Unit I

1 Introduction to Modeling

Syllabus

What is Object Orientation ? What is OO development ? OO themes; Evidence for usefulness of OO development; OO modeling history Modeling as Design Technique : Modeling; abstraction; The three models. Class Modeling : Object and class concepts; Link and associations concepts; Generalization and inheritance; A sample class model; Navigation of class models; Practical tips.

Contents

Part I : Introduction

1.1 What is Object Orientation ?

1.2 What is OO Development ?

1.3 OO Themes

1.4 Evidence for Usefulness of OO Development as Design Technique

1.5 Abstraction

1.6 The Three Models

Part II : Class Modeling

1.7 Object and Class Concepts

1.8 Link and Associations Concepts

1.9 Generalization and Inheritance

1.10 A Sample Class Model ..... **Aug.-15, Dec.-15, April-17, May-17,** ..... Marks 8

1.11 Navigation of Class Models

Part I : Introduction

1.1 What is Object Orientation ?

Software analysis and design is a software engineering approach which models the system as interacting objects. Each object represents a system entity which plays a vital role in building of that system. Each object has a state and behavior.

Object Oriented Analysis (OOA) focuses on analysis of functional requirements for the system. The Object Oriented Design (OOD) takes analysis model as input and produces implementation specification.

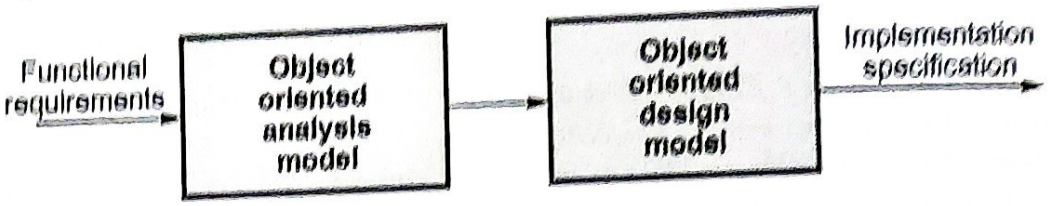


Fig. 1.1.1 Concept of OOAD

1.1.1 What is Object Orientation ?

Object orientation means organization of software as a collection of discrete objects. Each object consists of **data structure** and **behavior** which are closely coupled.

Following are some **basic characteristics** required by OO approach -

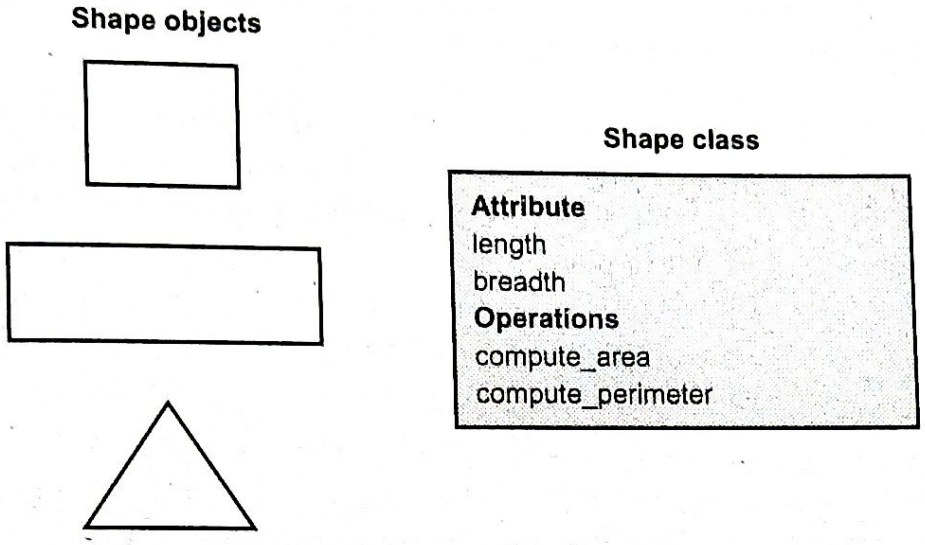
1. **Identity** : The identity means data is arranged in distinguishable entities called **objects**. Each object is treated as an inherent entity that means - each object is distinct even if their attribute values are same. In programming languages the object is referred by the memory references.

Suyash's ball

Neeta's pen

2. **Classification** : Classification is a technique in which objects of same data structure(attribute) and behavior(operations) are grouped into class. Each class can define infinite number of **objects and object is an instance of a class**. Each object can have its own values for attributes and operations but these objects can share the attributes and operations of other objects too.

For example - Shape can be a class and **rectangle, triangle and square** can be the objects of the class Shape. Following figure illustrates this idea -



3. **Inheritance :** Inheritance allows the class to share the property of another class in hierarchical manner. The **superclass** is a class whose properties are shared by the **subclass**. The **subclass** not only shares the properties of **superclass** but it can add its own unique properties. Thus the **super class** is a general class where as the **subclass** is a more specific class.

   For example - Animal is a super class but the **Dog, Cat, Cow** are the subclasses. Each subclass has its own unique features. For example if the sound is an attribute of class **Animal** then the sound of Dog, Cat, Cow are different.

   Due to inheritance property the repetition in the design can be avoided. Hence data reusability is the main ability in object oriented programming due to inheritance feature.

4. **Polymorphism :** Poly means many and morphs means forms. Hence polymorphism is a technique in which the same operation can be defined differently for the different classes. For example the operation **clean** can be used to clean the **dish** objects, **car** object or **vegetable** object.

In object oriented programming the language can automatically select the appropriate method to implement the required operation.

Example 1.1.1 *In object orientation if two objects are identical, does it mean they are equal ? Is object concrete or conceptual ?*

Solution : In object orientation, if two objects are identical, that does not mean they are equal. The term identity specifies that object is treated as inherent entity that means each object is distinct even if their attribute values are same. The object is an conceptual entity.

Review Question

1. *What do you mean by object-orientation ? Briefly discuss the characteristics of OO approach.*

1.2 What is OO Development ?

Development means the use of software life cycle such as analysis, design and implementation. In OO development the application concepts are identified and organized logically. The OO development and modeling is performed by focusing on -

- Modeling concepts and not implementation
- Use of OO methodology
- Use of three models

1.2.1 Modeling Concepts

- In OO development approach, the developer has to think in terms of application and therefore the data structures and functions are addressed effectively.
- In OO development the fundamental way of thinking is represented and the focus is not on the programming techniques.
- Using the modeling techniques the system that is to be built can be very well explained to specifiers, developers and the customers.

1.2.2 OO Methodology

- The OO development along with the graphical notations is used to represent the OO concepts.
- In this process the model of an application is built first and then the details are added to this model during the design.
- The same notations are used throughout the lifecycle of the software development process i.e. from analysis to design and from design to implementation. Due to this approach the important information need not be lost from one stage to another.
- **Stages of OO methodology :** The OO methodology has following stages -
  - **System conception :** In this phase the requirements of the system are specified by the business analysts or the users who want to use the system.
  - **Analysis :** The system analyst scrutinizes and reframes the requirements that are specified in the system conception stage. The analyst needs to understand the basic requirements of the system. He then prepares the analysis model. This model is **concise** and **precise abstraction** of **what** the system must do. It should **not** specify **how** the system should do. The implementation decisions should not be taken at

this stage. For instance - Database will be created using MySQL - This decision is not expected at this stage.

The analysis model has two parts - **domain model** and **application model**. The real world objects are described in domain model and in the description of application model the parts of the application system are described. For example domain objects for ATM system can be cash, card, card reader. The application model includes - updating the database on transactions, displaying the available balance to the Customer and so on.

- **System design** : During the system design the system architecture is built. The policies for the detailed design are decided in this phase. System designers take certain important decisions about the performance of the system. The tentative resource allocation is also done in this phase.
- **Class design** : The class designers add details to the analysis model using the system design strategies. Using OO notations and concepts the domain and application objects are elaborated. The **focus** of class design is on **data structures** and **algorithms**. For example - The class designers decide the data structures and algorithms for the **Customer** class.

  The classes and relationships are developed in this phase.
- **Implementation** : This is the phase in which the classes and relationships are transformed into the particular programming language, databases and so on. The implemented code must be straightforward and must follow good software engineering principles. It should be flexible and extensible.

- The same OO concepts such as identity, classification, inheritance and polymorphism are applied throughout the development process.
- The OO development process emphasizes on the **iterative development** process so that the parts of the system gets developed within several stages.
- Testing must be done at every stage of development. It is supposed to be the quality control activity.

1.2.3 Three Models

There are three types of model which describe the system. The **class model, state model** and the **interaction model**. These are considered to be the viewpoints of the system.

- **Class model** : The basic static structure of the objects and their relationships are described in this model. The **class diagram** is a graphical representation in which the nodes represent the **classes** and the arc represents the **relationships** among these classes.

- **State model** : The object changes its state over the time. This can be represented using the state model. **State diagram** is a graphical notation in which the nodes represent the **states** and the arcs within these nodes represent the **transitions** between the states caused by the events.
- **Interaction model** : The interaction model represents how the objects interact with each other in order to perform some task. The **use case diagram, sequence diagram** and **activity diagram** are the three kinds of diagrams drawn in interaction model. The **use case diagram** represents the functionality of the system, the **sequence diagram** represents the interacting objects and the time sequence of their interactions. The **activity diagram** represents important activities that are performed by the system.

Review Questions

1. *Describe in detail the stages of object oriented methodology.*
2. *Explain the 'three models' used in object oriented technology.*

1.3 OO Themes

Various OO themes are

1.3.1 Abstraction

Abstraction is a mechanism in which only essential aspects of an application are focused while other aspects are ignored. Due to abstraction developer is free to take some decisions before detailing the design.

1.3.2 Encapsulation

It is also known as **information hiding**. In this technique the internal implementation decisions of one object are hidden from the other object. In encapsulation, the data structure and behavior are bound together in one entity class. Due to encapsulation one can make changes in the behavior of one object without affecting the remaining part of the system.

1.3.3 Sharing

In OO technique, sharing of the code at different levels is possible. Using concept of **inheritance** the sharing is possible. There are two advantages of sharing - The code gets reduced and conceptual clarity can be brought in the application development. The sharing is possible not only within the modules of particular application but also in future projects. Using the libraries of abstraction, encapsulation and inheritance the library of reusable components can be created.

1.3.4 Synergy

The identity, classification, inheritance and polymorphism are the characteristics of OO languages that can be used in isolation. But if all these properties are used together then they complement each other synergistically. This is beneficial for the OO approach of development. Due to development using-the focus on the object results in clear and general development of the application.

Review Questions

1. *Explain characteristics and themes of object oriented systems in brief.*
2. *Briefly explain following characteristics and themes of object oriented systems : Classification, identity, inheritance, encapsulation, polymorphism,sharing, synergy.*
3. *What is object orientation ? Explain OO themes.*

1.4 Evidence for Usefulness of OO Development as Design Technique

Model is abstraction of the system and is created for the sake of understanding. Only important aspects of the system are represented in the model.

Designers build various types of models - For example - Blueprints can be made before building the house. Drawings to show the machine parts can be created. Following are some purposes - **"Why models are created ?"**

- **Communication with customer :** Architects and system designers represent the model to their customers in order to give the mock demonstration of the system.
- **Testing of physical entity :** If the actual system is built and then tested then it is costly affair. Some sort of testing can be done by creating a physical model or computer model. The simulation of the system by the model is not only cheaper but it informs the developers about the problems that may come occur in actual system.
- **Visualization :** Using the model system can be represented for visualization. The major elements of the system, some important activities that can be performed by the system, useful events and transitions can be shown by th model. The Customer can visualize the system due to model.
- **Complexity reduction :** The complexities of the system can be reduced by modeling the system and separating small number of important things to deal with them separately.

Review Question

1. *Elaborate - Why models are created ?*

1.5 Abstraction

- Abstraction means examining the selective area of the problem.
- The **goal** of abstraction is to separate out all the important aspects of the specific area of the problem and ignore all unimportant aspects of that area.
- There is always some purpose for the abstraction. Similarly many different abstractions can be possible for the same thing.
- All abstractions are incomplete or unimportant. In fact the abstraction is nothing but the **small summary of particular problem.** Describing simply summary of the problem is also useful thing. Because by abstracting the problem in this way, it becomes easy for us to understand the problem. Thus purpose of an abstraction is to limit the universe so that we can understand the problem.
- A good model should capture only the crucial aspects of the problem and should omit the others. Sometimes a model may contain certain extra details of the problem because of which developers attention gets diverted from the real issues. In such a situation abstraction is the only thing which helps the developer to focus the attention on specific goal of the problem.

1.6 The Three Models

- There are three different viewpoints used to model the system.
- The **class model** represents the **static, structural, data** aspects of the system.
- The **state model** represents the **temporal, behavioral, control** aspect of the system.
- The **interaction model** represents collaboration of objects and **interaction** aspect of the system.
- Normally all these three aspects are **incorporated within a single system.** There is use of data structures (class model), operations of the system are executed in some specific sequence (state model) and data as well as control is passed among the objects (interaction model).
- Using these models **distinct views** of the system can be created.
- These models are not completely independent. There is **limited and explicit interconnection** among these objects. The good design isolates different aspects of the system and have limited coupling.
- These models **evolve** during the development.
- Analysts construct a model without thinking of implementation. The designers add solution constructs to the model. Implementers write the code for model and solution construct.

- Thus the model has **two dimensions** - first dimension includes the **views** such as class model, state model and interaction model. Other dimension includes the **stage of development** such as analysis, design and implementation.

1.6.1 Class Model

- The class model describes the structure of the object, the relationship of one object with other objects, attributes and operations of the objects.
- The state and interaction models use the context of the class model.
- The goal of constructing class model is to capture the concepts from the real world that are useful for the application.
- While modeling the particular system the terms that are familiar to that system must be used.
- The analysis model should not contain the **computer constructs** but the design model contains the **computer constructs** and solution to the problem.
- The class model is represented by the **class diagram**. The classes in this diagram define the **attributes and operations** of each object. The **association relationship** is used to link the classes together. The **generalization relationship** is used for a group of classes to share common data structures and operations.

1.6.2 State Model

- The state model is concerned with the time and sequencing of the operations of the object.
- On occurrence of the events the object change their current state.
- The state model captures the **control** aspect of the system.
- The state model is represented by **state diagram.** Each state diagram represents state and the event sequences.
- Actions and events in the state diagram becomes the operations of the object in the class model. References between the state diagram become interactions in interaction model.

1.6.3 Interaction Model

- Interaction model describe how one object collaborates with other in order to achieve behavior of the system.
- The overall behavior of the system can be represented with the help of state and interaction model.

- The interaction model includes **use case diagram, sequence diagram and activity diagram.**
- The use case diagrams show how the outsider actors interact with the system to achieve the functionality.
- The sequence diagram represents the objects that interact and the time sequence of their interaction.
- The activity diagram represents flow of control among various objects.

1.6.4 Relationship Among the Models

- The models are related with each other. Every model represents one aspect of the system.
- The class model describes the data structures. The state and interaction model represent the operations performed on these data structures.
- In class model every class performs some set of operations. These operations cause the events and actions which are highlighted by the state and interaction model.
- The state model describes the control structure of objects. The state model represent the decisions depending upon which the object values and states get changed.
- The interaction model represents how the objects interact with each other.
- There are some things that the models do not capture adequately but using the natural languages or application specific notations the detailing of the system can be done.

Review Questions

1. *Which different purposes are served by models ? Explain all three models which are required to describe the complete system.*
2. *Explain object oriented model, dynamic model, functional model.*
3. *Explain object oriented model, dynamic model, functional model and interaction model and relation among these models.*
4. *What is model ? Explain purposes of models. Also explain types of models.*

Part II : Class Modeling

1.7 Object and Class Concepts

1.7.1 Objects

- The **main purpose** of class model is to describe objects.
- Object is an instance or occurrence of a class. It is a concept, abstraction or thing with identity and meaning.
- The objects can be conceptual entities, real-word entities or important things from implementation point of view.
- The objects are normally nouns. The choice of object is done by judgements.
- All objects are distinguishable entities.
- For example - If **Student** is a class then **Anuja, Supriya** and **Shilpa** are the objects of the class students. Each student has its own *name, roll number* and *address.* Each object is different from other object. That is every object has its own **identity**.
- **Identity** means the objects are distinct due to their inherent existence and not by their description.

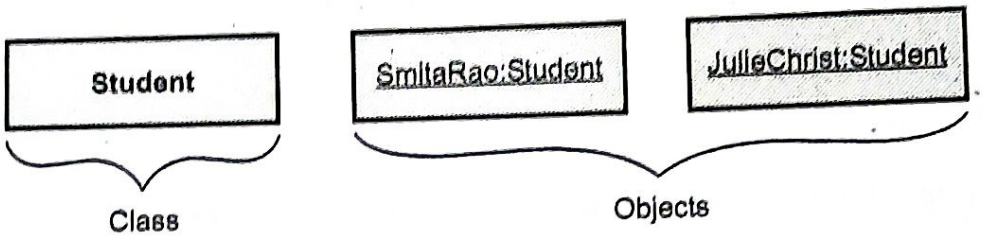
1.7.2 Classes

- Class is a group of objects having same attributes and operations, relationships and semantics.
- For example - Student, Employee, Company, Course all are classes. Each Student has rollnumber, name and address.
- The classes appear as common nouns or noun phrases.
- Objects of particular class have same attributes and behaviour but each object is different from the other due to values of its attribute, different behaviour and relationship with other object.
- Objects in a class share a common semantic purpose. For example both the **dog** and **cat** have the properties like tail and legs and they belong to the same class **Animal**.
- Each object belongs to some class and that object is aware of its belonging class.
- Grouping the objects into corresponding classes make the design abstract.

1.7.3 Class Diagrams

- The class model is represented by two types of diagrams - **Class diagram and object diagram**.

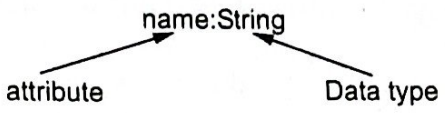
- Class diagram is a **graphical representation** used for modelling classes and their relationships. It describes all possible objects belonging to the classes.
- Class diagram is used for abstract modelling and for implementing actual program.
- The class diagram is concise and can be understood easily.
- It is most common to make use of class diagram in the OO modelling.
- The **object diagram** represents the objects and their relationships with each other.
- Using the one class diagram various object diagrams can be created.
- The object diagram is used for documenting the **test cases.**
- The class and object is represented graphically by the box. The class name must be written at the center of the box and the first letter is capitalised. It is written in boldface.
- The object name is followed by the colon and the class name. Everything must be underlined in the box that represents the object.
- The multi-words are written without space having the first letter of the separating word capitalized.



1.7.4 Values and Attributes

- Attribute is a named property of a class and it can hold some value.
- Adjectives in the problems statement generally represent some attributes.
- For example - Student class might have name, address, phone number as the attributes. Each attribute have values. For instance - name can be "AAA" for one object and can be "PPP" for another object.
- Different objects may have same or different values for its attributes.
- The name of the attribute is always unique within the class.
- The values are always assigned to the attributes. The values alone have no identity. They can be recognised with the help of belonging attributes. For instance - If you say "AAA" then it is just meaningless but if "AAA" is assigned as a **name** of the **Student** then it makes some sense.

- The attributes are always written in the second compartment of the class. It may be written along with its data type and default value. For example -



  The default values can be written by using equal to sign and the value.
- Many times objects have **unique identifiers**. For example **studentNumber** is an unique identifier which is an internal identifier and it should not be considered as an attribute. But there are some real-world identifiers such as **RollNumber, EMPID, TelephoneNumber** and so on these must be mentioned as the attributes. In fact, these kind of attributes are important attributes.

1.7.5 Operations and Methods

- Objects have procedures or functions which are called as **operations**.
- All the objects in the same class share the common set of operations.
- For example - The class **Shape** can have various objects such as **rectangle, triangle** or **square** having the common set of operations such as - **move, draw, get_area** and so on.
- The same operation can be applied to different classes or it can be used for different purposes. Such operations are called as polymorphic operations. For example **get_area** operation can calculate the area of square or triangle or rectangle.
- **Method** is an implementation of operation for a class. Some piece of code is used to implement each method. For example code written for calculating the area of square is the method.
- Any number of arguments can be passed to the operations.
- **Signature** is nothing but the number and types of the arguments and type of the return value of the operation. For example, **int get_area(int l, int b)** - Here the two arguments int l and int b along with the return type int represent the signature of the operation.
- The generic word used for either attribute or an operation is called the **feature**.
- The operations are always written in the third compartment of the class box.
- The names of the operations must be written in regular face. The first letter of the operation must be in lower case. We can mention the names of the arguments along with the data type. The arguments can be specified within the parenthesis. Similarly we can specify the return data type of particular operation by preceding the colon. For example -

  get_area(l:int,b:int):int

1.7.6 Summary of Class Notations

- As we know, the class can be represented using the box. The first compartment contains the **name of the class**, the second compartment contains the **attributes** and the third compartment contains the **operations** belonging to that class. We can specify the data types as well as the default values to the attributes and operations. The list of arguments can also be passed to the operations.

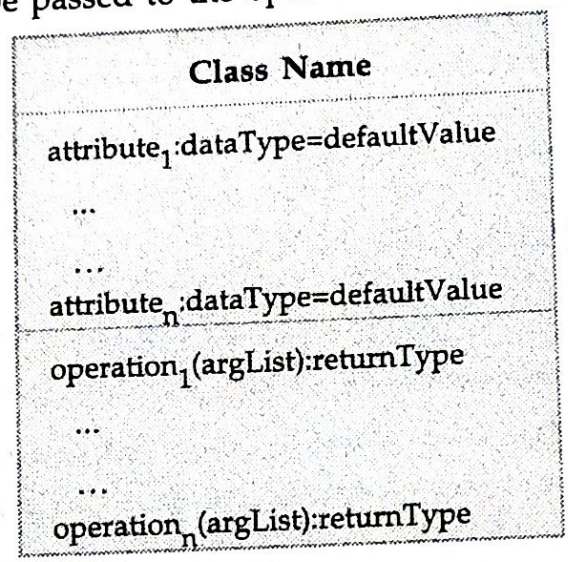


Fig. 1.7.1 Class

- Each argument of the operation can have **direction**. The direction can be input(in), output(out) or any modifiable mode(inout). Using the equal to sign the default value can be represented.
- The attributes or operation compartments are optional. Missing attribute compartment means the attributes are not specified. Similarly missing operation compartment means that the operations are not specified.

1.8 Link and Associations Concepts

1.8.1 Links and Associations

- Link is a connection between the objects. It represents a simple association of one object with another.
- Mathematically link can be defined as a list of objects. A link is basically an instance of association.
- An **association** is a group of links that have common structure and common semantics.
- The links and association often appear as **verbs** in the problem statement. The link is between the two objects and association is between the two classes.

- For example - Following is a part of **Online Course Management system**. The upper part represents the part of a class diagram and the bottom part represents the partial object diagram of the system. The * indicates the multiplicity.

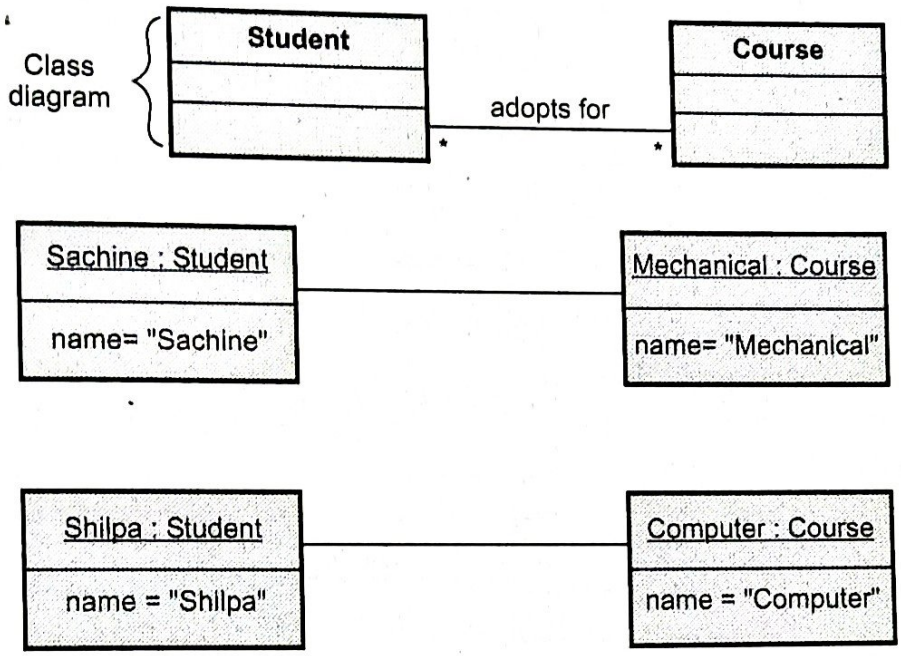


Fig. 1.8.1 Links and associations

- If the model is unambiguous then the association name is optional.
- The associations are inherently bidirectional, however the association is read in one particular direction. For example in above class diagram, the **Student** adopts for particular **Course** represents the flow in one direction.
- Associations can be implemented with the help of **references**. Reference means the attribute of one object can be referred by another associated object. For example attribute of class **Student** can refer the object **Course**.
- Due to **encapsulation** the operations are kept private to a class. But the association breaks this encapsulation, in the sense one class can be correlated with the other class. Thus hidden things in the program can be avoided and program can be extended further if needed.

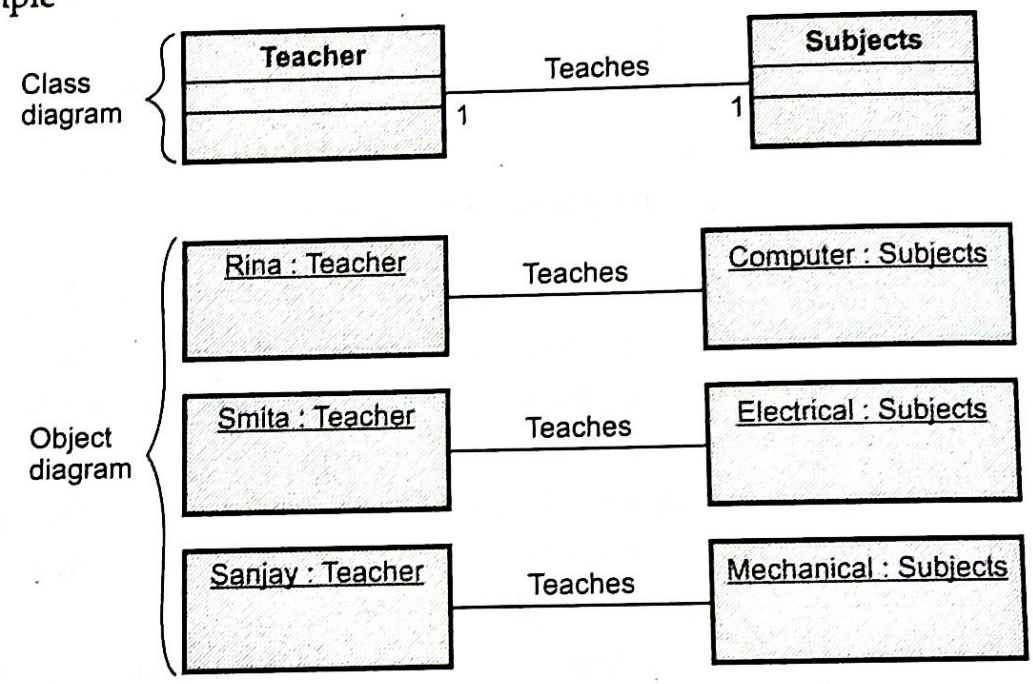
1.8.2 Multiplicity

- Multiple objects can be related to some objects. The multiplicity represents "how many" objects are connected.
- The multiplicity can be written as expression. It describes "one" or "many" objects associated with other object.
- The multiplicity is specified at the end of the association link.

- Following are the notations that can be used to denote the multiplicity of associations -

Notations	Description
1	Only one instance
0..1	Zero or one instance
*	Many instances
0..*	Zero or many instances
1..*	One or many instances
2..10	You can specify the integer range

For example -



- There is a difference between two terms - the **multiplicity** and **cardinality**. The multiplicity is the constraint on the collection of the associated **objects** whereas the cardinality is the count of the objects that are in collection. The multiplicity is actually the constraint on the cardinality.
- While making the class model first decide about the classes and their associations and then think about their multiplicity.

1.8.3 Association End Names

- The multiplicity can be assigned to the ends of the association. For example if there is one to many multiplicity then at one end **one** is written and at the other end **many** is written.

- For clear understanding of the association relationship we can write the names at the association.
  For example -

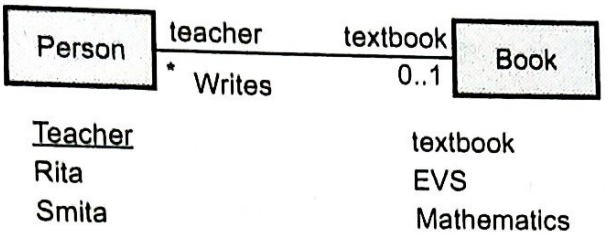


Fig. 1.8.2 Association end names

- The traversing direction can be understood due to the association end names.
- Association names server the important role in associating two objects of the same class. In the following Fig. 1.8.3, **container** and **contents** are the two usage of the class **Form**. A form can contain a link to another form or it contains some content to be filled up by the Student. Hence the self association is made to the class **Form**.

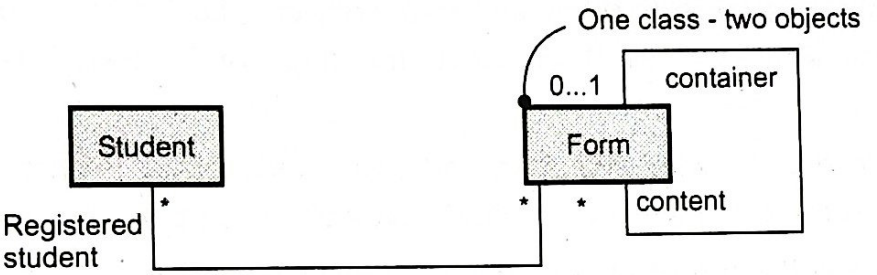
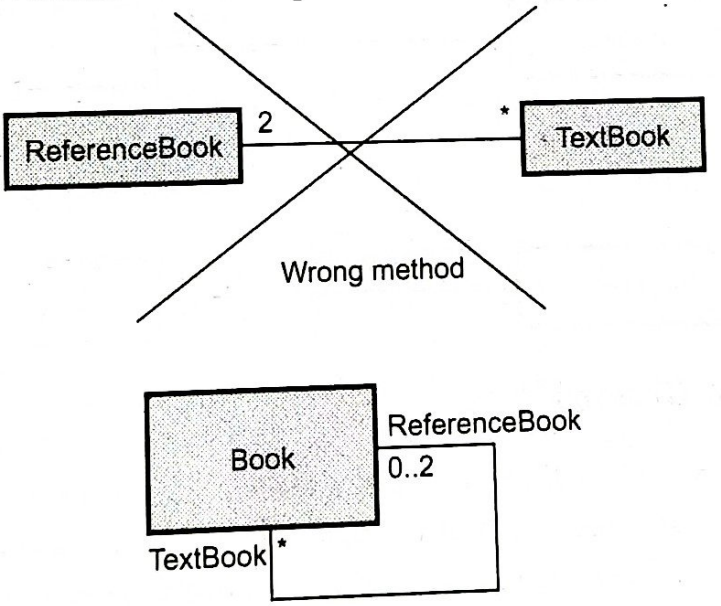


Fig. 1.8.3 Association end names for the two objects of same class

- If there are multiple references of the same class then for each reference separate classes must not be created. For example -



Correct method

Fig. 1.8.4 Association end names

- The association end name must not be the same as attribute name.

