CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | | |

Internal Assesment Test - II

Sub:	JAVA and J2EE							Code:	**10CS753**
Date:	04 / 11 / 2016	Duration:	90 mins	Max Marks:	50	Sem:	VII	Branch:	CSE/ISE

Answer Any **FIVE FULL** Questions

		Marks	OBE	
			CO	**RBT**
1 (a)	What is Multithreading? How Synchronization is implemented in Java, explain with an example.	[10]	**CO5**	**L1**
2 (a)	List out and briefly explain the different types of database drivers.	[5]	**CO6**	**L1**
(b)	Discuss thread priority with an example.	[5]	**CO5**	**L2**
3 (a)	Write a Java program to establish connection to a database and read the contents of the table.	[10]	**CO6**	**L2**
4 (a)	Explain the lifecycle of the Servlet with code snippets for init(), service() and destroy() methods.	[10]	**CO6**	**L1**
5 (a)	Discuss with an example how session tracking is handled in Java with Servlets.	[10]	**CO6**	**L2**
6 (a)	Explain the mechanism of Event Delegation Model. With a program example briefly explain handling of mouse events.	[10]	**CO6**	**L1**
7 (a)	Briefly explain the following i) Event classes ii) Event listener interface iii) Adapter class iv) Event sources	[10]	CO6	L1

Course Outcomes		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1:	Explain of the principles of OOP - Abstraction, Polymorphism, Inheritance, Encapsulation.	2	1	1	1	1	-	-	-	1	1	-	-
CO2:	Apply object oriented concepts to design simple Java Programs	2	2	1	1	1	-	-	-	1	1	-	-
CO3:	Use Exception Handling in Java.	2	2	1	1	1	-	-	-	1	-	-	1
CO4:	Implement User interface using Swing and Applets	2	2	1	1	1	-	-	-	1	-	-	-
CO5:	Explain the concepts of Multithreaded Programming in Java	2	2	1	1	1	-	-	-	1	-	-	1
CO6:	Implement JSP Script communicate with different databases using JDBC driver and remote servers.	2	2	1	1	2	-	-	-	1	-	-	1

Cognitive level	KEYWORDS
L1	List, define, tell, describe, identify, show, label, collect, examine, tabulate, quote, name, who, when, where, etc.
L2	summarize, describe, interpret, contrast, predict, associate, distinguish, estimate, differentiate, discuss, extend
L3	Apply, demonstrate, calculate, complete, illustrate, show, solve, examine, modify, relate, change, classify, experiment, discover.
L4	Analyze, separate, order, explain, connect, classify, arrange, divide, compare, select, explain, infer.
L5	Assess, decide, rank, grade, test, measure, recommend, convince, select, judge, explain, discriminate, support, conclude, compare, summarize.

PO1 - *Engineering knowledge*; PO2 - *Problem analysis*; PO3 - *Design/development of solutions*; PO4 - *Conduct investigations of complex problems*; PO5 - *Modern tool usage*; PO6 - *The Engineer and society*; PO7- *Environment and sustainability*; PO8 – *Ethics*; PO9 - *Individual and team work*; PO10 - *Communication*; PO11 - *Project management and finance*; PO12 - *Life-long learning*

CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | | |

Internal Assesment Test - II

Sub:	JAVA and J2EE							Code:	**10CS753**
Date:	04 / 11 / 2016	Duration:	90 mins	Max Marks:	50	Sem:	VII	Branch:	CSE/ISE

Answer Any **FIVE FULL** Questions

		Marks	OBE	
			CO	**RBT**
1(a)	What is Multithreading? How Synchronization is implemented in Java, explain with an example.	[10]	**CO5**	**L1**

Soln. Multithreading is a conceptual programming concept where a program (process) is divided into two or more subprograms (process), which can be implemented at the same time in parallel. A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution. A ***process*** consists of the memory space allocated by the operating system that can contain one or more threads. A thread cannot exist on its own; it must be a part of a process.

When two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time. The process by which this synchronization is achieved is called thread synchronization. The synchronized keyword in Java creates a block of code referred to as a critical section. Every Java object with a critical section of code gets a lock associated with the object. To enter a critical section, a thread needs to obtain the corresponding object's lock.

