External Agent

(a) Use noun phrases to label external agents.

(b) External agents cannot interact directly with a store or any other external agent through data flows. Data must be moved through a process as shown in Fig. 5.26 and 5.27.

Data Store

(a) Use meaningful noun phrases to label data stores.

(b) Data cannot move directly from data stores to other data stores or external agents. It must move through a process.

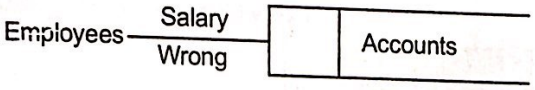


Fig. 5.26

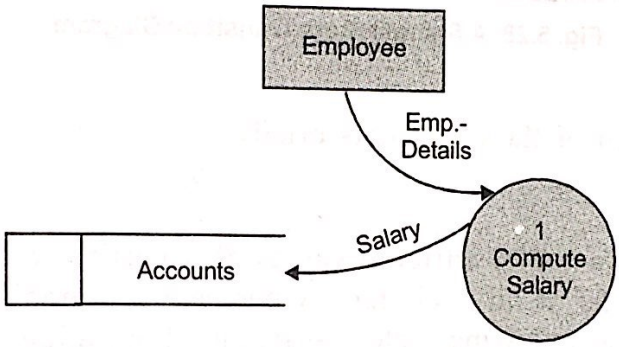


Fig. 5.27

5.3.4 State Transition Diagram (STD)

State Transition Diagram (STD) models the dynamic view *i.e.*, time dependent behavior of the system. In the past it was used only for special category of systems *i.e.*, real time systems, process control systems, space shuttle programs etc. But in today's scenario this diagram is also used for modeling some part of business systems *e.g.*, user interface. Major components of the diagram are:

- State
- Action
- Arrow
- Condition.

A sample STD for an ATM machine would be as shown in Fig. 5.28.

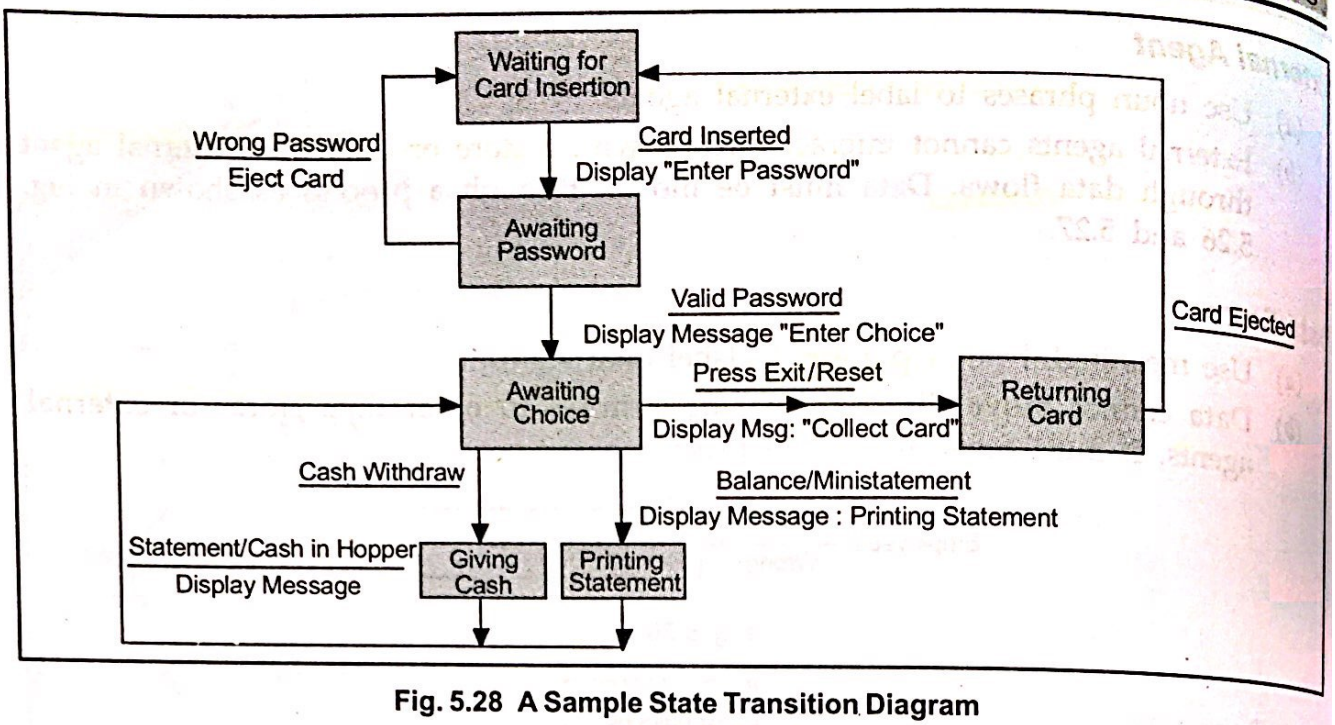


Fig. 5.28 A Sample State Transition Diagram

Next we discuss each of these concepts briefly.

State

A state is described by a set of attribute values at a particular instant of time and is represented by a rectangle or an oval sign. A person for example depending upon his age can be in one of the following states — infant, child, adult, middle-age man, old man.

Fig. 5.29 Symbols for State

State can be either initial/start state, end/final state or in between state. In general a system can have only one initial state and single or multiple final states.

Action

Whenever system changes states in response to a condition, it performs one or more actions. It is also possible that it may not perform any action. Some of the actions can be displaying a message, running the electric motor, generating some output etc.

Arrows

Arrows connect two or more states indicating that state S1 changes to state S2 as a result of some condition C being satisfied.

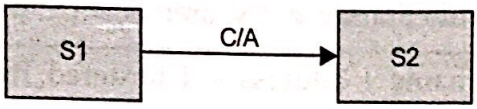


Fig. 5.30

Condition

A condition is some event in the external environment which causes the system to change from say state S1 to S2. This event can be anything, say an interrupt, signal or arrival of some data.

In the diagram shown in Fig. 5.31 *e.g.*, man changes state from child to adult when he attains age of 18 years.

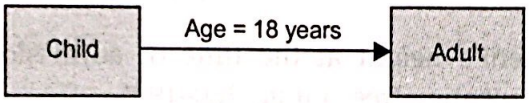


Fig. 5.31

5.3.5 Data Dictionary

Data dictionary is the most important part of structured analysis model. As far as definition is concerned, it is the organized listing of all data elements of the system with their precise and unambiguous definitions. Data dictionary contains information about

- Meaning of aggregate item with comments
- Units of elementary items
- Definition of data/control flows
- Definition of data stores
- Definition of entities, relationship, attributes, external agents
- Definition of external control/data flows
- Local data elements used in writing process specifications and many more.

A number of schemes are proposed to represent the details in data dictionary. One such popular scheme is given in Fig 5.32.

=	is defined as/composed of
+	and
{ }	iteration (0 or more occurrences)
min{ }max	iteration with min & max values specified
( )	optional data elements
[ ]	selection of one data from several choices separated by |
@	Store identifier
**	Comment.

Fig. 5.32 Notation for Writing Data Dictionary

6.3.1 Use Case Diagram (UCD)

A Use Case Diagram describes the system from the users point of view. In other words it shows the relationship between the system and the external world. This diagram is also used to show the boundary of the system. Use Case diagrams are used in the initial stages of software development i.e., requirements analysis stage and are very useful for

- Understanding the requirements of the problem domain
- Generating test cases
- Contributing to users understanding of the system
- Validating design
- Creating project schedule

There are six simple concepts which are used to draw the Use Case Diagram. They are:

- System
- Actor
- Use Case
- Associations
- Dependencies
- Generalization

In simple language a ***Use Case Diagram is a collection of Actors, Use Cases and their Communications.*** Next, we briefly describe each of these concepts used for drawing UCD.

System

A system icon shows the boundary of the system to be built and is represented by a rectangle as shown in Fig. 6.9.

Fig. 6.9 System Icon

Actor

Actor is a role played by a person, organization, other system or device which interacts with the system by using the use cases. In other words, an actor is one who initiates some events which in turn trigger the activities of the organization. Fig. 6.10 shows some actor icons used in UML.

It is to be kept in mind that actors are always outside the boundary of the system. So as said earlier, actors can be people, an organization, some other software, hardware, networks etc. Each of these can be represented by one or more actors. For example a person working in the bank can have an account in the same bank. Therefore in this case he plays the role of two actors *i.e.*, employee and the customer.

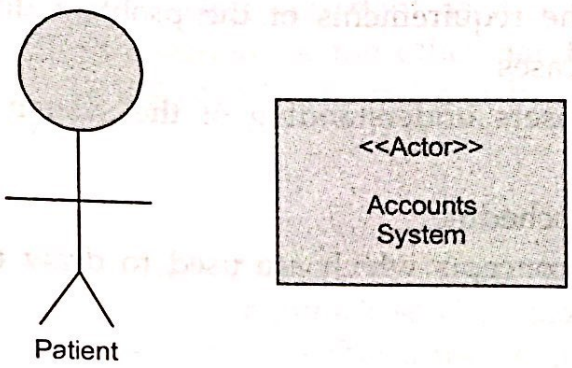


Fig. 6.10 Symbols for Actor

Use Cases

A Use Case shows the required functionality or goals of the system without emphasizing how it will be achieved. In other words, it defines the requirements of the system. In simple language, Use Cases are those features of the software which actors would like the system to support e.g., querying the admission status, printing the inventory report. If Use Case diagram is too complicated and big, it is better to partition the Use Case diagram in to several Use Case diagrams, where each diagram represents some major functionality of the problem domain. Use Cases are written using verb-object phrases e.g., withdraw cash, show balance etc. Notation for representing Use Cases is shown in Fig. 6.11.

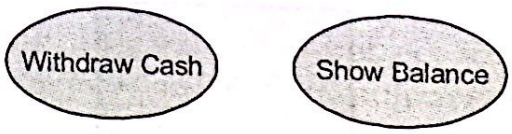


Fig. 6.11 Use Case Notation

Associations

Associations show the communication of actors with Use Cases and are represented by straight lines connecting actors and Use Cases. Associations can be unidirectional or bidirectional. Unidirectional associations can be shown by putting arrow on one side where as bidirectional association are represented by putting arrows on both sides or no arrows at all. In case of unidirectional association there is only one way communication where as in the bidirectional association the communication is both way (e.g., a Use Case supporting some kind of query).

Dependencies

Dependencies are used to show relationship between Use Cases and is shown through dashed arrows between two Use Cases. There can be two types of relationships between Use Cases **<<include>>** and **<<extend>>**. Stereotype notation *i.e.*, word in guillemets << >> is used to show relationship. The main advantage is that through this feature UML supports reuse in the software being developed. <<include>> dependency stereotype says that execution of one Use Case includes the working of other Use Case always whereas <<extend>> stereotype says that one Use Case may or may not use the other Use Case. Fig. 6.12 shows the associations and dependencies in UCD.

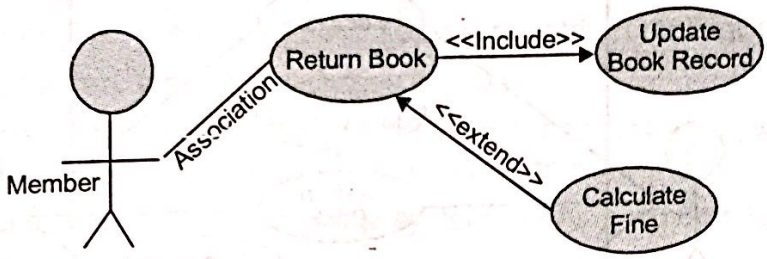


Fig. 6.12 Dependency and Association Notation

It is to be noted that direction of arrow in <<extend>> dependency is from the extension Use Case to the one which is calling it. On the other hand in the <<include>> dependency the arrow direction is from the main Use Case to the one which is being required by it. In Use Case diagram shown in Fig. 6.12, Return-book Use Case will always call update book_record Use Case but it may or may not call calculate-fine Use Case.

