CMR
INSTITUTE OF
TECHNOLOGY

USN

Internal Assessment Test 2– May 2017

Sub:	Advanced Java Programming							Code:	13MCA 42
Date:	08.05.2017	*Duration:*	*90 mins*	*Max Marks:*	*50*	*Sem:*	*4*	*Branch:*	*MCA*

Answer any five of the following. 5x10=50M

1. Explain in detail, how to use java bean in JSP documents.

 A Java Bean is a software component that has been designed to be reusable in variety of different envi ronments.
 There is no restriction on the capability of a Bean. It may perform a simple function, such as spelling check of a document, or a complex function such as forecasting the performance of a stock portfolio.
 A Bean may be visible to an end user (ex. A button)
 A Bean may be invisible to a user (ex. Software to decode a stream of multimedia information in a real time)
 A Bean may be designed to work autonomously on a user's workstation or to work in cooperation with a set of other distributed components.
 JavaBeans makes it easy to reuse software components.
 Developers can use software components written by others without having to understand their inner workings.
 To understand why software components are useful, think of a worker assembling a car. Instead of building a radio from scratch, for example, she simply obtains a radio and hooks it up with the rest of the car.
A Java Bean is a java class that should follow following conventions:
It should have a no-arg constructor.
It should be Serializable.
It should provide methods to set and get the values of the properties, known as getter and setter methods.
According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object, so we can access this object from multiple places. Moreover, it provides the easy maintenance
Ex:

```
package mypack;
public class Employee implements java.io.Serializable{
private int id;
private String name;

public Employee(){}
public void setId(int id){this.id=id;}
public int getId(){return id;}
public void setName(String name){this.name=name;}
public String getName(){return name;}
  }
```

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

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

2.a. Explain the different type of JDBC drivers

• JDBC driver specification classifies JDBC drivers into four groups.
They are…

Type 1: JDBC-to-ODBC Driver

• Microsoft created ODBC (Open Database Connection), which is the basis from which Sun created JDBC. Both have similar driver specifications and an API.
• The JDBC-to-ODBC driver, also called the JDBC/ODBC Bridge, is used to translate DBMS calls between the JDBC specification and the ODBC specification.
• MS Access and SQL Server contains ODBC driver written in C language using pointers, but java does not support the mechanism to handle pointers.
• So JDBC-ODBC Driver is created as a bridge between the two so that JDBC-ODBC bridge driver translates the JDBC API to the ODBC API.

➔ Type-1 ODBC Driver for MS Access and SQL Server

Drawbacks of Type-I Driver:
○ ODBC binary code must be loaded on each client.
○ Transaction overhead between JDBC and ODBC.
○ It doesn‟t support all features of Java.
○ It works only under Microsoft, SUN operating systems.

Type 2: Java/Native Code Driver or Native-API Partly Java Driver

• It converts JDBC calls into calls on client API for DBMS.
• The driver directly communicates with database servers and therefore some database client software must be loaded on each client machine and limiting its usefulness for internet
• The Java/Native Code driver uses Java classes to generate platform- specific code that is code only understood by a specific DBMS.

Ex: Driver for DB2, Informix, Intersoly, Oracle Driver, WebLogic drivers

Drawbacks of Type-I Driver:
○ Some database client software must be loaded on each client machine
○ Loss of some portability of code.
○ Limited functionality
○ The API classes for the Java/Native Code driver probably won‟t work with another manufacturer‟s DBMS.

Type 3: Net-Protocol All-Java Driver

• It is completely implemented in java, hence it is called pure java driver. It translates the JDBC calls into vendor‟s specific protocol which is translated into DBMS protocol by a middleware server
• Also referred to as the Java Protocol, most commonly used JDBC driver.
• The Type 3 JDBC driver converts SQL queries into JDBC- formatted statements, in-turn they are translated into the format required by the DBMS.

Ex: Symantec DB
Drawbacks:
• It does not support all network protocols.
• Every time the net driver is based on other network protocols.

