
* System: A specific sequence of actions and interactions between actors and the system. A system may also be referred to as a scenario.
* Goals: The end result of most use cases. A successful diagram should describe the activities and variants used to reach the goal.



Use case diagram symbols and notation

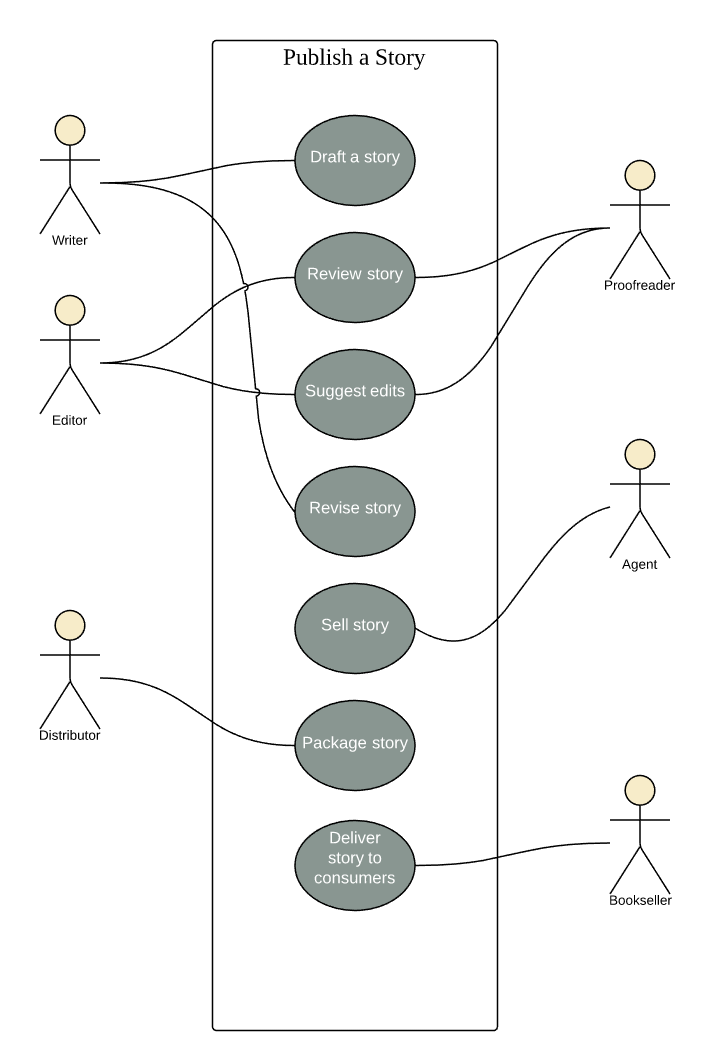
The notation for a use case diagram is pretty straightforward and doesn't involve as many types of symbols as other UML diagrams.

Use cases: Horizontally shaped ovals that represent the different uses that a user might have.

* Actors: Stick figures that represent the people actually employing the use cases.
* Associations: A line between actors and use cases. In complex diagrams, it is important to know which actors are associated with which use cases.
* System boundary boxes: A box that sets a system scope to use cases. All use cases outside the box would be considered outside the scope of that system. For example, Psycho Killer is outside the scope of occupations in the chainsaw example found below.
* Packages: A UML shape that allows you to put different elements into groups. Just as with component diagrams, these groupings are represented as file folders.

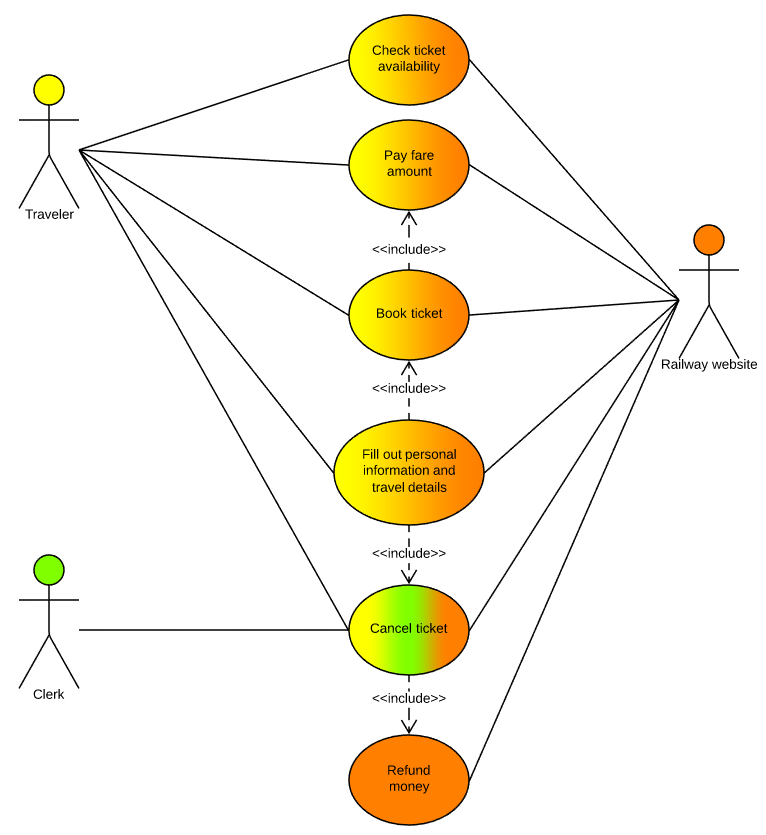
Use case diagram examples

Book publishing use case diagram example

This use case diagram is a visual representation of the process required to write and publish a book. Whether you’re an author, an agent, or a bookseller, inserting this diagram into your use case scenario can help your team publish the next big hit.  

### Railway reservation use case diagram example

You can adapt this template for any process where a customer purchases a service. With attractive color schemes, text that’s easy to read and edit, and a wide-ranging UML shape library, you’re ready to go! Click to [try out this template](https://www.lucidchart.com/documents/editNewOrRegister/6641f344-e2ce-4138-a2c8-7fad5ad13fb2/) on your own.