Generalization

Finally is_a relationship *i.e.*, concept of inheritance among Use Cases and actor is shown through generalization. A solid line with hollow triangle is used to show generalization. A sample Use Case diagram to show generalization is shown in Fig. 6.13.

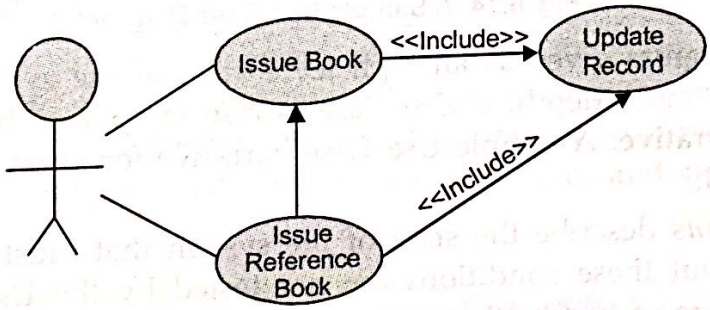


Fig. 6.13 Generalization Among Use Cases

Guidelines For Drawing Use Case Diagrams

The guidelines used for drawing Use Case diagram are:

1. Set the boundary of the system.

2. Identify actors.
3. Identify Use Cases.
4. Establish associations between actors and Use Cases.
5. Identify <<include>> and <<extend>> relationships between the Use Cases.
6. Identify generalization between actors and Use Cases.
7. Refine the diagram by reviewing it more than once.

A sample Use Case Diagram for a part of Library is shown in Fig. 6.14.

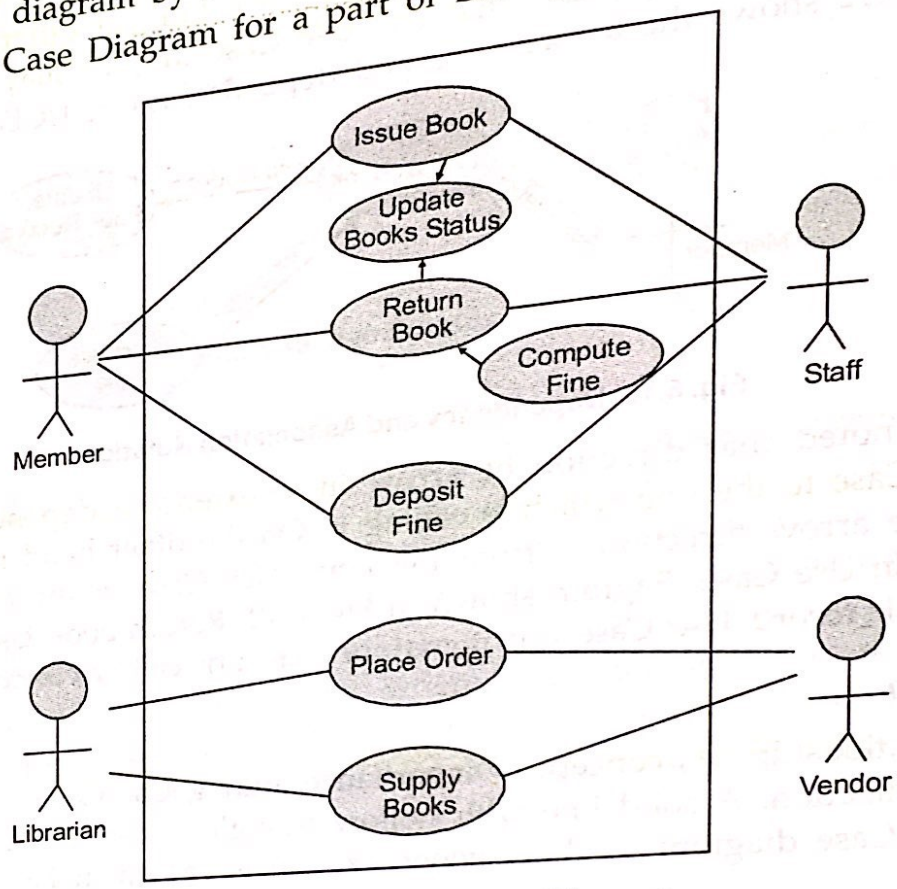


Fig. 6.14 A Sample Use Case Diagram

A Use Case Diagram gives us an high level view of the system. Therefore in order to understand it in more depth, a Use Case is also supported by a textual description called **Use Case narrative**. A sample Use Case narrative for Use Case Register Complaint is shown in Fig. 6.15.

Here ***Assumptions*** describe the state of the system that must be true before we can use the Use Case but these conditions are not tested by the Use Case. ***Pre-conditions*** also describe the state of system that must be true before we use the Use Case but these conditions are tested by the Use Case before proceeding further. For example in the Use Case narrative shown in Fig. 6.15 unless a proper patient code in proper format is entered, Use Case will not start. Similarly ***post-conditions*** describe the state of the system that must be true when the Use Case ends. Though there is only one way to start the Use Case, there are many ways to end it. These are documented in Use Case termination part of the Use Case narrative.

Name — Register Complaint
Author — S. Sabharwal
Number — 22
Last update — 14/01/2004
Assumption — User has permission to use this feature
Preconditions — Provide a valid patient registration number

Flow of Events/Dialog:

1. Use Case begins when the patient selects register complaint option.
2. The patient enters his/her registration number.
3. The system verifies the registration number. If incorrect the system will prompt the patient to reenter the registration number or exit.
4. If registration number is correct, the system opens a window for the patient to record his/her complaint.
5. After writing, the patient exits by pressing the exit button.

Post conditions: In case of normal termination complaint is saved in complaints database for management use.

Use Case termination:
1. The user presses exit after he is done.
2. The Use Case may time out.
3. The user may exit anytime during the execution of use case by pressing the Cancel Key.

Fig. 6.15 The Register Complaint Use Case Description

6.3.2 Classes and Class Diagram

Class diagram is the most important diagram which shows the static view of the system by showing the classes and their relationships. A class integrates the statics (data) and dynamics (behavior) into one cohesive unit. UML notation to represent a **class** is a rectangle consisting of three parts - class name, class attributes and operations as shown in Fig. 6.16.

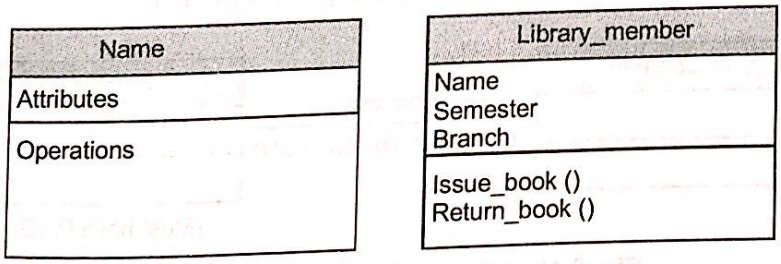


Fig. 6.16 Class Icon

Additionally, it also specifies the visibility of attributes and operations *i.e.*, which other objects can see the attributes and operations. This typically includes public (+), private (–), protected (#) and package (~) as visibility values.

While writing operation specification, it must specify all arguments and their data types (separated by colon), return data type, its visibility and the constraints within { }. For example Compute_Cost operation is written as

+ Compute_cost(Order: Order): Rupees {The total cost is sum of all items cost (unit price * quantity) mentioned in the order less the trade discount}

A detailed class specification for patient class is as shown in Fig. 6.17.

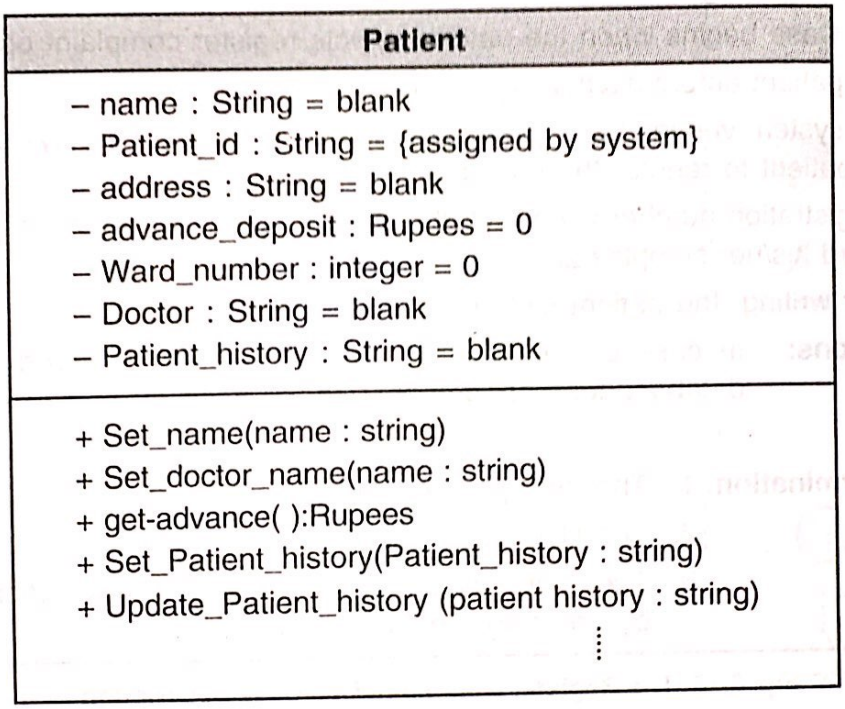


Fig. 6.17 Detailed Class Specification

Three types of relations *viz.*, **association, aggregation** and **generalization** can exist between classes in the class diagram. **Association** relates a class *A* to class *B* and is shown as a link between classes. While reading textual description of requirements they correspond to verbs and classes correspond to nouns.

Multiplicity (number of participating objects), role and constraints are also shown in the association. Constraints are shown between { } below the class icons.

An association between two classes is shown in Fig. 6.18.

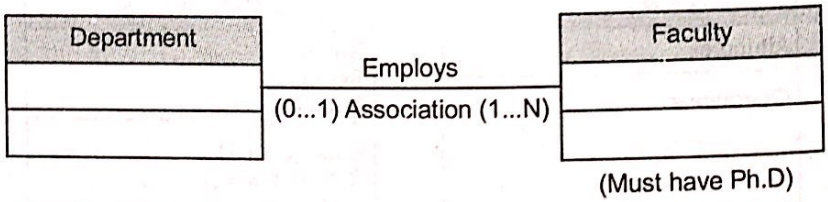


Fig. 6.18 Association Between Classes

Is_a relationship between classes is shown through **generalization.** The classes between which is_a relationship is identified are called superclass and subclass. Subclass inherits the features of superclass in addition to features which are specific to subclass. A generalization is represented as shown in Fig. 6.19. Here Department is the superclass and classes Computer Engineering and Mechanical Engineering are sub classes.

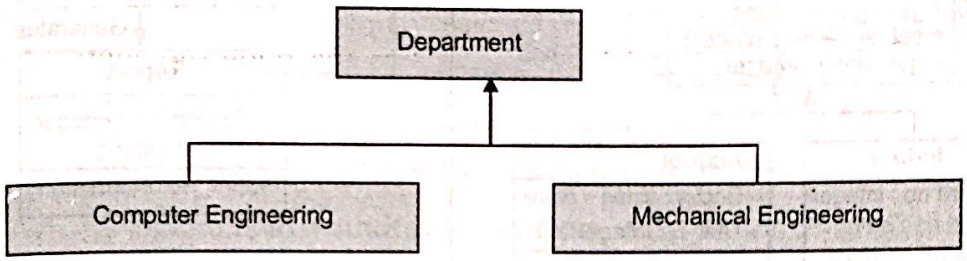


Fig. 6.19 Generalization in Classes

Aggregation is a special type of association which supports building complex objects out of existing objects. For example different components can be assembled to make a car as shown in Fig. 6.20.

