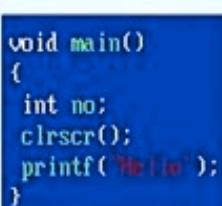





**College
Projects**


**Basic
Program**

**C
Tutorial**


**JAVA
Programming**



Tutorial4Us

Tutorials for Beginners



Tutorial4Us

Tutorials for Beginners



Tutorial4Us

Tutorials for Beginners

tutorial4us.com

A Perfect Place for All Tutorials Resources

Core Java | Servlet | JSP | JDBC | Struts | Hibernate | Spring

Java Projects | C | C++ | DS | Interview Questions | JavaScript

College Projects | eBooks | Interview Tips | Forums | Java Discussions

For More Tutorials Stuff Visit

www.tutorial4us.com

A Perfect Place for All Tutorials Resources

Design Patterns

(Natraj Notes)

www.tutorial4us.com

Design pattern :-

Which comes as test solution for real time problems of application development.

Design patterns are best practices to use slow technologies effectively in project of application development.

→ Design pattern are Supporting code while developing slow projects by using plain technologies.

→ The worst solution for ~~real~~ time problem of project or application development is called as "Anti-pattern".

→ The organization ISO maintains both design pattern and anti-pattern.

→ Design patterns are implementation takes place in coding phase of project development.

Generally PL/TL designs the task for developers specified the need and utilization of

Design patterns and developers will implement them while doing Task completion.

→ There are 500+ design patterns in java Environment.

→ Design patterns can be implemented by using many slow technology. (or) programming Language
→ Jdk level design patterns :-

Singleton java class

Synchronized Singleton java class

Factory method / Factory pattern

Abstract Factory

Template method

Builder pattern

prototype pattern

Flyweight pattern

IOC (Inversion of control) pattern

Adapter class

Fastline reader

~~Business object / value class object~~

V.O class / D.T.O class object
object

→ Weblevel design patterns :-

value object class / date transfer object class

View Helper

Composite view

MVC1 MVC3

MVC2

Front Controller

Intercepting ~~Filter~~ ~~Interceptor~~

Abstract Controller

- Integration Layer Design patterns :-

Session Facade

Message Facade

Service Locator

Business Delegate

→ senior developers designs solutions for recurring problems of technology utilization whereas the junior developers take best solutions out of them and uses that as designed patterns.

- Model Layer :-

DAO (Data Access Obj).

Abstract DAO

① Singleton Java class

problem :- Instead of creating multiple objects of Java class having same data and wasting memory, degrading performance. It is recommended to create one object and use it for multiple times.

solution :- Use Singleton java class.

The java class ^{that} allows to create one object per jvm is called as Singleton java class.

→ The logger class of the Log4j API is given as Singleton java class.

→ The Service locator class will be implemented as Singleton java class.

Rules to implement Singleton java class :-

(1) it must have only private constructors.

(2) must have ^{private} static reference variable of same class

(3) override clone() to suppress cloning process.

(4) must have ^{public} static factory method having the logic of Singleton. (Create and return only one object).

All these rules close all the doors of creating objects for java class and opens only one door to create object (i.e. Factory method where Singleton logic is placed).

→ The method of a class i.e. capable of creating and returning same or other class objects is called as Factory method

// SingletonTest.java

```
class STest
```

```
{
```

```
    // static reference variable
```

```
    private static STest n=null;
```

```
    // private constructor
```

```
    private STest()
```

```
{
```

```
    S.O.P("STest : 0-param Constructor(private)");
```

```
}
```

```
// static factory method
```

```
public static STest create()
```

```
{
```

```
    // object creation having Singleton logic
```

```
    if(n==null)
```

```
        n=new STest();
```

```
    }  
    return n;
```

```
} // class
```

```
public class SingletonTest
```

```
{
```

```
    public void main (String[] args)
```

```
{
```

```
        STest t1=STest.create();
```

```
        t2 =
```

```
        t3 =
```

```
        S.O.P("t1 obj hashCode is "+t1.hashCode());
```

```
        S.O.P("t2 " + " " + t2.hashCode());
```

```
        S.O.P("t3 " + " " + t3.hashCode());
```

```
        STest t4=(STest)t3.clone();
```

```
        S.O.P("t4 obj " + " " + t4.hashCode());
```

```
}
```

```
> javac SingletonTest.java
```

```
> java SingletonTest.
```

→ java.lang.Runtime class is a pre-defined JDK level Singleton Java class.