Example without Synchronization

```
class update
{
    void updatesum(int i)
    {
        Thread t = Thread.currentThread();
        for(int n=1; n<=5; n++)
        {
            System.out.println(t.getName()+" : "+(i+n));
        }
    }
}
class A extends Thread
{
    update u = new update();
    public void run()
    {
        u.updatesum(10);
    }
}
class syntest
{
    public static void main(String args[])
    {
        A a = new A();
        Thread t1 = new Thread(a);
        Thread t2 = new Thread(a);
        t1.setName("Thread A");
        t2.setName("Thread B");
        t1.start();
        t2.start();
    }
}
```

Example withSynchronization

```
class update
{
    synchronized void updatesum(int i)
    {
        Thread t = Thread.currentThread();
        for(int n=1; n<=5; n++)
        {
            System.out.println(t.getName()+" : "+(i+n)
        }
    }
}
class A extends Thread
{
    update u = new update();
    public void run()
    {
        u.updatesum(10);
    }
}
class syntest
{
    public static void main(String args[])
    {
        A a = new A();
        Thread t1 = new Thread(a);
        Thread t2 = new Thread(a);
        t1.setName("Thread A");
        t2.setName("Thread B");
        t1.start();
        t2.start();
    }
}
```

2 (a)	List out and briefly explain the different types of database drivers.	[5]	CO6	L1

Soln.

- JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates. Sun has divided the implementation types into four categories, Types 1, 2, 3, and 4.

Type 1: JDBC-ODBC Bridge Driver

- In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine. Using ODBC, requires configuring on your system a Data Source Name (DSN) that represents the target database.
- When Java first came out, this was a useful driver because most databases only supported ODBC access but now this type of driver is

recommended only for experimental use or when no other alternative is available.

Type 2: JDBC-Native API

- In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls, which are unique to the database. These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge. The vendor-specific driver must be installed on each client machine.

- If we change the Database, we have to change the native API, as it is specific to a database and they are mostly obsolete now, but you may realize some speed increase with a Type 2 driver, because it eliminates ODBC's overhead.

Type 3: JDBC-Net pure Java

- In a Type 3 driver, a three-tier approach is used to access databases. The JDBC clients use standard network sockets to communicate with a middleware application server. The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.

- This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.

Type 4: 100% Pure Java

- In a Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself.

- This kind of driver is extremely flexible, you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.

(b) Discuss thread priority with an example. [5] | **CO5** | **L2**

Soln. Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled. Thread priorities are integers that specify the relative priority of one thread to another. As an absolute value, a priority is meaningless; a higher-priority thread doesn't run any faster than a lower-priority thread if it is the only thread running. Instead, a thread's priority is used to decide when to switch from one running thread to the next. This is called a context switch.

- A thread can voluntarily relinquish control. This is done by explicitly yielding, sleeping, or blocking on pending I/O. In this scenario, all other threads are examined, and thehighest-priority thread that is ready to run is given the CPU.
- A thread can be preempted by a higher-priority thread. In this case, a lower-priority thread that does not yield the processor is simply preempted—no matter what it is doing— by a higher-priority thread. Basically, as soon as a higher-priority thread wantsto run, it does. This is called preemptive multitasking.
- Java priorities are in the range between **MIN_PRIORITY (a constant of 1)** and **MAX_PRIORITY (a constant of 10)**. By default, every thread is given priority **NORM_PRIORITY (a constant of 5)**.
- Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads. However, thread priorities cannot guarantee the order in which threads execute and very much platform dependent.

```
class prioritytest
{
    public static void main(String args[])
    {
        C c = new C();
        A a = new A();
        a.setPriority(10);
        c.setPriority(1);
        c.start();
        a.start();
    }
}
```

3(a)	Write a Java program to establish connection to a database and read the contents of the table.	[10]	**CO6**	**L2**

Soln.

5 Steps to connect to the database in java

There are 5 steps to connect any java application with the database in java using JDBC. They are as follows:

1. **Register the driver class**
   - The **forName()** method of "**Class**" class is used to dynamically load the driver class.
   - **public static void** forName(String className)**throws** ClassNotFoundException
     To register the OracleDriver class
   - Class.forName("oracle.jdbc.driver.OracleDriver");
2. **Creating connection**
   - The **getConnection()** method of **DriverManager class** is used to establish connection with the database.
   - **public static** Connection getConnection(String url)**throws** SQLException
   - **public static** Connection getConnection(String url,String name,String password) **throws** SQLException
   - **Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","pass");**
   - jdbc is the API, oracle is the database, thin is the driver, localhost is the server name on which oracle is running (IP address), 1521 is the port number and XE is the Oracle service name

3. Creating statement

•The **createStatement()** method of **Connection interface** is used to create statement. The object of statement is responsible to execute queries with the database.