Type 4: Native-Protocol All-Java Driver or Pure Java Driver

- Type 4 JDBC driver is also known as the Type 4 database protocol.
- The driver is similar to Type 3 JDBC driver except SQL queries are translated into the format required by the DBMS.
- SQL queries do not need to be converted to JDBC-formatted systems.
- This is the fastest way to communicated SQL queries to the DBMS.
- Here the driver uses network protocol this protocol is already built-into the database engine; here the driver talks directly to the database using java sockets. This driver is better than all other drivers, because this driver supports all network protocols.
- Use Java networking libraries to talk directly to database engines

Ex: Oracle, MYSQL

Only disadvantage: need to download a new driver for each database engine

b. Explain about Batch Updates with example

Batch Processing allows you to group related SQL statements into a batch and submit them with one call to the database.
When you send several SQL statements to the database at once, you reduce the amount of communication overhead, thereby improving performance.
JDBC drivers are not required to support this feature. You should use the Database MetaData.supports BatchUpdates() method to determine if the target database supports batch update processing. The method returns true if your JDBC driver supports this feature.
The addBatch() method of Statement, PreparedStatement, and CallableStatement is used to add individual statements to the batch. The executeBatch() is used to start the execution of all the statements grouped together.
The executeBatch() returns an array of integers, and each element of the array represents the update count for the respective update statement.
Just as you can add statements to a batch for processing, you can remove them with the clearBatch() method. This method removes all the statements you added with the addBatch() method. However, you cannot selectively choose which statement to remove.
Here is a typical sequence of steps to use Batch Processing with Statement Object −
Create a Statement object using either createStatement() methods.
Set auto-commit to false using setAutoCommit().
Add as many as SQL statements you like into batch using addBatch() method on created statement object.
Execute all the SQL statements using executeBatch() method on created statement object.
Finally, commit all the changes using commit() method.

EX:

```
// Create statement object
Statement stmt = conn.createStatement();


// Set auto-commit to false
conn.setAutoCommit(false);


// Create SQL statement
String SQL = "INSERT INTO Employees (id, first, last, age) " +
             "VALUES(200,'Zia', 'Ali', 30)";
// Add above SQL statement in the batch.
```

```
stmt.addBatch(SQL);


// Create one more SQL statement
String SQL = "INSERT INTO Employees (id, first, last, age) " +
             "VALUES(201,'Raj', 'Kumar', 35)";
// Add above SQL statement in the batch.
stmt.addBatch(SQL);


// Create one more SQL statement
String SQL = "UPDATE Employees SET age = 35 " +
             "WHERE id = 100";
// Add above SQL statement in the batch.
stmt.addBatch(SQL);


// Create an int[] to hold returned values
int[] count = stmt.executeBatch();


//Explicitly commit statements to apply changes
conn.commit();
```

3.a. Write the short note about Prepared statement

 The preparedStatement object allows you to execute parameterized queries.
 A SQL query can be precompiled and executed by using the PreparedStatement object.
Ex: Select * from publishers where pub_id=?
  Here a query is created as usual, but a question mark is used as a placeholder for a value that is inserted into the query after the query is compiled.
 The preparedStatement() method of Connection object is called to return the PreparedStatement object.

Ex:
PreparedStatement stat;
stat= con.prepareStatement(“select * from publisher where pub_id=?”)

```
//Program using preparedstatement
import java.sql.*;

public class JdbcDemo {
   public static void main(String args[]){
     try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con=DriverManager.getConnection("jdbc:odbc:MyDataSource","khutub","");
        PreparedStatement pstmt;
        pstmt= con.prepareStatement("select * from employee whereUserName=?");
        pstmt.setString(1,"khutub");
        ResultSet rs1=pstmt.executeQuery();
        while(rs1.next()){
           System.out.println(rs1.getString(2));
        }
      } // end of try
      catch(Exception e){System.out.println("exception"); }
   } //end of main
   } // end of class
```

b.Write a JSP program to implement all the attributes of page directive tag.

