(2½ hours)

Total Marks: 75

N. B.: (1) **All** questions are **compulsory**.
(2) Make **suitable assumptions** wherever necessary and **state the assumptions** made.
(3) Answers to the **same question** must be **written together**.
(4) Numbers to the **right** indicate **marks**.
(5) Draw **neat labeled diagrams** wherever **necessary**.
(6) Use of **Non-programmable** calculators is **allowed**.

1	**Attempt *any two* of the following:**	**10**
a	**Explain the use of adapter class with suitable example.** Java provides a special feature, called an adapter class that can simplify the creation of event handlers in certain situations. An adapter class provides an empty implementation of all methods in an event listener interface. Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface. You can define a new class to act as an event listener by extending one of the adapter classes and implementing only those events in which you are interested. For example, the MouseMotionAdapter class has two methods, mouseDragged( ) and mouseMoved( ), which are the methods defined by the MouseMotionListener interface. If you were interested in only mouse drag events, then you could simply extend MouseMotionAdapter and override mouseDragged( ). The empty implementation of mouseMoved( ) would handle the mouse motion events for you. The following example demonstrates an adapter. It displays a message in the status bar of an applet viewer or browser when the mouse is clicked or dragged. However, all other mouse events are silently ignored. The program has three classes. AdapterDemo extends Applet. Its init( ) method creates an instance of MyMouseAdapter and registers that object to receive notifications of mouse events. It also creates an instance of MyMouseMotionAdapter and registers that object to receive notifications of mouse motion events. Both of the constructors take a reference to the applet as an argument. MyMouseAdapter extends MouseAdapter and overrides the mouseClicked( ) method. The other mouse events are silently ignored by code inherited from the MouseAdapter class. MyMouseMotionAdapter extends MouseMotionAdapter and overrides the mouseDragged( ) method. The other mouse motion event is silently ignored by code inherited from the MouseMotionAdapter class. Note that both of the event listener classes save a reference to the applet. This information is provided as an argument to their constructors and is used later to invoke the showStatus( ) method. // Demonstrate an adapter. import java.awt.*; import java.awt.event.*; import java.applet.*; /* <applet code="AdapterDemo" width=300 height=100> </applet> */ public class AdapterDemo extends Applet { public void init() { addMouseListener(new MyMouseAdapter(this)); addMouseMotionListener(new MyMouseMotionAdapter(this));	

```
}
}
class MyMouseAdapter extends MouseAdapter {
AdapterDemo adapterDemo;
public MyMouseAdapter(AdapterDemo adapterDemo) {
this.adapterDemo = adapterDemo;
}
// Handle mouse clicked.
public void mouseClicked(MouseEvent me) {
adapterDemo.showStatus("Mouse clicked");
}
}
class MyMouseMotionAdapter extends MouseMotionAdapter {
AdapterDemo adapterDemo;
public MyMouseMotionAdapter(AdapterDemo adapterDemo) {
this.adapterDemo = adapterDemo;
}
// Handle mouse dragged.
public void mouseDragged(MouseEvent me) {
adapterDemo.showStatus("Mouse dragged");
}
}
```

As you can see by looking at the program, not having to implement all of the methods defined by the MouseMotionListener and MouseListener interfaces saves you a considerable amount of effort and prevents your code from becoming cluttered with empty methods. As an exercise, you might want to try rewriting one of the keyboard input examples shown earlier so that it uses a KeyAdapter.

b **Explain Delegation Event model. What are two steps in using the java delegation event model?**

The modern approach to handling events is based on the delegation event model, which defines standard and consistent mechanisms to generate and process events. Its concept is quite simple: a source generates an event and sends it to one or more listeners. In this scheme, the listener simply waits until it receives an event. Once received, the listener processes the event and then returns. The advantage of this design is that the application logic that processes events is cleanly separated from the user interface logic that generates those events. A user interface element is able to “delegate” the processing of an event to a separate piece of code.

In the delegation event model, listeners must register with a source in order to receive an event notification. This provides an important benefit: notifications are sent only to listeners that want to receive them .Previously, an event was propagated up the containment hierarchy until it was handled by a component. This required components to receive events that they did not process, and it wasted valuable time.

The delegation event model eliminates this overhead.

1) Events

In the delegation model, an event is an object that describes a state change in a source. It can be generated as a consequence of a person interacting with the elements in a graphical user interface. Some of the activities that cause events to be generated are pressing a button, entering a character via the keyboard, selecting an item in a list, and clicking the mouse. Many other user operations could also be cited as examples. Events may also occur that are not directly caused by interactions with a user interface.

For example, an event may be generated when a timer expires, a counter exceeds a value, software or hardware failure occurs, or an operation is completed.

2)Event Sources

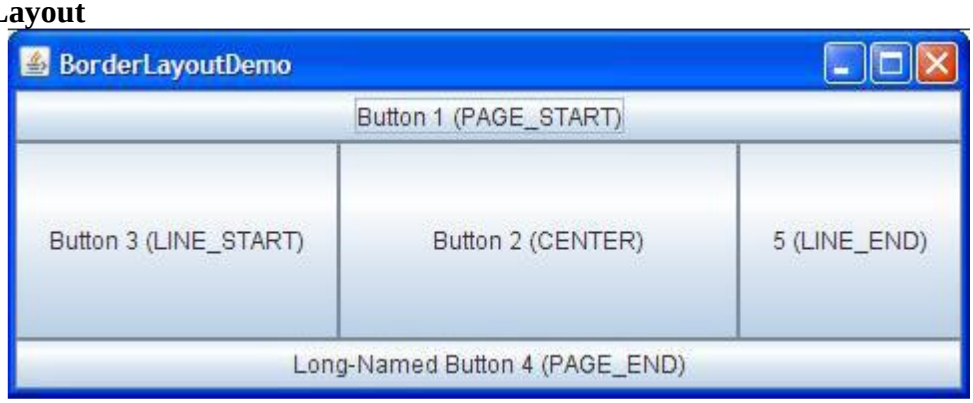
A source is an object that generates an event. This occurs when the internal state of that object changes in some way. Sources may generate more than one type of event.
A source must register listeners in order for the listeners to receive notifications about a specific type of event. Each type of event has its own registration method.

