II - ASSIGNMENT

(Start Writing From Here)

1. Demonstrate the usage of thread synchronization in java with a suitable example.

Thread synchronization in java is a way of programming several threads to carry out independent tasks easily. It is capable of controlling access to multiple threads to a particular shared resource. The main reason for using thread synchronization are as follows

- To prevent interference between threads
- To prevent the problem of consistency

Types of Thread Synchronization

In java, there are two types of thread synchronization

1. Process synchronization
2. Thread synchronization

Process synchronization: The process means a program in execution and runs independently isolated from other processes. CPU time, memory, etc resources are allocated by the operating system

Thread synchronization:- It refers to the concept where only one thread is executed at a time while other threads are in the waiting state. This process is called thread synchronization. It is used because it avoids interface of thread and the problem of inconsistency

In java, thread synchronization is further divided into two types:

- Mutual exclusive - it will keep the threads from interfacing with each other while sharing any resources
- Inter-thread communication. It is a mechanism in java in which a thread running in the critical section is paused and another thread is allowed to enter or lock the same critical section that

is executed

Mutual Exclusive

It is used for preventing threads from interfering with each other and to keep distance between the threads. Mutual exclusive is achieved using the following:

- Synchronized Method
- Synchronized Block
- static Synchronization.

Synchronized Method

If we use the synchronized keywords in any method then the method is Synchronized Method.

- It is used to lock an object for any shared resources.
- The object gets the lock when the synchronized method is called.
- The lock won't be released until the thread completes its function.

Synchronized Block

Suppose you don't want to synchronize the entire method, you want to synchronize few lines of code in the method, then a synchronized block helps to synchronize those few lines of code. It will take the object as a parameter.

Static Synchronization

In java, every object has a single lock (monitor) associated with it. The thread which is entering into synchronized method or synchronized block will get that lock, all others threads which are remaining to use the shared resources have to wait for the completion of first thread and release of the lock

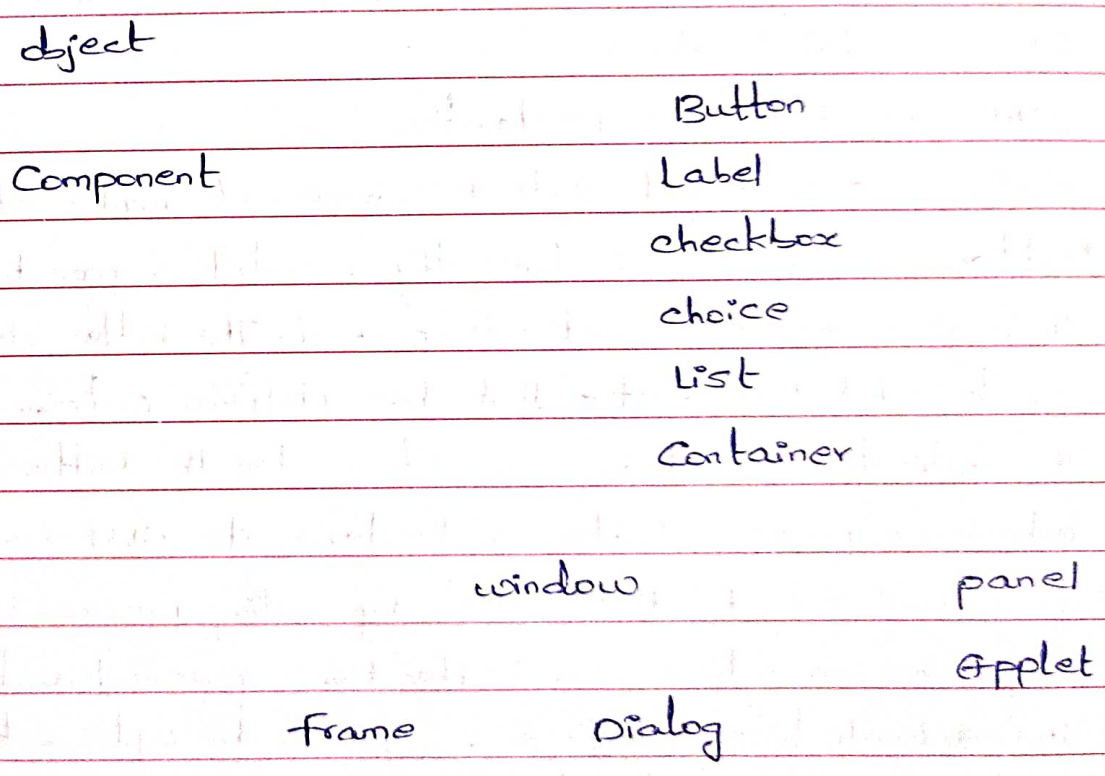
Program:-

```
package first Thread;
public class Bank1 implements Runnable()
{
 int amount, account = 5000;
public Bank1 (int amount)
{
 this. amount = amount;
}
 public synchronized void run() {
String s = (" Thread Current thread().getName()");
if (account >= amount) {
System.out.println (s+: "Amount withdrawn successfully").
account = account - amount;
}
else {
 System.out.println (s+" Amount insufficient");
}
public static void main (String args[]) {
 Bank1 b = new Bank1(5000);
Thread t1 = new Thread (b, "Sowmya");
Thread t2 = new Thread (b, "Teja");
Thread t3 = new Thread (b, "Ammu");
t1.start();
t2.start();
```

```
la.start();
}
}
```

