DEPARTMENT OF IT

IT6413 - SOFTWARE ENGINEERING LABORATORY

FOURTH SEMESTER LAB MANUALS

REGULATION 2013

ANNA UNIVESITY CHENNAI

[Type the abstract of the document here. The abstract is typically a short summary of the contents of the document. Type the abstract of the document here. The abstract is typically a short summary of the contents of the document.]

IT6413 - SOFTWARE ENGINEERING LABORATORY

OBJECTIVES:

- To understand the software engineering methodologies for project development.
- To gain knowledge about open source tools for Computer Aided Software Engineering.
- To develop an efficient software using case tools.

SOFTWARE REQUIRED:

- Open source Tools: StarUML / UMLGraph / Topcased

Prepare the following documents for each experiment and develop the software using software engineering methodology.

1. **Problem Analysis and Project Planning -**Thorough study of the problem –Identify Project scope, Objectives and Infrastructure.
2. **Software Requirement Analysis -** Describe the individual Phases/modules of the project and Identify deliverables.
3. **Data Modelling -** Use work products – data dictionary, use case diagrams and activity diagrams, build and test class diagrams, sequence diagrams and add interface to class
4. diagrams.
5. **Software Development and Debugging** – implement the design by coding
6. **Software Testing** - Prepare test plan, perform validation testing, coverage analysis, memory leaks, develop test case hierarchy, Site check and site monitor.

Sample Experiments:

Academic domain

1. Course Registration System
2. Student marks analyzing system

Railway domain

3. Online ticket reservation system
4. Platform assignment system for the trains in a railway station

Medicine domain

5. Expert system to prescribe the medicines for the given symptoms
6. Remote computer monitoring

Finance domain

7. ATM system
8. Stock maintenance

Human Resource management

9. Quiz System
10. E-mail Client system.

CHAPTER NO	TITLE	PAGE NO

A.	**SOFTWARE ENGINEERING PHASES**	**3**
	A. SOFTWARE DEVELOPMENT ACTIVITIES	3
	I. ANALYSIS PHASE	4
	II. DESIGN PHASE	4
	III.IMPLEMENTATION / CODING PHASE	6
	IV. TESTING	6
	V. DEPLOYMENT	6
	VI. MAINTENANCE	6
B.	**INTRODUCTION TO UML (UNIFIED MODELING LANGUAGE)**	**7**
	A. NOTATION ELEMENTS	8
	B. USE CASE DEFINITION	10
	C. CLASS DIAGRAM	10
	D. ACTIVITY DIAGRAM	11
	E. SEQUENCE DIAGRAM	12
	F. COLLABORATION DIAGRAM	12
C.	**SAMPLE EXPERIMENTS**	**13**
	A. COURSE REGISTRATION SYSTEM	13
	B. ONLINE TICKET RESERVATION SYSTEMS	19
	C. ATM SYSTEM	26
D.	**BEYOND THE SYLLABUS**	**33**
	A. JDBC CONNECTIVITY	33
	B. CREATING JDBC APPLICATION	34
	C. JDBC PROGRAM	35

CHAPTER 1. SOFTWARE ENGINEERING PHASES

Scope of Software Engineering: Software engineering is a discipline whose aim is the production of fault-free software that is delivered on time, within budget, and satisfies the user‟s needs. Software engineering is about teams. The problems to solve are so complex or large, that a single developer cannot solve them anymore. Software engineering is also about communication. Teams do not consist only of developers, but also of testers, architects, system engineers, customer, project managers, etc.

Software projects can be so large that we have to do careful planning. Implementation is no longer just writing code, but it is also following guidelines, writing documentation and also writing unit tests. But unit tests alone are not enough. The different pieces have to fit together. And we have to be able to spot problematic areas using metrics. They tell us if our code follows certain standards. Once we are finished coding, that does not mean that we are finished with the project: for large projects maintaining software can keep many people busy for a long time. Since there are so many factors influencing the success or failure of a project.

We also need to learn a little about project management and its pitfalls, but especially what makes projects successful. There are four fundamental phases in most, if not all, software engineering methodologies. These phases are analysis, design, implementation, and testing. These phases address what is to be built, how it will be built, building it, and making it high quality.

1.1. SOFTWARE DEVELOPMENT ACTIVITIES

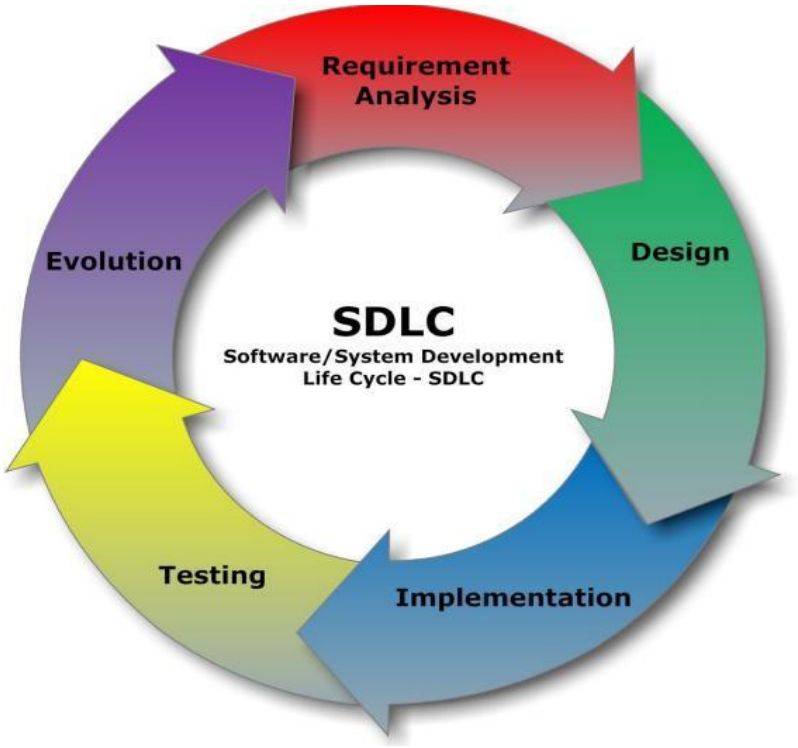


Fig 1.1. **Software Development Activities**

- Gathering Requirements, Software Design, Coding, Testing, Documentation & Software Maintenance.

I. ANALYSIS PHASE

The analysis phase defines the requirements of the system, independent of how these requirements will be accomplished. This phase defines the problem that the customer is trying to solve. The deliverable result at the end of this phase is a requirement document. Ideally, this document states in a clear and precise fashion what is to be built. This analysis represents the "what" phase. The requirement document tries to capture the requirements from the customer‟s perspective by defining goals and interactions at a level removed from the implementation details.

The requirement document may be expressed in a formal language based on mathematical logic. Traditionally, the requirement document is written in English or another written language. The requirement document does not specify the architectural or implementation details, but specifies information at the higher level of description. The problem statement, the customer‟s expectations, and the criteria for success are examples of high-level descriptions. There is a fuzzy line between high-level descriptions and low-level details.

Sometimes, if an exact engineering detail needs to be specified, this detail will also appear in the requirement document. This is the exception and should not be the rule. These exceptions occur for many reasons including maintaining the consistency with other established systems, availability of particular options, customer‟s demands, and to establish, at the requirement level, a particular architecture vision. An example of a low-level detail that might appear in the requirement document is the usage of a particular vendor‟s product line, or the usage of some accepted computer industry standard, or a constraint on the image size of the application behaviors.

The deliverable design document is the architecture. The design document describes a plan to implement the requirements. This phase represents the "how "phase. Details on computer programming languages and environments, machines, packages, application architecture, distributed architecture layering, memory size, platform, algorithms, data structures, global type definitions, interfaces, and many other engineering details are established.

II. DESIGN PHASE

The result of the software requirements analysis (SRA) usually is a specification. The design helps us turning this specification into a working system. As we have seen there are different kinds of software designs.

- *Architectural Design:* the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.
- *Detailed Design:* the process of refining and expanding the preliminary design of a system or component to the extent that the design is sufficiently complete to begin implementation.

- *Functional Design:* the process of defining the working relationships among the components of a system.

- *Preliminary Design:* the process of analyzing design alternatives and defining the architecture, components, interfaces, and timing/sizing estimates for a system or components.

There are many aspects to consider in the design of a piece of software. The importance of each should reflect the goals the software is trying to achieve. Some of these aspects are:

- *Compatibility* - The software is able to operate with other products that are designed for interoperability with another product. For example, a piece of software may be backward-compatible with an older version of itself.

- *Extensibility* - New capabilities can be added to the software without major changes to the underlying architecture.

- *Fault-tolerance* - The software is resistant to and able to recover from component failure.

- *Maintainability* - The software can be restored to a specified condition within a specified period of time. For example, antivirus software may include the ability to periodically receive virus definition updates in order to maintain the software's effectiveness.

- *Modularity* - the resulting software comprises well defined, independent components. That leads to better maintainability. The components could be then implemented and tested in isolation before being integrated to form a desired software system. This allows division of work in a software development project.

- *Packaging* - Printed material such as the box and manuals should match the style designated for the target market and should enhance usability. All compatibility information should be visible on the outside of the package. All components required for use should be included in the package or specified as a requirement on the outside of the package.

- *Reliability* - The software is able to perform a required function under stated conditions for a specified period of time.

- *Reusability* - the software is able to add further features and modification with slight or no modification.

- *Robustness* - The software is able to operate under stress or tolerate unpredictable or invalid input. For example, it can be designed with resilience to low memory conditions.

- *Security* - The software is able to withstand hostile acts and influences.

- *Usability* - The software user interface must be usable for its target user/audience. Default values for the parameters must be chosen so that they are a good choice for the majority of the users.