# Most efficient way to write use cases

Because use cases model requirements, they are highly dynamic by nature. The more we examine a scenario, the more we learn and the more things change. To further complicate the issue, changes to one use case can lead to changes in others. Therefore, we want a flexible, highly efficient method for writing use cases that maximizes stakeholder value and minimizes risk early in the project and minimizes costly rework later.

An iterative, breadth-before-depth approach is best. This breadth-first approach involves two aspects: finding and outlining use cases and detailing individual use cases.

## Finding and outlining use cases

Use cases exist in sets, and the relationships between the various use cases and Actors are important. As you learn more about the Actors, you also learn more about the system's boundaries and transactions. Likewise, as you learn more about the system's transactions, you learn more about its Actors. Therefore, it is more efficient to capture and outline the primary use cases before detailing individual use cases. This way, you can identify and understand the importance and risk associated with each use case before committing time to detailing them.  This aspect of the breadth-before-depth approach is embodied in the process by the explicit separation of two tasks [Find and Outline Requirements](https://people.cs.clemson.edu/~johnmc/courses/Publish/openup/tasks/find_and_outline_requirements_90D272B9.html) and [Detail Requirements](https://people.cs.clemson.edu/~johnmc/courses/Publish/openup/tasks/detail_requirements_9747F71E.html).

## Detailing individual use cases

Similarly, it makes sense to write each individual use case iteratively. Start by detailing the main scenario.  As you do this, you can identify various alternative and error flows that the use case might follow, and then evaluate, rearrange, or eliminate them and prioritize them before detailing the surviving scenarios, so you can focus your effort in the right place.

### Detail the basic flow of events (main scenario)

As a starting point, use the step-by-step description of the main scenario that you created during [Task: Find and Outline Requirements](https://people.cs.clemson.edu/~johnmc/courses/Publish/openup/tasks/find_and_outline_requirements_90D272B9.html). Then, gradually add details to this scenario, describing what the use case does, not how to solve problems internal to the system.

A flow of events description covers:

* How and when the use case starts
* When the use case interacts with the Actors and what data they exchange
* When the use case uses data stored in the system or stores data in the system
* How and when the use case ends

It does not describe:

* The GUI
* Technical details of hardware or software
* Design issues

Describe the flow of events for the main scenario in the form:

1. The use case starts when <Actor name> <does something>.
2. The system <does something in response>.
3. The <Actor name> does something else.

A use case consists of a number of scenarios, each representing specific instances of the use case that correspond to specific inputs from the Actor or to specific conditions in the environment. Each scenario describes alternate ways that the system behaves, or it may describe failure or exception cases.

Review the list of alternative flows that you captured during [Task: Find and Outline Requirements](https://people.cs.clemson.edu/~johnmc/courses/Publish/openup/tasks/find_and_outline_requirements_90D272B9.html). As you detail the main scenario, you may identify additional alternate flows by asking these questions:

* Are there different options available, depending on input from the Actor? (Example: If the Actor enters an invalid PIN number while accessing an ATM.)
* What business rules may come into play? (Example: The Actor requests more money from the ATM than is available in her account.)
* What could go wrong? (Example: No network connection available when required to perform a transaction.)

It is best to develop these scenarios iteratively, as well.

1. Begin by identifying them.
2. Examine each possible scenario to determine whether it is relevant, that it can actually happen, and that it is distinct from other scenarios.
3. Eliminate redundant or unnecessary scenarios.
4. Prioritize the remaining scenarios and start detailing the more important ones.

In addition to the detailing the flow of events of each alternative flow in the form described above, each alternative flow should describe:

* Where the alternative flow can be inserted in the basic flow of events
* The condition that needs to be fulfilled for the alternative behavior to start
* How and where the basic flow of events is resumed, or how the use case ends

## Structure the use case

It is useful to structure the use case according to scenarios. This helps both to simplify communication and maintenance and to permit the use cases to be implemented iteratively. Name and describe each key scenario so that these may be prioritized, assigned to an iteration and an individual, and referenced from the [Work Items List](https://people.cs.clemson.edu/~johnmc/courses/Publish/openup/workproducts/work_items_list_39D03CC8.html) for implementation.

In addition to structuring the use cases according to scenarios, it is often useful to structure the scenarios themselves into subflows. This provides an additional level of granularity for planning work and tracking progress. Unless a subflow involves only a minor part of the complete flow of events (which can be described in the body of the text), describe each subflow in a separate section of the Flow of Events section. Subflows that should be in a separate section include these examples:

* Subflows that occupy a large segment of a given flow of events.
* Exceptional and alternative flows of events. This helps the use case's basic flow of events to stand out more clearly.
* Any subflow that can be run at several intervals in the same flow of events.

If there are subflows identified that are common to multiple use cases, you can re-factor your use-case model to include these. See [Concept: Use-Case Model](https://people.cs.clemson.edu/~johnmc/courses/Publish/openup/guidances/concepts/use_case_model_CD178AF9.html) for more information on the <<include>> dependency.

For more information on structuring the use case, see the "Flow of Events - Structure" section in [Concept: Use Case](https://people.cs.clemson.edu/~johnmc/courses/Publish/openup/guidances/concepts/use_case_BB199D1B.html).

The [Template: Use-Case Specification](https://people.cs.clemson.edu/~johnmc/courses/Publish/openup/guidances/templates/uc_specification_E97E98B0.html) provides a suggested structure for the use case specification. See [Example: Evolution of the Use-Case Model](https://people.cs.clemson.edu/~johnmc/courses/Publish/openup/guidances/examples/uc_model_evolve_960F136B.html) and [Example: Use-Case Specification](https://people.cs.clemson.edu/~johnmc/courses/Publish/openup/guidances/examples/use_case_spec_CD5DD9B1.html) for examples of structuring the use-case model and use case specifications, respectively.

