Fundamentals of Java Enterprise Components

Session: 2

Enterprise Application Architecture

Objectives

- Explain enterprise application design
- Use various design patterns in application design
- Describe Model-View-Controller design
- Explain communication among application components
- Describe network topologies and clustering
- Describe the layered architecture of application design

Application Architecture

Application architecture has evolved in the following ways:

- **Client Server Architecture** - This has a client tier and server tier where a client request for service/data from the server and the server responds.
- **Three tier Architecture** - This has an additional middle tier which accepts requests from different types of clients and manages the data flow between the database and different types of clients.
- **N-tier Architecture** - Has more granularity. This architecture includes elements of Security, Web, and so on, which are independently developed and made to function together with each other.

Designing an Enterprise Application

Following are the steps involved in designing an Enterprise Application:

- Identify the objects in the domain
- Identify the interactions among the objects
- Create UML diagrams
- Multi-layered design is used for large scalable applications

Designing Multi-Layered Applications

- Various components of the applications are placed in different layers and assigned responsibilities
- These components are loosely coupled
- This architecture makes the application scalable as any number of components can be added without effecting the existing components
- Application development time is reduced by reuse of code
- Using design patterns for developing the solution also reduces the application development time

Design Patterns

- Encourage reuse of solution for certain pattern of objects and their interactions in a domain
- Identify a pattern in the problem based on the classes, objects, collaborations, and distribution of responsibilities
- Offer a tried and tested solution for the problem based on the pattern identified
- The pattern is identified based on data about the application which is organized as a template

Design Patterns Template 1-2

- **Pattern name and classification** – This is a conceptual category for the pattern
- **Intent** – Describes the type of problems this particular pattern can solve
- **Also known as** – Implies other common names of the pattern
- **Motivation** – A scenario which illustrates the problem
- **Applicability** – Describes the situations where the current pattern can be used
- **Structure** – Diagram using an object modelling technique such as UML
- **Participants** – Implies the classes and objects in the design
- **Collaborations** – Defines how the classes and objects in the domain collaborate with each other

Design Patterns Template 2-2

- **Consequences** – Describes the outcome of using this pattern and also describes any kind of side effects it may lead to
- **Implementation** – Describes the implementation specific or language specific issues that might arise on using this design pattern
- **Sample code** – Provides an example of the design pattern created in languages such as Java or any other object-oriented language and enables the developer to translate the code with similar classes and objects
- **Known uses** – Illustrates some real world scenarios where this design pattern has been successfully put to use earlier
- **Related patterns** – Provides comparison with similar patterns so that the developer can choose the right pattern for his/her application

Types of Design Patterns

Creational Patterns

- Defines the object creation process in the application

Structural Patterns

- Defines the organization of classes and objects in an application

Behavioral Patterns

- Defines the interaction between various objects in the application

Concurrency Patterns

- Defines the access of resources by objects of the application

Creational Patterns

Describe how and what kind of objects are created in the application.

Used in applications where the composition of the object instantiated is crucial for the functionality of the application.

There are two variants of creational patterns:

- Singleton pattern
- Abstract factory pattern

Singleton Pattern 1-2

- Allows instantiation of only one object for the class.
- Implemented by creating a class with a method which creates a new instance of the class only when there is no other instance of the class existing.
- The constructor of such a class is made `private` so that the object of the class cannot be instantiated in any other manner.

Singleton Pattern 2-2

Following code demonstrates usage of singleton pattern:

```
public class SingleInstance {
// declare an instance of the class
private static SingleInstance inst = null;
// set constructor as private
private SingleInstance() {}
// define a synchronized method for creating the instance
  public static synchronized SingleInstance getInstance() {
       // check for existence of instance
     if (inst == null) {
       // create instance if it does not exist
     inst = new SingleInstance();
     }
    return inst;  // return the instance
  }
}
```

Structural Patterns

- Define the organization of different classes in the solution
- Define the usage of mechanisms such as inheritance and polymorphism
- Enable multiple classes and objects to function together
- Adapter, proxy, and decorator patterns are commonly used structural patterns