III. IMPLEMENTATION / CODING PHASE

On receiving system design documents, the work is divided in modules/units and actual coding is started. Since, in this phase the code is produced so it is the main focus for the developer. This is the longest phase of the software development life cycle. Implementation is the part of the process where software engineers actually program the code for the project. The implementation phase deals with issues of quality, performance, baselines, libraries, and debugging. The end deliverable is the product itself.

IV. TESTING

After the code is developed it is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase. During this phase unit testing, integration testing, system testing, acceptance testing are done.

V. DEPLOYMENT

After successful testing the product is delivered / deployed to the customer for their use.

VI. MAINTENANCE

Once when the customers starts using the developed system then the actual problems comes up and needs to be solved from time to time. This process where the care is taken for the developed product is known as maintenance.

CHAPTER 2. INTRODUCTION TO UML (UNIFIED MODELING LANGUAGE)

The UML is a language for specifying, constructing, visualizing, and documenting the software system and its components. The UML is a graphical language with sets of rules and semantics. The rules and semantics of a model are expressed in English in a form known as OCL (Object Constraint Language). OCL uses simple logic for specifying the properties of a system. The UML is not intended to be a visual programming language. However it has a much closer mapping to object-oriented programming languages, so that the best of both can be obtained. The UML is much simpler than other methods preceding it. UML is appropriate for modeling systems, ranging from enterprise information system to distributed web based application and even to real time embedded system. It is a very expensive language addressing all views needed to develop and then to display system even though understand to use. Learning to apply UML effectively starts forming a conceptual mode of languages which requires learning.

Three major language elements:

- UML basic building blocks
- Rules that dictate how this building blocks put together
- Some common mechanism that apply throughout the language The primary goals in the design of UML are:

1. Provides users ready to use, expressive visual modeling language as well 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 formal basis for understanding the modeling language.
5. Encourage the growth of the OO tools market.
6. Support higher-level development concepts.
7. Integrate best practices and methodologies.

Every complex system is best approached through a small set of nearly independent views of a model. Every model can be expressed at different levels of fidelity. The best models are connected to reality. The UML defines nine graphical diagrams:

1. Class diagram
2. Use-case diagram
3. Behavior diagram
    - 3.1. Interaction diagram
        - 3.1.1. Sequence diagram
        - 3.1.2. Collaboration diagram
    - 3.2. State chart diagram
    - 3.3. Activity diagram
4. Implementation diagram
    - 4.1 component diagram
    - 4.2 deployment diagram

1. *UML class diagram:*

   The UML class diagram is also known as object modeling. It is a static analysis diagram. These diagrams show the static structure of the model. A class diagram is a connection of static model elements, such as classes and their relationships, connected as a graph to each other and to their contents.

2. *Use-case diagram:*

   The functionality of a system can be described in a number of different use cases, each of which represents a specific flow of events in a system. It is a graph of actors, a set of use-cases enclosed in a boundary, communication, associations between the actors and the use-cases, and generalization among the use-cases.

3. *Behavior diagram:*

   It is a dynamic model unlike all the others mentioned before. The objects of an object oriented system are not static and are not easily understood by static diagrams. The behavior of the class‟s instance (an object) is represented in this diagram. Every use-case of the system has an associated behavior diagram that indicates the behavior of the object. In conjunction with the use-case diagram we may provide a script or interaction diagram to show a time line of events. It consists of sequence and collaboration diagrams

4. *Interaction diagram*

   It is the combination of sequence and collaboration diagram. It is used to depict the flow of events in the system over a timeline. The interaction diagram is a dynamic model which shows how the system behaves during dynamic execution.

5. *State chart diagram:*

   It consists of state, events and activities. State diagrams are a familiar technique to describe the behavior of a system. They describe all of the possible states that a particular object can get into and how the object's state changes as a result of events that reach the object. In most OO techniques, state diagrams are drawn for a single class to show the lifetime behavior of a single object.

6. *Activity diagram:*

   It shows organization and their dependence among the set of components. These diagrams are particularly useful in connection with workflow and in describing behavior that has a lot of parallel processing. An activity is a state of doing something: either a real-world process, or the execution of a software routine.

7. *Implementation diagram:*

   It shows the implementation phase of the systems development, such as the source code structure and the run-time implementation structure. These are relatively simple high level diagrams compared to the others seen so far. They are of two sub diagrams, the component diagram and the deployment diagram.

8. *Component diagram:*

   These are organizational parts of a UML model. These are boxes to which a model can be decomposed. They show the structure of the code itself. They model the physical components such as source code, user interface in a design. It is similar to the concept of packages.

9. *Deployment diagram:*

The deployment diagram shows the structure of the runtime system. It shows the configuration of runtime processing elements and the software components that live in them. They are usually used in conjunction with deployment diagrams to show how physical modules of code are distributed on the system.

A. NOTATION ELEMENTS:

These are explanatory parts of UML model. They are boxes which may apply to describe and remark about any element in the model. They provide the information for understanding the necessary details of the diagrams.

Relations in the UML:

These are four kinds of relationships used in an UML diagram, they are:

Dependency
Association
Generalization
Realization

Dependency:

It is a semantic relationship between two things in which a change one thing affects the semantics of other things. Graphically a dependency is represented by a non-continuous line.

Association:

It is a structural relationship that describes asset of links. A link is being connected among objects. Graphically association is represented as a solid line possibly including label.

Generalization:

It is a specialized relationship in which the specialized elements are substitutable for object of the generalized element. Graphically it is a solid line with hollow arrow head parent.

Realization:

It is a semantic relation between classifiers. Graphically it is represented as a cross between generalization and dependency relationship.

Where UML can be used:

UML is not limited to modeling software. In fact it is expressive to model non-software such as to show in structure and behavior of health case system and to design the hardware of the system.

Conceptual model be UML:

UML you need to form the conceptual model of UML. This requires three major elements:

- UML basic building blocks.
- Rules that dictate how this building blocks are put together.
- Some common mechanism that apply throughout the language.

Once you have grasped these ideas, you may be able to read. UML create some basic ones. As you gain more experience in applying conceptual model using more advanced features of this language.

Building blocks of the UML:

The vocabulary of UML encompasses these kinds of building blocks.

B. USE CASE DEFINITION:

Description:

A use case is a set of scenarios tied together by a common user goal. A use case is a behavioral diagram that shows a set of use case actions and their relationships.

Purpose:

The purpose of use case is login and exchange messages between sender and receiver (Email client).

Main flow:

First, the sender gives his id and enters his login. Now, he enters the message to the receiver id.

Alternate flow:

If the username and id by the sender or receiver is not valid, the administrator will not allow entering and "Invalid password" message is displayed.

Pre-condition:

A person has to register himself to obtain a login ID.

Post-condition:

The user is not allowed to enter if the password or user name is not valid.

C. CLASS DIAGRAM:

Description:

A class diagram describes the type of objects in system and various kinds of relationships that exists among them. Class diagrams and collaboration diagrams are alternate representations of object models. During analysis, we use class diagram to show roles and responsibilities of entities that provide email client system behaviors design. We use to capture the structure of classes that form the email client system architecture.

The classes used in system are:

1. user
2. login

A class diagram is represented as:

<<Class name>>

<<Attribute 1>>

<<Attribute n>>

<<Operation ()>>

Relationship used:

A change in one element affects the other

Generalization:

It is a kind of relationship

State chart:
Description:

The state chart diagram made the dynamic behavior of individual classes. State chart shows the sequences of states that an object goes through events and state transitions.

A state chart contains one state „start" and multiple „end" states.

The important objectives are: Decision:

It represents a specific location state chart diagram where the work flow may branch based upon guard conditions.

Synchronization:

It gives a simultaneous workflow in a state chart diagram. They visually define forks and joints representing parallel workflow.

Forks and joins:

A fork construct is used to model a single flow of control.

Every work must be followed by a corresponding join.

Joints have two or more flow that unit into a single flow.

State:

A state is a condition or situation during a life of an object in which it satisfies condition or waits for some events.

Transition:

It is a relationship between two activities and between states and activities.

Start state:

A start state shows the beginning of a workflow or beginning of a state machine on a state chart diagram.

End state:

It is a final or terminal state.

D. ACTIVITY DIAGRAM

Description:

Activity diagram provides a way to model the workflow of a development process. We can also model this code specific information such as class operation using activity diagram. Activity diagrams can model different types of diagrams.

There are various tools involved in the activity diagram.

Activity:

An activity represents the performance of a task on duty. It may also represent the execution of a statement in a procedure.

Decision:

A decision represents a condition on situation during the life of an object, which it satisfies some condition or waits for an event.

Start state:

It represents the condition explicitly the beginning of a workflow on an activity.

Object flow:

An object on an activity diagram represents the relationship between activity and object that creates or uses it.

Synchronization:

It enables us to see a simultaneous workflow in an activity.

End state:

An end state represents a final or terminal state on an activity diagram or state chart diagram.

E. SEQUENCE DIAGRAM:

Description:

A sequence diagram is a graphical view of scenario that shows object interaction in a time based sequence what happens first what happens next. Sequence diagrams are closely related to collaboration diagram. The main difference between sequence and collaboration diagram is that sequence diagram show time based interaction while collaboration diagram shows objects associated with each other. The sequence diagram for the e-mail client system consists of the following objectives:

Object:

An object has state, behavior and identity. An object is not based is referred to as an instance.

The various objects in e-mail client system are:

User

Website
Login
Groups

Message icon:

A message icon represents the communication between objects indicating that an action will follow. The message icon is the horizontal solid arrow connecting lifelines together.

F. COLLABORATION DIAGRAM:

Description:

Collaboration diagram and sequence diagrams are alternate representations of an interaction. A collaboration diagram is an interaction diagram that shows the order of messages that implement an operation or a transaction. Collaboration diagram is an interaction diagram that shows the order of messages that implement an operation or a transaction. Collaboration diagram shows object s, their links and their messages.