1.8.4 Ordering

- This type of association is used to denote the set of objects at one end must appear in some specific order. For example -

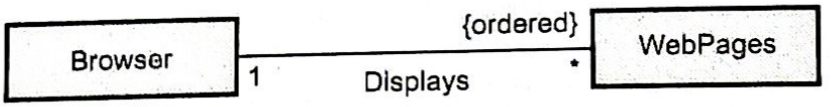


Fig. 1.8.5 Ordering the objects

- The web browser displays the web pages in some specific order (first come first served). Hence the {ordered} keyword is used at the association end (the end at which the objects should appear in some specific order).

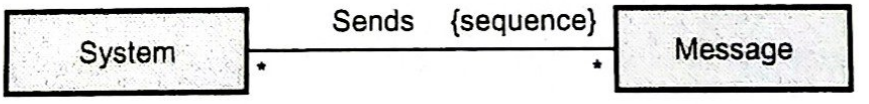
1.8.5 Bags and Sequences

In association relationship there are some objects that may be duplicated or may appear in some sequence. In these cases the **bags** and **sequences** are used at the association ends.

Bag is a collection of objects at other end that are duplicated. That means if the word **bag** appears at some end then the object at that end can appear more than once.

Sequence is a collection of objects at other end that are duplicated or appear in some sequence.

For example -



1.8.6 Association Classes

- The abstract class is a class that allows the association to be a class itself. That means when two classes are related with each other by an association link, then the association itself can have attributes and operations. Hence this association can be represented by a class. For example - In the given illustration the **Enrollment** is an association class for the classes **Student** and **Course**.

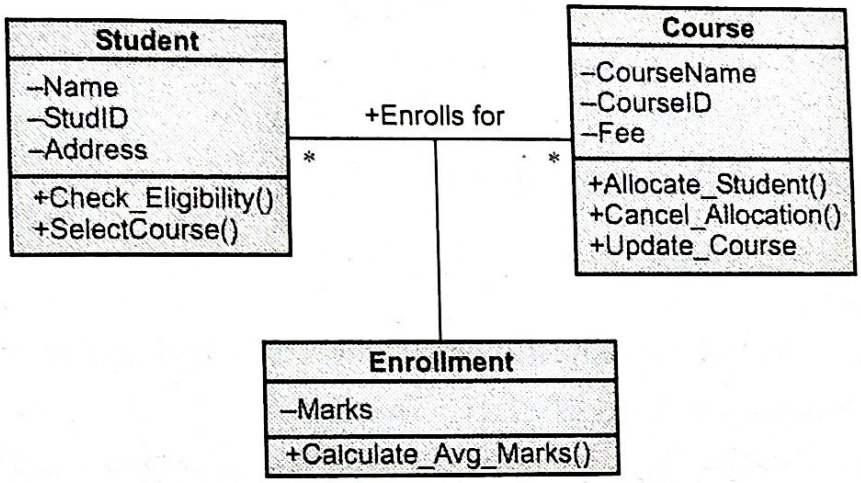


Fig. 1.8.6 Association class

1.8.7 Qualified Associations

- The qualified association has **qualifier** which is used to select particular object from the set of objects. For example - For selection of particular **Student** from the **Course** the qualifier will be **StudentID**.

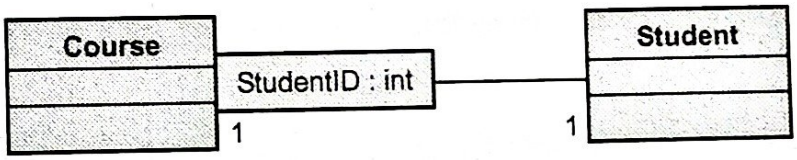


Fig. 1.8.7

- A qualifier is a property which defines **selection key.** UML allows having a qualifier on each end of a single association. The multiplicity can be associated with the association link.

Review Questions

1. *Define the purpose of the following term with suitable example and UML notation with respect to class model - Multiplicity.*
2. *Define the purpose of the following term with suitable example and UML notation with respect to class model - Association class.*
3. *Is association class same as ordinary class ? Explain with example.*
4. *Define the purpose of the following term with suitable example and UML notation with respect to class model - Qualified association.*
5. *Define the following terms :*
   1. Ordering 2. Bags 3. Sequence 4. Association class
   5. Qualified associations 6. OCL 7. Link.

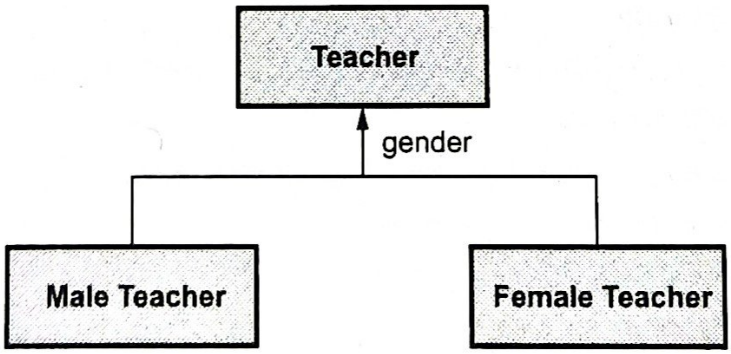
6. *What is inheritance ? List the different types of inheritance and explain how it encourages reusability and sharing.*
7. *Explain 'ordered', 'bags', 'sequences' in class diagram with suitable examples.*

1.9 Generalization and Inheritance

1.9.1 Definition

- **Generalization** is the relationship between a superclass and one or more subclasses.
- Each subclass inherits the features of its superclass.
- The **superclass** is more **general** and it holds common attributes, operations and associations. Whereas the **subclass** is more **specific**. It uses the common attributes, operations and association of superclass but adds its specific attributes, operations and associations.
- Generalization is also called as **is-a** relationship. This is because, each instance of subclass is an instance of the superclass.
- Generalization relationship can be represented using an arrow having hollow arrowhead. The arrowhead points to the superclass.

- There are various types of generalizations, each subclass can have single immediate superclass or there can be multiple levels of the generalizations.
- The classes from which the subordinate classes inherit the properties are called **ancestors** whereas the classes that use the properties of the parent classes are called **descendant**.
- **Generalization set name** is an enumerated attributes which indicated the factor on which the subclasses are inherited. For example - In the following figure, **gender** is the Generalization set name



- The depth of the generalization hierarchy must not be too deep. Otherwise it becomes difficult to understand and maintain.

- **Examples of Generalization**

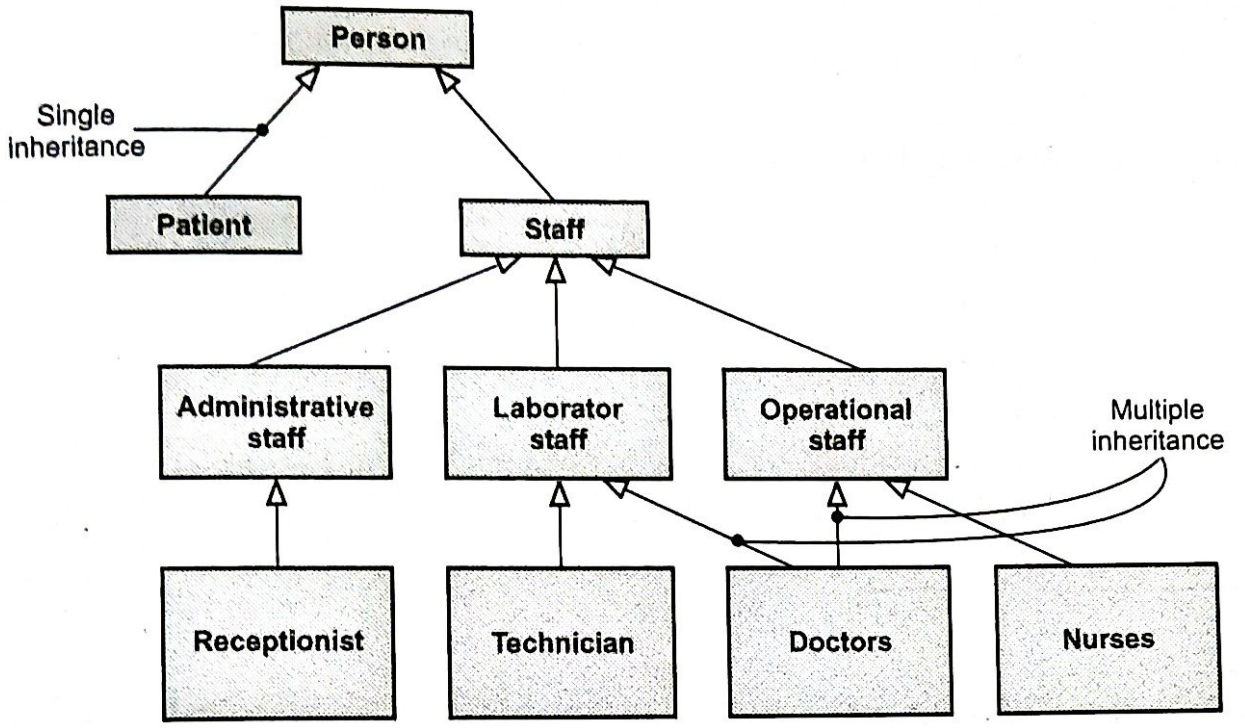
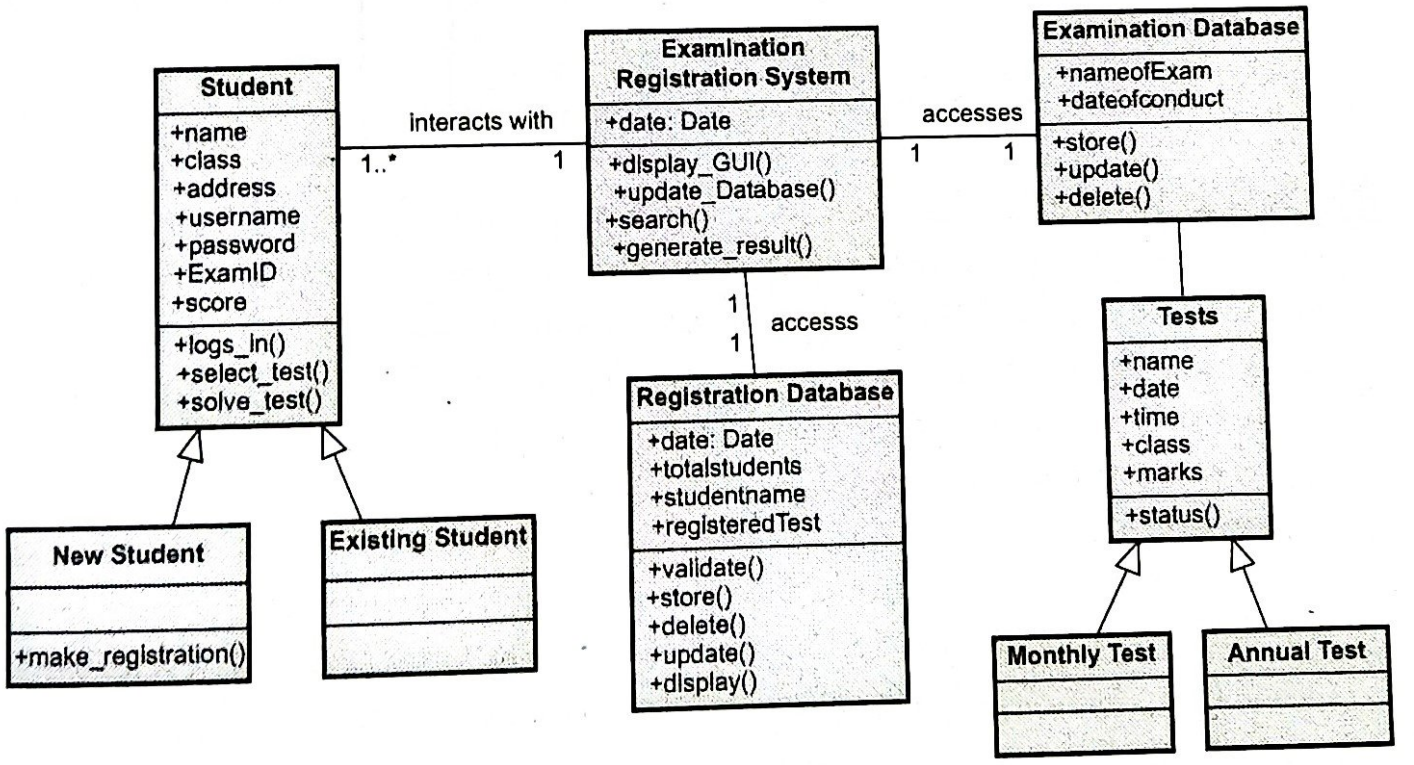


Fig. 1.9.1 Types of inheritance

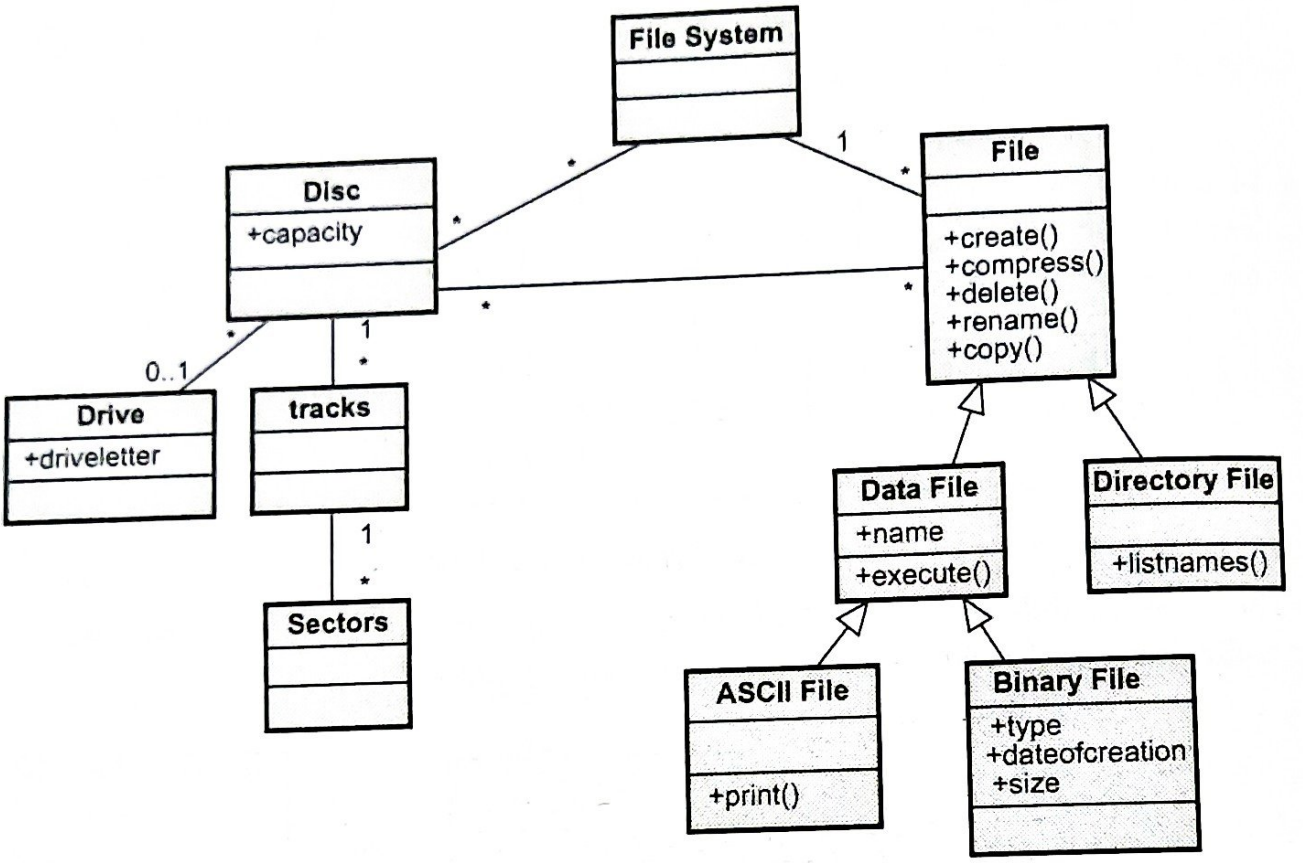
The class model for online examination registration system is as given below. The generalization relationship is shown for the Student class and Tests class.



Example 1.9.1 *Prepare a class diagram for each group of classes. Add at least 10 relationships (association and generalizations) to each diagram.*

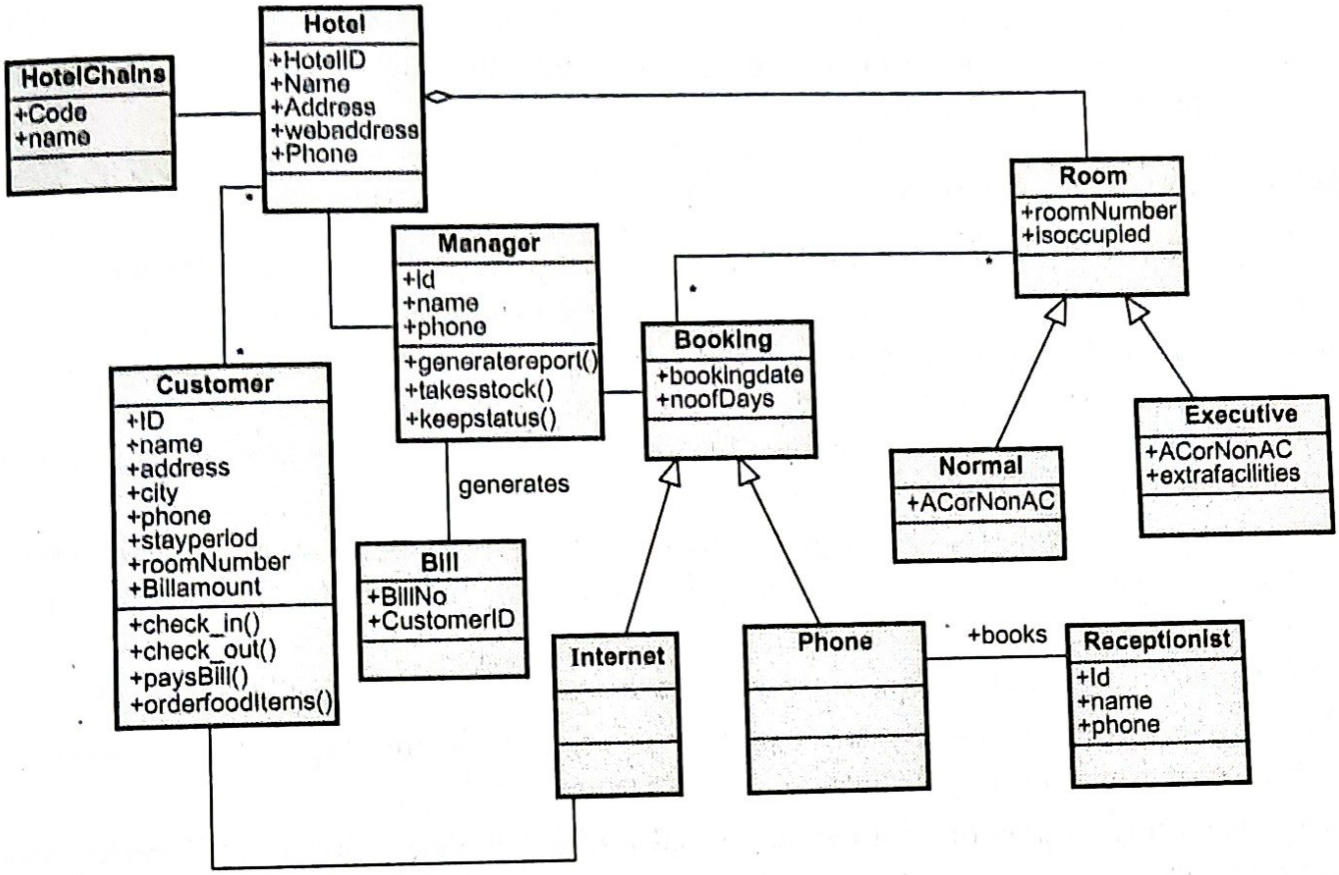
File system, file, ASCII file, binary file, directory file, disc, drive, track, sector

Solution : The required class diagram is given below -



Example 1.9.2 *Prepare a class model for the hotel management system. The system should support chain of hotels. A hotel contains two categories of rooms : Executive and normal, both AC and non-AC. The customers of executive rooms can avail extra facilities like games, swimming, food service in rooms, etc. The booking is possible by Internet or by phone. If the booking is through phone, process is done by receptionist and if booking is done through Internet the process is carried out by customer through hotel website. Depending on the number of days customer stays, appropriate bill is generated. The bill also contains amount for transport, food and other facilities enjoyed by the customer along with necessary taxes. The manager should be able to generate reports like list of customers staying in the hotel, list of rooms empty, monthly/yearly income, etc.*

Solution :



1.9.2 Use of Generalization

- There are **three main uses** of the generalization class -
  1. **Polymorphism :** Polymorphism is an ability of object oriented design in which the operation can be called at the superclass level, and the same operation with the same name can be defined at the subclass level doing different functionalities. The compiler automatically selects appropriate operation that matches the object's class. Due to this feature the software design becomes flexible. The new subclasses can be added without changing the rest of the code.
  2. **Structuring of description of objects :** Due to generalization, the designer can form the taxonomy of objects. Based on similarities and difference of the objects the classes can be arranged. That also means if there are some classes that share a **is-a** kind of relationship then using generalization concept these classes can be structured in hierarchical form.
  3. **Reusability of code :** As the common properties defined by the superclass are also used by the subclasses, there is reusability of code. If a library of some useful or most commonly required classes is created then due to generalization, the designer can inherit some classes from the library. This actually brings reusability of the code.
- **Generalization, specialization** and **inheritance** are used for in a common reference. The terms generalization and specialization are complement to each other. The

generalization refers to some common properties while specialization refers to the specific property or purpose. The inheritance is the technique in which attributes, operations and relationships are shared by the set of classes using the generalization/specialization concepts.

1.9.3 Overriding Features

- **Overriding features** include overriding methods or default values of attributes.
- The subclass overrides the superclass features. That means the overriding features are redefined or replaced by some another value in the subclass, but they posses the same name. for instance - the **display** method of superclass can be overridden in subclass. It means we can redefine the **display** method in subclass having different functional body but possessing the same name.
- The overridden features are used in OO design for various reasons - i) For defining the different behaviour in the subclass ii) For having the specific functionality in the subclass.
- The signature or datatype must not be overridden.
- While overriding, attribute type, number of arguments and type of arguments and return type must not be changed.
- The overriding of general method to special method increases the overall performance of the code.

Example -

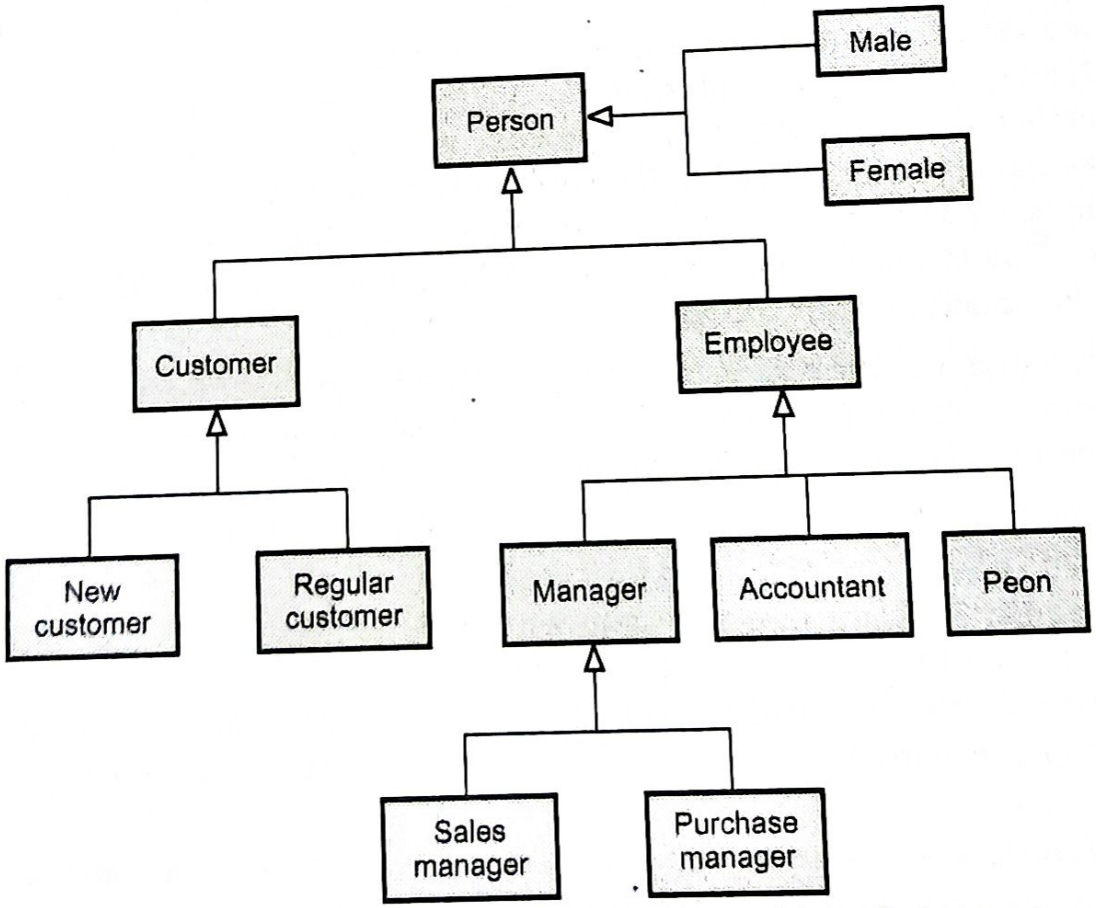


Fig. 1.9.2 Class model representing hierarchies of classes

Review Question

1. *Explain the following terms in relation to generalization and inheritance.*
   a. Generalization set name b. Override.

