



22MCA204

USN

1	B	Y							
---	---	---	--	--	--	--	--	--	--

BMS INSTITUTE OF TECHNOLOGY AND MANAGEMENT

(An Autonomous Institute affiliated to Visvesvaraya Technological University, Belagavi)

SEMESTER END EXAMINATION QUESTION PAPER**Second Semester MCA Degree Examination**

Regular / Make-up / Arrears / Supplementary

JAVA PROGRAMMING

Time: 3 hrs.

Max. Marks: 100

Note: 1. Answer FIVE full questions, choosing ONE full question from each module.

Q. No	Module – 1	Marks	CO, RBT
1a.	What is method overriding? Write a program in java to illustrate it.	10	CO1, K2
b.	Illustrate the calling super class constructors and super class members using super.	10	CO1, K2
OR			
2a.	What are instance and static variables? Explain with a suitable program.	10	CO1, K2
b.	Define method overloading and constructor overloading? Can final methods be overridden .Justify Your answer.	10	CO1, K2
Module – 2			
3a.	What are interfaces? What are their benefits? Explain how it is implemented in JAVA with a suitable example.	10	CO1, K2
b.	What is an exception? Explain the exception handling mechanism with suitable example.	10	CO1, K2
OR			
4a.	Explain how to create custom exceptions with an example.	10	CO1, K2
b.	Write the following Java program to create package and import it in other program: i) Create a package called shape ii) Write a class called Triangle.java in shape package. Triangle.java should calculate the area of a triangle iii)Compile and import shape	10	CO1, K2
Module – 3			
5a.	What is multithreading? Write a JAVA program to create multithreads in JAVA by implementing runnable interface.	10	CO2, K2
b.	With a suitable programming example explain inter-thread communication.	10	CO2, K2
OR			
6a.	Explain wait (), notify (), notify All () in thread communication with example.	10	CO2, K2
b.	Design a Java program to create multiple threads by implementing runnable interface.	10	CO2, K2
Module – 4			
7a.	Define Servlet. Explain the basic servlet structure and its life cycle methods.	10	CO3, K2

b.	Write a JSP program to generate sum of even numbers between 1 to 30.	10	CO3, K2
OR			
8a.	Write JSP program to read data from a HTML form (gender data from radio buttons and colours data from check boxes) and display	10	CO3, K2
b.	Write a Java servlet program to illustrate GET and POST request. Mention any four difference between GET and POST method.	10	CO3, K2
Module – 5			
9a.	Explain about Stateful Session Bean.	10	CO4, K2
b.	Explain the different type of JDBC drivers.	10	CO4, K2
OR			
10a.	Explain the various steps of JDBC with code snippet.	10	CO4, K2
b.	Write a short note on Singleton bean.	10	CO4, K2

Course Outcomes (COs):

COs	At the end of the course, the student will be able to
CO-1	Demonstrate the basic programming constructs of Java and OOP concepts to develop Java applications.
CO-2	Illustrate the concepts of generalization and run time polymorphism to develop reusable components.
CO-3	Exemplify the usage of Multithreading in building efficient applications.
CO-4	Build web applications using Servlets and JSP.
CO-5	Design applications using JDBC and Enterprise Java Beans.
K1- Remembering K2 - Understanding K3 – Applying K4- Analyzing K5 - Evaluating K6 -Creating	

“Success is the progressive realization of a worthy goal.”