They can also contain simple class instances and class utility instances. During, analysis indicates the semantics of the primary and secondary interactions. Design, shows the semantics of mechanisms in the logical design of system.

CHAPTER 3. SAMPLE EXPERIMENTS

Ex. No.: 1 COURSE REGISTRATION SYSTEM

Aim

To create a system through which students can register to the courses desired by them.

Problem statement

- The system is built to be used by students and managed by an administrator.
- The student and employee have to login to the system before any processing can be done.
- The student can see the courses available to him and register to the course he wants.
- The administrator can maintain the course details and view all the students who have registered to any course.

USE-CASE DIAGRAM

The course registration system has the following use-cases

- Login
- View course details
- Registration
- Display details
- Maintain course details
- Logout

The actors involved in the system are

1. Student
2. Administrator

Use-case name: Login

The user enters the username and password and chooses if the user is student or administrator. If entered details are valid, the user‟s account becomes available. If it is invalid, an appropriate message is displayed to the user.

Use-case name: View course details

In this use case, a student can search all the courses available to him and choose the best course he wants. The student can view the course duration, faculty and department of the courses he may choose.

Use-case name: Registration

When a student has successfully chosen a course, he can register to that course. Upon registration, the student‟s details are stored in the database.

Use-case name: Display details

After registration to any course, the student may see the details of his current course. He may wish to know details about fees and other information. The administrator also has the privilege to display details of the the students and the corresponding course for which they have registered.

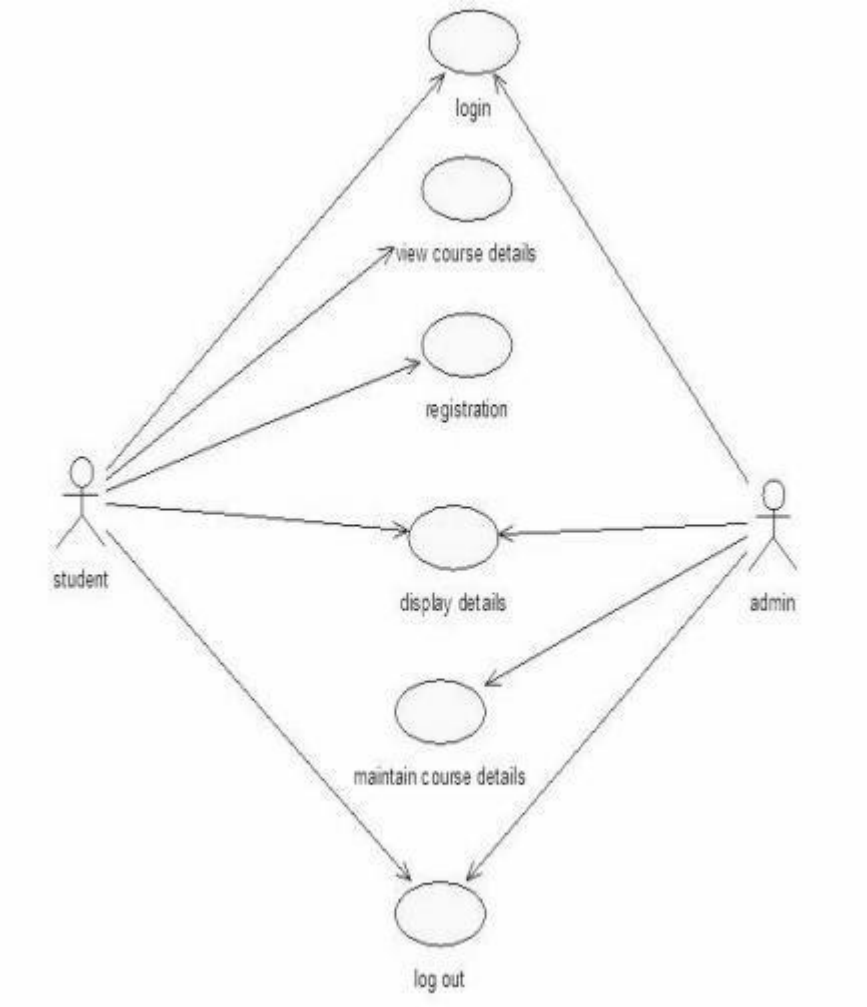
Use-case name: Maintain course details

The administrator has to perform the duties of maintaining the course details. Any change to the course structure is maintained by the administrator.

Use-case name: Logout

After all the desired transactions are made, the user may choose to logout from the system to save all he changes they have made.

Use-case diagram for course registration system



CLASS DIAGRAM

The class diagram is a graphical representation of all the classes used in the system and their operations, attributes and relationships.

The course registration system makes use of the following classes:

1. Stud (student details)
2. Administrator

1) Student

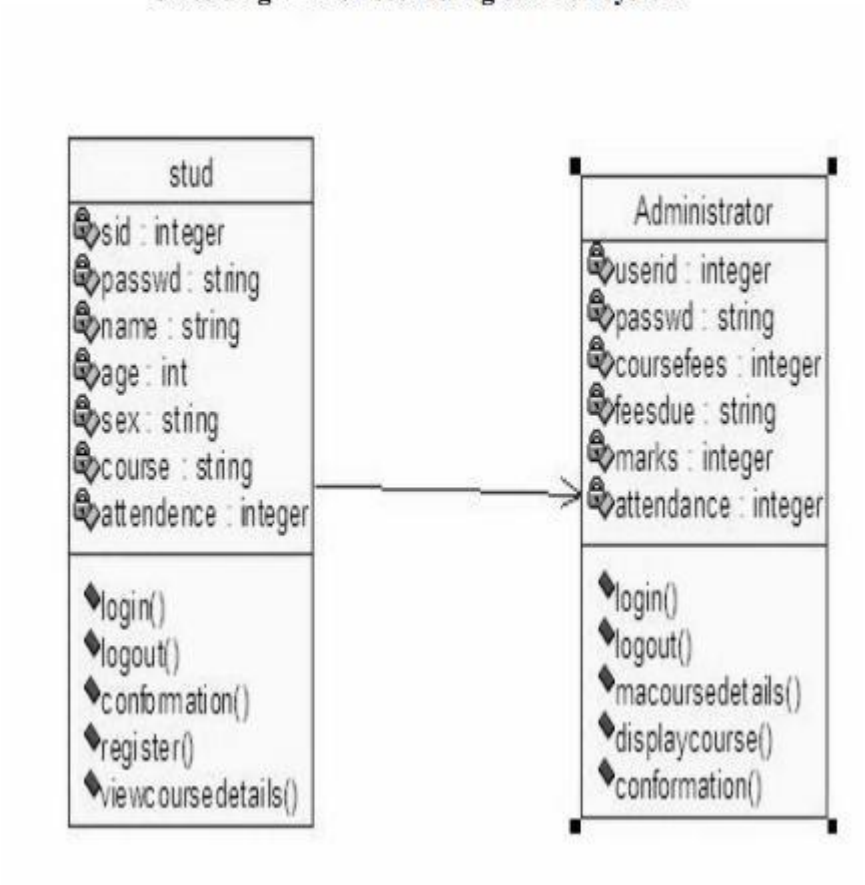
It consists of the details of all the students present in the database. The attributes present in this class are student id, password, name, age, sex, course and attendance. The object of this class is created as soon as the student registers to a course. The operations available to this class are login (), logout (), confirmation (), register (), and view course details ().

2) Administrator

It consists of details of all the courses available to the student. The attributes present in this class are username, password, course fees, fees due, marks, and attendance. The

operations available to this class are login (), logout (), ma course details (), display course (), and confirmation ().

Class diagram for course registration system

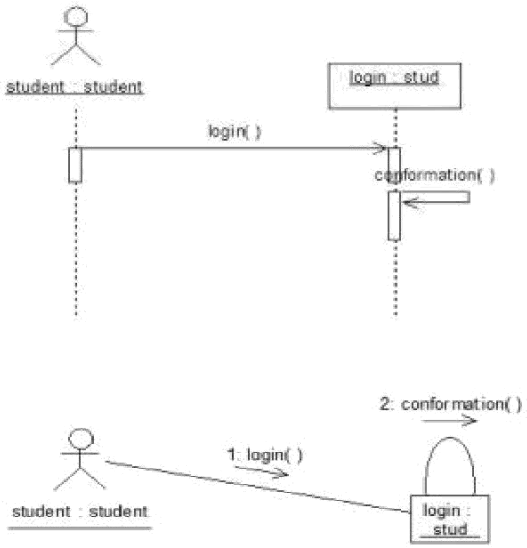


SEQUENCE DIAGRAM

A sequence diagram represents the sequence and interactions of a given usecase or scenario. Sequence diagrams can capture most of the information about the system. Most object-to-object interactions and operations are considered events and events include signals, inputs, decisions, interrupts, transitions and actions to or from users or external devices.

