Module 4 Solutions

1. What is multithreading? Explain any two advantages of multithreaded programs.

Multithreading: The process of executing multiple threads simultaneously is known as **multithreading. In** Multithreading (Thread-based multi-tasking), a single program concurrently executes several tasks e.g. a text editor printing and spell-checking text. Java provides built-in support for multithreading. The main purpose of multithreading is to provide simultaneous execution of two or more parts of a program to maximum utilize the CPU time. A multithreaded program contains two or more parts that can run concurrently. Each such part of a program is called thread. Each thread defines a separate path of execution.

Advantages of Multithreading:

i) Threads are light weight tasks as compared processes which are heavyweight tasks
ii) inter-thread communication is inexpensive, threads share the same memory address space
iii) context-switching from one thread to another is low-cost

2. Provide typical applications where multithreaded programming is used.

Few practical applications where multithreaded programming is used are described below:

- A word processor, **one thread may** check for spelling mistakes while another thread stores or process the inputs simultaneously.
- Web server: where multiple threads works on multiple requests sent by clients.
- Many GUI frameworks are multi-threaded.
- Image processing can often be done in parallel (e.g. split the image into 4 and do the work in 1/4 of the time)
- Rendering of animation (from 3DMax, etc.) is massively parallel as each frame can be rendered independently to others, meaning that 10's or 100's of computers can be chained together to help out.

Concrete Examples:

- **Microsoft Word**: Edit document while the background grammar and spell checker works to add all the green and red squiggle underlines.
- **Microsoft Excel**: Automatic background recalculations after cell edits
- **Mozilla Firefox** Web Browser: Dispatch multiple threads to load each of the several HTML references in parallel during a single page load. Speeds page loads and maximizes TCP/IP data throughput.

3. What is multithreading? Write a program to create multiple threads in JAVA

What is multithreading? Refer question 1 above

```
// A program may spawn as many threads as it needs.
class NewThread implements Runnable {
    String name;
    Thread t;
    NewThread(String threadname) {
            name = threadname;
            t = new Thread(this, name);
            System.out.println("New thread: " + t);
            t.start();
    }
    public void run() {
            try {
                    for (int i = 5; i > 0; i--) {
                            System.out.println(name + ": " + i);
                            Thread.sleep(1000);
                    }
            } catch (InterruptedException e) {
                    System.out.println(name + "Interrupted");
            }
            System.out.println(name + " exiting.");
    }
}
```

```java
// The MultiThreadDemo class creates three threads then waits until they all finish:
class MultiThreadDemo {
   public static void main(String args[]) {
          new NewThread("One");
          new NewThread("Two");
          new NewThread("Three");
          try {
                 Thread.sleep(10000);
          } catch (InterruptedException e) {
                 System.out.println("Main thread Interrupted");
          }
          System.out.println("Main thread exiting.");
   }
}
```

4. **What is meant by multithreaded programming?** Refer question 1 above. Multithreading and multithreaded programming are same.

5. **What do you mean by thread? Explain the different ways of creating threads. OR**
   What is thread? Explain two ways of creating a thread in JAVA with example.

Thread: A multithreaded program contains two or more parts that can run concurrently. Each such part of a program is called **thread**. Threads are lightweight tasks:

1) they share the same address space of the process they belong to
2) they cooperatively share the same process
3) inter-thread communication is inexpensive
4) context-switching from one thread to another is low-cost

Explain two ways of creating a thread in JAVA with example.