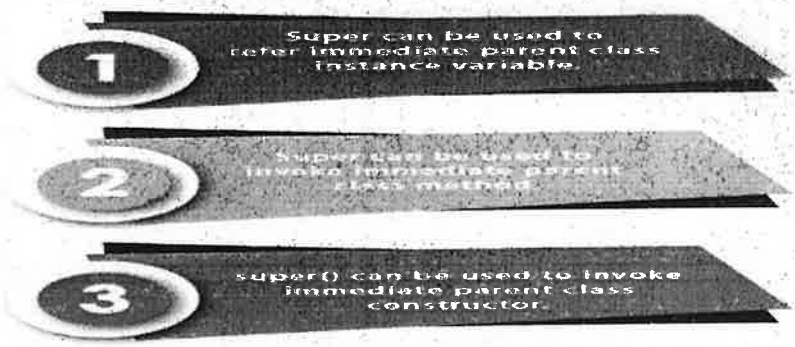
DESCRIPTIVE QUESTIONS PATTERN

COURSE CODE:- MCA204 COURSE NAME JAVA PROGRAMMING

No	Scheme and Solutions	Marks
	MODULE-1	
1a	<p>If a sub class has the same method as declared in the parent class then it is called as method overriding.</p> <p>Usage: It is used to provide specific implementation of a method that is already provided by its super class used for run time polymorphism.</p> <p>Rules: 1. Method must have same name and parameters as in parent class. 2. Method must be having is-a relation ship.</p> <p>Ex:</p> <pre> class A { int i,j; A(int a, int b) { i=a; j=b; } void show() { System.out.println("i and j value"+i+" "+j); } } Class B extends A { int k; B(int a, int b,int c) { Super(a,b); k=c; } void show() { System.out.println("k value"+k); } } Class Override { Public static void main(String[] k) { B subob=new B(1,2,3); Subob.show(); } } </pre>	<p>54</p> <p>6</p>

DESCRIPTIVE QUESTIONS PATTERN

COURSE CODE:- MCA204 COURSE NAME JAVA PROGRAMMING

Q.No	Scheme and Solutions	Marks
1b	<p>The super keyword refers to superclass (parent) objects. The most common use of the super keyword is to eliminate the confusion between super classes and subclasses that have methods with the same name</p> <p style="text-align: center;">Usage of Super Keyword</p>  <p>1) super is used to refer immediate parent class instance variable. We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.</p> <pre> 1. class Animal{ 2. String color="white"; 3. } 4. class Dog extends Animal{ 5. String color="black"; 6. void printColor(){ 7. System.out.println(color);//prints color of Dog class 8. System.out.println(super.color);//prints color of Animal class 9. } 10. } 11. class TestSuper1{ 12. public static void main(String args[]){ 13. Dog d=new Dog(); 14. d.printColor(); 15. }}</pre> <p>2) super can be used to invoke parent class method The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.</p> <pre> 1. class Animal{ 2. void eat(){System.out.println("eating...");} 3. } 4. class Dog extends Animal{ 5. void eat(){System.out.println("eating bread...");} 6. void bark(){System.out.println("barking...");} 7. void work(){</pre>	<p>2</p> <p>3</p> <p>5</p>

DESCRIPTIVE QUESTIONS PATTERN

COURSE CODE:- MCA204 COURSE NAME JAVA PROGRAMMING

Q. No	Scheme and Solutions	Marks
2a	<pre> 11. } 12. class TestSuper2{ 13. public static void main(String args[]){ 14. Dog d=new Dog(); 15. d.work(); 16. }}</pre> <p>3) super is used to invoke parent class constructor. The super keyword can also be used to invoke the parent class constructor. Let's see a simple example:</p> <pre> 1. class Animal{ 2. Animal(){System.out.println("animal is created");} 3. } 4. class Dog extends Animal{ 5. Dog(){ 6. super(); 7. System.out.println("dog is created"); 8. } 9. } 10. class TestSuper3{ 11. public static void main(String args[]){ 12. Dog d=new Dog(); 13. }}</pre> <p style="text-align: center;">OR</p> <p>Class variables also known as static variables are declared with the <i>static</i> keyword in a class, but outside a method, constructor or a block. There would only be one copy of each class variable per class, regardless of how many objects are created from it. Static variables are rarely used other than being declared as constants. Constants are variables that are declared as <i>public/private, final and static</i>. Constant variables never change from their initial value. Static variables are stored in static memory. It is rare to use static variables other than declared final and used as either public or private constants. Static variables are created when the program starts and destroyed when the program stops. Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class. Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned during the declaration or within the constructor. Additionally values can be assigned in special static initializer blocks. Static variables can be accessed by calling with the class name as <i>ClassName.VariableName</i>. When declaring class variables as public static final, then variables names (constants) are all in upper case. If the static variables are not public and final the naming syntax is the same as instance and local variables.</p> <p>Instance Variables:</p> <ul style="list-style-type: none"> • Instance variables are declared in a class, but outside a method, constructor or any block. • When a space is allocated for an object in the heap, a slot for each instance variable value is created. • Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed. • Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class. • Instance variables can be declared in class level before or after use. • Access modifiers can be given for instance variables. • The instance variables are visible for all methods, constructors and block in the class. Normally, it is 	<p>2</p> <p>3</p>

recommended to make these variables private (access level). However visibility for subclasses can be given for these variables with the use of access modifiers.

