IAT-3, 7th Sem CSE, JAVA and J2EE 10CS753
Model Answer/Solution
Dr. P. N. Singh, Professor(CSE)

Q1 What is synchronisation? Explain the producer-consumer problem with a program. **10 M**

Ans:

Synchronization in java is the capability to control the access of multiple threads to any shared resource.
Java Synchronization is better option where we want to allow only one thread to access the shared resource.
The synchronization is mainly used to

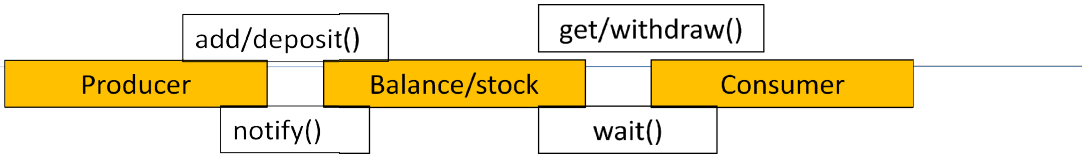
- To prevent thread interference.
- To prevent consistency problem.

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive
   - Synchronized method.
   - Synchronized block.
   - static synchronization.
2. Cooperation (Inter-thread communication in java)



Producer-Consumer Process:

- The producer–consumer problem (also known as the bounded-buffer problem) is a classic example of a multi-process synchronization problem.
- The problem describes two processes, the producer and the consumer, which share a common, fixed-size buffer used as a queue.
- The producer's job is to generate data, put it into the buffer, and start again.
- At the same time, the consumer is consuming the data (i.e. removing it from the buffer), one piece at a time.

Problem
To make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.
Solution

- The producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again.
- In the same way, the consumer can go to sleep if it finds the buffer to be empty.
- The next time the producer puts data into the buffer, it wakes up the sleeping consumer.

```
// Inter-thread communication & Producer-consumer process
   class Customer{
     int amount=20000;
     synchronized void withdraw(int amount){
        System.out.println("going to withdraw : "+amount);
        if(this.amount<amount){
          System.out.println("Less balance! waiting for deposit..");
          try{wait();}catch(Exception e){}
        }
        this.amount-=amount;
        System.out.println("withdraw completed, Balance is "+this.amount);
      }
```

```
    synchronized void deposit(int amount){
        System.out.println("going to deposit..."+amount);
        this.amount+=amount;
        System.out.println("deposit completed, Balance is "+this.amount);
        notify();
    }
  }
class PCTest{
    public static void main(String args[]){
        final Customer c=new Customer();
        new Thread(){public void run(){c.withdraw(15000);}}.start();
        new Thread(){public void run(){c.deposit(10000);}}.start();
    }
  }
```

Output:
going to withdraw : 15000
withdraw completed, Balance is 5000
going to deposit...10000
deposit completed, Balance is 15000

Q 2 (a) What are events, event listener & event sources? **5M**

Ans:
Event:

- Changing the state of an object is known as an event
- Describes the change in status of source
- Generated as result f user interaction with GUI
- Example: click on button, dragging mouse etc.
- Java application written for windows, is event driven

The java.awt.event package provides many event classes and Listener interfaces for event handling.

Event Listener:

- Also known as event handler
- Responsible for generating response to an event
- Listener is also an object
- Waits until it receives an event, process and returns it
- Needs to be registered with the source object

Event Source:

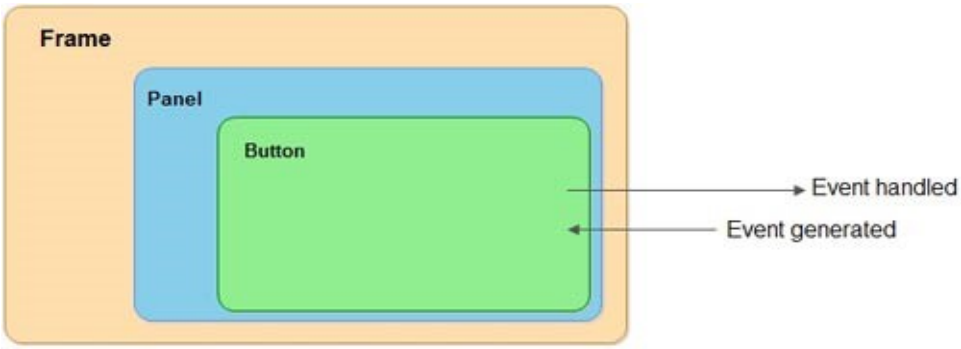
- An object on which event occurs
- Responsible for providing information of the occurred event to its handler

Q2(b): Explain the delegation event model used to handle events in Java**.** **5M**

Ans:

Delegation Event Model:

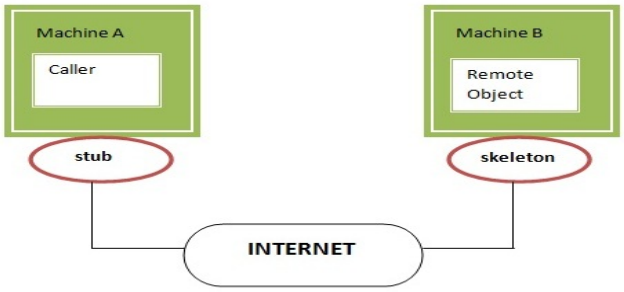
- Logical approach to handle events
- Concept of source and listener (2 key participants:)
- **Source:**
  - An object on which event occurs
  - Responsible for providing information of the occurred event to its handler
- **Listener:**
- Also known as event handler
- Responsible for generating response to an event
- Listener is also an object
- Waits until it receives an event, process and returns it
- Needs to be registered with the source object



Q3 What is RMI? Briefly explain the working of RMI in Java. **10 M**

Ans:

The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.
The RMI provides remote communication between the applications using two objects stub and skeleton which communicate with remote object.
A **remote object** is an object whose method can be invoked from another JVM.
The **stub** is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object.
The **skeleton** is an object, acts as a gateway for the server side object. All the incoming requests are routed through it.



Stub object does the following tasks:

- It initiates a connection with remote Virtual Machine (JVM),
- It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
- It waits for the result
- It reads (unmarshals) the return value or exception, and
- It finally, returns the value to the caller.

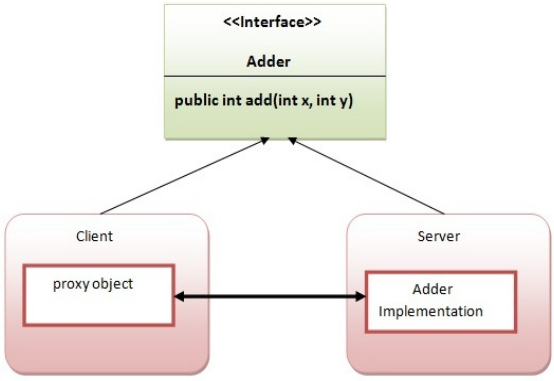
When the skeleton receives the incoming request, it does the following tasks:

- It reads the parameter for the remote method
- It invokes the method on the actual remote object, and

It writes and transmits (marshals) the result to the caller.

6 steps to write RMI program:

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmiregistry tool
5. Create and start the remote application
6. Create and start the client application



Q4 Explain different types of tags that can be used in a JSP program. **10 M**

Ans:

In JSP, java code can be written inside the jsp page using the scriptlet tag. The scripting elements / JSP tags 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

Other tags are

4. comment & 5. directive tags

JSP scriptlet tag

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

<% java source code %>

Example:

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

JSP expression tag

The code placed within JSP expression tag is *written to the output stream of the response*. So we need not write out.print() to write data. Syntax:

<%= statement %>

Example:

```
<html>
<body>
<%= "welcome to jsp" %> </body>
</html>
```

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.