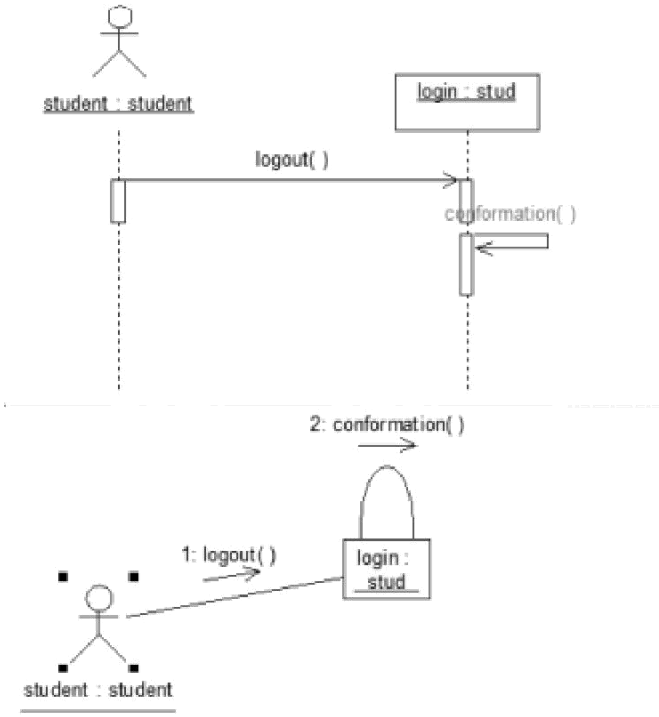
An event also is considered to be any action by an object that sends information. The event line represents a message from one object to another, in which the “from” object is requesting an operation be performed by the “to” object. The “to” object performs the operation using a method that the class contains. It is also represented by the order in which things occur and how the objects in the system send message to one another. The sequence diagram for each use-case that exists when a user logs in, adds, views, updates or deletes records in the system

Sequence and collaboration diagram for login to the system



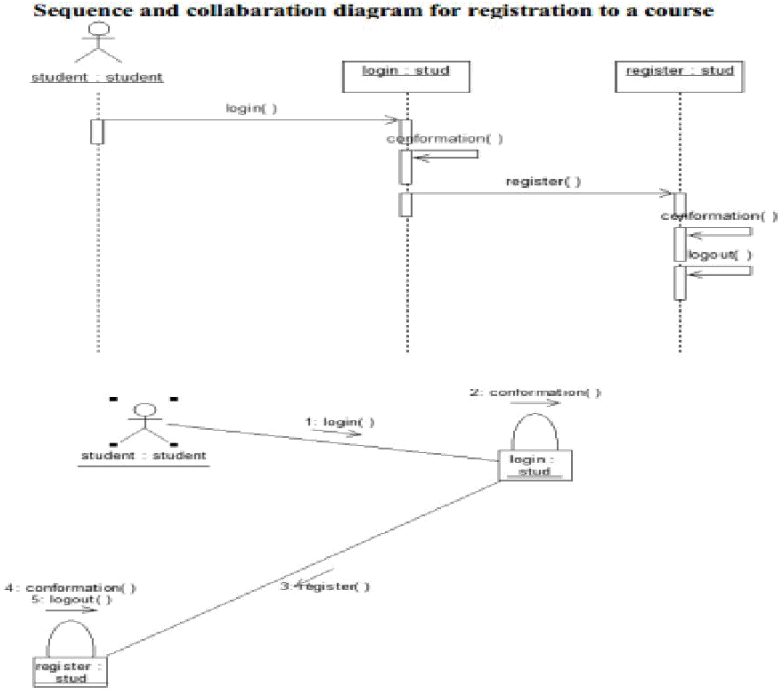
Users have to first login to the system before performing any operation. The user has to provide the necessary details to the system for login.

Sequnce and collabaration diagram for logout



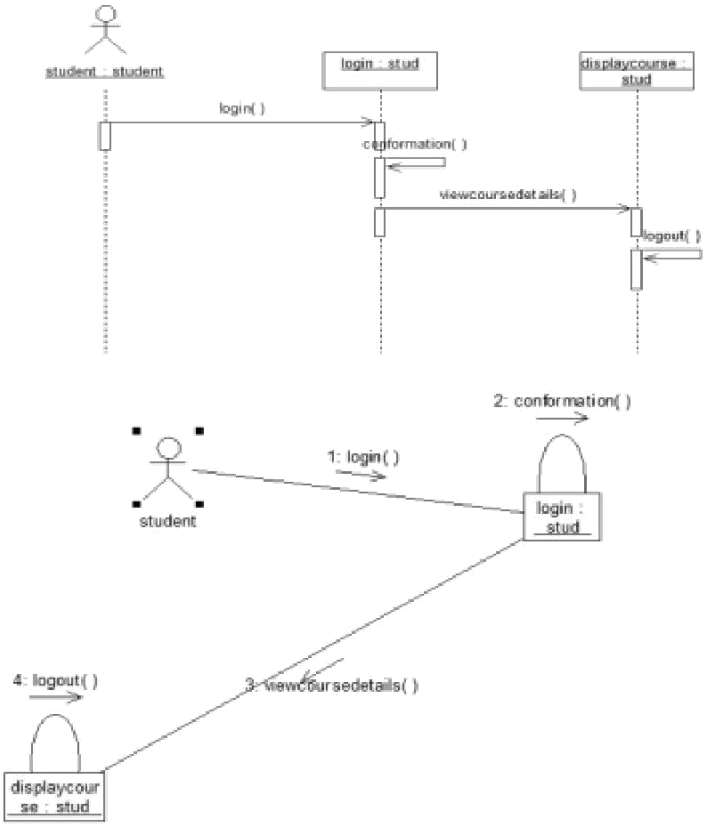
When the necessary operations have been performed on the system, the user may choose to save the changes and logout from the system.

Sequence and collabaration diagram for registration to a course



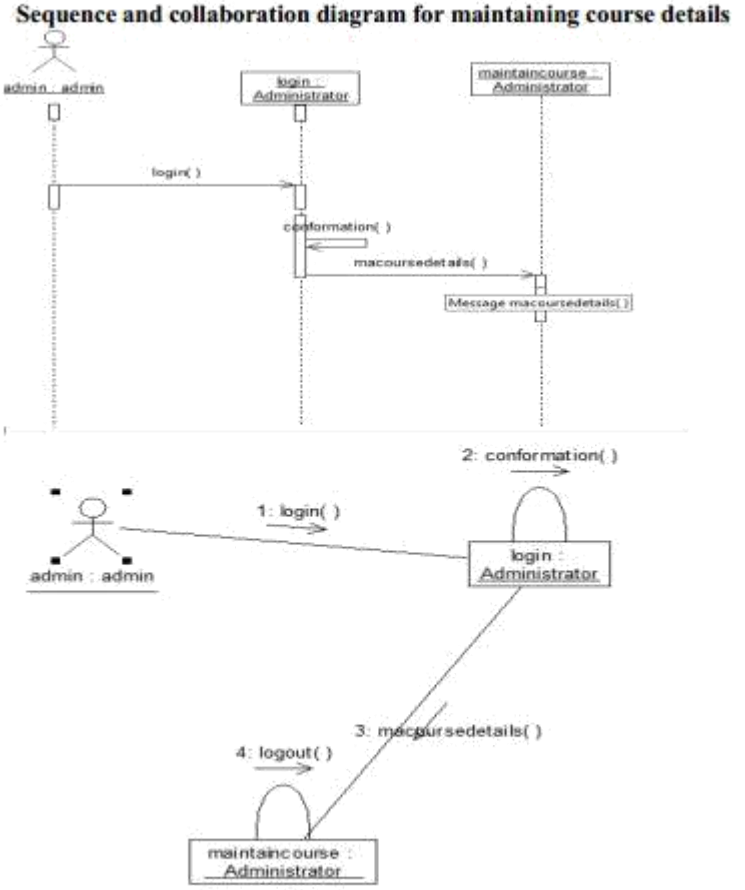
After login, the student has to register to a course of his choice. The student can view all the courses available to him and register to a course suitable to him. The student may view the course details before registration.

Sequence and collaboration diagram for viewing course details



A student may wish to view course details before registration. For this, the student has to first login and select the course details he wishes to see.

Sequence and collaboration diagram for maintaining course details



Course details may be changed as per the requirement every year. So the administrator can edit the details of the course as necessary.

RESULT:

Thus the documentation for course registration system is created. The output is verified.

Ex. No.: 2 ONLINE TICKET RESERVATION SYSTEMS

Introduction:

The manual system of ticket reservation takes more time and the number of reservations per day is limited. To increase the efficiency of the process, we go for online ticket reservation system. This system supports online ticket booking.

Problem statement

This system is built for user to directly access the system online to book tickets. The user can book, print, delete tickets without the help of a clerk. The administrator has control over the adding train available for booking and has control over deleting train that are not necessary. The administrator and user can both enter the system using their respective login details

Use-case diagram

The online ticket reservation system uses the following use cases:

1. Login
2. Book ticket
3. Print ticket
4. Cancel ticket
5. View train
6. Add train
7. Delete train
8. Logout

Actors involved

1) Administrator
2) Passenger

Use-case name: login

The user enters a username and a password. And if the entered details are valid, the user‟s details are brought to the screen; if they are invalid then an appropriate message is displayed.

Use-case name: Book ticket

The user is allowed to book a ticket on the train he/she requires and the date and time as is necessary for the user. The user has to provide details such as name, train number, date of travel, departure time, and can view the price of the ticket.

Use-case name: Print ticket

The user after booking a ticket can print a copy of the ticket reserved. The user has to provide the details about ticket number for searching in the database and passenger name for confirming passenger identity.

Use-case name: Cancel ticket

A passenger can decide to cancel a ticket after the ticket is booked. The passenger has to provide details about ticket for searching and details about him for confirmation of identity.

Use-case name: View Train

The passenger can view the train availability in the database for deciding which trains ticket he/she wishes to book. The passenger can view the details of trains such as, train number, Train name, price, departure and arrival times.

Use-case name: Add train

Only the administrator has privilege to add train. The administrator can add the train on which tickets can be booked by the passengers. The administrator has to provide details about a new train such as train number, train name, price, departure time, date of travel.