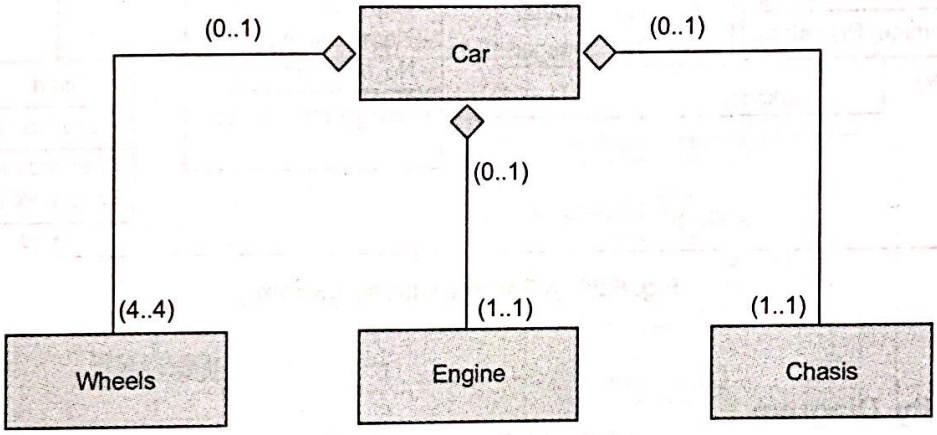


Fig. 6.20 Aggregation Relationship

To represent aggregation a diamond symbol is attached to the aggregate class. A sample class diagram is shown in Fig. 6.21.

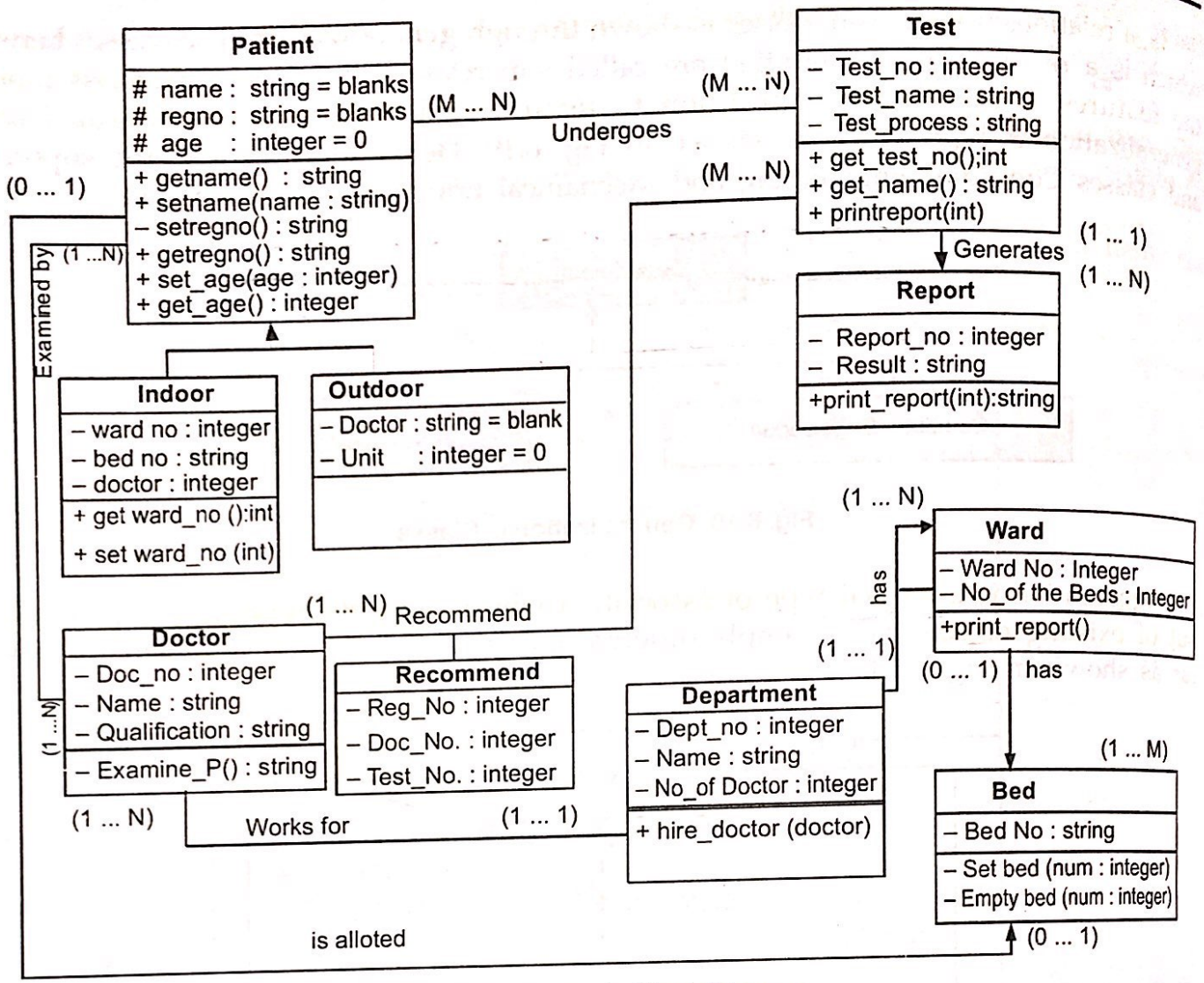


Fig. 6.21 A Sample Class Diagram

6.3.3 Activity Diagram

Activity diagram is used to model the functional view and focuses on flow of activities for a process. It resembles a flow chart. In simple language, an activity diagram is a series of interconnected activities which are linked by transitions.

Each of the activities in the activity diagram is a step (computation of data, query, verification of data, report generation etc.) in the process. Transition lakes place when activity is over and/or the guard condition is satisfied. From decision point various paths exit for each condition. Similarly at a merge point two or more paths meet and continue as one. Each activity diagram has a starting point and single/multiple ending points. Activity diagrams can also be divided into object *swimlanes* to indicate which object is responsible for which activity. Transitions can also fork into multiple parallel activities.

Notation used to draw activity diagram is given in Fig. 6.22.

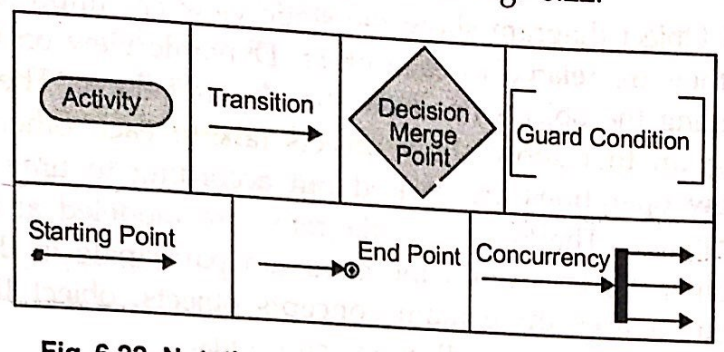


Fig. 6.22 Notation for Drawing Activity Diagram

A sample Activity diagram to withdraw cash from ATM is shown in Fig. 6.23

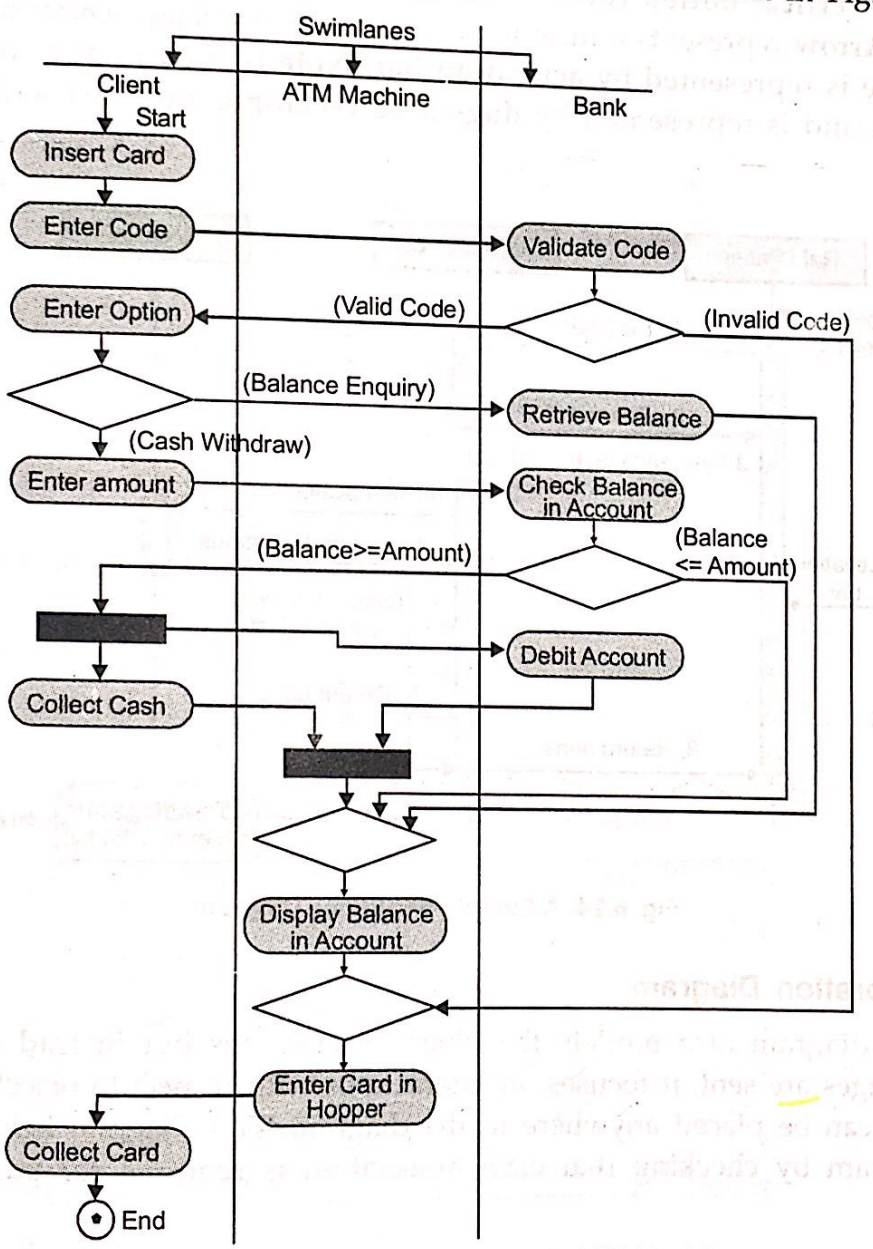


Fig. 6.23 An Activity Diagram for an ATM Machine

6.3.4 The Sequence Diagram

Class diagram and Object diagram show the static view *i.e.*, important objects of problem domain and how they are related to each other. Dynamic view on the other hand shows the interactions among the objects i.e., how the objects behave. The Sequence diagram is an interaction diagram that shows how objects talk to each other. In other words the diagram shows how operations are carried out according to time and how objects are created and manipulated. The Sequence diagrams are modeled at objects level to allow for scenarios. Multiple objects from same class can participate in the Sequence diagram. The Sequence diagram uses three main concepts objects, object lifeline and messages. Messages can be synchronous as well as synchronous.

Fig. 6.24 shows a sample Sequence diagram to create bill at a diagnostics lab. In this diagram each **vertical dotted line** is **lifeline** which represents the time for which an object exists. **Arrow** represents a message call to the other object. Duration for which an object is active is represented by **activation bar. Note** is used to add comments to the UML diagram and is represented by **dog-eared rectangle** with text written inside the rectangle.

