Explain any two components of swing with examples.

The basic AWT library deals with user interface elements by delegating their creation and behavior to the native GUI toolkit on each target platform (Windows, Solaris, Macintosh, and so on).

This peer-based approach worked well for simple applications. User interface elements such as menus, scrollbars, and text fields can have subtle differences in behavior on different platforms. Moreover, some graphical environments do not have as rich a collection of user interface components as does Windows or the Macintosh Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
2)	AWT components are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC**.

Java Swing Example 1 :

Using java button:

```
1.  import javax.swing.*;
2.  public class FirstSwingExample {
3.  public static void main(String[] args) {
4.  JFrame f=new JFrame();//creating instance of JFrame
5.  JButton b=new JButton("click");//creating instance of JButton
6.  b.setBounds(130,100,100, 40);//x axis, y axis, width, height
7.  f.add(b);//adding button in JFrame
8.  f.setSize(400,500);//400 width and 500 height
9.  f.setLayout(null);//using no layout managers
10. f.setVisible(true);//making the frame visible
11. }
12. }
```

Example 2:

Java JLabel Example

```
1.  import javax.swing.*;
2.  class LabelExample
3.  {
4.  public static void main(String args[])
5.  {
6.  JFrame f= new JFrame("Label Example");
7.  JLabel l1,l2;
8.  l1=new JLabel("First Label.");
9.  l1.setBounds(50,50, 100,30);
10. l2=new JLabel("Second Label.");
11. l2.setBounds(50,100, 100,30);
12. f.add(l1); f.add(l2);
13. f.setSize(300,300);
14. f.setLayout(null);
15. f.setVisible(true);
16. }
17. }
```

==========

What is the MVC design pattern in swing? Explain Event handling in swing with suitable examples.

Swing actually makes use of a simplified variant of the MVC design called the model-delegate . This design combines the view and the controller object into a single element that draws the component to the screen and handles GUI events known as the UI delegate . Bundling graphics capabilities and event handling is somewhat easy in Java, since much of the event handling is taken care of in AWT. As you might expect, the communication between the model and the UI delegate then becomes a two-way street, as shown in Figure below:

With Swing, the view and the controller are combined into a UI-delegate object

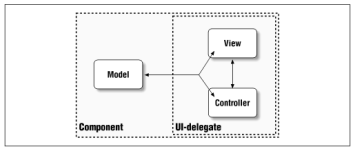


Figure 1-7. With Swing, the view and the controller are combined into a UI-delegate object

Each Swing component contains a model and a UI delegate. The model is responsible for maintaining information about the component's state. The UI delegate is responsible for maintaining information about how to draw the component on the screen. In addition, the UI delegate (in conjunction with AWT) reacts to various events that propagate through the component.

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has a code which is known as an event handler, that is executed when an event occurs.

Create the following Java program using any editor of your choice in say D:/ > SWING > com > example> gui >

SwingControlDemo.java

```
1)  package com.tutorialspoint.gui;
2)  import java.awt.*;
3)  import java.awt.event.*;
4)  import javax.swing.*;
5)  public class SwingControlDemo {
6)  private JFrame mainFrame;
7)  private JLabel headerLabel;
8)  private JLabel statusLabel;
9)  private JPanel controlPanel;
10) public SwingControlDemo(){
11) prepareGUI();
12) }
13) public static void main(String[] args){
14) SwingControlDemo swingControlDemo = new SwingControlDemo();
15) swingControlDemo.showEventDemo();
16) }
17) private void prepareGUI(){
18) mainFrame = new JFrame("Java SWING Examples");
19) mainFrame.setSize(400,400);
20) mainFrame.setLayout(new GridLayout(3, 1));
21) headerLabel = new JLabel("",JLabel.CENTER );
22) statusLabel = new JLabel("",JLabel.CENTER);
23) statusLabel.setSize(350,100);
24) mainFrame.addWindowListener(new WindowAdapter() {
25) public void windowClosing(WindowEvent windowEvent){
26) System.exit(0);
27) }
28) });
29) controlPanel = new JPanel();
30) controlPanel.setLayout(new FlowLayout());
31) mainFrame.add(headerLabel);
32) mainFrame.add(controlPanel);
33) mainFrame.add(statusLabel);
34) mainFrame.setVisible(true);
35) }
36) private void showEventDemo(){
37) headerLabel.setText("Control in action: Button");
38) JButton okButton = new JButton("OK");
39) JButton submitButton = new JButton("Submit");
40) JButton cancelButton = new JButton("Cancel");
41) okButton.setActionCommand("OK");
42) submitButton.setActionCommand("Submit");
43) cancelButton.setActionCommand("Cancel");
44) okButton.addActionListener(new ButtonClickListener());
45) submitButton.addActionListener(new ButtonClickListener());
46) cancelButton.addActionListener(new ButtonClickListener());
47) controlPanel.add(okButton);
48) controlPanel.add(submitButton);
49) controlPanel.add(cancelButton);
50) mainFrame.setVisible(true);
51) }
52) private class ButtonClickListener implements ActionListener{
53) public void actionPerformed(ActionEvent e) {
54) String command = e.getActionCommand();
55) if( command.equals( "OK" )) {
56) statusLabel.setText("Ok Button clicked.");
57) } else if( command.equals( "Submit" ) ) {
58) statusLabel.setText("Submit Button clicked.");
59) } else {
60) statusLabel.setText("Cancel Button clicked.");}}}}
```

Cookies in Servlet

A cookie is a small piece of information that is persisted between the multiple client requests.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

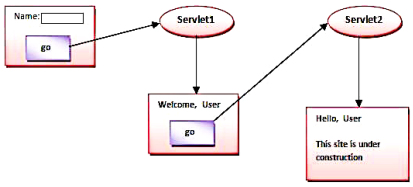
How Cookie works?

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.

Simple example of Servlet Cookies

In this example, we are storing the name of the user in the cookie object and accessing it in another servlet.

As we know well that session corresponds to the particular user. So if you access it from too many browsers with different values, you will get the different value.



index.html

```
1)  <form action="servlet1" method="post">
2)  Name:<input type="text" name="userName"/><br/>
3)  <input type="submit" value="go"/>
4)  </form>
```

FirstServlet.java

```
1)  import java.io.*;
2)  import javax.servlet.*;
3)  import javax.servlet.http.*;
4)  public class FirstServlet extends HttpServlet {
5)  public void doPost(HttpServletRequest request, HttpServletResponse response){
6)  try{
7)  response.setContentType("text/html");
8)  PrintWriter out = response.getWriter();
9)  String n=request.getParameter("userName");
10) out.print("Welcome "+n);
11) Cookie ck=new Cookie("uname",n);//creating cookie object
12) response.addCookie(ck);//adding cookie in the response
13) //creating submit button
14) out.print("<form action='servlet2'>");
15) out.print("<input type='submit' value='go'>");
16) out.print("</form>");
17) out.close();
18) }catch(Exception e){System.out.println(e);}
19) }
20) }
```

SecondServlet.java

```
1)  import java.io.*;
2)  import javax.servlet.*;
3)  import javax.servlet.http.*;
4)  public class SecondServlet extends HttpServlet {
5)  public void doPost(HttpServletRequest request, HttpServletResponse response){
6)  try{
7)  response.setContentType("text/html");
8)  PrintWriter out = response.getWriter();
9)  Cookie ck[]=request.getCookies();
10) out.print("Hello "+ck[0].getValue());
11) out.close();
12) }catch(Exception e){System.out.println(e);}
13) }
14) }
```

web.xml

```
1)  <web-app>
2)  <servlet>
3)  <servlet-name>s1</servlet-name>
4)  <servlet-class>FirstServlet</servlet-class>
5)  </servlet>
6)  <servlet-mapping>
7)  <servlet-name>s1</servlet-name>
8)  <url-pattern>/servlet1</url-pattern>
9)  </servlet-mapping>
10) <servlet>
11) <servlet-name>s2</servlet-name>
12) <servlet-class>SecondServlet</servlet-class>
13) </servlet>
14) <servlet-mapping>
15) <servlet-name>s2</servlet-name>
16) <url-pattern>/servlet2</url-pattern>
17) </servlet-mapping>
18) </web-app>
```

Explain different types of tags in JSP with examples.

JSP Scripting elements

The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

1) scriptlet tag
2) expression tag
3) declaration tag

1. JSP scriptlet tag

A scriptlet tag is used to execute java source code in JSP. Syntax is as follows:

```
<% java source code %>
```

Example of JSP scriptlet tag that prints the user name

In this example, we have created two files index.html and welcome.jsp. The index.html file gets the username from the user and the welcome.jsp file prints the username with the welcome message.

File: index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

File: welcome.jsp

```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
</form>
</body>
</html>
```

2. JSP expression tag

The code placed within JSP expression tag is written to the output stream of the response. So you need not write out.print() to write data. It is mainly used to print the values of variable or method.

Example of JSP expression tag that prints the user name

In this example, we are printing the username using the expression tag. The index.html file gets the username and sends the request to the welcome.jsp file, which displays the username.

File: index.jsp

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname"><br/>
<input type="submit" value="go">
</form>
</body>
</html>
```

File: welcome.jsp

```
<html>
<body>
<%= "Welcome "+request.getParameter("uname") %>
</body>
</html>
```

3. JSP declaration tag

The JSP declaration tag is used to declare fields and methods.

The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet.

So it doesn't get memory at each request.

Example of JSP declaration tag that declares method

In this example of JSP declaration tag, we are defining the method which returns the cube of given number and calling this method from the jsp expression tag. But we can also use jsp scriptlet tag to call the declared method.

index.jsp

```
<html>
<body>
<%!
int cube(int n){
return n*n*n;
}
%>
<%= "Cube of 3 is:"+cube(3) %>
</body>
</html>
```

==========

Why are adapter classes important? Compare it with the listener interface with suitable examples.

Java adapter classes provide the default implementation of listener interfaces. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it saves code.

Pros of using Adapter classes:

- It assists the unrelated classes to work combinedly.
- It provides ways to use classes in different ways.
- It increases the transparency of classes.
- It provides a way to include related patterns in the class.
- It provides a pluggable kit for developing an application.
- It increases the reusability of the class.

Listeners are used when the programmer intends to utilize most of the methods listed under the interface. If a listener interface is implemented directly by a class, all the methods within that interface need to be implemented, making the code unreasonably large. This complexity can be resolved by calling upon an adapter class. An adapter class proves essential in instances where an event calls for only specific methods.

The programmer has only to create a subclass of it and override the interest methods to use an adapter class. Adapter classes are, therefore, beneficial for listener interfaces in JAVA having more than one method. To better understand this, let us consider the example of the MouseListener interface. This interface is notified whenever there is a change in the state of the mouse. It has five methods, mouse clicked, mouseExited, mouseEntered, mousePressed, and mouseReleased.

Java WindowAdapter Example

In the following example, we are implementing the WindowAdapter class of AWT and one its methods windowClosing() to close the frame window.

AdapterExample.java