- Instance variables have default values. For numbers the default value is 0, for Booleans it is false and for object

references it is null. Values can be assigned during the declaration or within the constructor.

- Instance variables can be accessed directly by calling the variable name inside the class. However within static methods

and different class (when instance variables are given accessibility) should be called using the fully qualified name

.ObjectReference.VariableName

Ex:

Class Sample

```
{
int i,n;
static int count=0;
Sample(int a)
{
n=a;
}
```

Void even()

```
{
for(i=2;i<=n;i++)
{
If(i%2==0)
Count++;
}
}
```

Class Demo

```
{
Public static void main(String k[])
{
Sample s=new Sample(50);
s.even();
System.out.println(Sample.count+even numbers are present upto"+s.n);
}
}
```

2b

If a class have multiple methods by same name but different parameters, it is known as Method Overloading.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Method overloading increases the readability of the program.

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

class Calculation

```
{
void sum(int a,int b)
{
System.out.println(a+b);
}
void sum(int a,int b,int c)
{
System.out.println(a+b+c);
}
}
```

Class MethodOverload

```
{
public static void main(String args[])
{
Calculation obj=new Calculation();
obj.sum(10,10,10);
obj.sum(20,20);
}
```

5

2

3

DESCRIPTIVE QUESTIONS PATTERN

```

}
}
Constructor Overloading
Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.
EX:
class Student{
int id;
String name;
int age;
Student(int i,String n)
{
id = i;
name = n;
}
Student(int i,String n,int a)
{
id = i;
name = n;
age=a;
}
void display()
{
System.out.println(id+" "+name+" "+age);
}
public static void main(String args[]){
Student s1 = new Student(111,"Karan");
Student s2 = new Student(222,"Aryan",25);
s1.display();
s2.display();
}
}

```

5

MODULE-3

3a

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface. Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types.

Method bodies exist only for default methods and static methods. Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements. Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class. An interface is similar to a class in the following ways –

5

```

interface Animal
{
public void eat();
public void travel(); }
Implementing Interfaces

```

When a class implements an interface, you can think of the class as signing a contract, agreeing to perform the specific behaviors of the interface. If a class does not perform all the behaviors of the interface, the class must declare itself as abstract. A class uses the implements keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

Example

```

public class MammalInt implements Animal {
public void eat() {

```

5

```

System.out.println("Mammal eats");
}
public void travel() {
System.out.println("Mammal travels");
}
public int noOfLegs() {
return 0;
}
public static void main(String args[]) {
MammalInt m = new MammalInt();
m.eat();
m.travel();
}
}

```

This will produce the following result –

Output

Mammal eats

Mammal travels

3b

When overriding methods defined in interfaces, there are several rules to be followed –

A Java exception is an object that describes an exceptional (that is, error) condition that has occurred in a piece of code. When an exceptional condition arises, an object representing that exception is created and thrown in the method that caused the error. That method may choose to handle the exception itself, or pass it on. Either way, at some point, the exception is caught and processed. Exceptions can be generated by the Java run-time system, or they can be manually generated by your code. Java exception handling is managed via five keywords: try, catch, throw, throws, and finally. Program statements that you want to monitor for exceptions are contained within a try block. If an exception occurs within the try block, it is thrown. Your code can catch this exception (using catch) and handle it in some rational manner. System-generated exceptions are automatically thrown by the Java run-time system. To manually throw an exception, use the keyword throw. Any exception that is thrown out of a method must be specified as such by a throws clause. Any code that absolutely must be executed after a try block completes is put in a finally block.