Adapter Patterns 1-2

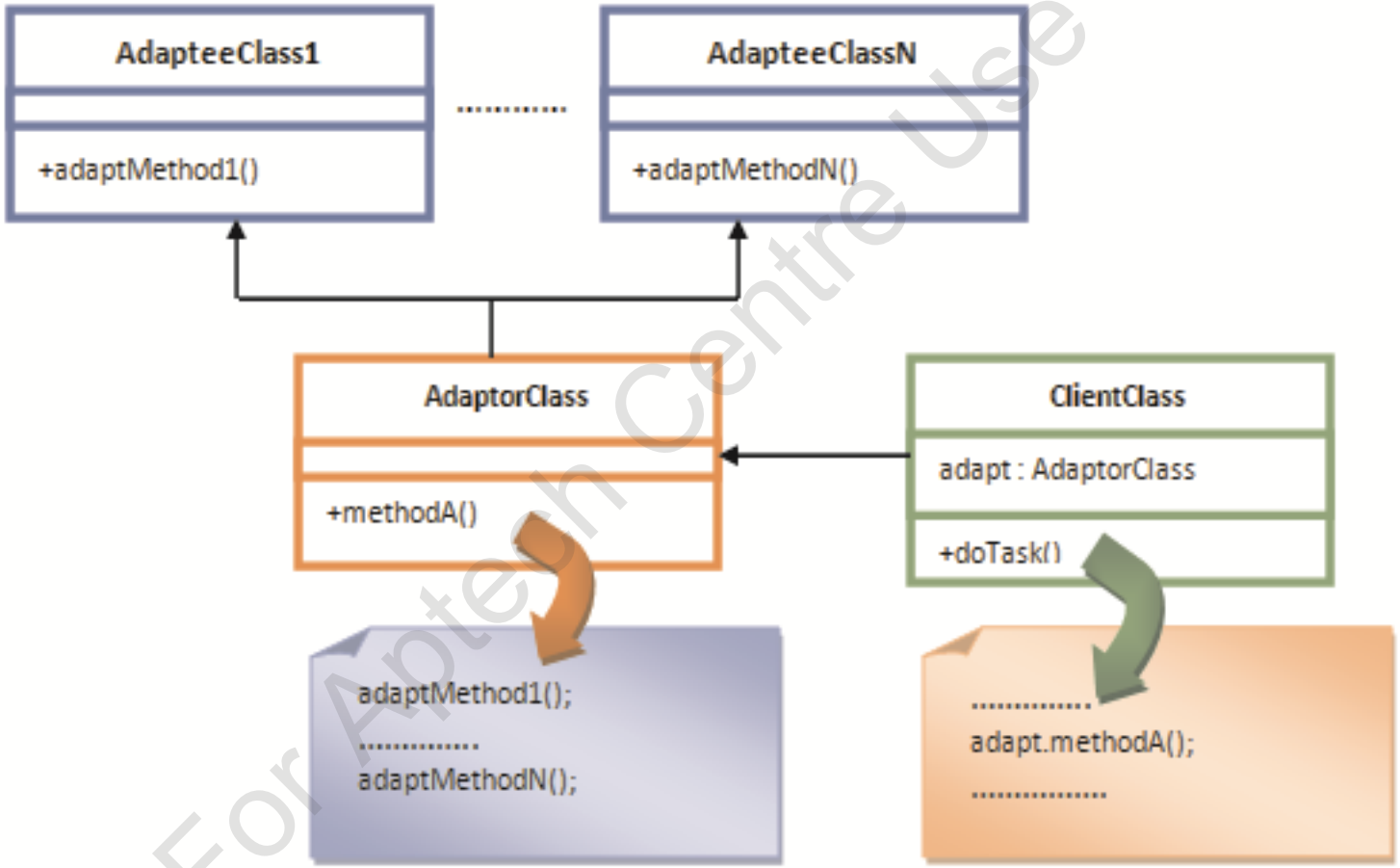
An adapter pattern is used in the following situations:

When the developer needs to use an existing class and its interface does not exactly match what is required by the developer

When the developer has to create a reusable class which has to function along with other incompatible classes

Adapter Patterns 2-2

- Following figure demonstrates Adapter patterns:



Behavioral and Concurrency Patterns

Behavioral Patterns

- Behavioral patterns are those that determine the interaction of objects
- They simplify the complex behavior by specifying the responsibilities of the objects and their communication method
- Most common behavioral pattern is Observer pattern
- Separates the presentation of data from the actual data store