Here is the general form:
public void addTypeListener(TypeListener el)
Here, Type is the name of the event and el is a reference to the event listener. For example, the method that registers a keyboard event listener is called addKeyListener( ).

3) Event Listeners
A listener is an object that is notified when an event occurs. It has two major
2 requirements.
First, it must have been registered with one or more sources to receive notifications about specific types of events. Second, it must implement methods to receive and process these notifications.
Just follow these two steps:
Implement the appropriate interface in the listener so that it can receive the type of event desired.
Implement code to register and unregister (if necessary) the listener as a recipient for the event notifications.
A source may generate several types of events. Each event must be registered separately. Also, an object may register to receive several types of events, but it must implement all of the interfaces that are required to receive these events.

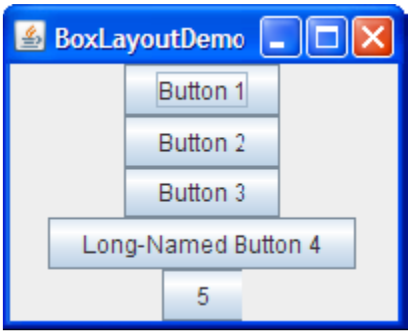
c **Explain the various Layout Managers available in AWT**

BorderLayout



Every content pane is initialized to use a BorderLayout. A BorderLayout places components in up to five areas: top, bottom, left, right, and center. All extra space is placed in the center area. Tool bars that are created using JToolBar must be created within a BorderLayout container, if you want to be able to drag and drop the bars away from their starting positions.

BoxLayout

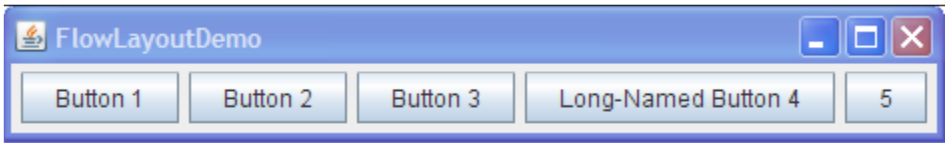


The BoxLayout class puts components in a single row or column. It respects the components' requested maximum sizes and also lets you align components

CardLayout

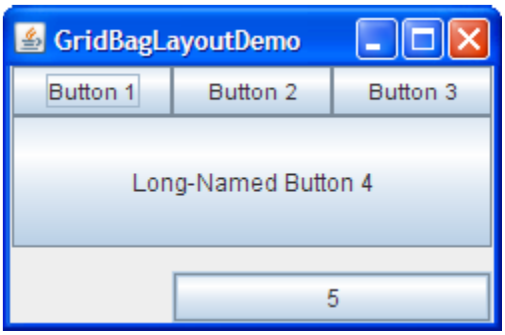
The CardLayout class lets you implement an area that contains different components at different times. A CardLayout is often controlled by a combo box, with the state of the combo box determining which panel (group of components) the CardLayout displays. An 6 alternative to using CardLayout is using a tabbed pane, which provides similar functionality but with a pre-defined GUI.

FlowLayout



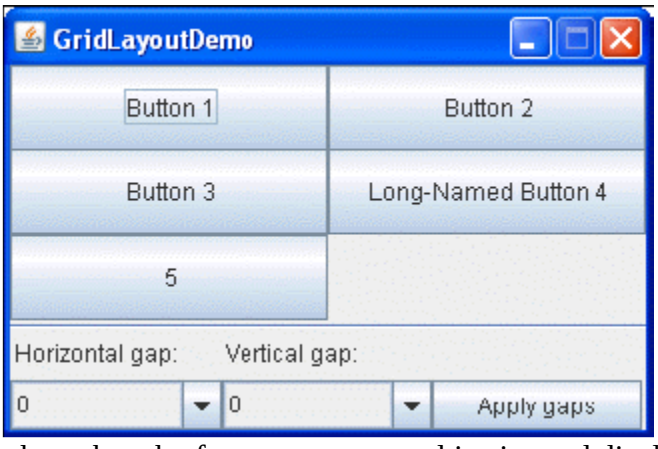
FlowLayout is the default layout manager for every JPanel. It simply lays out components in a single row, starting a new row if its container is not sufficiently wide. Both panels in CardLayoutDemo, shown previously, use FlowLayout.

GridBagLayout



GridBagLayout is a sophisticated, flexible layout manager. It aligns components by placing them within a grid of cells, allowing components to span more than one cell. The rows in the grid can have different heights, and grid columns can have different widths.

GridLayout



GridLayout simply makes a bunch of components equal in size and displays them in the requested number of rows and columns.

d	**Write AWT based Java program that will read a string from user and display the length of the string.**	
2	**Attempt *any two* of the following:**	**10**
a	**Write a Java program using swing components that displays table containing student information (Name, Address, 10th %, 12th %, Emailed, Contact No.).**	

b **What is use of JcolorChooser? Write down the constructors and methods of the same.**

JColorChooser provides a pane of controls designed to allow a user to manipulate and select a color.

Constructors

JColorChooser(): Creates a color chooser pane with an initial color of white.

JColorChooser(Color initialColor): Creates a color chooser pane with the specified initial color.

JColorChooser(ColorSelectionModel model): Creates a color chooser pane with the specified ColorSelectionModel.

Methods

Color getColor(): Gets the current color value from the color chooser.

void setColor(Color color): Sets the current color of the color chooser to the specified color.

void setColor(int c): Sets the current color of the color chooser to the specified color.

void setColor(int r, int g, int b): Sets the current color of the color chooser to the specified RGB color.

static Color showDialog(Component component, String title, Color initialColor): Shows a modal color-chooser dialog and blocks until the dialog is hidden.

c **Distinguish Between AWT & JFC**

JFC (Swing) is a huge set of components which includes labels, frames, tables, trees, andstyled text documents. Almost all Swing components are derived from a single parent called **JComponent** which extends the **AWT** Container class. Swing is a layer on top of AWT rather than a substitution for it.

