Solution
Internal Assessment Test III – Nov.2019

Sub:	Software Architecture & Design Patterns							Code:	17IS72
Date:	16/11/2019	Duration:	90mins	Max Marks:	50	Sem:	VII	Branch:	ISE

Note: Answer Any Five Questions

1. Explain the **Adapter design pattern** w.r.t following:
   a) Intent b) motivation c) structure d) participants e) collaborations

Ans.

→ Adapter:-

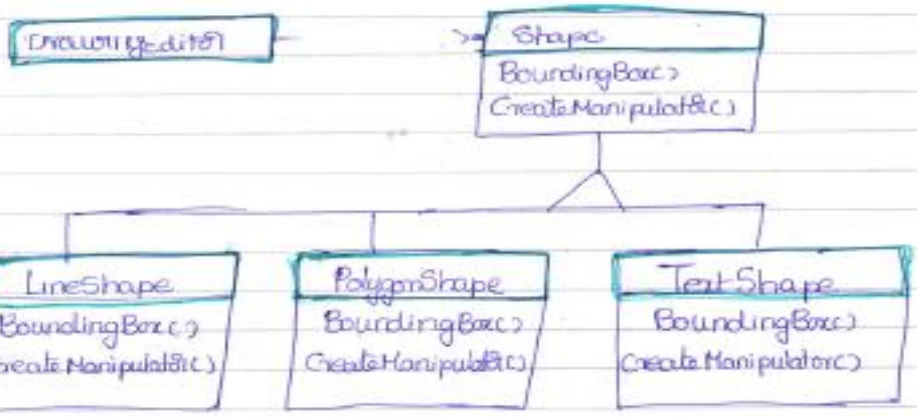
Intent:-
- Convert the interface of a class into another interface clients expect.
- Adapter lets classes work together that couldn't work ~~together~~ because of incompatible interfaces.
- Also known as Wrapper ✓

Motivation:-
* Sometimes a toolkit class designed for reuse isn't reusable only because its interface doesn't match the domain-specific interface an application requires.
* Eg:- A drawing editor:-
  - The key abstraction is the graphical object, which has an editable shape and can draw itself.
  - The classes used are as follows:-

a) **Shape** class – An abstract class that provides an interface for graphical objects.

b) **LineShape** class, **PolygonShape** class – Subclasses of **Shape** for each kind of graphical object i.e., line, Polygon.
- They are easy to implement because their drawing and editing capabilities are inherently limited.

c) **TextShape** subclass – can display and edit text. It is more difficult to implement, since even basic text editing involves complicated screen update and buffer management

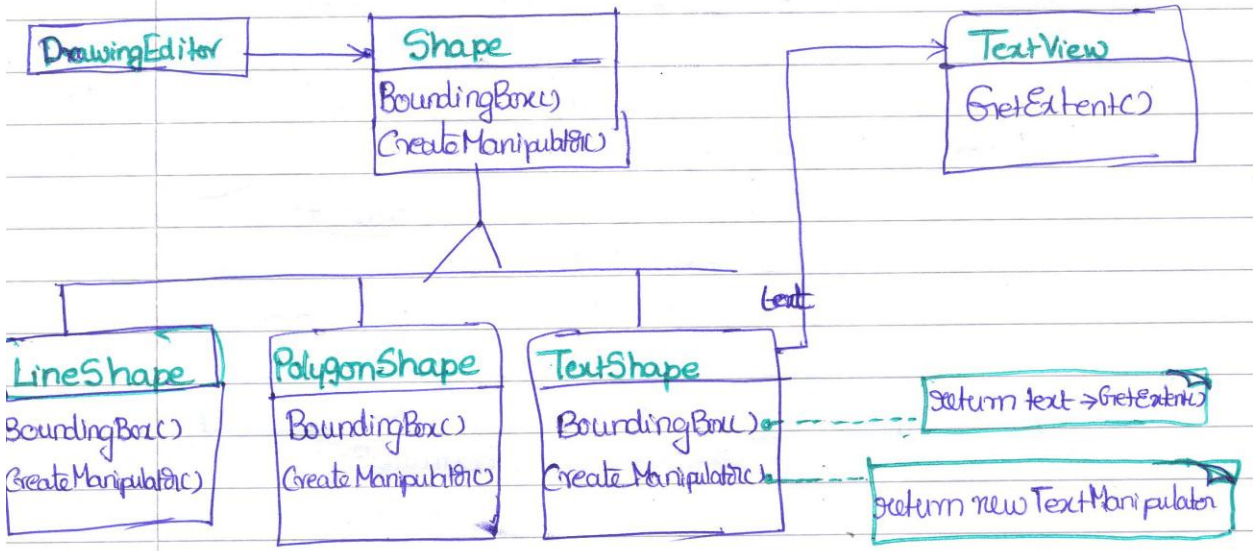


* An off-the-shelf user interface toolkit might already provide a sophisticated **TextView** class for displaying and editing text.