## Detail special requirements

Be sure to also capture any requirements that are related to the use case but are not taken into consideration in the flow of events of the use case. These requirements include business rules, design constraints, usability requirements, performance requirements, reliability requirements, supportability requirements, and interface requirements. These requirements all influence the implementation and associated cost as much as the flow of events. Therefore, they must be agreed upon and prioritized.

Typically, nonfunctional requirements that refer to a specific use case are captured in the Special Requirements section of the use case. For more information, see the "Special Requirements" section in [Concept: Use Case](https://people.cs.clemson.edu/~johnmc/courses/Publish/openup/guidances/concepts/use_case_BB199D1B.html).

If there are nonfunctional requirements that apply to more than one use case, capture these in the [Artifact: Supporting Requirements Specification](https://people.cs.clemson.edu/~johnmc/courses/Publish/openup/workproducts/supporting_requirements_spec_7D9DD47C.html). For more information on supporting requirements, see [Concept: Supporting Requirements](https://people.cs.clemson.edu/~johnmc/courses/Publish/openup/guidances/concepts/supporting_requirements_B2C4D610.html).

For guidance in writing clear, concise, unambiguous special and supporting requirements, see guidelines [Writing Good Requirements](https://people.cs.clemson.edu/~johnmc/courses/Publish/openup/guidances/guidelines/writing_good_requirements_48248536.html) and [Requirement Pitfalls](https://people.cs.clemson.edu/~johnmc/courses/Publish/openup/guidances/guidelines/requirement_pitfalls_14EE9652.html).

See Section 8 of [Example: Use-Case Specification](https://people.cs.clemson.edu/~johnmc/courses/Publish/openup/guidances/examples/use_case_spec_CD5DD9B1.html) for examples of special requirements.

## Describe preconditions and post-conditions

A precondition of a use case explains the state that the system must be in for the use case to be able to start. Be careful in describing the system state. Avoid describing the detail of other, incidental activities that may already have taken place.

A post-condition of a use case lists possible states that the system can be in after the use case runs. The system must be in one of those states. A post-condition also states actions that the system performs at the end of the use case, regardless of what occurred in the use case. Post-conditions may be categorized as Minimal Guarantees or Success Guarantees:

* Minimal Guarantees represent conditions that will be true when the use cases end, regardless of how they terminate.
* Success Guarantees represent condition that will be true when the use cases end successfully, regardless of which paths they took.

Consider the following when specifying preconditions and post-condition:

* The states described by pre- or post-conditions should be states that the user can observe. "The user has logged on to the system" or "The user has opened the document" are examples of observable states.
* A precondition is a constraint on when a use case can start. It is not the event that starts the use case.
* A precondition for a use case is not a precondition for only one subflow, although you can define preconditions and post-conditions at the subflow level.
* A post-condition for a use case should be true regardless of which alternative flows were executed; it should not be true only for the main flow. If something could fail, you would cover that in the post-condition by saying "The action has completed," or if something failed, "The action was not performed," rather than just "The action is completed."

**Use Case Example**

**Problem Description**

The Bank accounts and Transactions (BAT for short) system is to be built for the Big Bank Corporation. It must handle clients’ bank accounts and the (standard) services on these accounts: deposit, withdraw, transfer, and get balance.

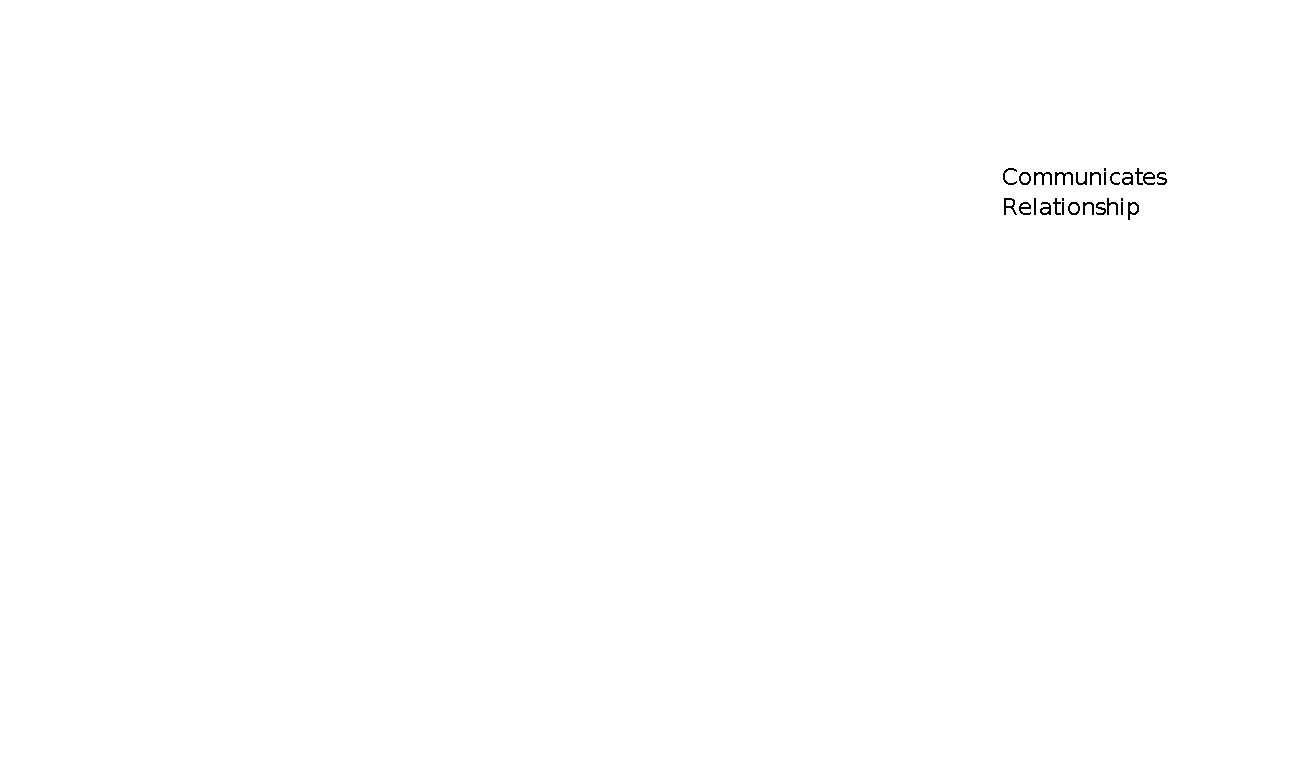
The transactions are recorded, because at the end of each month, the system sends out account statements to all clients showing all transactions performed for their accounts during the last period; the system sends the statements to the printer from where a junior clerk posts them.