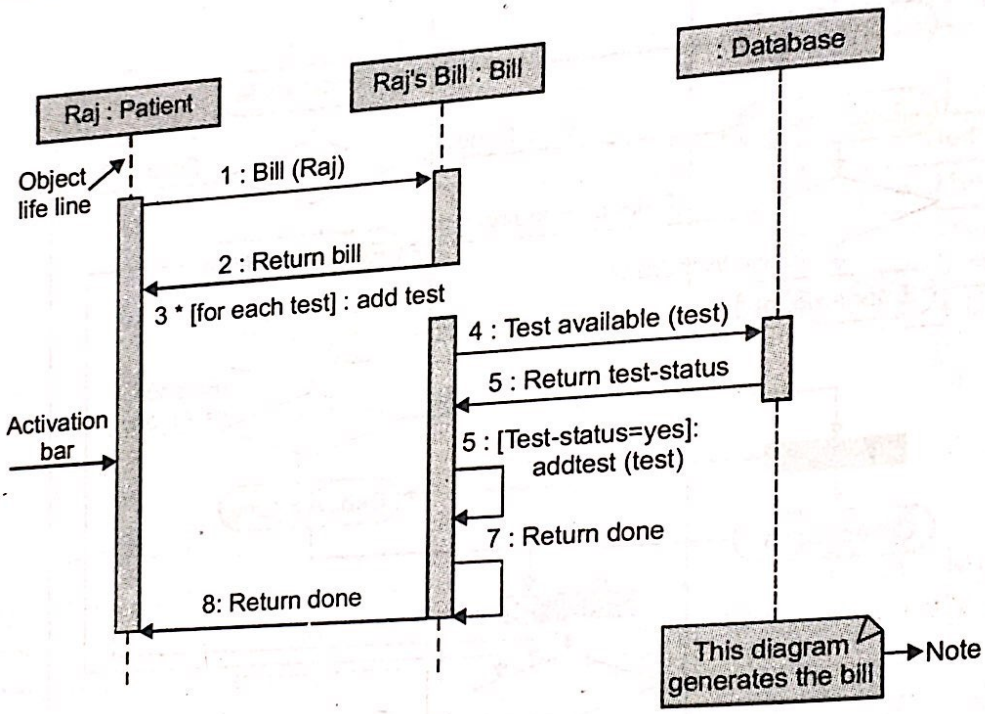


Fig. 6.24 A Sample Sequence Diagram

6.3.5 Collaboration Diagram

Collaboration diagram also models the object interactions but instead of focusing on time the messages are sent, it focuses on interactions with respect to objects relationships. Hence objects can be placed anywhere in the diagram. This diagram helps in validation of Class diagram by checking that each association is required for passing messages

among the classes. This helps in identification of new operations and thus refines the Class diagram. A sample Collaboration diagram is shown in Fig. 6.25.

As shown in Fig. 6.25, object roles rectangles can be labeled with object names or class or both. Further all the messages are numbered to show the order of execution of messages. The format for specifying the message is

message_sequence no. : [condition] operation or return.

Same format as clear from Fig. 6.24 is also used in Sequence diagram.

Collaboration diagram does not give any information about when object is active or inactive which is clear in Sequence diagram.

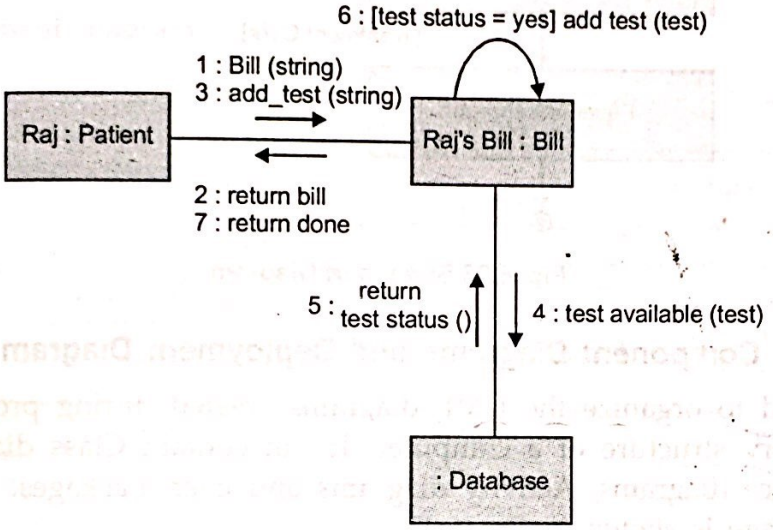


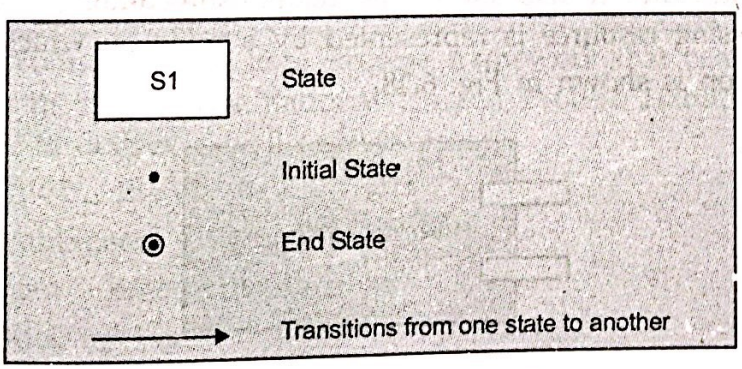
Fig. 6.25 A Sample Collaboration Diagram

6.3.6 Statechart Diagram

Statechart diagram also models the dynamic view. It shows all the possible states in an object's lifetime and events that trigger the change in states.

It is similar to state transition diagram discussed in chapter 5. The notation used to draw Statechart diagram is given in Table 6.1

Table. 6.1 Notation for Statechart Diagram



Events that cause change of state and corresponding actions are labeled on arrows. A sample Statechart diagram is shown in Fig. 6.26.

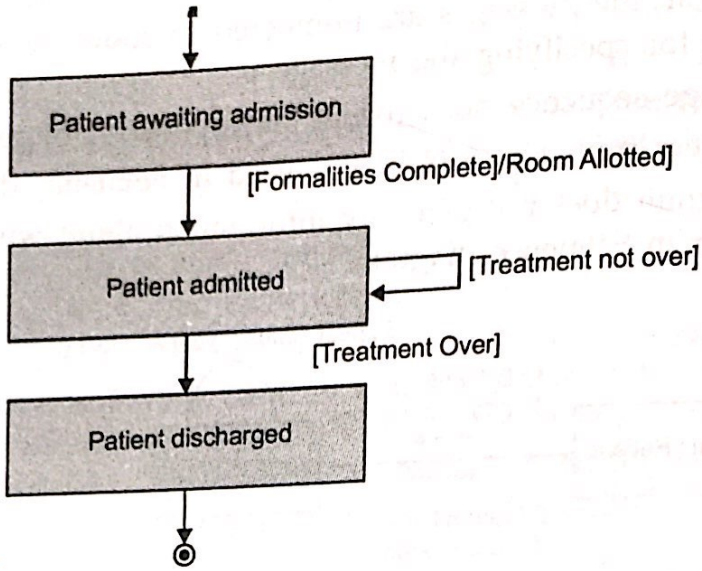


Fig. 6.26 Statechart Diagram

6.3.7 Packages, Component Diagrams and Deployment Diagram

Packages are used to organize the UML diagrams created during project logically and work like directory structure of a computer. It can contain Class diagrams, Use Case diagrams, Sequence diagrams, Activity diagrams and even Packages. The symbol used to represent Package is shown in Fig. 6.27.

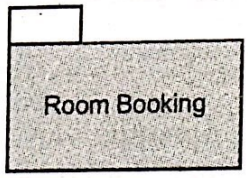


Fig. 6.27 A Packaging Icon

Physical Implementation of the software is shown using Component diagram where components represents software modules. Physical architecture of the hardware is modeled with Deployment diagram. It also shows the relationship among various software and hardware. A processing resource is represented by a node on which software resides.

A component icon is shown in Fig. 6.28.

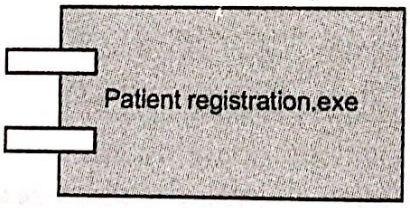


Fig. 6.28 A Component Icon

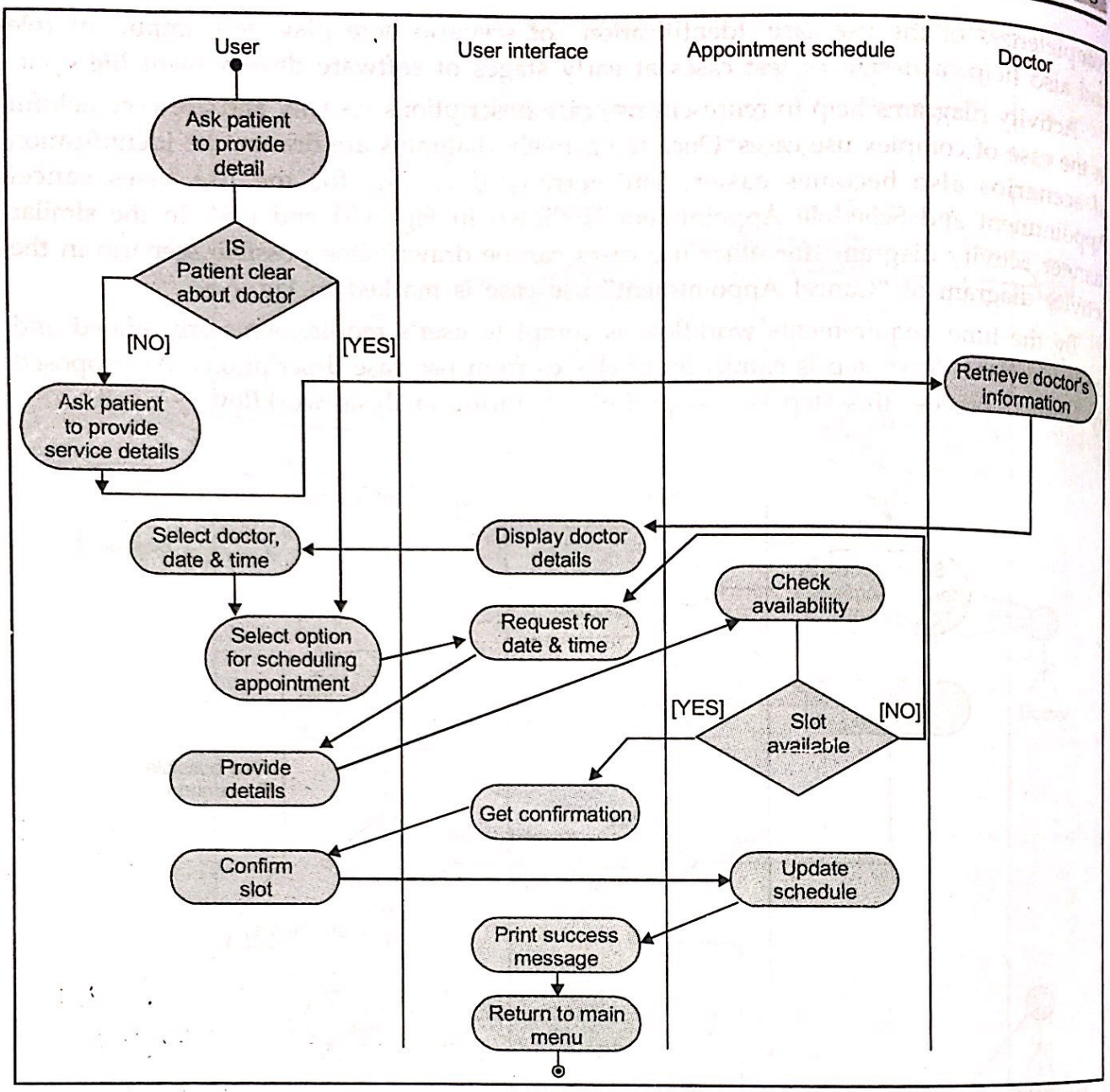


Fig. 6.54 Activity Diagram for Schedule Appointment

Extracting Classes

During analysis workflow, requirements identified in requirements workflow are used as input and output is a set of classes and subsystem that can be directly mapped to software. It is to be noted that at this stage, classes are represented at very high level of abstraction, containing only high level attributes and relationships between the classes. A class can be of three types

1. Entity class
2. Boundary class
3. Control class.

Entity Class: Entity classes are used to model real world objects that are long lived. In this problem patient class and doctor class are examples of entity class because information about patients and doctors has to be stored in the information system.