* Reusing **TextView** class to implement **TextShape** class:-
There are two ways
(i) By composing a **TextView** instance within a **TextShape** and implementing **TextShape** in terms of **TextView**'s interface — Object Version

(ii) By inheriting Shape's interface and TextView's implementation — class version of the Adapter pattern

* The following diagram illustrates object Adapter case:



- It shows how BoundingBox requests, declared in class Shape are converted into GetExtent requests defined in TextView.
- Since TextShape adapts TextView to the Shape interface, the drawing editor can reuse incompatible TextView class

Note:- Manipulator is an abstract class for objects that know how to animate a Shape in respose to user input, like dragging the shape to new location.
There are subclasses of Manipulator for different shapes; Eg:- TextManipulator is the subclass of TextShape.

Applicability :- Adapter pattern can be used,

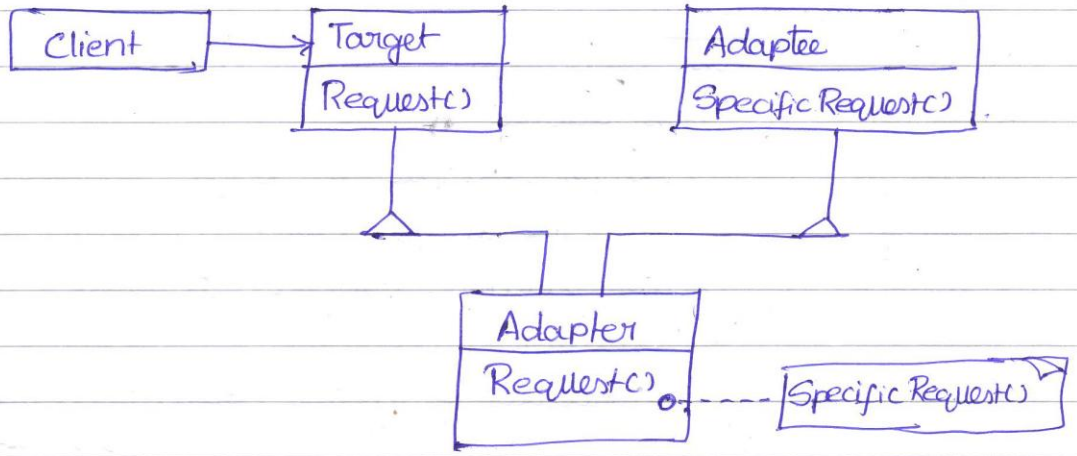
(i) when we want to use an existing class, and its interface doesn't match the one we need.

(ii) when we want to create a reusable class that cooperates with unrelated or unforeseen con having incompatible interfaces

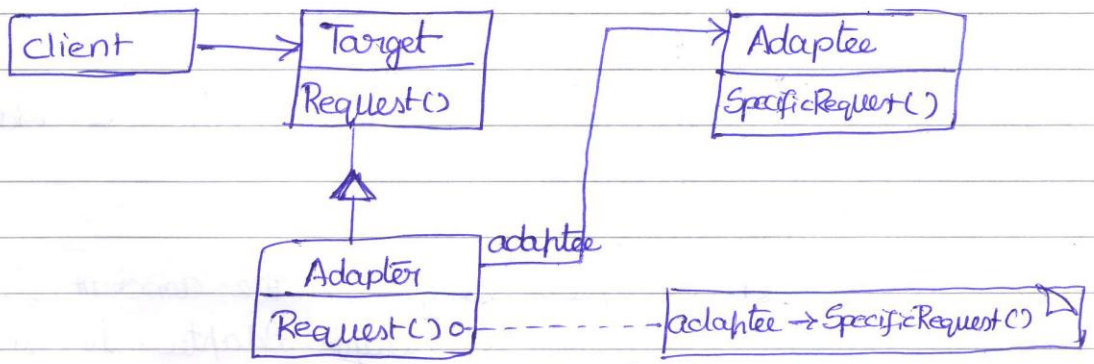
(iii) when there are several subclasses, but its impractical to adapt their interface by subclassing everyone. An object adapter can adapt the interface of its parent class.

Structure :

(i) Class adapter – A class adapter uses multiple inheritance to adapt one interface to another.



(ii) Object adapter – An object adapter relies on object composition :



Participants

(i) Target (Shape) – defines the domain-specific interface the client uses.

(ii) Client (DrawingEditor) – collaborates with objects conforming to the Target interface

(iii) Adaptee (TextView) – Defines an existing interface that needs adapting

(iv) Adapter (TextShape) – Adapts the interface of Adaptee to the Target interface