② Synchronized Singleton Java class :-

When multiple threads acts on single object then there is a possibility of getting multithreading issues. This chance is also there while working with the methods on Singleton Java class. To overcome this problem it is recommended to design Synchronized Singleton Java class. That means the static factory method and other user defined methods of that class must be taken as synchronized methods.

③ Factory method / Factory pattern :-

problem:- Using "new" keyword we can't create object with flexibility and by applying restrictions

solution:- Use Factory method / Factory pattern

The method of a Java class i.e. capable of constructing and returning its own class(or) other class object is called as "Factory method".

There are two types of factory method.

- 1) Static factory method
- 2) Instance factory method.

Examples on static factory methods :-

Thread t = Thread.currentThread();

Class c = Class.forName("Test");

Runtime rt = Runtime.getRuntime();

Calendar cl = Calendar.getInstance();

↳ It is not calendar class object. It is object of gregorianCalendar class which is subclass of Calendar class (abstract class).

→ static factory methods are useful to create object of Java classes outside of its class, that class contains only private constructors. These are also useful while designing Singleton Java classes.

prototype of static factory method :-

public static <class name / abstract Classname> interface name > method (param name)

↳ It is recommended

Examples on instance factory method :-

String s = new String("ok");

String s1 = s.concat("hello"); // okhello

StringBuffer sb = new StringBuffer ("hello");

method of String class returns String class obj

String s2 = sb.substring (0,3); // hell

Date d = new Date();

String s3 = d.toString();

→ If u want to create 'new' object by using existing obj data and behaviour then use Instance

factory method.

→ factory method can return either its own class object (or) other class object (or) subclass object

→ The object given by factory method represents certain process / logic i.e. executed in factory method

→ better not to ~~use~~ design factory methods returning empty objects (objects containing default data)

Application:-

For example code on Factory pattern / Factory method. refer page no 1& 2

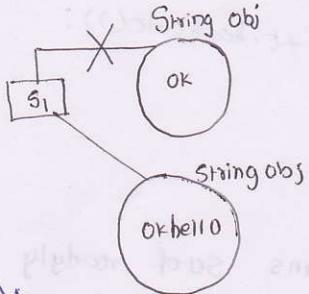
Prototype of InstanceFactory method :-

public <classname> <interface name> / <abstract classname> method (-)

String s1 = new String ("ok");

String s1 = s1.concat ("hello"); // okhello

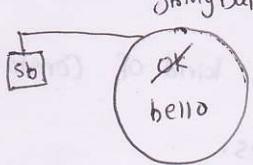
(String class is immutable class)



StringBuffer sb = new StringBuffer ("ok");

sb.append ("hello");

(StringBuffer class is Mutable class)



most of the predefined classes of programmer developed java classes are generally mutable classes.

1) Test.java

```
class Test
{
    int a;
    float b;

    Test(int a, int b)
    {
        this.a = a;
        this.b = b;
    }
}
```

~~Method~~ modify(~~data~~)

```
public void modifyData()
{
    a = a + a;
    b = b + b;
}

public void disp()
{
    System.out.println("a=" + a + "b=" + b);
}
```

2) class

```
public class MutableTest
```

```
{ public static void main(String[] args)
```

```
{
```

```
    Test t = new Test(10, 20);
```

```
    t.disp(); // object hashCode: 111. hashCode()
```

```
    t.modifyData();
```

```
    t.disp(); // object hashCode: 111. hashCode()
```

```
}
```

3) class

01/09/2012

4) String Constant Pool :-

→ String Constant pool maintains set of readyly available objects. Due to this objects availability new objects will not be created they will be accessed from pool as needed.

→ To utilize this String Constant pool the java.lang.String class is given as immutable class for remaining classes this kind of Constant pools are not available. so, they are given as mutable classes.

5) To develop user-defined immutable Java class :-

- class must be final class.

- member variables must be taken as private, final variable

- when data modification is required it should be done through factory methods having logic to return new objects.

NOTE :- When object data is modified if it is reflecting with the same object. Then it is called "mutable object" (its class is called mutable class).

→ If object data is modified if it is not reflecting with the current object but reflecting in a newly created object and reflecting by returning new object, then that object is called as immutable object (It's class is called as immutable class).