This is the general form of an exception-handling block:

```
try {
```

```
// block of code to monitor for errors
```

```
}
```

```
catch (ExceptionType1 exOb) {
```

```
// exception handler for ExceptionType1
```

```
}
```

```
catch (ExceptionType2 exOb) {
```

```
// exception handler for ExceptionType2
```

```
}
```

```
// ...
```

```
finally {
```

```
// block of code to be executed after try block ends
```

```
}
```

6

4

DESCRIPTIVE QUESTIONS PATTERN

<p>4a</p>	<p style="text-align: center;">OR</p> <p>User-defined exceptions in Java are also known as Custom Exceptions.</p> <p>Steps to create a Custom Exception with an Example</p> <p>CustomException class is the custom exception class this class is extending Exception class.</p> <p>Create one local variable message to store the exception message locally in the class object.</p> <p>We are passing a string argument to the constructor of the custom exception object. The constructor set the argument string to the private string message.</p> <p>toString() method is used to print out the exception message.</p> <p>We are simply throwing a CustomException using one try-catch block in the main method and observe how the string is passed while creating a custom exception. Inside the catch block, we are printing out the message.</p> <p>Example</p> <pre>class CustomException extends Exception { String message; CustomException(String str) { message = str; } public String toString() { return ("Custom Exception Occurred : " + message); } } public class MainException { public static void main(String args[]) { try { throw new CustomException("This is a custom message"); } catch(CustomException e) { System.out.println(e); } } }</pre> <p>Output</p> <p>Custom Exception Occurred : This is a custom message</p>	<p style="text-align: center;">4</p> <p style="text-align: center;">6</p>
<p>4b</p>	<pre>package shape; public class triangle { public void display3() { System.out.println("formula of triangle =0.5*length*breadth"); } } package shape; public class square { public void display2() { System.out.println("formula of Square =length*width") } } package shape; public class circle { public void display1()</pre>	<p style="text-align: center;">5</p>

```

{
System.out.println("formula of circle is 3.142*r*r");
}
}
import shape.*;
public class prgrm7
{
public static void main(String arg[])
{
circle ob1 = new circle();
square ob2 =new square();
triangle ob3 =new triangle();
ob1.display1();
ob2.display2();
ob3.display3();
}
}

```

5

MODULE-3

5a

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. Thus, multithreading is a specialized form of multitasking.

Implementing Runnable: The easiest way to create a thread is to create a class that implements the Runnable interface. Runnable abstracts a unit of executable code. You can construct a thread on any object that implements Runnable. To implement Runnable, a class need only implement a single method called run(), which is declared like this:

```

public void run( )

```

Inside run(), you will define the code that constitutes the new thread. It is important to understand that run() can call other methods, use other classes, and declare variables, just like the main thread can. The only difference is that run() establishes the entry point for another, concurrent thread of execution within your program. This thread will end when run() returns. After you create a class that implements Runnable, you will instantiate an object of type Thread from within that class. Thread defines several constructors. The one that we will use is shown here:

```

Thread(Runnable threadOb, String threadName)

```

In this constructor, threadOb is an instance of a class that implements the Runnable interface. This defines where execution of the thread will begin. The name of the new thread is specified by threadName. After the new thread is created, it will not start running until you call its start() method, which is declared within Thread. In essence, start() executes a call to run().

The start() method is shown here:

```

void start( )

```

Here is an example that creates a new thread and starts it running:

```

// Create a second thread.
class NewThread implements Runnable {
Thread t;
NewThread() {
// Create a new, second thread
t = new Thread(this, "Demo Thread");
System.out.println("Child thread: " + t);
t.start(); // Start the thread
}
}

```

5

5

DESCRIPTIVE QUESTIONS PATTERN

```
// This is the entry point for the second thread.
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[]) {
new NewThread(); // create a new thread
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.");
}
}
```

The output produced by this program is as follows

Child thread: Thread[Demo Thread,5,main]

Main Thread: 5

Child Thread: 5

Child Thread: 4

Main Thread: 4

Child Thread: 3

Child Thread: 2

Main Thread: 3

Child Thread: 1

Exiting child thread.