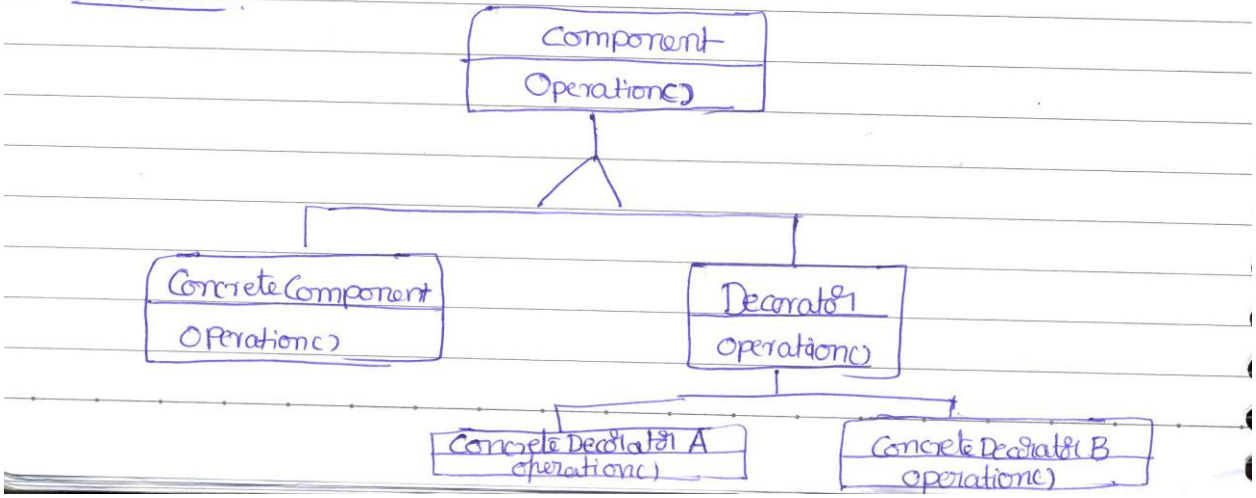
Collaborations

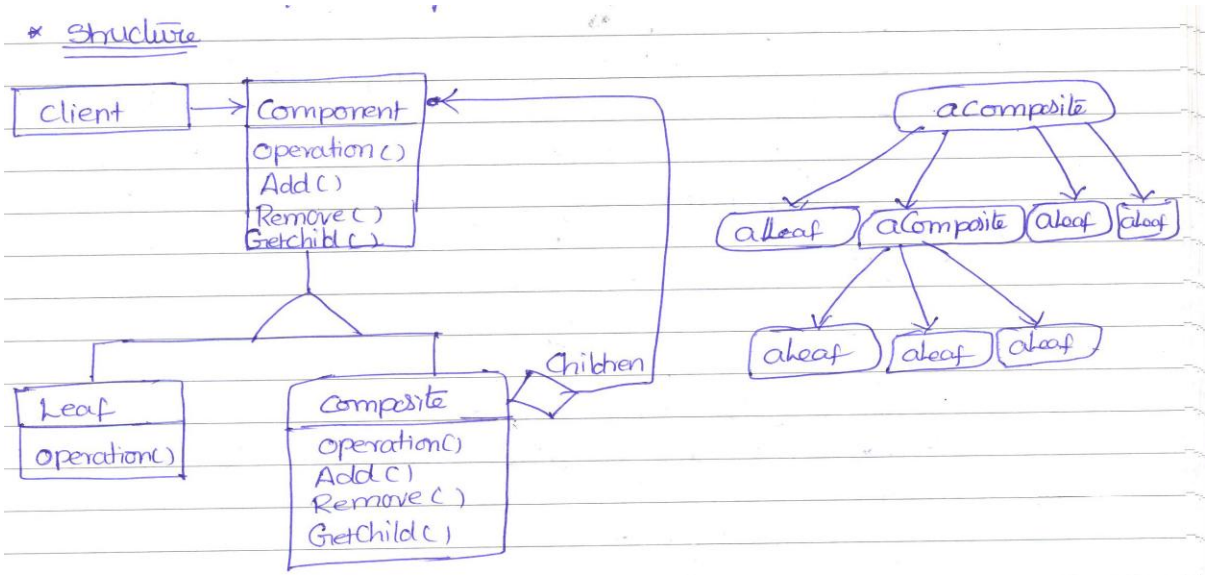
Clients call operations on an Adapter instance. In turn, the adapter calls Adaptee operations that carry out the request.