Concurrency Patterns

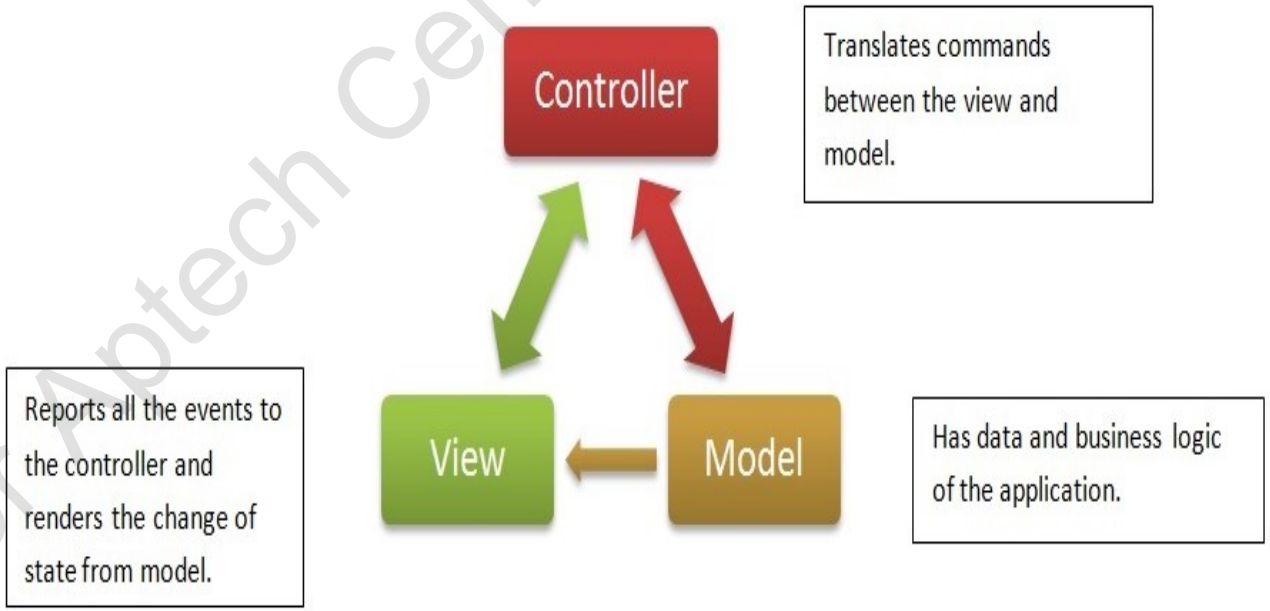
- Concurrency patterns are applied in cases where there are certain shared resources
- Assures consistent and coordinated access of resources
- Single thread execution is the most common Concurrency pattern
- Active object, double-checked locking pattern, and reactor pattern are other Concurrency patterns

Model-View-Controller

Model-View-Controller is a methodology of designing user interfaces for applications.

- View refers to the perspective of an end user of the application
- Model refers to the context specific processing done by the application
- Controller accepts data from the user interface and converts it into a format understandable by the Model

Following figure shows the interactions in Model-View-Controller architecture:



MVC Example 1-4

Following code provides an example of Model-View-Controller:

```
package mvcpkg;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class MVCdemo {
   // initialize the GUI controls
   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;
   public MVCdemo(){
      prepareGUI();  // invoke prepareGUI method
   }
   public static void main(String[] args){
     // instantiate the class
      MVCdemo objMVC = new MVCdemo();
      objMVC.clickOK();
   }
// creates the GUI
   private void prepareGUI(){
```

MVC Example 2-4

```
// design the GUI
      mainFrame = new JFrame("To Demonstrate MVC");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      headerLabel = new JLabel("",JLabel.CENTER );
      statusLabel = new JLabel("",JLabel.CENTER);
      statusLabel.setSize(350,100);
      // add listener to the frame
      mainFrame.addWindowListener(new WindowAdapter() {
         @Override
         public void windowClosing(WindowEvent windowEvent){
                  System.exit(0);
         }
      });
      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());
      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }
```