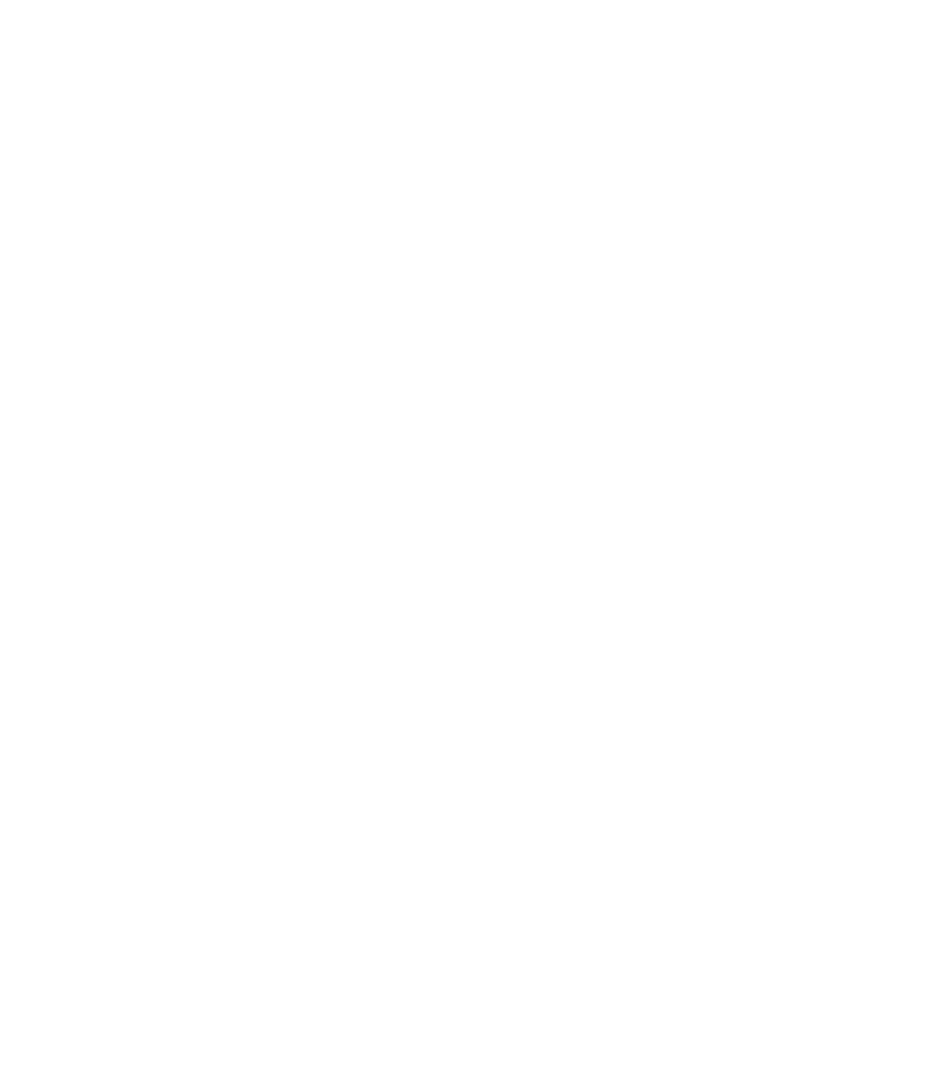
The system is accessed by the bank’s clients only indirectly, i.e., either via a teller, an ATM, or the internet. All transactions and queries are possible via a teller; all transactions and queries are possible except deposits via an ATM; and all except deposits and withdraw via the internet.

Opening an account can be performed only via a teller and the internet; however, if a client opens an account via the internet they must identify themselves with a teller to have their account activated (this is government policy to avoid money laundering, e.g.).

Closing an account can only be performed by a teller, and it requires a final statement to be sent out to the client. The Bank offers various account types, which fall into two categories: saving and checking. Savings accounts cannot be overdrawn. There can be a credit limit, subject to agreement by the bank, on checking accounts; a checking account cannot be overdrawn beyond this limit.

**Use Case Diagram for Bank Accounts and Transactions System**



**Use Case Diagram for Bank Accounts and Transactions System**

# Bank Accounts and Transactions System

**USE CASE DESCRITPION**

1. ***Use Case: Open Account***

**Overview:** This use case is for opening an account of the new customer.

**Precondition**: For opening an account customer needs to first authenticate himself by giving some identification proof.