Boundary Class: Boundary classes are used to model the interaction between the users of the system i.e., actors and the information system itself. These classes are used to support input and output in the system. Examples are user interface class, classes to support interfaces with devices like printer, window etc. Hence in the example of clinic system we can have boundary classes to take input and to print different reports.

Control Class: Control classes are used to model the dynamics of the system and are used to model complex computations, sequence of events, transactions and controlling other classes. In the clinic system we can identify a control class "Compute Payment" to calculate the payment to be made to the doctor.

The notations for these three types of classes is given in Fig. 6.55

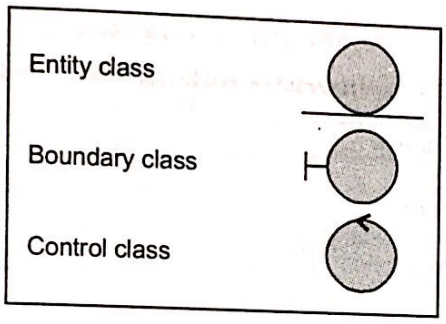


Fig. 6.55

The Procedure for extracting initial classes from Use cases is as following:

1. Read the Use case descriptions. Noun's are normally taken as entity classes. Identity the entity classes and establish relationship among them.
2. Draw the dynamic modal (state chant diagram, interaction diagrams) for each class or system as a whole to identity the operations that can be performed to the system or individual class
3. Model each input/output screen, printed report as a boundary class.
4. Identify the major computations to be done and represent them as control classes.

Now let us go through the Use case descriptions and identify different types of classes.

Identification of Entity Classes: After going through the Use case descriptions following entity classes are identified:

- Patient
- Doctor
- Service
- Visit
- Appointment
- Payment

The initial set of classes and their relationships are shown in Fig. 6.56. In this diagram the attributes are also mentioned.

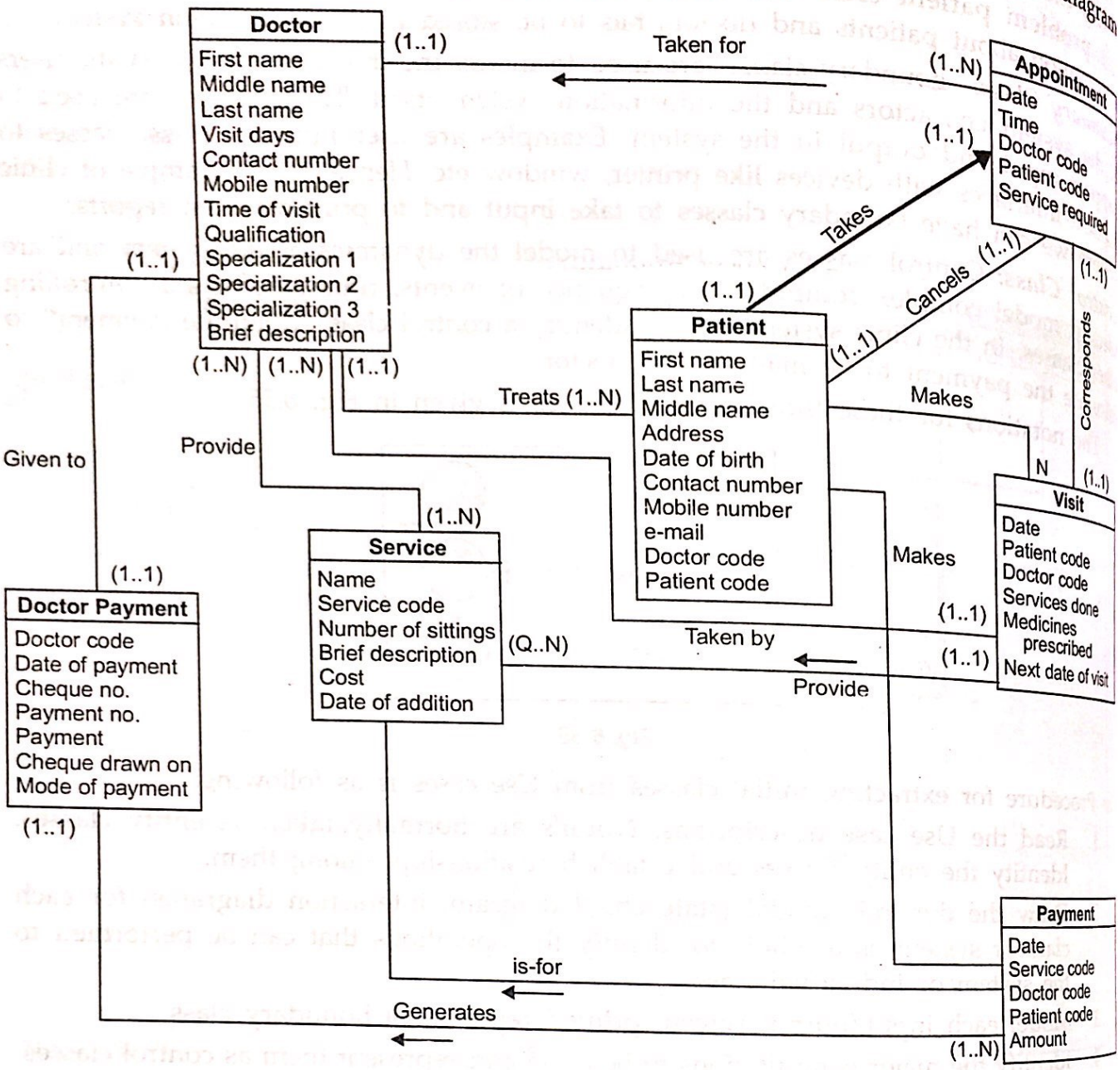


Fig. 6.56 Class Diagram

Identification of Boundary Classes – As boundary classes support interaction between users and the information system following can be modeled as boundary classes

- User Interface class for input/output screen
- Date outpolling class

In this case he can broadly think of two screens – one for using all Use cases i.e. different features of system and other for generating reports. Hence our user interface screen will some what look like as shown in Fig 6.57.

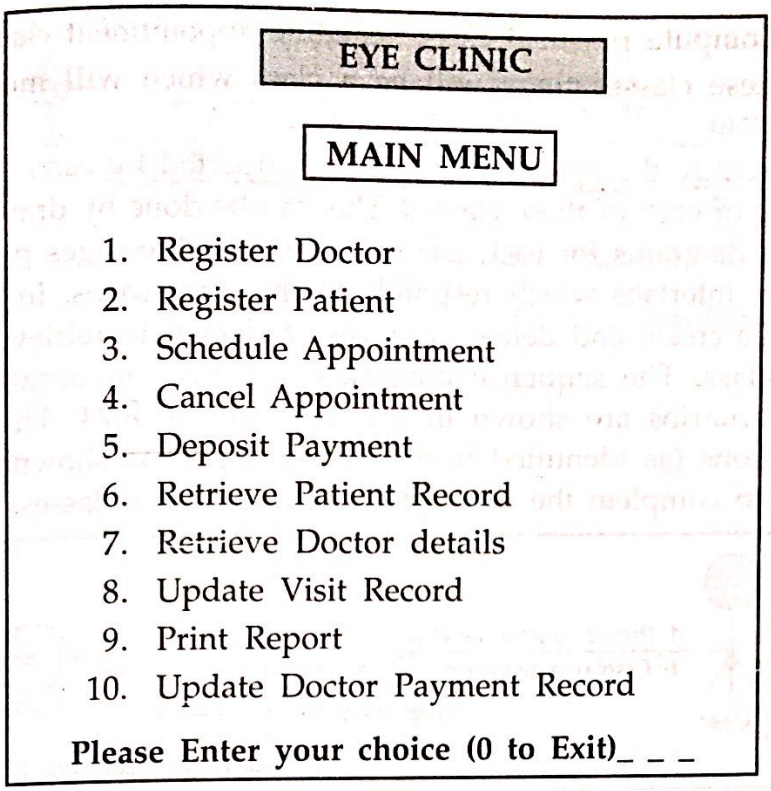


Fig. 6.57

Second Screen is for generating different Reports and is called Report generation interface. The initial version of Screen will be as shown in Fig. 6.58.

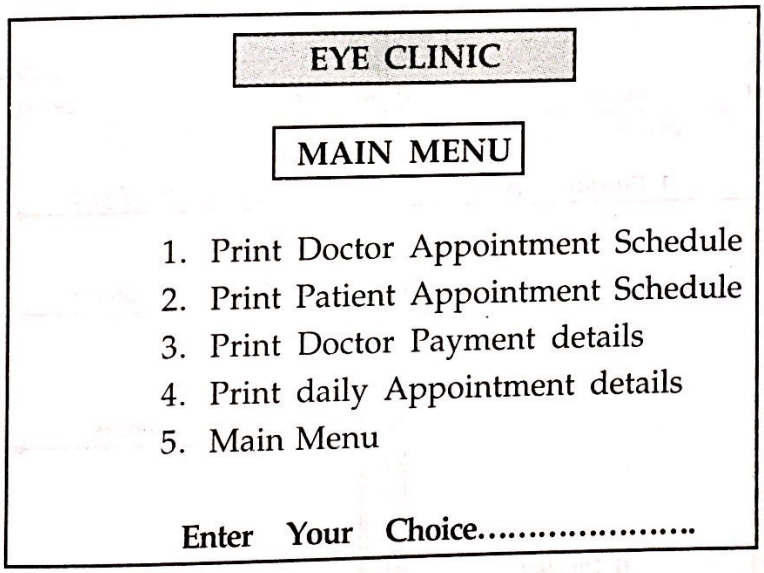


Fig. 6.58

Hence we see that the following classes are finally identified.

Entity Classes – Doctor, Appointment, Patient, Visit, Service, Payment, Doctor_payment

Boundary Class – User Interface Class, Report interface class, Print Patient appointment report class, Print Doctor appointment report class, Print payment report class.

Control class – Compute_payment class, Schedule_appointment class.

In addition to these classes there will be a class which will monitor the overall working of the program.

Next step is to identify the operations that are supported by each of these classes or to define the interface of each of these classes. This can be done by drawing the sequence and the collaboration diagrams for each use case scenario. Messages passed to an object are part of the object interface which responds to these messages. In addition to these there are operations to create and delete instances of classes, to retrieve, set and modify the attributes of a class. The sequence diagrams and the collaboration diagrams for different use case scenarios are shown in Fig. 6.59 to Fig. 6.74. The visit class with attributes and operations (as identified from dynamic view) is shown in Fig. 6.76. The readers are advised to complete the description of rest of the classes.