2. Sketch the AWT hierarchy with a neat diagram and explain some of the user interface components.

The hierarchy of java AWT classes are given below



User Interface Components

a. labels
b. Buttons
c. Scrollbars
d. TextComponent → Textfield, Textarea
e. checkbox
f. checkbox groups

g. choices
h. Lists
i Panels
j. ScrollPane
k. Dialogs
l. Menubar.

a. Labels: The object of the label class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by a programmer but user cannot edit it directly.

It is called a passive control as it doesnot create any event when it is accessed. To create a label we need to create the object of label class

AWT label class Declaration

public class Label extends Component implements Accessible

b. Buttons: A button is basically a control component with a label that generates an event when pushed. The Button class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. When we press a button and release it, AWT sends an instance of ActionEvent to that button by calling processEvent on the button. To perform an action on a button being pressed and released, the ActionListener interface needs to be implemented.

Button class Declaration

public class Button extends Component implements Accessible

c. Scroll bars: The object of ScrollBar class is used to add horizontal and vertical Scrollbar. ScrollBar is a GUI Component allows us to see invisible number of rows and

columns

Scrollbar class declaration

public class Scrollbar extends component implements Adjustable, Accesible

d. checkbox :- The checkbox 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 "of" to "on"

Checkbox class declaration :-

public class checkbox extends Component implements Item Selectable, Accessible.

e. Checkbox groups :- The object of CheckboxGroup class is used to group together a set of checkbox. At a time only one checkbox button is allowed to be in "on" state and remaining checkbox button in "off" state. It inherits the object class.

CheckboxGroup enables you to create radio buttons in AWT

Checkbox Group class Declaration

public class CheckboxGroup extends Object implements Serializ

f. choice :- The object of choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits Component class.

Choice class Declaration :-

public class Choice extends Component implements ItemSelectable, Accessible

3. Distinguish Event Listener from Event Adapters.

Event listeners :-

When we implement a listener interface in any class then we must have to implement all the methods declared in that interface because all the methods in an interface are abstract and must be override in class which implements it. For example consider the following program which demonstrates handling of key events by implementing listener interface.

KeyListener Example.java

```
import java.awt.*;
import java.awt.event.*;
public class KeyListenerExample extends Frame implements KeyListener{
  Label l;
  TextArea area;
  KeyListenerExample() {
     l = new Label();
     l.setBounds(20, 50,100,20);
     area = new TextArea();
     area.setBounds (20, 80, 300,300);
     area.addKeyListener(this);
     add(l);
     add(area);
     setSize (400, 400);
     setLayout(null);
     setVisible(true);
```

```
}
public void keyPressed (KeyEvent e) {
    l.setText ("Key Pressed");
}
public void keyReleased (KeyEvent e) {
    l.setText ("Key Released");
}
public void keyTyped (KeyEvent e) {
    l.setText ("Key Typed");
}
public static void main (String[] args) {
    new KeyListenerExample();
  }
}
```

On the above example for handling key events implements KeyListener interface on the KeyEvent has to implement all the three methods listed below.

1. public void keyTyped (KeyEvent e)
2. public void keyPressed (KeyEvent e)

3. public void keyReleased (KeyEvent e)

It is not suitable solution to implement all the methods every time even we don't need them. Adapter class makes it easy to deal with this situation. An adapter class provides empty implementations of all methods defined by that interface.

4. Illustrate the Life Cycle methods of an applet with a suitable program.

Life cycle of an applet

In java, an applet is a special type of program embedded in the web page to generate dynamic content. Applet is a class in Java.

The applet life cycle can be defined as the process of how the object is created, started, stopped, and destroyed during the entire execution of its application. It basically has five core methods namely init(), start(), stop(), paint() and destroy(). These methods are invoked by the browser to execute.