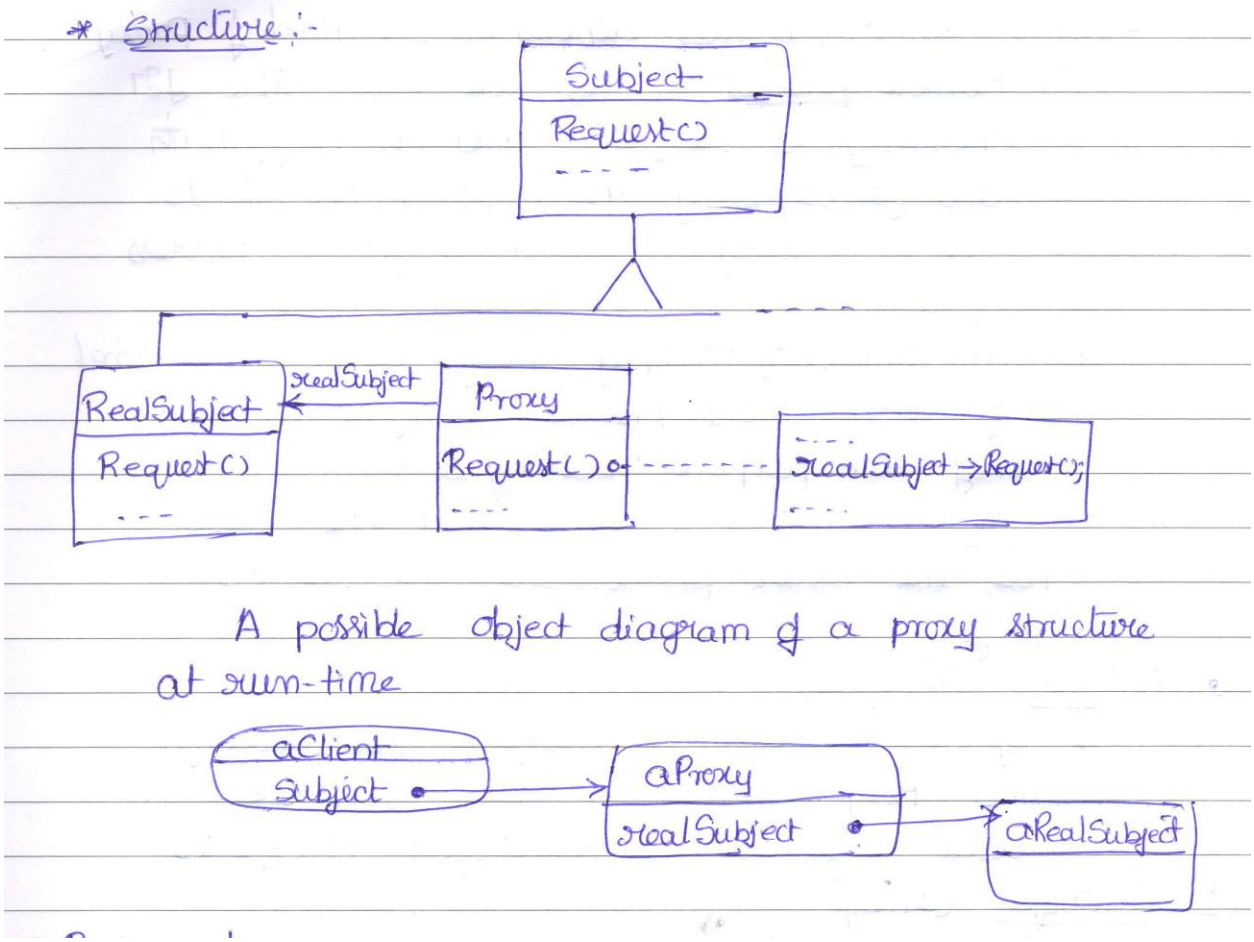
2. With a neat illustrations, explain the **structure** of the following patterns :
   a) Decorator b) Proxy c) Composite

Ans.

Structure







3. Write a short notes on:
   a) Bridge Pattern b) Façade Pattern

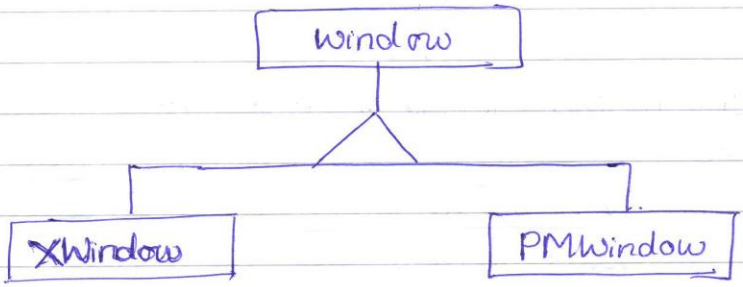
Bride pattern:

Intent -

- Bridge is used when we need to decouple an abstraction from its implementation so that the two can vary independently.

Motivation :-

- Consider the domain of "Portable Window abstraction"



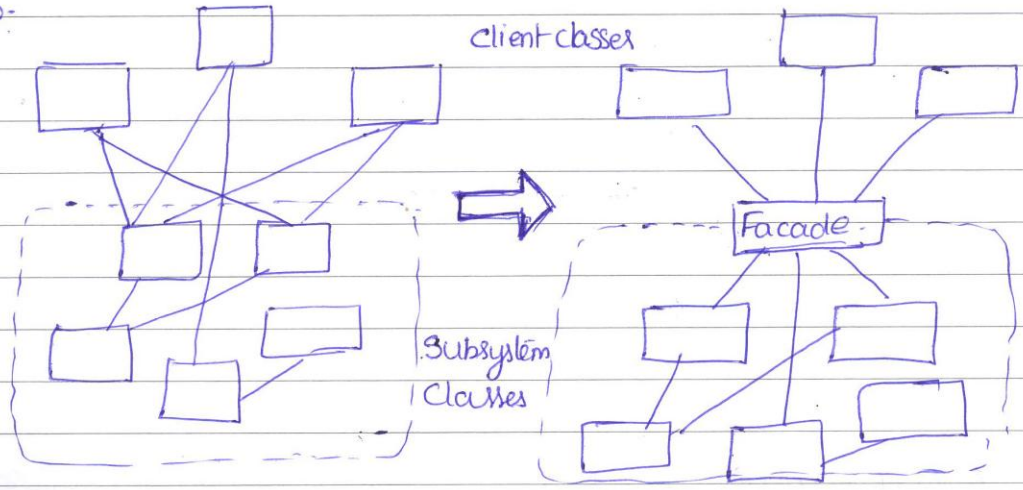
- This abstraction enables to write applications that work on XWindow & IBM's Presentation Manager (PM).
- This is implemented by defining an abstract class Window and subclasses XWindow and PMWindow using inheritance.