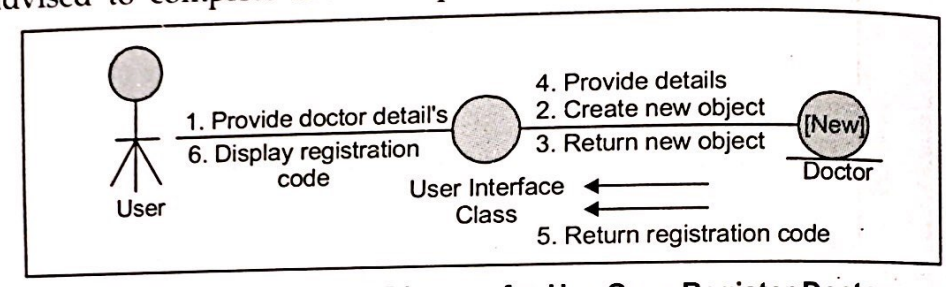


Fig. 6.59 Collaboration Diagram for Use Case Register Doctor

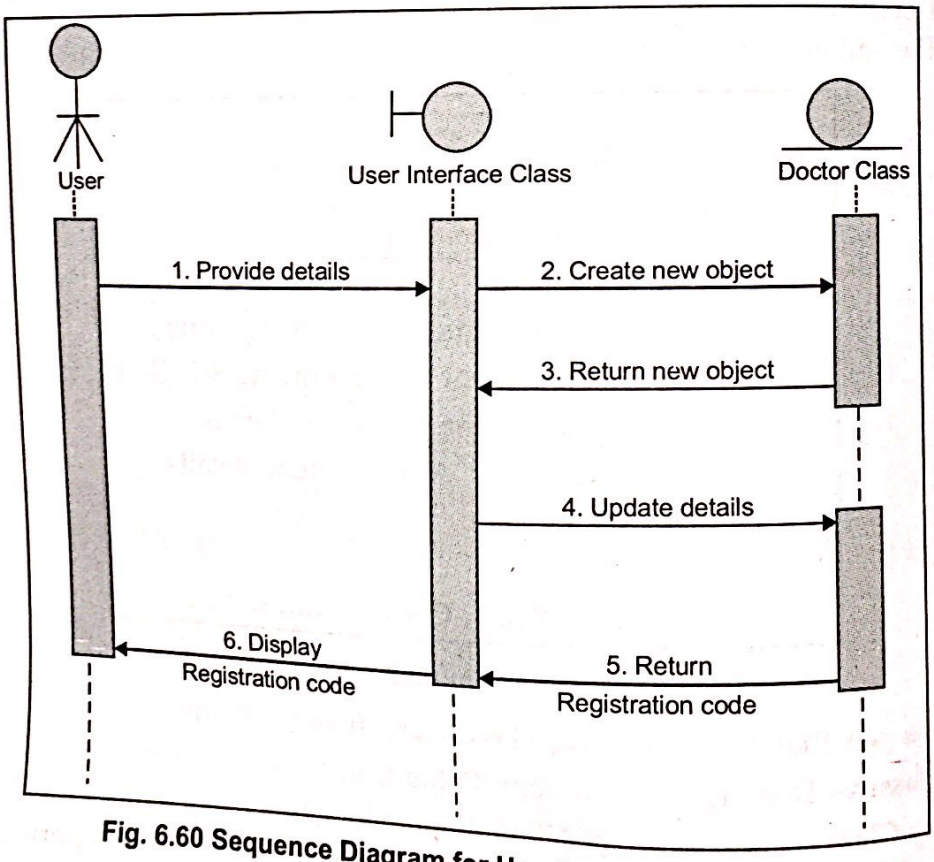


Fig. 6.60 Sequence Diagram for Use Case Register Doctor

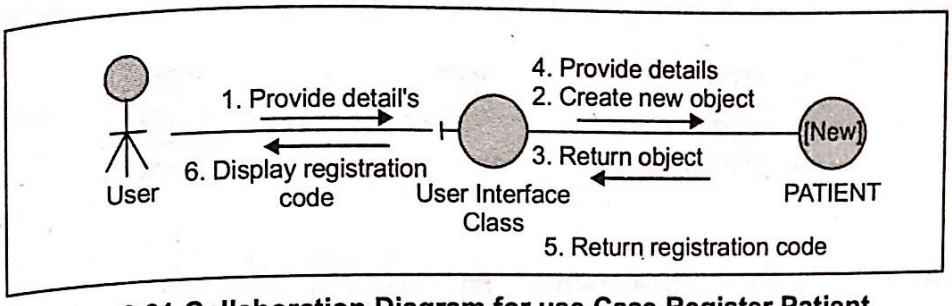


Fig. 6.61 Collaboration Diagram for use Case Register Patient

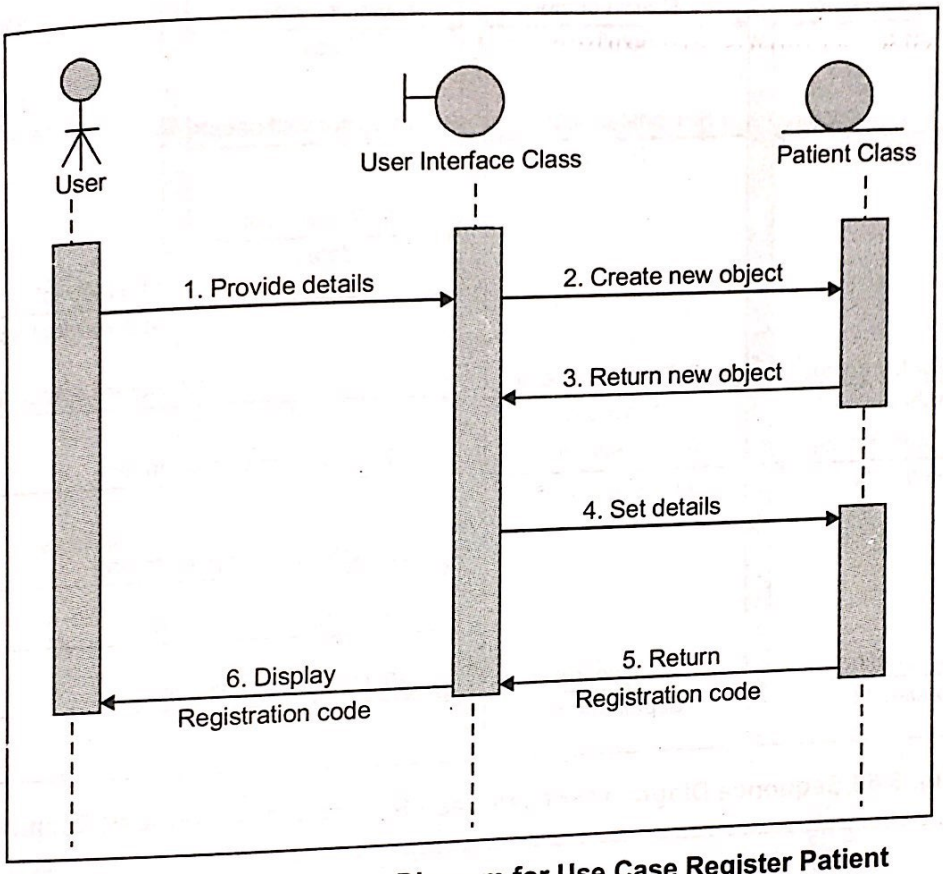


Fig. 6.62 Collaboration Diagram for Use Case Register Patient

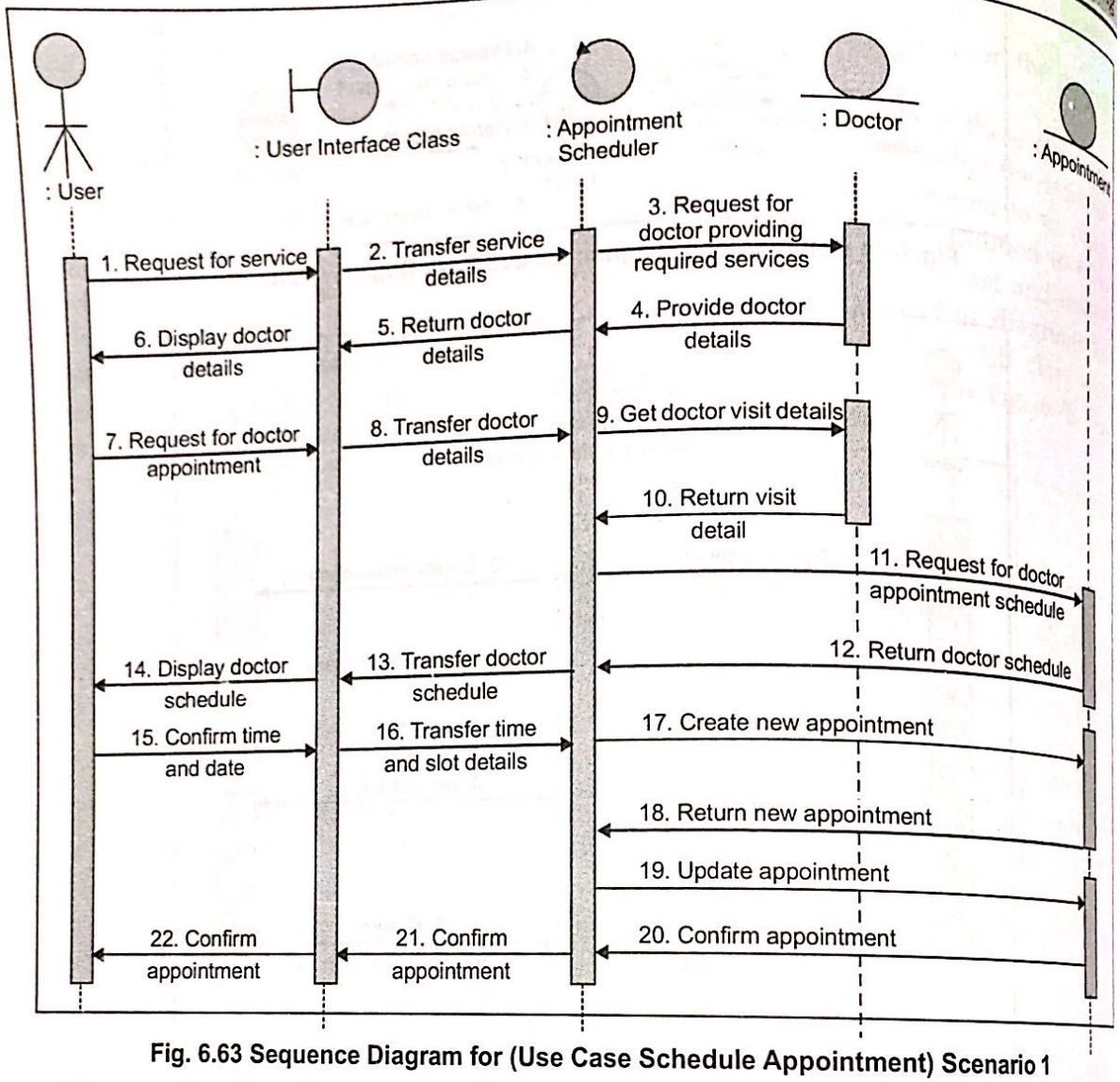


Fig. 6.63 Sequence Diagram for (Use Case Schedule Appointment) Scenario 1

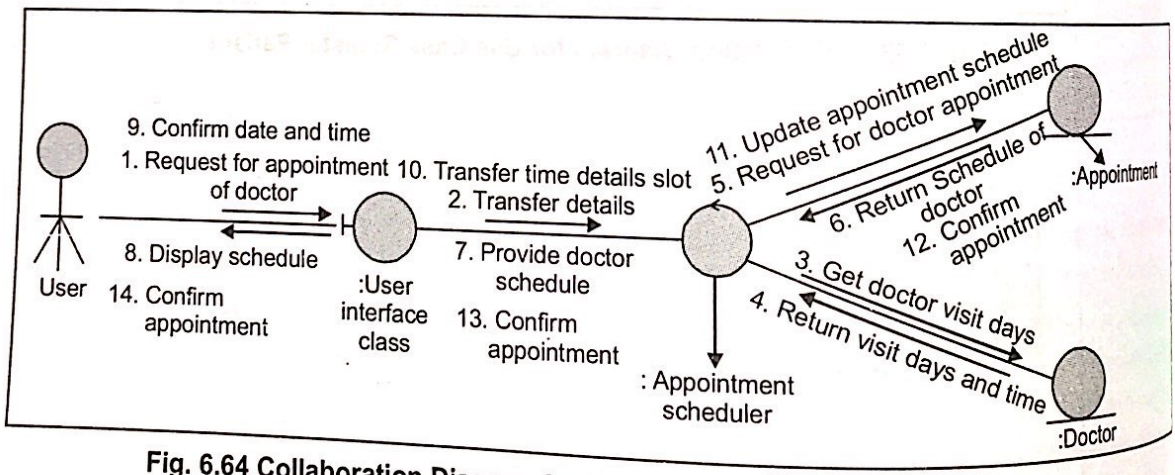


Fig. 6.64 Collaboration Diagram for Use Case Schedule Appointment Scenario-2

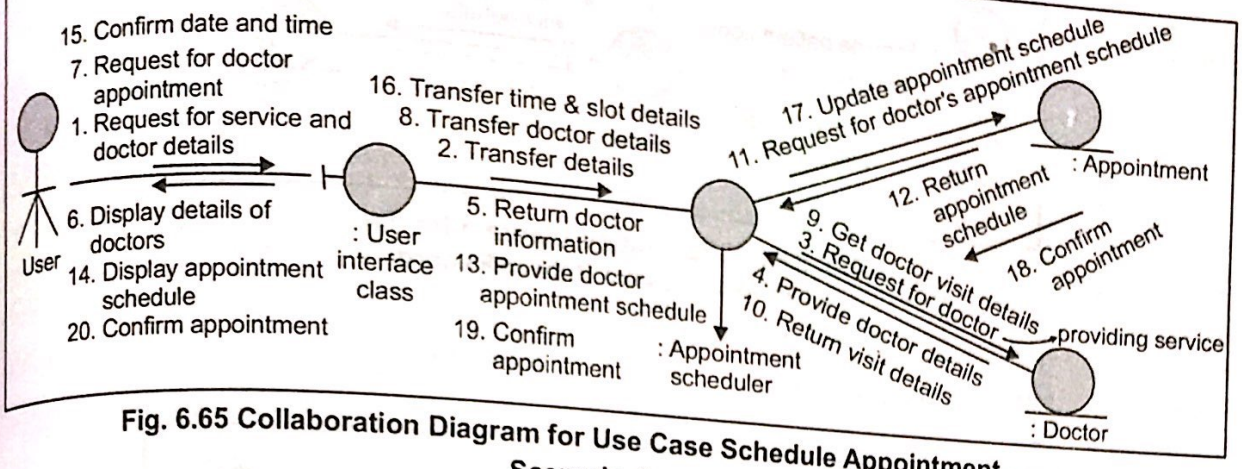


Fig. 6.65 Collaboration Diagram for Use Case Schedule Appointment Scenario-1

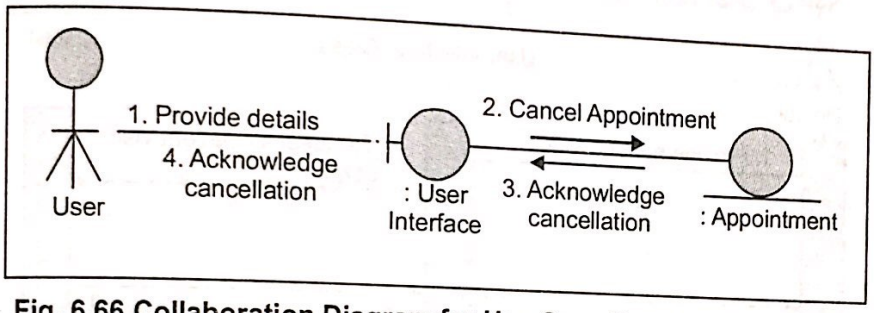


Fig. 6.66 Collaboration Diagram for Use Case Cancel Appointment

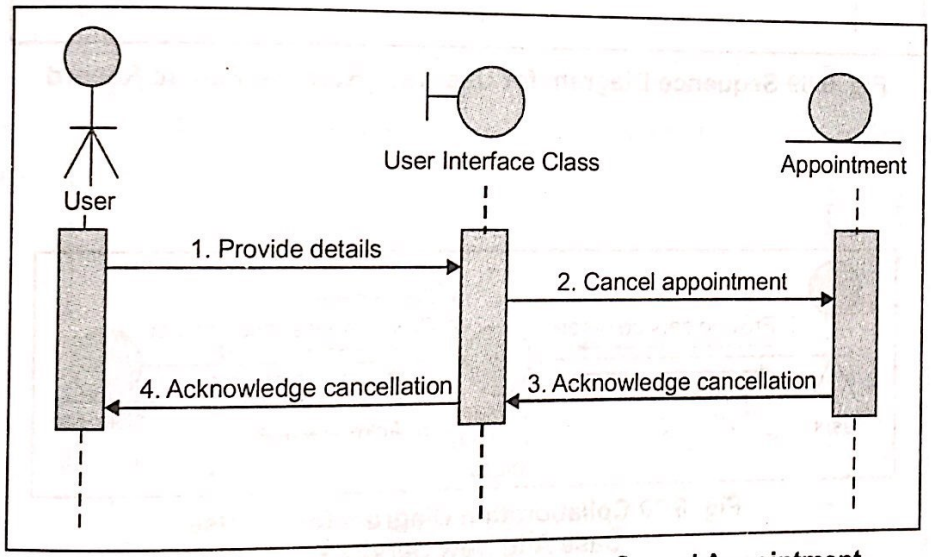


Fig. 6.67 Sequence Diagram for Use Case Cancel Appointment

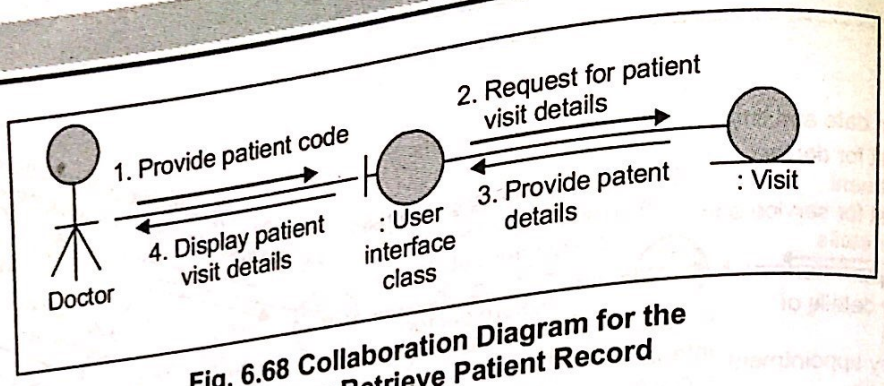


Fig. 6.68 Collaboration Diagram for the Use Case Retrieve Patient Record