//immutable Test.java

```
final class Test
{
    private final int a;
    private final String b;

    //constructors
    public Test(int a, String b)
    {
        this.a = a;
        this.b = b;
    }

    S.O.P ("Test: 2-param Constructor");
}

//write factory method when Data modification is required.
public Test modifyData (int a, String b)
{
    return new Test(a,b);
}

public Test modifyA (int a)
{
    return new Test (a,this.b);
}

public Test modifyB (String b)
{
    return new Test (this.a,b);
}

//display object data
public void disp()
{
    S.O.P("a="+a+"b="+b);
}

public class ImmutableTest
{
    public static void main(String[] args)
    {
        Test t=new Test(10,"ranga");
        S.O.P ("t obj hashCode "+t.hashCode());
        t.disp();

        Test t1=t.modifyData(20,"ramesh");
        S.O.P ("t1 obj hashCode "+t1.hashCode());
        t1.disp();
        t1.disp();

        Test t2=t.modifyA(30);
        S.O.P ("t2 obj hashCode "+t2.hashCode());
        t2.disp();
        t2.disp();
    }
}
```

NOTE:- only final classes are not immutable classes.

Eg:- System, StringBuffer, Integer classes are final classes but they are not immutable classes.

3. Main
3. Class

* Template method Design pattern :-

problem :-

task1 → a();
b();
c();
d();
x();

here to complete the task we need to multiple methods by remembering method names and their sequence of invocation.

solution :-

use template method

```
public void mymethod()  
{  
    a();  
    b();  
    c();  
    d();  
    x();  
}
```

task1 : mymethod();

now only one method needs to be call to complete task.

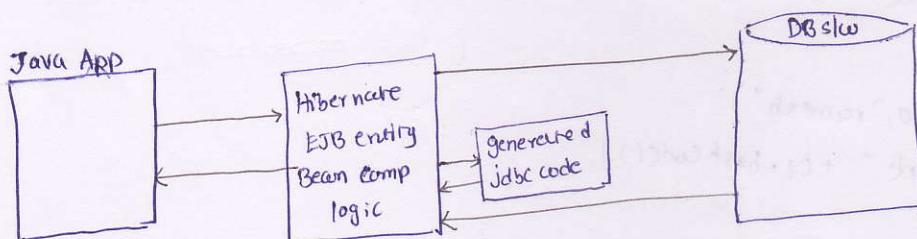
The process method of org.apache.struts.action.RequestProcessor class calls 19 process XXX() methods in a sequence to complete the task.

so, this method is called Template method.

→ ActionServlet should call all this 19 methods directly (or) Request Process class object to Complete Request Process. But it calls only one method i.e. process()

* FastLine Reader :-

problem :- using high end ORM s/w's (like hibernate) to develop persistence logic in our small and medium scale apps may degrade the performance because these ORM s/w's internally generate jdbc code to complete the persistence logic development

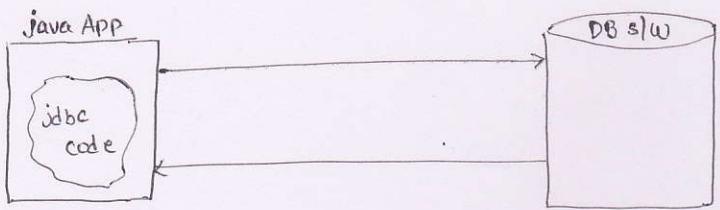


may degrade the performance.

solution :-

Use FastLine Reader design pattern

Write jdbc code directly in our small and medium scale applications to interact with the db slow to improve the performance.



03/09/2012

Abstract :- hiding the implementation

⑦ Abstract Factory pattern :-

problem :- while working with the object create and returned by factory method object

factory pattern we need to specify the class names of those objects.

Solution :- Use abstract factory pattern which is extension of factory pattern.

Here, in the class names of objects returned by factory method / pattern to implement common interface (or) should extend from common class or abstract class.

abstraction means hiding implementation. abstract factory hides the class names of the

object while receiving and utilizing those objects.

entire JDBC Specification JDBC drivers development and utilizing those drivers in our

JDBC code is happening based on abstract factory designed pattern.

```
Connection con = DriverManager.getConnection("jdbc:oracle:thin:", "scott", "tiger");
```

here con is JDBC con obj that it is the object of JDBC driver supplied Java class that

implements java.sql.Connection

its

here we are working with JDBC "con" obj without exposing and knowing its class name

even though DriverManager.getConnection(-,-,-) returns different classes objects

based on JDBC driver and JDBC URL we use. This is nothing but abstract factory

pattern implementation.