```
1)  import java.awt.*;
2)  import java.awt.event.*;

3)  public class AdapterExample {
4)  // object of Frame
5)  Frame f;
6)  // class constructor
7)  AdapterExample() {
8)  // creating a frame with the title
9)  f = new Frame ("Window Adapter");
10) // adding the WindowListener to the frame
11) // overriding the windowClosing() method
12) f.addWindowListener (new WindowAdapter() {
13) public void windowClosing (WindowEvent e) {
14) f.dispose();
15) }
16) });
17) // setting the size, layout and
18) f.setSize (400, 400);
19) f.setLayout (null);
20) f.setVisible (true);
21) }

22) // main method
23) public static void main(String[] args) {
24) new AdapterExample();
25) }
26) }
```

==========

What is Java bean? Explain Properties, Events and Methods design patterns.

A JavaBean is a Java class that should follow the following conventions:

- It should have a no-arg constructor.
- It should be Serializable.
- It should provide methods to set and get the values of the properties, known as getter and setter methods.

A JavaBean property may be read, write, read-only, or write-only. JavaBean features are accessed through two methods in the JavaBean's implementation class:

1. getPropertyName ()

For example, if the property name is firstName, the method name would be getFirstName() to read that property. This method is called the accessor.

2. setPropertyName ()

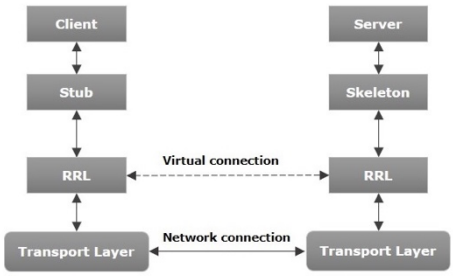
For example, if the property name is firstName, the method name would be setFirstName() to write that property. This method is called the mutator.

Property design patterns are used to identify the properties of a Bean. Not surprisingly, property design patterns are closely related to accessor methods. In fact, accessor methods are the means by which the JavaBean's automatic introspection facility determines the properties of a Bean. Basically, any time the JavaBeans introspector encounters a public getter or setter method, it assumes the member variable being get or set is a property, and then exposes the property to the outside world.

==========

Explain RMI Architecture in detail.

Remote Method Invocation (RMI) is an API that allows an object to invoke a method on an object that exists in another address space, which could be on the same machine or on a remote machine. Through RMI, an object running in a JVM present on a computer (Client-side) can invoke methods on an object present in another JVM (Server-side). RMI creates a public remote server object that enables client and server-side communications through simple method calls on the server object.



Transport Layer : This layer connects the client and the server. It manages the existing connection and also sets up new connections.

Stub : A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.

Skeleton: This is the object which resides on the server side. stub communicates with this skeleton to pass request to the remote object.

RRL(Remote Reference Layer): It is the layer which manages the references made by the client to the remote object.

Working of an RMI Application

The following points summarize how an RMI application works –

- When the client makes a call to the remote object, it is received by the stub which eventually passes this request to the RRL.
- When the client-side RRL receives the request, it invokes a method called invoke() of the object remoteRef. It passes the request to the RRL on the server side.
- The RRL on the server side passes the request to the Skeleton (proxy on the server) which finally invokes the required object on the server.
- The result is passed all the way back to the client.

Goals of RMI

Following are the goals of RMI –

- To minimize the complexity of the application.
- To preserve type safety.
- Distributed garbage collection.
- Minimize the difference between working with local and remote objects.

==========

Explain about different types of ResultSet used in JDBC.

There are two types of result sets namely, forward only and, bidirectional.

1. Forward only ResultSet:

The ResultSet object whose cursor moves only in one direction is known as forward only ResultSet. By default, JDBC result sets are forward-only result sets.

You can move the cursor of the forward only ResultSets using the next() method of the ResultSet interface. It moves the pointer to the next row from the current position. This method returns a boolean value. If there are no rows next to its current position it returns false, else it returns true.

Therefore, using this method in the while loop you can iterate the contents of the ResultSet object.

```
while(rs.next()){
}
```

2. Bidirectional ResultSet:

A bi-directional ResultSet object is the one whose cursor moves in both forward and backward directions.

The createStatement() method of the Connection interface has a variant which accepts two integer values representing the result set type and the concurrency type.

Statement createStatement(int resultSetType, int resultSetConcurrency)

To create a bi-directional result set you need to pass the type as ResultSet.TYPE_SCROLL_SENSITIVE or ResultSet.TYPE_SCROLL_INSENSITIVE, along with the concurrency, to this method as:

```
//Creating a Statement object
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
```

==========

Difference between RMI and CORBA

RMI	CORBA
RMI is a Java-specific technology.	CORBA has implementation for many languages.
It uses Java interface for implementation.	It uses Interface Definition Language (IDL) to separate interface from implementation.
RMI objects are garbage collected automatically.	CORBA objects are not garbage collected because it is language independent and some languages like C++ does not support garbage collection.
RMI programs can download new classes from remote JVM's.	CORBA does not support this code sharing mechanism.
RMI passes objects by remote reference or by value.	CORBA passes objects by reference.
The responsibility of locating an object implementation falls on JVM.	The responsibility of locating an object implementation falls on Object Adapter either Basic Object Adapter or Portable Object Adapter.
RMI uses the Java Remote Method Protocol as its underlying remoting protocol.	CORBA use Internet Inter-ORB Protocol as its underlying remoting protocol.

==========

==========

why do we use panels while creating GUI program in java?

By using Panels it's easier to group components and arrange them in a Frame. Every Panel can have it's own layout. So you could give your outer Frame a BorderLayout and put a Panel into the center.
The Panel then can have let's say a GridBagLayout for it's components.
A panel is a container.

- It allows you to group components together
- It allows you to devise complex interfaces, as each panel can have a different layout, allowing you to leverage the power of different layout managers.
- It allows you to build reusable components and isolate responsibility
- But most of all, it gives you the bases for deciding how the panel should be deployed. With a panel you can add it to a frame or applet or another component as needed...
- A panel also makes a good surface onto which to perform custom painting. For all the benefits mentioned above - its isolated and reusable...

==========

Cookies and using handling cookies in Java

Cookies are the textual information that is stored in key-value pair format to the client's browser during multiple requests. It is one of the state management techniques in session tracking. Basically, the server treats every client request as a new one so to avoid this situation cookies are used. When the client generates a request, the server gives the response with cookies having an id which are then stored in the client's browser. Thus if the client generates a second request, a cookie with the matched id is also sent to the server. The server will fetch the cookie id, if found it will treat it as an old request otherwise the request is considered new.

Methods in Cookies

1) clone(): Overrides the standard java.lang.Object.clone method to return a copy of this Cookie.
2) getComment(): Returns the comment describing the purpose of this cookie, or null if the cookie has no comment.
3) getDomain(): Gets the domain name of this Cookie.
4) getMaxAge(): Gets the maximum age in seconds of this Cookie.
5) getName(): Returns the name of the cookie.
6) getPath(): Returns the path on the server to which the browser returns this cookie.
7) getSecure(): Returns true if the browser is sending cookies only over a secure protocol, or false if the browser can send cookies using any protocol.
8) getValue(): Gets the current value of this Cookie.
9) getVersion(): Returns the version of the protocol this cookie complies with.
10) setValue(String newValue): Assigns a new value to this Cookie.
11) setVersion(int v): Sets the version of the cookie protocol that this Cookie complies with.

What is an Event Handling and describe the components in Event Handling in Java? Explain with example ?

The GUI in Java processes the interactions with users via mouse, keyboard and various user controls such as button, checkbox, text field, etc. as the events. These events are to be handled properly to implement Java as an Event-Driven Programming.

Components in Event Handling

- Events
- Event Sources
- Event Listeners/Handlers

1. Events
- The events are defined as an object that describes a change in the state of a source object.
- The Java defines a number of such Event Classes inside java.awt.event package
- Some of the events are ActionEvent, MouseEvent, KeyEvent, FocusEvent, ItemEvent and etc.

2. Event Sources
- A source is an object that generates an event.
- An event generation occurs when an internal state of that object changes in some way.
- A source must register listeners in order for the listeners to receive the notifications about a specific type of event.
- Some of the event sources are Button, CheckBox, List, Choice, Window and etc.

3. Event Listeners
- A listener is an object that is notified when an event occurs.
- A Listener has two major requirements, it should be registered to one more source object to receiving event notification and it must implement methods to receive and process those notifications.
- Java has defined a set of interfaces for receiving and processing the events under the java.awt.event package.
- Some of the listeners are ActionListener, MouseListener, ItemListener, KeyListener, WindowListener and etc.

Example

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.  import javax.swing.*;
4.  public class EventListenerTest extends JFrame implements ActionListener {
5.  JButton button;
6.  public static void main(String args[]) {
7.  EventListenerTest object = new EventListenerTest();
8.  object.createGUI();
9.  }
10. void createGUI() {
11. button = new JButton(" Click Me !");
12. setSize(300,200);
13. setLocationRelativeTo(null);
14. setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15. setVisible(true);
16. add(button);
17. button.addActionListener(this);
18. }
19. public void actionPerformed(ActionEvent ae) {
20. if(ae.getSource() == button) {
21. JOptionPane.showMessageDialog(null, "Generates an Action Event");
22. }
23. }
24. }
```

==========

Marshalling and Unmarshalling

Whenever a client invokes a method that accepts parameters on a remote object, the parameters are bundled into a message before being sent over the network. These parameters may be of primitive type or objects. In case of primitive type, the parameters are put together and a header is attached to it. In case the parameters are objects, then they are serialized. This process is known as marshalling.

At the server side, the packed parameters are unbundled and then the required method is invoked. This process is known as unmarshalling.

==========

SimplePropertyBeanMain.java(java bean Example)

```
public class SimplePropertyBeanMain
{
  public static void main(String args[])
  {
    SimplePropertyBean ob=new SimplePropertyBean();
    ob.setLength(5);
    ob.setBreadth(3);
    ob.setHeight(2);
    System.out.println("Volume
:"+ob.getLength()*ob.getBreadth()*ob.getHeight());
} }
```

==========

JavaBean and its properties with example.

A JavaBean is a Java class that should follow the following conventions:

- It should have a no-arg constructor.
- It should be Serializable.
- It should provide methods to set and get the values of the properties, known as getter and setter methods.

Why use JavaBean?

According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object so that we can access this object from multiple places. Moreover, it provides easy maintenance.