•**public** Statement createStatement()**throws** SQLException

```
Statement stmt=con.createStatement();
```

4. Executing queries

•The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

•**public** ResultSet executeQuery(String sql)**throws** SQLException

```
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
{
        System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

• Closing connection

○ By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

○ **public void** close()**throws** SQLException

○ con.close();

○ rs.close();

○ stmt.close();

4 (a)	Explain the lifecycle of the Servlet with code snippets for init(), service() and destroy() methods.	[10]	**CO6**	**L1**

Soln. Each servlet has the same life cycle:

- A server loads and initializes the servlet [`init`() method]
- The servlet handles zero or more client requests [`service`() method]
- The server removes the servlet (some servers do this step only when they shut down)[`destroy()` method]

Step 1: A user enters a URL to a browser. The browser generates an HTTP request for this URL and this request is sent to the appropriate server.

Step 2: The HTTP request is received by the web server. The server maps this request to aparticular servlet. This servlet is dynamically retrieved and loaded into the server.

Step 3: The server invokes the `init`() method of the servlet. This method is invoked only when the servlet is first loaded into the memory. We can pass

initialization parameters to the servlet.

Step 4: The server invokes the `service()` method of the servlet. This method is called toprocess the request HTTP request. The servlet can read data that has been provided in the HTTPrequest. The service method can also create a HTTP response for the client. The servlet remains in the server's address space and is available to process any other requests from other clients.The service method is called for each request.

Step 5: The server calls the `destroy()` method when a servlet has to be unloaded from the server memory. Once this method is called, the servlet will give up all file handles that wereallotted to it. Important data may be saved to a persistent store. The memory allocated tothe servlet and its objects is released.

```
import java.io.*;
import javax.servlet.*;
public class HelloServlet extends GenericServlet
{
        public void init()
        {
                System.out.println("Servlet Initiated");
        }

        public void service(ServletRequest request, ServletResponse response)
        throws ServletException, IOException
        {
                response.setContentType("text/html");
                PrintWriter pw = response.getWriter();
                pw.println("<B>Hello!");
                pw.close();
        }

        public void destroy()
        {
                System.out.println("Servlet destroyed");
        }

}
```

5 (a) Discuss with an example how session tracking is handled in Java with Servlets. [10] | **CO6** | **L2**

1. HTTP is a stateless protocol. Each request is independent of the previous one. But in some applications such as online shopping, banking, etc, it is necessary to save the state information so that the information can be collected from the user over several interactions. Sessions provide this mechanism.
2. A session can be created by the getSession() method of HttpServletRequest. This method returns an HttpSession object. The setAttribute(), getAttribute(),

removeAttribute() and getAttributeNames() methods of the HttpSession manage the bindings between the names and objects.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http;
public class DateServlet extends HttpServlet
{
	public void doGet(HttpServletreq, HttpResponse res) throws
ServletException, IOException
	{
		//get the http response object
		HttpSessionhs = req.getSession(true);
		//get writer
		res.setContentType("text/html");
		PrintWriter out = res.getWriter();
		//Display date and time of last access
		Date dt = (Date) hs.getAttribute("date");

	if(dt != null)
	{
		out.println("Last access was on " + dt);
	}

	//display current date dt =
	new Date();
	hs.setAttribute("date", dt);
	out.println("Current date is : " + dt);
	}
}
```

Cookies:

1. Cookies are small files which are stored on a user's computer by the server.
2. They can hold small amounts of data for a specific client and website.
3. Cookies can be accessed either by the web server or the client computer. The server can send a page custom-made for a particular client, or location, or time of day. Thus, we can say that cookies are used for session management.
4. A cookie can be read back by the server. Thus the server can "remember" the client. This is important because HTTP itself is a stateless protocol. Once the data is delivered by the server to the client browser, the server will not keep any further information about the client.
5. Cookies have a name and a single value. They may have optional attributes such as version number, expiry date, a comment for the user, etc.
6. Cookies are assigned by the server to the client. They is sent using fieldsadded to the HTTP response header. Cookies are passed back to the server using fields added to the HTTP request headers.

```
import java.io.*;
```

```
import javax.servlet.*;
import javax.servlet.http.*;


public class FirstServlet extends HttpServlet {

 public void doPost(HttpServletRequest request, HttpServletResponse response){
        try{

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String n=request.getParameter("userName");
        out.print("Welcome "+n);

        Cookie ck=new Cookie("uname",n);//creating cookie object
        response.addCookie(ck);//adding cookie in the response

        //creating submit button
        out.print("<form action='servlet2'>");
        out.print("<input type='submit' value='go'>");
        out.print("</form>");

        out.close();

}catch(Exception e){System.out.println(e);}
  }
}
```