1.10 A Sample Class Model

SPPU : Aug.-15, Dec.-15, April-17, May-17, Marks 8

Example 1.10.1 *Prepare a class model to describe geographical map. Map contains roads, rivers and mountains. All components are described by points representing longitude and latitude.*

Solution :

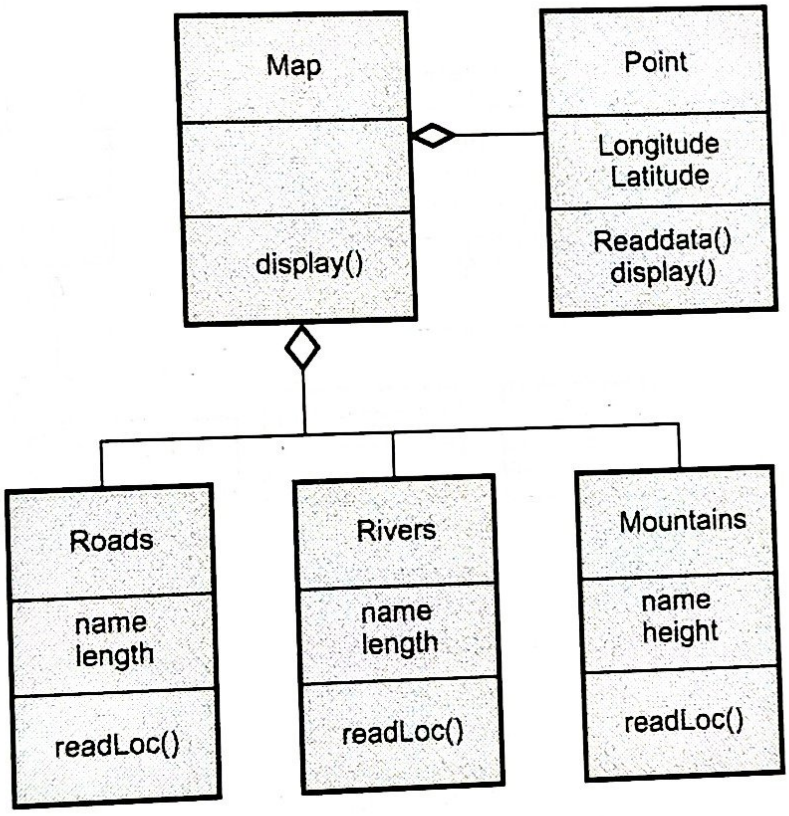


Fig. 1.10.1 Class model for geographical map

Example 1.10.2 *Create a class hierarchy to organize the following windows system : Window, scrolling window, panel, canvas, text window, scrolling canvas, button, choice item, shape, line, closed shape, ellipse, polygon.*

Solution :

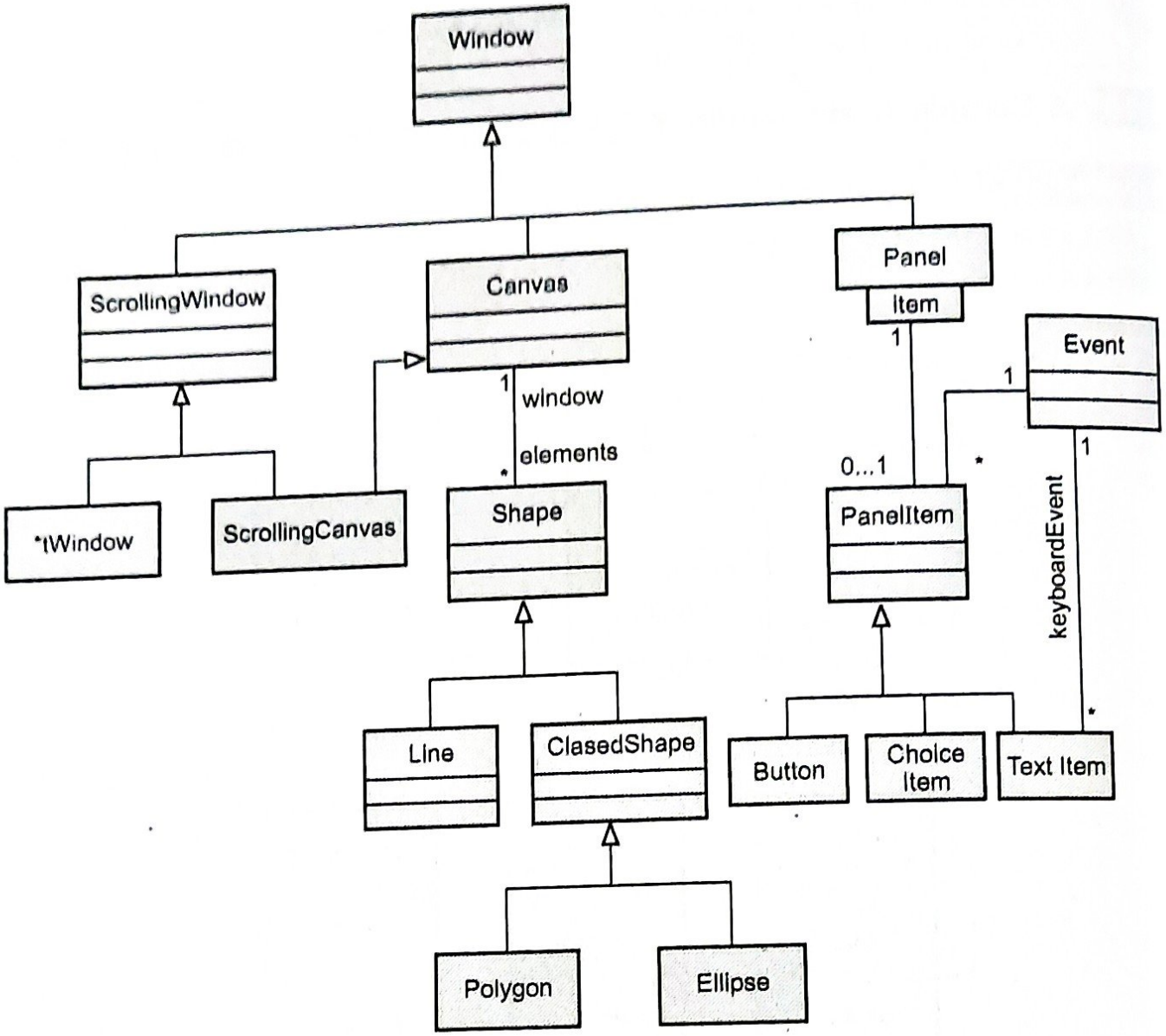


Fig. 1.10.2

Example 1.10.3 *Prepare a class diagram for group of classes. Sink, freezer, refrigerator, table, light, switch, window, smoke alarm, burglar alarm, cabinet, bread, cheese, ice, door, kitchen.*

Solution :

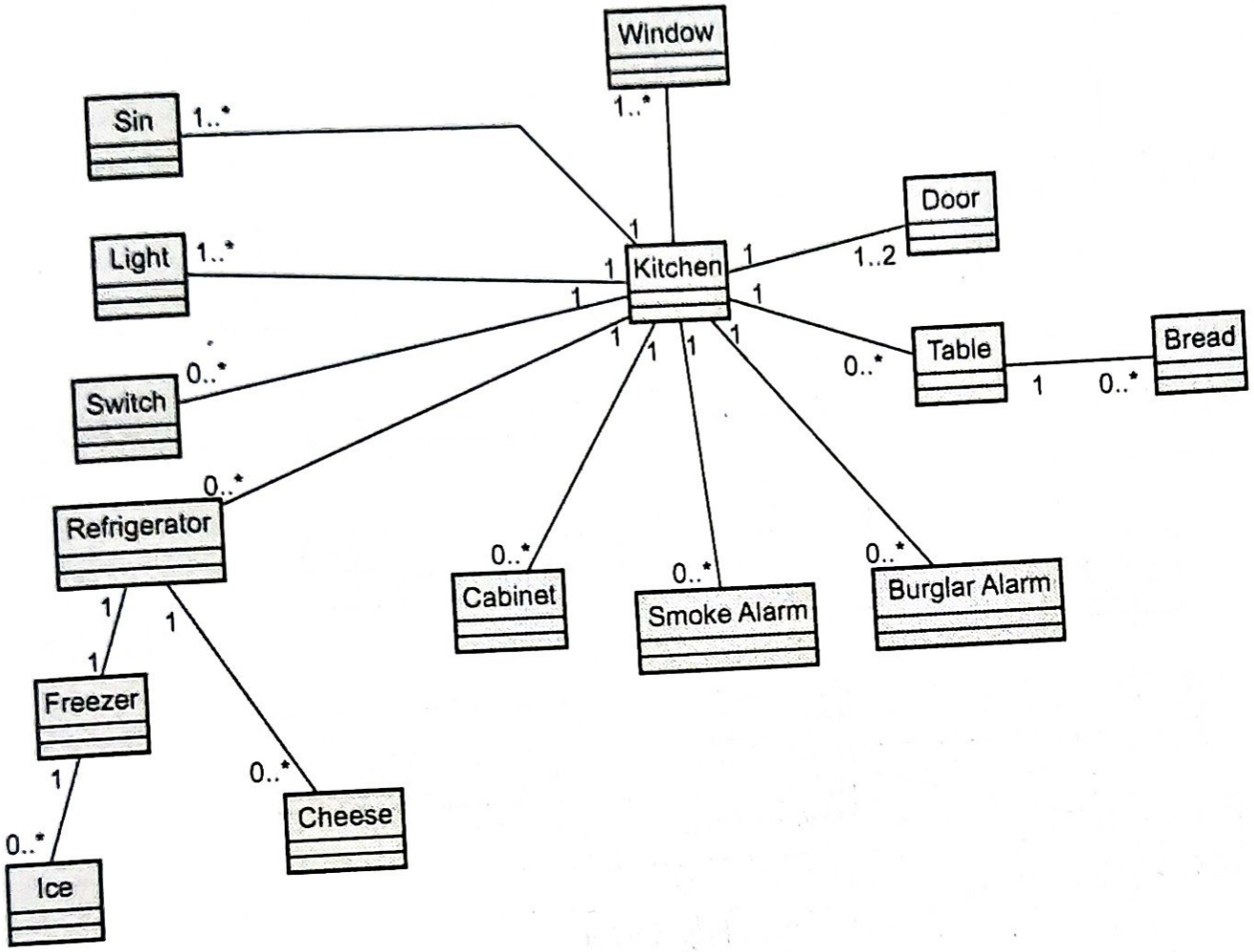


Fig. 1.10.3

Example 1.10.4 *Prepare a class diagram for following group of classes. Add at least 10 relationships (Associations and generalizations). Use association names and association end names where needed. Also use qualified associations and show multiplicity. Explain diagram.*

Classes :

School, playground, principal, school board, class room, book, student, teacher, cafeteria, rest room, computer, desk, chair, ruler, door, swing.

Solution :

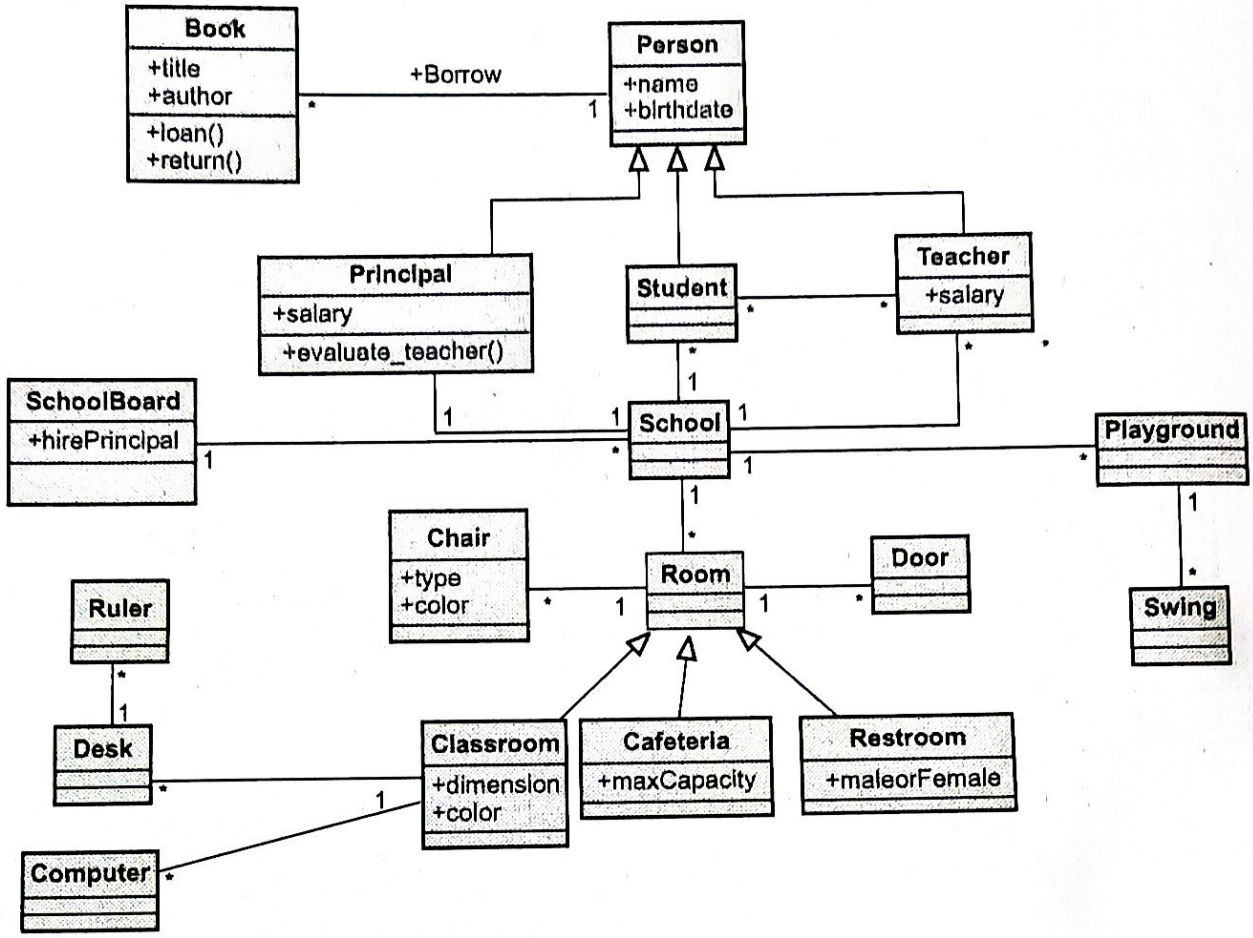


Fig. 1.10.4

Example 1.10.5 *Prepare a class diagram for the dining philosopher problem. There are 5 philosophers and 5 forks around a circular table. Each philosopher has access to 2 forks, one on either side. Each fork is shared by 2 philosophers. Each fork may be either on the table or in use by one philosopher. A philosopher must have 2 forks to eat.*

Solution :

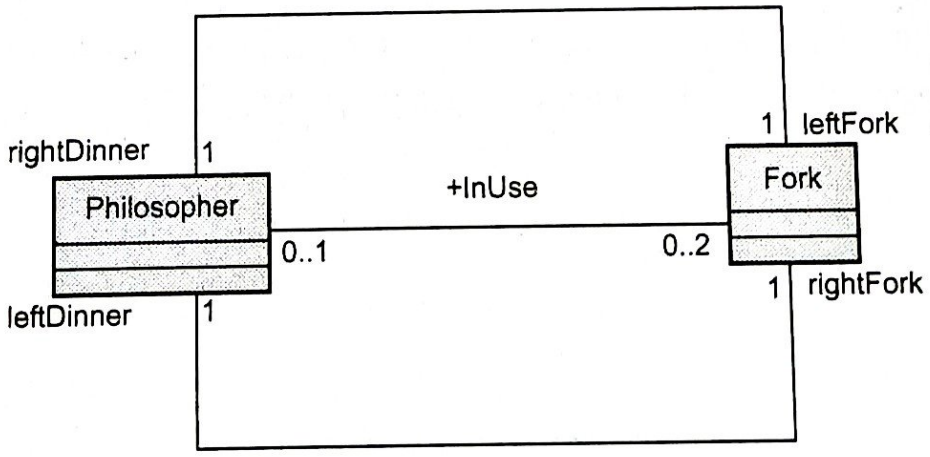


Fig. 1.10.5

Example 1.10.6 *Convert the following into a class diagram with appropriate classes, relationships, multiplicity.*

A computer program has many statements. An expression is a statement. A function is statement. An expression contains a variable, a constant and operator. A relational operator is an operator. An arithmetic operator is an operator. A function has an argument list, a return type.

SPPU : Aug.-15, In Sem, Marks 3

Solution :

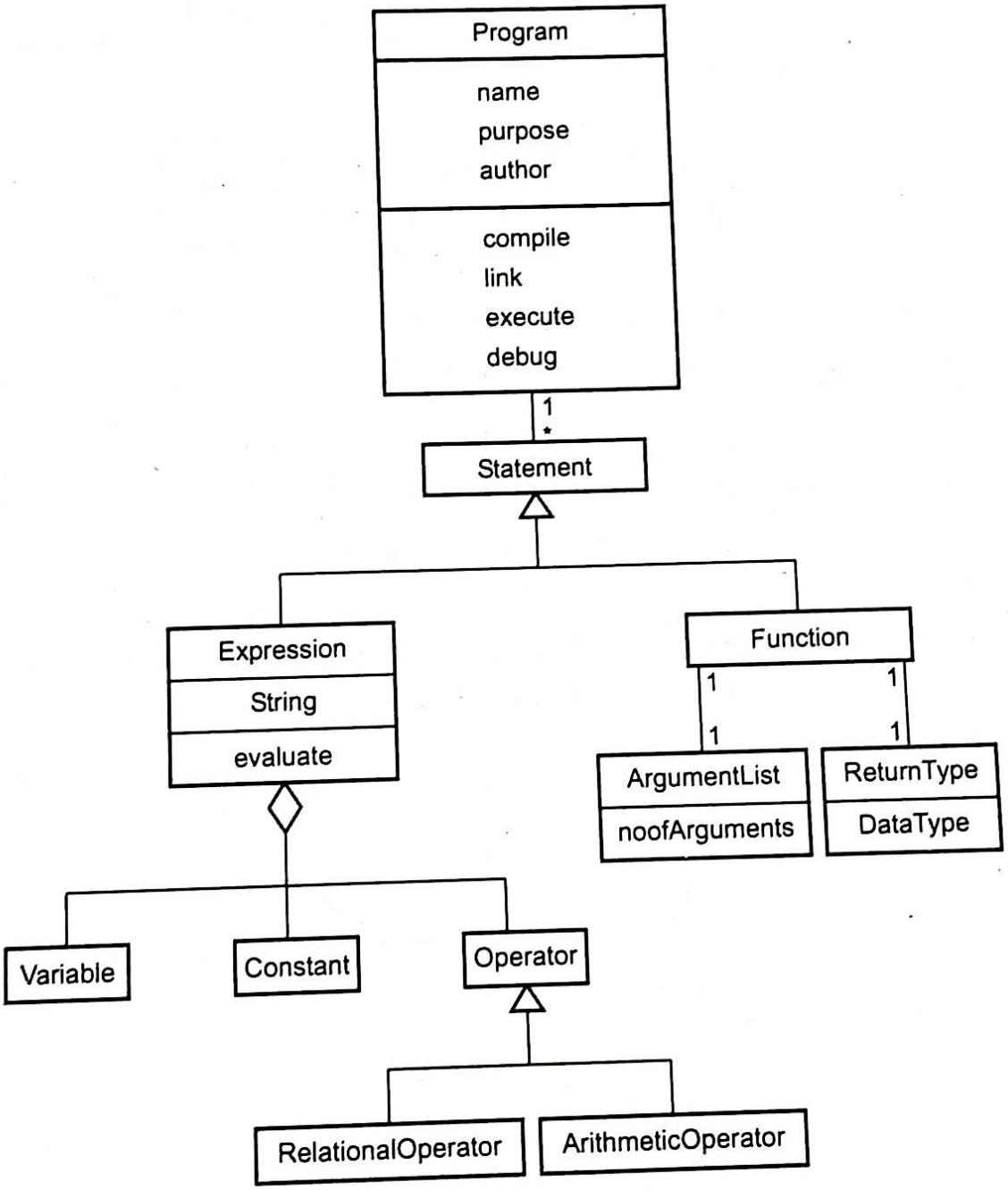


Fig. 1.10.6

Example 1.10.7 *Convert the following into a class diagram with appropriate classes, relationships, multiplicity.*

Car dealer sales cars. A car is owned by an owner. The owner has an address. The owner can be a person, a company or a bank. A car loan may be involved in the purchase of a car. Bank provides the loan.

SPPU : Aug.-15, In Sem, Marks 3

Solution :

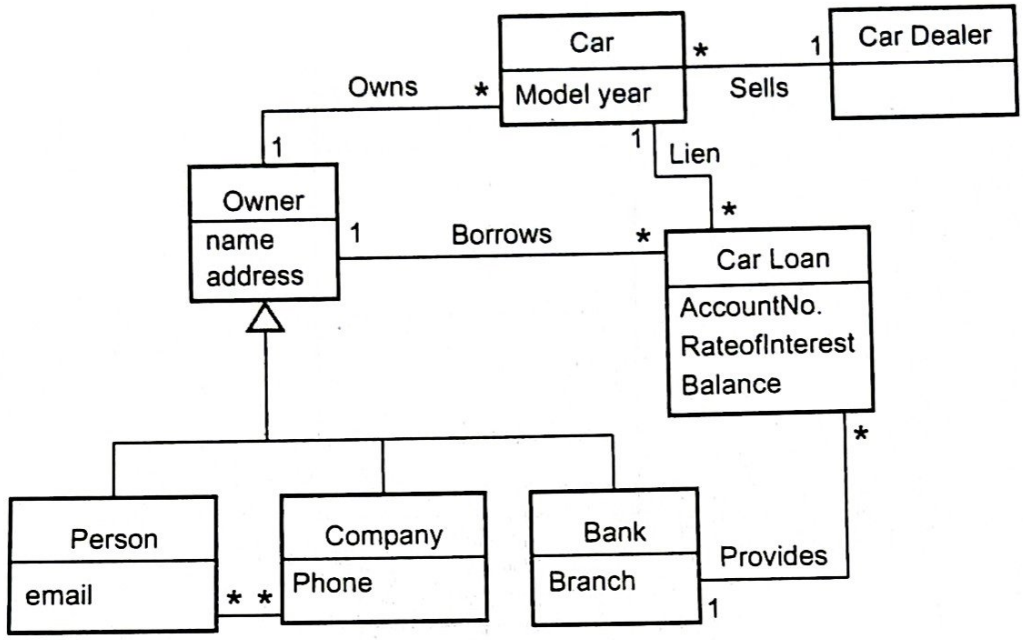


Fig. 1.10.7 Class diagram

Example 1.10.8 *A project has three to five students. A projects has a guide can guide one to three projects. For this description draw a class diagram. From the class diagram draw an object to show two projects, seven students and one guide. Do not write any explanation, just draw the diagrams.*

Solution :

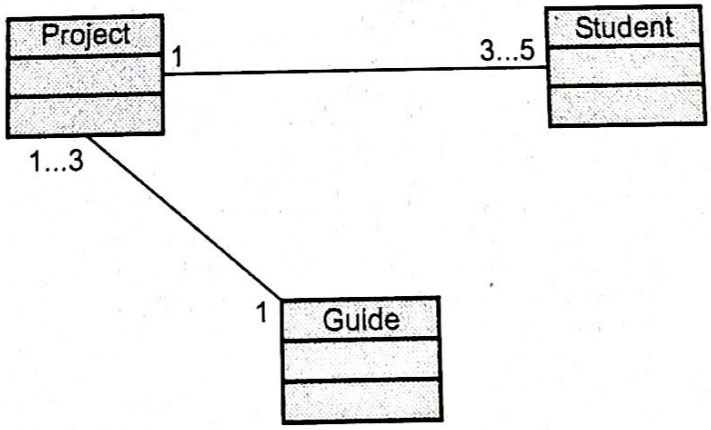


Fig. 1.10.8 Class diagram

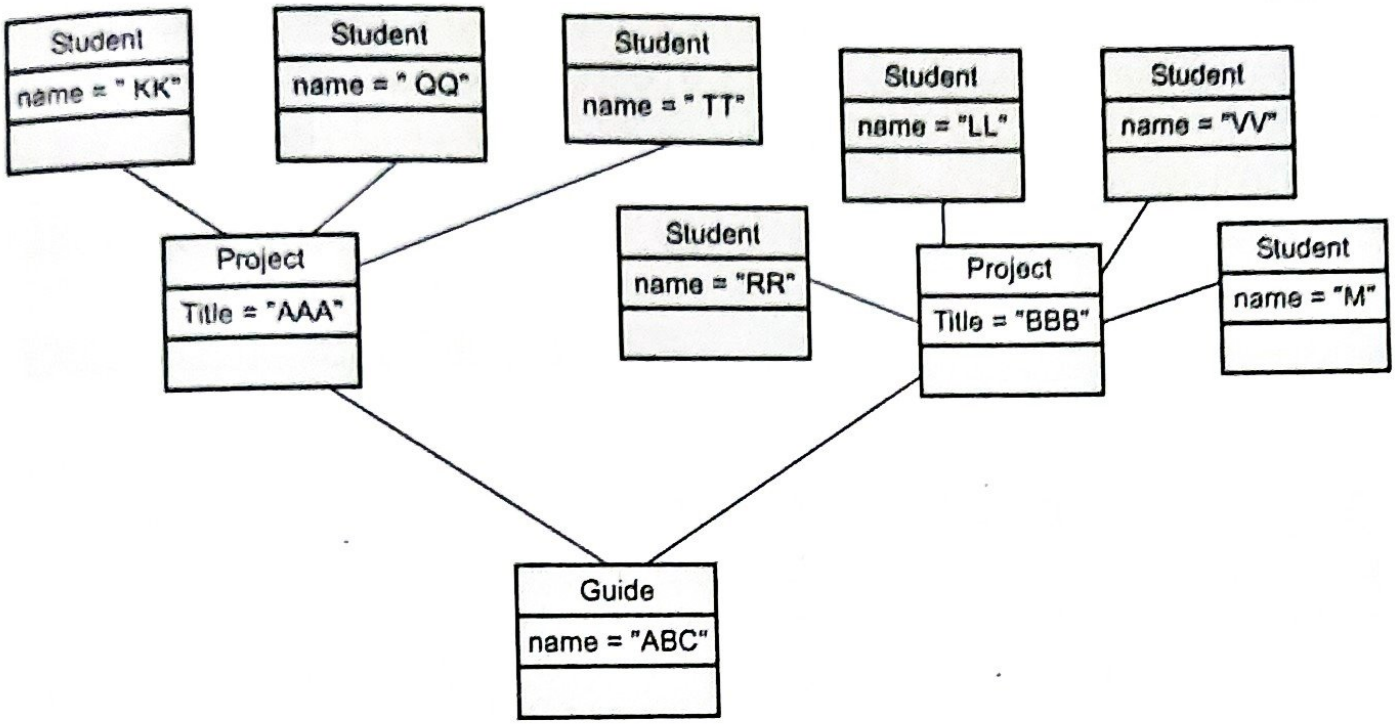
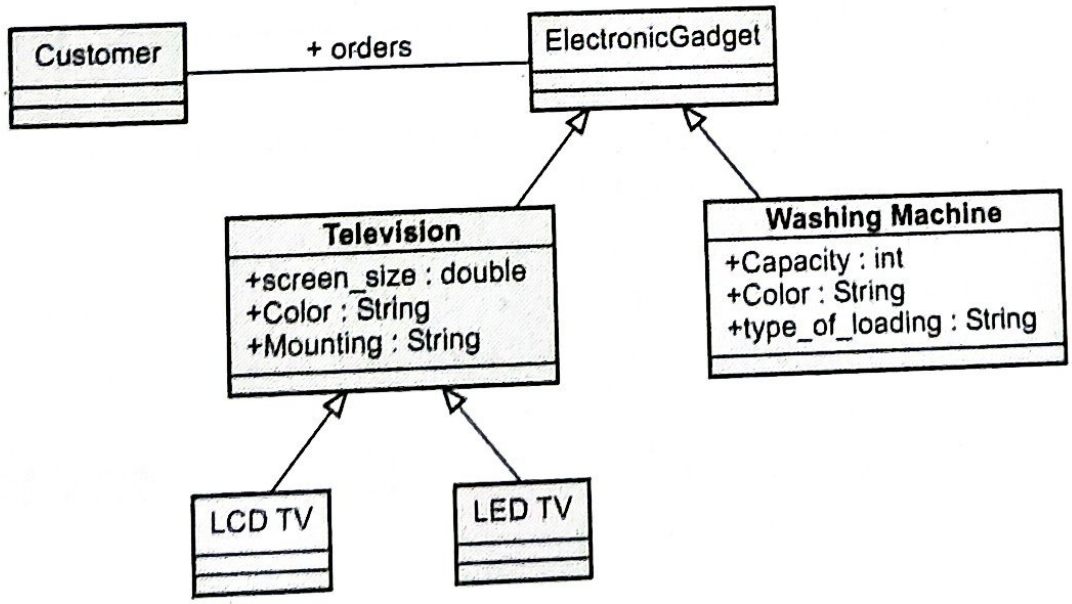


Fig. 1.10.9 Object diagram

Exapmle 1.10.9 *An electronic gadget shop has television and washing machines. Television has screen size inches, color, mounting(only wall or table mounts are possible). Televisions come in two types namely LCD and LED. A washing machine has a capacity in litres, color, type of loading(Top or front). A customer can order an electronic gadget. Draw a class diagram for this with attributes and relationships.*

Solution :

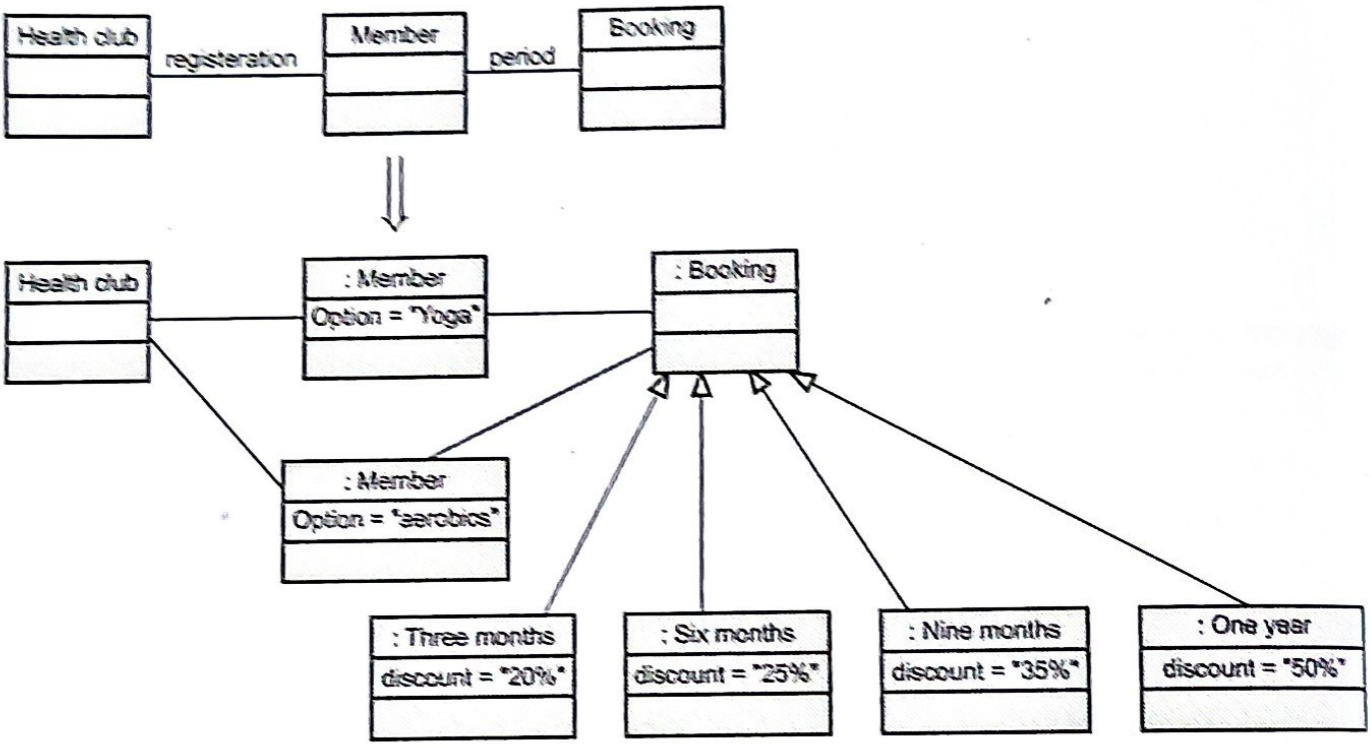


Example 1.10.10 *For the description given below, draw the class diagram*

Grand health club offers a scheme for membership of the health club. The options available for registering are 'yoga' and 'aerobics'. The monthly charges for aerobics membership are 2000.00. The monthly charges for yoga membership are 1000.00. The members can avail a single option out of the two options. If a person books for three months, he gets 20 % discount. If he books for six months, he gets 25 percent discount. If he books for nine months, he gets 35 % discount. If he books for one year, he gets 50 % discount.

SPPU : Dec.-15, End Sem, Marks 8

Solution :



Example 1.10.11 *For the description given below, draw the class dingram.*

When a book has isbn number, price, title and one or more authors. When it is bought in the library it gets 'purchased' state, when it is added to a catalogue it goes to 'catalogued' state. When the cataloguing is complete it goes to 'Available on stack' state. When it is borrowed by a member it goes to 'borrowed' state. When it is returned by a member it goes to 'available on stack' state.