Simple example of JavaBean class

```
1.  //Employee.java
2.
3.  package mypack;
4.  public class Employee implements java.io.Serializable{
5.  private int id;
6.  private String name;
7.  public Employee(){}
8.  public void setId(int id){this.id=id;}
9.  public int getId(){return id;}
10. public void setName(String name){this.name=name;}
11. public String getName(){return name;}
12. }
```

How to access the JavaBean class?

To access the JavaBean class, we should use getter and setter methods.

```
1.  package mypack;
2.  public class Test{
3.  public static void main(String args[]){
4.  Employee e=new Employee();//object is created
5.  e.setName("Arjun");//setting value to the object
6.  System.out.println(e.getName());
7.  }}
```

Note: There are two ways to provide values to the object. One way is by constructor and second is by setter method.

JavaBean Properties

A JavaBean property is a named feature that can be accessed by the user of the object. The feature can be of any Java data type, containing the classes that you define.

A JavaBean property may be read, write, read-only, or write-only. JavaBean features are accessed through two methods in the JavaBean's implementation class:

1. getPropertyName ()

For example, if the property name is firstName, the method name would be getFirstName() to read that property. This method is called the accessor.

2. setPropertyName ()

For example, if the property name is firstName, the method name would be setFirstName() to write that property. This method is called the mutator.

Advantages of JavaBean

The following are the advantages of JavaBean:/p>

- The JavaBean properties and methods can be exposed to another application.
- It provides an easiness to reuse the software components.

Disadvantages of JavaBean

The following are the disadvantages of JavaBean:

- JavaBeans are mutable. So, it can't take advantages of immutable objects.
- Creating the setter and getter method for each property separately may lead to the boilerplate code.

==========

Working with 2D Shapes

Starting with Java 1.0, the Graphics class from java.awt package has methods to draw strings, lines, rectangles, ellipses, and so on. For example to draw line, we use drawLine(x1, y1, x2, y2) method as show below,

```
class SimplePanel extends JPanel
{
 public void paintComponent(Graphics g)
 {
 g.drawLine(10,10,100,100);
 }
}
```

To draw rectangles, we use drawRect(x1, y1, w, h) method and to draw string we use
drawString(◦String◦, x1, y1) and so on. The drawings using Graphics class methods are very limited. For example, you cannot vary the line thickness and cannot rotate the shapes.

program :

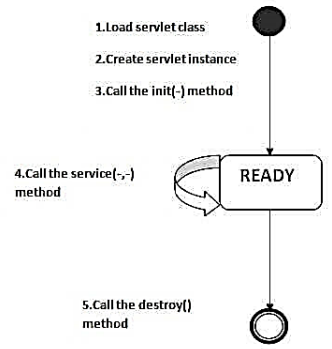
```
class SimplePanel extends JPanel
{
 public void paintComponent(Graphics g)
 {
 Graphics2D g2d = (Graphics2D)g;
Rectangle2D floatRect = new Rectangle2D.Float (10, 15.5F, 52.5F,
60.0F);
g2d.draw(floatRect);
 }
}
```

==========

Life Cycle of a Servlet (Servlet Life Cycle)

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

- Servlet class is loaded.
- Servlet instance is created.
- init method is invoked.
- service method is invoked.
- destroy method is invoked.



As displayed in the above diagram, there are three states of a servlet: new, ready and end. The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

1) Servlet class is loaded
The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

2) Servlet instance is created
The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

3) init method is invoked
The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:
public void init(ServletConfig config) throws ServletException

4) service method is invoked
The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

public void service(ServletRequest request, ServletResponse response)
throws ServletException, IOException

5) destroy method is invoked
The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:

public void destroy()

==========

Servlet vs JSP

Servlet	JSP
Servlets are faster as compared to JSP, as they have a short response time.	JSP is slower than Servlets, as the first step in the JSP lifecycle is the conversion of JSP to Java code and then the compilation of the code.
Servlets are Java-based codes.	JSP are HTML-based codes.
Servlets are harder to code, as here, the HTML codes are written in Java.	JSPs are easier to code, as here Java is coded in HTML.
In an MVC architecture, Servlets act as the controllers.	In MVC architectures, the JSPs act as a view to present the output to the users.
The service() function can be overridden in Servlets.	The service() function cannot be overridden in JSPs.
The Servlets are capable of accepting all types of protocol requests.	The JSPs are confined to accept only the HTTP requests.
Modification in Servlets is a time-consuming and challenging task, as here, one will have to reload, recompile, and then restart the servers.	Modification is easy and faster in JSPs as we just need to refresh the pages.

What is a LayoutManager and types of LayoutManager in Java?

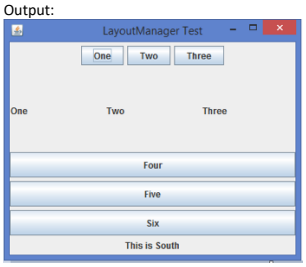
The Layout managers enable us to control the way in which visual components are arranged in the GUI forms by determining the size and position of components within the containers.
Types of LayoutManager
There are 6 layout managers in Java

- **FlowLayout**: It arranges the components in a container like the words on a page. It fills the top line from **left to right and top to bottom**. The components are arranged in the order as they are added i.e. first components appears at top left, if the container is not wide enough to display all the components, it is wrapped around the line. Vertical and horizontal gap between components can be controlled. The components can be **left, center or right aligned.**
- **BorderLayout**: It arranges all the components along the edges or the middle of the container i.e. **top, bottom, right and left** edges of the area. The components added to the top or bottom gets its preferred height, but its width will be the width of the container and also the components added to the left or right gets its preferred width, but its height will be the remaining height of the container. The components added to the center gets neither its preferred height or width. It covers the remaining area of the container.
- **GridLayout**: It arranges all the components in a grid of **equally sized cells**, adding them from the **left to right** and **top to bottom**. Only one component can be placed in a cell and each region of the grid will have the same size. When the container is resized, all cells are automatically resized. The order of placing the components in a cell is determined as they were added.
- **GridBagLayout**: It is a powerful layout which arranges all the components in a grid of cells and maintains the aspect ration of the object whenever the container is resized. In this layout, cells may be different in size. It assigns a consistent horizontal and vertical gap among components. It allows us to specify a default alignment for components within the columns or rows.
- **BoxLayout**: It arranges multiple components in either **vertically or horizontally**, but not both. The components are arranged from **left to right or top to bottom**. If the components are aligned **horizontally**, the height of all components will be the same and equal to the largest sized components. If the components are aligned **vertically**, the width of all components will be the same and equal to the largest width components.
- **CardLayout**: It arranges two or more components having the same size. The components are **arranged in a deck**, where all the cards of the same size and the **only top card are visible at any time**. The first component added in the container will be kept at the top of the deck. The default gap at the left, right, top and bottom edges are zero and the card components are displayed either **horizontally or vertically.**

Example

```
1)  import java.awt.*;
2)  import javax.swing.*;
3)  public class LayoutManagerTest extends JFrame {
4)  JPanel flowLayoutPanel1, flowLayoutPanel2,
    gridLayoutPanel1, gridLayoutPanel2, gridLayoutPanel3;
5)  JButton one, two, three, four, five, six;
6)  JLabel bottom, lbl1, lbl2, lbl3;
7)  public LayoutManagerTest() {
8)  setTitle("LayoutManager Test");
9)  setLayout(new BorderLayout());
10) // Set BorderLayout for JFrame
11) flowLayoutPanel1 = new JPanel();
12) one = new JButton("One");
13) two = new JButton("Two");
14) three = new JButton("Three");
15) flowLayoutPanel1.setLayout(new
    FlowLayout(FlowLayout.CENTER));
16) // Set FlowLayout Manager
17) flowLayoutPanel1.add(one);
18) flowLayoutPanel1.add(two);
19) flowLayoutPanel1.add(three);
20) flowLayoutPanel2 = new JPanel();
21) bottom = new JLabel("This is South");
22) flowLayoutPanel2.setLayout (new
    FlowLayout(FlowLayout.CENTER))
    ; // Set FlowLayout Manager
23) flowLayoutPanel2.add(bottom);
24) gridLayoutPanel1 = new JPanel();
25) gridLayoutPanel2 = new JPanel();
26) gridLayoutPanel3 = new JPanel();
27) lbl1 = new JLabel("One");
28) lbl2 = new JLabel("Two");
29) lbl3 = new JLabel("Three");
30) four = new JButton("Four");
31) five = new JButton("Five");
32) six = new JButton("Six");
33) gridLayoutPanel2.setLayout(new GridLayout(1, 3, 5, 5));
    // Set GridLayout Manager
34) gridLayoutPanel2.add(lbl1);
35) gridLayoutPanel2.add(lbl2)
36) gridLayoutPanel2.add(lbl3);
37) gridLayoutPanel3.setLayout(new GridLayout(3, 1, 5,
    5)); // Set GridLayout Manager
38) gridLayoutPanel3.add(four);
39) gridLayoutPanel3.add(five);
40) gridLayoutPanel3.add(six);
41) gridLayoutPanel1.setLayout(new GridLayout(2, 1)); //
    Set GridLayout Manager
42) gridLayoutPanel1.add(gridLayoutPanel2);
43) gridLayoutPanel1.add(gridLayoutPanel3);
44) add(flowLayoutPanel1, BorderLayout.NORTH);
45) add(flowLayoutPanel2, BorderLayout.SOUTH);
46) add(gridLayoutPanel1, BorderLayout.CENTER);
47) setSize(400, 325);
48) setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
49) setLocationRelativeTo(null);
50) setVisible(true);
51) }
52) public static void main(String args[]) {
53) new LayoutManagerTest();
54) }
55) }
```

Output:

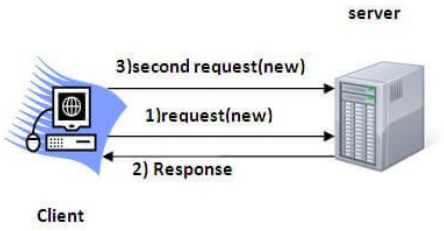


==========

Session Tracking in Servlets

Session simply means a particular interval of time.
Session Tracking is a way to maintain state (data) of an user. It is also known as session management in servlet.

Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.
HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:



We use session tracking To recognize the user It is used to recognize the particular user.
Session Tracking Techniques
There are four techniques used in Session tracking:

- Cookies
- Hidden Form Field
- URL Rewriting
- HttpSession

==========

Difference between scrollable and updateable resultSet with example.