6 (a)	Explain the mechanism of Event Delegation Model. With a program example briefly explain handling of mouse events.	[10]	**CO6**	**L1**

Soln. Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism have the code which is known as event handler that is executed when an event occurs. Java Uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events. The Delegation Event Model has the following key participants namely:

1. **Source** - The source is an object on which event occurs. Source is responsible for providing information of the occurred event to its handler. Java provide as with classes for source object.
2. **Listener** - It is also known as event handler. Listener is responsible for generating response to an event. From java implementation point of view the listener is also an object. Listener waits until it receives an event. Once the event is received, the listener process the event then returns.

The benefit of this approach is that the user interface logic is completely separated from the logic that generates the event. The user interface element is able to delegate the processing of an event to the separate piece of code. In this model, Listener needs to be registered with the source object so that the listener can receive the event notification. This is an efficient way of handling the event because the event notifications are sent only to those listeners that want to receive them.

Mouse Events Program

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

/*
<applet code="MouseEvents" width=300 height=100>
</applet>
*/

public class MouseEvents extends Applet
implements MouseListener, MouseMotionListener
{
	String msg = "";
	intmouseX = 0, mouseY = 0; // coordinates of mouse

	public void init()
	{
		addMouseListener(this);
		addMouseMotionListener(this);
	}
// Handle mouse clicked.
	public void mouseClicked(MouseEvent me)
	{
		// save coordinates
		mouseX = 0;
		mouseY = 10;
		msg = "Mouse clicked.";
		repaint();
```

```
}

// Handle mouse entered.
public void mouseEntered(MouseEvent me)
{
        // save coordinates
        mouseX = 0;
        mouseY = 10;
        msg = "Mouse entered.";
        repaint();
}

// Handle mouse exited.
public void mouseExited(MouseEvent me)
{
        // save coordinates
        mouseX = 0;
        mouseY = 10;
        msg = "Mouse exited.";
        repaint();
}

// Handle button pressed.
public void mousePressed(MouseEvent me)
{
        // save coordinates
        mouseX = me.getX();
        mouseY = me.getY();
        msg = "Down";
        repaint();
}

// Handle button released.
public void mouseReleased(MouseEvent me)
{
        // save coordinates
        mouseX = me.getX();
        mouseY = me.getY();
```

```
            msg = "Up";
            repaint();
        }

        // Handle mouse dragged.
        public void mouseDragged(MouseEvent me)
        {
            // save coordinates
            mouseX = me.getX();
            mouseY = me.getY();
            msg = "*";
            showStatus("Dragging mouse at " + mouseX + ", " + mouseY);
            repaint();
        }

        // Handle mouse moved.
        public void mouseMoved(MouseEvent me)
        {
            // show status
            showStatus("Moving mouse at " + me.getX() + ", " + me.getY());
        }