**Main Flow:**

1. First the customer will fill up the open account form and show his  
   documents to the teller.
2. Teller will verify the documents for the authentication purpose.
3. After authentication, a deposit slip will be given to customer.
4. After filling the deposit slip, it will be submitted to the deposit window.
5. The customer will deposit the money at the deposit window to open an account.
6. The clerk will open a new account for the customer and deposit the initial amount in the account.
7. A receiving slip will be given to the customer and account is opened Client requests Teller to open an account.
8. ***Use Case: Close Account***

**Overview:** This use case is for closing an account of the customer.

**Precondition:** Before closing an account, final statement will be sent to the client.

**Main Flow:**

# First the customer will submit an application for closing his account.

# Then customer will fill the withdraw slip to withdraw the entire amount from the account.

# The withdraw slip will be deposited at withdraw window.

# The customer will receive amount from the withdraw window.

# Then teller will close the customer account.

**3*. Use Case: Get Balance/Withdraw***

**Overview:** This use case is for checking the balance in his/her account.

**Precondition**: The Client has already identified him/herself

**Main Flow**:

1. If the customer is using Internet then first the customer will login by giving his/her user id and password.
2. Then customer will select the bank account number from which he wants to check the balance.
3. Customer then clicks on ‘Get Balance’ link to check the balance of his/her account.
4. If the customer is checking his account via ATM then he must insert his card into the ATM machine and enter his pin no. and password.
5. Customer then selects the check balance option.
6. System will display the balance.
7. System displays option for other transaction.
8. If customer doesn’t want any more transactions to occur then he will take out his card from the machine.
9. Then teller will close the customer account.



# Problem Domain Object Model for BAT System

**Travel Agent System**

**Problem Description**

Travel Agent System is to be built for Fata-Fat Travel Agency

The Travel Agency System must handle customer’s who wants to avail the avai le packages provided by the travel agency. The system must be capable of providing the services as:

* Package Description
* Package Booking
* Package Modification
* Package Cancellation and
* Other Services

1. **Package Description:** Customer provides the holiday plan based upon his requirements. According to that the system will provide the avai le packages. The system will describe the package according to the need.
2. **Package Booking:** If the package suits customers’ holiday plan, then customer can book the package online or via travel agent.
3. **Package Modification:** If the customer at some later point wants to extend or modify the tour he can contact the same agent.
4. **Package Cancellation:** The customer can cancel the tour and the associated package.
5. **Other Services:** Other services may include Taxi Booking, Local Site Booking, and Hotel Reservation.

# REQUIREMENTS

## Functional Requirements

1. Customer interacts with the travel agent and fills up customer details in terms of form.
2. The form will have the number of members, their age, booking date, traveling date, days of stay and mode of traveling.

## Package Description:

* Available package legs
* Cost of package
* Discount on package
* Package types (corporate family and others) and
* Mode of travel (via road, via sea or via air)

## Package Booking:

* Travel agent will book the package according to customer’s best-suited requirements after checking the status of available packages.
* The customer can make payment by cash or by credit card.

## Package Modification:

* Customer can extend his stay or can book another package to extend his holiday plan.

**Package Cancellation:**

* Customer can cancel the package within five days from the date of booking. After that 25% will be deducted from the package cost

## Other Services:

* Local Site Visit: If customer wants to visit any local site, he has to book in advance and have to pay extra cost for that.

## Performance Requirements

1. The system should be reliable
2. Instant response should be made
3. Customer payment details should be secure

**System Requirements**

The travel agent system must be capable of providing fast, accurate, and courteous ("user friendly") services for all requests supported. The system must be able to:

* Answer inquiries about flights and fares
* Generate, modify, and cancel traveler profiles and travel accounts,
* Make, change, complete, and cancel reservations,
* Obtain payment method and verify traveler credit line,
* Generate travel itineraries and arrange for delivery of flight tickets and flight itineraries.

Types of ticket delivery services include: mail, courier, and airport pick-up; types of itinerary delivery services include mail, fax, e-mail.