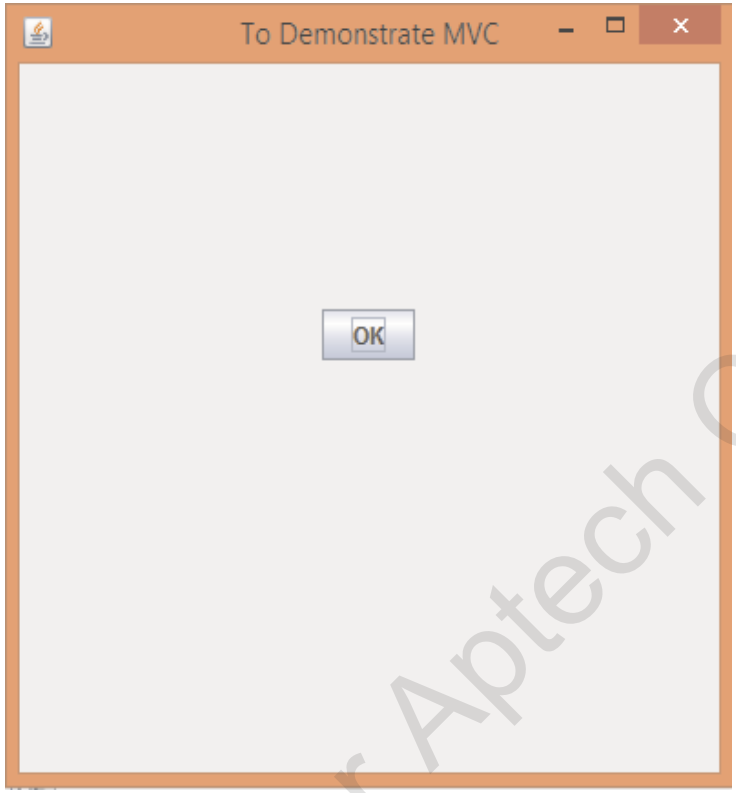
MVC Example 3-4

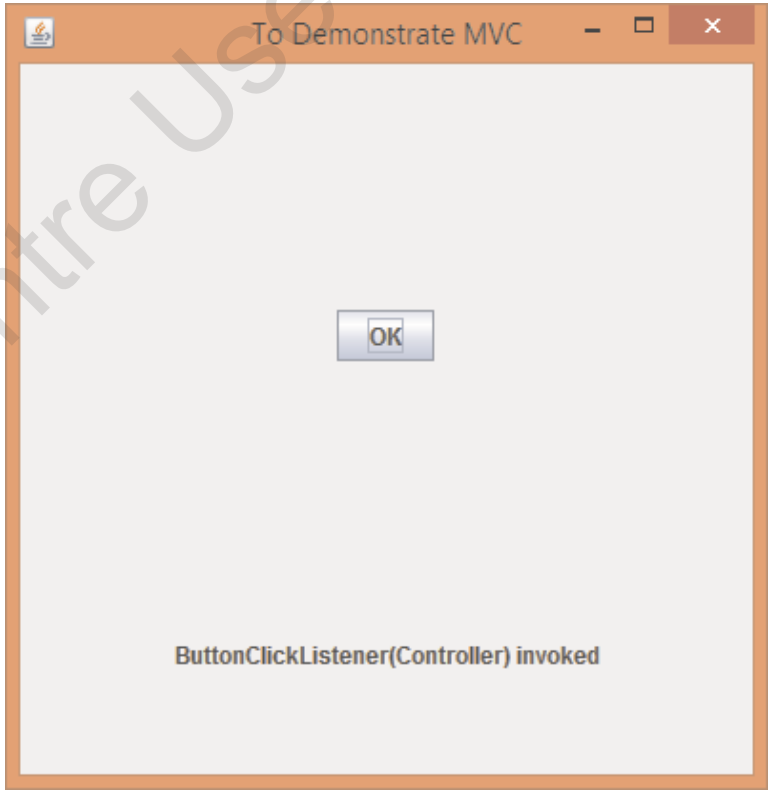
```
// handles the click of OK button
   private void clickOK(){
     JButton okButton = new JButton("OK");
      okButton.setActionCommand("OK");
      // add listener to the OK button
      okButton.addActionListener(new ButtonClickListener());
      controlPanel.add(okButton);
      mainFrame.setVisible(true);
   }
private class ButtonClickListener implements ActionListener{
        @Override
      public void actionPerformed(ActionEvent e) {
         String command = e.getActionCommand();
         if(command.equals( "OK" ))  {
            statusLabel.setText("ButtonClickListener(Controller) invoked");
         }
      }
   }

}
```

MVC Example 4-4

Following figure shows the output of the code:





Synchronous/Asynchronous Communication

For efficient collaboration of different components of an application, the communication between them plays an important role.