AWT	**Swing**
AWT stands for Abstract windows toolkit.	Swing is also called as JFC's (Java Foundation classes).
AWT components are called Heavyweight component.	Swings are called light weight component because swing components sits on the top of AWT components and do the work.
AWT components require java.awt package.	Swing components require javax.swing package.
AWT components are platform dependent.	Swing components are made in purely java and they are platform independent.
This feature is not supported in AWT.	We can have different look and feel in Swing. Swing has many advanced features like JTabel, Jtabbed pane which is not available in AWT. Also. Swing components are called "lightweight" because they do not require a native OS object to implement their functionality. JDialog and JFrame are heavyweight, because they do have a peer. So components like JButton, JTextArea,

	etc., are lightweight because they do not have an OS peer.
With AWT, you have 21 "peers" (one for each control and one for the dialog itself). A "peer" is a widget provided by the operating system, such as a button object or an entry field object.	With Swing, you would have only one peer, the operating system's window object. All of the buttons, entry fields, etc. are drawn by the Swing package on the drawing surface provided by the window object. This is the reason that Swing has more code. It has to draw the button or other control and implement its behavior instead of relying on the host operating system to perform those functions.
AWT is a thin layer of code on top of the OS.	Swing is much larger. Swing also has very much richer functionality.
Using AWT, you have to implement a lot of things yourself.	Swing has them built in.

d **Explain Root Pane, Glass Pane, Layered Pane, Content Pane and Desktop Pane.**

Swing offers some top-level containers such as - JApplet, JDialog, and JFrame. There are some problems for mixing lightweight and heavyweight components together in Swing, we can't just add anything but first, we must get something called a "content pane," and then we can add Swing components to that.

The Root Pane :

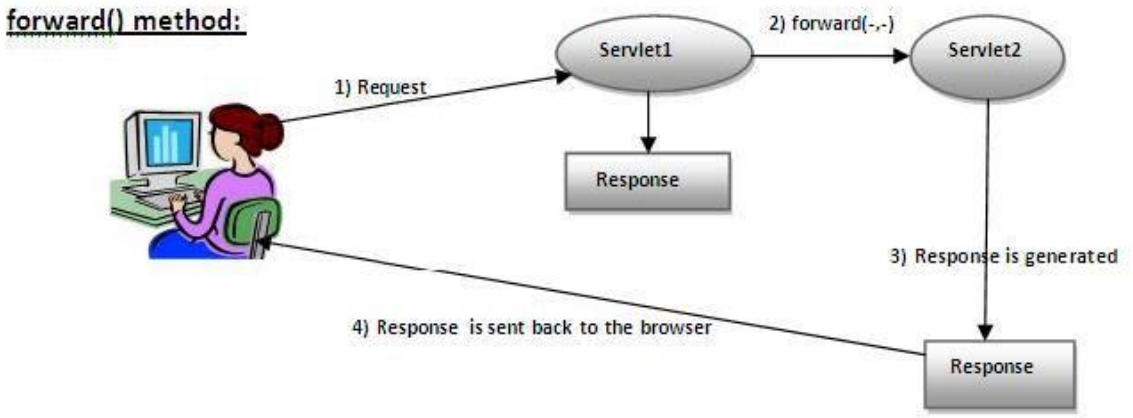
We don't directly create a JRootPane object. As an alternative, we get a JRootPane when we instantiate JInternalFrame or one of the top-level Swing containers, such as JApplet, JDialog, and JFrame. It's a lightweight container used behind the scenes by these toplevel containers. As the preceding figure shows, a root pane has four parts:

1) **The layered pane:** It Serves to position its contents, which consist of the content pane and the optional menu bar. It can also hold other components in a specified order. JLayeredPane adds depth to a JFC/Swing container, allowing components to overlap each other when needed.It allows for the definition of a several layers within itself for the child components. JLayeredPane manages its list of children like Container, but allows for the definition of a several layers within itself.
2) **The content pane:** The container of the root pane's visible components, excluding the menu bar.
3) **The optional menu bar:** It is the home for the root pane's container's menus. If the container has a menu bar, we generally use the container's setJMenuBar method to put the menu bar in the appropriate place.
4) **The glass pane:** It is hidden, by default. If we make the glass pane visible, then it's like a sheet of glass over all the other parts of the root pane. It's completely 9 transparent.The glass pane is useful when we want to be able to catch events or paint over an area that already contains one or more components. We can display an image over multiple components using the glass pane.

JdesktopPane:

The concept of showing multiple windows inside a large frame is implemented using Desktop pane. If we minimize the application frame, all of its windows are hidden at the same time. In Windows environment, this is called as the multiple document interfaceor MDI. Using it we can resize the internal frames inDesktop pane by dragging the resize corners.To achieve this we have follow these steps:

	1. We can use a regular JFrame window for the program. 2. Set the content pane of the JFrame to a JDesktopPane.	
3	**Attempt *any two* of the following:**	**10**
a	**What are servlets? What are the advantages of servlet over CGI?** **What are Servlet?** A Java servlet is a server side program that services HTTP requests and return the results as HTTP responses. A good analogy for a servlet is a non-visual applet and runs on a webserver.it has a lifecycle similar to that of an applet and runs inside a JVM at the web server.The javax.Servlet and javax.Servlet.http packages provide interfaces and classes for writing servlets. All servlets must implement the Servlet interface, which defines lifecycle methods. When implementing a generic service, we can use or extend the GenericServlet class provided with the Java Servlet API. The HttpServlet class provides methods, such as doGet() and doPost(), for handling HTTP-specific services. **What are the advantages of servlet over CGI?** • Servlets are loaded into memory once and run from memory thereafter. • Servlets are swapped as a thread, not as a process. • Servlets are powerful object oriented abstraction of http. • Servlets are portable across multiple web servers and platforms. • Servlets are tightly integrated with web server. • Servlets run within the secure and reliable scope of JVM • Servlets provide direct database access using native and ODBC based Dbdrivers. • Being on the server side provide code protection. • Servlets are robust, scalable, secure CGI replacement.	
b	**What is Request Dispatcher? What are its two methods?** **RequestDispatcher in Servlet** The RequestDispacher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp.This interface can also be used to include the content of antoher resource also. It is one of the way of servlet collaboration. **Methods of RequestDispatcher interface:** **1. public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException**:Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server. **2. public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException**:Includes the content of a resource (servlet, JSP page, or HTML file) in the resp	



c **Explain the ServletRequest interface of servlet API.**

When a servlet accepts a call from a client, it receives two objects.
ServletRequest encapsulates the communication from the client to server.
ServletResponse encapsulates the communication from the server to client.