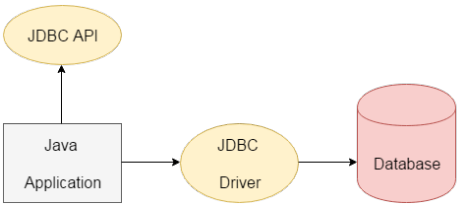
Scrollable result set	Updateable result set
you can move forward and backward through a result set and even jump to any position.	you can programmatically update entries so that the database is automatically updated.
Statement stmt = conn.createStatement(type, concurrency);	Statement stmt =conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);

==========

Java JDBC and it's drivers types

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

- JDBC-ODBC Bridge Driver,
- Native Driver,
- Network Protocol Driver, and
- Thin Driver



We can use JDBC API to handle database using Java program and can perform the following activities:

- Connect to the database
- Execute queries and update statements to the database
- Retrieve the result received from the database.

1) JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

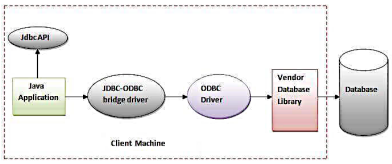


Figure- JDBC-ODBC Bridge Driver

2) Native-API driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

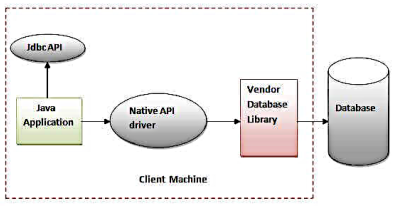


Figure- Native API Driver

3) Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

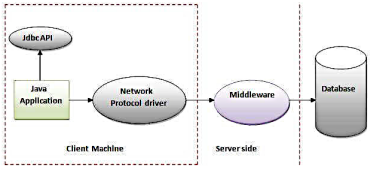


Figure- Network Protocol Driver

4) Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

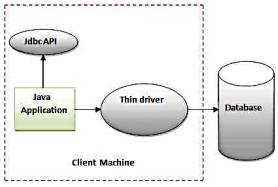


Figure- Thin Driver

What is WEB Framework in Java

Java Framework is the body or platform of pre-written codes used by Java developers to develop Java applications or web applications. In other words, Java Framework is a collection of predefined classes and functions that is used to process input, manage hardware devices interacts with system software. It acts like a skeleton that helps the developer to develop an application by writing their own code.

Spring

It is a light-weighted, powerful Java application development framework. It is used for JEE. Its other modules are Spring Security, Spring MVC, Spring Batch, Spring ORM, Spring Boot and Spring Cloud etc.

Hibernate

Hibernate is ORM (Object-Relation Mapping) framework that allows us establish communication between Java programming language and the RDBMS.

Grails

It is a dynamic framework created by using the Groovy programming language. It is an OOPs language. Its purpose is to enhance the productivity. The syntax of Grails is matched with Java and the code is compiled to JVM. It also works with Java, JEE, Spring and Hibernate.

==========

Difference between Library and framework

Library	Framework
Library is the collection of frequently used, pre-compiled classes.	Framework is the collection of libraries.
It is a set of reusable functions used by computer programs.	It is a piece of code that dictates the architecture of your project and aids in programs.
You are in full control when you call a method from a library and the control is then returned.	The code never calls into a framework, instead the framework calls you.
It is in corporate seamlessly into existing projects to add functionality that you can access using an API.	It cannot be seamlessly incorporated into an existing project. Instead it can be used when a new project is started.
They are important in program for linking and binding process.	They provide a standard way to build and deploy applications.
Libraries do no employ an inverted flow of control between itself and its clients.	Framework employs an inverted flow of control between itself and its clients.
Example: jQuery is a JavaScript library that simplifies DOM manipulation.	**Example:** Angular JS is a JavaScript-based framework for dynamic web applications.

==========

Write a program object to read personal details from HTML file or another JSP file.

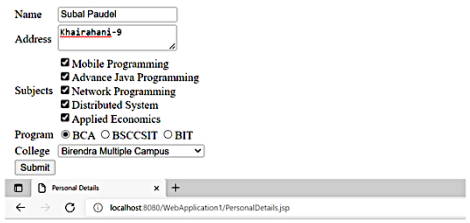
```
<html>
  <head>
    <title>Personal Details</title>
  </head>
  <body>
    <table>
      <form name="form1" action="PersonalDetails.jsp" method="post">
        <tr>
          <td>Name</td>
          <td><input type="text" name="txtName"></td>
        </tr>
        <tr>
          <td>Address</td>
          <td><textarea name="txtaddress" rows="2" cols="20"></textarea></td>
        </tr>
        <tr>
          <td>Subjects</td>
          <td><input id="ch1" type="checkbox" name="subject" value="Mobile Programming">Mobile Programming<br>
          <input id="ch2" type="checkbox" name="subject" value="Advance Java Programming">Advance Java Programming<br>
          <input id="ch3" type="checkbox" name="subject" value="Network Programming">Network Programming<br>
          <input id="ch4" type="checkbox" name="subject" value="Distributed System">Distributed System<br>
          <input id="ch5" type="checkbox" name="subject" value="Applied Economics">Applied Economics
          </td>
        </tr>
            <tr>
              <td>Program</td>
              <td><input type="radio" name="program" checked value="BCA">BCA
                <input type="radio" name="program" value="BSCCSIT">BSCCSIT
                <input type="radio" name="program" value="BIT">BIT
              </td>
            </tr>
            <tr>
              <td>College</td>
              <td><select name="college">
                  <option value="Birendra Multiple Campus">Birendra Multiple Campus</option>
                  <option value="Sahed Smarak College">Sahed Smarak College</option>
                  <option value="Saptagandaki Multiple Campus">Saptagandaki Multiple Campus</option>
                  <option value="Sungava College">Sungava College</option>
                  <option value="Indreni College">Indreni College</option>
                  <option value="Lumbini ICT College">Lumbini ICT College</option>
                </select>
              </td>
            </tr>
            <tr>
              <td><input type="submit" name="btnSubmit"></td>
            </tr>
      </form>
    </table>
  </body>
</html>
```

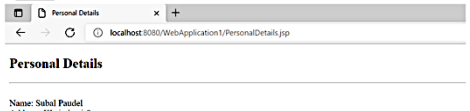
---------->

PersonalDetails.jsp

```
<html>
    <head>
        <title>Personal Details</title>
    </head>
    <body>
        <h2>Personal Details</h2>
        <hr>
        <%
            String name=request.getParameter("txtName");
            String address=request.getParameter("txtaddress");
            String[] subjects=request.getParameterValues("subject");
            String program=request.getParameter("program");
            String college=request.getParameter("college");
            out.println("<br>Name: "+name);
            out.println("<br>Address: "+address);
            out.println("<br>Subjects: ");
            for(int i=0;i<subjects.length;i++)
            {
                out.print(subjects[i]);
                if(i<subjects.length-1)
                out.print(", ");
            }
            out.println("<br>Program: "+program);
            out.println("<br>College: "+college);
        %>
    </body>
</html>
```

Output





==========

Differentiate between rowset and resultset

ResultSet	RowSet
A ResultSet always maintains connection with the database.	A RowSet can be connected, disconnected from the database.
It cannot be serialized.	A RowSet object can be serialized.
ResultSet object cannot be passed other over network.	You can pass a RowSet object over the network.
ResultSet object is not a JavaBean objectYou can create/get a result set using the executeQuery() method.	ResultSet Object is a JavaBean object.You can get a RowSet using the RowSetProvider.newFactory().createJdbcRowSet() method.
By default, ResultSet object is not scrollable or, updatable.	By default, RowSet object is scrollable and updatable.

==========

1. Statement :

It is used for accessing your database. Statement interface cannot accept parameters and useful when you are using static SQL statements at runtime. If you want to run SQL query only once then this interface is preferred over PreparedStatement.

Example –

```
//Creating The Statement Object
Statement GFG = con.createStatement();
//Executing The Statement
GFG.executeUpdate("CREATE TABLE STUDENT(ID NUMBER NOT NULL,
NAME VARCHAR)");
```

2. PreparedStatement :

It is used when you want to use SQL statements many times. The PreparedStatement interface accepts input parameters at runtime.

Example –

```
//Creating the PreparedStatement object
PreparedStatement GFG = con.prepareStatement("update STUDENT
set NAME = ? where ID = ?");
//Setting values to place holders
//Assigns "RAM" to first place holder
GFG.setString(1, "RAM");
//Assigns "512" to second place holder
GFG.setInt(2, 512);
//Executing PreparedStatement
GFG.executeUpdate();
```

==========
==

JDBC Configuration to execute sql

In this example we are using MySql as the database. So we need to know following information's for the mysql database:

1. Load a JDBC Driver class

To load a class at runtime into JVM, we need to call a static method called forName() of java.lang.Class. By calling the forName() method we can load the JDBC Driver class into the JVM.

2. Establish a Connection

By using the DriverManager class, we can establish the connection to the database. By calling the getConnection(String url, String user,String password) static factory method in DriverManager class, we can obtain a Connection object. To get the connection object we need to proved the connection url as a first parameter and database username and password as second and third parameters.

3. Create a Statement

Create a Statement : In order to send the SQL commands to database from our java program, we need Statement object. We can get the Statement object by calling the createStatement() method on connection.

4. Execute Sql queries

Inorder to execute the SQL commands on database, Statement interface provides provides three different methods: executeUpdate(), executeQuery() ,execute().

5. Process the ResultSet

For the select operations, we use executeQuery() method. The executed query returns the data in the form of ResultSet object. To process the data we need to go through the ResultSet.

6. Close Connection

Last but not least, finally we need to close the connection object. It is very important step to close the connection. Else we may get JDBCConnectionException exception.