Example:- Example on abstract factory implementation.

// Common interface

```
interface ABC
{
    public void xyz();
}
```

// implement class 1

```
class Test1 implements ABC
{

```

```
    public Test1()
    {

```

```
        S.O.P ("Test1");
    }
```

```
    public void xyz()
    {

```

```
        S.O.P ("xyz of Test1");
    }
```

```
}
```

// implementation class 2

```
class Test2 implements ABC
{

```

```
    public Test2()
    {

```

```
        S.O.P ("Test2");
    }
```

```
    public void xyz()
    {

```

```
        S.O.P ("xyz of Test2");
    }
```

```

}
```

```
}// Test2
```

Class Demo

```
{
```

```
    public static public static ABC m1(String name)
```

```
{
```

// factory pattern

```
    if (name.equals("a"))

```

```
        return new Test1();

```

```
    else if (name.equals("b"))

```

```
        return new Test2();

```

```
    else

```

```
        return null;
    }
}
```

```

    // client code
    public class AbstractFactoryTest
    {
        public static void main(String args[])
        {
            // abstract factory pattern behavior
            ABC ab1 = Demo.m1(args[0]);
            ab1.xyz();
        }
    }

```

```

>javac AbstractFactoryTest.java
>java -AbstractFactory a
>java -AbstractFactory b

```

for related information and related example on abstract factory pattern refer regenos: 2

⑧ Prototype :-

Problem :- creating new object from scratch level having the existing object data has initial data is quite time consuming process.

Solution :- Use cloning to create new object where new object contains existing data has initial data and constructor will not be executed during object creation.

⇒ When objects are created "new" keyword constructors will be executed for objects data initialization. ⇒ When objects are created to cloning, constructors will not be executed. Because there is no need of objects initializing new objects that are created to cloning (as they old existing ^{object} data as initial data).

⇒ Working with cloning is nothing but working with prototype designed pattern.

// CloneableTest.java

```

class Demo implements java.lang.Cloneable
{
    int a,b;
    public Demo(int x,int y)
    {
        System.out.println("Demo : 2-param constructor");
        a=x;
        b=y;
    }
}

```

```

public Demo()
{
    S.O.P("Demo:0-Param constructor");
}

public Object myClone() throws Exception
{
    Object obj = super.clone(); // performs cloning current invoking object
    return obj;
}

} // class

```

```

public void disp()
{
    S.O.P("a=" + a + " b=" + b);
}

```

```

public class CloneableTest
{
    public void main(String[] args) throws Exception
    {
        // create first object
        Demo d1 = new Demo(10, 20);

        S.O.P("d1 hashCode :" + d1.hashCode());
        d1.disp();

        // perform cloning
        Demo d2 = d1.myClone();
        S.O.P("d2 hashCode :" + d2.hashCode());
        d2.disp();
    }
} // main

```

```

} // class

```

// javac CloneableTest.java

// java CloneableTest

→ Compared to normal object creation the object creation through cloning takes less time

because there is no constructor execution.

→ When objects are created through cloning and Deserialization the constructors will not be executed

for related information on prototype Design pattern refer Rego's 11 & 12

9) Builder Design pattern:-

problem :- construction of complex objects directly by taking one complex class is not recommended process.

solution :- Construct complex objects from multiple simple objects step-by-step as needed. This is called builder design pattern. which improves the reusability of multiple individual objects by constructing the complex objects.

Example :- // BuilderTest.java

```
class Burger
{
    public int price()
    {
        return 25;
    }
}
```

```
class Fries
{
    public int price()
    {
        return 15;
    }
}
```

```
class Drink
{
    public int price()
    {
        return 30;
    }
}
```

```
class MealBuilder
{
    public int calcPrice()
    {
        //builder pattern logic
        return new Burger().price() + new Fries().price() + new Drink().price();
    }
}
```

```
class SnackBuilder
{
    public int calcPrice()
    {
        //builder pattern logic
        return new Fries().price() + new Drink().price();
    }
}
```

```

public class BuilderTest
{
    public void main(String[] args)
    {
        //Complex object 1
        MealBuilder mb = new MealBuilder();
        int val = mb.calcPrice();
        System.out.println("the meal price is " + val);
    }
}

```

```

//Complex object 2
SnackBuilder sb = new SnackBuilder();
int val1 = sb.calcPrice();
System.out.println("the snack price is " + val1);
}

```

1> javac BuilderTest.java

1> java BuilderTest

- The ActionMapping object of Struts 1.x environment is created based on Builder Design pattern.
- The request, response objects of Servlet Programming will be created by Container based on Builder Design Pattern.