student.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Student Information System</title>
<h4>Enter the details</h4>
</head>
<body>
<form action="process.jsp" method="post">
<table boder=1>
<tr><td>Usn No.</td><td><input type="text" name="usn"/></td></tr>
<tr><td>Student Name</td><td><input type="text" name="name"/></td></tr>
<tr><td>Department</td><td><input type="text" name="dept"/></td></tr>
</table>
<input type="submit" value="Submit"/>
<input type="reset" value="Clear"/>
</form>
</body>
</center>
</html>
```

process.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<% String name="",usn="",dept="";
usn=request.getParameter("usn");
name=request.getParameter("name");
```

```
dept=request.getParameter("dept");
out.println("<html><center><body bgcolor=grey>"); %>
<%@page errorPage="error.jsp" session="true" isThreadSafe="true" %>
<%synchronized(this)
{
wait(1000);
}
if(dept.equals("")||name.equals("")||usn.equals(""))
{
	throw new RuntimeException("FieldBlank");
}
else
{
	session.setAttribute("name",name);
	session.setAttribute("usn",usn);
session.setAttribute("dept",dept);
request.getRequestDispatcher("display.jsp").forward(request,response);
}
 %>
<%out.println("<body></center></html>");
 %>

</body>
</html>
```

error.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%@page isErrorPage="true"%>
<%=exception %>
</body>
</html>
```

display.jsp

```
<%@page import="java.util.*" session="true" contentType="text/html;"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
<h3 align="center"> Student information</h3>
<h4 align="right"><%= new Date() %></h4>
```

```
</head>
<center>
<body>
<table border=1 cellPadding=10 cellSpacing=10>
<tr>
<th>Name</th>
<th>USN</th>
<th>Dept</th>
</tr>
<tr>
<td><%=session.getAttribute("usn")%></td>
<td><%=session.getAttribute("name")%></td>
<td><%=session.getAttribute("dept")%></td>
</tr>
</table>
</body>
<a href="student.jsp"> Back to info</a>
</center>
</html>
```

4. Write a JAVA Program to insert data into Student DATA BASE and retrieve info based on particular queries(For example update, delete, search etc…).

```
package j2ee.p9;
import java.sql.*;
import java.io.*;

public class Studentdata {

        public static void main(String[] args) {
                Connection con;
                PreparedStatement pstmt;
                Statement stmt;
                ResultSet rs;
                String uname, pword;
                Integer marks,count;
                try
                {
                        Class.forName("com.mysql.jdbc.Driver"); // type1 driver

                        try{

        con=DriverManager.getConnection("jdbc:mysql://127.0.0.1/mca","root","system"); //  type1
access connection
                                BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
                          do
                          {



                                System.out.println("\n1. Insert.\n2. Select.\n3. Update.\n4. Delete.\n5.
Exit.\nEnter your choice:");
                                int choice=Integer.parseInt(br.readLine());
                                switch(choice)
                {
                    case 1: System.out.print("Enter UserName :");
                        uname=br.readLine();
                        System.out.print("Enter Password :");
                        pword=br.readLine();
                        pstmt=con.prepareStatement("insert into student
values(?,?)");
                        pstmt.setString(1,uname);
                        pstmt.setString(2,pword);
                        pstmt.execute();
                        System.out.println("\nRecord Inserted successfully.");
                    break;
                    case 2:
                        stmt=con.createStatement();
                        rs=stmt.executeQuery("select *from student");
                        if(rs.next())
                        {
                        System.out.println("User Name\tPassword\n-----------
-------------------");
                        do
                        {
                            uname=rs.getString(1);
                            pword=rs.getString(2);

                            System.out.println(uname+"\t"+pword);
                        }while(rs.next());
                        }
                        else
                            System.out.println("Record(s) are not
available in database.");
                        break;
                        case 3:
                            System.out.println("Enter User Name to
update :");
                            uname=br.readLine();
                            System.out.println("Enter new password :");
                            pword=br.readLine();
                            stmt=con.createStatement();
                            count=stmt.executeUpdate("update student set
password='"+pword+"'where username='"+uname+"'");
                            System.out.println("\n"+count+" Record
Updated.");
                        break;
                        case 4: System.out.println("Enter User Name to delete
record:");
                            uname=br.readLine();
                            stmt=con.createStatement();
                            count=stmt.executeUpdate("delete from
student where username='"+uname+"'");