Main Thread: 2

Main Thread: 1

Main thread exiting.

5b

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 is achieved is called synchronization. Key to synchronization is the concept of the monitor (also called a semaphore). A monitor is an object that is used as a mutually exclusive lock, or mutex. Only one thread can own a monitor at a given time. When a thread acquires a lock, it is said to have entered the monitor. All other threads attempting to enter the locked monitor will be suspended until the first thread exits the monitor.

These other threads are said to be waiting for the monitor. A thread that owns a monitor can reenter the same monitor if it so desires. You can synchronize your code in two ways.

- Using Synchronized Methods
- The synchronized Statement

While creating synchronized methods within classes that you create is an easy and

effective means of achieving synchronization, it will not work in all cases. To understand why, consider the following. Imagine that you want to synchronize access to objects of a class that was not designed for multithreaded access. That is, the class does not use synchronized methods. Further, this class was not created by you, but by a third party, and you do not have access to the source code. Thus, you can't add synchronized to the appropriate methods within the class. How can access to an object of this class be synchronized? Fortunately, the solution to this problem is quite easy:

You simply put calls to the methods defined by this class inside a synchronized block.

This is the general form of the synchronized statement:

```
synchronized(object) {  
    // statements to be synchronized  
}
```

Here, object is a reference to the object being synchronized. A synchronized block ensures that a call to a method that is a member of object occurs only after the current thread has successfully entered object's monitor. Here is an alternative version of the preceding example, using a synchronized block within the run() method:

// This program uses a synchronized block.

```
class Callme {  
    void call(String msg) {  
        System.out.print "[" + msg);  
        try {  
            Thread.sleep(1000);  
        } catch (InterruptedException e) {  
            System.out.println("Interrupted");  
        }  
        System.out.println("]");  
    }  
}  
  
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();  
    }  
    // synchronize calls to call()  
    public void run() {  
        synchronized(target) { // synchronized block  
            target.call(msg);  
        }  
    }  
}  
  
class Synch1 {  
    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");  
        // wait for threads to end  
        try {
```

DESCRIPTIVE QUESTIONS PATTERN

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

Here, the call() method is not modified by synchronized. Instead, the synchronized statement is used inside Caller's run()

method. Output

[Hello]

[Synchronized]

[World]

6a

OR

Inter thread communication or Cooperation is all about allowing synchronized threads

to communicate with each other.

Cooperation (Inter thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same

critical section to be executed. It is implemented by following methods of Object class:

wait()

notify()

notifyAll()

1) wait() method

Causes current thread to release the lock and wait until either another thread invokes the

notify(notify()) method or the notifyAll() method for this object, or a specified amount of time has elapsed.

The current thread must own this object's monitor, so it must be called from the synchronized

method only otherwise it will throw exception.

method only otherwise it will throw exception. Method

public final void wait() throws
InterruptedException

public final void wait(long
timeout) throws InterruptedException

Description

waits until object is notified.

waits for the specified amount of time.

2) notify() method

Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. Syntax:

public final void notify()

3) notifyAll() method

Wakes up all threads that are waiting on this object's monitor. Syntax:

public final void notifyAll()

6b

6

4

5

Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread scheduler schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

3 constants defined in Thread class:

```
public static int MIN_PRIORITY
public static int NORM_PRIORITY
public static int MAX_PRIORITY
```

Default priority of a thread is 5 (NORM_PRIORITY).

The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

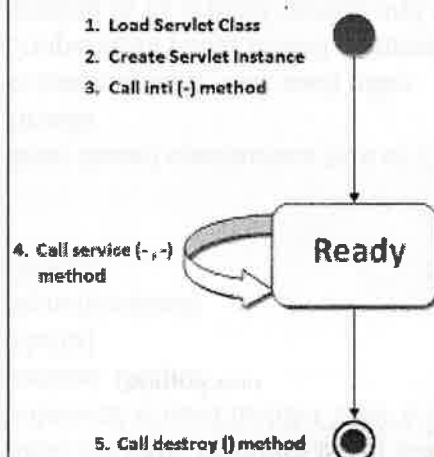
Example of priority of a Thread:

```
class TestMultiPriority1 extends Thread{
public void run(){
System.out.println("running thread name is:"+Thread.currentThread().getName());
System.out.println("running thread priority
is:"+Thread.currentThread().getPriority());
public static void main(String args[]){
TestMultiPriority1 m1=new TestMultiPriority1();
TestMultiPriority1 m2=new TestMultiPriority1();
m1.setPriority(Thread.MIN_PRIORITY);
m2.setPriority(Thread.MAX_PRIORITY);
m1.start();
m2.start();
```