For related information on Builder Design pattern refer page no: 8 to 11

Adapter class :-

Problem :- When class of the application directly implements interface it has provide implementation for all the ~~impl~~ methods of that interface even though it is not interested to provide implementation

for certain methods

```

interface XYZ
{
    public void a();
    public void b();
    public void c();
}

class Test implements XYZ
{
    public void a()
    {
        ...
    }

    public void b()
    {
        ...
    }

    public void c() // null method
    {
        ...
    }
}

```

'Test' is not interested to provide implementation for method 'cc' but it is forced to define it as null method definition.

Solution:- Take adapter class implementing interface and provide null method method definitions for interface methods. so the class of Application can override its own choice method by extending from Adapter class.

class ^{abstract} MyAdapter implements XYZ
 {
 public void a() {}
 public void b() {}
 public void cc() {}
 }

class Test extends MyAdapter
 {
 public void a()
 {}
 public void b()
 {}
 }

class Test1 extends MyAdapter
 {
 public void cc()
 {}
 }

→ generally the adapter classes will be taken as abstract classes, Bcz there classes doesn't

contain serious logics (contains null method definitions). This helps provide direct instantiation for Adapter class.

→ java.awt.event.WindowAdapter is an adapter class for 3 event listener interfaces

like WindowListener, WindowStateListener, WindowFocusListener

→ javax.servlet.GenericServlet class is partially an adaptor class for javax.servlet.Servlet interface

Q:- where did you use interfaces and Abstract classes in ur projects?

Ans:- PL designs API specification of the project having method declarations and concrete methods to programer. this process we uses abstract classes supplies both rules and guidelines and we uses interfaces to supply only rules (method declaration).

Sun micro system gives API specification (like JDBC, servlet & etc...) to Vendor Companies having

- rules and guidelines to develop slo's (like JDBC drivers and containers). In this process abstract classes will be used! Supply both rules and guidelines and Interfaces will be used. Supplied just rules.
- while developing adapter classes abstract classes will be taken. while developing business Components Service Interfaces are Java Interface.
 - provide Special Run-time Capabilities to object to JRE marker interfaces are required.
 - May ordinary Java class as Special component must be extended from Special classes, abstract classes are its must implements Special interface.

06/09/2012

(10) VO

class | DTO class :-

Problem:- ResultSet object is not Serializable object. so, we can't send it directly over the network.

Solution:-

Approach 1:- Use Rowsets instead of ResultSets

(all Rowsets are Serializable objects by default)

Approach 2:- Copy data of ResultSet obj to collection framework datastructure and send that

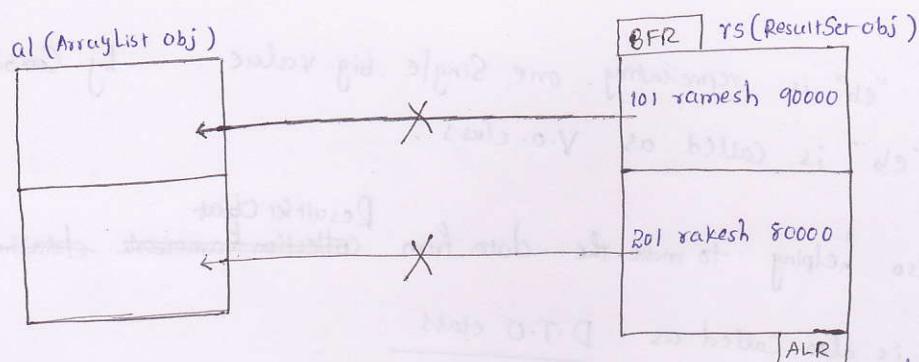
DataStructure over the netwrok. (All collection framework Datastructures are Serializable objects by default).

NOTE:- Since most of JDBC driver's are not Supporting Rowsets so use Approach 2 to solve the above problem.

- After moving data to datastructure (ds) if you are looking to perform Simultaneous Read operations then use non-synchronize datastructure like ArrayList, HashMap for better performance.
- After moving data to collection framework datastructure if you are looking to perform both read and write operations Simultaneously then use Synchronized datastructure like Vector for thread safety.