#### COMMUNICATES RELATIONSHIP

# 

#### USECASE: BOOK-TRIP



#### USECASE : CANCEL-TRIP

# 

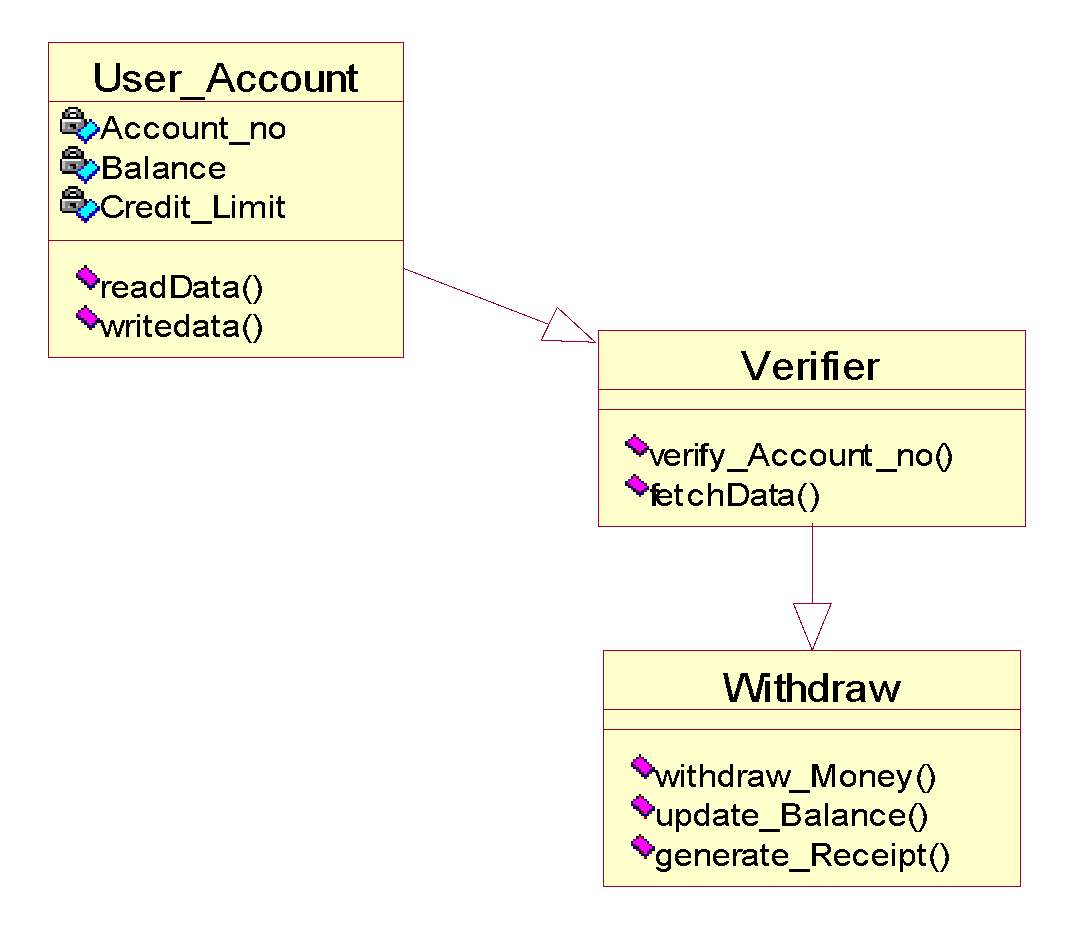
##### USECASE: MODIFY TRIP

**ANALYSIS MODEL**

**Exercise-4**

**Class Diagrams**

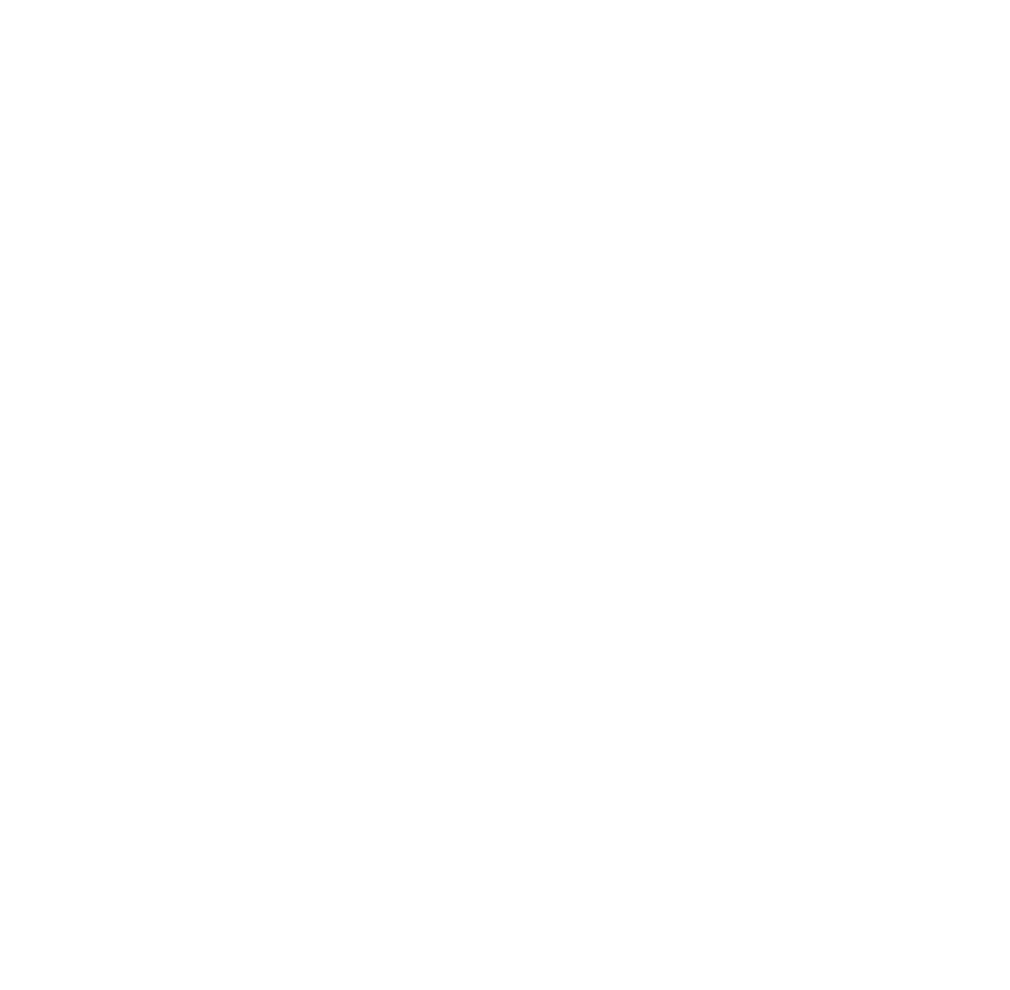