7a

Module – 4

Java Servlets are server-side programs to handle client requests to a web application through the browser, and respond. Life Cycle of Servlet consists mainly of three methods - init(), service(), and destroy(). HTTP requests from clients/browsers can be read. This includes cookies, parameters, and session handling.



DESCRIPTIVE QUESTIONS P A T T E R N

COURSE CODE:- MCA204 COURSE NAME JAVA PROGRAMMING

Q. No	Scheme and Solutions	Marks
7b	<p>It contains links to additional resources needed by the requested page. The client discovers that the full rendering of the page requires these additional resources from the server only after it downloads the page.</p> <p>Header Description</p> <p>Accept-CH It is a response-type header. It specify which Client Hints headers client should include in subsequent requests.</p> <p>Save-Data It is used to reduce the usage of the data on the client side.</p> <p>Viewport-Width It is used to indicates the layout viewport width in CSS pixels.</p>	5
8a	<p style="text-align: center;">OR</p> <pre> <!DOCTYPE html> <html> <head> <meta name="viewport" content="width=device-width, initial-scale=1"> <link rel="stylesheet" href="https://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css"> <script src="https://code.jquery.com/jquery-1.11.3.min.js"></script> <script src="https://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js"></script> </head> <body> <div data-role="page"> <div data-role="header"> <h1>Radio Buttons and Checkboxes</h1> </div> <div data-role="main" class="ui-content"> <form method="post" action="/action_page_post.php"> <fieldset data-role="controlgroup"> <legend>Choose your gender:</legend> <label for="male">Male</label> <input type="radio" name="gender" id="male" value="male" checked> <label for="female">Female</label> <input type="radio" name="gender" id="female" value="female"> </fieldset> <fieldset data-role="controlgroup"> <legend>Choose as many favorite colors as you'd like:</legend> <label for="red">Red</label> <input type="checkbox" name="favcolor" id="red" value="red" checked> <label for="green">Green</label> <input type="checkbox" name="favcolor" id="green" value="green"> <label for="blue">Blue</label> <input type="checkbox" name="favcolor" id="blue" value="blue" checked> </fieldset> </form> </div> </div> </pre>	7

```

</fieldset>
<input type="submit" data-inline="true" value="Submit">
</form>
</div>
</div>

</body>
</html>

```

Choose your gender:

Male

Female

Choose as many favorite colors as you'd like:

Red

Green

Blue

The <jsp:include> Action

This action lets you insert files into the page being generated. The syntax looks like this –

```
<jsp:include page = "relative URL" flush = "true" />
```

Unlike the include directive, which inserts the file at the time the JSP page is translated into a servlet, this action inserts the file at the time the page is requested.

Following table lists out the attributes associated with the include action

The <jsp:plugin> Action

The plugin action is used to insert Java components into a JSP page. It determines the type of browser and inserts the <object> or <embed> tags as needed.

If the needed plugin is not present, it downloads the plugin and then executes the Java component. The Java component can be either an Applet or a JavaBean.

The plugin action has several attributes that correspond to common HTML tags used to format Java components. The <param> element can also be used to send parameters to the Applet or Bean.

Following is the typical syntax of using the plugin action –

Module – 5

Stateful session beans (SFSBs) Stateful session beans differ from SLSBs in that every request upon a given proxy reference is guaranteed to ultimately

9a

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. 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.