SPPU : Dec.-15, End Sem, Marks 8

Solution :

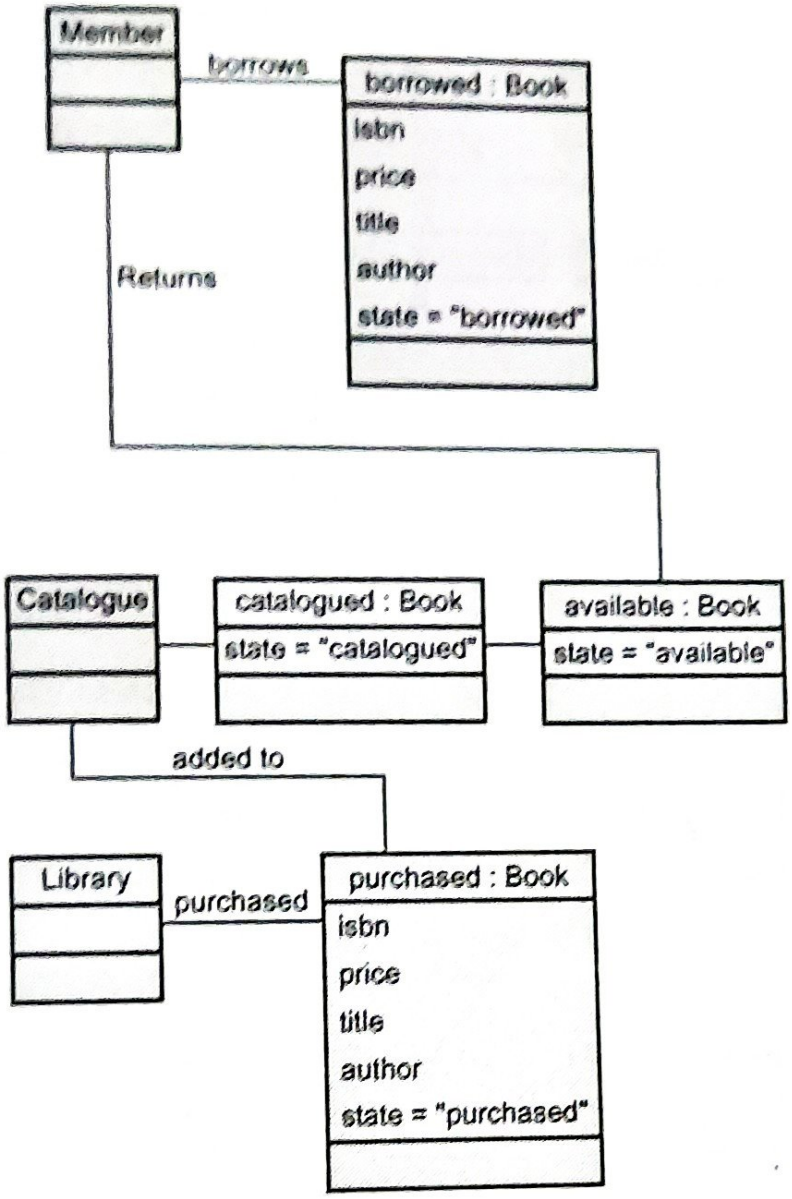


Fig. 1.10.10

Example 1.10.12 *Draw class diagram for following description -*

A drawing object is a text, a geometric object or a group. Group can contain at least two geometric objects. Point, polygon are geometrical objects. A polygon is composed of an ordered set of points.

SPPU : Aug.-15, In Sem, Marks 4

Solution :

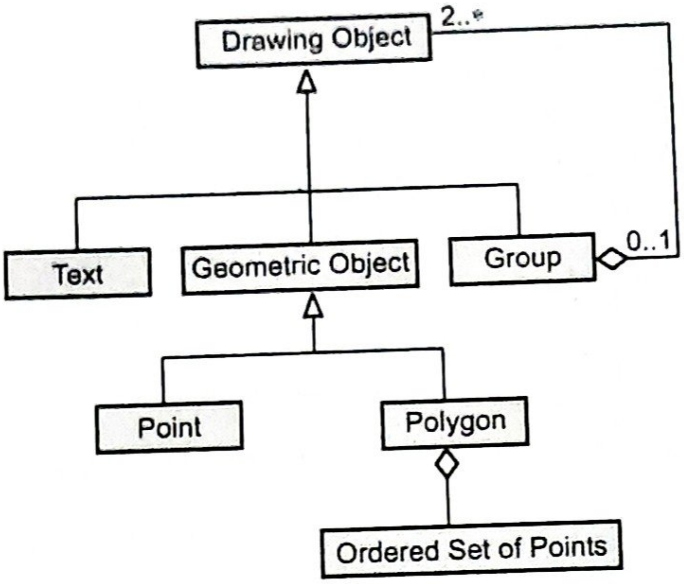


Fig. 1.10.11

Example 1.10.13 *Construct a class diagram using the following object diagram.*

SPPU : Aug.-15, In Sem, Marks 3

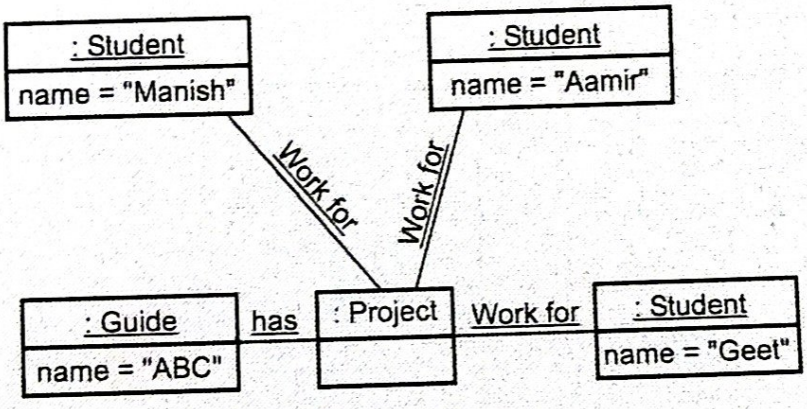


Fig. 1.10.12

Solultion :

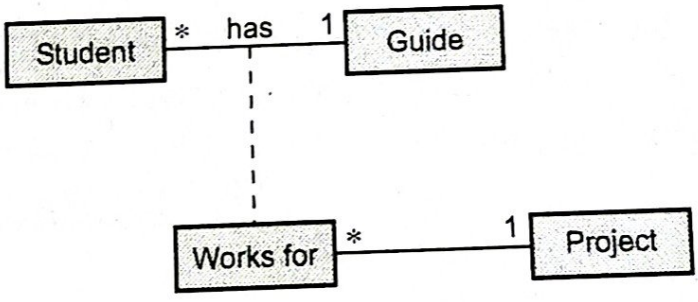


Fig. 1.10.12 (a)

Review Question

1. *Explain aggregation, composition and generalization with reference to class diagram.*

SPPU : April-17, May-17, Marks 6

1.11 Navigation of Class Models

- Class model is used to express the structure of an application. But sometimes class diagram also represents the behavior of navigations among the classes.
- For example - Consider an ATM system belonging to some bank and which handles the customer transactions at several places in a city. We can pose variety of questions while modeling for ATM system.
  - Whether the ATM card/pin is verified ?
  - How many attempts does a customer make to enter the pin at particular instance of time ?
  - How does system check whether the customer enters valid withdrawal amount or not ?
- The static class model is not sufficient to answer posed questions. There is a need for providing some logical statements along with this class model. These logical statements can be framed using a specialized language called **Object Constraint Language (OCL)**. Hence OCL can be defined as follows -
- Object Constraint **Language (OCL)** is a kind of language in which using formal language statements the constraints of the object can be represented.

1.11.1 OCL Constructs

The OCL can traverse the constructs in class models.

Attributes : One can traverse from object to an attribute value. The syntax is

objectName.attributeName

For example

aCustomerAccount.balance

Here *a CustomerAccount* is a object and *balance* is the attribute name.

Similarly the we can access attributes of each object in a collection. For example

To denote that *"every employee in the company must not be less than 20 years old"* we can write the OCL statement as follows -

```
Context Employee
inv: self.age>=20
```

Operations : We can invoke particular operation using the object of the class and suing the dot operator. OCL has special operations using which we can retrieve desired value from the class model. The operation can be invoked using -> operator. For example

```
aCustomer.accounts->select(balance>500)
```

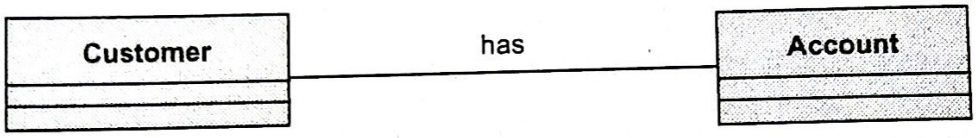
here **aCustomer** is an object and the attribute **accounts** is used. The **select** is a special operation defined by OCL which is useful for selecting specific elements that satisfy the given condition. By above state we will get the account of all those customer having > ₹ 500 balance in their account.

Simple Associations : OCL is useful in traversing the association. For example -

OCL Statement : aCusotmer.Account

Meaning - Yields the account number of the customers.

Class Model :



Qualified Associations : The qualified association helps you to find out particular object from the set of objects. For example

OCL Statement : Course.Student[10]

Meaning : To find out the entry for the student having the StudentID as 101 we can write the

Class Model :

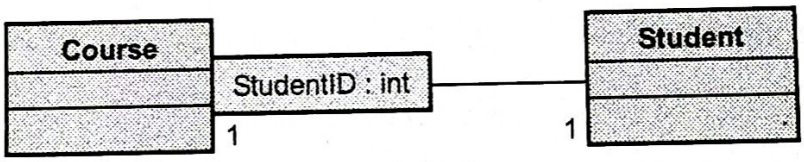


Fig. 1.11.1

Association Classes : The association class is a class that allows the association to a class itself. For example -

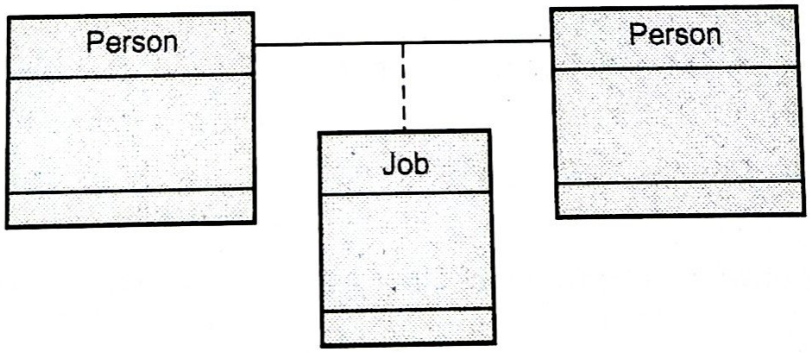
OCL Statement :

context Person

inv: self.isUnemployed = false implies self.job->size() >= 1

Meaning : A person has a job in a company.

Class Model



Generalizations : Traversal of generalization hierarchy is implicit for OCL notations.

Filters : Using the filters we can select the particular object with desirable property. Normally the **select** operation is useful for selecting the object with desired properties. For example - The customers whose bank balance is > ₹ 500 can be represented using following OCL statement -

```
aCustomer.accounts->select(balance>500)
```

1.11.2 Building OCL Expressions

- There are **four parts of OCL -**
  1. **Context :** It defines the situation for the design element.
  2. **Property :** It represents the characteristic of the element.
  3. **Operation :** Various arithmetical, logical or relational operations can be carried out of the design elements.
  4. **Keywords :** There are some reserved words in this language which are associated with the some meanings. For instance : if, else, not, or, implies.
- The **context** is usually the class to which the expression is attached.
- The OCL makes use of **Collection** class for describing the constraints. The subclasses of **collection** class are **self** and **sequence.**
- The **self** is the element of the UML diagram in context of which the OCL expression can be evaluated. Using the **dot operator** the **property or operations** can be invoked. For example, If the context is **Employee** then **self. Salary** denotes the value of attribute **salary** on the instance of **Employee.** That means **salary** of **Employee** is identified using the dot operator.
- The **invariant** is for describing the **constraints** by specifying the properties that must hold true for all the instances of the class
- **Syntax :**

```
context <classifier>
inv [<constraint name>]: <Boolean OCL expression>
```

- **Example :**

```
context Employee
inv: self.age>=20
```

This statement denotes that every employee in the company must not be less than 20 years old.

```
context Company
inv:StockRate()>5
```

This statement denotes that the stock price of company must not be less than 5.

- **Pre and Post Conditions :** The constraint assumed to be true **before** the execution of the operation. Then that constraint is called **precondition**. The constraint which is true after the execution of the operation is called **postcondition**.
- **Syntax of Precondition specification**

```
context <classifier> ::<operation>(<parameters>)
pre [<constraint name>]: <Boolean OCL expression>
```

- **Example :**

```
context Employee::Registration( i:integer )
pre: self.age>=18 and i>0
```

The meaning of this statement is that the precondition for the Employee for **Registration** operation is that his **age** must be greater than 18 years and value of integer **i** is greater than 0 .

- **Syntax of Postcondition specification**

```
context <classifier> ::<operation>(<parameters>)
post [<constraint name>]: <Boolean OCL expression>
```

- **Example :**

```
context Person::Authenticity()
post: self.isValid()=true
```

The meaning of this statement is that for the operation Authenticity() the postcondition is specified as the **Person** is authentic.

- The notation @ can be used to specify the value of variable at pre or post condition. For example **a@pre** means that the variable **a** existed prior to operation.

For example

```
context Customer::purchase(item)
pre: acctBal-item.price > 0
post: acctBal = acctBal@pre - item.price
```

The above code fragment denotes the **purchase** operation in which the actual balance before purchase must be more than the price of the item to be purchased. And post condition denotes that the actual balance must be equal to the actual balance at the precondition time minus the item's price.

1.11.3 Operations on Collection

Operation	Description
size	The number of elements in the collection.
count(object)	The number of occurrences of objects in the collection.
isEmpty( )	The Boolean operation, True when the collection is empty.
notEmpty( )	The Boolean operation, True when the collection is not empty.
includes(object)	If the object is an element of the collection then it returns true.
includesAll(object)	If all elements of the collection are present in the current collection.
sum(collection)	The addition of all the elements of the collection elements.
exists( )	There must be at least one element present in the collection.
forAll(expression)	The expression is true for all the elements.
select(expression)	This operation return particular element that satisfies the expression.
reject(expression)	This operation return particular element that does not satisfies the expression.
SortedBy(expression)	Returns a sequence of all the elements in the collection in the order specified.

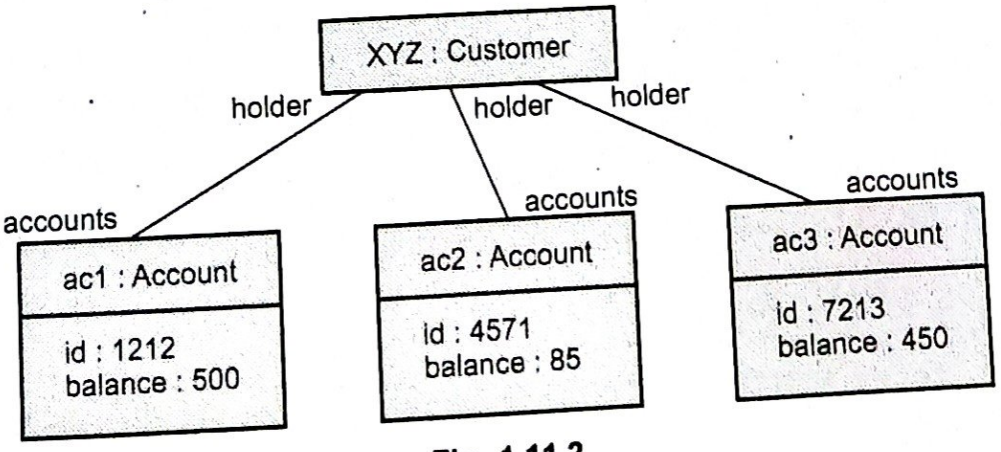


Fig. 1.11.2

The OCL for above UML diagram is

$$XYZ \cdot accounts \rightarrow forAll\ (a : Account \mid a \cdot balance > 0)$$
$$= true$$

XYZ. accounts→ select (balance > 300) = {ac1, ac3}
XYZ. accounts · balance →sum()=1035
XYZ.accounts · includes (ac4) = false

Review Question

1. *Write a short note on navigation in classes.*

❑❑❑

Unit II

2 Advanced Class Modeling and State Modeling

Syllabus

Advanced object and class concepts; Association ends; N-ary associations; Aggregation; Abstract classes; Multiple inheritance; Metadata; Reification; Constraints; Derived data; Packages; Practical tips.

State Modeling : Events, States, Transitions and Conditions; State diagrams; State diagram behavior; Practical tips.

Contents

Part I : Advanced Class Modeling

2.1 Advanced Object and Class Concepts

2.2 Association Ends

2.3 N-ary Associations

2.4 Aggregation

2.5 Abstract Classes

2.6 Multiple Inheritance

2.7 Metadata

2.8 Reification

2.9 Constraints

2.10 Derived Data

2.11 Packages

Part II : State Modeling

2.12 Introduction

2.13 Events

2.14 States

2.15 Transitions and Conditions

2.16 State Diagrams

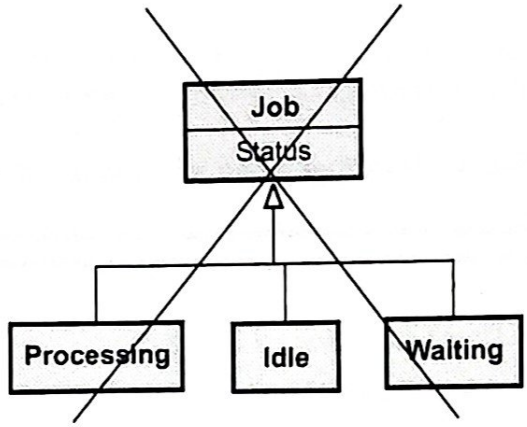
2.17 State Diagram Behavior ..... **Aug.-15, Dec.-15, April-16, 18,** ..... Marks 5

Part I : Advanced Class Modeling

2.1 Advanced Object and Class Concepts

2.1.1 Enumerations

The enumeration is a primitive data type which can have finite set of values and can be modelled as classes. For example - Job can have the status such as Processing, Idle or Waiting. These values can be represented by designing **Status** as class of type <<enumeration>>. Note that in the following figure, the keyword **enumeration** is within Guillemets i.e. << >>. The list of values are given in the second compartment.



Wrong method

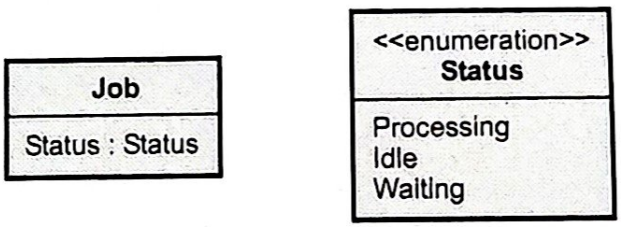
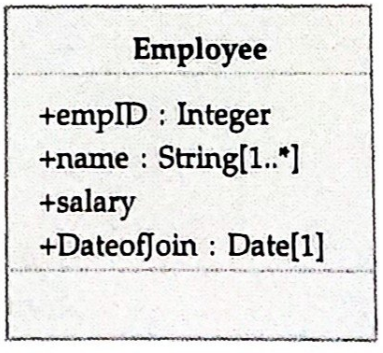


Fig. 2.1.1 Modeling enumeration

2.1.2 Multiplicity

Multiplicity denotes how many objects are associated with particular object. Multiplicity can be associated with the associations as well as attributes.Normally in databases the multiplicity is associated with the attributes. For example -



2.1.3 Scope

Scope represents whether particular feature is available for class or object. In UML the scope can be **static scope** or and **object** or **instance scope.**

1. **Object scope** - Each object of a class has its own value. It requires no notation for specifying the instance.
2. **Static scope** - All the instances of a class have just one value. This is referred as static scope. The static scope is mentioned by underlining the attribute.

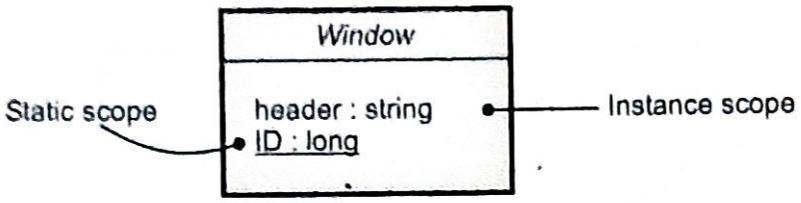


Fig. 2.1.2 Instance and scope

The private attributes of class makes use of static scoped features.

2.1.4 Visibility

The visibility specifies the how the attributes and operations are visible in the system. There are four levels of visibility -

1. **public** : The public element is visible to all elements that can access the classifier. It is denoted by '+' symbol.
2. **protected** : The protected element is visible to all the elements that have a generalization relationship to the namespace that owns it. **Protected** visibility is represented by '#' symbol.
3. **private** : A private element is only visible inside the namespace that owns it. Private visibility is represented by '-' symbol.
4. **package** : Only the elements that are declared in the same package can use this feature. It is specified using the symbol '~'.

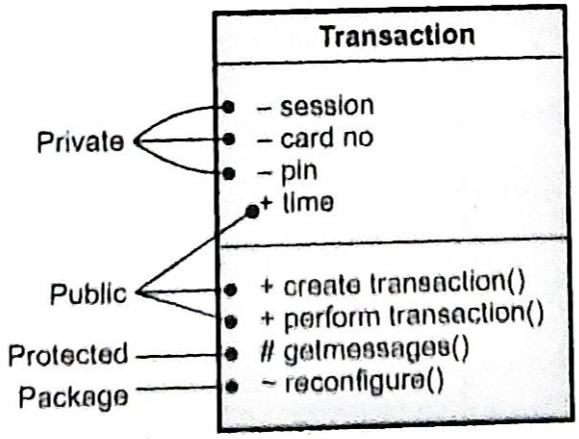
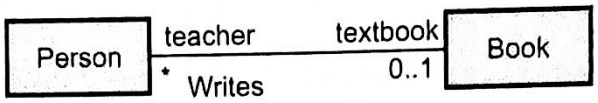


Fig. 2.1.3 Visibility

2.2 Association Ends

- Association end is an end of association.
- Binary association has two ends, ternary association has three ends and so on.
- **Properties of association end** - There are various properties of association end and these are enlisted below -
  - **Association End Name :** For clear understanding of the association relationship we can write the names at the association. The traversing direction can be understood due to association end names.



Teacher	textbook
Rita	EVS
Smita	Mathematics

Fig. 2.2.1 Association end names

  - **Multiplicity :** Multiple objects can be related to one particular object. It can be denoted by "1", "*", "1...*" and so on.
  - **Ordering :** This is a type of association which is used to denote the set of objects at one end must appear in specific order.

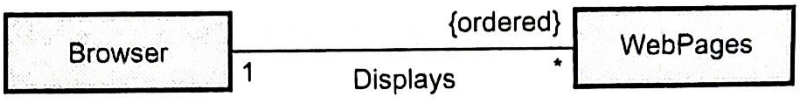


Fig. 2.2.2 Ordering the objects

  - **Qualification**

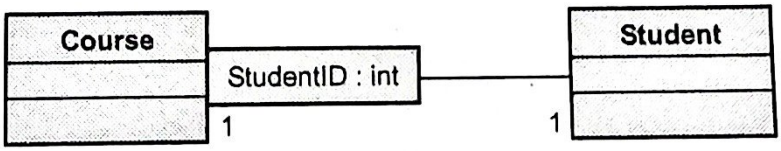
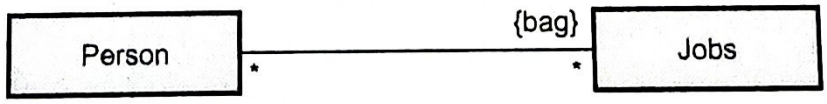
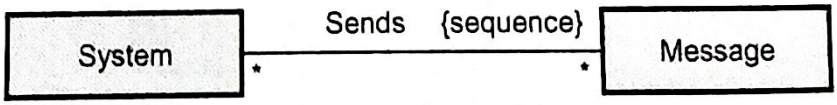


Fig. 2.2.3

  - **Bags and Sequences :** In association relationship, there are some objects that may be duplicated or may appear in some sequence. For that purpose bags and sequences are used.





- **Aggregation :** It is a kind of association in which one object is a part of another object.
- **Changeability :** The update status of an association end is denoted by this property. The values for this property can be *changeable* and *readonly*.
- **Navigability :** By default the association can be traversed in both the directions, however by showing the arrowhead at one end we can represent the navigability from one class to other.
- **Visibility :** The association ends can be public, private protected or package.

2.3 N-ary Associations

The n-ary association is the association among three or more classes. Normally having n-ary association in the class diagram is not preferred and therefore it is a practice to split n-ary association into binary association.

Following is a simple association that can be seen as n-ary association but it can be represented using binary association.

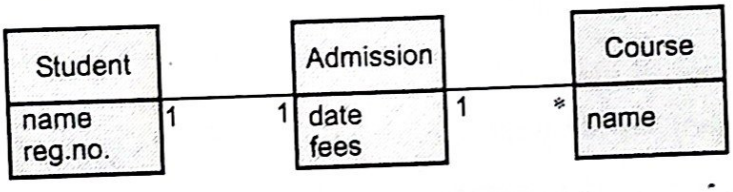


Fig. 2.3.1 n-ary association represented by binary association

Following is a typical example of n-ary association in which three classes take part actively. Note that such an association is represented using a diamond notation. The course management system can maintain the records for the students who are studying the courses in particular academic year.

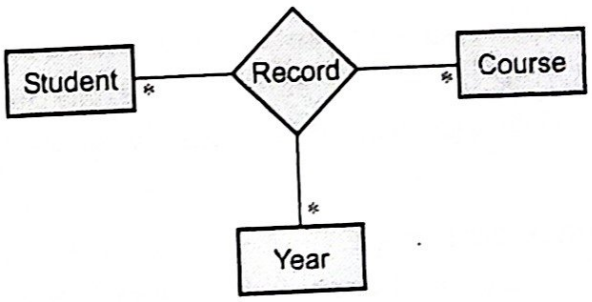


Fig. 2.3.2 Ternary association

Following is an another representation in which four classes are taking part making the association.

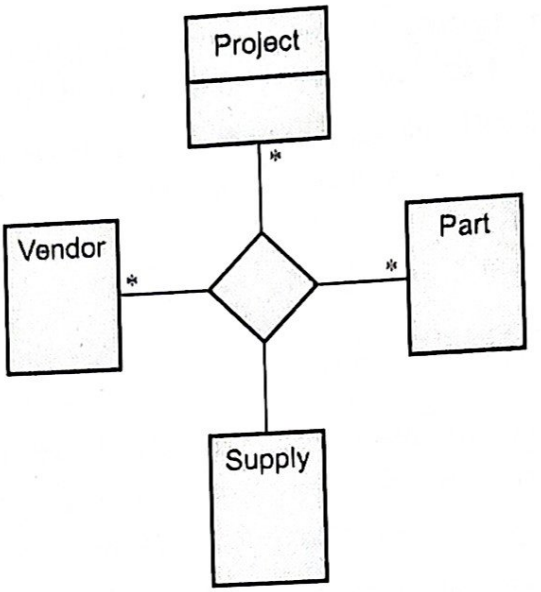


Fig. 2.3.3

2.4 Aggregation

- Aggregation is a type of association.
- It is used to represent **whole-part** relationship.
- It normally posses **has-a** relationship.
- It is denoted as follows -

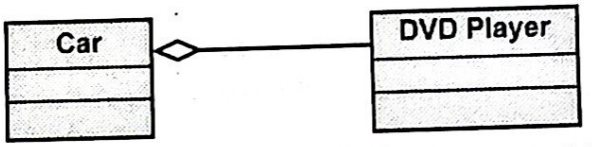


Fig. 2.4.1 Example - Aggregate relationship

- **For example -** DVD player and Car are the two classes that can be associated by aggregate relationship. Note that DVD player can exists without car and car can be without a DVD player. We can read this relationship as "DVD player is a part of Car".
- **Transitivity -** It is an important property of aggregation. It means of class X is a part of class Y and if class Y is a part of class Z then class X becomes the part of class Z.
- **Antisymmetric -** According to this property of aggregation if class X is a part of class Y then class Y is not a part of class X.

Example 2.4.1 *Prepare a class model to describe geographical map. Map contains roads, rivers and mountains. All components are described by points representing longitude and latitude.*

Solution :

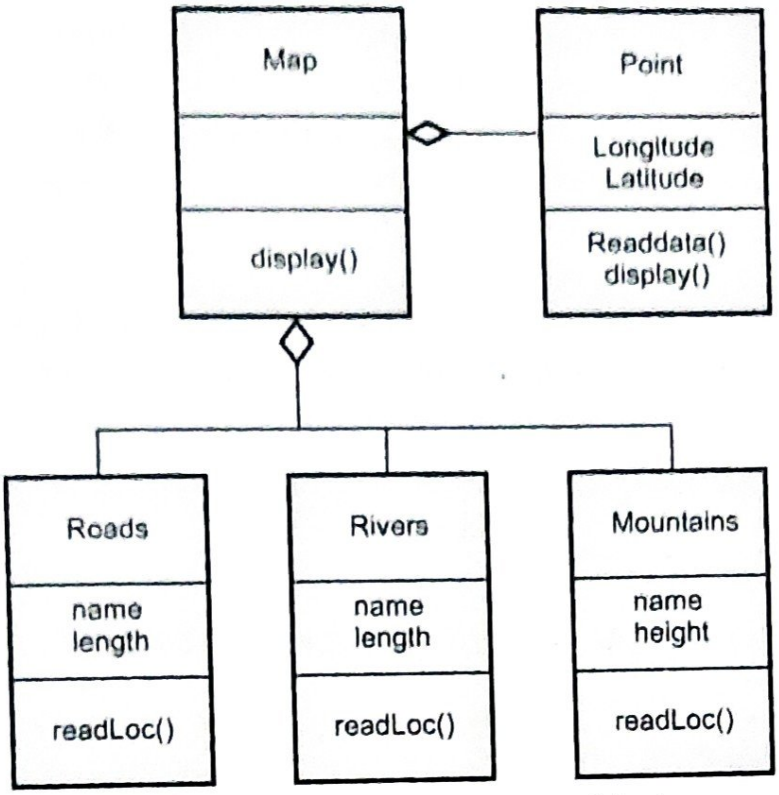


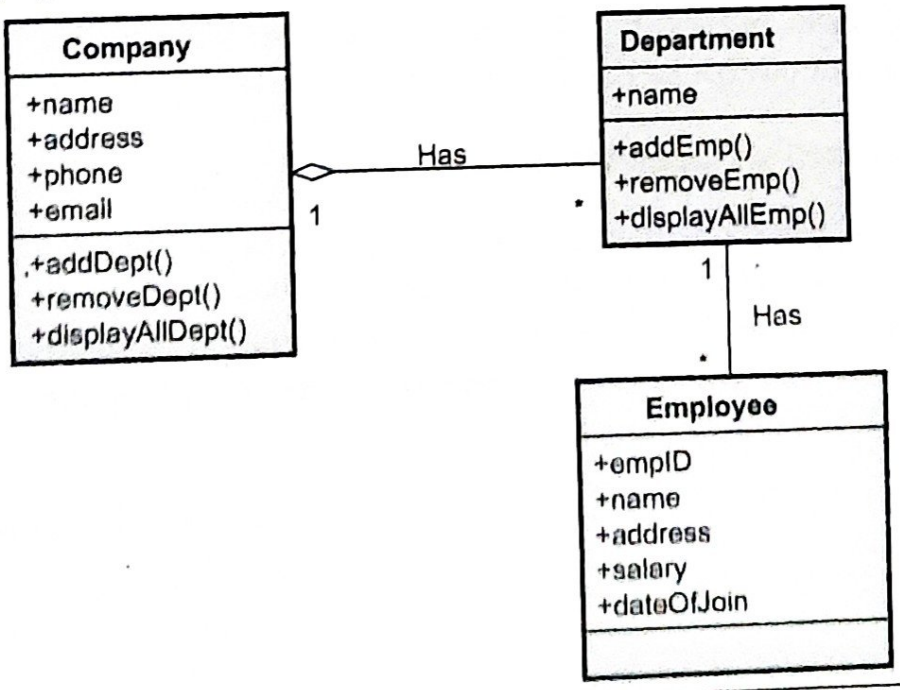
Fig. 2.4.2 Class model for geographical map

2.4.1 Difference between Aggregation and Association

The aggregation is a special kind of association.