In the above scenario we prefer working with ArrayList DataStructure.

understanding the problem related to copying ResultSet data to ArrayList



- Each record of ResultSet contains multiple values including multiple objects. Each element of ArrayList allows only one object. so, we can't move each record of ResultSet directly to each element of ArrayList.

To make the above operation possible create multiple objects for User-defined class having the data of multiple records and add these objects to multiple elements of ArrayList. In this process this User-defined Java class is called as V.O class / DTO class (Value Object) / (Data Transfer Object)

Example Scenario :-

V.O class / D.T.O class (java Bean)

EmpBean.java :-

```
public class EmpBean implements java.io.Serializable {
```

```
    private int no;
```

```
    private String name;
```

```
    private float sal;
```

// write SetXxx(-) and getXXX() methods

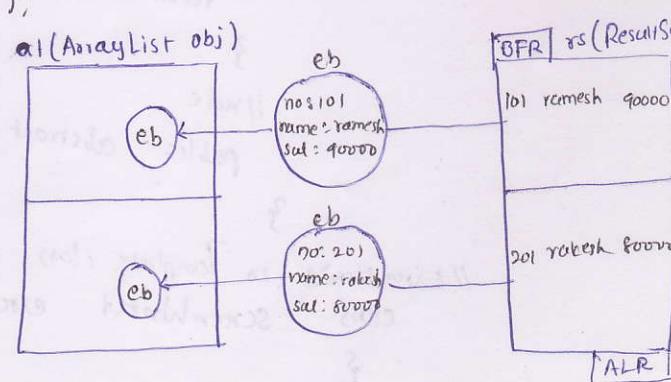
// Logic to copy ResultSet object records to ArrayList

```
ResultSet rs = st.executeQuery("select * from employee");
```

```
ArrayList al = new ArrayList();
```

```
while (rs.next())
```

```
{  
    // copy each record to one EmpBean obj  
    EmpBean eb = new EmpBean();  
    eb.setNo(rs.getInt(1));  
    eb.setName(rs.getString(2));  
    eb.setSal(rs.getFloat(3));
```



//Add each EmpBean object to ArrayList

al.add(eb);

} //while.

→ In the above Scenario "eb" is representing one Single big value by Combining multiple values. So, the class of "eb" is called as V.O. class.

Similarly "eb" is also helping to move the data from ~~Collection framework~~ ^{Resultset Object} datastructure to

ArrayList elements. So, "eb" is also called as D.T.O class ^{the class of}

NOTE:- we need to implement above design pattern only by working with JDBC.
This implementation is not required while working with Spring, hibernate think of they directly return List Data Structure.

(ii) DesignPattern *) Template class :-

problem:- Developing multiple classes of Same Category from Scratch level kills the reusability and increases the burden on the programmers.

Solution:- Use Template class providing mould for programmers having rules and guidelines.
so, programmers can develop their classes Based on this template class quite easily.

Template classes are generally abstract classes.

Example :-

```
//TemplateTest.java
import java.io.*;
//Template class
abstract class Writer1
{
    // guideline
    public int sum(int a,int b)
    {
        return a+b;
    }
    // rule
    public abstract void showResult(int res);
}
// Extension 1 to Template class
class ScreenWriter extends Writer1
{}
```

```
public void showResult(int res)
{
    s.o.p(res);
}

//ScreenWriter
```

//Extension2 to Template class

```
classs FileWriter1 extends Writer
```

```
{
    public void showResult(int res)
```

```
{
```

```
try
```

```
{
```

```
String s1 = String.valueOf(res);
```

```
BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(new FileOutputStream("abc.txt")));
```

```
BufferedWriter
```

```
bw.write(s1);
```

```
bw.flush();
```

```
bw.close();
```

```
} //try
```

```
catch (Exception e) {}
```

```
} //catch
```

```
} //class
```

```
public class TemplateTest
```

```
{
```

```
psvm (String[] args) throws Exception
```

```
{
```

```
ScreenWriter sc = new ScreenWriter();
```

```
int res = sc.sum(10, 20);
```

```
sc.showResult(res);
```

```
FileWriter1 fw = new FileWriter1();
```

```
res = fw.sum(10, 20);
```

```
fw.showResult(res);
```

```
} //main
```

```
} //class
```

(1) IOC / Dependency Injection :-

07/09/2012

problem:-

Dependency Lookup :-

Dependency Lookup of resource is getting its dependent values by searching for them explicitly is called as Dependency lookup. In Dependency lookup the Resource has to pull the value explicitly before utilizing them.