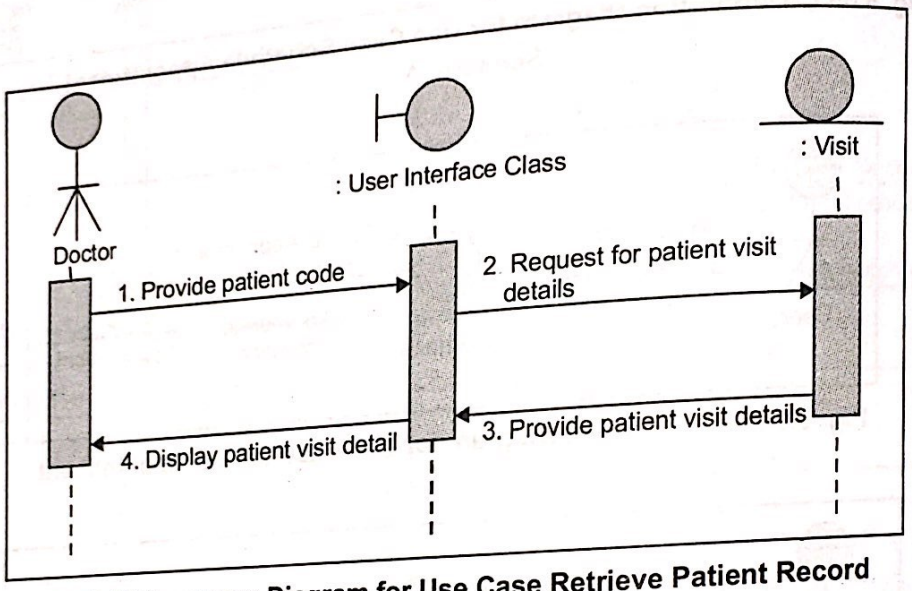


Fig. 6.69 Sequence Diagram for Use Case Retrieve Patient Record

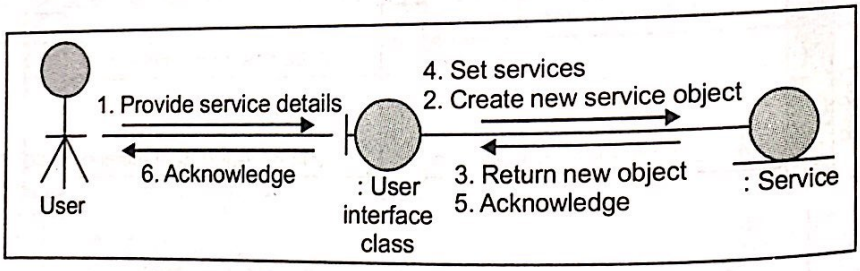


Fig. 6.70 Collaboration Diagram for the Use Case Add New Services

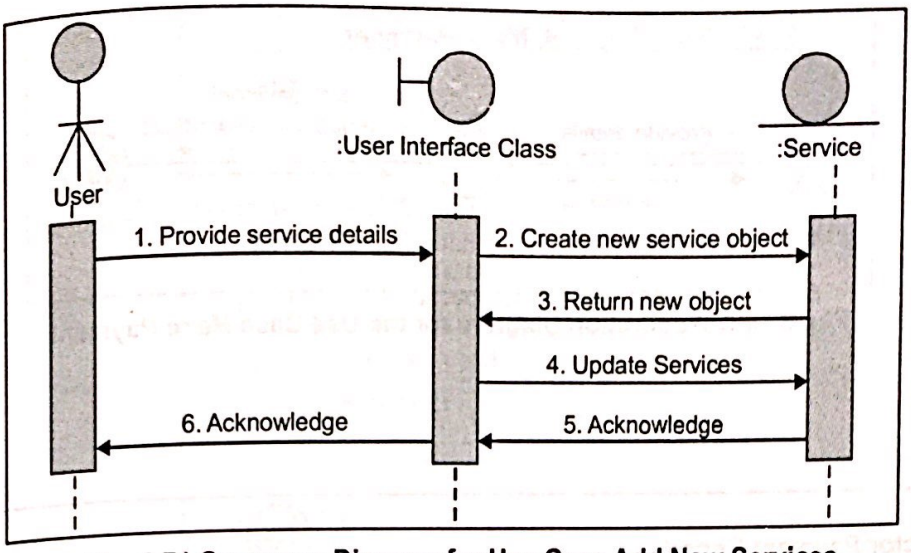


Fig. 6.71 Sequence Diagram for Use Case Add New Services

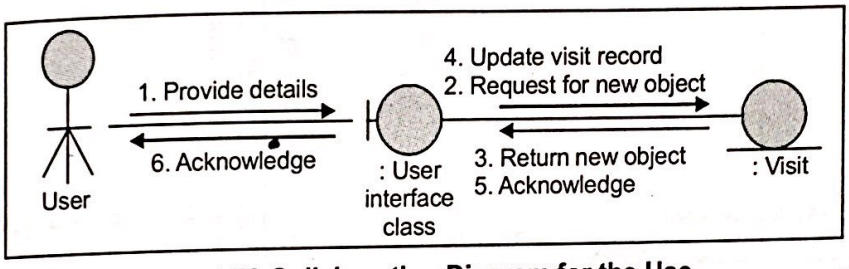


Fig. 6.72 Collaboration Diagram for the Use Case Update Patient Visit Record

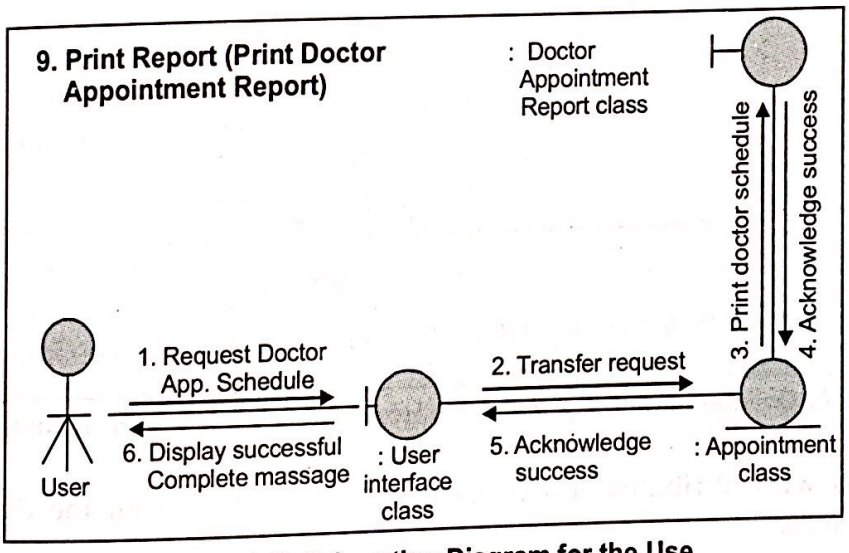


Fig. 6.73 Collaboration Diagram for the Use Case Print Doctor Appointment Report

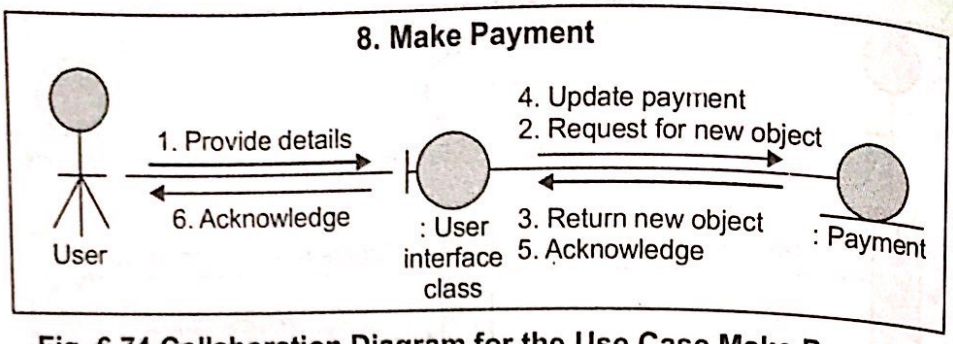


Fig. 6.74 Collaboration Diagram for the Use Case Make Payment

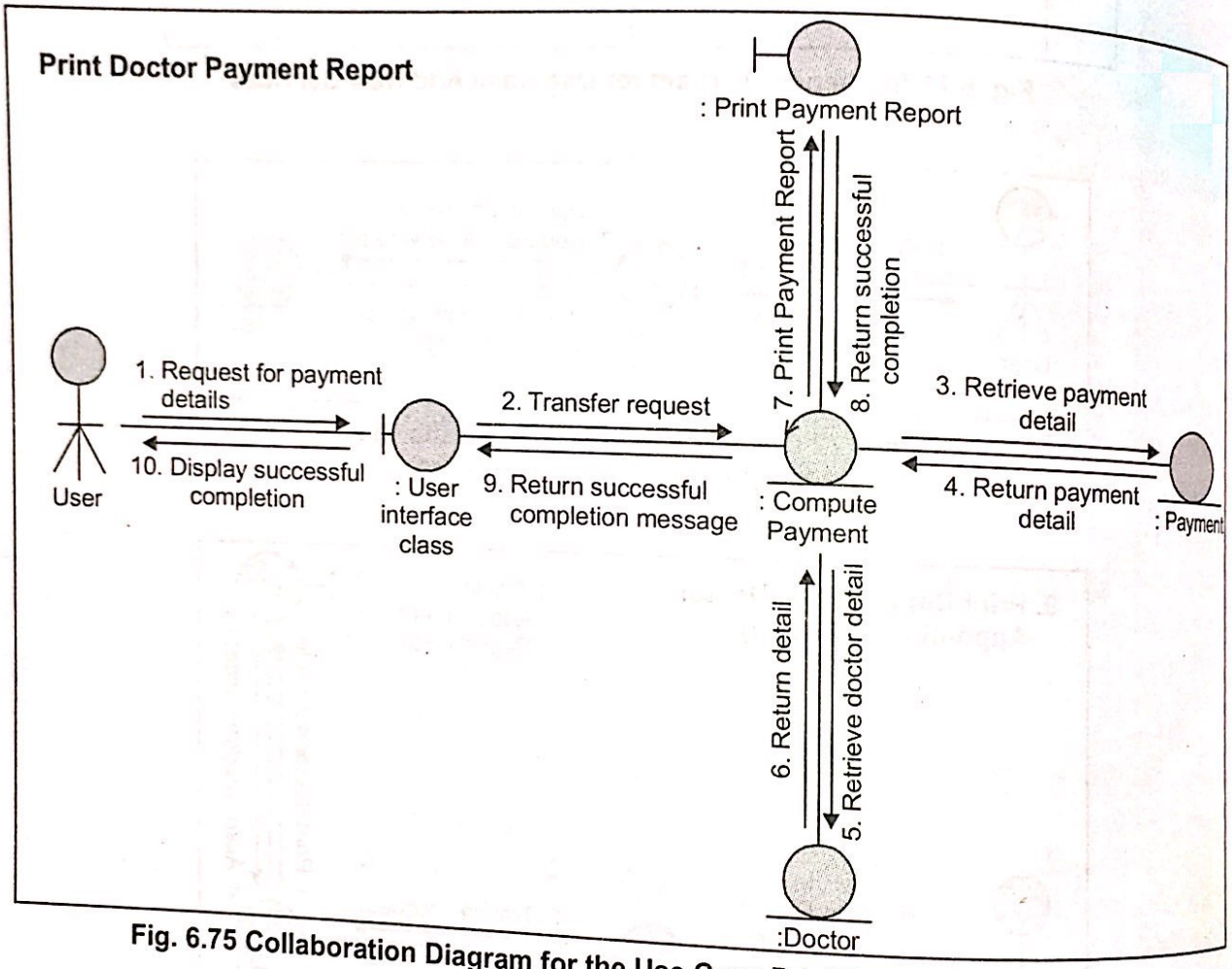


Fig. 6.75 Collaboration Diagram for the Use Case Print Doctor Payment Report

The Visit Class with attributes and operations as identified from the dynamic view is shown in Fig. 6.76.

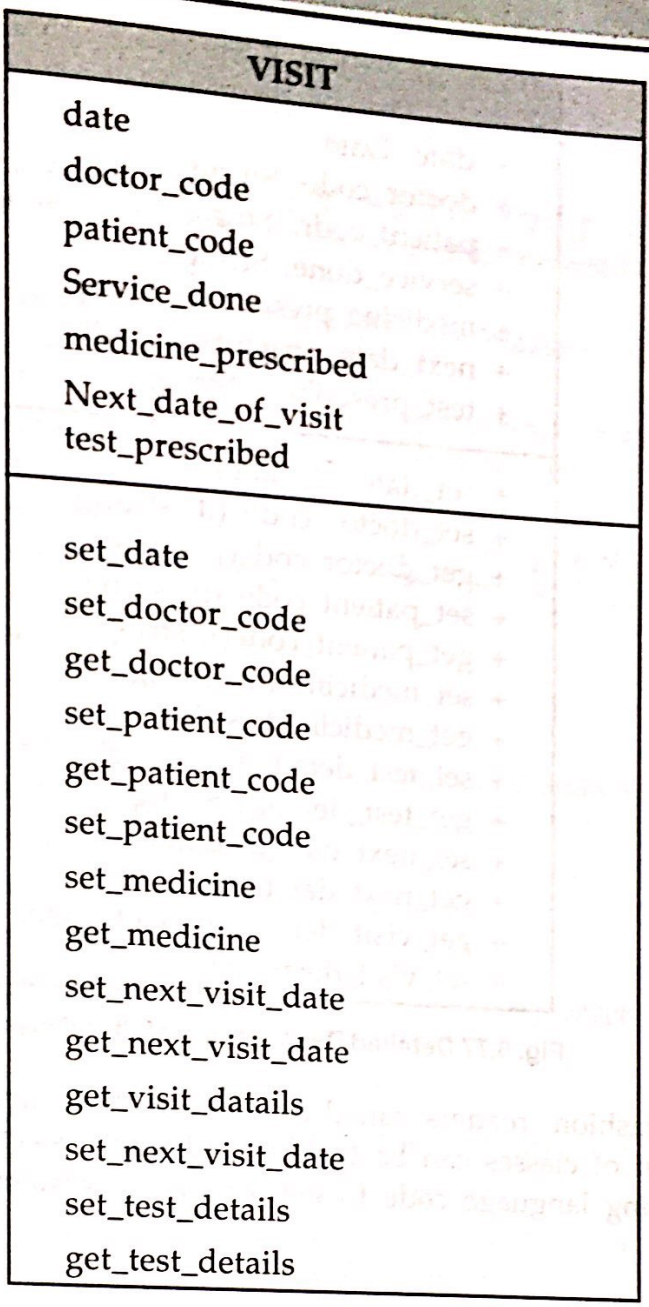


Fig. 6.76 Visit Class

Readers are adviced to identify and draw other classes also in the similar manner. In the design workflow, more details are added to the class in terms of data structures to be used with class attributis and the operations are also described in detailed description of Visit class is given in Fig. 6.77.