**Withdraw Money**

**Open Account**

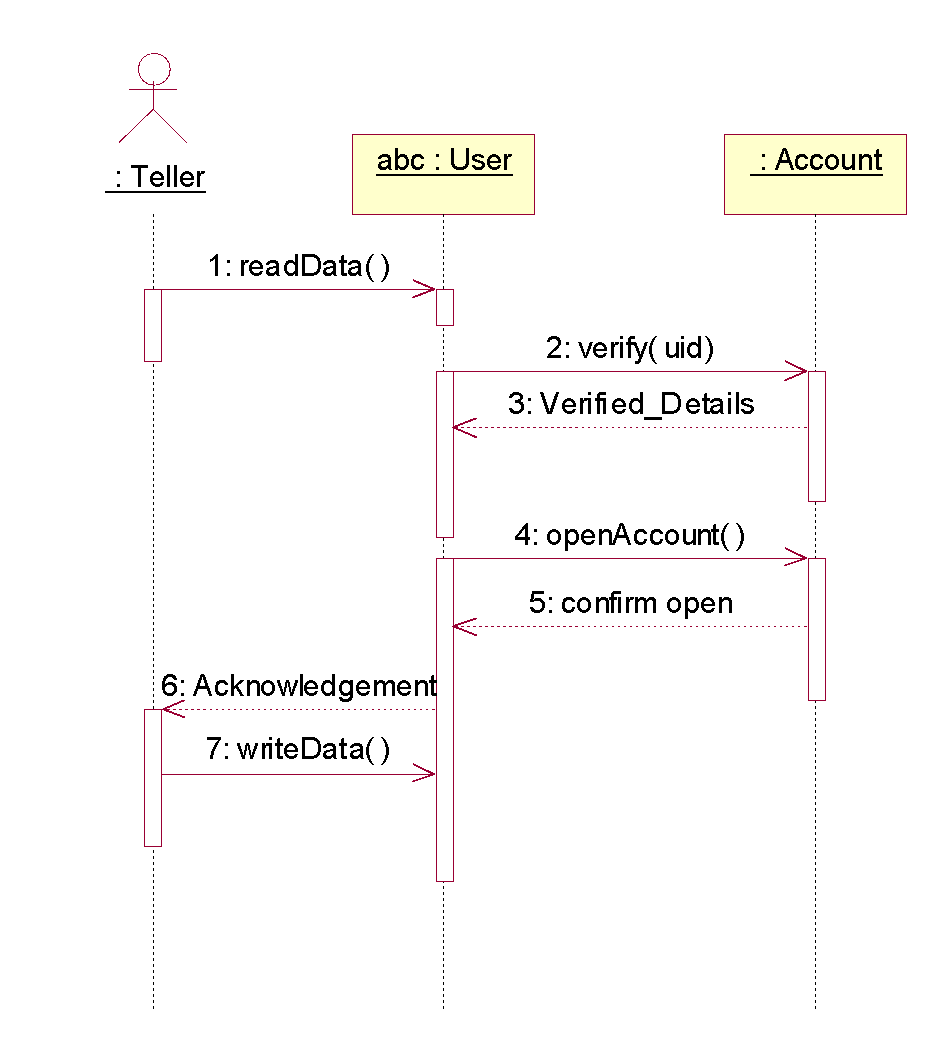
**Close Account**



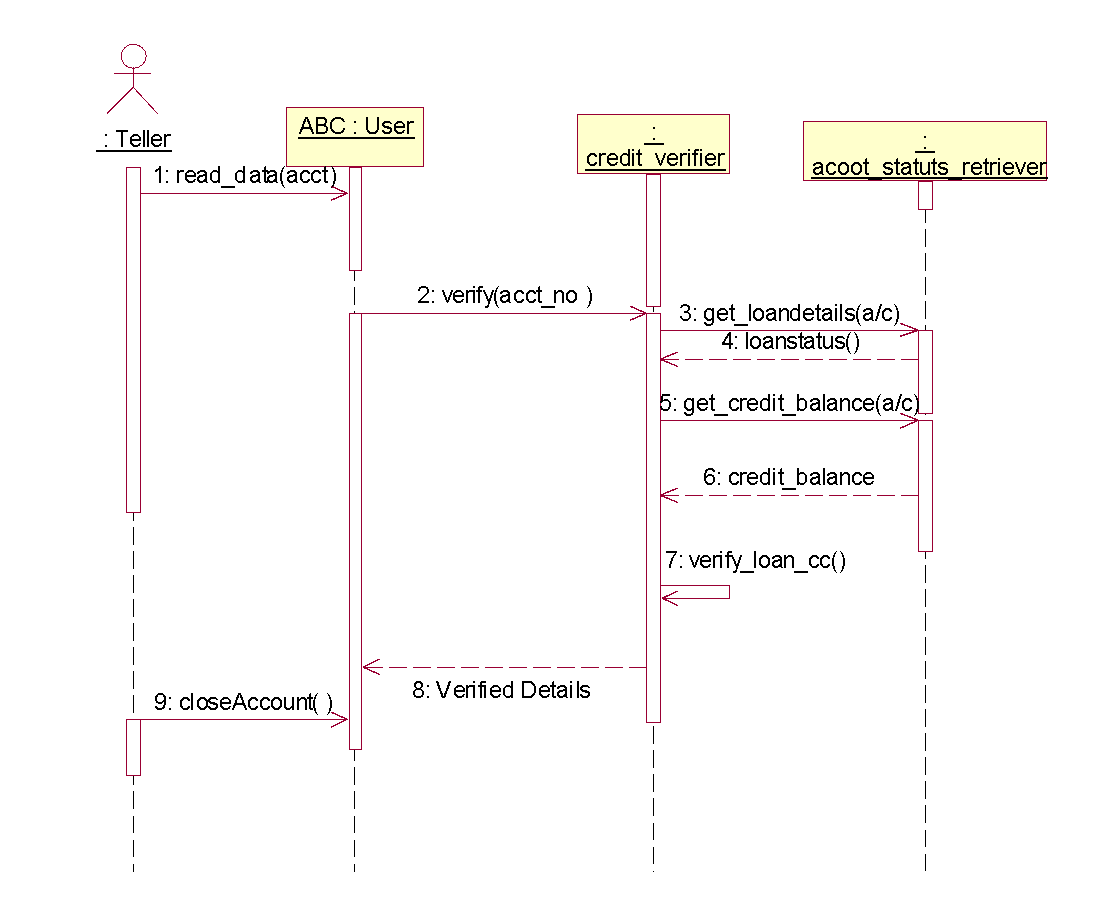
**Class Diagram for BAT**

**Sequence Diagrams**

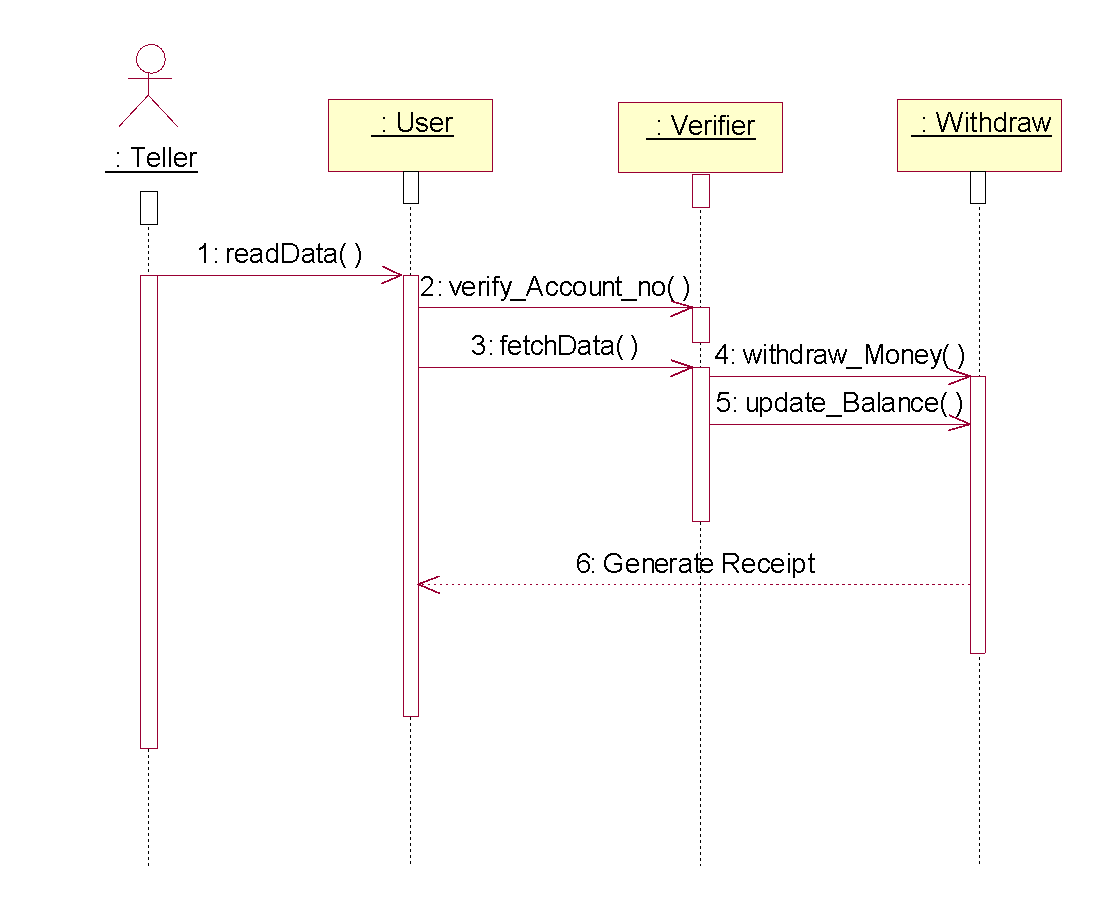
# Open Account Sequence Diagram



# Close Account Sequence