Following are the two ways of creating a new thread:
1. by implementing the Runnable interface
2. by extending the Thread class

1. **To create a new thread by implementing the Runnable interface:**
   1) create a class that implements the run method (inside this method, we define the code that constitutes the new thread):
   public void run()

   2) Instantiate a Thread object within that class, a possible constructor is:
   Thread(Runnable threadOb, String threadName)

   3) Call the start method on this object (start calls run):
   void start()

```java
class NewThread implements Runnable {
   Thread t;
   // Creating and starting a new thread. Passing this to the Thread constructor –
   // the new thread will call this object’s run method:
   NewThread() {
          t = new Thread(this, "Demo Thread");
          System.out.println("Child thread: " + t);
          t.start();
   }
   //This is the entry point for the newly created thread,
   // a five-iterations loop with a half-sec pause
   public void run() {
          try {
                 for (int i = 5; i > 0; i--) {
                        System.out.println("Child Thread: " + i);
                        Thread.sleep(500);
                 }
          } catch (InterruptedException e) {
                 System.out.println("Child interrupted.");
          }
```

```
            System.out.println("Exiting child thread.");
    }
}
class ThreadDemo {
    public static void main(String args[]) {
            // A new thread is created as an object NewThread:
            new NewThread();
            // After calling the NewThread start method, control returns here.
            // Both threads (new and main) continue concurrently.
            try {
                    for (int i = 5; i > 0; i--) {
                            System.out.println("Main Thread: " + i);
                            Thread.sleep(1000);
                    }
            } catch (InterruptedException e) {
                    System.out.println("Main thread interrupted.");
            }
            System.out.println("Main thread exiting.");
    }
}
```

2. The second way to create a new thread by extending Thread class

1) Create a new class that extends Thread
2) Create an instance of that class
3) Thread provides both run and start methods:
4) The extending class must override run method
5) It must also call the start method

```
// The new thread class extends Thread:
class NewThread extends Thread {
    // Create a new thread by calling the Thread's constructor and start method:
    NewThread() {
            super("Demo Thread");
            System.out.println("Child thread: " + this);
            start();
    }
    // NewThread overrides the Thread's run method:
    public void run() {
            try {
                    for (int i = 5; i > 0; i--) {
                            System.out.println("Child Thread: " + i);
                            Thread.sleep(500);
                    }
            } catch (InterruptedException e) {
                    System.out.println("Child interrupted.");
            }
            System.out.println("Exiting child thread.");
    }
}

class ExtendThread {
    public static void main(String args[]) {
            new NewThread();
            // After a new thread is created the new and
            // main threads continue concurrently…
            try {
                    for (int i = 5; i > 0; i--) {
                            System.out.println("Main Thread: " + i);
                            Thread.sleep(1000);
                    }
            } catch (InterruptedException e) {
                    System.out.println("Main thread interrupted.");
            }
            System.out.println("Main thread exiting.");
    }
}
```

6. Write a JAVA program to create two threads, one displays "computer science" and another displays "information science" five times.

```
// Create multiple threads.
class NewThread implements Runnable {
  String name; // name of thread
  Thread t;

  NewThread(String threadname) {
    name = threadname;
    t = new Thread(this, name);
    System.out.println("New thread: " + name);
    t.start(); // Start the thread
  }

  // This is the entry point for thread.
  public void run() {
    try {
      for(int i = 5; i > 0; i--) {
        System.out.println(name + ": " + i);
        Thread.sleep(1000);
      }
    } catch (InterruptedException e) {
      System.out.println(name + "Interrupted");
    }
    System.out.println(name + " exiting.");
  }
}

class MultiThreadDemo {
  public static void main(String args[]) {
    new NewThread("Computer Science"); // start threads
    new NewThread("Information Science");

    try {
      // wait for other threads to end
      Thread.sleep(10000);
    } catch (InterruptedException e) {
      System.out.println("Main thread Interrupted");
    }

    System.out.println("Main thread exiting.");
  }
}
```

7. Write a JAVA program which creates two threads. One thread displays “VTU – Belgaum” for every 100 seconds and another thread displays “Karnataka” for every 50 seconds, continuously.

```
class VTU implements Runnable {
	VTU() {
		new Thread(this, "VTU-Belgaum").start();
	}

	public void run() {
		try {
			while(true) {
				System.out.println("VTU-Belgaum");
				Thread.sleep(100000); // 100 seconds
			}
		} catch (InterruptedException e) {
			System.out.println("VTU thread interrupted.");
		}
	}
}
class Karnataka implements Runnable {
	Karnataka() {
		new Thread(this, "Karnataka").start();
	}
```

```
	public void run() {
		try {
			while(true) {
				System.out.println("Karnataka");
				Thread.sleep(50000); // 50 seconds
			}
		} catch (InterruptedException e) {
			System.out.println("Karnataka thread interrupted.");
		}
	}
}

class Display {
  public static void main(String args[]) {
    new VTU();
    new Karnataka();
    System.out.println("Press Control-C to stop.");
  }
}
```