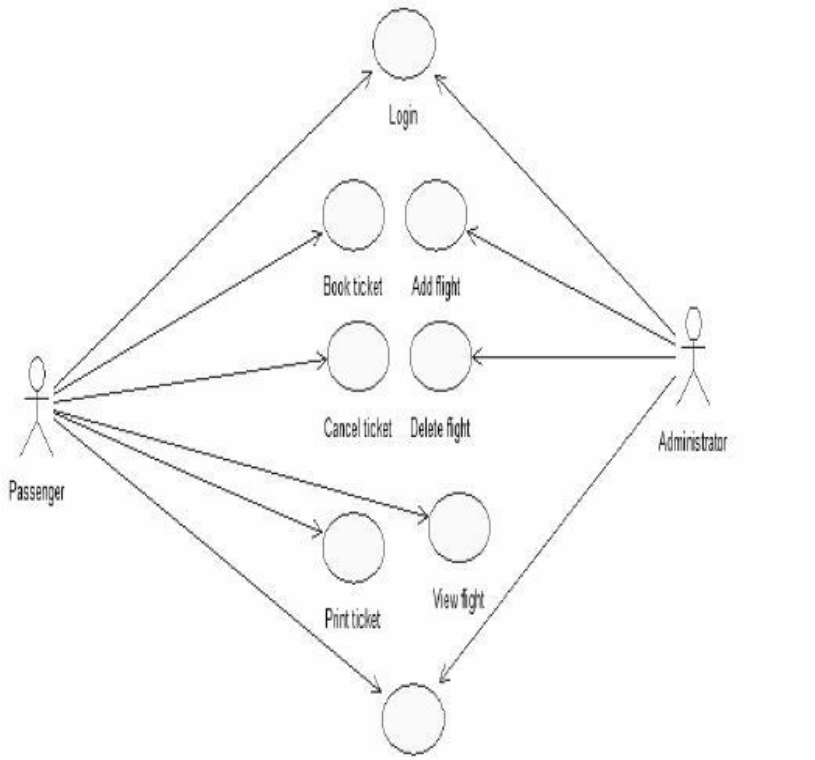
Use-case name: Delete Train

The administrator also has the privilege to delete train that is not necessary. The administrator has to provide details about the train for searching and inform any passengers that have booked tickets on the train about the change and make necessary arrangements.

Use-case name: Logout

After the necessary operations have been performed on the system, the user can choose to logout from the system.

Use-case diagram for airline reservation



Class diagram

The class diagram is a graphical representation of all the classes used in the system and their operations, attributes and relationships.

The online ticket reservation system makes use of the following classes:

- Ticket system
- Train details
- Ticket

Ticket system

It consists of two attributes and two operations. The attributes are username and password. The operations used are login () and logout ().

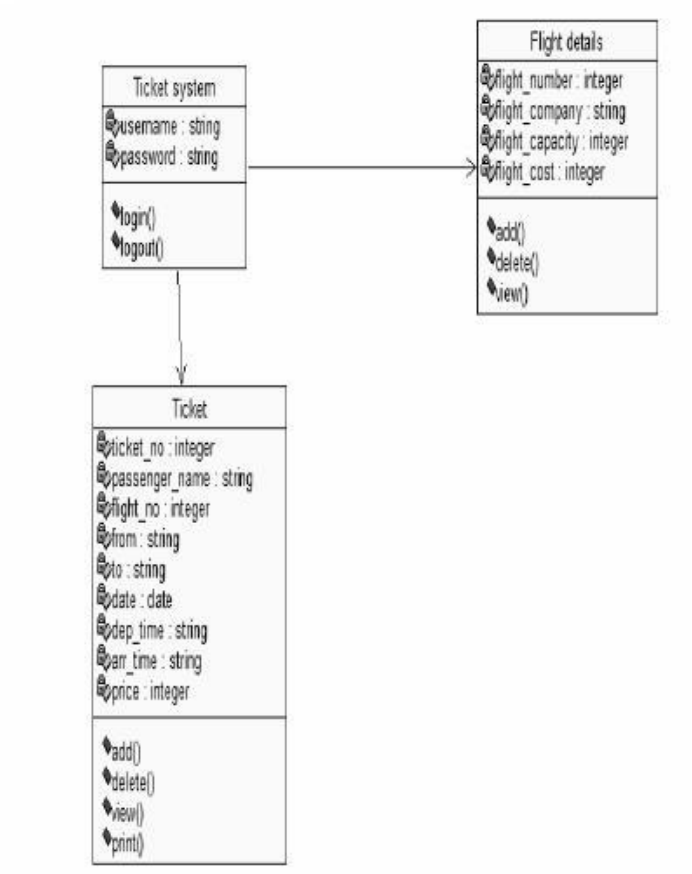
Train details

It stores the details of all the train such as train number, train name train capacity, and cost. The operations available are add (), delete () and view ().

Ticket

It records the details of every ticket booked such as ticket number, passenger name, and train number, from place, to place, date of travel, departure time, arrival time, and price. The operations available are add (), delete (), view (), and print ().

Class diagram for airline reservation system



Sequence diagram

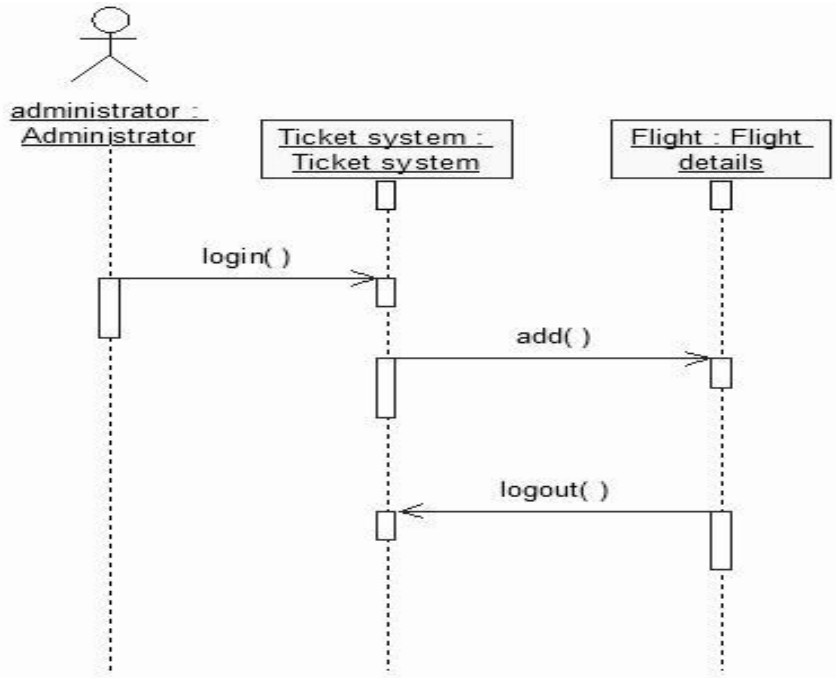
A sequence diagram represents the sequence and interactions of a given use case or scenario. Sequence diagrams can capture most of the information about the system. Most object-to-object interactions and operations are considered events and events include signals, inputs, decisions, interrupts, transitions and actions to or from users or external devices.

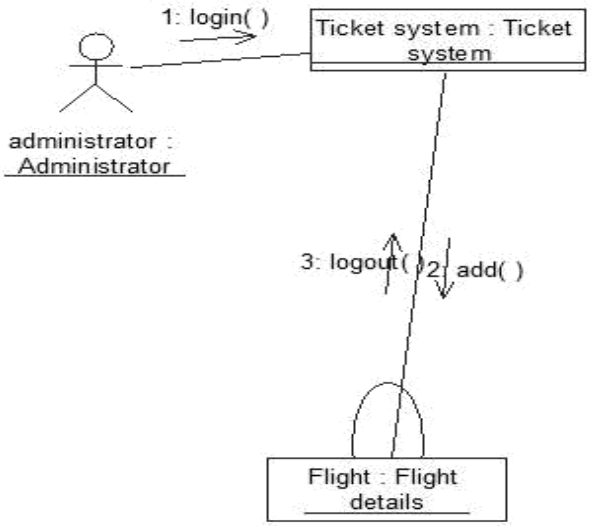
An event also is considered to be any action by an object that sends information. The event line represents a message from one object to another, in which the "from" object is requesting an operation be performed by the "to" object. The "to" object performs the operation using a method that the class contains.

It is also represented by the order in which things occur and how the objects in the system send message to one another.

The sequence diagram for each use-case that exists when a user logs in, adds, views, updates or deletes records in the system.

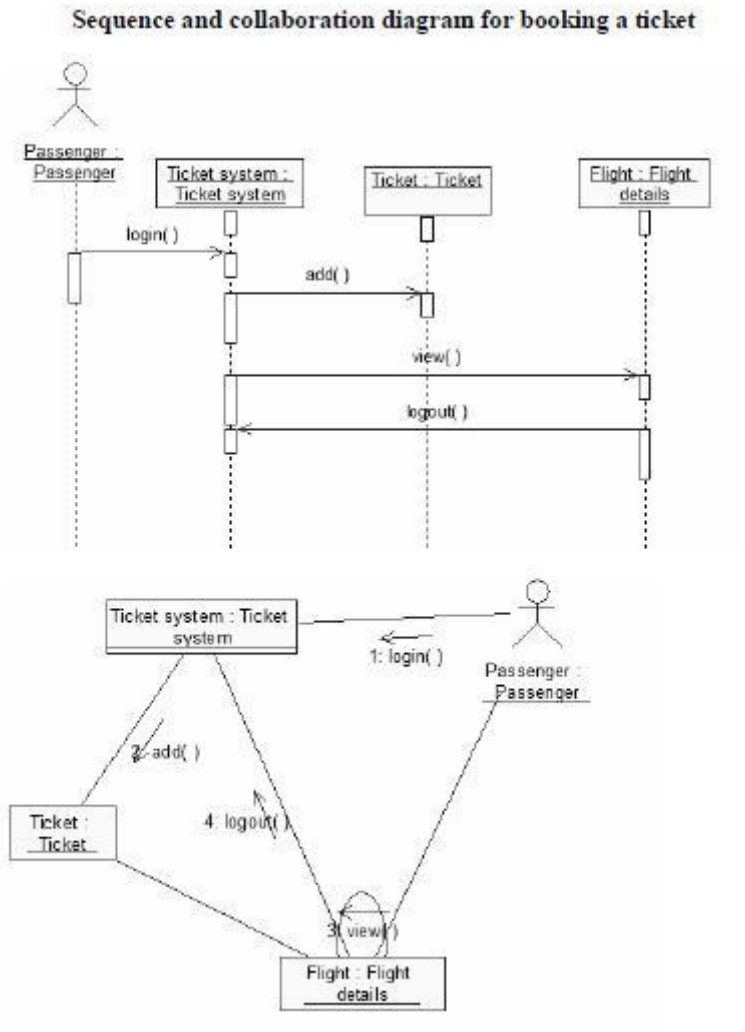
Sequence and collaboration diagram for adding a train