In aggregation relationship the two objects share a **whole-part** relationship. But if two objects are independent but still there are related to each other then the association relationship is indicated.

The use of aggregation instead of association often comes by judgement of designer.



2.4.2 Difference between Aggregation and Composition

Aggregations and Composition are two forms of **whole-part** relationship. The composition is restricted aggregation. That is, when one object contains another object and if contained object can not exist without the existence of the container object, then it is called composition. It is denoted as follows -

For example - Steering and Car are the two classes that are associated by the composite relationship. This is restricted relationship because a car can not exist without the steering.

Fig. 2.4.3 Example - Composite relationship

We can read the relationship as "Steering is a part of Car". The arrowhead is pointing towards the **whole** class.

Difference between aggregation and composition

Both the composition and aggregation represent the whole part relationship but the composition is more restricted than the aggregation. The composed object can not exist without the other object. Hence it is a strong relationship.

This restriction is not there in aggregation. The existence of contained object is entirely optional in case of aggregation.

For example - Books Library contains books and students. The student and Library share the aggregate relationship but the Book and Library share the composition relationship. As without books the library is not possible.

2.4.3 Propagation of Operations

Propagation of operations is a mechanism which allows to move particular operation through the network of objects. For example - Following Fig. 2.4.4 shows that the Canvas can have some Graphical image. The operation animates is used by this graphical image which propagates to Shapes in the Graphics then to Pixels of the shapes. This is also called as triggering.

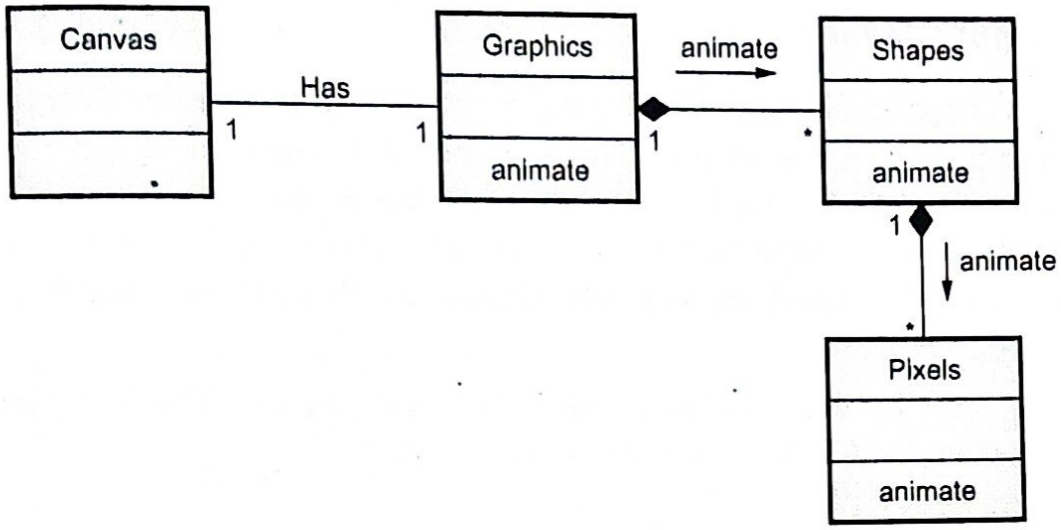
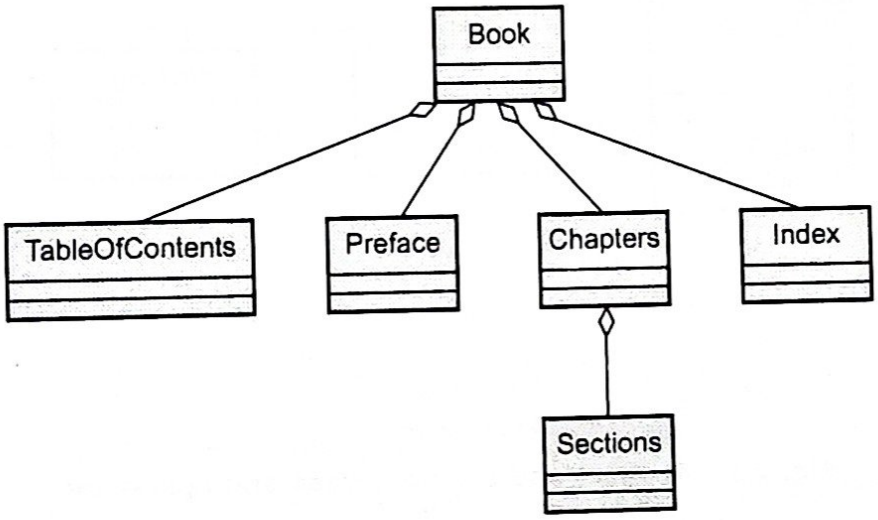


Fig. 2.4.4 Propagation

Example 2.4.2 *A book is composed of Table of Contents, Preface, many chapters and index. Chapters are composed of sections. Convert this using a class diagram.*

Solution :



Review Questions

1. *Define the purpose of the following term with suitable example and UML notation with respect to class model - Aggregation.*
2. *Explain aggregation. Is it same as association ?*
3. *What is aggregation ? Differentiate the following with example.*
   1. *Aggregation versus composition.*
   2. *Aggregation versus association.*

2.5 Abstract Classes

When the generalization is modeled the class at the **top level** is an abstract class and the more specific classes are at the bottom. Hence it is common to have the abstract classes at the top of the hierarchy. The **abstract classes** are those classes that do not have the direct instances. Note that the abstract classes have the names that are denoted in italics. On the other hand the **concrete classes** are those classes which have direct instances.

The classes that have no children are called **leaf classes**. The leaf class can be specified by writing {leaf} below the name of the class.

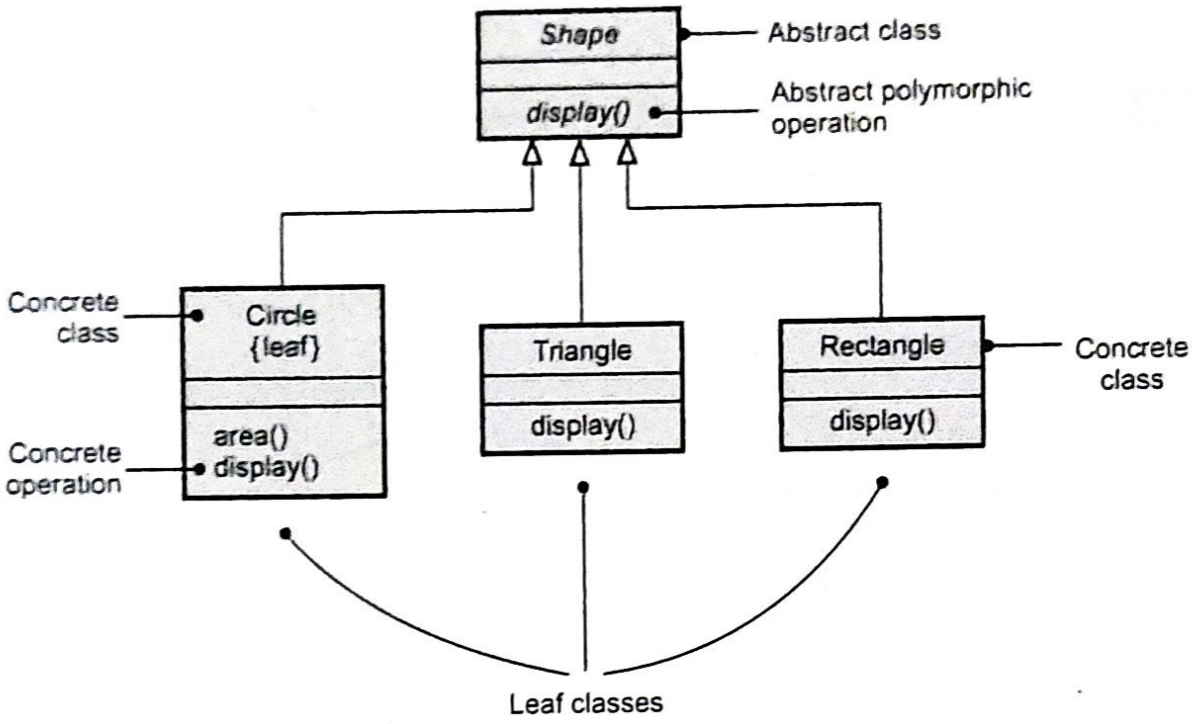


Fig. 2.5.1 Abstract and concrete class and operations

In above class diagram, the polymorphic operations are specified. The polymorphic operations are the operations that have the same signature but can be used for different purposes. The abstract polymorphic function is incomplete and is overridden by the child class functions.

2.6 Multiple Inheritance

Definition : The multiple inheritance is a kind of inheritance in which the derived classes are derived from more than one super classes. The class with more than one superclass is called **join class**.

This kind of implementation may lead to some conflicts in the child classes due to multiple paths from superclass. But such ambiguities can be eliminated during the implementation.

Example

If some parent has no more than one child in multiple inheritance then that class is disjoint class.

The hollow triangle indicates the disjoint subclasses whereas the solid triangle indicates the overlapping subclasses.

The Fig. 2.6.1 shows disjoint class.

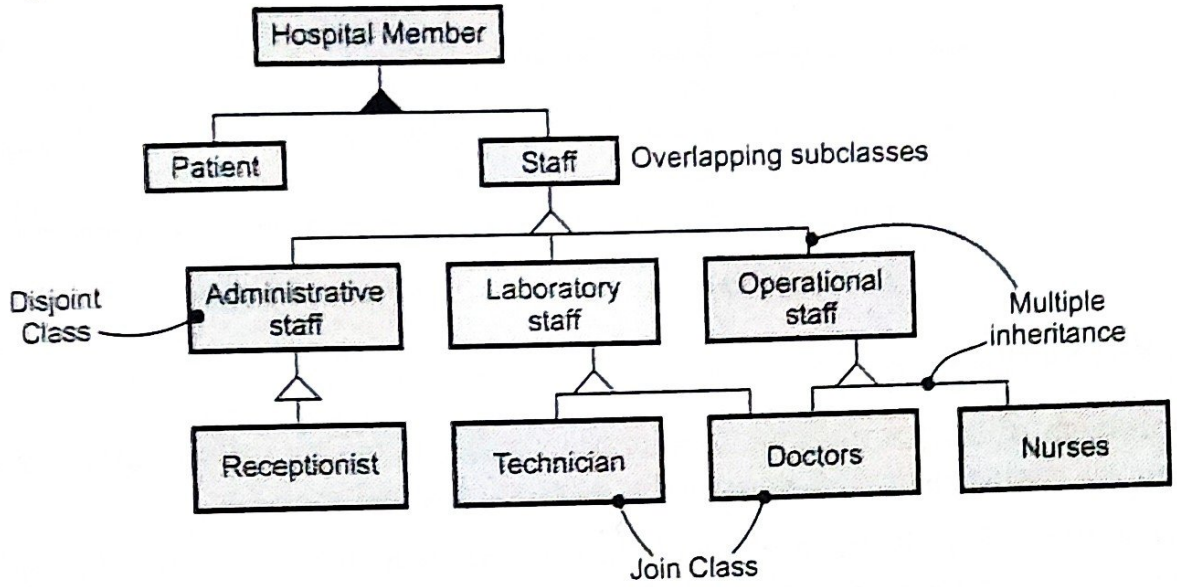


Fig. 2.6.1 Multiple inheritance from overlapping classes

Advantages :

1. The single classes can perform **multiple functionality.**
2. The **reusability** of the code gets increased.

Disadvantages :

1. The conceptual and implementation model becomes **complex.**
2. In multiple inheritance - There is a **mixing of rules and multiple paths** can be followed for reaching to derived classes, conflicts might arise in the implementations.

2.6.1 Accidental Multiple Inheritance

Accidental multiple inheritance be is a kind of inheritance in which one instance happens to participate in two overlapping classes.

An instance of a join class is inherently an instance of all the ancestors of the join class. For example doctor can be both the laboratory staff as well as operational staff, but one doctor can be attached to more than one hospital at the same time. There is no class that describes this combination. The object oriented language handles this issue by making the class **Person** as an object composed of multiple objects of class **Hospital Member**. See Fig. 2.6.2.

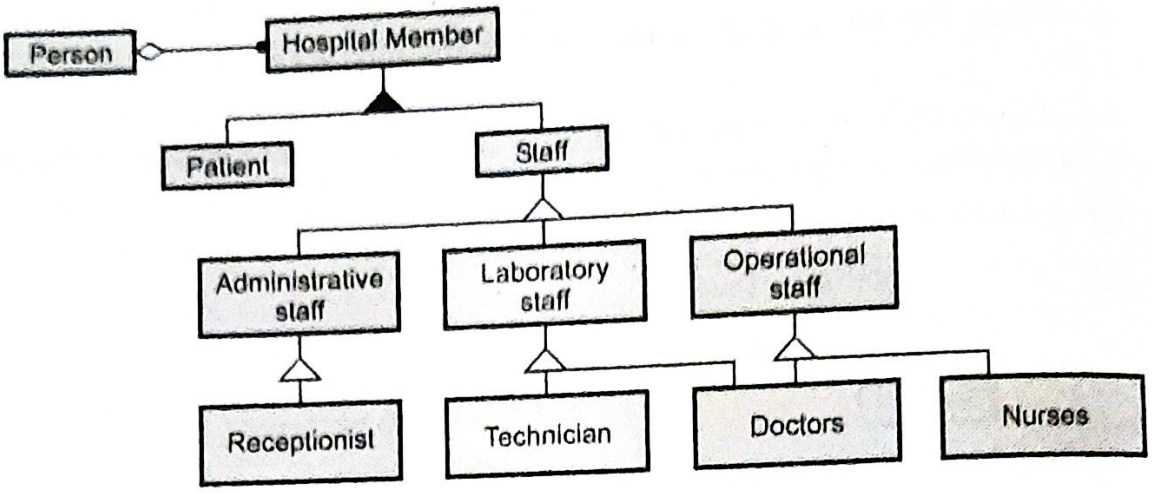


Fig. 2.6.2 Workaround for accidental multiple inheritance

2.6.2 Workarounds

The problem of lack of multiple inheritance can be solved by restructuring the model. Following are the workaround possible for restructuring the model -

1. **Delegation :** This is an **implementation mechanism** in which object forwards one operation to another object for execution. The advantage of delegation is that only meaningful operations are delegated and there is no danger of inheriting meaningless operations.
2. **Delegation using aggregation of roles :** A superclass with independent inheritance(generalization) can be replaced by aggregation. In this each component replaces a generalization. For example

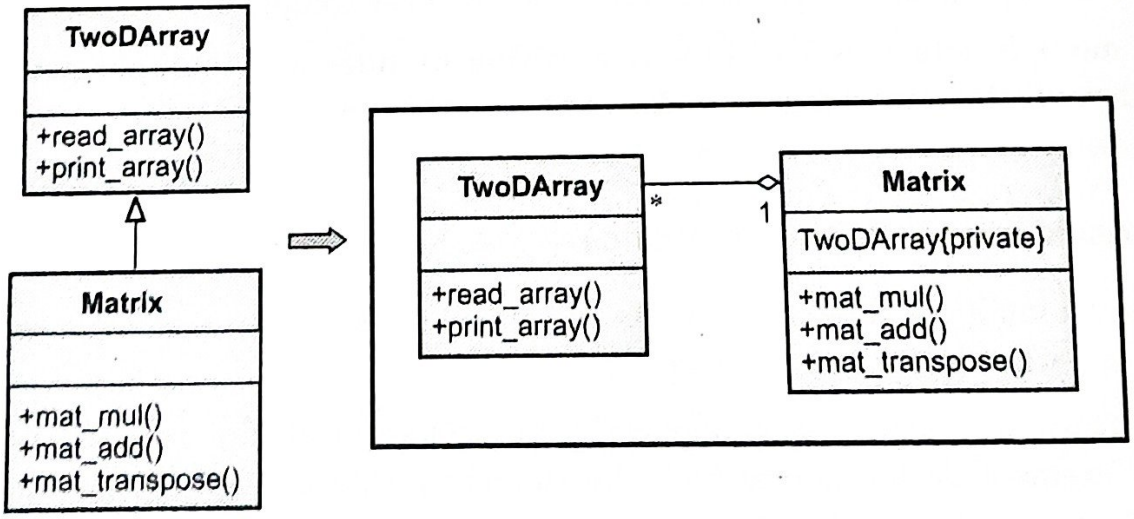


Fig. 2.6.3 Alternative design - Conversion from inheritance to aggregation

3. **Inherit most important class :** The join class is treated as an aggregation of the remaining superclass.

4. **Nested generalization :** In this technique, one generalization is factored then second, then third and so on. Thus all possible combinations of generalizations are factored. This preserves the inheritance but it duplicates the declarations and the code. The principles of object oriented techniques are also violated by this approach.

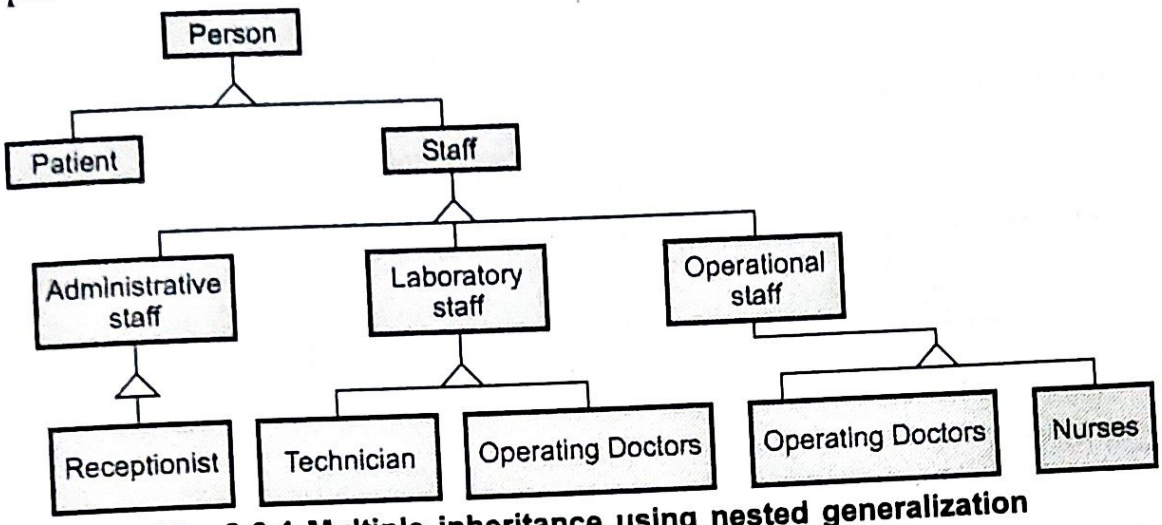


Fig. 2.6.4 Multiple inheritance using nested generalization

Issues in Workaround

Following are some issues while selecting the workaround -

1. If a subclass have several superclasses and all the superclasses are equally important then **make use of delegation** and **bring the symmetry** in the model.
2. If **one superclass** is more **important** than rest of the superclasses then implement **single inheritance** and **apply delegation.**
3. **Avoid nested generalization** if **large amount** of code gets duplicated.
4. If there are **less** number of **combinations** then consider **nested generalization.**
5. In the nested generalization, **factor most important criterion** first and then next most important and so on.
6. If one supeclass is having more features than other superclasses then **preserve** the **inheritance** through the path.
7. Maintain **strict identity** of the model.

Review Questions

1. *Discuss in brief multiple inheritance.*
2. *What do you understand by multiple inheritance ? Describe with suitable example.*
3. *What is multiple inheritance and what are its associated problems ? How does the concept of inheritance of specifications help to overcome these problems ? Explain.*
4. *Define generalization. Draw UML diagram that shows multilevel inheritance.*

2.7 Metadata

Metadata is a data that describes other data. The class definition is basically a metadata. In UML many times it is preferred to model the metadata as a separate class.

For example - Consider a banking system, in which the records for the customer and their accounts is maintained. A separate description class AccountDescription is maintained which describes the account number.

Many times classes have their own classes. These classes are called as metaclasses.

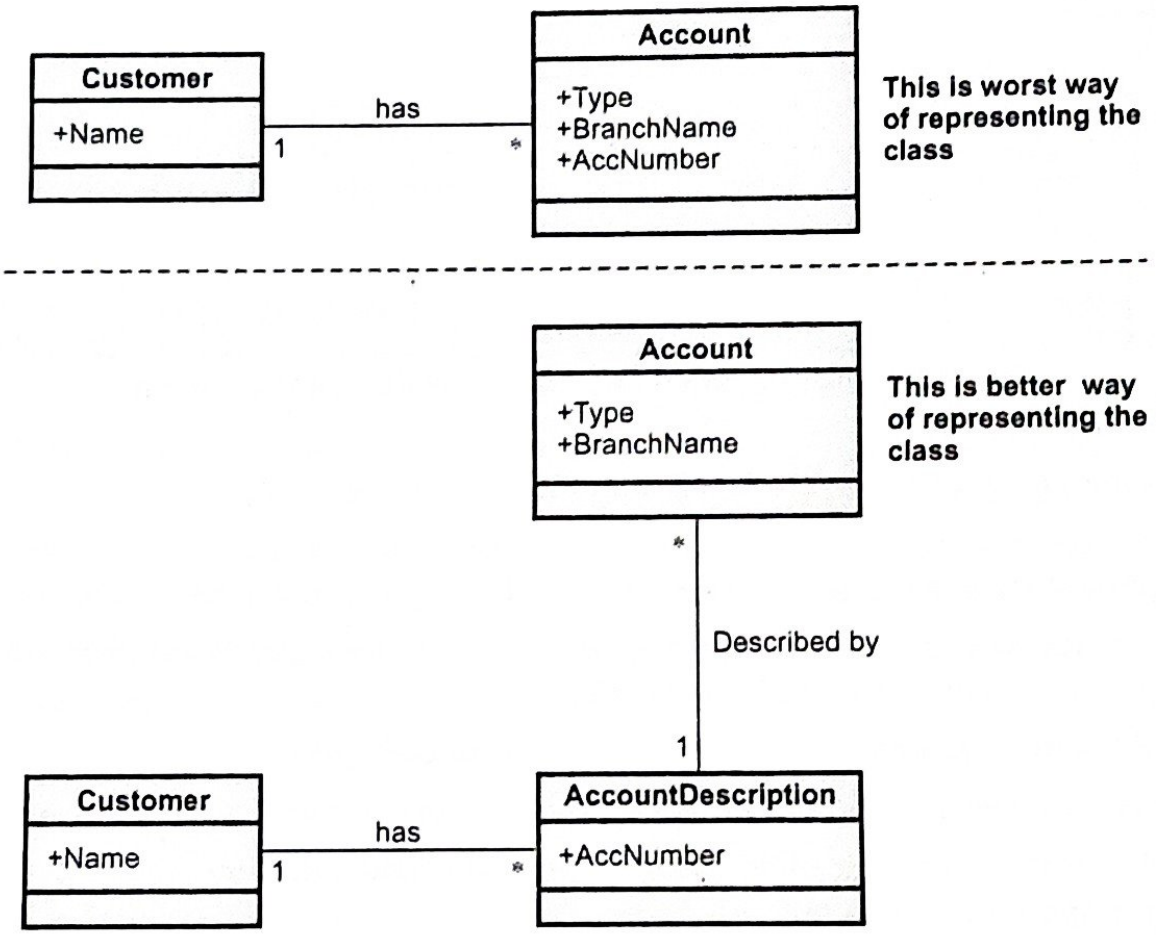


Fig. 2.7.1 Metadata

Review Question

1. *Define the purpose of the following term with suitable example and UML notation with respect to class model - Derived Data.*

2.8 Reification

Definition : Reification is the process of converting an element which is not an object **into object** within a scope of particular system.

By means of reification something that was preciously implicit, unexpressed can be explicitly formulated and is made available for conceptual manipulation.

For example consider an example of some software application program. The developers write a code and develop the software application which is meant for providing some data services to its user. Thus a every software application can have several aliases based on the service it provides. Instead of having **dataServiceName** as an attribute of the class **SoftwareApplication** we can promote this attribute to a class named **DataServiceName.** That means the reification takes place for converting a **dataServiceName** (which is actually an attribute and not an object) to **DataServiceName** (a class whose instance can be used for data handling).

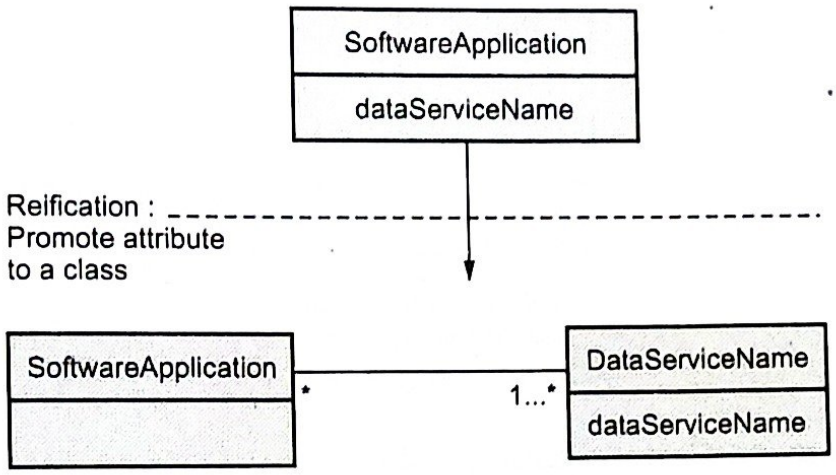


Fig. 2.8.1 Reification

2.9 Constraints

Constraints are commonly used in class diagram. In UML constraint is a condition or restriction on UML element. These elements can be objects, classes, attributes, links, associations and generalization sets. It must evaluate to Boolean value - true or false. Constraint must be satisfied by a correct design of the system. OCL i.e. Object Constraint Language is a constraint language predefined in UML. However, constraint can be specified by some other languages such as Java, Machine readable languages and natural languages.

2.9.1 Constraints on Object

Following is a simple example that shows how the constraint can be applied on the object. Normally, the constraint can be given in the form of expression and is specified within the curly brackets.

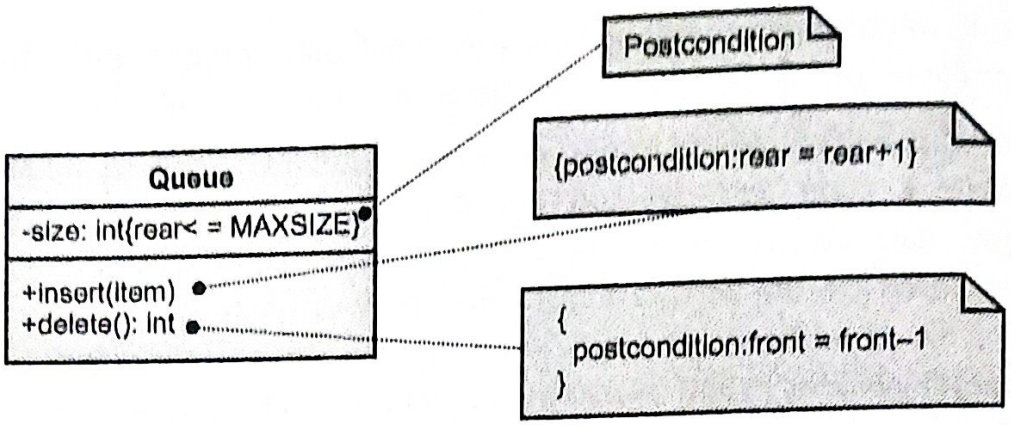
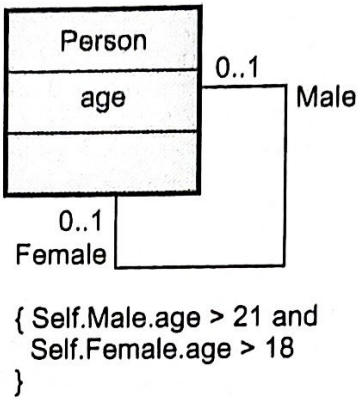


Fig. 2.9.1 Three ways to use constraints



2.9.2 Constraints on Generalization Sets

There are four types constraints on the Generalization sets. These are,

1. Complete

The complete specifies that all the children in the generalization are specified completely there is no specification possible.

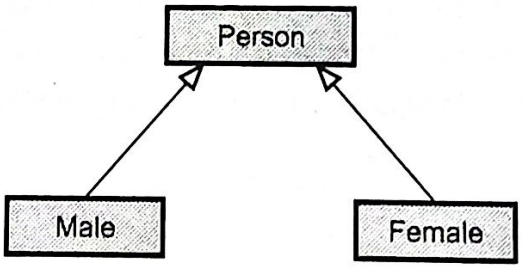


Fig. 2.9.2 Complete generalization

2. Incomplete

In the incomplete generalization not all the children in the generalization are specified completely.