- Drawbacks of this approach:
  (i) Its inconvinient to extend the Window abstraction to cover different kinds of windows on new platforms.
  Eg :- • An IconWindow subclass of Window that specializes the Window abstraction for icons.
  • To support IconWindows for both platforms, we have to implement two new classes, XIconWindow and PMIconWindow.
  • conclusion - we have to define two classes for every kind of window.

Façade pattern:

* Intent:
  - Provide a unified interface to a set of interfaces in a subsystem.
  - Facade defines a higher-level interface that makes the subsystem easier to use.

* Motivation:
  - Structuring a system into subsystems helps reduce complexity.
  - Clients of a subsystem may need to interact with a number of subsystem classes for their needs. This leads to a high degree of coupling b/w the client objects and the subsystems.
  - Facade pattern provides a higher level, simplified interface for a subsystem resulting in reduced complexity & dependency.
  - Clients interact with Facade object to deal with the subsystem instead of interacting directly with subsystem classes.



  - Eg:- A programming environment that gives applications access to its compiler subsystem. Subsystem classes — Parser, Scanner, ProgramNode, BytecodeStream, and ProgramNodeBuilder, that implement compiler.

4. Describe how **object-oriented system** is implemented on the **World-Wide-Web for Library Management System**

12.3 Implementing an Object-Oriented System on the Web

Without doubt, the world-wide web is the most popular medium for hosting distributed applications. Increasingly, people are using the web to book airline tickets, purchase a host of consumer goods, make hotel reservations, and so on. The browser acts as a general purpose client that can interact with any application that talks to it using the Hyper Text Transfer Protocol (HTTP).

One major characteristic of a web-based application system is that the client (the browser), being a general-purpose program, typically does no application-related computation at all. Of course, it is possible to ship a Java applet with a web page and have the applet do some computation, but this is not hugely popular. All business logic and data processing take place at the server. Typically, the browser receives web pages from the server in HTML and displays the contents according to the format, a number of tags and values for the tags, specified in it. In this sense, the browser simply acts as a 'dumb' program displaying whatever it gets from the application and transmitting user data from the client site to the server.

The HTML program shipped from a server to a client often needs to be customised: the code has to suit the context. For example, when we make a reservation on a flight, we expect the system to display the details of the flight on which we made the reservation. This requires that HTML code for the screen be dynamically constructed. This is done by code at the server.

For server-side processing, there are competing technologies such as Java Server Pages and Java Servlets, Active Server Pages (ASP), and PHP. In this book we study Java Servlets.

12.3.2 Deploying the library system on the world-wide web

We now undertake the task of designing and developing a web-based version of the library system. Of course, we cannot do everything exactly as in a real library: in particular, we do not have machines that scan bar codes on books, but we will do as close a job as possible as a real system.

Developing user requirements

As in any system, the first task is to determine the system requirements. We will, as has been the case throughout the book, restrict the functionality so that the system's size is manageable.

1. The user must be able to type in a URL in the browser and connect to the library system.

2. Users are classified into two categories: *superusers* and *ordinary members*. Superusers are essentially designated library employees, and ordinary members are the general public who borrow library books. The major difference between the two groups of users is that superusers can execute any command when logged in from a terminal in the library, whereas ordinary members cannot access some 'privileged commands'. In particular, the division is as follows:

   (a) Only superusers can issue the following commands: add a member, add a book, return a book, remove a book, process holds, save data to disk, and retrieve data from disk.

   (b) Ordinary members and superusers may invoke the following commands: issue and renew books, place and remove holds, and print transactions.

   (c) Every user eventually issues the exit command to terminate his/her session.

3. Some commands can be issued from the library only. These include all of the commands that only the superuser has access to and the command to issue books.

4. A superuser cannot issue any commands from outside of the library. They can log in, but the only command choice will be to exit the system.

5. Superusers have special user ids and corresponding password. For regular members, their library member id will be their user id and their phone number will be the password.