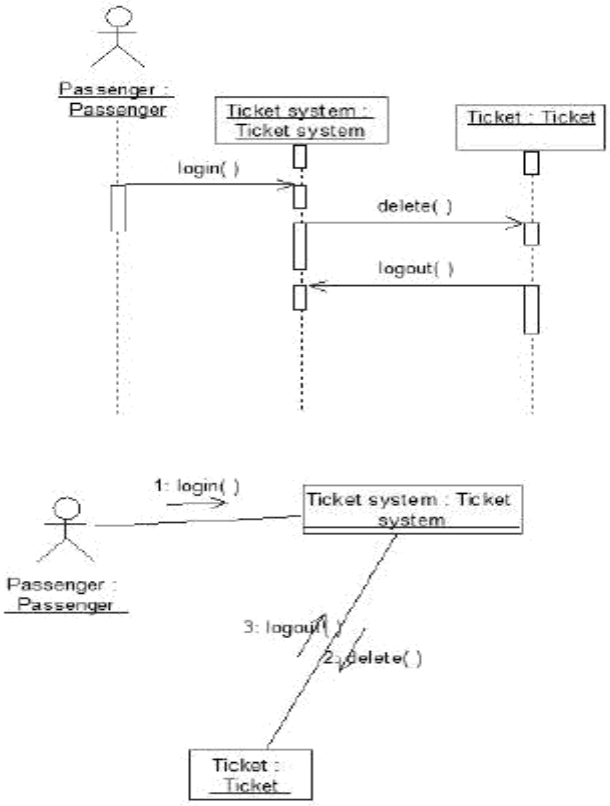
The Administrator has the privilege to add train. He/she has to provide details about the new train that is being created in the database.

Sequence and collaboration diagram for booking a ticket



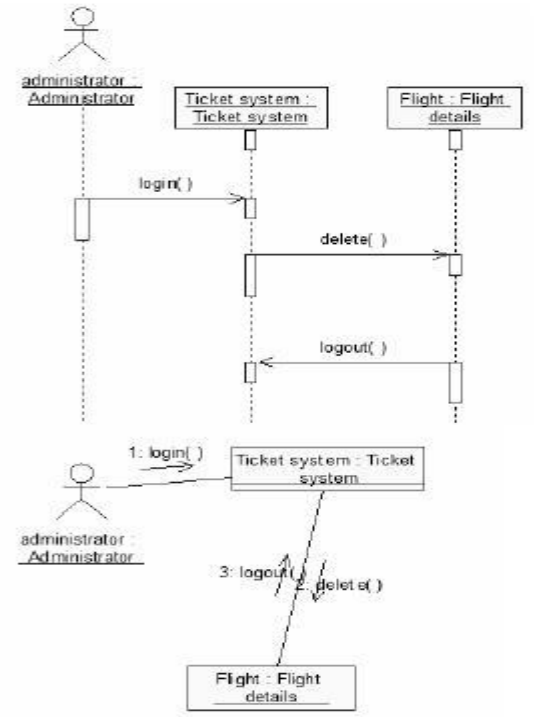
A passenger can book a ticket by himself. He/she can view the train details he/she wants to book a ticket on and as per his necessity may book an appropriate ticket.

Sequence and collaboration diagram for canceling a ticket



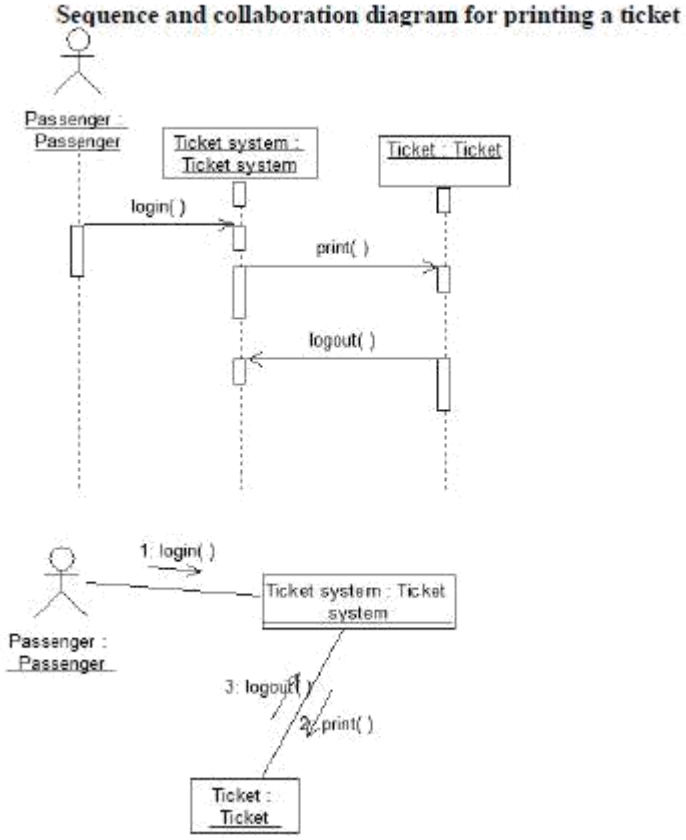
Canceling a ticket can be performed by the passenger himself. He/she may view the ticket he/she wants and cancel it.

Sequence and collaboration diagram for deleting a flight



A train can be deleted only by the administrator. The train to be deleted is selected and removed from the database.

Sequence and collaboration diagram for printing a ticket



The passenger may choose to print a ticket after booking a ticket. The ticket which is booked is selected and printed by the passenger.

RESULT:

Thus the documentation for train reservation system is created. The output is verified.

EX. NO.:3 ATM SYSTEMS

Aim:

To create a system to perform Bank ATM transaction

Problem statement:

- This system is build for the bank client and the manager.
- The bank client must be able to deposit and withdraw amount from his/her accounts using the ATM machine. Each transaction must be recorded and the client must be able to review all transactions performed in his/her account. Recorded transactions must include the date, time, transaction type, amount and account balance after the transaction.
- The bank manager must be able to view the ATM machine status that is the total balance of the ATM machine, today‟s withdrawal, today‟s balance and the limitations of the machine.
- The bank client is provided by login verification. If it is valid he/she will access their account otherwise an appropriate message is displayed to the client.

USE-CASE diagram:

The ATM transaction use cases in our system are:

1. Login
2. Withdraw
3. Mini statement
4. ATM machine status
5. Deposit

Actors involved:

1. User
2. Bank manager

USE-CASE name: Login

The user enters a user name and password. If it is valid, the user‟s account becomes available. If it is invalid, an appropriate message is displayed to the user.

USE-CASE name: Withdraw

The user tries to withdraw an amount from his or her checking account. The amount is less than or equal to the checking account‟s balance, the transaction is performed and the available information is displayed. The system creates a record of the transaction and the display confirmation message is displayed to the client.

USE-CASE name: Mini statement

The bank user requests a history of transactions for a checking account. The system displays the transaction history for the checking account. The transaction history consists of amount, date, transaction type and balance of the particular account.

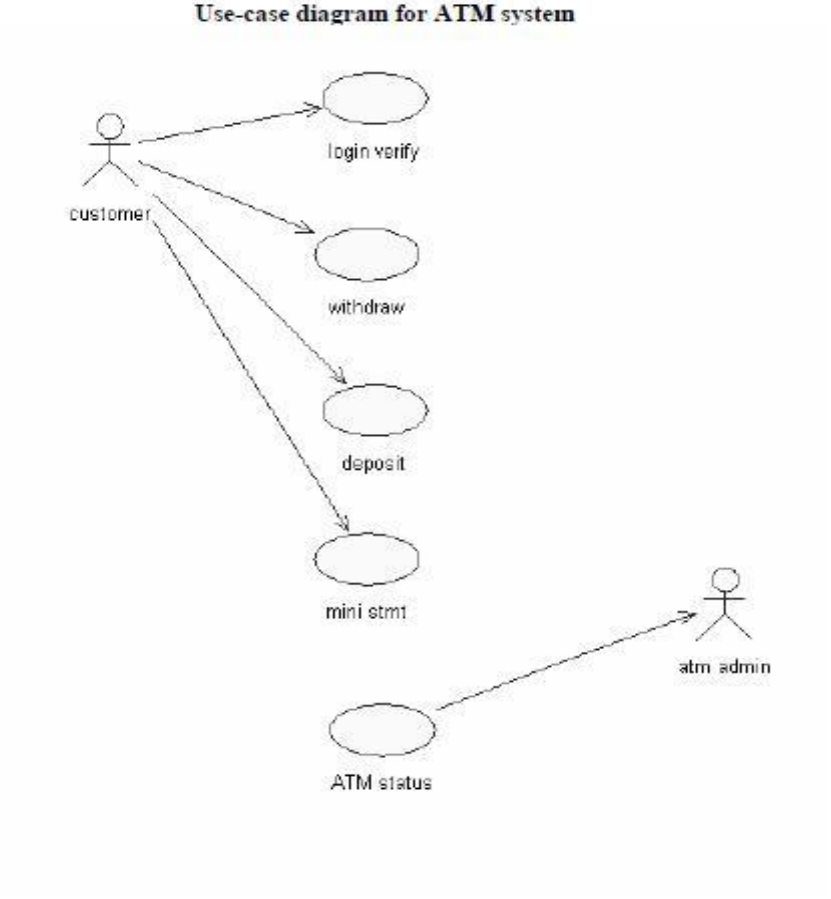
USE-CASE name: ATM machine status

The bank manager enters a username and password. If it is valid, the bank manager accesses the machine status. If it is invalid, an appropriate message is displayed to the user.