Syntax :

<%! field or method declaration %>

Example - declares field

```
<html>
<body>
<%! int data=50; %>
```

```
<%= "Value of the variable is:"+data %>
</body>
</html>
```
Example - declares method:
```
<html>
<body>
<%!
int cube(int n){
return n*n*n*;
}
%>
<%= "Cube of 3 is:"+cube(3) %>
</body>
</html>
```

JAP Directive Tag:
A Directive tag opens with <%@ and closes with %>.
Example:
<%@ page import = "java.sql.*" %>

JSP Comment Tag:
A comment tag opens with <%-- and closes with --%>,
Example:
<%-- jsp comment tag --%>

Q5 Explain a code snippet to implement Remote Interface at client & the server side **10 M**

Ans:

1. create the remote interface
Extend the Remote interface and declare the RemoteException with all the methods of the remote interface. Here, example of a remote interface that extends the Remote interface. There is only one method named add() and it declares RemoteException.

```
//Adder.java
import java.rmi.*;
public interface Adder extends Remote{
public int add(int x,int y)throws RemoteException;
}
```

2. Provide the implementation of the remote interface
Either extend the UnicastRemoteObject class, or use the exportObject() method of the UnicastRemoteObject class.
In case, we extend the UnicastRemoteObject class, you must define a constructor that declares RemoteException

```
//AdderRemote.java
import java.rmi.*;
import java.rmi.server.*;
public class AdderRemote extends UnicastRemoteObject implements Adder{
AdderRemote()throws RemoteException{
super();
}
public int add(int x,int y){return x+y;}
}
```

3. create the stub and skeleton objects using the rmic tool on command prompt
rmic AdderRemote
4. Start the registry service by the rmiregistry tool on command prompt
rmiregistry 5000
5. Create and run the server application

```
//Myserver.java
import java.rmi.registry.*;
public class MyServer{
  public static void main(String args[]){
    try{ Adder stub=new AdderRemote();
         Naming.rebind("rmi://localhost:5000/sonoo",stub);
        }catch(Exception e){System.out.println(e);}
  }
}
```

6. Create & run client application

```
//MyClient.java
import java.rmi.*;
public class MyClient{
public static void main(String args[]){
try{
Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");
System.out.println(stub.add(34,4));
}catch(Exception e){}
}
}
```

Q6. Write notes on i) Stateless versus stateful session Bean ii) Entity Java Bean (with skeleton) iii) Message driven bean. **10 M**

Ans:

Stateless Session bean :

- A business object that represents business logic only.
- It doesn't have state (data).
- Conversational state between multiple method calls is not maintained by the container in case of stateless session bean.
- The stateless bean objects are pooled by the EJB container to service the request on demand.
- It can be accessed by one client at a time. In case of concurrent access, EJB container routes each request to different instance.

Important annotations: @Stateless, @PostConstruct, @PreDestroy

Stateful Session bean
A business object that represents business logic like stateless session bean but it maintains state (data).
Conversational state between multiple method calls is maintained by the container in stateful session bean.
Annotations: @Stateful, @PostConstruct, @PreDestroy, @PrePassivate, @PostActivate

Entity Java Bean

- An *entity bean* represents a business object in a persistent storage mechanism.
- Some examples of business objects are customers, orders, and products.
- In the J2EE SDK, the persistent storage mechanism is a relational database.
- Typically, each entity bean has an underlying table in a relational database, and each instance of the bean corresponds to a row in that table.

Message-Driven Bean

- A *message-driven bean* is an enterprise bean that allows J2EE applications to process messages asynchronously.
- It acts as a JMS message listener, which is similar to an event listener except that it receives messages instead of events.
- The messages may be sent by any J2EE component--an application client, another enterprise bean, or a Web component--or by a JMS application or system that does not use J2EE technology.