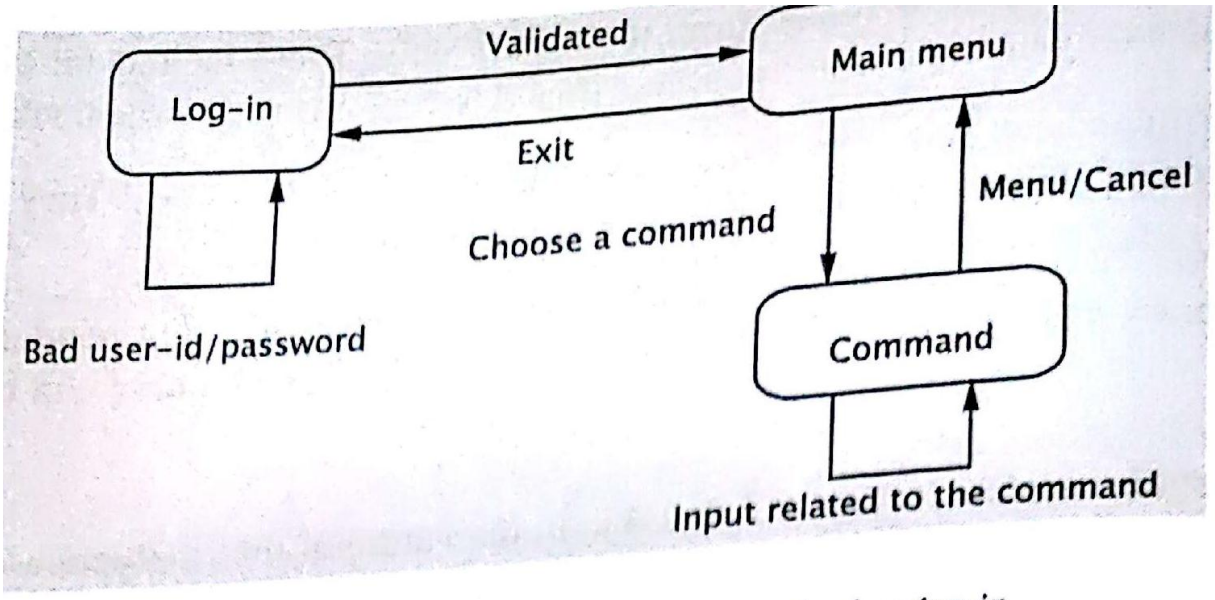


Figure 12.8 *State transition diagram for logging in*

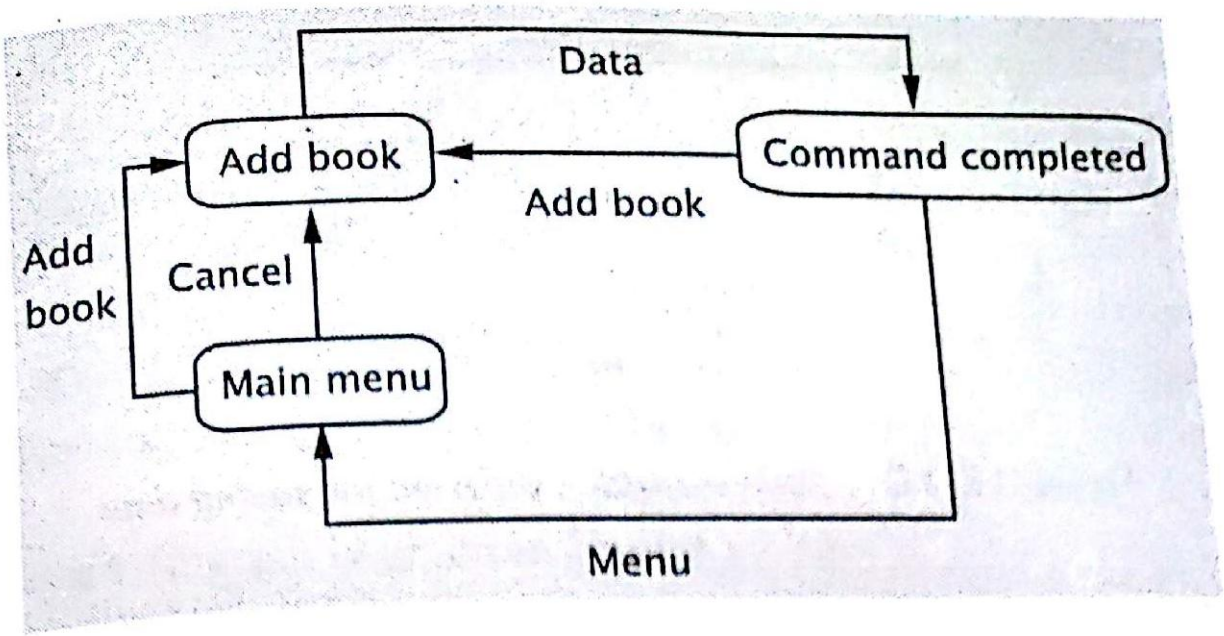


Figure 12.9 *State transition diagram for add book*

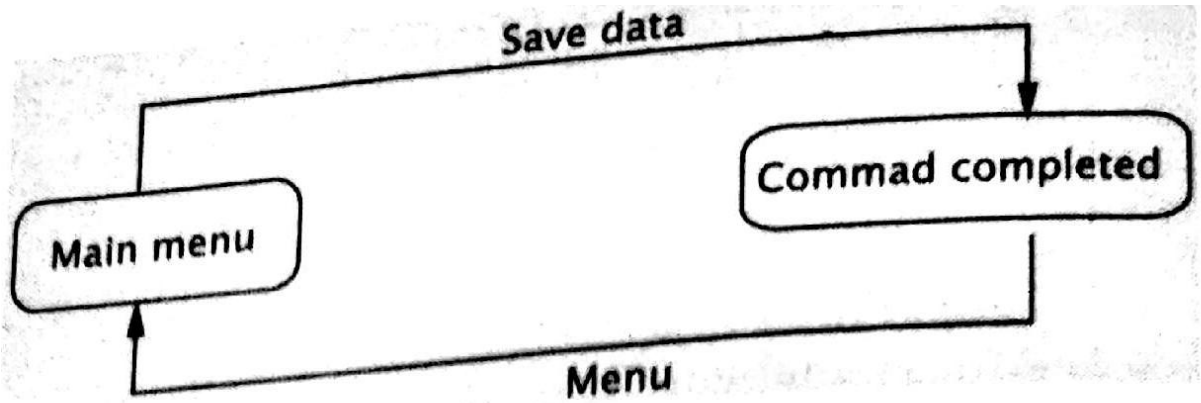


Figure 12.10 *State transition diagram for saving data*

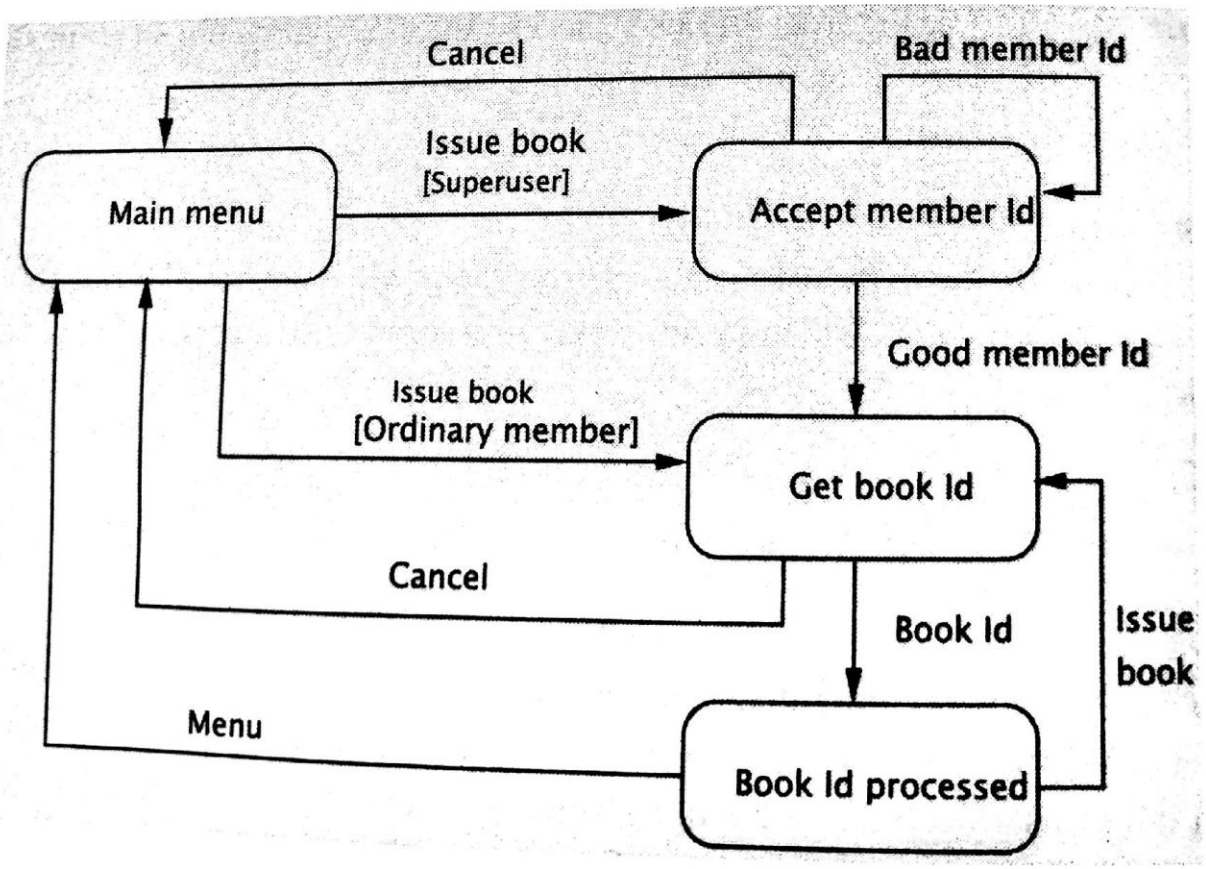


Figure 12.11 *State transition diagram for issuing books*

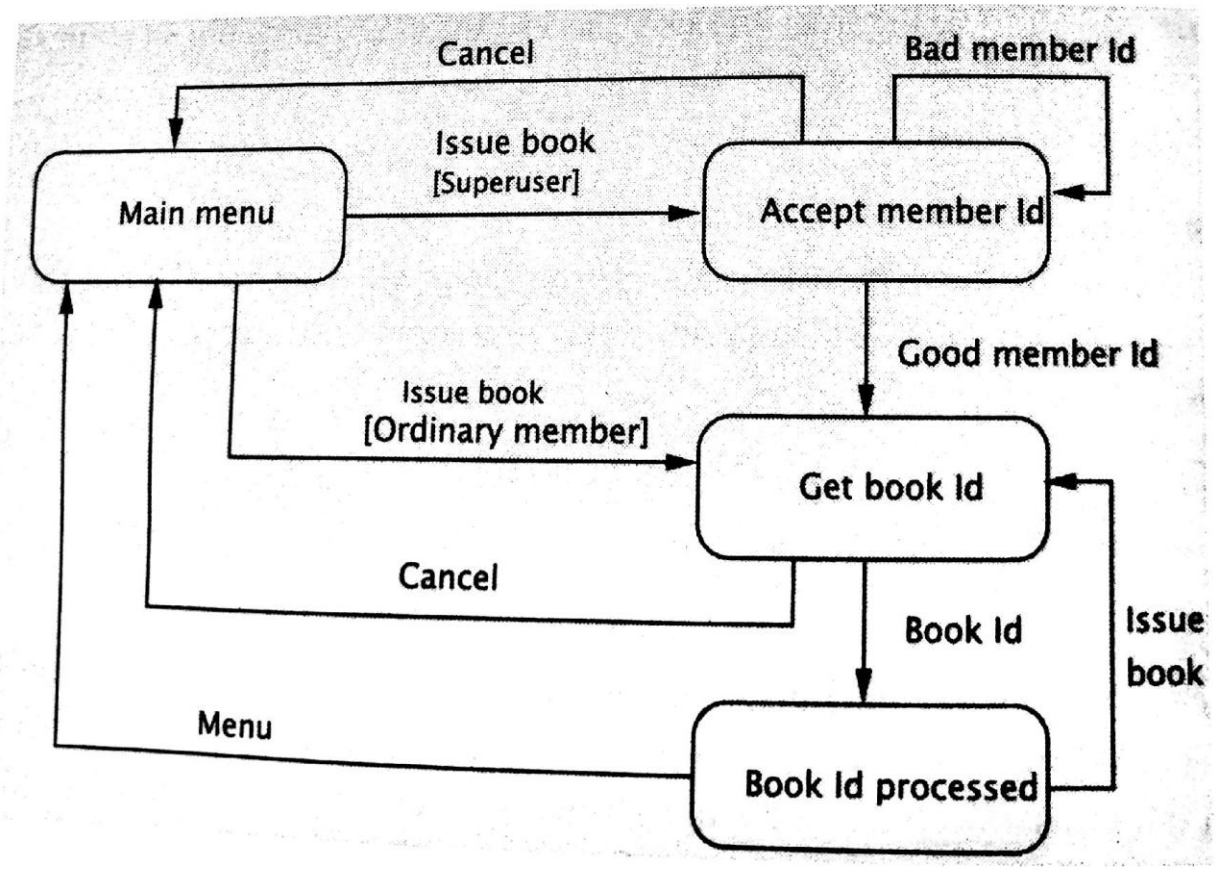


Figure 12.11 *State transition diagram for issuing books*

5. List and explain the necessary steps for hosting **distributed applications**