        // Display msg in applet window at current X,Y location.
        public void paint(Graphics g)
        {
            g.drawString(msg, mouseX, mouseY);
        }
    }
```

7 (a)	Briefly explain the following	[10]	CO6	L1

Soln.

i) Event classes

- The class **AWTEvent**, defined within the **java.awt**package, is a subclass of **EventObject**. It is the superclass (either directly or indirectly) of all AWT-based events used by the delegation event

model. Its **getID( )**method can be used to determine the type of the event.

- **The ActionEvent Class**
  An **ActionEvent**is generated when a button is pressed, a list item is double-clicked, or a menu item is selected. The **ActionEvent**class defines four integer constants that can be used to identify any modifiers associated with an action event: **ALT_MASK**, **CTRL_MASK**, **META_MASK**, and **SHIFT_MASK**. In addition, there is an integer constant, **ACTION_PERFORMED**, which can be used to identify action events.

Event Class	Description
ActionEvent	Generated when a button is pressed, a list item is double-clicked, or item is selected.
AdjustmentEvent	Generated when a scroll bar is manipulated.
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes
ContainerEvent	Generated when a component is added to or removed from a conta
FocusEvent	Generated when a component gains or loses keyboard focus.
InputEvent	Abstract superclass for all component input event classes.
ItemEvent	Generated when a check box or list item is clicked; also occurs when a selection is made or a checkable menu item is selected or deselec
KeyEvent	Generated when input is received from the keyboard.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or re also generated when the mouse enters or exits a component.
MouseWheelEvent	Generated when the mouse wheel is moved.
TextEvent	Generated when the value of a text area or text field is changed.
WindowEvent	Generated when a window is activated, closed, deactivated, deicon iconified, opened, or quit.

- **The AdjustmentEvent Class**
  An **AdjustmentEvent**is generated by a scroll bar. There are five types of adjustment events. The **AdjustmentEvent**class defines integer constants that can be used to identify them.

BLOCK_DECREMENT	The user clicked inside the scroll bar to decrease its value.
BLOCK_INCREMENT	The user clicked inside the scroll bar to increase its value.
TRACK	The slider was dragged.
UNIT_DECREMENT	The button at the end of the scroll bar was clicked to decrease its value.
UNIT_INCREMENT	The button at the end of the scroll bar was clicked to increase its value.

- **The ComponentEvent Class**
  A **ComponentEvent**is generated when the size, position, or visibility of a component is changed. There are four types of component events. The **ComponentEvent**class defines integer constants that can be used to identify them.

COMPONENT_HIDDEN	The component was hidden.
COMPONENT_MOVED	The component was moved.
COMPONENT_RESIZED	The component was resized.
COMPONENT_SHOWN	The component became visible.

- **The ContainerEvent Class**
  A **ContainerEvent**is generated when a component is added to or removed from a container. There are two types of container events. The **ContainerEvent**class defines **int**constants that can be used to identify them: **COMPONENT_ADDED** and **COMPONENT_REMOVED**. They indicate that a component has been added to or removed from the container. **ContainerEvent**is a subclass of **ComponentEvent**and has this constructor:

  ContainerEvent(Component *src*, int*type*, Component *comp*)

  Here, *src*is a reference to the container that generated this event. The type of the event is specified by *type,* and the component that has been added to or removed from the container is *comp.*
- **The FocusEvent Class**
  A **FocusEvent**is generated when a component gains or loses input focus. These events are identified by the integer constants **FOCUS_GAINED** and **FOCUS_LOST**. **FocusEvent**is a subclass of **ComponentEvent**and has these constructors:

  FocusEvent(Component *src*, int*type*)
  FocusEvent(Component *src*, int*type*, boolean*temporaryFlag*)
  FocusEvent(Component *src*, int*type*, boolean*temporaryFlag*, Component *other*)

Here, *src*is a reference to the component that generated this event. The type of the event is specified by *type.* The argument *temporaryFlag*is set to **true** if the focus event is temporary. Otherwise, it is set to **false**. (A temporary focus event occurs as a result of another user interface operation. For example, assume that the focus is in a text field. If the user moves the mouse to adjust a scroll bar, the focus is temporarily lost.) The other component involved in the focus change, called the *opposite component,* is passed in *other.* Therefore, if a **FOCUS_GAINED** event occurred, *other* will refer to the component that lost focus. Conversely, if a **FOCUS_LOST** event occurred, *other* will refer to the component that gains focus.

ii) Event listener interface

- The delegation event model has two parts: sources and listeners. Listeners arecreated by implementing one or more of the interfaces defined by the **java.awt.event**package. When an event occurs, the event source invokes the appropriate method defined by the listener and provides an event object as its argument.

- **The ActionListener Interface**

This interface defines the **actionPerformed( )**method that is invoked when an action event occurs.

Interface	Description
ActionListener	Defines one method to receive action events.
AdjustmentListener	Defines one method to receive adjustment events.
ComponentListener	Defines four methods to recognize when a component is hidden, moved, resized, or shown.