Eg(1):- student gets material only when he demands for it.

Eg(2):- The way DataSource object is gathered from Registry s/w through JNDI code.

solution:- Dependency Injection. / (IOC).

Solution:- dependency injection code.

Dependency Injection:-

If underlying container / Framework / server / Run-time environment dynamically assigns values to resource then it is called "dependency injection".

→ In Dependency injection underlying server / container... pushes the values to ~~resource~~ so, the resource need ~~not~~ to spend additional time to gather the values.

Eg(1):- The way JVM executes constructors automatically to initialize the object ~~when~~ object is created.

Eg(2):- The way ActionServlet writes formdata to form bean.

Eg(3):- The way Spring Container performs Setter, Constructor and Interface injections on Spring Bean

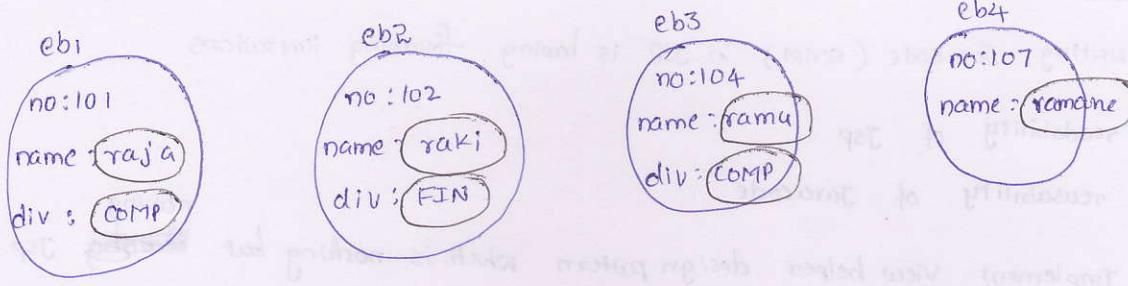
→ The way object is initialized through constructor execution comes under Dependency Injection.

→ The way object is assigned with data through method calls comes under Dependency lookup.

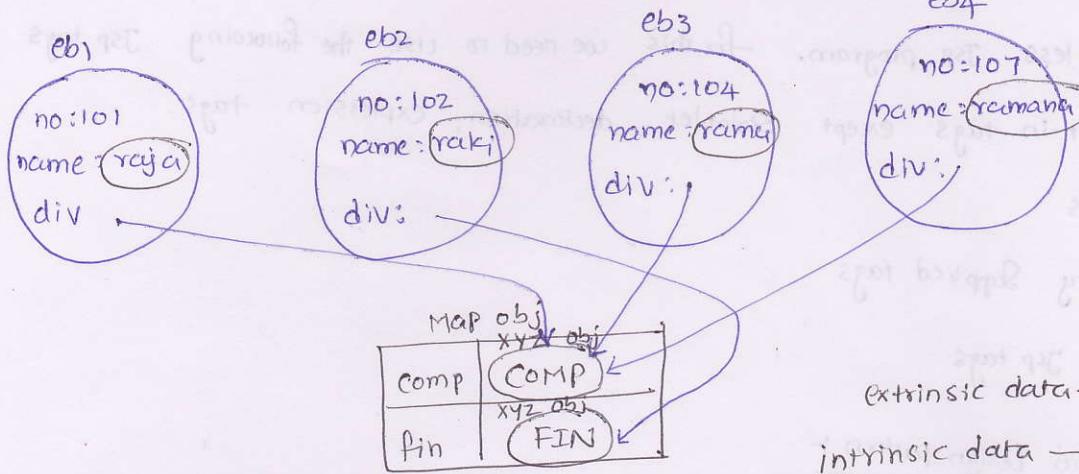
The Flyweight

Fly weight pattern :-

problem:- 3 objects of employee class representing 4 employees details



solution:-



extrinsic data → Uncommon data

intrinsic data → common data

problem:- when you create multiple objects for a class having different values still there is a possibility of having some same, common data/values in certain properties of these multiple objects. Instead of object think about allocating common memory and using multiple objects.

→ The common data of multiple objects created for a class is called ~~xyz obj~~ intrinsic data. Similarly the data that is specific to each object is called extrinsic data.

In the above diagram no name values in every object comes under

extrinsic data and "div" comes under intrinsic data.

Solution :- store extrinsic data in every object and intrinsic data in common memory (common obj) and use it in shared every object through Flyweight Design pattern implementation.