ServletRequest interface allows the Servlet to access information such as
- Names of parameter passed by client
- Protocol scheme such as HTTP POST and PUT methods being used by client.
- Names of the remote host that made the request.
- Servlet that receives it
- An input stream for reading binary data from the request body.

Subclass os ServletRequest allow the servlet to retrieve more protocol specific data.
Eg. HttpServletRequest :Defines an object to provide client request information to a servlet. The servlet container creates a ServletRequest object and passes it as an argument to the servlet's service method. A ServletRequest object provides data including parameter name and values, attributes, and an input stream.

Methods
java.lang.Object getAttribute(java.lang.String name) Returns the value of the named attribute as an Object, or null if no attribute of the given name exists.
java.util.Enumeration getAttributeNames() Returns an Enumeration containing the names of the attributes available to this request.
java.lang.String getParameter(java.lang.String name) Returns the value of a request parameter as a String, or null if the parameter does not exist.
java.util.Enumeration getParameterNames() Returns an Enumeration of String objects containing the names of the parameters contained in this request.
java.lang.String[] getParameterValues(java.lang.String name) Returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist.
RequestDispatcher getRequestDispatcher(java.lang.String path) Returns a RequestDispatcher object that acts as a wrapper for the resource located at the given path.
java.lang.String getServerName() Returns the host name of the server that received the request.
void removeAttribute(java.lang.String name) Removes an attribute from this request.
void setAttribute(java.lang.String name, java.lang.Object o) Stores an attribute in this request.

d Write a servlet program to display the factorial of a given number.

4	**Attempt *any two* of the following:**	**10**
a	**List and explain the Directive tags of JSP.**	

Directive Tag gives special instruction to Web Container at the time of page translation. Directive tags are of three types: **page**, **include** and **taglib**.

Directive	Description
<%@ page ... %>	defines page dependent properties such as language, session, errorPage etc.
<%@ include ... %>	defines file to be included.
<%@ taglib ... %>	declares tag library used in the page

The **Page directive** defines a number of page dependent properties which communicates with the Web Container at the time of translation. Basic syntax of using the page directive is <%@ page attribute="value" %> where attributes can be one of the following :

- *import* attribute
- *language* attribute
- *extends* attribute
- *session* attribute
- *isThreadSafe* attribute
- *isErrorPage* attribute
- *errorPage* attribute
- *contentType* attribute
- *autoFlush* attribute
- *buffer* attribute

The *include* directive tells the Web Container to copy everything in the included file and paste it into current JSP file. Syntax of **include** directive is:

<%@ include file="filename.jsp" %>

The **taglib** directive is used to define tag library that the current JSP page uses. A JSP page might include several tag library. JavaServer Pages Standard Tag Library (JSTL), is a collection of useful JSP tags, which provides mahy commonly used core functionalities. It has support for many general, structural tasks such as iteration and conditionals, readymade

tags for manipulating XML documents, internationalization tags, and for performing SQL operations. Syntax of taglib directive is:

<%@ taglib prefix="prefixOfTag" uri="uriOfTagLibrary" %>
The prefix is used to distinguish the custom tag from other libary custom tag. Prefix is prepended to the custom tag name. Every custom tag must have a prefix.

b **Explain JDBC architecture**.

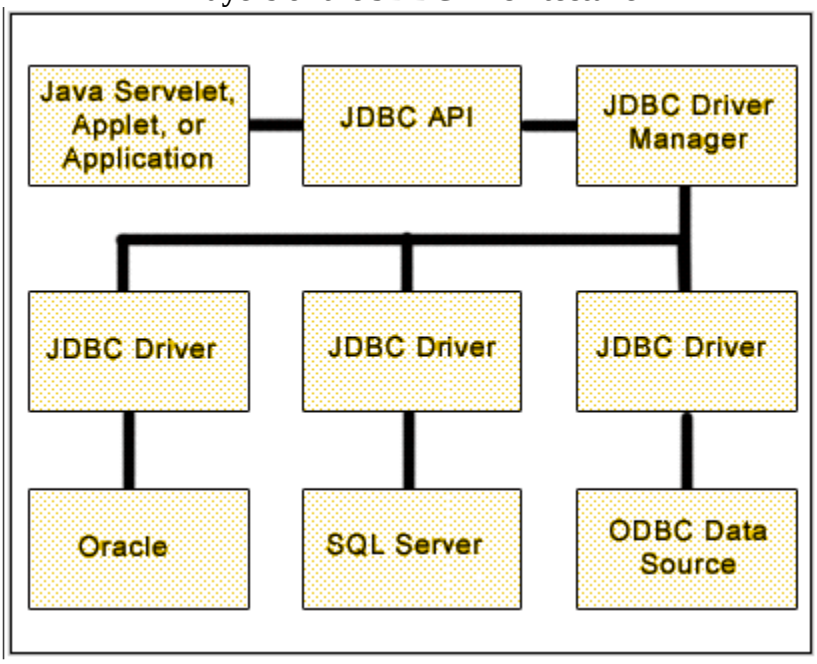
JDBC is a Java API for executing SQL statements and supports basic SQL functionality. The JDBC (Java Database Connectivity) is an API that defines interfaces and classes for writing database applications in Java by making database connections. It is a program designed to access many popular database products on a number of operating system platforms.Using JDBC we can send SQL, PL/SQL statements to almost any relational database. It provides RDBMS access by allowing us to embed SQL inside Java code.