The communication among components of an application can be synchronous or asynchronous.

- Synchronous communication is one where a request is sent and the sender waits for the response from the receiver before sending the next request
- Asynchronous communication is one where the request is sent but the sender does not wait for response from the receiver
- Remote Method Invocation (RMI)/Remote Procedure Call (RPC) in Java is an implementation of synchronous communication
- Java Messaging Service (JMS) is an implementation of asynchronous communication

Clustering 1-2

Clustering refers to a technique where an application is deployed over a group of servers which is termed as a cluster of servers.

Clustering technique is important to scale the application across multiple users and make the application highly available.

Essential services and components such as JNDI, EJB, JSP, and so on run on a cluster to make it highly available.

Following are the various purposes of clusters:

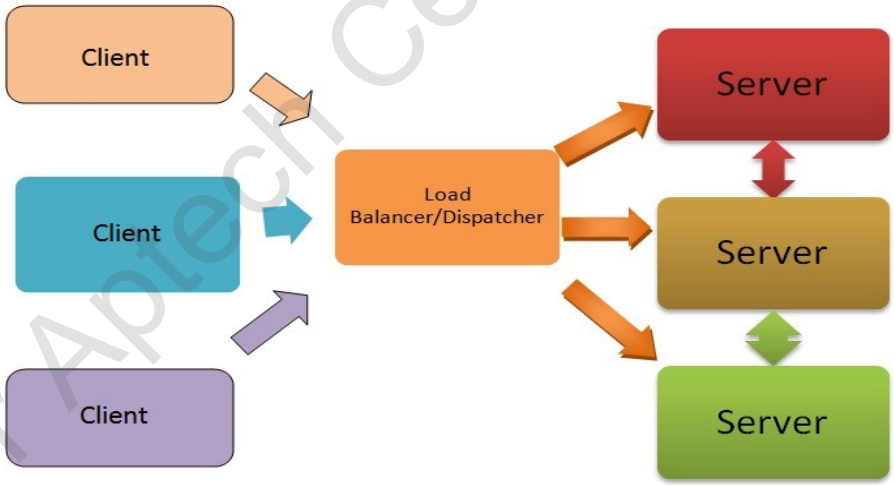
- Load balancing
- Fail over

In case of Web applications:

- Web load balancing
- HTTP fail over

Clustering 2-2

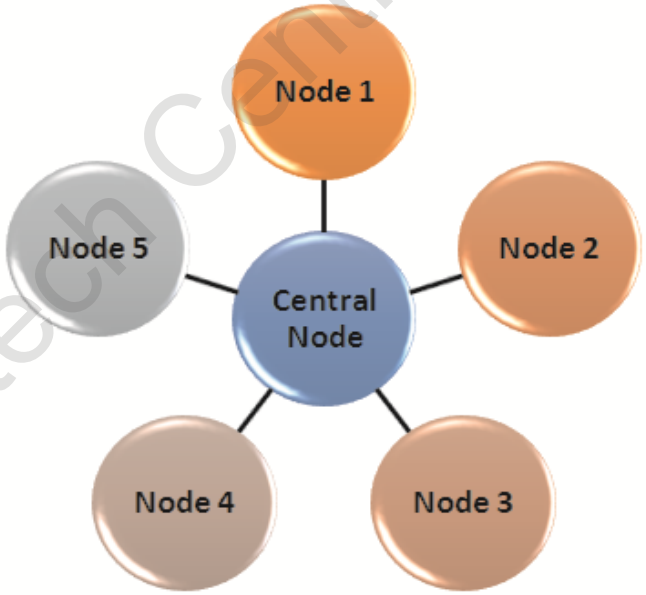
- Load balancing is a process wherein if a server is overwhelmed with client requests, certain requests are serviced by another server. A load balancer manages the client request and maps it onto appropriate server.
- Fail over of server refers to a situation of server crash.
- A dispatcher is responsible for redirecting the requests of a crashed server to another server.
- Following figure shows clustering of servers:



Network Topologies 1-4

Star Topology

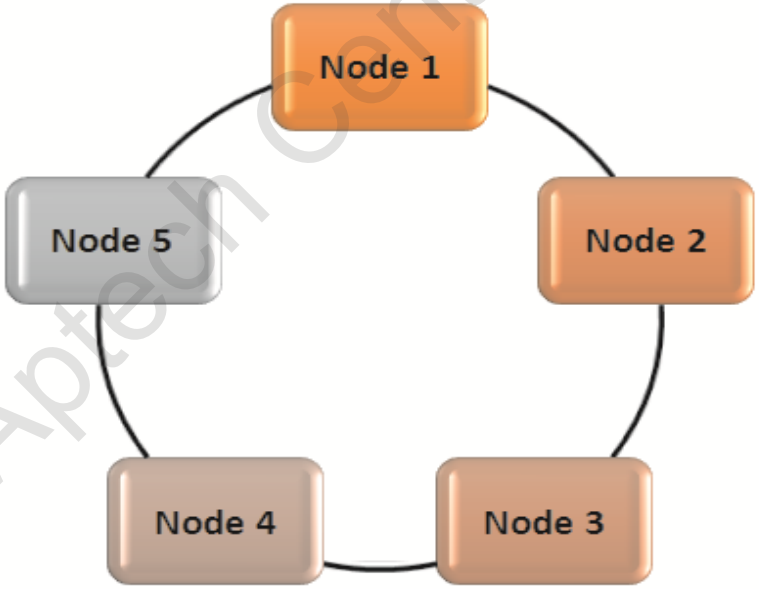
- In Star topology, every node is connected to a central node called a hub or switch which is the server and the peripherals are the clients.
- Following figure depicts the Star topology:



Network Topologies 2-4

Ring Topology

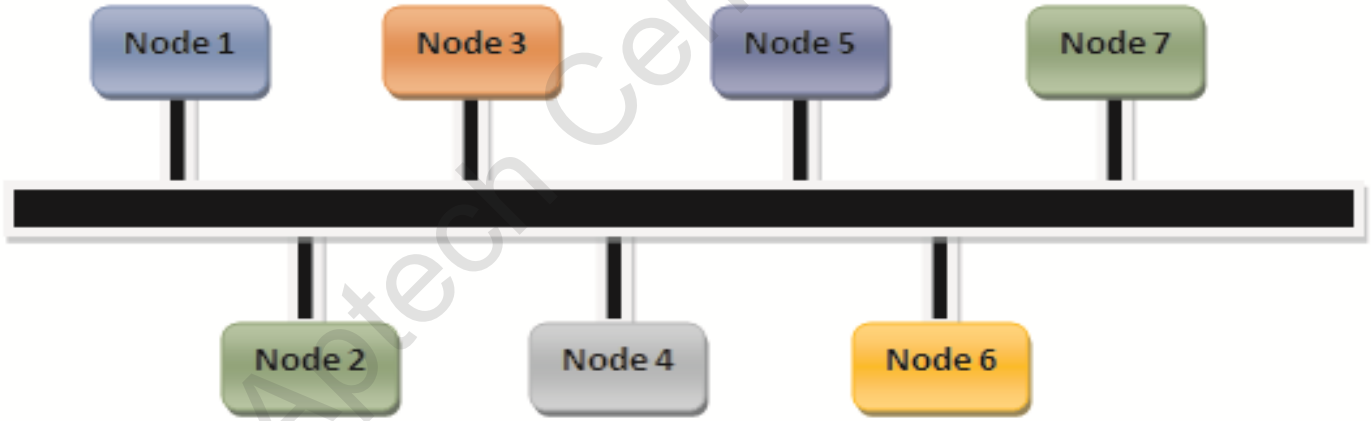
- In Ring topology, each node is connected to exactly two other nodes with the last node connecting to the first node forming a circular ring.
- Following figure depicts the Ring topology:



Network Topologies 3-4

Bus Topology

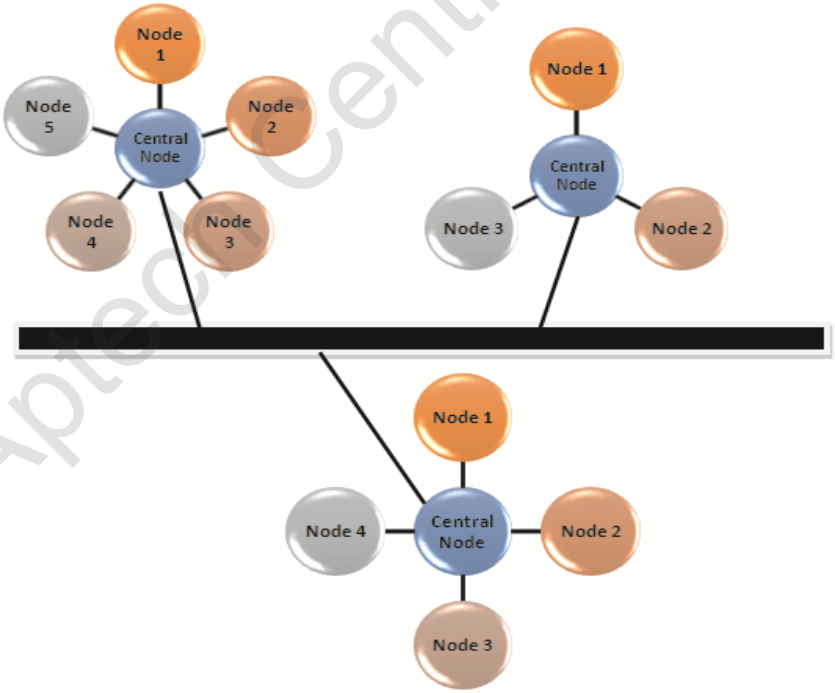
- In Bus topology, all the nodes are connected in a daisy chain in a linear sequence of buses. The host on a bus network is known as a workstation.
- Following figure depicts the Bus topology:



Network Topologies 4-4

Tree Topology

- A Tree topology follows a hybrid approach by combining the Star and Bus topologies to improve network scalability.
- All the servers in the cluster are connected like a tree.
- The tree structure can be a hierarchical or a non-hierarchical tree.
- Following figure depicts the Tree topology:



Layering in Java Applications 1-3

Layering refers to logical separation of the components.

Java applications can have the following layers:

- Client layer
- Presentation layer
- Business logic layer
- Data access layer (Persistence)
- Integration layer
- Services layer

Layering in Java Applications 2-3

Client Layer

- Client layer comprises all those entities that access the application.
- The clients can be browser clients, application container clients, CORBA (Common Object Request Broker Architecture) clients, JMS clients, and Web service clients.

Presentation layer

- Presentation layer generates the user interface for the client.
- Servlets, JSPs (Java Server Pages), HTML, CGI, and so on are used in presentation layer.

Business logic layer

- This layer is responsible for the implementation of business logic of the application.
- The components that are used for defining the business logic are session beans, entity beans, and message driven beans.

Layering in Java Applications 3-3

Data access layer (Persistence)

- Data access layer or persistence layer deals with persistent data objects such as database rows.

Integration layer

- Integration layer enables independent components of the application to function together.

Services layer

- The services layer comprises services supporting business functionality.

Summary

- Application design process in Java is object-oriented and uses design patterns based on the requirements and other attributes of the application.
- The various design patterns such as creational, behavioral, structural, and concurrency are based on the context of the application, components, and so on.
- Model-View-Controller architecture separates the presentation and the logic of the application which enables independent implementation of these components during application development.
- Asynchronous and synchronous communication are two different communication methodologies used to allow independently developed components to work together.
- In Star topology, there is a central server/router which connects to all other servers in the network cluster.
- Layering of the Java applications is essential to logically divide the functions of Java components. The implementation of these layers can be independent or combined based on the application requirement.
- The services layer comprises services supporting business functionality such as data access policy, security services through policy management tools, and cryptographic services for the application.