USE-CASE name: Deposit

The bank user requests the system to deposit money to an account. The user accesses the account for which a deposit is going to be made and enters the amount. The system creates a record of the transaction and an appropriate confirmation message (display confirmation) is displayed to the client. The transaction must include the date, type, amount and account balance after the transaction.

Use-case diagram for ATM system



CLASS DIAGRAM

The class diagram, also referred to as object modeling is the main static analysis diagram. The main task of object modeling is to graphically show what each object will do in the problem domain. The problem domain describes the structure and the relationships among objects.

The ATM system class diagram consists of four classes:

1. User class
2. ATM machine status
3. Account
4. Transaction

1) User class:

It consists of four attributes and two operations. The attributes are user name, password, address and DOB. The operations of this class are read (), display () and write ().

2) ATM machine status:

The attributes of this class are ATM balance, today‟s withdrawal, today‟s balance, and limitations. The operations are login verification (), ATM status () and display confirmation ().

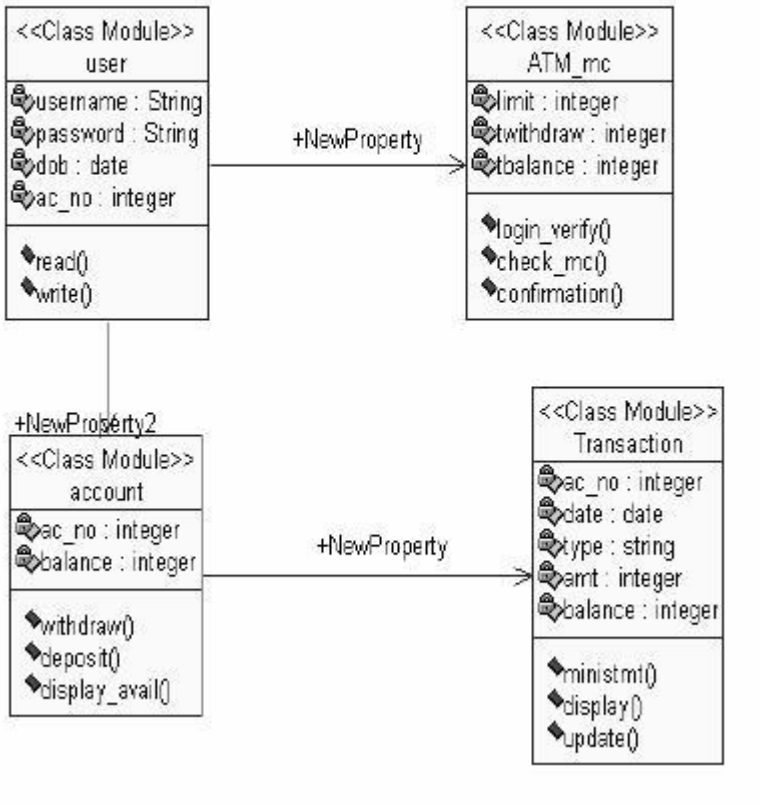
3) Account:

The attributes are account no. and balance and the operations are withdraw (), deposit () and display availability ().

4) Transaction:

The attributes of this class are account no, transaction type, data, amount, balance and the operations are mini statement () and create transaction ().

Class diagram for ATM system



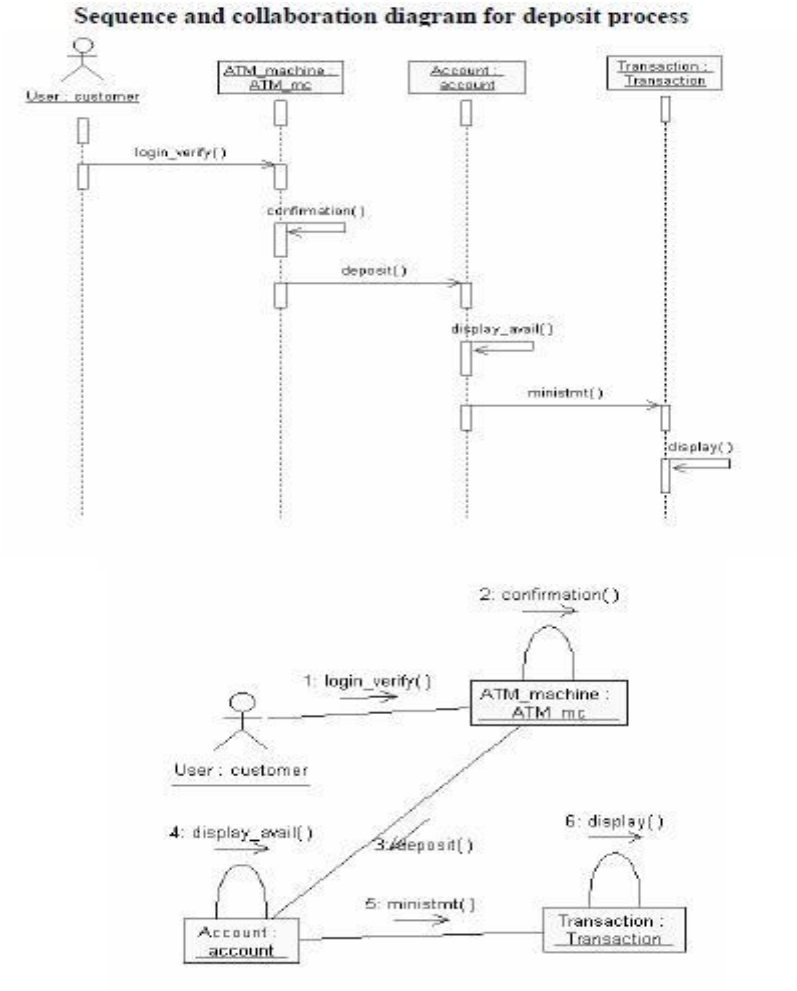
SEQUENCE DIAGRAM:

A sequence diagram represents the sequence and interactions of a given USECASE or scenario. Sequence diagrams can capture most of the information about the system. Most object to object interactions and operations are considered events and events include signals, inputs, decisions, interrupts, transitions and actions to or from users or external devices.

An event also is considered to be any action by an object that sends information. The event line represents a message sent from one object to another, in which the "form" object is requesting an operation be performed by the "to" object.

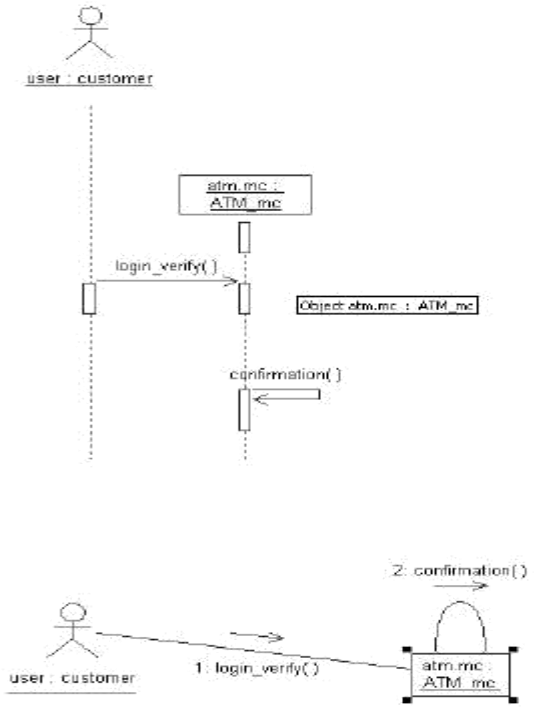
The "to" object performs the operation using a method that the class contains. It is also represented by the order in which things occur and how the objects in the system send message to one another.

The sequence diagram for each USE-CASE that exists when a user withdraws, deposits, needs information about ATM machine status and account are drowned.



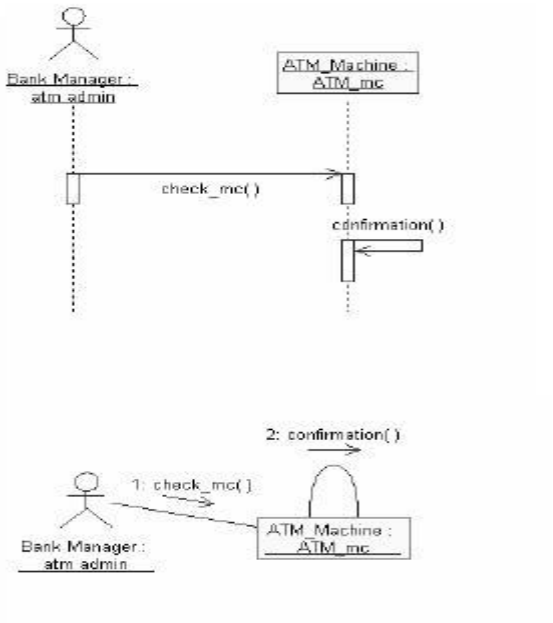
The diagrams show the entire deposit process in an ATM system. The user has to login to the ATM machine and deposit the amount of money as required by the user. The user may wish to get a mini statement and a screen about the details of the transaction.