4

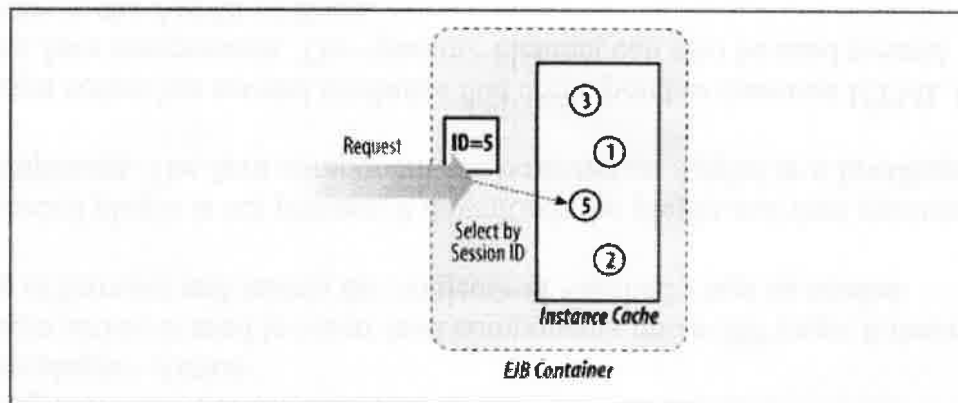


Figure 2-3. Stateful session bean creating and using the correct client instance, which lives inside the EJB Container, to carry out the invocation

6

9b

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.

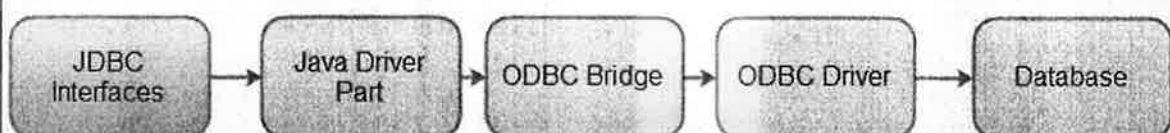
So JDBC-ODBC Driver is created as a bridge between the two so that JDBC-ODBC bridge driver translates the JDBC API to the ODBC API.

Type-1 ODBC Driver for MS Access and SQL Server

Drawbacks of Type-I Driver:

- o ODBC binary code must be loaded on each client.
- o Transaction overhead between JDBC and ODBC.
- o It doesn't support all features of Java.
- o It works only under Microsoft, SUN operating systems.

5



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

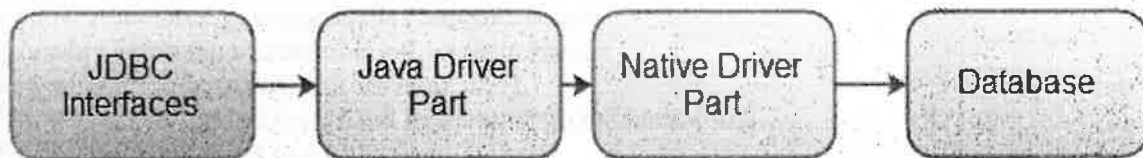
It converts JDBC calls into calls on client API for DBMS. The driver directly communicates with database servers and therefore some database

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, Intersol, Oracle Driver, WebLogic drivers

Drawbacks of Type-I Driver:

- o Some database client software must be loaded on each client machine
- o Loss of some portability of code:
- o Limited functionality
- o The API classes for the Java/Native Code driver probably won't work with another manufacturer's DBMS.



5

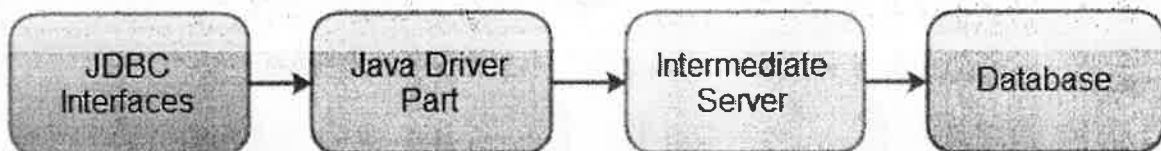
Type 3: Net-Protocol All-Java Driver It is completely implemented in java, hence it is called pure java driver. It translates the

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

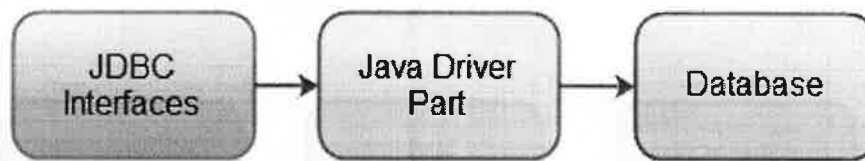
Drawbacks: It does not support all network protocols. Every time the net driver is based on other network protocols.