JDBC Architecture
The main function of the JDBC API is to provide a means for the developer to issue SQL statements and process the results in a consistent, databaseindependently. JDBC provides wealthy object-oriented access to databases by defining classes and interfaces that represent objects such as:

- Database connections : **(Short explanation Expected)**
- SQL statements : **(Short explanation Expected)**
- Result Set: **(Short explanation Expected)**
- Database metadata: **(Short explanation Expected)**
- Prepared statements: **(Short explanation Expected)**
- Binary Large Objects (BLOBs): **(Short explanation Expected)**
- Character Large Objects (CLOBs): **(Short explanation Expected)**
- Callable statements : **(Short explanation Expected)**
- Database drivers: **(Short explanation Expected)**
- Driver manager: **(Short explanation Expected)**

The JDBC API uses a Driver Manager and databaseprecise drivers to provide clear connectivity to heterogeneous databases. The JDBC driver manager ensures that the correct driver is used to access each data source. The Driver Manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.

Layers of the JDBC Architecture



c **Write the purpose of the following JDBC classes**

	i. DriverManager ii. ResultSet iii. Statement iv. Connection v. PreparedStatement **DriverManager :** Loads all the drivers in the memory at run time. **ResultSet :** is a database result set generated from currently executed SQL statement. **Statement :** interface represents a static SQL statement that can be used to retrieve Resultset object(s). The objective is to pass to the database the SQL command for execution and to retrieve output results from the database in th eform of Resultset. **Connection :** interface reprsents a connection with a data source. Can be used to retrieve information regarding the tables in th edatabase to which connection is opened. **PreparedStatement :** is an SQL statement that is pre-compled and stored. This object can then be executed multiple times much more efficiently than preparing and issuing the same statement each time it is needed.	
d	**What are the advantages and disadvantages of Java Server pages?** JavaServer Pages (JSP) is a serverside development technology that is used to create dynamic web pages and applications. It is introduced after Java Servlets. With Servlets Java became a full-fledged application server programming language. This is achieved by embedding Java code into HTML, XML, DHTML, or other document types. When a client such as a web browser makes a request to the Java application container, which is typically a web server, the static page is converted behind the scenes, and displayed as dynamic content to the viewer. **1 Advantage of JSP:** • The JSP serves all facilities of Java i.e. write once run anywhere. • JSP is ideal for Web based Technology. • The JSP pages are translated and compiled into JAVA Servlet but are easier to develop than JAVA Servlet. • The JSP uses simplified scripting language based syntax for embedding HTML into JSP. • JSP containers provide easy coding for accessing standard objects and actions. • JSP acquire all the benefits provided by JAVA Servlets and web container environment. • The JSP use HTTP as default request /response communication model. **2: Disadvantage:** • The JSP implementation is normally causes for poor diagnostics. • Difficult looping in jsp. • The space used to store JSP page is comparatively more. • The first time loading of JSP is little bit time consuming because JSP pages must be compiled on the server when first accessed.	
5	**Attempt *any two* of the following:**	**10**
a	**Explain Model-view-controller architecture.** **MVC Architecture** The MVC stands for Model, View, and Controller architecture. The MVC architecture separates the business logic and application data from the presentation data to the user.MVC is design pattern which allows a developer to write their applications in a specific format, following the same directory structure, using the same configuration, allowing making unique chain between the components & documents of the application. Core parts of MVC architecture. 1) **Model:** The model object only represents the data of an application. The model object knows about all the data that need to be displayed. The model is aware about all the operations that can be applied to transform that object. The model represents enterprise data and the business rules that govern access to and updates of this data. Model is not concern about the presentation data and how that data will be displayed to the browser.	

2) **View:** The view represents the presentation of the application. The view object refers to the model. It uses the query methods of the model to obtain the contents and renders it. The view is not dependent on the application logic. It remains same if there is any modification in the business logic. It is the responsibility of the view's to maintain the consistency in its presentation when the model (data or logic) changes.
3) **Controller:** Whenever the user sends a request for something then it always go through the controller. The controller is responsible for intercepting the requests from view and passes it to the model for the appropriate action. After the action has been taken on the data, the controller is responsible for directing the appropriate view to the user. In GUI applications the views and the controllers often work very closely together.

b **List various phases of JSF lifecycle. Explain in short.**

JSF lifecycle

To understand how the framework treats the underlying request & Servlet API also how Faces processes each request, we'll go through the JSF request processing lifecycle. A Java Server Faces page is represented by a tree of UI components, called a view. During the lifecycle, the Java Server Faces implementation must build the view while considering state saved from a previous submission of the page. When the client submits a page, the Java Server Faces implementation performs several tasks, such as validating the data input of components in the view and converting input data to types specified on the server side. The Java Server Faces implementation performs all these tasks as a series of steps in the Java Server Faces request response life cycle.

JSF Life cycle handles two kinds of requests:
Initial request: A user requests the page for the first time.
Postback: A user submits the form contained on a page that was previously loaded into the browser as a result of executing an initial request.

The phases of the JSF application lifecycle are as follows:

Phase 1: Restore view

In this phase, JSF classes build the tree of UI components for the incoming request. When a request for a JavaServer Faces page is made, such as when a link or a button is clicked, the JavaServer Faces implementation begins the restore view phase. The JSF framework controller uses the view ID means a name of JSP to look up the components for the current view. If the view isn't available, the JSF controller creates a new one. If the view already exists, the JSF controller uses it. The view contains all the GUI components and there is a great deal of state management by JSF to track the status of the view – typically using HTML hidden fields.

Phase 2: ApplyRequest values

In this phase, the request parameters are examined and their values are used to set the values of the corresponding UI components. This process is called decoding. If the conversion of the value fails, an error message associated with the component is generated. This message will be displayed during the render response phase, along with any validation errors resulting from the process validations phase.

Phase 3: Process validations

In this phase triggers calls to all registered Validators.The components validate the new values coming from the request against the application's validation rules. Any input can be scanned by any number of Validators.These Validators can be predefined or defined by the developer. Any validation errors will abort the requesthandling process and skip to rendering the response with validation and conversion error messages.

Phase 4: Update Model Values

The Update Model phase brings a transfer of state from the UI component tree to any and all backing beans, according to the value expressions defined for the components themselves. In this phase converters are invoked to parse string representations of various values to their proper primitive or object types. If the data cannot be converted to the types specified by the bean properties, the life cycle calls directly to the render response phase so that the page is re-

rendered with errors displayed. In Apply Request Values phase, it moves values from client side HTML form controls to server side UI components; while Update model values phase the information moves from the UI components to the backing beans.