                            if(count!=0)
                                System.out.println("\nRecord
"+uname+" has deleted.");
                            else
```

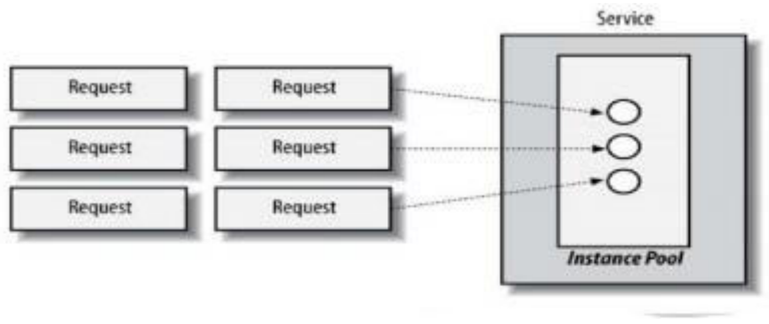
```
                                                System.out.println("\nInvalid USN,
Try again.");
                                        break;

                                        case 5: con.close(); System.exit(0);
                                        default: System.out.println("Invalid choice, Try
again.");
                                }//close of switch
                                }while(true);
                                }//close of nested try
                                catch(SQLException e2)
                                {
                                        System.out.println(e2);
                                }
                                catch(IOException e3)
                                {
                                        System.out.println(e3);
                                }
                        }//close of outer try
                        catch(ClassNotFoundException e1)
                        {
                                System.out.println(e1);
                        }
                }
}
```

5. Explain the container services provided by component model of EJB

1. Instance Pooling/Caching:

Because of the strict concurrency rules enforced by the Container, an intentional bottleneck is often introduced where a service instance may not be available for processing until some other request has completed. If the service was restricted to a singular instance, all subsequent requests would have to queue up until their turn was reached

EJB addresses this problem through a technique called instance pooling, in which each module is allocated some number of instances with which to serve incoming requests Many vendors provide configuration options to allocate pool sizes appropriate to the work being performed, providing the compromise needed to achieve optimal throughput.

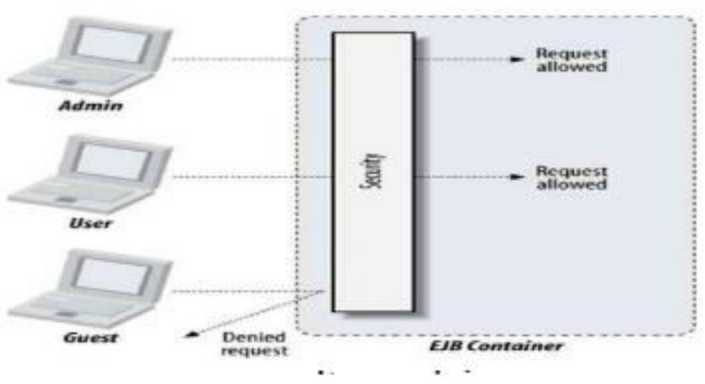


2.Transactions

Transactions provide a means for the developer to easily delegate the creation and control of transactions to the container. When a bean calls createTimer(), the operation is performed in the scope of the current• transaction. If the transaction rolls back, the timer is undone and it's not created The timeout callback method on beans should have a transaction attribute of RequiresNew.• This ensures that the work performed by the callback method is in the scope of containerinitiated

3.Security:

Most enterprise applications are designed to serve a large number of clients, and users are not necessarily equal in terms of their access rights. An administrator might require hooks into the configuration of the system, whereas unknown guests may be allowed a read-only view of data. If we group users into categories with defined roles, we can then allow or restrict access to the role itself, as illustrated in Figure 15-1.



This allows the application developer to explicitly allow or deny access at a fine-grained level based upon the caller's identity

4.Timers

We dealt exclusively with client-initiated requests. While this may handle the bulk of an application's requirements, it doesn't account for scheduled jobs: • A ticket purchasing system must release unclaimed tickets after some timeout of inactivity. • An auction house must end auctions on time. • A cellular provider should close and mail statements each month. The EJB Timer Service may be leveraged to trigger these events and has been enhanced in the 3.1 specification with a natural-language expression syntax.

6. Write a short note about
a)Implementation class

The contract, implemented as interfaces in Java, defines what our service will do, and leaves it up to the implementation classes to decide how it's done. Remember that the same interface cannot be used for both @Local and @Remote, so we'll make some common base that may be extended.