Along with the browser, the applet also works on the client side, thus having less processing time.

Methods of Applet Life Cycle

There are five methods of an applet life cycle and they are

- init(): The init() method is the first method to run that initializes the applet. It can be invoked only once at the time of initialization. The web browser creates the initialized objects, i.e., the web browser runs the init() method within the applet.
- start(): The start() method contains the actual code of the applet and starts the applet. It is invoked immediately after the init() method is invoked. Every time in the browser is loaded or refreshed, the start() method is invoked. It is in an inactive state until the init() method is invoked.
- stop(): The stop() method stops the execution of the applet. The stop() method is invoked whenever the applet is stopped, minimized, or moving from one tab to another in the browser, the stop() method is invoked.
- destroy(): The destroy() method destroys the applet after its work is done. It is invoked when the applet window is closed or when the tab containing the webpage is closed. It removes the applet object from memory and is executed only once. We cannot start the applet once is destroyed.
- paint(): The paint() method belongs to the Graphics class in Java. It is used to draw shapes like circle, square, trapezium, etc., in the applet. It is executed after the start() method and when the browser or applet windows are resized.

Flow of Applet life Cycle:

These methods are invoked by the browser automatically. There

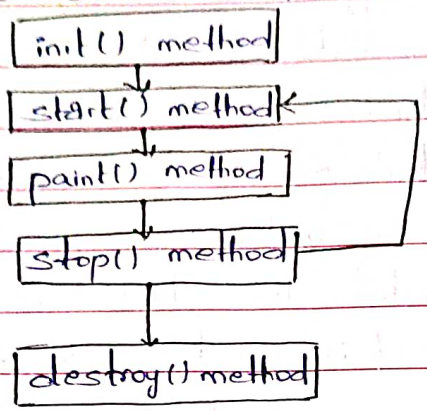
There are five methods of an applet life cycle and they are

- init(): The init() method is the first method to run that initializes the applet. It can be invoked only once at the time of initialization. The web browser creates the initialized objects, i.e., the web browser runs the init() method within the applet.
- start(): The start() method contains the actual code of the applet and starts the applet. It is invoked immediately after the init() method is invoked. Every time in the browser is loaded or refreshed, the start() method is invoked. It is in an inactive state until the init() method is invoked.
- stop(): The stop() method stops the execution of the applet. The stop() method is invoked whenever the applet is stopped, minimized, or moving from one tab to another in the browser, the stop() method is invoked.
- destroy(): The destroy() method destroys the applet after its work is done. It is invoked when the applet window is closed or when the tab containing the webpage is closed. It removes the applet object from memory and is executed only once. We cannot start the applet once is destroyed.
- paint(): The paint() method belongs to the Graphics class in Java. It is used to draw shapes like circle, square, trapezium, etc., in the applet. It is executed after the start() method and when the browser or applet windows are resized.

Flow of Applet life Cycle:

These methods are invoked by the browser automatically. There

is no need to call them explicitly.



Program :-

```
import java.applet.Applet;
import java.awt.Graphics;
public class SimpleApplet extends Applet {
    // Initialization method
    public void init() {
        // Initialization code goes here
    }
    // Start method
    public void start() {
        // Code to start or resume execution
    }
    // Paint method (required for rendering)
    public void paint(Graphics g) {
        // Drawing code goes here
        g.drawString("Hello, this is a simple Java Applet!", 20,20);
    }
```

```
    // Stop method
    public void stop() {
        // Code to stope or pause execution
    }
    // Destroy method
    public void destroy() {
        // Cleanup code goes here
    }
}
```

5. Swing provides plat-form-independent and lightweight Components Justify.

Swing is a GUI (Graphical User Interface) toolkit that provides platform-independent and lightweight components. It achieves platform independence through the use of Java's write-once-run-anywhere principle, allowing Swing applications to run on various operating systems without modifications. The term "lightweight" refers to the fact that Swing components are not dependent on the underlying operating system's native components, making them more flexible and consistent across different platforms.

Swing components are platform-independent because they are implemented in java and are not tied to specific operating system details. The JVM interprets and executes the java bytecode, allowing Swing applications to run on any system

that has a Compatible JVM.

Swing Components are considered lightweight because they are not directly tied to native components of the underlying operating system. Unlike heavyweight components, which rely on the native GUI components of the platform. Swing components are written entirely in Java and rendered by the Java runtime environment. This makes swing components consistent across different platforms and avoids reliance on platform-specific graphical elements.

Swing's platform independence and lightweight nature contribute to its versatility, allowing developers to create JAVA GUI applications that can be easily deployed and run on various operating systems without sacrificing a consistent user interface.