Phase 5: Invoke Application

This phase handles any application level events. Normally this takes the form of a call to process the action event generated by the submit button that the user clicked. In this phase…

- Application level events are handled
- Application methods are invoked
- Navigation outcome are calculated

Phase 6: Render Response

The Render Response Phase brings several contrary behaviors together in one process like values are transferred back to the UI components from the bean including any modifications that may have been made by the bean itself or by the controller; The UI components save their state, not just their values, but other attributes having to do with the presentation itself. This can happen at serverside, but by default state is written into the HTML as hidden input fields and thus returns to the JSF implementation with the next request. If the request is a Postback and errors were encountered during the apply request values phase, process validations phase, or update model values phase, the original page is rendered during this phase. If the pages contain message or messages tags, any queued error messages are displayed on the page.

c **What is Facelet? Write the features of Facelet?**

Facelets is commonly used term to refer to JavaServer Faces View Defination Framework which is page declaration language developed for use with Java server Faces technology. The concept of VDL introduced in JavaServer Faces 2.0 allows declaration of UI components in different presentation technologies. Both JSP and facelets are considered different inpmementation of VDL.

Facelet is built specifically for JavServer Faces. It is now theprefered technology for building JavaServer Faces based applications and offers several advantages over using JSP technology.

Facelets is a powerful but lightweight page declaration language that is used to build JavaServer Faces views using HTML style templates and to build component trees. Facelets features include the following:

- Facelets as presentation technology.
- Templatting and Composite Components through Faceltes.
- New HTML tags for easier page creation.
- Bookmarkability to generate hyperlinks based on component properties at render time .
- New components and event types for additional functionality.
- Resource registration and relocation using annotations.
- Implicit Navigation Rules if none are present in the application configuration resource files.
- Support for Bean Validation.
- Project Stage to describe the status of the application in the project lifecycle.
- Support for Ajax Integration.

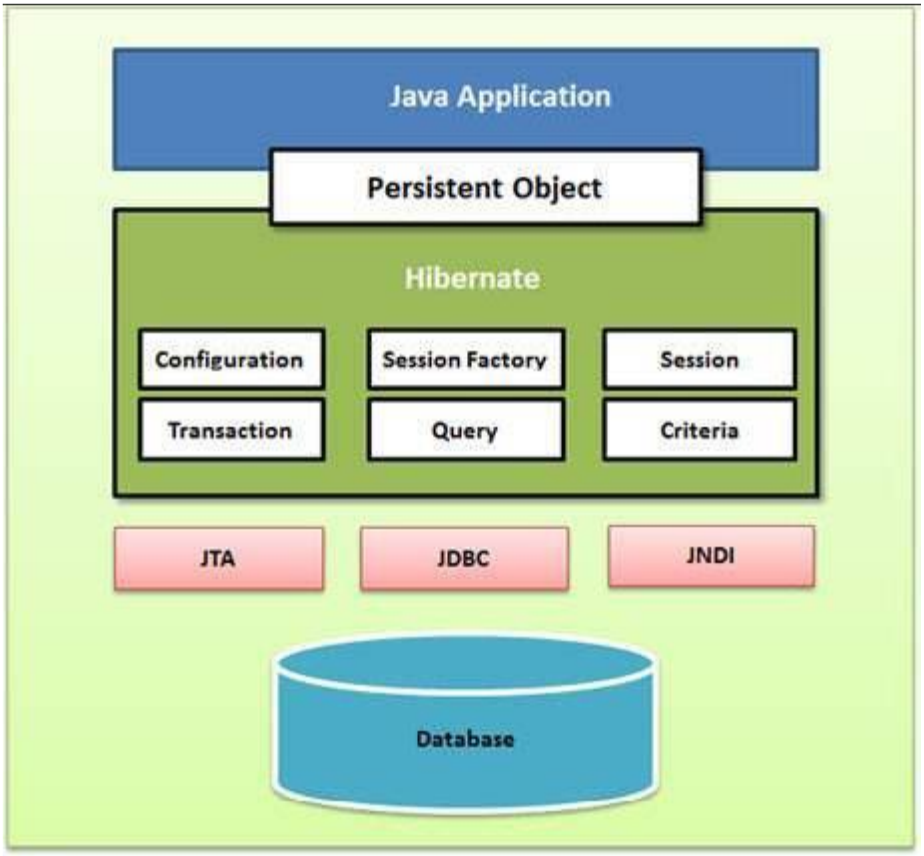
d **Explain the benefits of EJB**.

- Complete focus only on business logic
- Reusable components
- Portable
- Fast Building Application
- One business logic having many presentation logics

- Distributed Deployment
- Application Interoperability

6 Attempt *any two* of the following: **10**

a **Explain Architecture of Hibernate framework in detail.**



Hibernate architecture has three main components as follows:

1 Connection Management:

Hibernate Connection management service provide well-organized management of the database connections. Database connection is the most expensive part of interacting with the database as it requires a lot of resources of open and close the database connection.

2: Transaction management:

Transaction management service of hibernate provides the ability to the user to execute more than one database statements at a time.

3 Object relational mapping:

Object relational mapping is technique of mapping the data representation from an object model to a relational data model. This part of hibernate is used to select, insert, update and delete the records form the underlying table. When we pass an object to a **Session.save()** method, Hibernate reads the state of the variables of that object and executes the necessary query.

4 configuration object

The configuration object represents a configuration or properties file for Hibernate. It is usually created once during application initialization.

5.Session factory

The seession factory is created with the help of a configuration object during the application start up.it serves as as factory for spawning session objects when required**.**

6. session

Session objects are light weight and inexpensive to create. They provide the main interface to perform actual database operation.

7.Transaction

Transaction represents a unit of work with the database.

8.Query
Persistent objects are retrieved using a query object.
9.Criteria
Persistent object can also be retrieved using a criteria object.

b **What is value stack in struts? State and exaplin the execution flow of value stack.**

Value Stack: Value Stack is nothing but stack of objects. The Value Stack is a storage area that holds all of the data associated with the processing of a Request.