Conn.close();

To connect java application with the mysql database, mysqlconnector.jar file is
required to be loaded.
download the jar file mysql-connector.jar

Example

```
import java.sql.*;
import javax.swing.*;
public class JDBCProgram {
 public static void main(String args[]) {
 try
 {
 //step-1 Load a JDBC Driver class
 Class.forName("com.mysql.cj.jdbc.Driver");
 //step-2 Establish a Connection
 Connection
conn=DriverManager.getConnection("jdbc:mysql://localhost:3307/tes
t");
 //step-3 Create a Statement
 Statement stmt=conn.createStatement();
 String sql="select * from student";
 //step-4 Execute Sql queries
 ResultSet rs=stmt.executeQuery(sql);
 //step-5 Process the ResultSet
 while(rs.next())
 {
 System.out.println(rs.getInt(2)+" "+rs.getString(3)+"
"+rs.getString(4));
}
//step-6 Close Connection
 conn.close();
 }
 catch(Exception e)
 {
 JOptionPane.showMessageDialog(null,e);
 }
}
}
```

```
run:
101 Ram Chitwan
102 Mahesh Butwal
105 Nabin Tandi
103 Kalpana Pokhara
BUILD SUCCESSFUL (total time: 0 seconds)
```

==========

Write a program to calculate simple interest using swing?

```
1)  import javax.swing.*;
2)  import java.awt.*;
3)  import java.awt.event.*;
4)  public class SimpleIntrest extends JFrame implements
    ActionListener
5)  {

6)  JLabel lblPrinciple = new JLabel("Principle");
7)  JLabel lblRate = new JLabel("Rate");
8)  JLabel lblTime = new JLabel("Time");
9)  JLabel lblResult = new JLabel("Result");

10) JTextField txtPrinciple = new JTextField();
11) JTextField txtTime = new JTextField();
12) JTextField txtRate = new JTextField();
13) JTextField txtResult = new JTextField();

14) SimpleIntrest()
15) {
16) setSize(400,500);
17) setTitle("Simple Intrest");
18) setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

19) setLayout(null);

20) JButton btn = new JButton("Compute");
21) btn.addActionListener(this);

22) lblPrinciple.setBounds(50,20,60,20);
23) lblTime.setBounds(50,50,60,20);
24) lblRate.setBounds(50,80,60,20);
25) lblResult.setBounds(50,110,60,20);

26) txtPrinciple.setBounds(150,20,100,20);
27) txtTime.setBounds(150,50,100,20);
28) txtRate.setBounds(150,80,100,20);
29) txtResult.setBounds(150,110,100,20);

30) btn.setBounds(150,140,100,20);

31) add(lblPrinciple);
32) add(lblTime);
33) add(lblRate);
34) add(lblResult);

35) add(txtPrinciple);
36) add(txtTime);
37) add(txtRate);
38) add(txtResult);

39) add(btn);

40) setVisible(true);
41) }

42) public void actionPerformed(ActionEvent ae)
43) {
44) int p,t,r;
45) float res;
46) p = Integer.parseInt(txtPrinciple.getText());
47) t = Integer.parseInt(txtTime.getText());
48) r = Integer.parseInt(txtRate.getText());
49) res=(p*t*r)/100;
50) txtResult.setText(String.valueOf(res));
51) }

52) public static void main(String args[])
53) {
54) new SimpleIntrest();
55) }
56) }
```

==========

Differentiate between statement and prepared Statement,

Statement	PreparedStatement
It is used when SQL query is to be executed only once.	It is used when SQL query is to be executed multiple times.
You can not pass parameters at runtime.	You can pass parameters at runtime.
Used for CREATE, ALTER, DROP statements.	Used for the queries which are to be executed multiple times.
Performance is very low.	Performance is better than Statement.
It is base interface.	It extends statement interface.
Used to execute normal SQL queries.	Used to execute dynamic SQL queries.
We can not use statement for reading binary data.	We can use Preparedstatement for reading binary data.
It is used for DDL statements.	It is used for any SQL Query.
We can not use statement for writing binary data.	We can use Preparedstatement for writing binary data.
No binary protocol is used for communication.	Binary protocol is used for communication.

The following JSP program calculates factorial values for an integer number, while the input is taken from an HTML form.

input.html

```
1)  <html>
2)  <body>
3)  <form action="Factorial.jsp">
4)  Enter a value for n: <input type="text" name="val">
5)  <input type="submit" value="Submit">
6)  </form>
7)  </body>
8)  </html>
```

Factorial.jsp

```
1)  <html>
2)  <body>
3)  <%!
4)  long n, result;
5)  String str;

6)  long fact(long n) {
7)  if(n==0)
8)  return 1;
9)  else
10) return n*fact(n-1);
11) }
12) %>
13) <%
14) str = request.getParameter("val");
15) n = Long.parseLong(str);
16) result = fact(n);
17) %>
18) <b>Factorial value: </b> <%= result %>
19) </body>
20) </html>
```

==========

Creating Frames using Swings in Java with examples:

Swing is a part of JFC (Java Foundation Classes). Building Graphical User Interface in Java requires the use of Swings. Swing Framework contains a large set of components that allow a high level of customization and provide rich functionalities and is used to create window-based applications.

Java swing components are lightweight, platform-independent, provide powerful components like tables, scroll panels, buttons, lists, color chooser, etc. In this article, we'll see how to make frames using Swings in Java. Ways to create a frame:

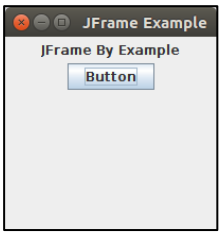
Methods:

- By creating the object of Frame class (association)
- By extending Frame class (inheritance)
- Create a frame using Swing inside main()

JFrame Example

```
1)  import java.awt.FlowLayout;
2)  import javax.swing.JButton;
3)  import javax.swing.JFrame;
4)  import javax.swing.JLabel;
5)  import javax.swing.JPanel;
6)  public class JFrameExample {
7)  public static void main(String s[]) {
8)  JFrame frame = new JFrame("JFrame Example");
9)  JPanel panel = new JPanel();
10) panel.setLayout(new FlowLayout());
11) JLabel label = new JLabel("JFrame By Example");
12) JButton button = new JButton();
13) button.setText("Button");
14) panel.add(label);
15) panel.add(button);
16) frame.add(panel);
17) frame.setSize(200, 300);
18) frame.setLocationRelativeTo(null);
19) frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOS
    E);
20) frame.setVisible(true);
21) }
22) }
```

Output:



Explain JCheckBOX and JRadio Button component of Java swing library?

Java JCheckBox

The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ".It inherits JToggleButton class.

Java JCheckBox Example

```
1)  import javax.swing.*;
2)  public class CheckBoxExample
3)  {
4)  CheckBoxExample(){
5)  JFrame f= new JFrame("CheckBox Example");
6)  JCheckBox checkBox1 = new JCheckBox("C++");
7)  checkBox1.setBounds(100,100, 50,50);
8)  JCheckBox checkBox2 = new JCheckBox("Java", true);
9)  checkBox2.setBounds(100,150, 50,50);
10) f.add(checkBox1);
11) f.add(checkBox2);
12) f.setSize(400,400);
13) f.setLayout(null);
14) f.setVisible(true);
15) }
16) public static void main(String args[])
17) {
18) new CheckBoxExample();
19) }}
```

Java JRadioButton

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.

It should be added in ButtonGroup to select one radio button only.

Java JRadioButton Example

```
1)  import javax.swing.*;
2)  public class RadioButtonExample {
3)  JFrame f;
4)  RadioButtonExample(){
5)  f=new JFrame();
6)  JRadioButton r1=new JRadioButton("A) Male");
7)  JRadioButton r2=new JRadioButton("B) Female");
8)  r1.setBounds(75,50,100,30);
9)  r2.setBounds(75,100,100,30);
10) ButtonGroup bg=new ButtonGroup();
11) bg.add(r1);bg.add(r2);
12) f.add(r1);f.add(r2);
13) f.setSize(300,300);
14) f.setLayout(null);
15) f.setVisible(true);
16) }
17) public static void main(String[] args) {
18) new RadioButtonExample();
19) }
20) }
```

==========

What is javabean ?how it is different from java class?

A JavaBean is a Java class that follows a certain standard:

- JavaBean class properties should be set to private – accessing /mutating these properties is done using getters/setters methods.
- A JavaBean class should have a public no-argument constructor.
- A JavaBean class should implement the java.io.Serializable interface. The Serializable interface allows the saving, storing, and restoring of a JavaBean's state while in use.

The only difference between both the classes is Java make java beans objects serialized so that the state of a bean class could be preserved in case required.So due to this a Java Bean class must either implements Serializable or Externalizable interface.

Example of javabeans

```
public class Employee implements java.io.Serializable {
  private int id;
  private String name;
  public Employee(){}
  public void setId(int id){this.id=id;}
  public int getId(){return id;}
  public void setName(String name){this.name=name;}
  public String getName(){return name;}
}
```

Write a JSP script for enabling and disabling session?

```
<%@ page language="java" session="false"%>
<html>
<head>
<title>Session Disabled</title>
</head>
<body>
<p>Session is Disabled in this page
</body>
</html>

<%@ page language="java" session="true"%>
<html>
<head>
<title>Session Enabled</title>
</head>
<body>
<p>Session is Enabled in this page
</body>
</html>
```

==========

jsp form program to add two numbers .

index.html

```
<html>
<head>
<title>calculator </title>
<meta charset="UTF-8">
<meta name="viewportal" content = "width=device-width,initial-scale=1.0">
</head>
<body>
<form action ="add.jsp">
Enter first number : <input type="text" name="num1"><br><br>
Enter second number:<input type="text" name="num2"><br><br>
<input type="submit" value="sum">
</body>
</html>
```

JSP for adding two number : add.jsp

```
<html>
<head>
<meta http-equiv="Content-type" content="text/html; charset=UTF-8">
<title>JSP page for result sum </title>
</head>
<body>
<h1>sum</h1>
<%="<h4>sum is " + (Integer.parseInt(request.getParameter("num1"))+
Integer.parseInt(request.getParameter("num2")))+"</h4>"%>
</body>
</html>
```