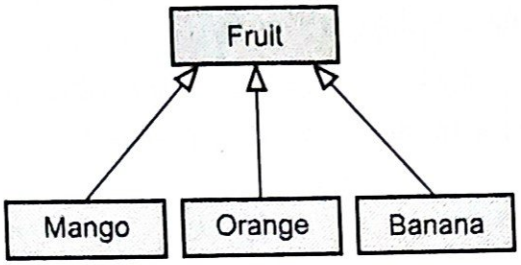


Fig. 2.9.3 Incomplete generalization

3. Disjoint

It specifies that objects of parent may not have more than one of the children as type. For example - In the following figure the class **AdministrativeStaff** has only one child.

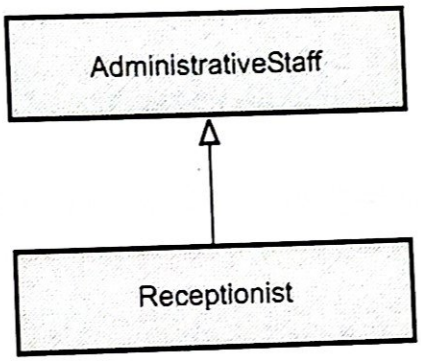


Fig. 2.9.4 Disjoint constraint

4. Overlapping

It specifies that objects of parent may have more than one of the children as type. For example - Here train can be passenger type vehicle or goodscarrier.

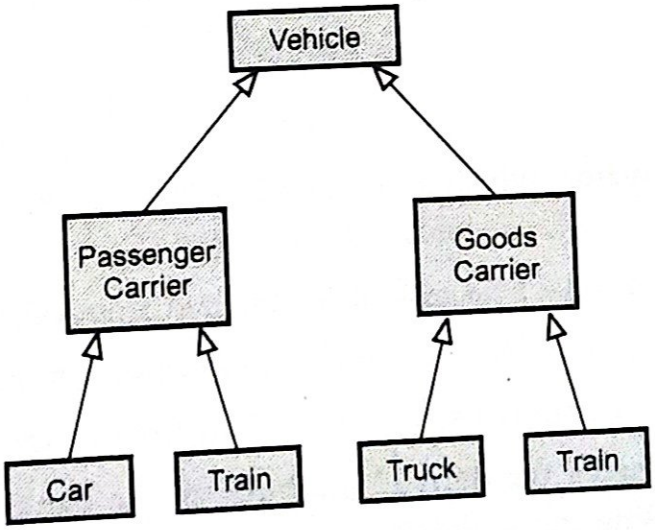


Fig. 2.9.5 Overlapping generalization

2.9.3 Constraints on Links

The constraints on the links can be allowing the multiplicity. The multiplicity denotes how many objects are related to the given object. Multiplicity for an attribute specifies how many values are possible for that attribute.

By applying constraints on link we can make the objects to appear in some specific order. The constraint {order} can be applied for this link. Following figure represents how constraint can be applied on the links -

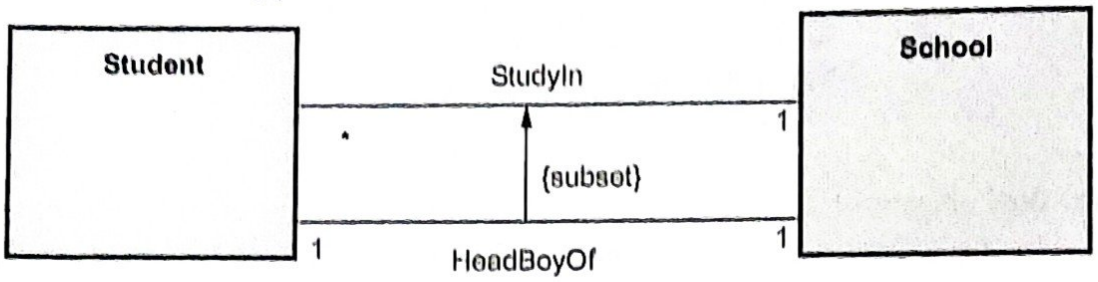


Fig. 2.9.6 Subset constraints on links

2.9.4 Use of Constraints

In UML we can specify the constraints in two ways - first by writing the constraints in curly braces and second by specifying the constraints in a dog-eared comment box. Refer Fig. 2.9.1 and Fig. 2.9.6.

Review Questions

1. *Define the purpose of following terms with suitable example and UML notations with respect to class model : i) Qualified association ii) Association class iii) Aggregation iv) Multiplicity v) Constraint.*
2. *What is a constraint ? Explain constraints on objects, constraints on generalization sets and constraints on links.*

2.10 Derived Data

Derived data is a function which is obtained using one or more functions or data elements.

The derived data is redundant because it is obtained from other element.

In class diagram, data attributes, associations and classes can be derived.

The notation for derived data is use of slash "/" in front of the element name.

Following figure represents the derived attribute. In the banking system, the account balance can be derived using the withdrawal.

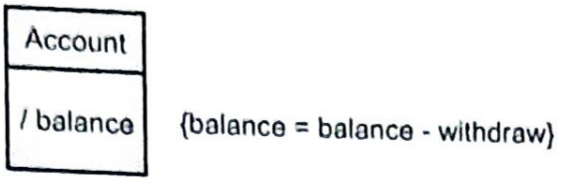


Fig. 2.10.1 Derived attribute

Following is an example in which derived association and derived class is represented for the baking system.

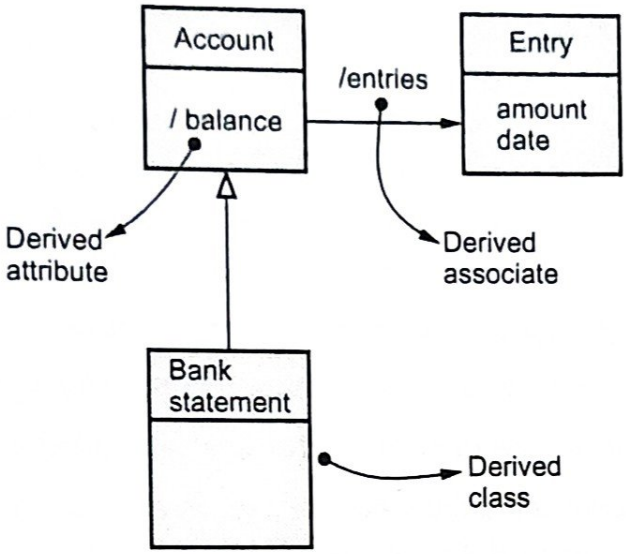


Fig. 2.10.2 Derived class and association

2.11 Packages

In UML, package is a general purpose mechanism for modeling the elements into groups. Using packages the elements can be organized properly. The package can be represented as shown in the Fig. 2.11.1.

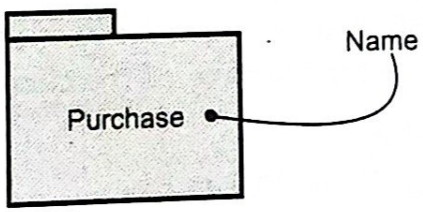


Fig. 2.11.1 Package

Review Questions

1. *Define the purpose of the following term with suitable example and UML notation with respect to class model - Package.*
2. *Define following terms : Derived data, Abstract class, generalization, multiplicity, constrains, metadata, package.*

Part II : State Modeling

2.12 Introduction

State model describes the sequence of operations that occur in response to external stimuli. The state model is represented by using the **state diagram**. The state diagram is a graphical representation in which the events and states are represented. Events represent external stimuli and the state represents values of objects.

2.13 Events

- An **Event** is a specification of noteworthy occurrence of something that has location in time and space. For state model, the event is an occurrence of **stimulus** because of which the state **transitions** take place. For example - user switch off the alarm. The timer counts down.
- The events are often denoted by the **verbs in past tense**.
- One event may follow another event, or events might occur independently.
- The two events that are independent of each other are called **concurrent events**.
- If the time difference between occurrence of two events is very large then the events are called concurrent events. If the event occurs at distant places then also they are called as concurrent events. The ordering between concurrent events need not be specified.
- There are various types of events, the most commonly used events are signal events, change events and time event.

2.13.1 Signal Event

- A message is a named object that can be passed asynchronously by one object to another. The **signal** is a type of the message or the classifier for the message.
- Signals are similar to the classes.
  - The signals have instances just like the classes.
  - The signals allow to use the generalization relationship so that the hierarchy of events can be modeled. Because there might be some general events and some might be the specific events. For example: **BeepSound** signal can be **BootingUpBeep** or **DiskFailureBeep**
  - The signals might have the attributes and operations associated with them.
- There are various ways by which the signal can be sent -
  - In state machine, the signal can be sent by **action of transition.**
  - For the two roles of interaction, the signal can be modeled as a message.

- By executing some method, the signal can be sent. Thus the signal specifies the behavior of the element.

- In UML the signal can be represented as stereotyped class or a dependency relationship. When the signal is modeled as class, it has the stereotype <<signal>> and then the signal is modeled by executing some message it is modeled using the stereotype **<<signal>>**

- **Example :**

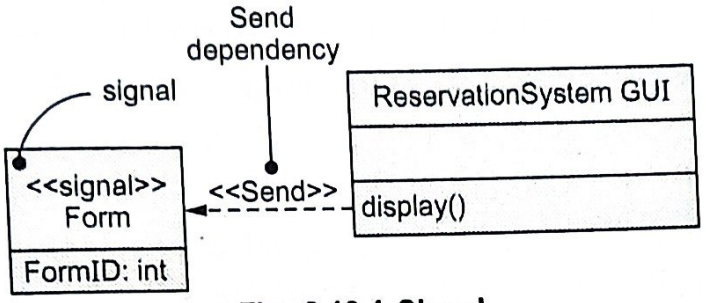


Fig. 2.13.1 Signal

As the Fig. 2.13.1 shows that the signal can be modeled as stereotyped classes or we can use a dependency relation to show a signal. (In Fig. 2.13.1 it is shown using the stereotype **send**)

2.13.2 Change Event

- The **change event** represents a change in state or satisfaction of some condition. The keyword **when** followed by some Boolean expression is used for denoting the change from false to true.
- For example - when (temp<10 °C) , when(power > maximum limit)

2.13.3 Time Event

- The time event represents the passing of time. The keyword **after** followed by some expression is used for denoting the time event. This expression represents the period of time. For example : After 5 seconds
- The occurrence of time event can be at absolute time. This can be represented using the keyword **at** . For example **at(1September 2020, 6:00 UT)** specifies that on 1st September 2020 at universal time 6 O'clock.
- Following Fig 2.13.2 represents the time and change event in the state model -

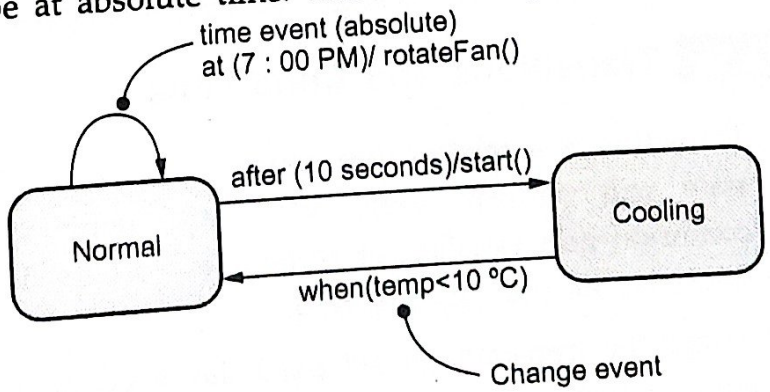


Fig. 2.13.2 Time and change event

Review Questions

1. *What do you mean by an event in state diagram ? Discuss various types of events.*
2. *What is an event ? Explain types of events.*

2.14 States

- State is a condition or situation during the life of an object during which it satisfies some condition or performs some activity or waits for some event.
- Object remains in particular state for finite amount of time. For example - Microwave might be in various states such as **Start** (Initial State when switched on), Activating (waiting for some buttons to get pressed), Active (working/progressing) and Stopped.
- The state can be represented in UML by rounded box containing some name. The name is optional but it represents the name of the state. It is written in boldface, at center of the box and the first letter is capitalized.

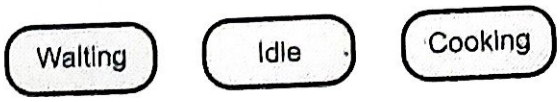


Fig. 2.14.1

- If the attributes of the state are do not affect the behavior of the object then we do not represent them in the state.
- **Difference between event and state**

Events represent particular moment of time whereas the events represent interval of time. A state corresponds to the interval between two events received by the objects. Both events and state depend upon the level of abstraction.

Review Question

1. *Differentiate state and event. List and explain different types of events.*

2.15 Transitions and Conditions

- Transition is a relationship between two states. It indicates that the object in the first state will perform some action and enter in some another state when specific condition gets satisfied or some event occurs. When the state gets changed then it is said that the transition is fired.
- Before the transition to get fired the object is in source state and after the transition gets fired the object is in target state.

- For example : The **Microwave** changes its state from **Start** to **Activating** when the **OneMinute** button gets pressed (for cooking the food). Here pressing the button is an event that causes transition from one state to another.
- There are five parts of transitions -
  1. **Source state :** This is the starting state for the transition and is affected by the transition.
  2. **Event trigger :** It is recognized by the object in the source state. It causes the transition to get fired.
  3. **Guard condition :** It is a Boolean expression. It is evaluated when the transition is triggered. When the expression is evaluated to true then the transition is fired, and then it is evaluated to false the transition is not fired.
  4. **Effect :** An executable behavior such as action that acts on the object.
  5. **Target state :** The active state after completion of transition.
- **Example :**

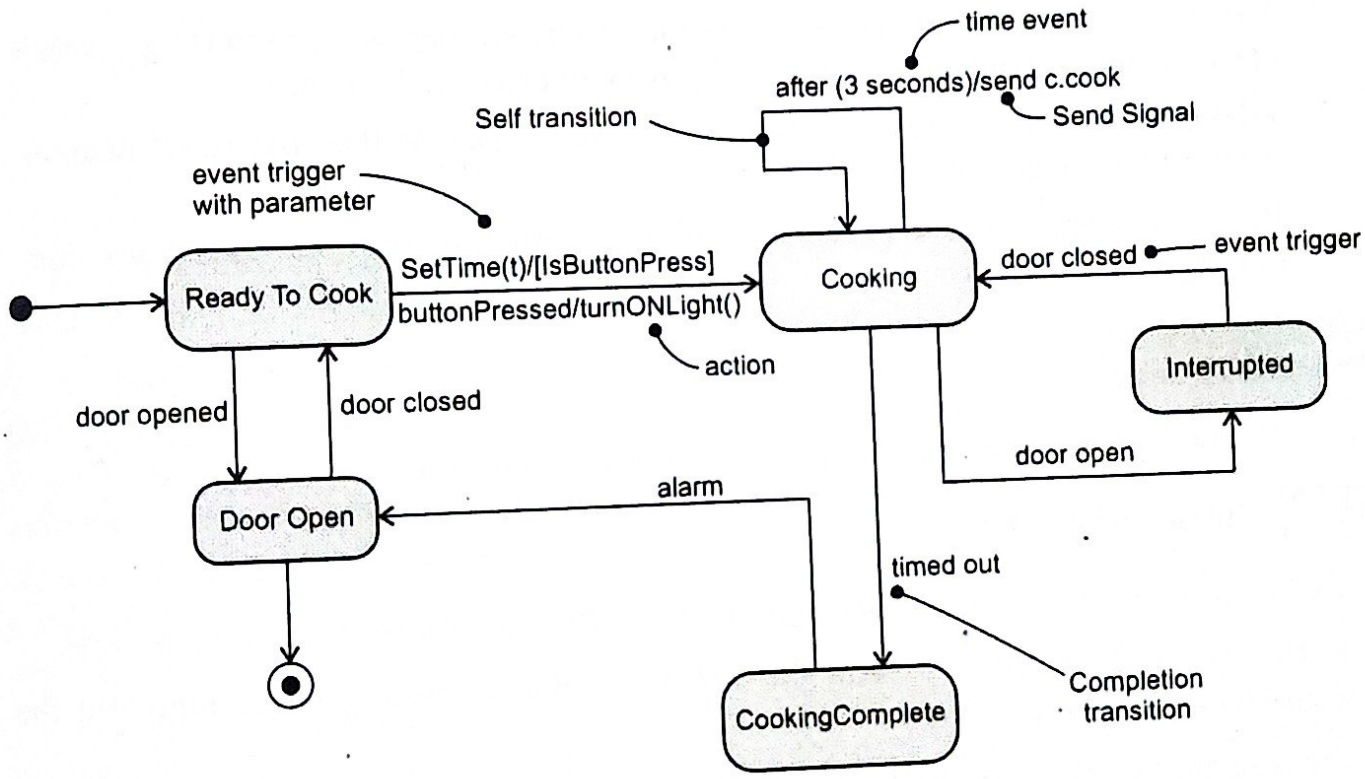


Fig. 2.15.1 Transitions

- **Event trigger :**
  - Event is occurrence of stimulus that can trigger a state transition.
  - Event can be calls, signals, passing of time or a change in state.

- An event in the form of a signal or a call may have parameters that are available when the transition occurs. The guard conditions and actions are also available for the event triggers with parameters.
- The completion transition can also be represented with no event trigger.

- **Guard condition :**
  - It is a Boolean expression which is placed inside the square bracket.
  - It is placed after trigger event.
  - A guard condition is evaluated only after the trigger event for its transition occurs.
  - Multiple transitions with same trigger event and from same source can be possible.
  - At the time when the event occurs, the guard condition is evaluated just once for each transition and if the transition is retriggered then the guard condition is evaluated once again.
  - The condition of the object can also be specified along with the guard condition.
- **Effect :**
  - Effect is the behavior which is executed when the transition fires.
  - Various examples of effects are - operation calls, sending and destructing another object, inline computations, sending signal to an object and so on.
  - When the sending of signal is shown in the state machine then the signal name is prefixed by the keyword **<<send>>**
  - When the state machine is not executing any effect of previous transitions then only the transitions occur.

Review Question

1. *Define event, state and transition. Using example draw state diagram.*

2.16 State Diagrams

- The state diagram is a **graph** that represents the **flow of control** from state to state.
- In this graph the nodes represent the **states** and the connecting edges represent the **transitions** between the states.
- The sequence of states is due to the events that occur in the system.
- The name of the states must be unique. The state should lie within the scope of the state diagram.
- All objects of a class diagram execute the state diagram for that class. That means the states of various objects can be represented using the state diagram.
- The **implementation of state diagram** means converting the semantic(meaning) of the states and transitions into the programming code.

- **State model** consists of multiple state diagrams probably representing behavior of one class for each state diagram. The diagrams in the state model must match at their interfaces.

Sample state diagram

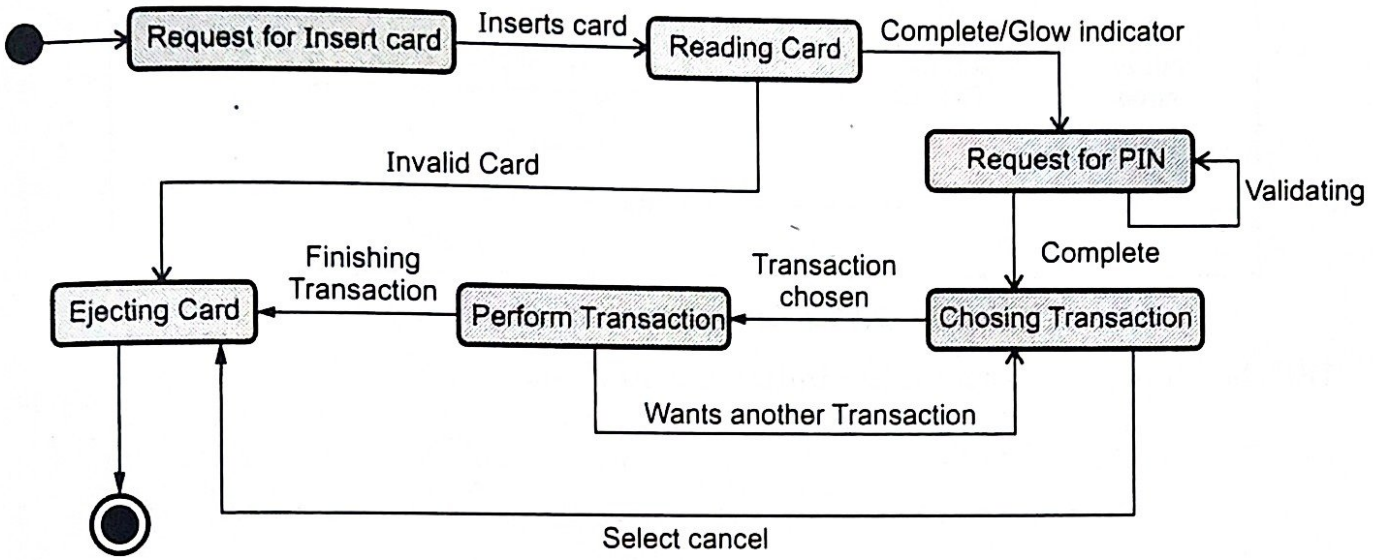


Fig. 2.16.1 State diagram for ATM transaction

In this state diagram, the state sequence caused when the customer interacts with the ATM machine for performing transactions is represented.

At the start the ATM machine is in idle state and when the customer interacts with the ATM machine then it requests for insertion of the ATM card. When the customer inserts the card, it is validated. Then the PIN is requested by the machine. The PIN entered by the customer is then validated. Only the valid PIN and Card holder is allowed to perform further transaction(such as withdrawal/ deposition of money or even checking the account balance.). On completion of the transaction, the ATM card will be ejected by the machine.

Note that this state diagram does not expose all the states or the values of the object. For instance - if the PIN is invalid then ejecting the card- this sequence is not shown in the diagram, instead, the sequence that is followed on the valid PIN is represented.

2.16.1 One Shot State Diagram

The state diagram that has finite number of states representing one-shot life cycle or representing continuous loop is called One shot state diagram.

The one shot state diagram represents the initial and final states with finite number of states. For example following Fig. 2.16.2 represents the one shot state diagram.

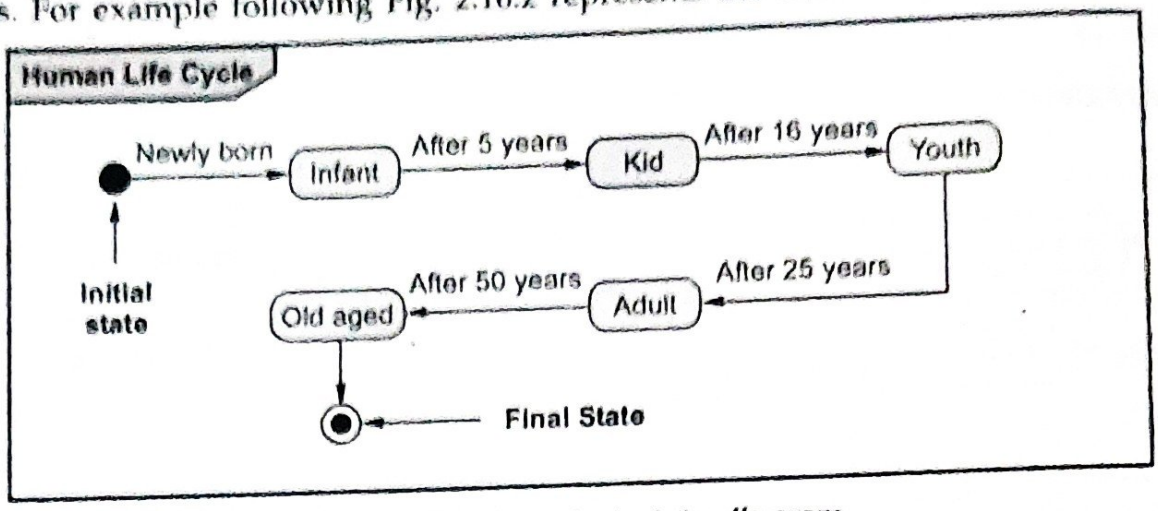


Fig. 2.16.2 One shot state diagram

The alternative notations for the **initial** and **final states** are

Initial State	○
Final State	⊗

Example 2.16.1 *What does one shot diagram represent ? Show one shot diagram for chess game.*

Solution : One shot diagram : Refer section 2.16.1.

Following Fig. 2.16.3 represents the one shot diagram for chess game -

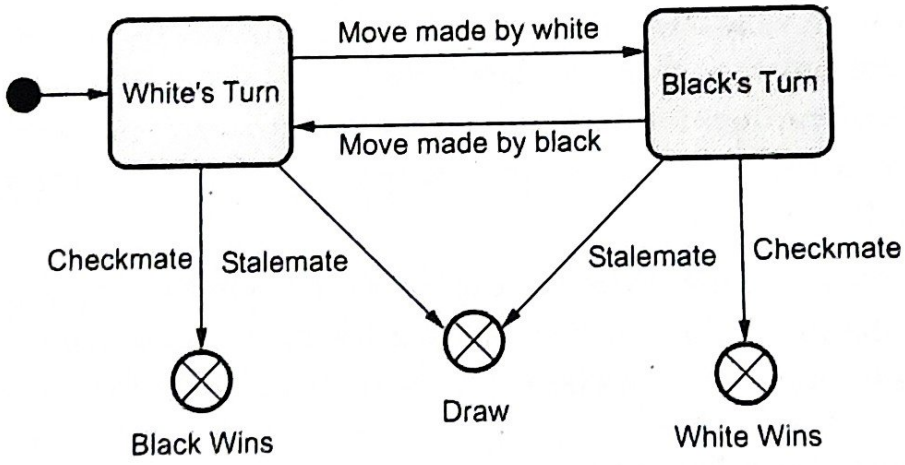


Fig. 2.16.3

2.16.2 Summary of Basic Notations

Following are some basic notations that are commonly used in the state diagram.

State : State is a condition of the object at particular moment of time. This time is particularly between the events. For example -

1. Waiting for user to enter PIN 2. Authenticating user 3. Dispensing cash.

The state is represented by the rounded box. There are special notations used for representing the initial and final states. (Refer section 2.16.1)

Transition : The relationship between two states is indicated using transition. When some event occurs then the object moves or transits from one state to another. This is represented using transition. The transition is represented by an edge with arrow head. The arrow head points to the target state.

Event : An event is significance occurrence of something. For example - Customer inserts ATM card.

Following figure represents these concepts.

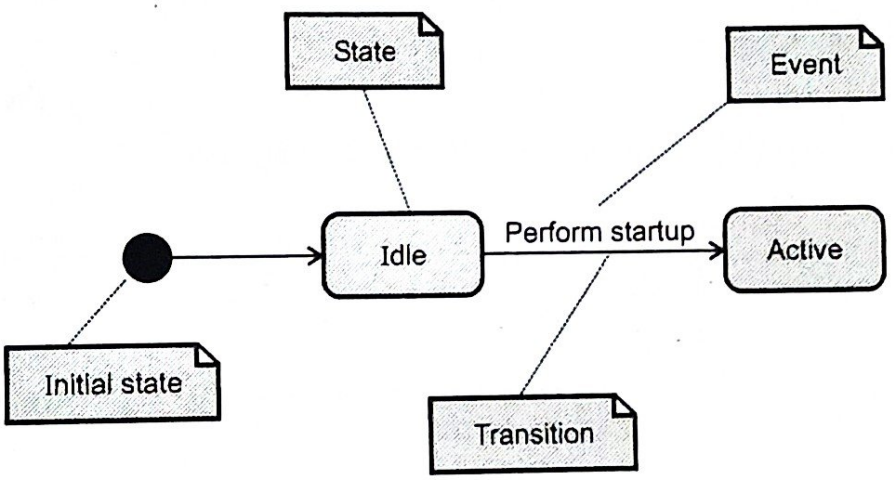


Fig. 2.16.4 State events and transitions in state diagram

State diagram : This is basically a graph that contains the states as nodes and edges as transitions. It is enclosed in a rectangular frame with the diagram name in a small pentagonal tag in upper left corner.

Guard condition :

- It is a Boolean expression which is placed inside the square bracket.
- It is placed after the trigger event.
- The guard condition is evaluated only after the event is triggered and represented along its transitions.