ContainerListener	Defines two methods to recognize when a component is added to or removed from a container.
FocusListener	Defines two methods to recognize when a component gains or loses keyboard focus.
ItemListener	Defines one method to recognize when the state of an item changes.
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.
MouseListener	Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.
MouseMotionListener	Defines two methods to recognize when the mouse is dragged or moved.
MouseWheelListener	Defines one method to recognize when the mouse wheel is moved.

- **The ComponentListener Interface**

This interface defines four methods that are invoked when a component is resized, moved,shown, or hidden. Their general forms are shown here:

void componentResized(ComponentEvent*ce*)
void componentMoved(ComponentEvent*ce*)
void componentShown(ComponentEvent*ce*)
void componentHidden(ComponentEvent*ce*)

- **The ContainerListener Interface**

This interface contains two methods. When a component is added to a container,**componentAdded( )**is invoked. When a component is removed from a container,**componentRemoved( )**is invoked. Their general forms are shown here:

void componentAdded(ContainerEvent*ce*)
void componentRemoved(ContainerEvent*ce*)

- **The FocusListener Interface**

This interface defines two methods. When a component obtains keyboard focus, **focusGained( )**is invoked. When a component loses keyboard focus, **focusLost( )**is called. Their generalforms are shown here:

void focusGained(FocusEvent*fe*)
void focusLost(FocusEvent*fe*)

iii) Adapter class

- Adapter classes are used to simplify the process of event handling in Java. As we know that when we implement any interface all the methods defined in that interface needs to be override in the class, which is not desirable in the case of Event Handling.
- Adapter classes are useful as they provide empty implementation of all methods in an event listener interface. In this you can define a new class to act as event listener by extending one of the adapter

classes and implementing only those methods that you want to use in your program.

- Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface. You can define a new class to act as an event listener by extending one of the adapter classes and implementing only those events in which you are interested.
- For example, the **MouseMotionAdapter**class has two methods, **mouseDragged( )**and **mouseMoved( )**, which are the methods defined by the **MouseMotionListener**interface. If you were interested in only mouse drag events, then you could simply extend **MouseMotionAdapter**and override **mouseDragged( )**. The empty implementation of **mouseMoved( )**would handle the mouse motion events.

iv) Event sources

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

public void add*Type*Listener(*Type*Listener*el*)

- *Type* is the name of the event, and *el* is a reference to the event listener. For example, the method that registers a keyboard event listener is called **addKeyListener( )**. When an event occurs, all registered listeners are notified and receive a copy of the event object. This is known as *multicasting* the event. Notifications are sent only to listeners that register to receive them. Some sources may allow only one listener to register.

public void add*Type*Listener(*Type*Listener*el*)
throwsjava.util.TooManyListenersException

- *Type* is the name of the event, and *el* is a reference to the event listener. When such an event occurs, the registered listener is notified. This is known as *unicasting* the event. A source must also provide a method that allows a listener to unregister an interest in a specific type of event.

public void remove*Type*Listener(*Type*Listener*el*)

- *Type* is the name of the event, and *el* is a reference to the event listener. For example, to remove a keyboard listener, you would call **removeKeyListener( )**. The methods that add or remove listeners are

provided by the source that generates events.

Course Outcomes		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1:	Explain of the principles of OOP - Abstraction, Polymorphism, Inheritance, Encapsulation.	2	1	1	1	1	-	-	-	1	1	-	-
CO2:	Apply object oriented concepts to design simple Java Programs	2	2	1	1	1	-	-	-	1	1	-	-
CO3:	Use Exception Handling in Java.	2	2	1	1	1	-	-	-	1	-	-	1
CO4:	Implement User interface using Swing and Applets	2	2	1	1	1	-	-	-	1	-	-	-
CO5:	Explain the concepts of Multithreaded Programming in Java	2	2	1	1	1	-	-	-	1	-	-	1
CO6:	Implement JSP Script communicate with different databases using JDBC driver and remote servers.	2	2	1	1	2	-	-	-	1	-	-	1

Cognitive level	KEYWORDS
L1	List, define, tell, describe, identify, show, label, collect, examine, tabulate, quote, name, who, when, where, etc.
L2	summarize, describe, interpret, contrast, predict, associate, distinguish, estimate, differentiate, discuss, extend
L3	Apply, demonstrate, calculate, complete, illustrate, show, solve, examine, modify, relate, change, classify, experiment, discover.
L4	Analyze, separate, order, explain, connect, classify, arrange, divide, compare, select, explain, infer.
L5	Assess, decide, rank, grade, test, measure, recommend, convince, select, judge, explain, discriminate, support, conclude, compare, summarize.

PO1 - *Engineering knowledge*; PO2 - *Problem analysis*; PO3 - *Design/development of solutions*; PO4 - *Conduct investigations of complex problems*; PO5 - *Modern tool usage*; PO6 - *The Engineer and society*; PO7- *Environment and sustainability*; PO8 – *Ethics*; PO9 - *Individual and team work*; PO10 - *Communication*; PO11 - *Project management and finance*; PO12 - *Life-long learning*