Sequence and collaboration diagram for login



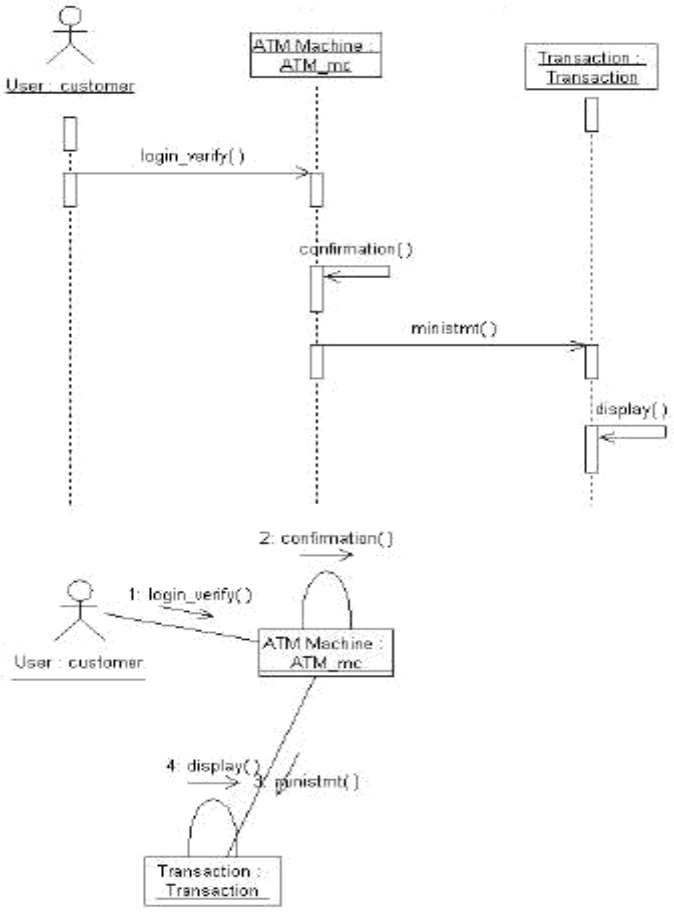
The diagrams show the process of login by the user to the ATM system. The user has to enter his details. The details entered are verified by the system and the user is approved if the details match, otherwise an appropriate error message is displayed.

Sequence and collaboration diagram for checking machine status



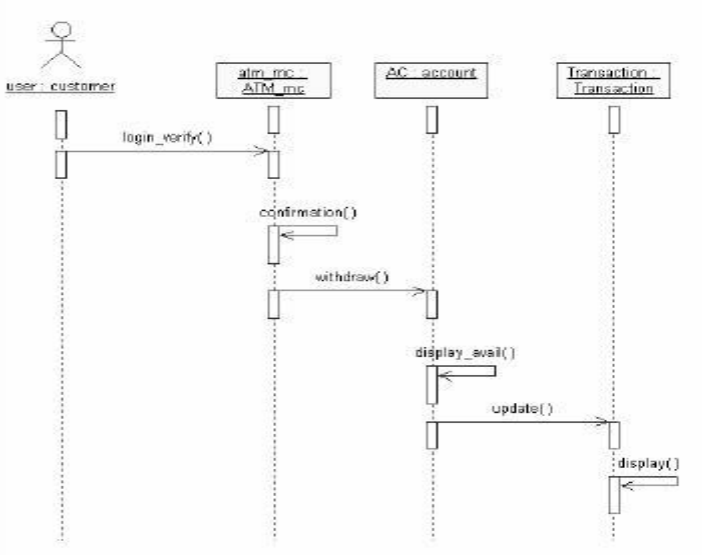
The Administrator of the ATM system has to maintain the details about the ATM, He has to check if there is enough money in the ATM and if the ATM is functional without major errors. For this, he may check the ATM machine status occasionally. The process is shown in the above diagrams.

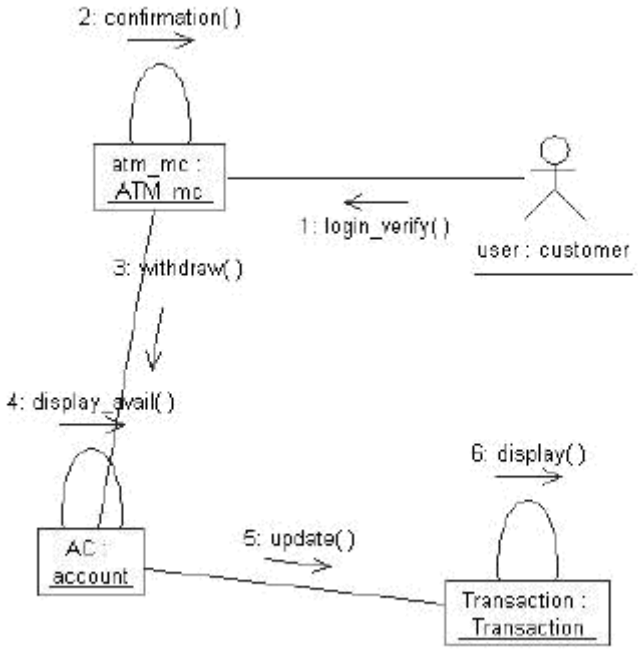
Sequence and collaboration diagram for printing ministatement



After a transaction is carried out successfully, the user must get a mini statement to tell him his account‟s details such as balance and transaction number. This process is depicted in the above diagrams.

Sequence and collaboration diagram for withdraw process





The user can make withdraw money from his account. The process is depicted in the diagrams above. The user has to login to the system using his username and password, which are verified by the system. After successful verification, the user can choose the amount of money he wants to withdraw from his account. The amount specified by the user is checked by the system to make sure there is enough balance in his account to carry out the transaction. After the transaction is carried out the resulting amount is displayed and the details are updated to the database.

RESULT:

Thus the system for ATM is created and executed. The output is verified.

CHAPTER 4. BEYOND THE SYLLABUS

A. JDBC CONNECTIVITY

JDBC stands for Java Database Connectivity, which is a standard Java API for database independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks commonly associated with database usage:

- Making a connection to a database
- Creating SQL or MySQL statements
- Executing that SQL or MySQL queries in the database
- Viewing & Modifying the resulting records

Pre-Requisite:

You need to have good understanding on the following two subjects to learn JDBC:

- Core JAVA Programming
- SQL or MySQL Database

JDBC - Environment Setup

Make sure you have done following setup:

1. Core JAVA Installation
2. SQL or MySQL Database Installation

Apart from the above you need to setup a database which you would use for your project. Assuming this is EMP and you have created on table Employees within the same database.

B. CREATING JDBC APPLICATION

There are six steps involved in building a JDBC application

```
//STEP 1. Import required packages
        import java.sql.*;

//STEP 2: Register JDBC driver
        Class.forName("com.mysql.jdbc.Driver");

//STEP 3: Open a connection & // Database credentials
        static final String USER = "username";
        static final String PASS = "password";
        System.out.println("Connecting to database...");
        conn = DriverManager.getConnection(DB_URL,USER,PASS);

//STEP 4: Execute a query [This requires using an object of type Statement or
        PreparedStatement for building and submitting an SQL statement to the
        database as follows:]
        System.out.println("Creating statement...");
        stmt = conn.createStatement();
        String sql;
        sql = "SELECT id, first, last, age FROM Employees";
        ResultSet rs = stmt.executeQuery(sql);

//STEP 4: Execute a query [If there is an SQL UPDATE,INSERT or DELETE statement
        required, then following code snippet would be required ]
        System.out.println("Creating statement...");
        stmt = conn.createStatement();
        String sql;
        sql = "DELETE FROM Employees";
        ResultSet rs = stmt.executeUpdate(sql);

//STEP 5: Extract data from result set
        while(rs.next()){ //Retrieve
        by column name int id =
        rs.getInt("id");
        int age = rs.getInt("age");
        String first = rs.getString("first");
        String last = rs.getString("last");
        //Display values System.out.print("ID:
        " + id); System.out.print(", Age: " +
        age); System.out.print(", First: " +
        first); System.out.println(", Last: " +
        last);}

//STEP 6: Clean-up environment
        rs.close();
        stmt.close();
        conn.close();
```

C. JDBC PROGRAM

Based on the above steps, we can have following consolidated sample code which we can use as a template while writing our JDBC code

```
//STEP 1. Import required packages
                    import java.sql.*;
                    public class FirstExample { //
          JDBC driver name and database URL
                    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
                     static final String DB_URL = "jdbc:mysql://localhost/EMP";
           // Database credentials
                    static final String USER = "username";
                    static final String PASS = "password";
                    public static void main(String[] args) {
                    Connection conn = null;
                    Statement stmt = null;
                     try{
 //STEP 2: Register JDBC driver
                    Class.forName("com.mysql.jdbc.Driver");
 //STEP 3: Open a connection
                     System.out.println("Connecting to database...");
                     conn = DriverManager.getConnection(DB_URL,USER,PASS);
 //STEP 4: Execute a query
                     System.out.println("Creating statement...");
                     stmt = conn.createStatement();
                    String sql;
                     sql = "SELECT id, first, last, age FROM Employees";
                     ResultSet rs = stmt.executeQuery(sql);
 //STEP 5: Extract data from result set
         while(rs.next()){ //Retrieve by
                           column name
                    int id = rs.getInt("id"); int
                     age = rs.getInt("age");
                     String first = rs.getString("first");
                     String last = rs.getString("last");
           //Display values
                      System.out.print("ID: " + id);
                    System.out.print(", Age: " + age);
                    System.out.print(", First: " + first);
                    System.out.println(", Last: " + last);
                    }
 //STEP 6: Clean-up environment
                     rs.close();
                     stmt.close();
                     conn.close();
                      }catch(SQLException se){
           //Handle errors for JDBC
                     se.printStackTrace();
                      }catch(Exception e){
           //Handle errors for Class.forName
                    e.printStackTrace();
                    }finally{
           //finally block used to close resources
                    try{
                     if(stmt!=null)
                    stmt.close();
                     }catch(SQLException se2){
           }// nothing we can do
                     try{
                     if(conn!=null)
                     conn.close();
                     }catch(SQLException se){
                     se.printStackTrace();
           }//end finally try
           }//end try
                     System.out.println("Goodbye!");
}//end main
}//end FirstExample
```