1. Define the functionality that must be made available to clients. This is accomplished by creating **remote interfaces**.
2. Implement the remote interfaces via **remote classes**.
3. Create a server that serves the remote objects.
4. Set up the client.

Figure 12.3 *Using a proxy*

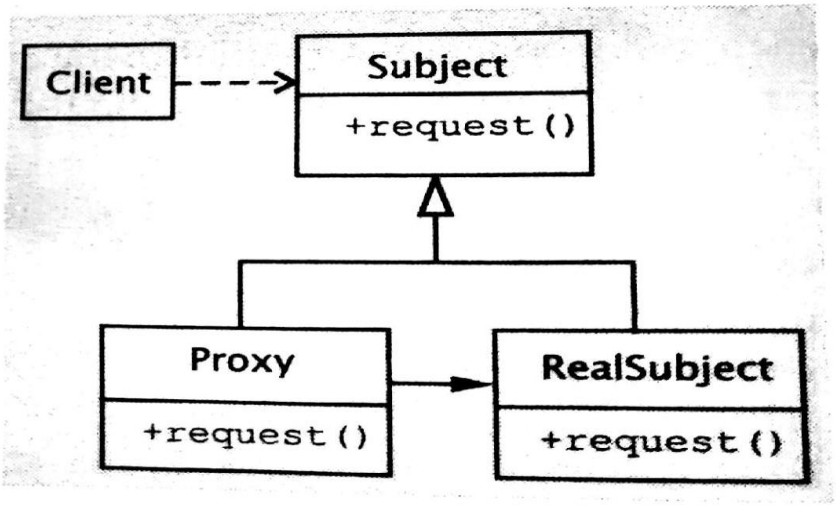


Figure 12.4 *Client/Server systems*

6. Write short notes on:
a) **Marshalling** & **Demarshalling** b) **GET** or **POST**

Marshalling & Demarshalling:

- **Marshalling** is the process of taking a collection of data items and assembling them into a form suitable for transmission in a message.
- **Unmarshalling** is the process of disassembling them on arrival to produce an equivalent collection of data items at the destination.
- Thus marshalling consists of the translation of structured data items and primitive values into an external data representation.
- Similarly, unmarshalling consists of the generation of primitive values from their external data representation and the rebuilding of the data structures.