==========

write a swing program to connect database and display student/employee details in Jtable.

```
1)  import java.awt.*;
2)  import java.awt.event.*;
3)  import java.sql.*;
4)  import java.util.Vector;
5)  import javax.swing.*;
6)  import javax.swing.table.DefaultTableModel;
7)  public class DisplayEmpData extends JFrame
    implements ActionListener {
8)  JFrame frame1;
9)  JLabel l0, l1, l2;
10) JComboBox c1;
11) JButton b1;
12) Connection con;
13) ResultSet rs, rs1;
14) Statement st, st1;
15) PreparedStatement pst;
16) String ids;
17) static JTable table;
18) String[] columnNames = {"User name", "Email",
    "Password", "Country"};
19) String from;
20) DisplayEmpData() {
21) l0 = new JLabel("Fatching Employee Information");
22) l0.setForeground(Color.red);
23) l0.setFont(new Font("Serif", Font.BOLD, 20));
24) l1 = new JLabel("Select name");
25) b1 = new JButton("submit");
26) l0.setBounds(100, 50, 350, 40);
27) l1.setBounds(75, 110, 75, 20);
28) b1.setBounds(150, 150, 150, 20);
29) b1.addActionListener(this);
30) setTitle("Fetching Student Info From DataBase");
31) setLayout(null);
32) setVisible(true);
33) setSize(500, 500);
34) setDefaultCloseOperation(WindowConstants.EXIT_ON_
    CLOSE);
35) add(l0);
36) add(l1);;
37) add(b1);
38) try {
39) Class.forName("oracle.jdbc.driver.OracleDriver");
40) con =
    DriverManager.getConnection("jdbc:oracle:thin:@mcnd
    esktop07:1521:xe", "sandeep", "welcome");
41) st = con.createStatement();
42) rs = st.executeQuery("select uname from emp");-------→
43) Vector v = new Vector();
44) while (rs.next()) {
45) ids = rs.getString(1);
46) v.add(ids);
47) }
48) c1 = new JComboBox(v);
49) c1.setBounds(150, 110, 150, 20);
50) add(c1);
51) st.close();
52) rs.close();
53) } catch (Exception e) {
54) }
55) }
56) public void actionPerformed(ActionEvent ae) {
57) if (ae.getSource() == b1) {
58) showTableData();
59) }
60) }
61) public void showTableData() {
62) frame1 = new JFrame("Database Search Result");
63) frame1.setDefaultCloseOperation(JFrame.EXIT_ON_CLO
    SE);
64) frame1.setLayout(new BorderLayout());
65) //TableModel tm = new TableModel();
66) DefaultTableModel model = new DefaultTableModel();
67) model.setColumnIdentifiers(columnNames);
68) //DefaultTableModel model = new
    DefaultTableModel(tm.getData1(),
    tm.getColumnNames());
69) //table = new JTable(model);
70) table = new JTable();
71) table.setModel(model);
72) table.setAutoResizeMode(JTable.AUTO_RESIZE_ALL_CO
    LUMNS);
73) table.setFillsViewportHeight(true);
74) JScrollPane scroll = new JScrollPane(table);
75) scroll.setHorizontalScrollBarPolicy(
76) JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
77) scroll.setVerticalScrollBarPolicy(
78) JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED);
79) from = (String) c1.getSelectedItem();
80) //String textvalue = textbox.getText();
81) String uname = "";
82) String email = "";
83) String pass = "";
84) String cou = "";
85) try {
86) pst = con.prepareStatement("select * from emp where
    UNAME='" + from + "'");
87) ResultSet rs = pst.executeQuery();
88) int i = 0;
89) if (rs.next()) {
90) uname = rs.getString("uname");
91) email = rs.getString("umail");
92) pass = rs.getString("upass");
93) cou = rs.getString("ucountry");
94) model.addRow(new Object[]{uname, email, pass, cou});
95) i++;
96) }
97) if (i < 1) {
98) JOptionPane.showMessageDialog(null, "No Record
    Found", "Error", JOptionPane.ERROR_MESSAGE);
99) }
100) if (i == 1) {
101) System.out.println(i + " Record Found");
102) } else {
103) System.out.println(i + " Records Found");
104) }
105) } catch (Exception ex) {
106) JOptionPane.showMessageDialog(null, ex.getMessage(),
    "Error", JOptionPane.ERROR_MESSAGE);
107) }
108) frame1.add(scroll);
109) frame1.setVisible(true);
110) frame1.setSize(400, 300);
111) }
112) public static void main(String args[]) {
113) new Displaystudent Data();
114) }
115) }
```

==========

Write a swing program to get two number input using two text field and multiply it by clicking button and display it in third text field.

```
1)  import java.awt.FlowLayout;
2)  import java.awt.event.*;
3)  import javax.swing.*;
4)  public class MultiplyUsingSwing extends JFrame
    implements ActionListener {
5)  JLabel l1 = new JLabel("First No:");
6)  JTextField t1 = new JTextField(10);
7)  JLabel l2 = new JLabel("Second No:");
8)  JTextField t2 = new JTextField(10);
9)  JTextField t3 = new JTextField(10);
10) JButton b = new JButton("Multiply");
11) String fno = "";
12) String sno = "";
13) public MultiplyUsingSwing() {
14) setLayout(new FlowLayout());
15) setSize(250, 250);
16) setVisible(true);
17) setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
18) add(l1);
19) add(t1);
20) add(l2);
21) add(t2);
22) t3.setEditable(false);
23) add(t3);
24) add(b);
25) b.addActionListener(this);
26) }
27) @Override
28) public void actionPerformed(ActionEvent e) {
29) int f = Integer.parseInt(t1.getText());
30) int s = Integer.parseInt(t2.getText());
31) int r = f * s;
32) t3.setText("" + r);
33) }
34) public static void main(String[] args) {
35) new MultiplyUsingSwing();
36) }
37) }
```

==========

Write a swing program to read two input user using input dialog and sum those values and display it using message dialog.

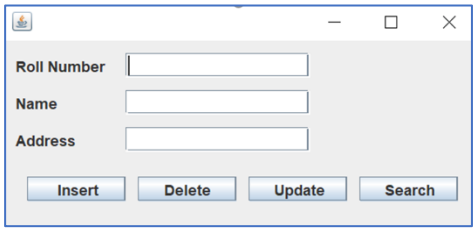
```
1)  /*
2)  To change this license header, choose License Headers
    in Project Properties.
3)  To change this template file, choose Tools | Templates
4)  and open the template in the editor.
5)  */
6)  package inputdialogexample;
7)  import javax.swing.*;
8)  /**
9)  *
10) @author DELL
11) */
12) public class InputDialogExample {

13) /**
14) @param args the command line arguments
15) */
16) public static void main(String[] args) {
17) int num,num1;
18) num=Integer.parseInt(JOptionPane.showInputDialog("
    Enter first values"));
19) num1=Integer.parseInt(JOptionPane.showInputDialog("
    Enter second values"));
20) JOptionPane.showMessageDialog(null,(num+num1));
21) }

22) }
```

==========

Write a servlet program to select students id name age from student table and display in html table.

```
1)  public void doGet(HttpServletRequest req,
    HttpServletResponse res)
2)  throws ServletException,IOException{
3)  res.setContentType("text/html");
4)  PrintWriter out=res.getWriter();
5)  //selecting data
6)  try {
7)  Connection conn=DbConnection.getConn();
8)  String sql="SELECT * FROM employees";
9)  PreparedStatement pst=conn.prepareStatement(sql);
10) ResultSet rs=pst.executeQuery();
11) out.println("<html><body>");
12) out.println("<a href='index.html'>
13) Goto Index </a> <br><br>");
14) //displaying data
15) out.println("<table>");
16) out.println("<tr>");
17) out.println("<th> Eid </th>");
18) out.println("<th> Name </th>");
19) out.println("<th> Address </th>");
20) out.println("</tr>");
21) while(rs.next()) {
22) out.println("<tr>");
23) out.println("<td>"+rs.getInt(1)+"</td>");
24) out.println("<td>"+rs.getString(2)+"</td>");
25) out.println("<td>"+rs.getString(3)+"</td>");
26) out.println("<td>");
27) out.println("</td>");
28) out.println("</tr>");
29) }
30) out.println("</table>
31) </body>
32) </html>")
```

==========




```
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.sql.*;
public class AllOp
{
AllOp() {
   JFrame f=new JFrame();
   JLabel lblRollno=new JLabel("Roll Number");
   lblRollno.setBounds(10, 10, 150, 20);
   f.add(lblRollno);
   JTextField txtRollno=new JTextField();
   txtRollno.setBounds(100, 10, 150, 20);
   f.add(txtRollno);
   JLabel lblName=new JLabel("Name");
   lblName.setBounds(10, 40, 150, 20);
   f.add(lblName);
   JTextField txtName=new JTextField();
   txtName.setBounds(100, 40, 150, 20);
   f.add(txtName);
   JLabel lblAddress=new JLabel("Address");
   lblAddress.setBounds(10, 70, 150, 20);
   f.add(lblAddress);
   JTextField txtAddress=new JTextField();
   txtAddress.setBounds(100, 70, 150, 20);
   f.add(txtAddress);
   JButton btnInsert=new JButton("Insert");
   btnInsert.setBounds(20, 110, 80, 20);
   f.add(btnInsert);
   JButton btnDelete=new JButton("Delete");
   btnDelete.setBounds(110, 110, 80, 20);
   f.add(btnDelete);
   JButton btnUpdate=new JButton("Update");
   btnUpdate.setBounds(200, 110, 80, 20);
   f.add(btnUpdate);
   JButton btnSearch=new JButton("Search");
   btnSearch.setBounds(290, 110, 80, 20);
   f.add(btnSearch);
   btnInsert.addActionListener(new ActionListener()
   {
      public void actionPerformed(ActionEvent ae)
      {
         try
         {
         Class.forName("com.mysql.cj.jdbc.Driver");
         Connection
conn=DriverManager.getConnection("jdbc:mysql://localhost:3307/tes
t","root","");
         String sql="insert into tblStd
(ID,RollNo,Name,Address) values(NULL,?,?,?)";
         PreparedStatement ptst=conn.prepareStatement(sql);
         ptst.setInt(1,
Integer.parseInt(txtRollno.getText()));
         ptst.setString(2, txtName.getText());
         ptst.setString(3, txtAddress.getText());
         ptst.executeUpdate();
         if(ptst.getUpdateCount()>0)
         JOptionPane.showMessageDialog(null,"Data inserted
Successfully");
         conn.close();
} }
);
}
catch(Exception e)
{
   JOptionPane.showMessageDialog(null, ae);
}
   btnDelete.addActionListener(new ActionListener()
   {
      public void actionPerformed(ActionEvent ae)
      {
         try
         {
         Class.forName("com.mysql.cj.jdbc.Driver");
         Connection
conn=DriverManager.getConnection("jdbc:mysql://localhost:3307/tes
t","root","");
         String sql="delete from tblstd where RollNo=?";
         PreparedStatement ptst=conn.prepareStatement(sql);
         ptst.setInt(1,
Integer.parseInt(txtRollno.getText()));
         ptst.executeUpdate();
         if(ptst.getUpdateCount()>0)
         JOptionPane.showMessageDialog(null,"Data deleted
Successfully");
Found");
} }
);
else
JOptionPane.showMessageDialog(null,"Data not
conn.close();
}
catch(Exception e)
{
   JOptionPane.showMessageDialog(null, ae);
}
   btnUpdate.addActionListener(new ActionListener()
   {
      public void actionPerformed(ActionEvent ae)
      {
         try
         {
         Class.forName("com.mysql.cj.jdbc.Driver");
         Connection
conn=DriverManager.getConnection("jdbc:mysql://localhost:3307/tes
t","root","");
         String sql="update tblstd set
RollNo=?,Name=?,Address=? where RollNo=?";
         PreparedStatement ptst=conn.prepareStatement(sql);
         ptst.setInt(1,
Integer.parseInt(txtRollno.getText()));
         ptst.setString(2, txtName.getText());
         ptst.setString(3, txtAddress.getText());
         ptst.setInt(4,
Integer.parseInt(txtRollno.getText()));
         ptst.executeUpdate();
         if(ptst.getUpdateCount()>0)
         JOptionPane.showMessageDialog(null,"Data Updated
Successfully");
found");
else
 JOptionPane.showMessageDialog(null,"Data not
conn.close();
}
catch(Exception e)
{
   JOptionPane.showMessageDialog(null, ae);
}
} }
);
   btnSearch.addActionListener(new ActionListener()
   {
      public void actionPerformed(ActionEvent ae)
      {
         try
         {
         Class.forName("com.mysql.cj.jdbc.Driver");
         Connection
conn=DriverManager.getConnection("jdbc:mysql://localhost:3307/tes
t","root","");
         String sql="select * from tblstd where RollNo=?";
         PreparedStatement ptst=conn.prepareStatement(sql);
         ptst.setInt(1,
Integer.parseInt(txtRollno.getText()));
         ResultSet rs=ptst.executeQuery();
         if(rs.next())
         {
         txtName.setText(rs.getString("Name"));
         txtAddress.setText(rs.getString("Address"));
         }
         else
         {
Found");
}
} }
);
JOptionPane.showMessageDialog(null,"Data not
conn.close();
}
catch(Exception e)
{
   JOptionPane.showMessageDialog(null, ae);
}
   f.setSize(400,500);
   f.setLayout(null);
   f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   f.setVisible(true);
 }
 public static void main(String args[])
 {
   new AllOp();
 }}
```

==========

How JSP is differing from servlet? Write JSP program to display " Tribhuvan University" 10 times.