8. With syntax, explain use Of isAlive( ) and join( ) methods.
Write a Java program that creates multiple child threads and also ensures that the main thread is the last to stop.

Use Of isAlive( ) and join( ) methods

How can one thread know when another thread has ended? Two methods are useful:

1. **final boolean isAlive():** returns true if the thread upon which it is called is still running and false otherwise
2. **final void join() throws InterruptedException**: waits until the thread on which it is called terminates

Example of isAlive() and join() : Java program that creates multiple child threads and also ensures that the main thread is the last to stop

```
class NewThread implements Runnable {
		String name;
		Thread t;
		NewThread(String threadname) {
			name = threadname;
			t = new Thread(this, name);
			System.out.println("New thread: " + t);
			t.start();
		}
		public void run() {
			try {
				for (int i = 5; i > 0; i--) {
					System.out.println(name + ": " + i);
					Thread.sleep(1000);
				}
			} catch (InterruptedException e) {
				System.out.println(name + " interrupted.");
			}
			System.out.println(name + " exiting.");
		}
}
class DemoJoin {
	public static void main(String args[]) {
		NewThread ob1 = new NewThread("One");
		NewThread ob2 = new NewThread("Two");
		NewThread ob3 = new NewThread("Three");
		//Checking if those threads are still alive:
		System.out.println(ob1.t.isAlive());
		System.out.println(ob2.t.isAlive());
		System.out.println(ob3.t.isAlive());
		// Waiting until all three threads have finished:
		try {
			System.out.println("Waiting to finish.");
			ob1.t.join();
			ob2.t.join();
```

```
            ob3.t.join();
        } catch (InterruptedException e) {
            System.out.println("Main thread Interrupted");
        }
        System.out.println(ob1.t.isAlive());
        System.out.println(ob2.t.isAlive());
        System.out.println(ob3.t.isAlive());
           System.out.println("Main thread exiting.");
    }
}
```

9. What is synchronization? Explain with an example how synchronization is implemented in JAVA. OR What is the need of synchronization? Explain with an example how synchronization is implemented in JAVA. OR How synchronization can be achieved for threads in Java? Explain with syntax. OR Discuss the significance of synchronization in Java. OR What is synchronization? What is role of synchronization in threads? Demonstrate a program using synchronized methods. OR What is the need for synchronization? How can synchronization be achieved in Java?

What is Synchronization? (It's Significance)

Multi-threading introduces asynchronous behavior to a program. Therefore it is necessary to prevent two threads from simultaneously writing and reading the same object.

When several 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. This way is called **synchronization**.

How synchronization is achieved/implemented in JAVA?

- Synchronization uses the concept of monitors:

  1) only one thread can enter a monitor at any one time
  2) other threads have to wait until the thread exits the monitor

- Java implements synchronization in two ways:

  1) Using synchronized methods:

     - Classes can define so-called synchronized methods
     - Each object has its own implicit monitor that is automatically entered when one of the object's synchronized methods is called
     - Once a thread is inside a synchronized method, no other thread can call any other synchronized method on the same object

     ```
     synchronized void method() {
                  …
     }
     ```

  2) Using synchronized statement:

     In order to synchronize access to instances of a class that was not designed for multithreading, calls to the methods of this class are put inside the synchronized block:

     ```
     synchronized(object) {
            …
     }
     ```

Synchronization Example:

```
// The call method tries to print the message string inside brackets, pausing the current thread
for one second in the middle:
```

```
class Callme {
	synchronized void call(String msg) {
		System.out.print("[" + msg);
		try {
			Thread.sleep(1000);
		} catch(InterruptedException e) {
			System.out.println("Interrupted");
		}
		System.out.println("]");
	}
}

// Caller constructor obtains references to the Callme object and String, stores them in the target
and msg variables, then creates a new thread:

class Caller implements Runnable {
	String msg;
	Callme target;
	Thread t;
	public Caller(Callme targ, String s) {
		target = targ;
		msg = s;
		t = new Thread(this);
		t.start();
	}
	// The Caller's run method calls the call method on the target instance
	// of Calllme, passing in the msg string:
	public void run() {
		target.call(msg);
	}
}
// Synch class creates a single instance of Callme and three of Caller, each with a message.
// The Callme instance is passed to each Caller:
class Synch {
	public static void main(String args[]) {
		Callme target = new Callme();
		Caller ob1 = new Caller(target, "Hello");
		Caller ob2 = new Caller(target, "Synchronized");
		Caller ob3 = new Caller(target, "World");
		// Waiting for all three threads to finish:
		try {
			ob1.t.join(); ob2.t.join(); ob3.t.join();
		} catch(InterruptedException e) {
			System.out.println("Interrupted");
		}
	}
}
```

10. Explain the producer-consumer problem with a program. OR
Write a java program to implement a producer-consumer problem, using threads. OR
What is producer-consumer problem? Explain solution for producer-consumer problem with a program. OR
Explain the role of synchronization with producer and consumer problem.

Producer-Consumer (Queuing) problem

Consider the classic queuing problem where one thread (producer) is producing some data and another (consumer) is consuming this data:

1) producer should not overrun the consumer with data
2) consumer should not consume the same data many times

This producer-consumer system uses wait and notify to synchronize the behavior of the producer and consumer.

```
// The queue class introduces a boolean variable valueSet used by the get and put methods:
class Q {
	int n;
	boolean valueSet = false;
	// Inside get, wait is called to suspend the execution of Consumer until Producer notifies
that some data is ready:
```

```
	synchronized int get() {
		if (!valueSet)
		try {
			wait();
		}
		catch(InterruptedException e) {
			System.out.println("InterruptedException");
		}
		// After the data has been obtained, get calls notify to tell Producer
		// that it can put more data on the queue:
		System.out.println("Got: " + n);
		valueSet = false;
		notify();
		return n;
	}
	// Inside put, wait is called to suspend the execution of Producer until Consumer has removed
	the item from the queue:
	synchronized void put(int n) {
		if (valueSet)
		try {
			wait();
		} catch(InterruptedException e) {
			System.out.println("InterruptedException");
		}
		// After the next item of data is put in the queue, put calls notify to
		// tell Consumer that it can remove this item:
		this.n = n;
		valueSet = true;
		System.out.println("Put: " + n);
		notify();
	}
}
// Producer creates a thread that keeps producing entries for the queue:
class Producer implements Runnable {
	Q q;
	Producer(Q q) {
		this.q = q;
		new Thread(this, "Producer").start();
	}
	Public void run() {
		int i = 0;
		while(true) {
			q.put(i++);
		}
	}
}
// Consumer creates a thread that keeps consuming entries in the queue:
class Consumer implements Runnable {
	Q q;
	Consumer(Q q) {
		this.q = q;
		new Thread(this, "Consumer").start();
	}
	public void run() {
			while(true) {
			q.get();
		}
	}
}
// The PCFixed class first creates a single Queue instance q, then creates a Producer and Consumer
that share this q:
class PCFixed {
	public static void main(String args[]) {
		Q q = new Q();
		new Producer(q);
		new Consumer(q);
		System.out.println("Press Control-C to stop.");
	}
}
```

11. **Explain with an example how inter thread communication is implemented in JAVA**

In Java, Inter-thread communication relies on three methods provided by the Object class:

1) final void wait() throws InterruptedException
   - wait() tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify().
2) final void notify()
   - notify() wakes up the first thread that called wait() on the same object
3) final void notifyAll()
   - notifyAll() wakes up all the threads that called wait() on the same object; the highest-priority thread will run first.