Effects : It is an executable behavior such as **action** in response to event. It typically acts on the object. It is represented along the transition and listed after "/"

For example

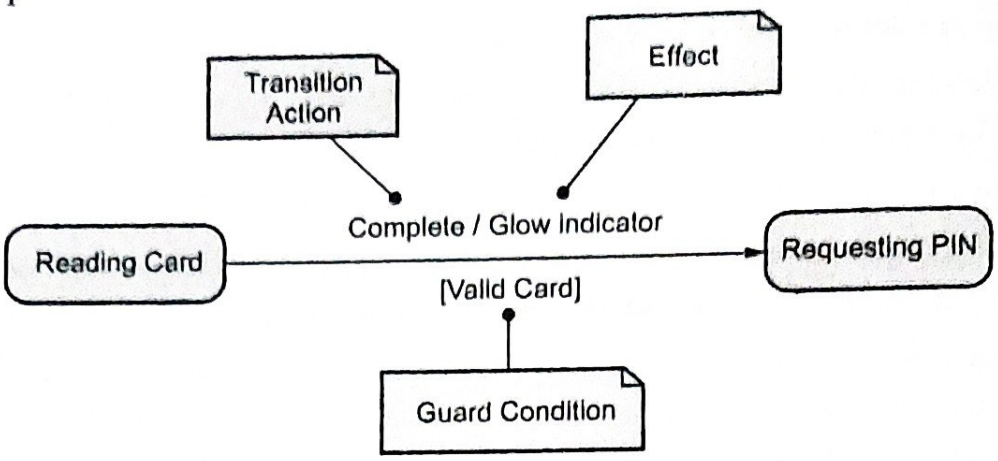


Fig. 2.16.5 Guard condition and effects in state diagram

Example 2.16.2 *A vending machine has a coin insertion slit, a display panel and a dispensing tray. At start, a vending machine is in IDLE state. Being into IDLE state, when the coins are inserted in the coin insertion slits, it goes to accepting coins state. When the amount becomes equal to the price of drink, it goes to select drink state. In this state it asks for selection of drink. When it is selected, it goes into dispensing state. In this state, it delivers the drink in the dispensing tray and goes back to idle state. Identify the transitions and write those trigger[guard condition]/effect in the state diagram.*

Solution :

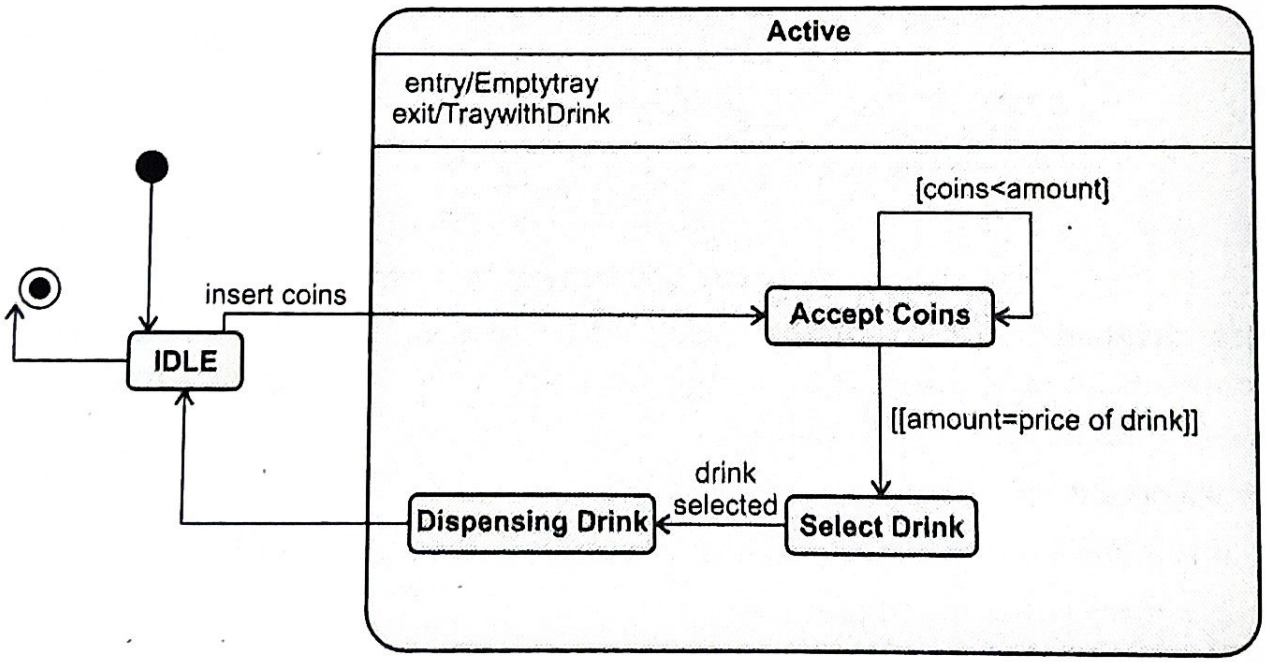


Fig. 2.16.6

Review Question

1. *Define a. State b. Guard condition.*

2.17 State Diagram Behavior

SPPU : Aug.-15, Dec.-15, April-16, 18, Marks 5

The state diagram will make use of some more notations to represent how object goes through different transitions. Let us see these effects one by one -

2.17.1 Activity Effects

- It is an executable behavior such as **action** in response to event. An **activity** is an actual behavior that may occur due to any number of effects.
- Activity can be performed at certain times such as -
  i) along the transition
  ii) on entry and exit of the states or
  iii) on occurrence of some event within the state.
- The activities are also used to represent the internal control operations. For example 'generate beep sound' when temperature exceeds 100 °C.
- The activity can be represented by "/" and the activity name following the event due to which the activity occurs.
- Using the keyword **do** the ongoing activity can be represented.
- Following is a simple state diagram which represents the use of activity within it.

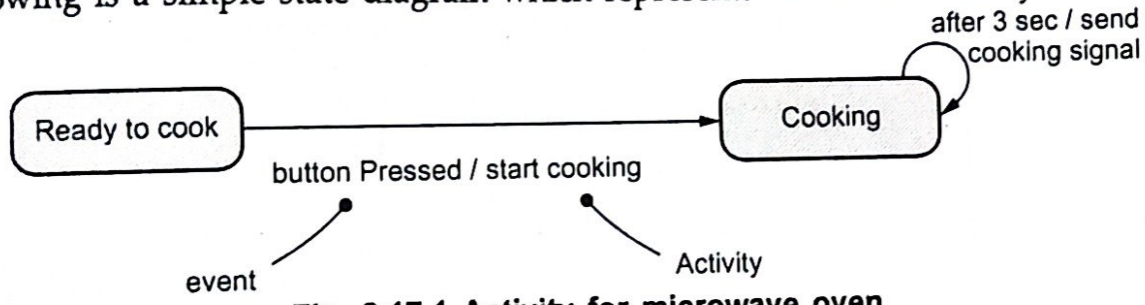


Fig. 2.17.1 Activity for microwave oven

2.17.2 Do-Activities

- When object is in some state then that object may perform some work. This ongoing activity is modeled by **do-activities.**
- The object continues with its activities until it is interrupted or the activity is performed completely.
- The notation **"do/"** is used to represent the do-activities.
- The do activity occurs within in the state and can not be attached to the transition.

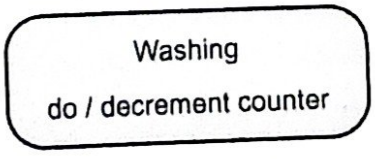


Fig. 2.17.2 Do-activity for washing machine

- For example - In the Washing machine, When the washing activity is on going then the counter is getting decremented. This can be represented using the keyword **do.**

2.17.3 Entry and Exit Activities

- There are some activities that can be performed on entry of particular state. These activities are called **entry activities.** Similarly, while exiting the state some clean up actions need to be performed. These activities are called **exit activities.**
- While specifying these activities the keyword **entry** and **exit** must be prefixed to that action.
- The exit activities are less common that entry activities but those are used for some events.
- Following Fig. 2.17.3 represents the **entry and exit activities.**

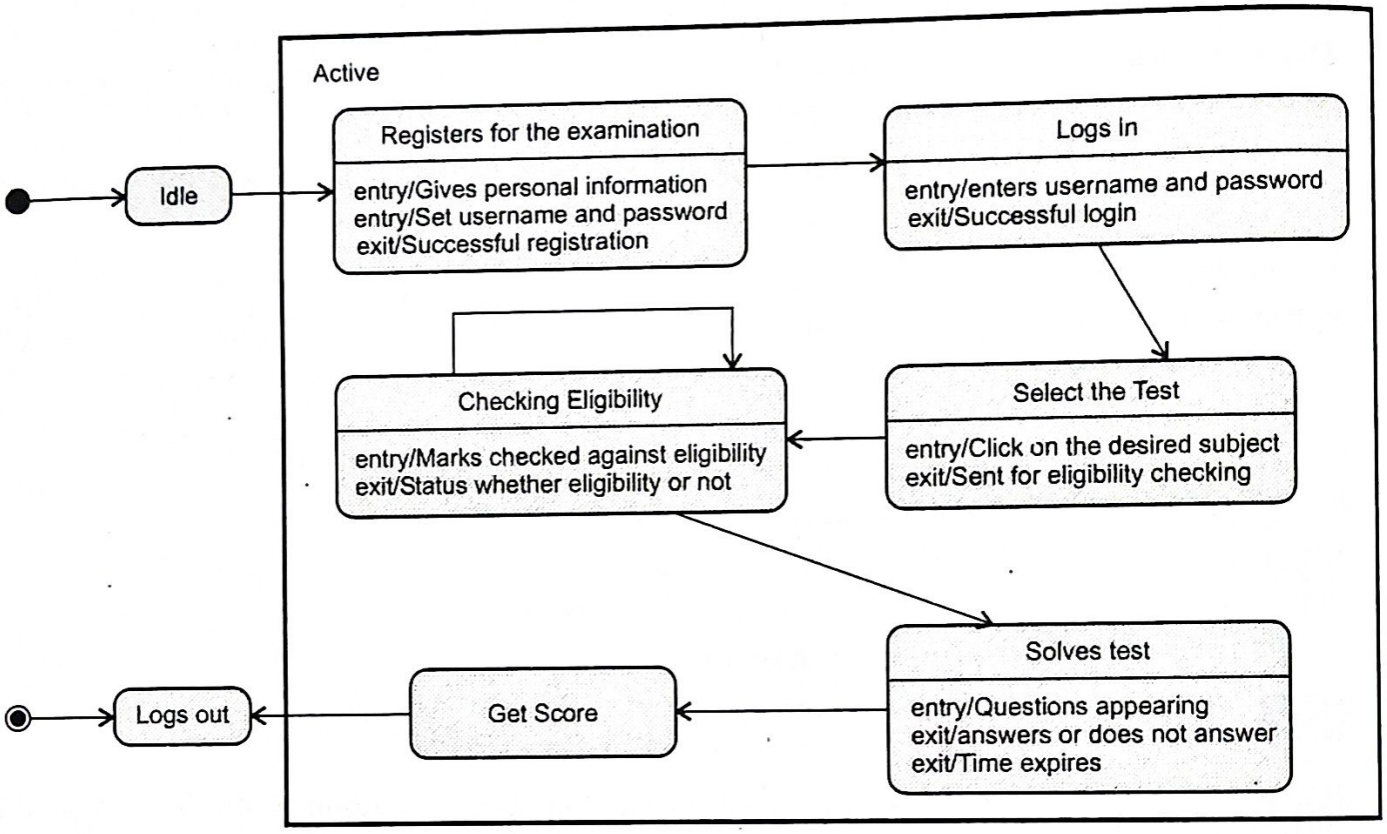


Fig. 2.17.3 Representing entry-exit activities in state diagram

- If a state has multiple activities then they will occur in following order -
  - Activities on incoming transitions.
  - Entry activities
  - Do activities
  - Exit activities
  - Activities on outgoing transitions
- The events might interrupt the ongoing activities. Even if the do activity is interrupted the exit activity will executed.

- **Difference between Events within a state and self transition:** The events within the state cause an activity to be performed. On the other hand the self transition causes entry and exit activities to be executed. For example -

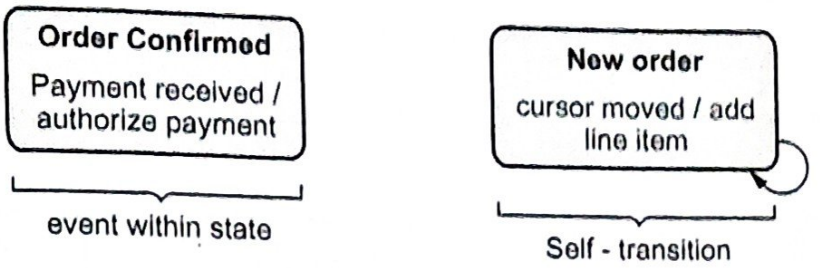


Fig. 2.17.4 Event and self transition

2.17.4 Completion Transition

- State diagrams are created to complete some activity by performing sequential tasks. When an activity gets completed a transition to another state occurs.
- The completion transition is represented using the arrow without event name. Such unlabeled transitions are called the completion transitions. For example - Refer Fig. 2.17.5.

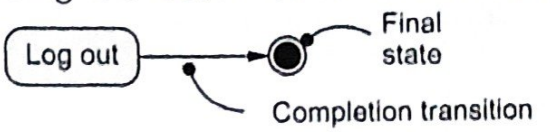


Fig. 2.17.5 Completion transition

- Once the completion transition occurs no completion event occurs.
- Normally guard conditions are not represented along the completion transitions.

2.17.5 Sending Signals

- Due to occurrence of some event the signals can be sent from one object to another.
- The objects can interact with each other by sending the signals.
- For example - the **cooking** state a signal **timeout** to **Cookingcomplete** state.
- When a states accept several events from more than one object then it might affect the final states. This is called the **race condition**. The race condition is undesired in the system design.

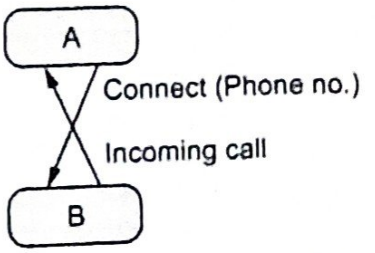


Fig. 2.17.6 Race condition

2.17.6 Examples

Example 2.17.1 *State diagram for stock maintenance system.*

Solution :

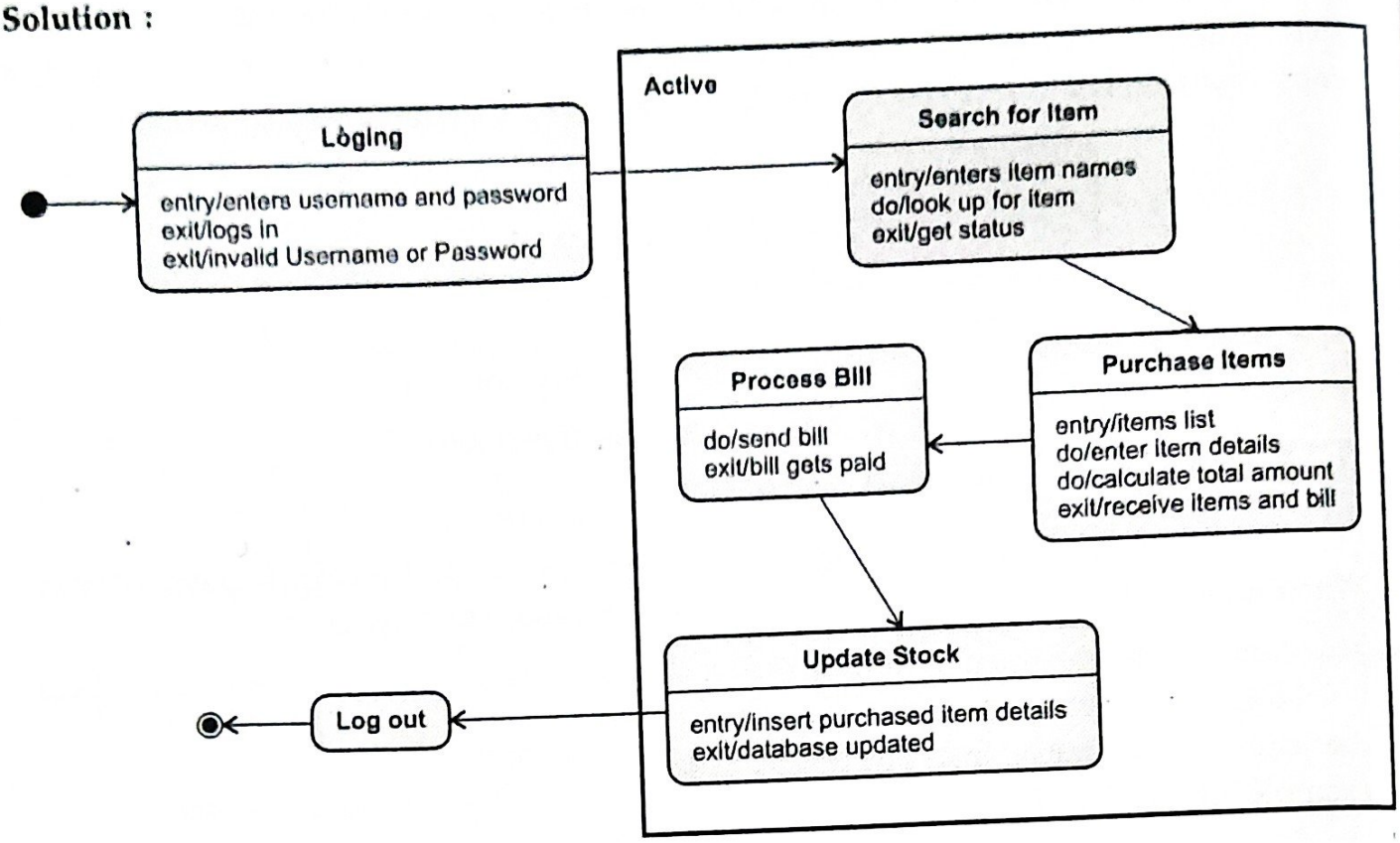


Fig. 2.17.7

Example 2.17.2 *State diagram for online reservation system.*

Solution :

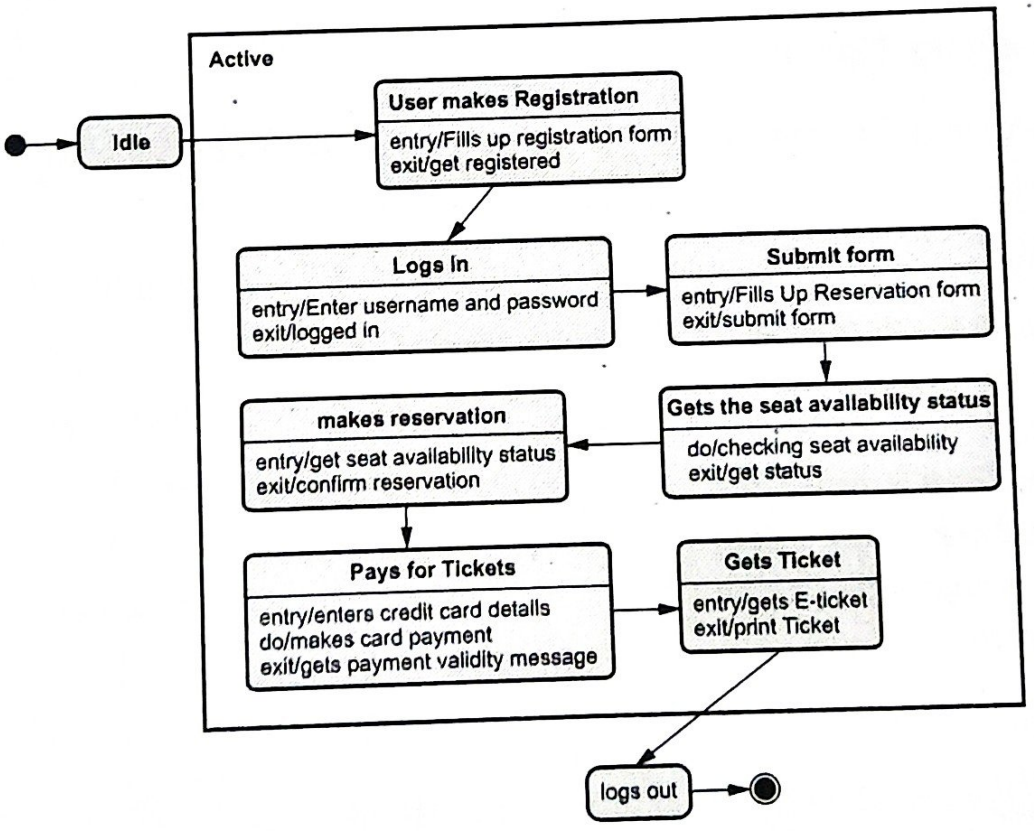


Fig. 2.17.8

Example 2.17.3 *State diagram for elevator.*

Solution :

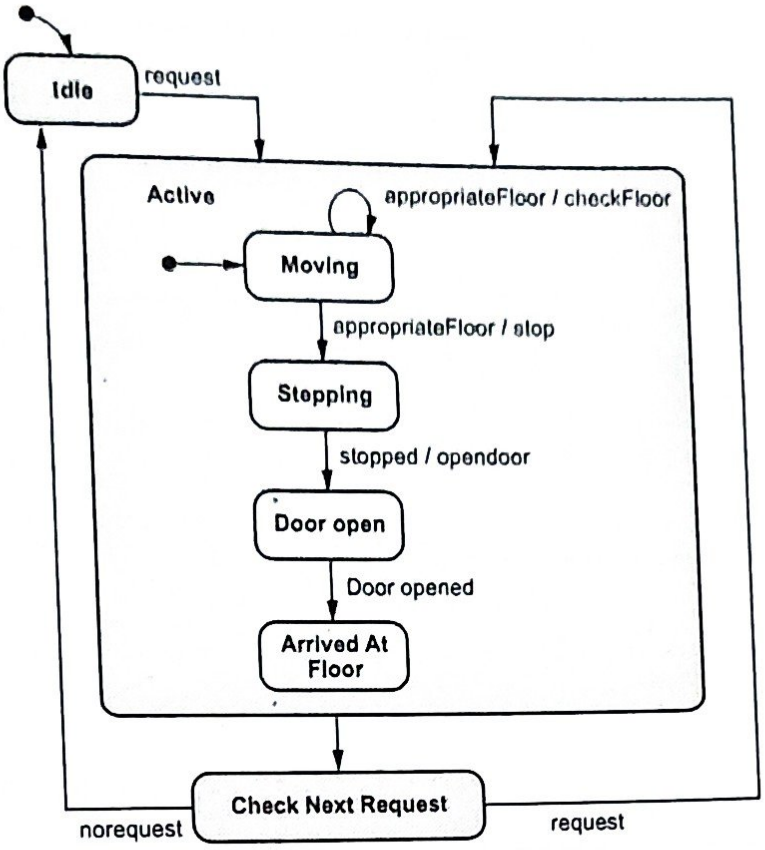


Fig. 2.17.9 Modeling lifetime of object

Example 2.17.4 *State diagram for home security system.*

Solution

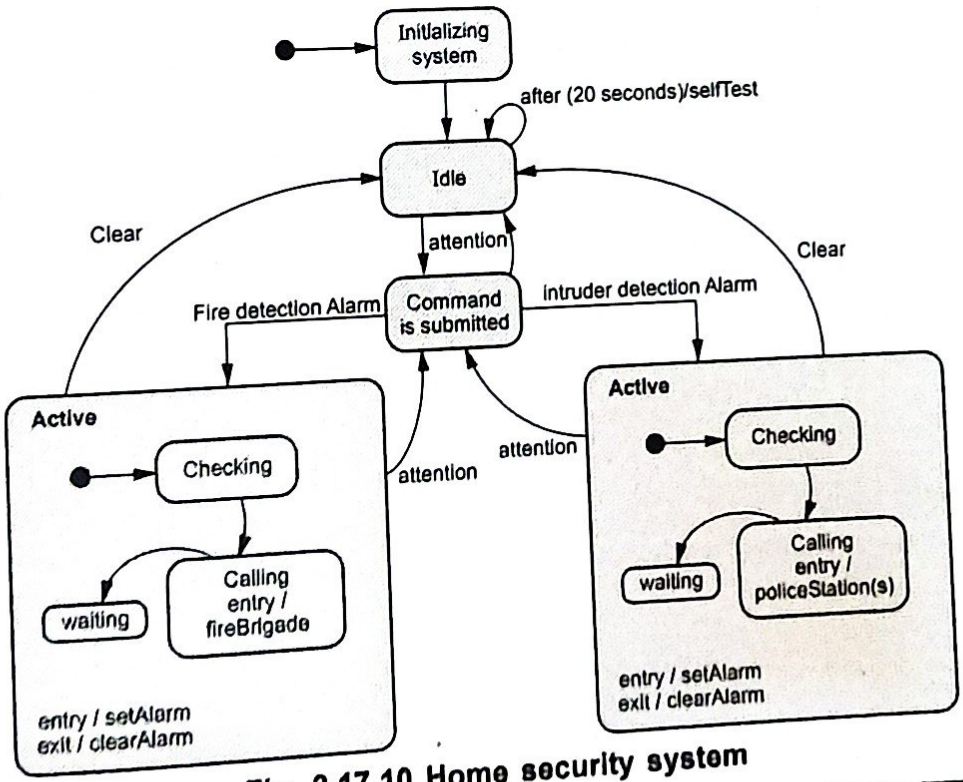


Fig. 2.17.10 Home security system

Example 2.17.5 *State diagram for microwave oven.*

Solution :

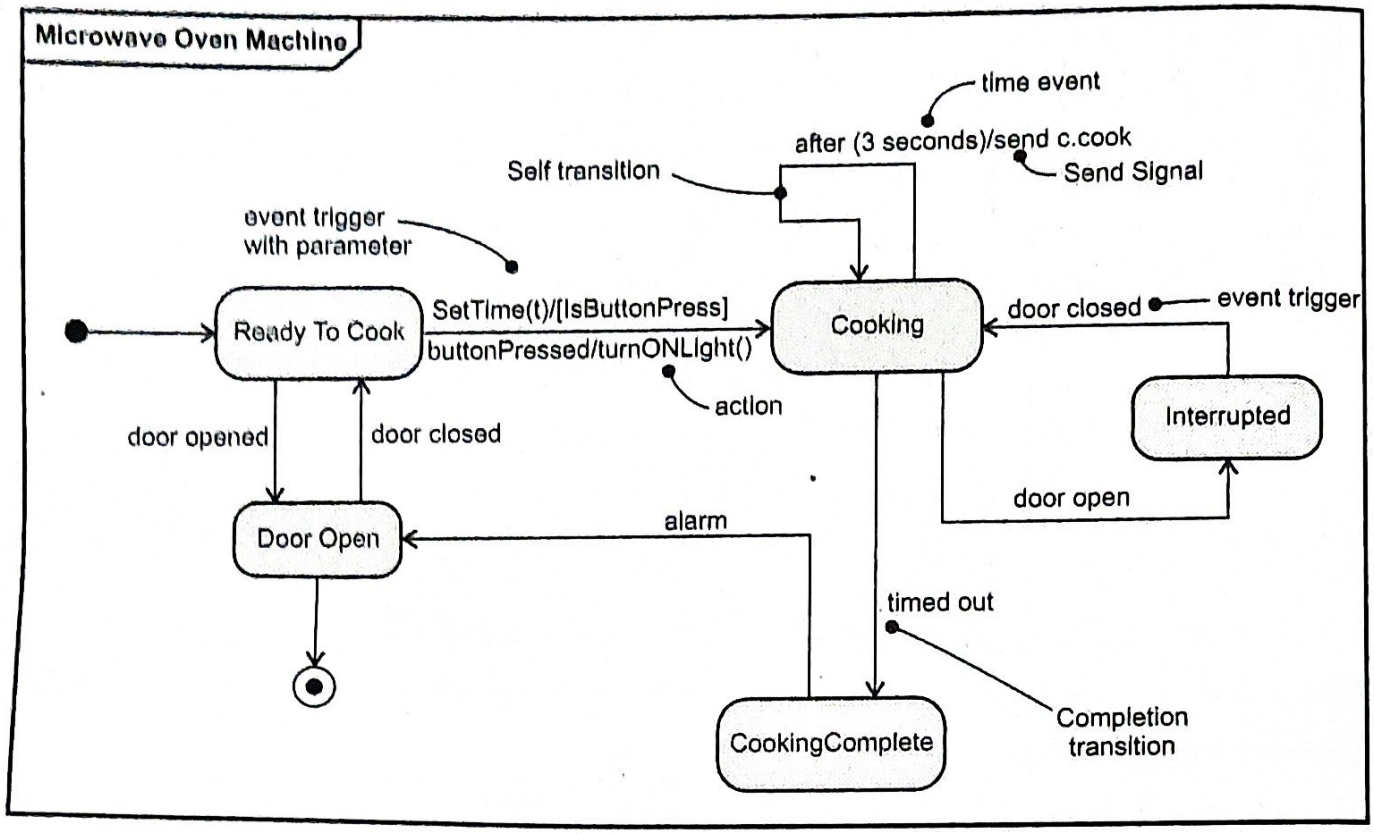


Fig. 2.17.11 Transitions

Example 2.17.6 *State diagram for photocopier machine.*

Solution :

Various states can be	Various events can be -
i) Off	i) Power turn off
ii) Warming	ii) Power turn on
iii) Ready	iii) Startbutton pressed
iv) Makecopies	iv) Paper jammed
v) Paper Jam	v) Paper tray empty
vi) Unjammed	

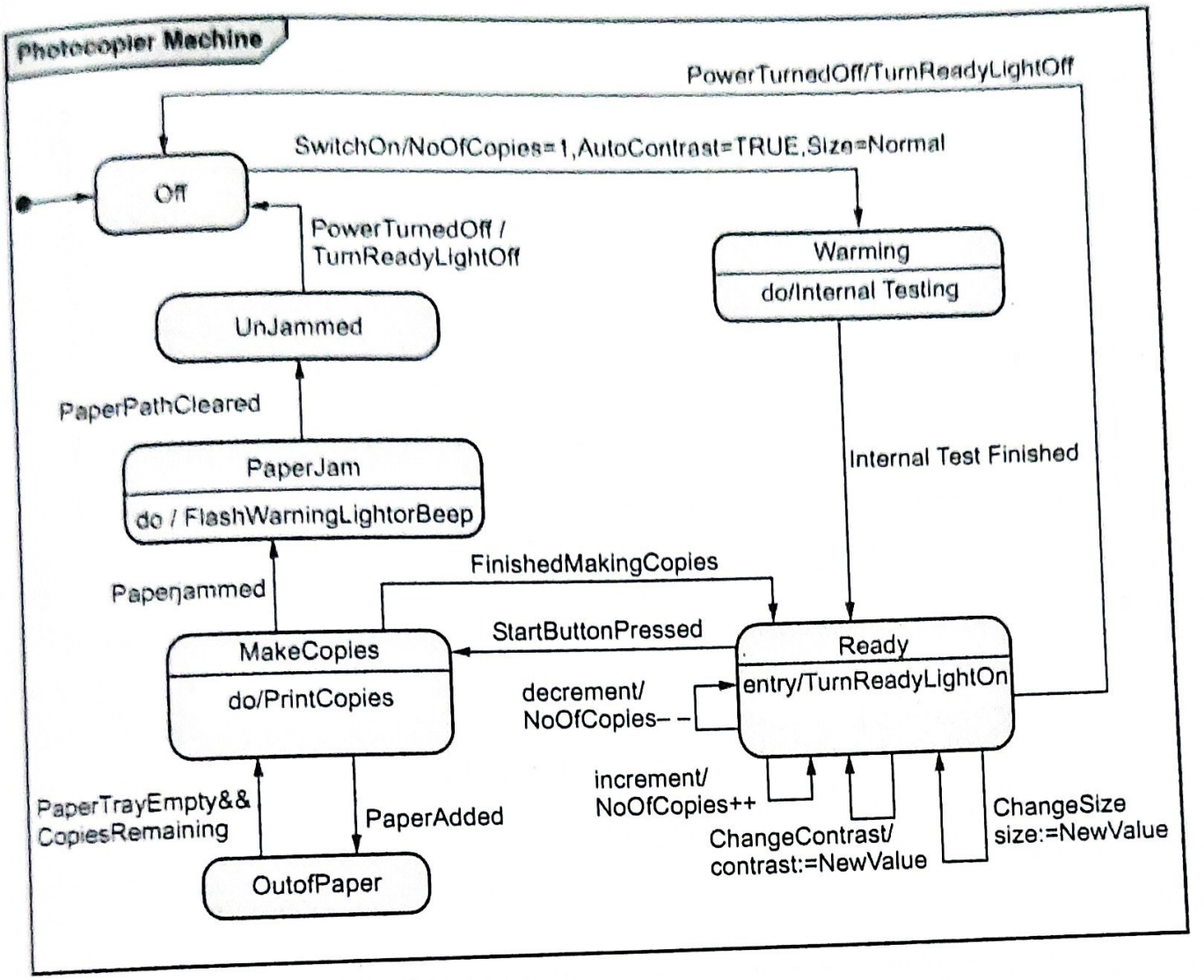


Fig. 2.17.12 State diagram for photocopy machines

Example 2.17.7 *Draw state diagram for the control of a telephone answering machine. The machine detects an incoming call on the first ring and answers the call with a prerecorded announcement. When the announcement is complete, the machine records the caller's message. When the caller hands up, the machine hangs up and shuts off. Place the following in the diagram : call detected, answer call, play announcement, record message, caller hangs up, announcement complete.*