```
public interface CalculatorCommonBusiness {
 /** * Adds all arguments * *
 @return The sum of all arguments */
int add(int... arguments);
 }
public class CalculatorBeanBase implements CalculatorCommonBusiness {
 /** *
 {
@link CalculatorCommonBusiness#
add(int...)
}
 */ @Override
 public int add(final int... arguments)
{ // Initialize int result = 0;
// Add all arguments for (final int arg : arguments)
{
result += arg;
}
// Return return result;
}
```

```
 }
```
This contains the required implementation of CalculatorCommonBusiness. add(int...). The bean implementation class therefore has very little work to do.

```
 import javax.ejb.LocalBean;
 import javax.ejb.Stateless;
@Stateless
@LocalBean
public class SimpleCalculatorBean extends CalculatorBeanBase
 {
/* * Implementation supplied by common base class */
}
```
The function of our bean implementation class here is to bring everything together and define the EJB metadata. Compilation will embed two important bits into the resultant .class file. First, we have an SLSB, as noted by the @Stateless annotation. And second, we're exposing a no-interface view

b) Integration Testing of EJB

There are three steps involved in performing integration testing upon an EJB.
First, we must package the sources and any descriptors into a standard Java Archive
Next, the resultant deployable must be placed into the container according to a vendor-specific mechanism.
Finally, we need a standalone client to obtain the proxy references from the Container and invoke upon them.

Packaging
A standard jar tool that can be used to assemble classes, resources, and other metadata into a unified JAR file, which will both compress and encapsulate its contents.

Deployment into the Container The EJB Specification intentionally leaves the issue of deployment up to the vendor's discretion.

The client
Instead of creating POJOs via the new operator, we'll look up true EJB references via JNDI. JNDI is a simple store from which we may request objects keyed to some known address.

7. What is session bean? Explain in detail about the types of session bean

Session Beans If EJB is a grammar, session beans are the verbs.• Session beans contain business methods.• The client does not access the EJB directly, which allows the Container to perform all sorts of• magic before a request finally hits the target method. It's this separation that allows for the client to be completely unaware of the location of the server,• concurrency policies, or queuing of requests to manage resources.

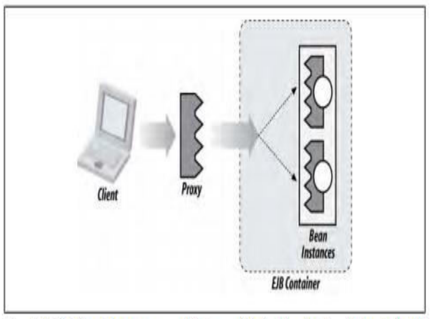


Figure 2-1. Client invoking upon a proxy object, responsible for delegating the call along to the EJB Container