All three methods must be called from within a synchronized context.

Example: **Discuss the Producer-Consumer example from the question 10 above.**

12. **What is meant by thread priority? How to assign and get the thread priority? OR**
    Describe the thread priority? How is it assigned?

What is meant by Thread Priority? Or Describe Thread Priority

- Thread Priority is used by the scheduler to decide when each thread should run.
- In theory, higher-priority thread gets more CPU than lower-priority thread and threads of equal priority should get equal access to the CPU.
- In practice, the amount of CPU time that a thread gets depends on several factors besides its priority.

How to assign and get the thread priority?

- Threads can be assigned the priority using the following method
  final void setPriority(int level)
  where level specifies the new priority setting between:
  MIN_PRIORITY (1)
  MAX_PRIORITY (10)
  NORM_PRIORITY (5)
- To get the current priority setting of the thread following method is used
  final int getPriority()

13. **Explain the delegation event model used to handle events in Java. What are events, event listeners and event sources? OR**
    Define the delegation event model. Briefly explain the role of: i) Event classes ii) Event listener interfaces iii) Source of events OR
    Explain the mechanism of event delegation model.

Delegation Event Model:

The **Delegation Event Model** is a Model used by Java to handle user interaction with GUI components. It describes how your program can respond to user interaction. Three important players of the Delegation Event Model are:

1. Event Source
2. Event Listener/Handler
3. Event Object

Event Source: Event Source is a GUI component that generates the event. Example: a button or text field

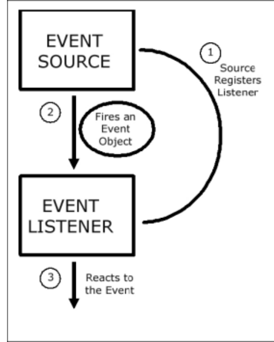
Event Listener/Handler

- Receives and handles events
- Contains business logic
- Example: displaying information useful to the user, computing a value

Event Object:

- Created when an event occurs (i.e., user interacts with a GUI component)
- Contains all necessary information about the event that has occurred e.g. Type of event that has occurred, Source of the event etc.
- Represented by an Event class

Mechanism / Control Flow of Delegation Event Model



- An Event listener should be registered with an event source
- Once registered, listener waits until an event occurs
- When an event occurs
    - An event object created by the event source
    - Event object is fired by the event source to the registered listeners (method of event listener is called with an event object as a parameter)
- Once the listener receives an event object from the source
    - Deciphers the event
    - Processes the event that occurred.

14. Write a note on event listener interface.

Event Listener Interfaces: Event Listeners are created 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. Following Table lists commonly used listener interfaces and provides a brief description of the methods that they define.

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.
TextListener	Defines one method to recognize when a text value changes.
WindowFocusListener	Defines two methods to recognize when a window gains or loses input focus.
WindowListener	Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

TABLE 22-3 Commonly Used Event Listener Interfaces

15. Explain any three event listener interfaces with its function or methods.

- **The KeyListener Interface:** This interface defines three methods. The **keyPressed( )** and **keyReleased( )** methods are invoked when a key is pressed and released, respectively. The **keyTyped( )** method is invoked when a character has been entered. The general forms of these methods are shown here:
  void keyPressed(KeyEvent *ke*)
  void keyReleased(KeyEvent *ke*)
  void keyTyped(KeyEvent *ke*)

- **The MouseListener Interface:** This interface defines five methods. If the mouse is pressed and released at the same point, **mouseClicked( )** is invoked. When the mouse enters a component, the **mouseEntered()** method is called. When it leaves, **mouseExited()** is called. The **mousePressed()** and **mouseReleased()** methods are invoked when the mouse is pressed and released, respectively. The general forms of these methods are shown here:
  void mouseClicked(MouseEvent *me*)
  void mouseEntered(MouseEvent *me*)
  void mouseExited(MouseEvent *me*)
  void mousePressed(MouseEvent *me*)
  void mouseReleased(MouseEvent *me*)