Message-driven beans currently process only JMS messages, but in the future they may be used to process other kinds of messages

Q7 Explain JAR files with steps and example class program. **10M**

Ans:

The **jar (Java Archive)** tool of JDK provides the facility to create the executable jar file. An executable jar file calls the main method of the class if we double click it.
To create the executable jar file, we need to create **.mf file**, also known as manifest file.
Creating manifest file: To create manifest file, you need to write Main-Class, then colon, then space, then classname then enter. For example: **myfile.mf**
Main-Class: First
In mf file, new line is must after the class name.
Creating executable jar file using jar tool:
The jar tool provides many switches, some of them are as follows:
-c creates new archive file
-v generates verbose output. It displays the included or extracted resource on the standard output.
-m includes manifest information from the given mf file.
-f specifies the archive file name
-x extracts files from the archive file
We need to write **jar** then **swiches** then **mf_file** then **jar_file** then **.classfile**
jar -cvmf myfile.mf myjar.jar First.class

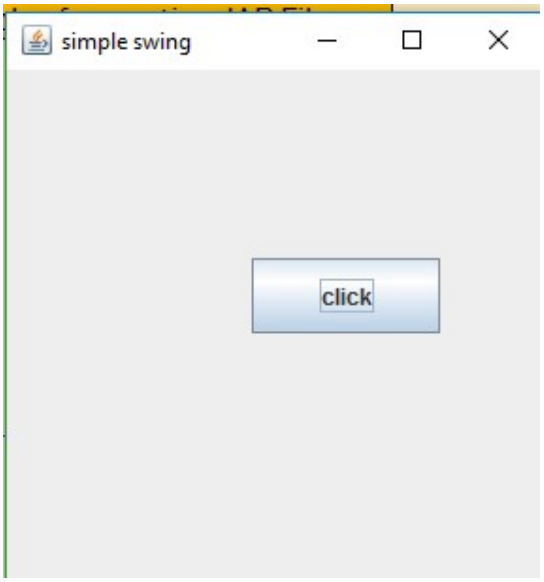
```
//First.java
import javax.swing.*;
public class First{
  First(){
    JFrame f=new JFrame();
    JButton b=new JButton("click");
    b.setBounds(130,100,100, 40);
    f.add(b);
    f.setSize(300,400);
    f.setLayout(null);
    f.setVisible(true);
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
public static void main(String[] args) {
   new First();
}
}
```

//contents of myfile.mf
Main-Class: First

Now executing:
D:\JavaProgs\myjar>javac First.java

D:\JavaProgs\myjar>jar -cvmf myfile.mf myjar.jar First.class
added manifest
adding: First.class(in = 736) (out= 506)(deflated 31%)
D:\JavaProgs\myjar>myjar.jar

Expected output:



Q8 (a): Describe the concept of deployment descriptors. List the deployment descriptor of EJB2.0. **5 M**

Definition : A *deployment descriptor* is a file that defines the following kinds of information: **EJB structural information,** such as the EJB name, class, home and remote interfaces, bean type (session or entity), environment entries, resource factory references, EJB references, security role references, as well as additional information based on the bean type. **Application assembly information,** such as EJB references, security roles, security role references, method permissions, and container transaction attributes. Specifying assembly descriptor information is an optional task that an Application Assembler performs.

Deployment descriptor of EJB 2.0: ejb-jar.xml & weblogic-ejb-jar.xml

Q8(b): Differentiate RMI & EJB. **5M**

Ans:

RMI	EJB
In RMI, middleware services such as security, transaction management, object pooling etc. need to be done by the java programmer.	In EJB, middleware services are provided by EJB Container automatically.

RMI is not a server-side component. It is not required to be deployed on the server.	EJB is a server-side component, it is required to be deployed on the server.
RMI is built on the top of socket programming.	EJB technology is built on the top of RMI.

**********Act like a person of thought, think like a person of action. **********