Types of Session Bean
There are 3 types of session bean.
1) Stateless Session Bean: It doesn't maintain state of a client between multiple method calls.
2) Stateful Session Bean: It maintains state of a client across multiple requests.

3) Singleton Session Bean: One instance per application, it is shared between clients and supports concurrent access.

Stateless session beans (SLSBs) Stateless session beans are useful for functions in which state does not need to be carried from• invocation to invocation. The Container will often create and destroy instances however it feels will be most efficient• How a Container chooses the target instance is left to the vendor's discretion.• Because there's no rule linking an invocation to a particular target bean instance, these instances may be used interchangeably and shared by many clients. This allows the Container to hold a much smaller number of objects in service, hence keeping• memory footprint down.

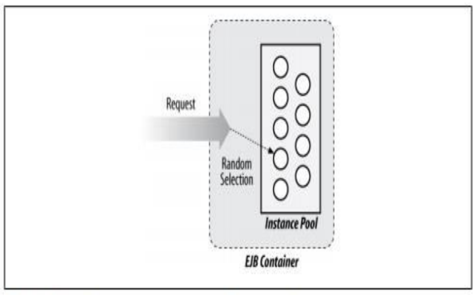


Figure 2-2. An SLSB Instance Selector picking an instance at random

Stateful session beans (SFSBs) Stateful session beans differ from SLSBs in that every request upon a given proxy reference is• guaranteed to ultimately invoke upon the same bean instance. SFSB invocations share conversational state.• Each SFSB proxy object has an isolated session context, so calls to one session will not affect• another. Stateful sessions, and their corresponding bean instances, are created sometime before the first• invocation upon a proxy is made to its target instance (Figure 2-3). They live until the client invokes a method that the bean provider has marked as a remove event,• or until the Container decides to remove the session

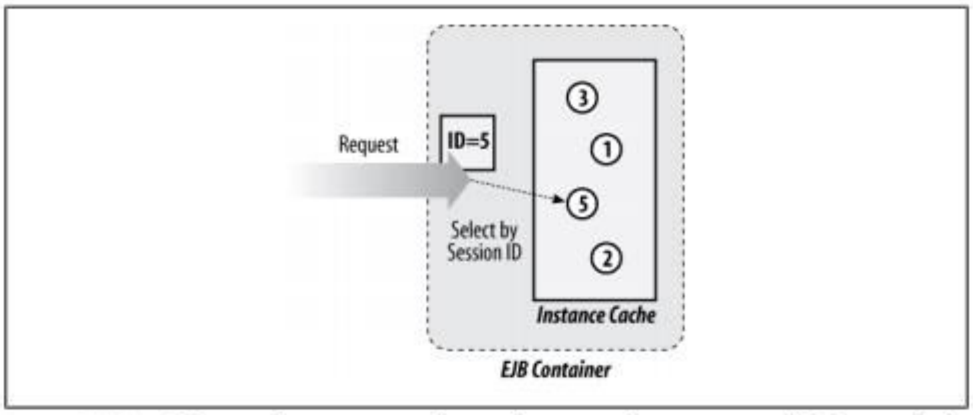


Figure 2-3. Stateful session bean creating and using the correct client instance, which lives inside the EJB Container, to carry out the invocation

Singleton beans Sometimes we don't need any more than one backing instance for our business objects.• All requests upon a singleton are destined for the same bean instance,• The Container doesn't have much work to do in choosing the target (Figure 2-4).• The singleton session bean may be marked to eagerly load when an application is deployed;• therefore, it may be leveraged to fire application lifecycle events. This draws a relationship where deploying a singleton bean implicitly leads to the invocation of its• lifecycle callbacks. We'll put this to good use when we discuss singleton beans.

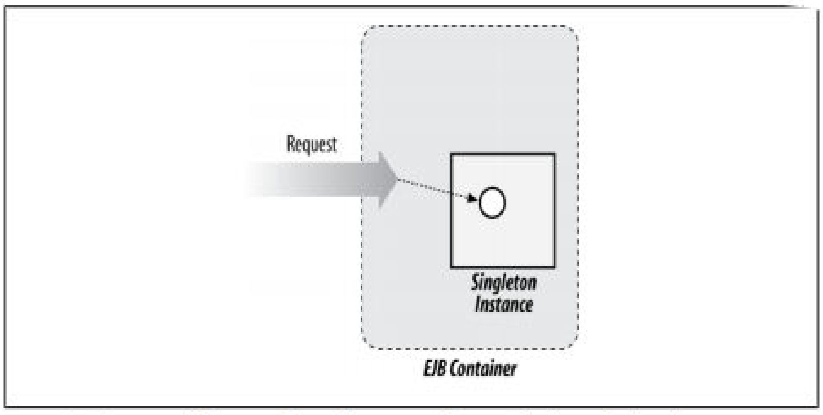


Figure 2-4. Conceptual diagram of a singleton session bean with only one backing bean instance

[illegible]