Solution : The state model for answering machine is as given below -

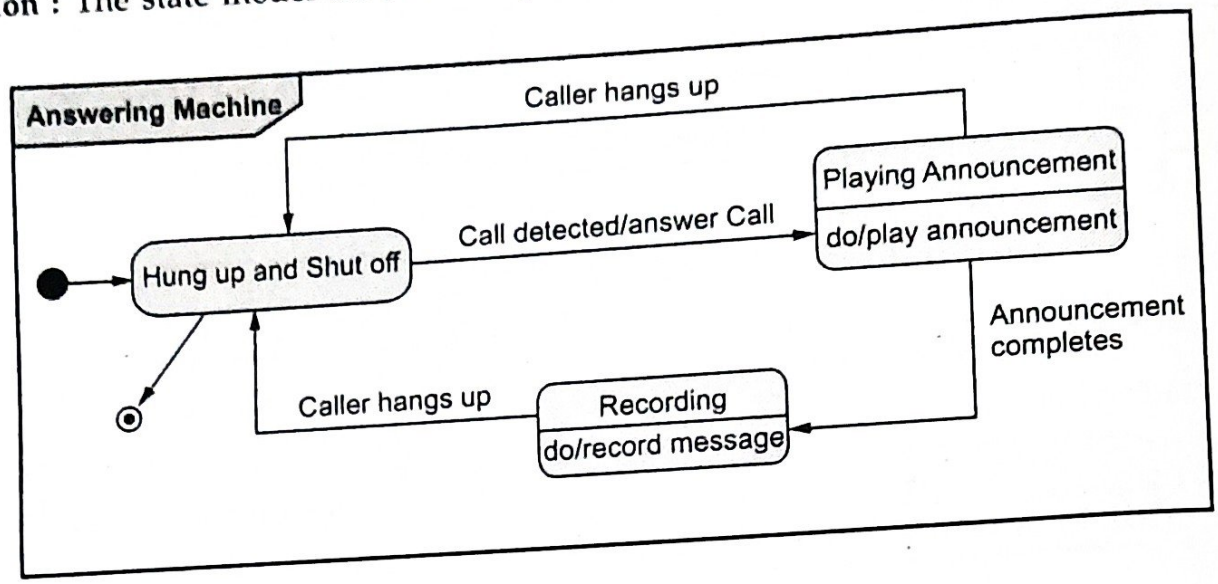


Fig. 2.17.13

Example 2.17.8 *In a personal computer, a disk controller is typically used to transfer a stream of bytes from a floppy disk drive to a memory buffer with the help of a host such as the Central Processing Unit (CPU) or a Direct Memory Access (DMA) controller. Draw state diagram for the control of the data transfer. Add the following to the diagram : reset indicate data not available, indicate data available, data read by host, new data ready, indicate data lost.*

Solution :

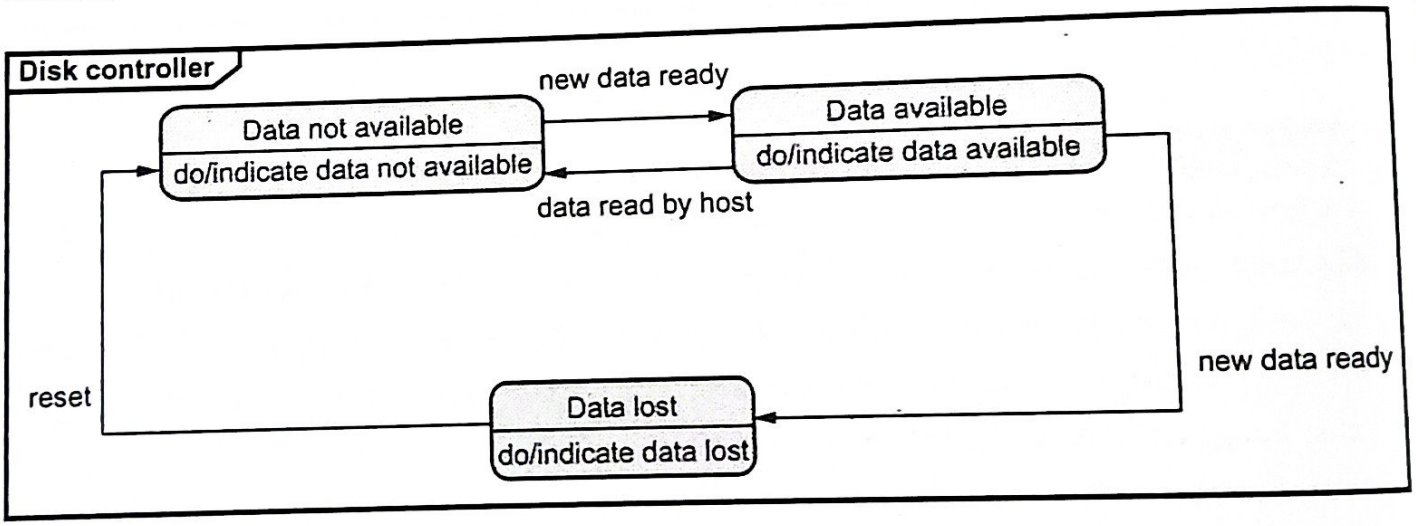


Fig. 2.17.14

Example 2.17.9 *Draw state diagram for the control of a telephone answering machine. The machine detects an incoming call on the first ring and answers the call with a prerecorded announcement when the announcement is complete, the machine records the caller's message. When the caller hangs up, the machine hangs up and shuts off. What changes need to be made if machine answers after five rings ?*

Solution : State diagram for telephone answering machine : Refer example 2.17.7.

Changes that need to be made if machine answers after five rings are represented by following state diagram -

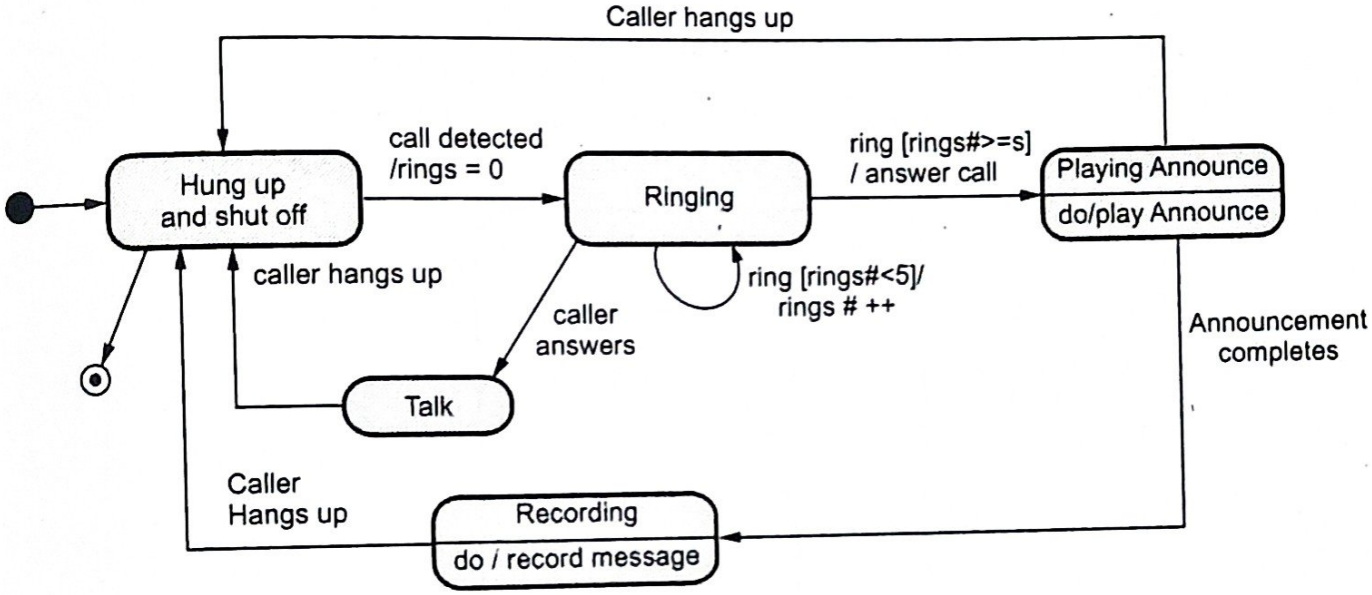


Fig. 2.17.15

Example 2.17.10 *Write the characteristics of a state alarm ringing.*

Solution : The characterization of state **Alarm Ringing** can be specified as follows -

State Name : Alarm Ringing

Description : The alarm on clock is ringing to indicate the target time.

Event sequences that produce the state :

Set Alarm for target time

When current time = Target time then **clear Alarm**

Conditions :

Alarm = Set on, Set to the target time.target time<=current time<=target time+25 seconds.

No button has been pushed since target time.

Events accepted :

Event	Response	Next state
current time = Target time + 25 seconds	Reset alarm	Normal
anybuttonpushed	Reset alarm	Normal

Example 2.17.11 *Draw and explain the state diagram of a telephone call system.*

Solution :

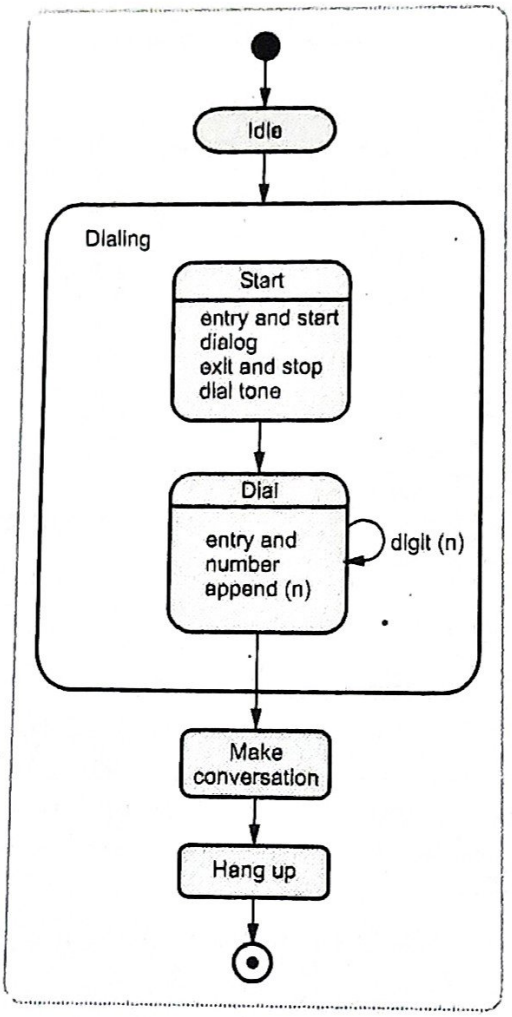


Fig. 2.17.16 Telephone call system

Example 2.17.12 *Draw a state diagram for a printer.*

Solution : The printer has two explicit states **on** and **off.** When it is **on** it undergoes through various substates such as **Idle, Working, Low tray, Empty tray** and **Full tray** states. When printer is in **off** state it is triggered to **on** state by pushing the button to

on, similarly when printer is on then it can be brought to off state by pushing the button.

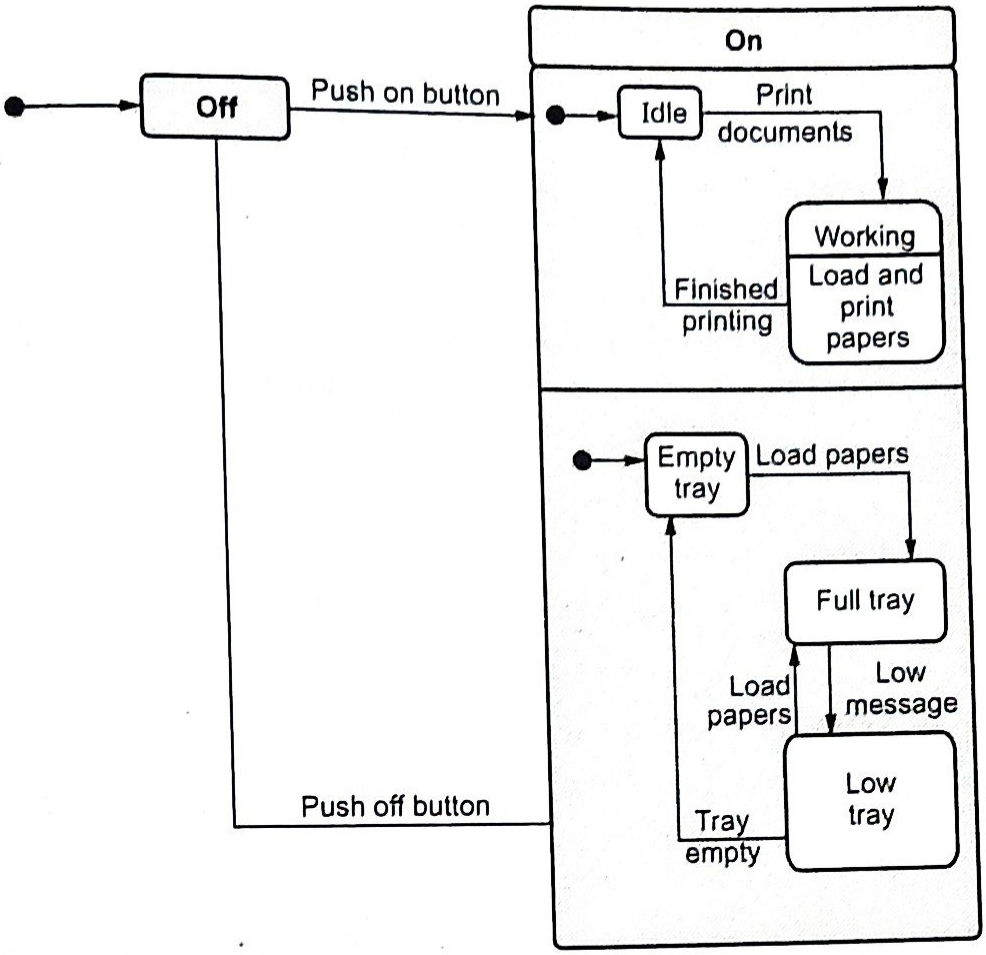


Fig. 2.17.17 State - machine for printer

Example 2.17.13 *A simple digital watch has a display and two buttons, button 'A' and 'B' to set it. The watch has two modes of display, 'display time' mode and 'set time' mode. In the display time mode, the watch displays the hours and minutes separated by a flashing colon. The set time mode has two sub modes, set hours, and set minutes. The 'A' button selects modes. Each time it is pressed, the mode advances in a sequence : display time, set hours and set minutes. Within the sub modes, the B button advances the hours or minutes once each time it is pressed. Buttons must be released before they can generate another event. Prepare a state diagram of the watch.* **SPPU : Aug.-15, In Sem, Marks 5**

Solution :

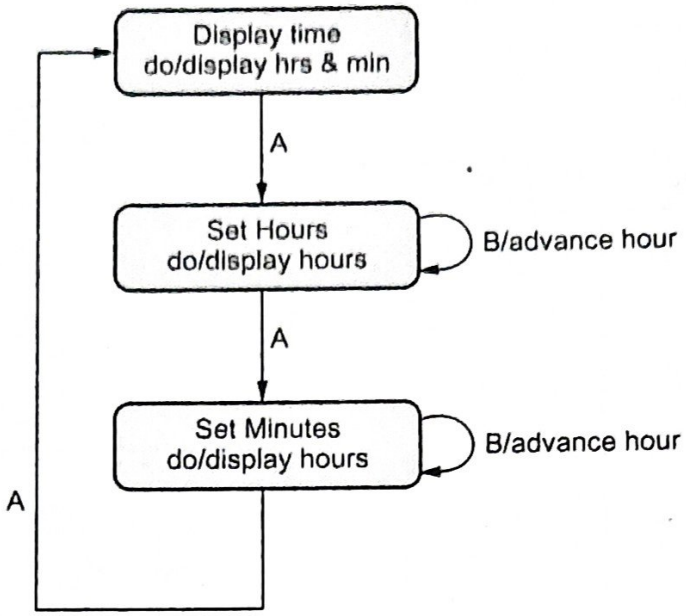


Fig. 2.17.18 Digital watch

Example 2.17.14 *Draw state diagram for ATM machine. ATM machine is idle until an ATM card is inserted. Then it becomes active. Within being in the active state, it validates the card (validating) first. After card validation, it can process a transaction like (transaction processing) a withdrawal or deposit cash or a view balance or print receipt. After the processing is complete, card is ejected and it goes to idle state.*

SPPU : Dec.-15, End Sem, Marks 5

Solution :

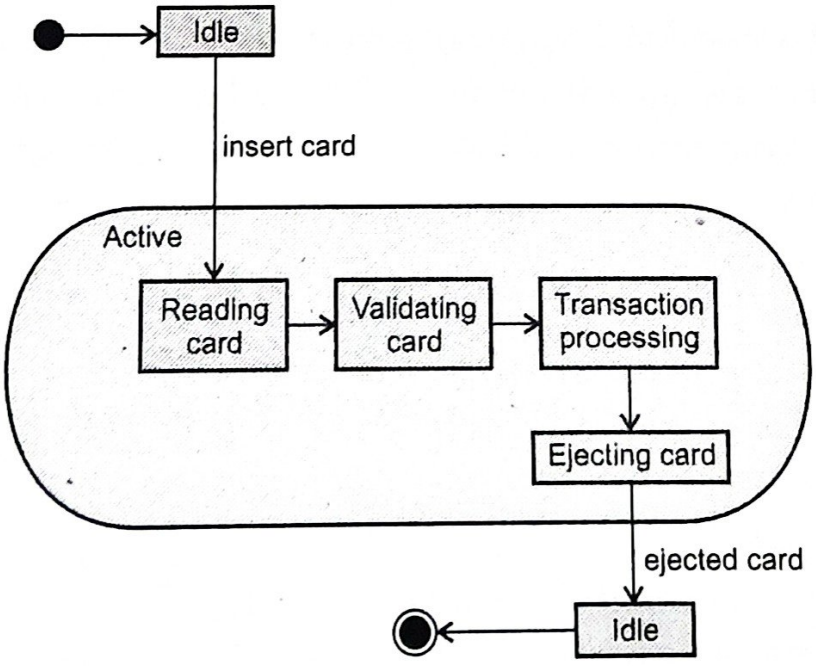


Fig. 2.17.19

Example 2.17.15 *Consider the hospital management system application with the following requirements*

i) System should handle the in-patient, out-patient information through receptionist.

ii) Doctors are allowed to view the patient history and give their prescription.

iii) There should be a information system to provide the required information.

Give the state chart, component and deployment diagrams.

Solution :

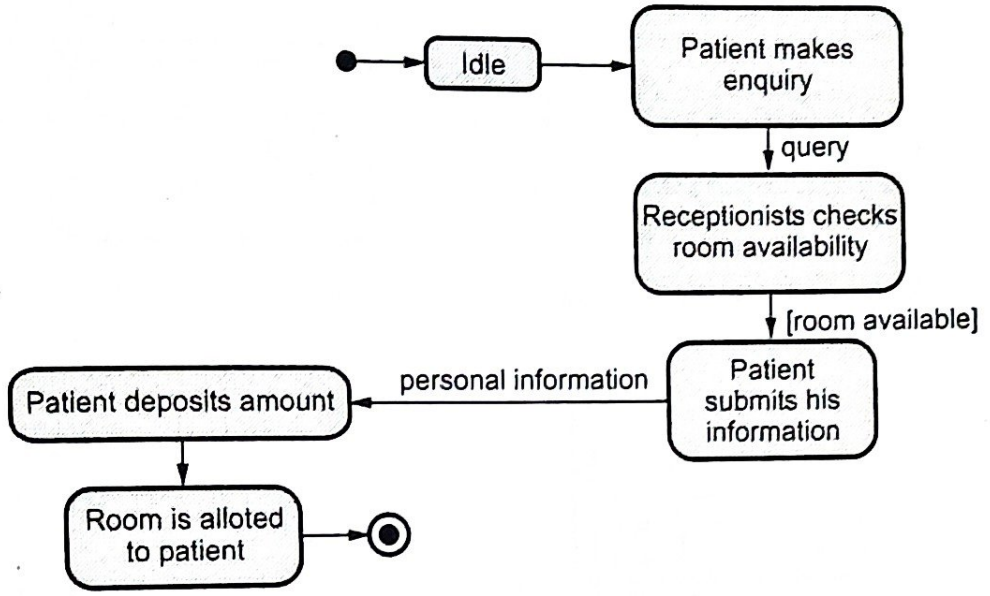


Fig. 2.17.20 State diagram for in-patient admission to hospital

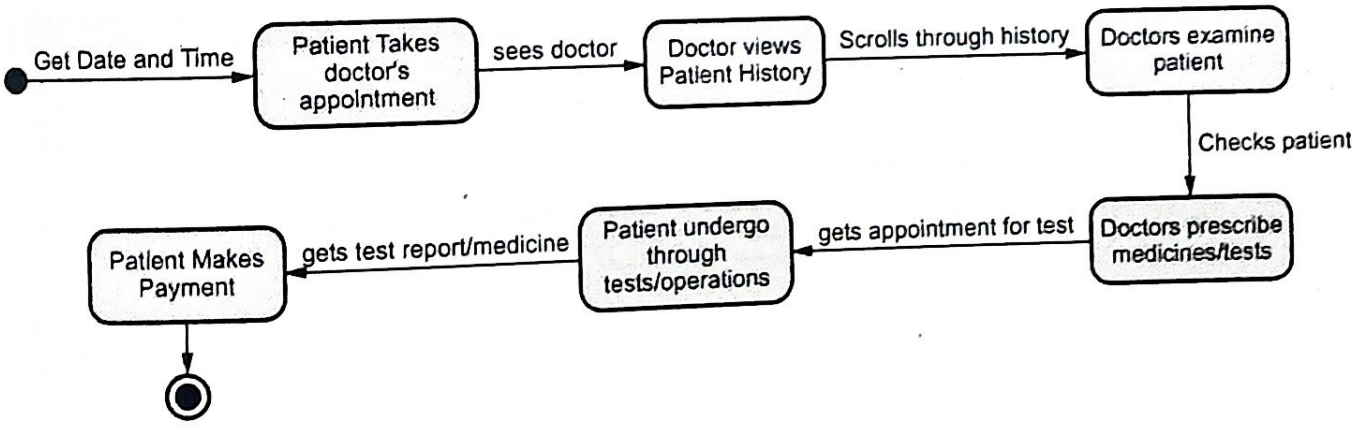


Fig. 2.17.21 State diagram for in-patient/out-patient treatment

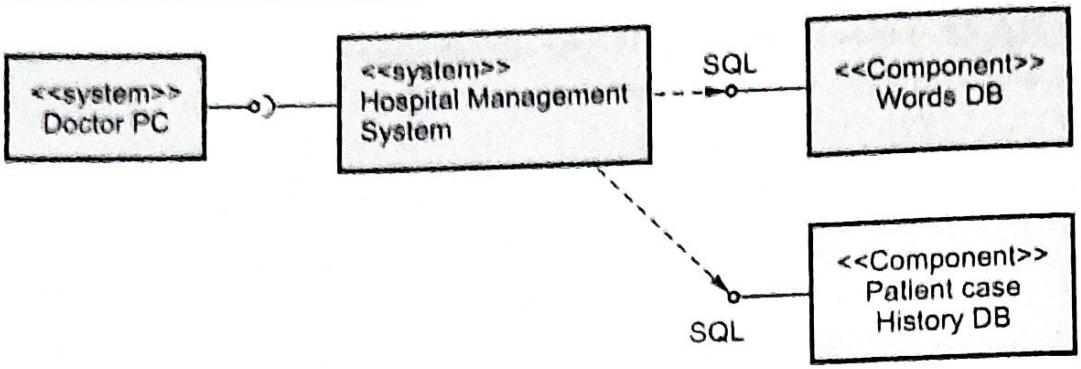


Fig. 2.17.22 Component diagram for hospital management system

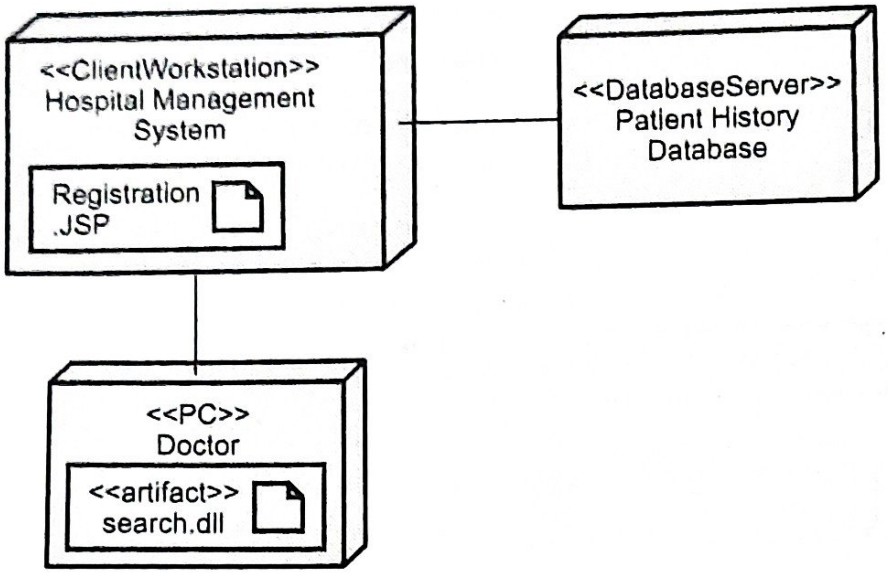


Fig. 2.17.23 Deployment diagram

Example 2.17.16 *Draw a state machine diagram for coffee vending machine.*

SPPU : April-16, In Sem, Marks 5

Solution :

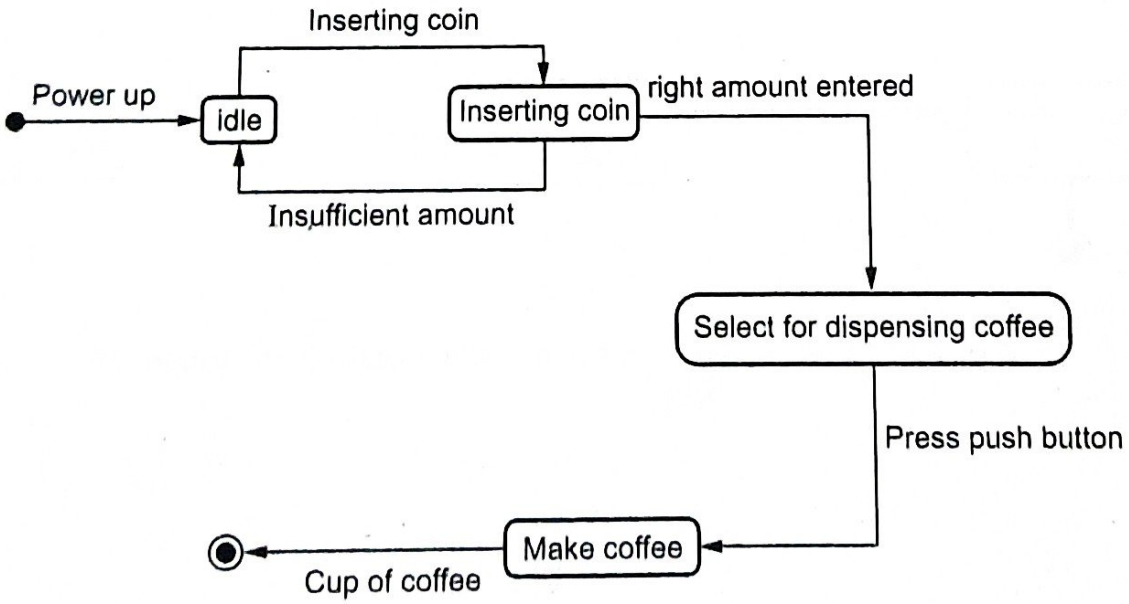


Fig. 2.17.24

Example 2.17.17 *Draw a state machine diagram for ATM machine. ATM is initially turned off. After the power is turned on, ATM performs startup action and enters self test state. If the test fails, ATM goes into out of service state, otherwise there is triggerless transition on the idle state. In this state ATM waits for customer interaction.*

SPPU : April-18, In Sem, Marks 5

Solution :

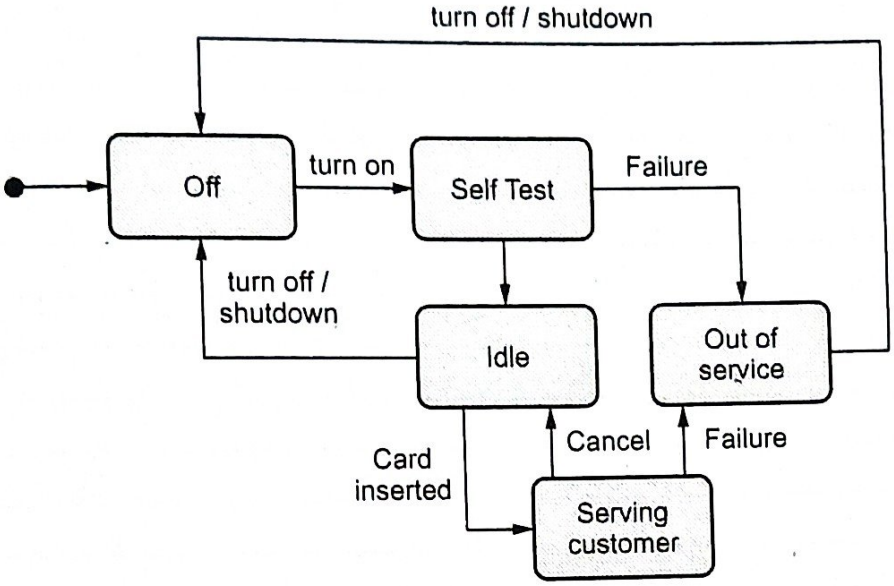


Fig. 2.17.25

Review Question

1. *Explain state diagram behavior.*