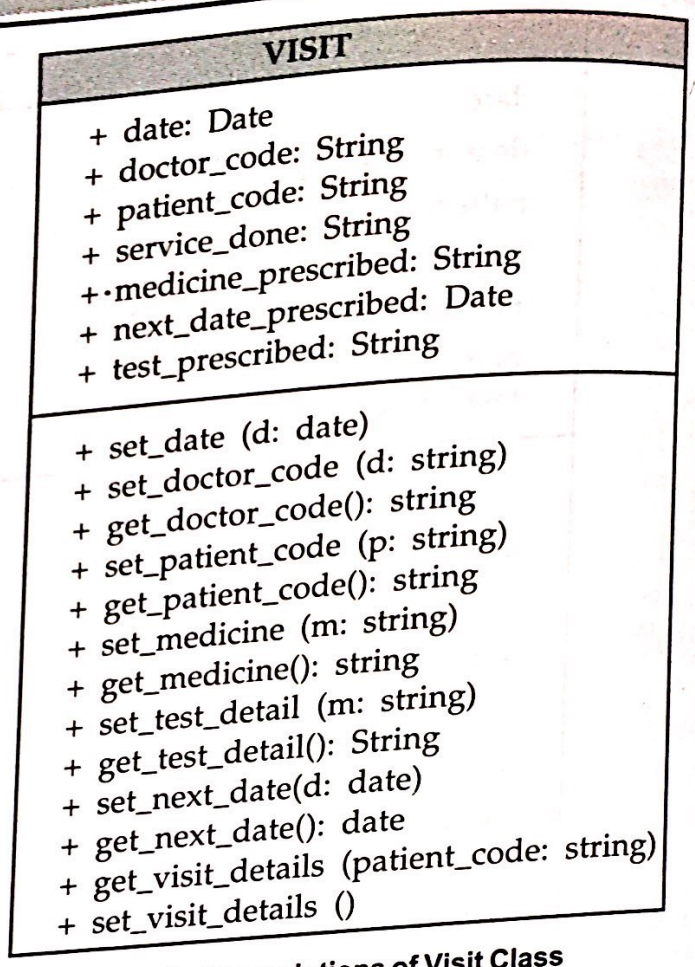


Fig. 6.77 Detailed Descriptions of Visit Class

In the similar fashion, readers can drow the detalied description of other classes. Once done, then set of classes can be transformed into C++/JAVA or any other object oriented programming language code to get a working system.

SUMMARY

- OORA is a technique for modeling problem domain in terms of objects and interactions among objects.
- A number of object oriented analysis and design methodologies like OOA, OMT, UML etc., are proposed but UML has almost become industry standard for developing software blueprints.
- UML makes use of 9 diagrams to model static, functional and dynamic view of the application domain.
- Class diagram is the most important diagram used for generating code and for reverse engineering.
- Other diagrams of UML i.e., Use Case diagram, Activity diagram, Collaboration diagrams, Sequence diagram and State chart diagram help in understanding the