Diagram



# Withdraw Money Sequence Diagram



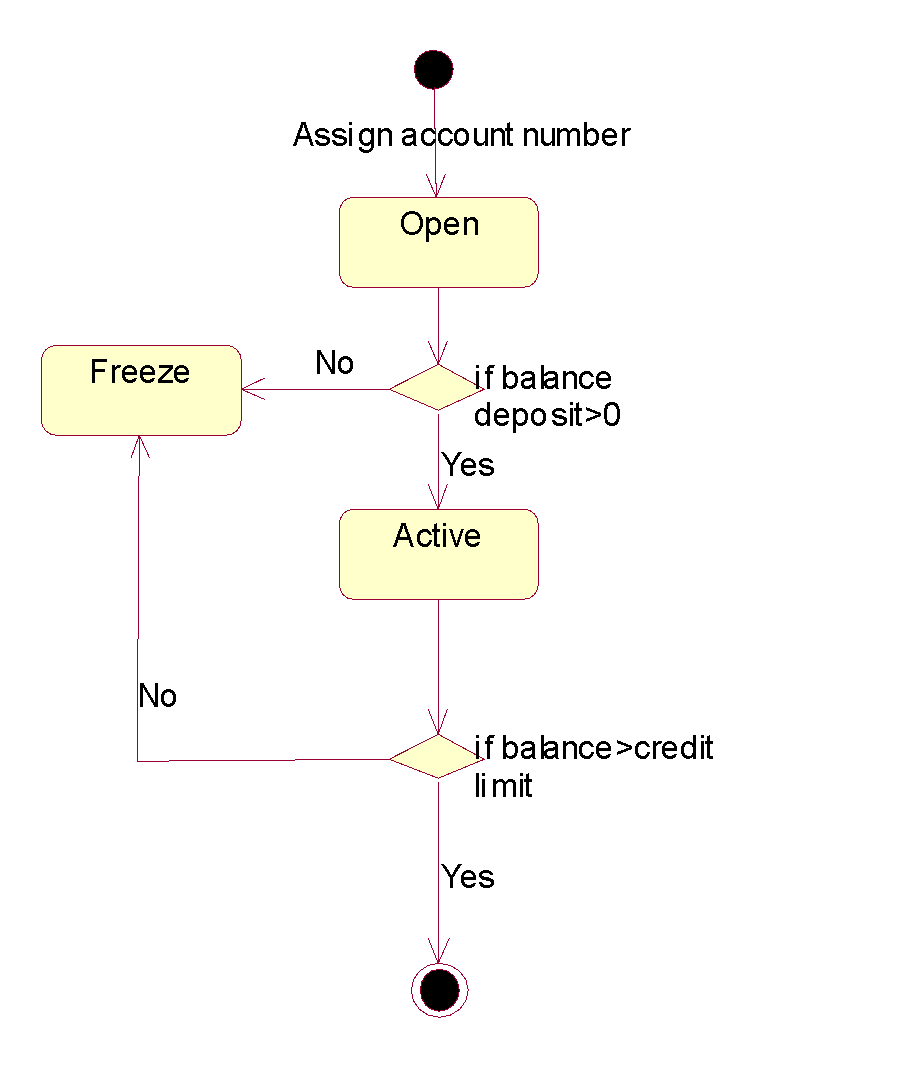
**State Chart Diagrams**

**Account State Chart Diagram**





**State Chart Diagram for Withdraw Amount**



**Exercise-7**

**Activity Diagram**

**Activity Diagram for Cash Withdrawal**