- **The WindowListener Interface:** This interface defines seven methods. The **windowActivated()** and **windowDeactivated()** methods are invoked when a window is activated or deactivated, respectively. If a window is iconified, the **windowIconified()** method is called. When a window is deiconified, the **windowDeiconified()** method is called. When a window is opened or closed, the **windowOpened()** or **windowClosed()** methods are called, respectively. The **windowClosing()** method is called when a window is being closed. The general forms of these methods are:
  void windowActivated(WindowEvent *we*)
  void windowClosed(WindowEvent *we*)
  void windowClosing(WindowEvent *we*)
  void windowDeactivated(WindowEvent *we*)
  void windowDeiconified(WindowEvent *we*)
  void windowIconified(WindowEvent *we*)
  void windowOpened(WindowEvent *we*)

16. Write a note Event Classes provided by JAVA.

The classes that represent events are at the core of Java's event handling mechanism. At the root of the Java event class hierarchy is **EventObject**, which is in **java.util**. It is the superclass for all events. The package **java.awt.event** defines many types of events that are generated by various user interface elements. Table shows several commonly used event classes and provides a brief description of when they are generated.

Event Class	Description
ActionEvent	Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.
AdjustmentEvent	Generated when a scroll bar is manipulated.
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible.
ContainerEvent	Generated when a component is added to or removed from a container.
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 choice selection is made or a checkable menu item is selected or deselected.
KeyEvent	Generated when input is received from the keyboard.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; 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, deiconified, iconified, opened, or quit.

TABLE 22-1 Main Event Classes in **java.awt.event**

17. Give an example for using keyboard event.

```
// Demonstrate the key event handlers.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
```

```
  <applet code="SimpleKey" width=300 height=100>
  </applet>
*/
public class SimpleKey extends Applet
  implements KeyListener {
  String msg = "";
  int X = 10, Y = 20; // output coordinates
  public void init() {
    addKeyListener(this);
  }
  public void keyPressed(KeyEvent ke) {
    showStatus("Key Down");
  }
  public void keyReleased(KeyEvent ke) {
    showStatus("Key Up");
  }
  public void keyTyped(KeyEvent ke) {
    msg += ke.getKeyChar();
    repaint();
  }
  public void paint(Graphics g) { // Display keystrokes.
    g.drawString(msg, X, Y);
  }
}
```

18. Give an example for using mouse event.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class MouseEvents extends Applet
       implements MouseListener {
       String msg = "";
       int mouseX = 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();
        }
        // Display msg in applet window at current X,Y location.
                public void paint(Graphics g) {
                g.drawString(msg, mouseX, mouseY);
        }
}
```

19. Explain the adapter classes, with examples. OR
What is an adapter class? Demonstrate, with an example. OR
Describe the significance of adapter class, with an example.

Adapter Classes

An adapter class provides an empty implementation of all methods in an event listener interface. Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface.You can define a new class to act as an event listener by extending one of the adapter classes and implementing only those events in which you are interested. Adapter classes simplify the creation of event handlers

For example, the **MouseMotionAdapter** class has two methods, **mouseDragged()** and **mouseMoved()**, which are the methods defined by the **MouseMotionListener** interface. If you were interested in only mouse drag events, then you could simply extend **MouseMotionAdapter** and override **mouseDragged( )**. The empty implementation of **mouseMoved( )** would handle the mouse motion events for you.Table lists the commonly used adapter classes in **java.awt.event.**

Adapter Class	Listener Interface
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener

Adapter Class Example: Mouse Adapter

```
// Demonstrate an adaptor.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
  <applet code="AdapterDemo" width=300 height=100>
  </applet>
*/
public class AdapterDemo extends Applet {
        public void init() {
                addMouseListener(new MyMouseAdapter(this));
        }
}

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

20. Briefly explain the role of:
    a. ActionEvent class
    b. AdjustmentEvent class.