Servlets are faster as compared to JSP, as they have a short response time. JSP is slower than Servlets, as the first step in the JSP lifecycle is the conversion of JSP to Java code and then the compilation of the code. Servlets are Java-based codes. JSP are HTML-based codes. In Servlet we have to implement everything like business logic and presentation logic in just one servlet file.In JSP business logic is separated from presentation logic by using JavaBeansclient-side.

Program:

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/htm; charset=UTF-8">
    <title> JSP program to display "Tribhuvan University" 10 times</title>
  </head>
  <body>
        <h1> Displaying "Tribhuvan University" 10 times!!</h1>
        <table>
          <%
          for(int i=1; i<=10; i++){
          %>
          <tr><td> Tribhuvan University</td></tr>
          <% } %>
        </table>
  </body>
  </html>
```

==========

How to develop MDI application on java explain?

MDI stands for Multiple Document Interface. In an MDI application, one main window is opened, and multiple child windows are open within the main window.

In an MDI application, we can open multiple frames that will be instances of the JInternalFrame class.

We can organize multiple internal frames in many ways. For example, we can maximize and minimize them; we can view them side by side in a tiled fashion, or we can view them in a cascaded form.

The following are four classes we will be working with in an MDI application:

- JInternalFrame
- JDesktopPane
- DesktopManager
- JFrame

An instance of the JInternalFrame class acts as a child window that is displayed inside the area of its parent window.

We can add Swing components to JInternalFrame content pane, pack them using the pack() method, and make it visible using the setVisible(true) method.

To listen to window events such as activated, deactivated, etc., we need to add an InternalFrameListener to the JInternalFrame instead of a WindowListener, which is used for a JFrame.

The following code shows how to use an instance of the JInternalFrame class:

```
String title = "A Child  Window";
Boolean  resizable = true;
Boolean  closable = true;
Boolean  maximizable = true;
Boolean  iconifiable = true;
JInternalFrame iFrame = new JInternalFrame(title, resizable, closable, maximizable, iconifiable);
Add components to the iFrame using
iFrame.add(...)
Pack eth frame and make it visible
iFrame.pack();
iFrame.setVisible(true);
```

JDesktopPane is used as a container for all JInternalFrame. It uses a null layout manager.

```
JDesktopPane  desktopPane = new JDesktopPane();
// Add all  JInternalFrames to the  desktopPane
desktopPane.add(iFrame);
```

We can get all JInternalFrames that are added to a JDesktopPane using its getAllFrames() method.

```
JInternalFrame[] frames  = desktopPane.getAllFrames();
```

A JDesktopPane uses an instance of the DesktopManager interface to manage all internal frames.

The DefaultDesktopManager implements DesktopManager interface.

The desktop manager has many useful methods. For example, to close an internal frame programmatically, use its closeFrame() method.

```
desktopPane.getDesktopManager().closeFrame(frame1);
```

The following code demonstrates how to develop an MDI application.

```
1)  import java.awt.BorderLayout;
2)  import java.awt.Dimension;
3)  /*from  w w  w. j av a2 s. co  m*/
4)  import javax.swing.JDesktopPane;
5)  import javax.swing.JFrame;
6)  import javax.swing.JInternalFrame;
7)  import javax.swing.JLabel;
8)  import javax.swing.SwingUtilities;
9)  import javax.swing.UIManager;
10) public class Main extends JFrame {
11) private final JDesktopPane desktopPane = new
        JDesktopPane();
12) public Main() {
13) JInternalFrame frame1 = new JInternalFrame("Frame 1",
        true, true, true,
14) true);
15) JInternalFrame frame2 = new JInternalFrame("Frame 2",
        true, true, true,
16) true);
17) frame1.getContentPane().add(new JLabel("Frame 1
        contents..."));
18) frame1.pack();
19) frame1.setVisible(true);
20) frame2.getContentPane().add(new JLabel("Frame 2
        contents..."));
21) frame2.pack();
22) frame2.setVisible(true);
23) int x2 = frame1.getX() + frame1.getWidth() + 10;
24) int y2 = frame1.getY();
25) frame2.setLocation(x2, y2);
26) desktopPane.add(frame1);
27) desktopPane.add(frame2);
28) this.add(desktopPane, BorderLayout.CENTER);
29) this.setMinimumSize(new Dimension(300, 300));
30) }
31) public static void main(String[] args) {
32) try {
33) UIManager.setLookAndFeel(UIManager.getSystemLook
        AndFeelClassName());
34) } catch (Exception e) {
35) e.printStackTrace();
36) }
37) SwingUtilities.invokeLater(() -> {
38) Main frame = new Main();
39) frame.pack();
40) frame.setVisible(true);
41) frame.setExtendedState(frame.MAXIMIZED_BOTH);
42) });
43) }
44) }
```

==========

What are JTextField and JTextArea in Java?

JTextField

A JTextFeld is one of the most important components that allow the user to an input text value in a single line format. A JTextField will generate an ActionListener interface when we trying to enter some input inside it. The important methods in the JTextField class are setText(), getText(), setEnabled(), etc.

Example

```
1)  import javax.swing.*;
2)  import java.awt.*;
3)  public class JTextFieldTest {
4)  public static void main(String[] args) {
5)  final JFrame frame = new JFrame("JTextField Demo");
6)  JLabel lblFirstName = new JLabel("First Name:");
7)  JTextField tfFirstName = new JTextField(20);
8)  lblFirstName.setLabelFor(tfFirstName);
9)  JLabel lblLastName = new JLabel("Last Name:");
10) JTextField tfLastName = new JTextField(20);
11) lblLastName.setLabelFor(tfLastName);
12) JPanel panel = new JPanel();
13) panel.setLayout(new FlowLayout());
14) panel.add(lblFirstName);
15) panel.add(tfFirstName);
16) panel.add(lblLastName);
17) panel.add(tfLastName);
18) frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19) frame.setSize(300, 100);
20) frame.getContentPane().add(panel, BorderLayout.CENTER);
21) frame.setVisible(true);
22) }
23) }
```

JTextArea

A JTextArea is a multi-line text component to display text or allow the user to enter text. A JTextArea will generate a CaretListener interface.

The important methods in the JTextArea class are setText(), append(), setLineWrap(), setWrapStyleWord(), setCaretPosition(), etc.

Example

```
1)  import java.awt.*;
2)  import javax.swing.*;
3)  import javax.swing.event.*;
4)  public class JTextAreaTest {
5)  public static void main(String args[]) {
6)  JFrame frame = new JFrame("JTextArea Example");
7)  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
8)  JTextArea textArea = new JTextArea();
9)  JScrollPane scrollPane = new JScrollPane(textArea);
10) frame.add(scrollPane, BorderLayout.CENTER);
11) CaretListener listener = new CaretListener() {
12) public void caretUpdate(CaretEvent caretEvent) {
13) System.out.println("Dot: "+ caretEvent.getDot());
14) System.out.println("Mark: "+caretEvent.getMark());
15) }
16) };
17) textArea.addCaretListener(listener);
18) frame.setSize(250, 150);
19) frame.setVisible(true);
20) }
21) }
```

==========

Difference between session and cookie. WAP ti implement session.

Session	Cookie
Server side files that contain user data	Client side files on a local computer that hold user information
It can hold an indefinite quantity of data	It can only store a certain amount of info.
They are more secured.	They are not secured.
Store data in text file.	Save data in encrypted form.
They have maximum capacity of 4KB	They have maximun memory of 128KB

```
1)  import java.io.*;
2)  import java.util.*;
3)  import javax.servlet.*;
4)  import javax.servlet.http.*;
5)  @WebServlet("/test_session")
6)  public class TestSessionServlet extends HttpServlet {
7)  private static final long serialVersionUID = 1L;
8)  public TestSessionServlet() {
9)  super();
10) }
11) protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
12) throws ServletException, IOException {
13) HttpSession session = request.getSession();
14) PrintWriter writer = response.getWriter();
15) writer.println("Session ID: " + session.getId());
16) writer.println("Creation Time: " + new
        Date(session.getCreationTime()));
17) writer. println("Last Accessed Time: " + new
        Date(session.getLastAccessedTime()));
18) }
19) }
```

==========

WAP to calculate sum of any two number using rmi.

```
1)  Adder.interface
2)  import java.rmi.Remote;
3)  import java.rmi.RemoteException;
4)  public interface Adder extends Remote{
5)  void Add(Integer a,Integer b) throws RemoteException;
6)  }
7)  ImplExample.java
8)  public class ImplExample implements Adder{
9)  ImplExample()
10) {
11) super();
12) }
13) public void Add(Integer x,Integer y) {
14) System.out.println("sum = "+(x+y));
15) }
16) }
17) ServerRMI.java
18) import java.rmi.registry.Registry;
19) import java.rmi.registry.LocateRegistry;
20) import java.rmi.RemoteException;
21) import java.rmi.server.UnicastRemoteObject;
22) public class ServerRMI extends ImplExample{
                public ServerRMI() {}
                public static void main(String args[]) {
                        try {
23) ImplExample obj = new ImplExample();
24) Hello skeleton =(Hello)
    UnicastRemoteObject.exportObject(obj, 0);
25) Registry registry = LocateRegistry.getRegistry();
26) registry.bind("RMITest",skeleton);
27) System.err.println("server Ready");
28) }
29) catch(Exception e) {
30) System.err.println("Server Exception:"+ e.toString());
31) e.printStackTrace();
32) }
33) }
34) }
35) ClientRMI.java
36) import java.rmi.registry.LocateRegistry;
37) import java.rmi.registry.Registry;
38) public class ClientRMI{
39) private ClientRMI() {}
40) public static void main(String[] args) {
41) try {
42) Registry registry = LocateRegistry.getRegistry();
43) Hello stub = (Hello) registry.lookup("RMITest");
44) stub.Add(5,10);
45) }
46) catch(Exception e) {
47) System.err.println("Client Exception:"+e.toString());
48) }
49) }
50) }
```

==========

Write a swing program to show color panel using dialog box to get color.