Execution Flow of Value Stack:
1. The framework receives a request and decides on the action the URL maps to
2. The framework moves the data to the Value Stack whilst preparing for request processing
3. The framework manipulates the data as required during the action execution
4. The framework reads the data from there and renders the results i.e. Response page struts to maintain a stack of the following objects in the **Value Stack:**
1) Temporary Objects: Theseare generated and placed in the **Value Stack** during execution. These objects provide temporary storage and are normallygenerated while processing a request.For example, the current iteration value for a collection being looped over in JSP tag.
2) The Model Object: If the application user Model objects, the current model object is placed in the Value Stack before the Action executes.
3) The Action Object: It is the Action that is currently being executed.
4) Named Objects: Any object assigned to an identifier called as a **Named Object.**These objects can either be created by the developer of pre-defined such as #application

Accessing Value Stack : The **Value Stack** can be accessed by simply using the **tags** provided for JSP.
- When the Value Stack Is queried for an **attribute value,** each stack element, in the provided order, is asked whether it holds the queried property.
- If it holds the queried property, then the value is returned.
- If it does not hold the queried property, then the next element down is queried.

This continues until the last element in the stack is scanned.

c **Write a short on Interceptors in struts.**

Interceptors
Interceptors allow developing code that can be run before and/or after the execution of an action. A request is usually processed as follows:
- A user requests a resource that maps to an action
- The Struts 2 framework invokes the appropriate action to serve the request

If interceptors are written, available and configured, then:
- **Before the action is executed** the invocation could be intercepted by another object
- **After the action executes**, the invocation could be intercepted again by another object

Such objects who intercept the invocation are called **Interceptors.** Conceptually, interceptors are very similar to **Servlet Filters** or the JDKs **Proxy class**. **Interceptor Configuration:**
Interceptors configured in the struts.xml file appear as:

```
<interceptors>
<interceptor name="test1" class="..."/>
<interceptor name=”test2" class="..."/>
</interceptors>
<action name="WelcomeSturts">
<interceptor-ref name=" test 1"/>
<interceptor-ref name=" test 2"/>
```

<result name="SUCCESS">/ Welcome.jsp</result>
<result name="ERROR">/ Error.jsp</result>
</action>

In above code two interceptors named test1 and test2 are defined. Both of these are them mapped to the action named WelcomeSturts.

Interceptor Stack: We can bind Interceptor together using an Interceptor **Stack** which can be referenced together. We can use the same set of interceptors multiple times. So, instead of configuring a number of interceptors every time, an interceptor stack can be configured with all the required interceptors held within. To use Intercept Stack we need to modify the struts.xml file as follows:

<interceptors>
<interceptor name="test1" class="..."/>
<interceptor name= test2" class="..."/>
<interceptor-stack name="MyStack">
<interceptor-ref name=" test1"/>
<interceptor-ref name=" test2"/>
</interceptor-stack>
</interceptors>
<action name="WelcomeSturts " class="test.WelcomeSturts ">
<result name="SUCCESS">/ Welcome.jsp</result>
<result name="ERROR">/ Error.jsp</result>
</action>

In above code two interceptors named **test1**and **test2** are defined and a stack named **MyStack**group them both.The stack holding both these interceptors is then mapped to the Action named **WelcomeStruts.**
Execution Flow of Interceptors: Interceptors are executed as follows:
1. The framework receives a request and decides on the action the URL maps to
2. The framework consults the application's configuration file, to discover which interceptors should fire and in what sequence
3. The framework starts the invocation process by executing the first Interceptor in the **Stack**
4. After all the interceptors are invoked, the framework causes the action itself to be **executed.**

d **Explain structure of hibernate.cfg.xml file**.

Hibernate uses the "**hibernate.cfg.xml**" file to create the connection & setup the required environment.
This file contains information such as…..
1) Database Connection 2) Resource mapping

hibernate.cfg.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url"> jdbc:mysql://localhost:3306/Feedback
</property>
<property name="hibernate.connection.username">root</property>

	<property name="hibernate.connection.password">root</property> </session-factory> </hibernate-configuration> **Explaination:** **1. hibernate.connection.driver_class:** It is the JDBC connection class for the specific database. **2. hibernate.connection.url:** It is the full JDBC URL to the database. 3. **hibernate.connection.username:** It is the username used to connection the database. **4. hibernate.connection.password:** It is the password used to authenticate the username. **5. hibernate.dialect:** It is the name of SQL dialect for the database.	
7	**Attempt *any three* of the following:**	**15**
a	**What is CheckBoxGroup? Explain with Example**. A *check box* is a control that is used when there are multiple options and multiple selections to turn an option on or off. It consists of a small box that can either contain a check mark or not. **CheckboxGroup** It is possible to create a set of mutually exclusive check boxes in which one and only one check box in the group can be checked at any one time. These check boxes are often called ***radio buttons***. Check box groups are objects of type **CheckboxGroup**. Only the default constructor is defined, which creates an empty group. Methods: ***Checkbox getSelectedCheckbox( )*** : Reads the selected radio button option from group ***void setSelectedCheckbox(Checkbox which)***: Selects radio button option of a group. **Example:** ***import java.applet.*;*** ***import java.awt.*;*** ***/**** ***<applet code="TCBCBG" width=200 height=150>*** ***</applet>*** ****/*** ***public class TCBCBG extends Applet {*** ***Checkbox morning,noon,evening;*** ***public void init() {*** ***this.add(new Label("How will you pay for your pizza?"));*** ***CheckboxGroup cbg = new CheckboxGroup();*** ***this.add(new Checkbox("Visa", cbg, false));*** ***this.add(new Checkbox("Cash", cbg, true)); // the default*** ***this.add(new Label("Select Time Slots for Delivery"));*** ***morning = new Checkbox("Morning", null, true);*** ***noon = new Checkbox("Afternoon");*** ***evening = new Checkbox("Evening");*** ***this.add(morning);*** ***this.add(noon);*** ***this.add(evening);*** ***}*** ***}***	
b	**Write a java program to demonstrate the use of Tabbed Panes.**	
c	**What is the purpose of WEB-INF file? Explain.** A Web application exists as a structured hierarchy of directories. The root of this hierarchy serves as the document root for files that are part of the application.	