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 communicate 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



OR

10a

The following 5 steps are the basic steps involve in connecting a Java application with Database using JDBC.

Register the Driver

Create a Connection

Create SQL Statement

Execute SQL Statement

Closing the connection

Register the Driver Class.forName() is used to load the driver class explicitly.

Example to register with JDBC-ODBC Driver

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

Create a Connection getConnection() method of DriverManager class is used to create a connection.

Syntax

getConnection(String url)

getConnection(String url, String username, String password)

getConnection(String url, Properties info)

Example establish connection with Oracle Driver

Connection con = DriverManager.getConnection

("jdbc:oracle:thin:@localhost:1521:XE","username","password");

Create SQL Statement createStatement() method is invoked on current Connection object to create a SQL Statement.

Syntax

public Statement createStatement() throws SQLException

Example to create a SQL statement

Statement s=con.createStatement();

Execute SQL Statement executeQuery() method of Statement interface is used to execute SQL statements.

Syntax

public ResultSet executeQuery(String query) throws SQLException

Example to execute a SQL statement

ResultSet rs=s.executeQuery("select * from user");

while(rs.next())

```

{
System.out.println(rs.getString(1)+" "+rs.getString(2));
}
  
```

Closing the connection

After executing SQL statement you need to close the connection and release the session. The close() method of Connection interface is used to close the connection.

Syntax

public void close() throws SQLException

6

Example of closing a connection

```
con.close();
import java.sql.*;
class OracleCon{
public static void main(String args[]){
try{
//step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");
//step2 create the connection object
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
//step3 create the statement object
Statement stmt=con.createStatement();
//step4 execute query
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
//step5 close the connection object
con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

4

10b

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.

5

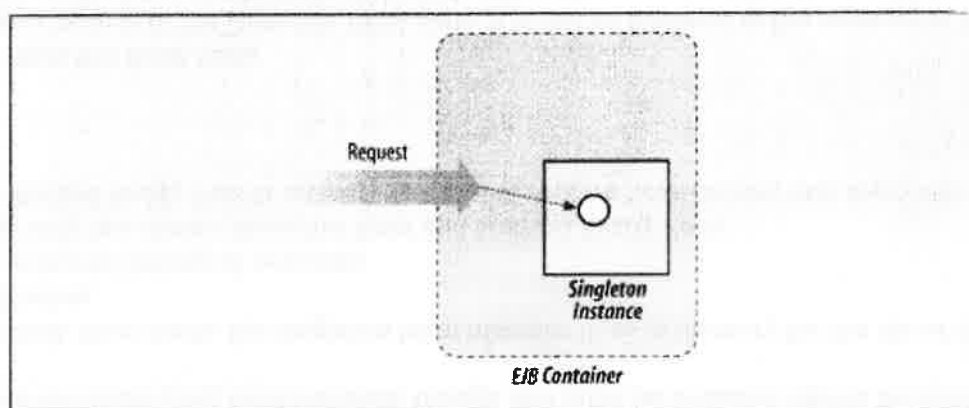


Figure 2-4. Conceptual diagram of a singleton session bean with only one backing bean instance

LifeCycle:

The life of a singleton bean is very similar to that of the stateless session bean; it is either not yet instantiated or ready to service requests. In general, it is up to the Container to determine when to create the underlying bean instance, though this must be available before the first invocation is executed. Once made, the singleton bean instance lives in memory for the life of the application and is shared among all requests. It has only two states: Does Not Exist and Method Ready Pool. The Method Ready Pool is an instance pool of session bean objects that are not in

The Does Not Exist State

When a bean is in the Does Not Exist state, it is not an instance in the memory of the system. In other words, it has not been instantiated yet.