```
1. import java.awt.Color;
2. import java.awt.event.
3. import javax.swing.*;
4. public class ShowColor implements ActionListener
5. Button btn;
6. Color color;
7. Panel panel;
8. Frame frame;
9. public void showColor() {
10. panel=new JPanel () ;
11. panel.setBackground (color) ;
12. btn=new JButton ("Change color') ;
13. btn.addActionlistener (this) ;
14. panel.add (btn);
15. Frame = New JFrame("sample frame");
16. frame.getContentPane () .add (panel) ;
17. frame.setSize (500, 500) ;
18. frame.setLocationRelativeTo(null);
19. frame.setVisible (true) ;
20. frame.setDefaultCloseOperation (JFrame.EXIT ON CLOSE);
21. }
23. Public void actionPerformed(ActionEvent e)
25. Color = JColorChooser.showDialog(frame,"choose a color
", color );
26. If (color =null). color=color.LIGTH_GRAY;
28. panel.setBackground(color);
29. }
30. public static void main (String args[]){
31. ShowColor show=new ShowColor();
32. show. showColor();
33. }
34. }
```

==========

Write a program to show font.

```
1)  import java.awt.Color;
2)  import java.awt.Font;
3)  import java.awt.Graphics;
4)  import javax.swing.JPanel;
5)  public Class Fonts {
6)  public static void main (String args[])
7)  {
8)  JFrame frame=new Frame ("Font styles..");
9)  frame.add (new FontExample());
10) frame.setSize(480,150);
11) frame.setlocationRelativeTo(null) ;
12) frame.setDefaultCloseOperation(JFrame.EXIT ON
    CLOSE) ;
13) frame. getVisible(true);
14) }
15) }
16) public class FontExample extends JPanel{
17) public void paint (Graphics g) {
18) g.setFont (new Font ("Arial", Font.BOLD, 12)) ;
19) g.drawString ("This is a sample text", 20, 30) ;
20) g. setFont (new Font ("Serif",Font.ITALIC, 18)) ;
21) g. drawString ("This is another sample text", 20, 60);
22) g. setColor (Color.RED)
23) g.setFont (new Font ("Monospaced", Font.BOLD, 20) )
    ;
24) g. drawString ("This is again, another sample text",
    20, 90) :
25) }
26) }
```

==========

Write program to create Different shapes.

```
1)  import java.awt.Color;
2)  import java.awt.Graphics;
3)  import javax.swing.JPanel;
4)  public class DrawShapes
5)  {
6)  public static void main (String args[])
7)  {
8)  Frame frame=new Frame ("Sample shapes.");
9)  frame.add(new Draw());
10) frame.setSize (400, 200) .
11) frame. setLocationRelativeTo (null) ;
12) frame.setDefaultCloseOperation (JFrame.EXIT ON
    CLOSE) ;
13) frame.setvisible(true)
14) }
    15)     }
16) public class Draw extends JPanel{
17) public void paint (Graphics g) {
18) g. setColor (Color.RED) ;
19) g. drawLine(5,30,380,30);
20) g. setColor (Color. BLUE) ;
21) g. drawRect (5,40, 90,55) :
22) g. fillRect (100, 40, 90,55) ;
23) g.setColor (Color. GREEN)
24) g. drawOval (195, 40, 90,
25) g. filloval (290, 40, 90, 55) :
26) g. setColor (Color. CYAN) ;
27) g. fillOval (5, 100, 90, 90) ;
28) }
29) }
```

==========

Write a GUI program to create table , insert and show product information which include id, name and price.

```
o   import java.sql.*;
o   public class CreateProductTable{
o   public static void main (String [] args){
o   try{
o   Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriv
    er");
o   String DB URL="jdbc: sqlserver://<usernames> \\
    SQLEXPRESS2:1433;databaseName=BCA
    ;integratedSecurity=true;";
o   Connection con = DriverManager. getConnection( DB URL) ;
o   System. out.println ("Database connected...");
o   Statement statement = con.createStatement () ;
o   String sql="CREATE TABLE PRODUCT (ID VARCHAR (10) ,
    NAME VARCHAR (30) , PRICE FLOAT, PRIMARY KEY (ID)) ;";
o   statement.executeUpdate ( sql);
o   }
o   catch ( Exception e)
o   {
o   e.printstackTrace();
o   }}}
o   public class InsertProduct{
o   public static void main (String[ ] args)(
o   try{
o   String DB URL="jdbc:sqlserver: / /<username>\
    \(SQLEXPRESS 2: 1433; databaseName-BCA;
    integratedSecurity=true;";
o   Connection con = DriverManager. getConnection( DB_URL) ;
o   System. out.println ("Database connected") ;
o   Statement statement = con.createStatement();
o   String sql="INSERT INTO PRODUCT VALUES ('A102', 'pant',
    950) ; ";
o   statement. executeUpdate ( sql):
o   Catch (Exception e)
o   {
o   e.printStackTrace();
o   }
o   }
o   }
o   public class ViewProduct{
o   public static void main (String[]args){
o   try{
o   Class.
    forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
o   String DB URL="idbc: sqlserver://<username>\\
    SQLEXPRESS2:1433;
    databaseName=BCA;integratedSecurity-true;";
o   Connection con = DriverManager. getConnection( DB_URL) ;
o   System. out.println ("Database connected...") ;
o   Statement statement= con. createStatement () ;
o   String sql="SELECT * FROM PRODUCI;";
o   ResultSet rs=statement.executeQuery (sql) ;
o   String id,name,price;
o   while(rs.next())
o   {
o   id= rs.getString("ID");
o   name=rs.getString("Name");
o   Price = rs.getString("PRICE");
o   System.out.println(id+"\t"+name+"\"+price);
o   }
o   }catch (Exception e )
o   {
o   e.printStackTrack();
o   }
o   }
o   }
```

==========

Write a program using jdbc to enter and display data to the database.

```
1)  Import javax. swing.*;
2)  Import java.awt.event.*;
3)  public class InformationForm implements ActionListener{
4)  JLabel lblname, lbladdress, lbllevel, lname, laddress, llevel;
5)  JTextfield txtname, txtaddress,txtlevel;
6)  JButton btnregister;
7)  JPanel panel;
8)  Jfarme frame;
9)  void createForm() {
10) lblname= new JLabel("Name");
11) Lbladress= new JLabel("address:");
12) Lbllevel = new JLabel("Level:");
13) txtname = new JTextField();
14) txtadress = new JtextField(10);
15) txtlevel =new JTextField(10);
16) btnregister = newButton ( "register");
17) btnregister.addActionlistener(this);
18) lblname.setBounds(10,20,80,25);
19) lbladdress.setBounds(10,60,80,25);
20) lbllevel.setBounds(10,100,80,25) ;
21) txtname. setBounds(100, 20,160,25);
22) txtaddress.setBounds(100,100,160,25);
23) txtlevel.setBounds(100,100,160,25);
24) btnregister.setBounds(100,150,80,25);
25) JPanel panel=new Jpanel();
26) panel.setlayout(null);
27) panel.ad(lblname);
28) panel.add (lbladdress);
29) panel.add (lbllevel);
30) panel.add (txtname);
31) panel.add (txtaddress);
32) panel.add (txtlevel);
33) panel.add (btnregister);
34) JFrame= new uFrame ("Data entry Form frame....");
35) frame.add (panel);
36) frame. setSize(300,240);
37) frame.setlocationRelativeTo(null):
38) san..getDefaultCloseOperation (Frame. EXIT_ON_CLOSE);
39) }
40) public void actionPerformed(ActionEvent e){
41) if(e.getsource).equals(btnregister)){
42) JOptionPane.showMessageDialog(frame,"you have
    registered your information");
43) String tname=txtname.getText();
44) taddress=txtaddress.getText()
45) String taddress=txtaddress.getText():
46) String tlevel=txtlevel.getText():
47) createNewForm (tame, taddress,tlevel):
48) }
49) void createNewForm(String tname,String address, String
    tlevel){
50) lblname = new JLabel ("Name:");
51) lbladdress = new JLabel("Address:");
52) lbllevel = new JLabel("Level:);
53) lname= new JLabel();
54) laddress=newJLabel();
55) llevel =newJLabel();
56) lname.setText (tname) :
57) laddress.setText(taddress):
58) llevel.setText(tlevel);
59) lblname.setBounds (10, 20, 80, 25) ;
60) lbladdress.setBounds (10, 60,80,25);
61) lbllevel.setBounds (10, 100,80,25);
62) lblname.setBounds(100,20,160,25);
63) lbladdress.setBounds (10, 60,80,25);
64) level.setBounds(100,100,160,25);
65) JPanel panel =new JPanel();
66) panel.setLayout(null);
67) panel.setLayout(null);
68) panel.add(lblname);
69) panel.add(lbladdress);
70) panel.add(lbllevel);
71) panel.add(lname);
72) panel.add(laddress);
73) panel.add(llevel);
74) JFrame frame = new JFrame ("Data entry Form...");
75) frame.add (panel);
76) frame.setSize(300,240);
77) frame.setlocationRelativeTo (null) ;
78) frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
79) frame.setVisible (true) ;
80) }
81) Public static void main (Sting args[]){
82) InformationformForm info=new InformationForm() ;
83) info.createForm();
84) }
85) }
```

==========

Write servlet program to add two number.

Index.html

```
<!DOCTYPE html>
<html>
<head>
<title>Practical Work</title>
</head>
<body>
<form action="add_me">
<label>First number </label> <input type="text" name="num1"/> <br/><br/>
<label>Second number </label> <input type="text" name="num2"/>
<br/><br/>
<button type="submit" name="calculate">Product and Sum </button><br/>
</form>
</body>
</html>
```

Add_Numbers.java

```
package com.programmerbay;
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.http.*;
public class Add_Numbers extends HttpServlet{
public void service(HttpServletRequest request,HttpServletResponse
response) throws IOException
{
int num1 = Integer.parseInt(request.getParameter("num1"));
int num2 = Integer.parseInt(request.getParameter("num2"));
int sum = num1 + num2;
int product = num1 * num2;
PrintWriter output = response.getWriter();
output.println("The Answer :"+sum +"\n The product :"+product);
}
}
```

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID"
version="3.1">

<servlet>
<servlet-name>Add</servlet-name>
<servlet-class>com.programmerbay.Add_Numbers</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>Add</servlet-name>
<url-pattern>/add_me</url-pattern>
</servlet-mapping>
</web-app>
```

==========