	A special directory exists within the application hierarchy named "WEB-INF". This directory contains all things related to the application that aren't in the document root of the application. The WEB-INF node is not part of the public document tree of the application. No file contained in the WEB-INF directory may be served directly to a client by the container. However, the contents of the WEBINF directory are visible to servlet code using the getResource and getResourceAsStream method calls on the ServletContext, and may be exposed using the RequestDispatcher calls. Hence, if the Application Developer needs access, from servlet code, to application specific configuration information that he does not wish to be exposed directly to the Web client, he may place it under this directory. Since requests are matched to resource mappings in a case-sensitive manner, client requests for '/WEB-INF/foo', '/WEb-iNf/foo', for example, should not result in contents of the Web application located under /WEB-INF being returned, nor any form of directory listing thereof. The contents of the WEB-INF directory are: • The /WEB-INF/web.xml deployment descriptor. • The /WEB-INF/classes/ directory for servlet and utility classes. The classes in this directory must be available to the application class loader. • The /WEB-INF/lib/*.jar area for Java ARchive files. These files contain servlets, beans, and other utility classes useful to the Web application. The Web application class loader must be able to load classes from any of these archive files. The Web application class loader must load classes from the WEB-INF/ classes directory first, and then from library JARs in the WEB-INF/lib directory. Also, any requests from the client to access the resources in WEB-INF/ directory must be returned with a SC_NOT_FOUND(404) response.	
d	**Write a JDBC program that inserts values in database. [table Name : Book, Fileds : Bookid, Title, Author, Publisher]**	
e	**What are the different types of enterprise beans? Explain**. Enterprise java beans are reusable modules of code that combine related tasks into well-defined interface. These enterprise beans EJB components contain the method that executes business logic and access data sources. **1: Session beans:** **A) Stateful Session Beans:** Stateful Session Beans are business objects having state, means they can keep track of which calling client they are dealing with throughout a session and thus access to the bean instance is strictly limited to "only one client at a time". In the case of concurrent access to a single bean is attempted anyway the container serializes those requests, but via the **@AccessTimeout** annotation the container can throw an exception instead. Stateful session beans' state may be persisted automatically by the container to free up memory after the client hasn't accessed the bean for some time. **Example:** The hotel check out may be handled by a stateful session bean that would use its state to keep track of where the customer is in the checkout process, possibly holding locks on the items the customer is charged for services. **B) Stateless Session Beans:** Stateless Session Beans are business objects that do not have state associated with them. Access to a single bean instance is limited to only one client at a time and thus concurrent access to the bean is banned. In case concurrent access to a single bean is attempted anyway the container simply routes each request to a different instance. Instances of Stateless Session beans are typically pooled. If a second client accesses a specific bean right after a method call on it made by a first client has finished, it might get the same instance. **Example:** Sending an Email to customer support may be handled by a stateless bean since this is a oneoff operation and not part of a multistep process. **C) Singleton Session Beans:** Singleton Session Beans are business objects having a global shared state in a JVM. Concurrent access to the one and only bean instance can be controlled by the container or by the bean itself. Container Managed concurrency can be tuned using the	

@Lock annotation, that designates whether a read lock or a write lock will be used for a method call. Also the Singleton Session Beans can explicitly request to be instantiated when the EJB container starts up, using the **@Startup** annotation.
Examples: Loading a daily price list that will be the same for every user might be done with a singleton session bean, since this will prevent the application having to do the same query to a database over and over again.

2.Message Driven Beans:
Message Driven Beans are business objects whose execution is happened by the messages instead by method calls. Like session beans, a Message Driven Beans does not have any type of client view, i.e. clients cannot lookup a Message Driven Beans instance. It just listens for any incoming message on, for example, a JMS queue or topic and processes them automatically. Message Driven Beans can support many messaging protocols. The difference between session- and message driven beans is only in method calling and messaging.
Example: Submitting a job to a work bunch might be done by sending a JMS message to a 'message queue' and could also be handled by a Message Driven Bean.

f. **What is OGNL? Explain the execution flow of OGNL.**

OGNL [Object-Graph Navigation Language]

The Object-Graph Navigation Language is a fully featured expression language for Retrieving and Setting properties of the Java object s. It helps data transfer and type conversion.
In the Value Stack,Searching or evaluating, a particular expression can be done using OGNL. OGNL provides a mechanism to navigate object graphs using a dot notation and evaluate expressions, including calling methods on the objects being retrieved.

OGNL supportsType conversion, calling methods, Collectionmanipulation, Generation, Projection across collections, Expression evaluation, Lambda expressions etc
OGNL Examples

The following are a few examples where OGNL is used:
emp.name:
It returns the value that is actually returned when getEmp().getName() is Invoked
emp.toSpring():
It returns the value that is actually returned when getEmp().toString() is invoked.
@test.auth.name@fName():
It returns the value that is actual returned when the static method named fName()
is invoked on the class name.
firstName in {"Sharanam","vaishali"}:
invokes getfName() and determines the value returned is either Tushar or
Sonali. If it is, then returns True.

Execution flow of OGNL :
1. A user enters the data in and data entry form and submits the form
2. The Request enters the Struts 2 framework and is made available to the Java Language as an HttpServletRequest object.
3. The request paramets are stored as name/value paris where the name are the Names of the data entry from's text fields and the Value are the Value entered by the user when the form is submitted
4. Now the OGNL comes into picture , to handle the transfer and type conversion of the data from these request parameters
5. Using the OGNL expression , the value stack is scanned to locate the destination property where the data has to be moved

	6. On locating the correct property in the Value Stack, the data is moved to the property by invoking the property's SETTER method with the appropriate value. The value stack acts as a place holder for viewing the data model throughout the execution 7. Whilst moving such data, OGNL consults its set of available type converters to determine if any of them can handle this particular conversion, if a conversion is required. The value is converted and set on the object's property. This makes the data available them action begins its job, immediately after the available Interceptors ate fired. 8. After Action completes to its job successfully, a Result fires that renders the result they to the user. 9. Results have access to the Value Stack, via the OGNL expression language with the help of tags. These tags retrieve data from the Value Stack by referencing specific values using OGNL expressions. 10. Whilst rendering the view , once again, the value that is accessed from the value Slack is converted from the java type to a String that is written on the HTML page	