For example application on Flyweight Design Pattern, refer given supplementary handout.

II) * Web level Design pattern *

① View Helper :-

problem :- Writing Java code (script) in JSP is having following limitations

- ① kills the readability of JSP
- ② kills the reusability of Java code.

solution :- Implement view helpers design pattern which is nothing but ~~writing~~ making JSP programs as

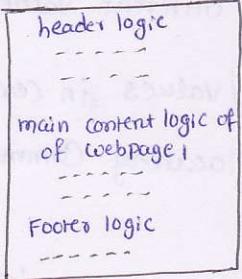
Java code less JSP program. For this we need to use the following JSP tags

- ① JSP built-in tags except Scriptlet, declaration, expression tags.
- ② JSTL tags
- ③ Third party Supplied tags
- ④ Custom JSP tags

② Composite View Design pattern :-

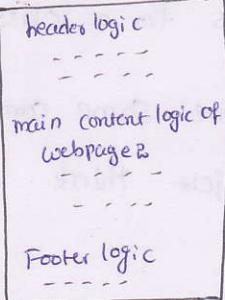
Problem :-

Soult 1 / JSP pg 1



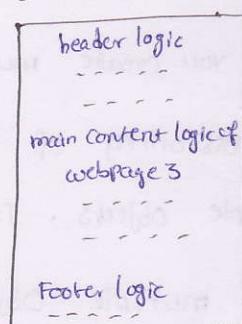
(generates webpage1)

Soult 2 / JSP pg 2



(generates webpage2)

Soult 3 / JSP pg 3



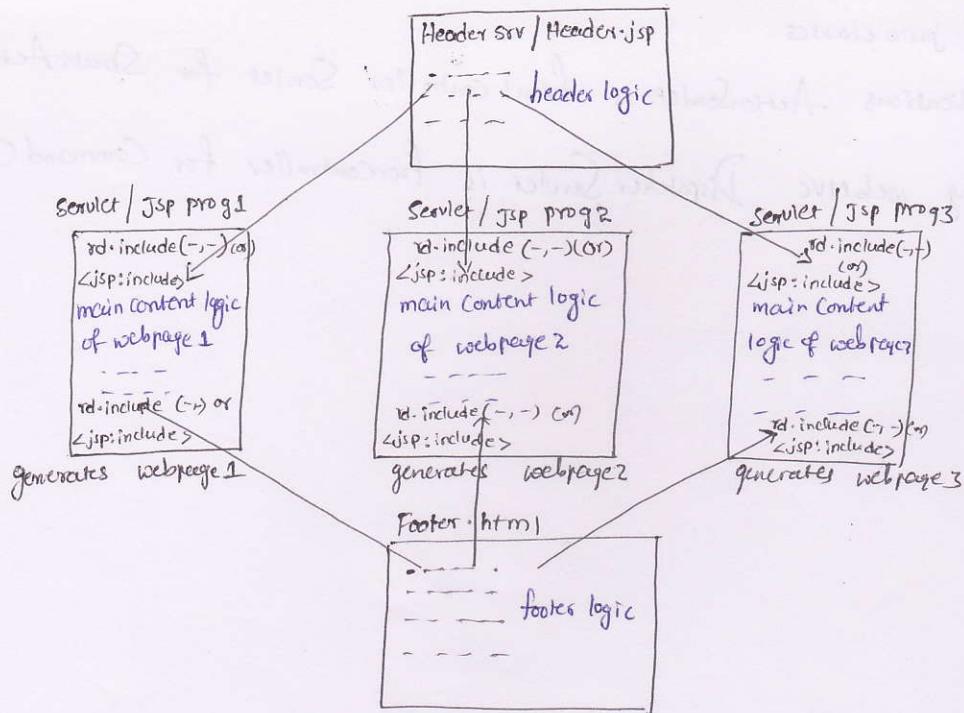
(generates webpage3)

Here the common logic of multiple web resources program are not reusable logic (like header, footer logic) because they are ~~not~~ hard coded in every web resource program.

→ The multiple web pages of web site contain same header and footer contents, But main Content of each page is different.

Solution :- make the Common logics as reusable logics by keeping them in Separate webresource programs and include their output. ~~to make~~ The main webresource programs its generate webpages for this we need to use <%@include file="include.jsp" %> include >

This whole process is called implementation of Composite View Design pattern.



NOTE:- here Common logics (header and footer logics are reusable logics)

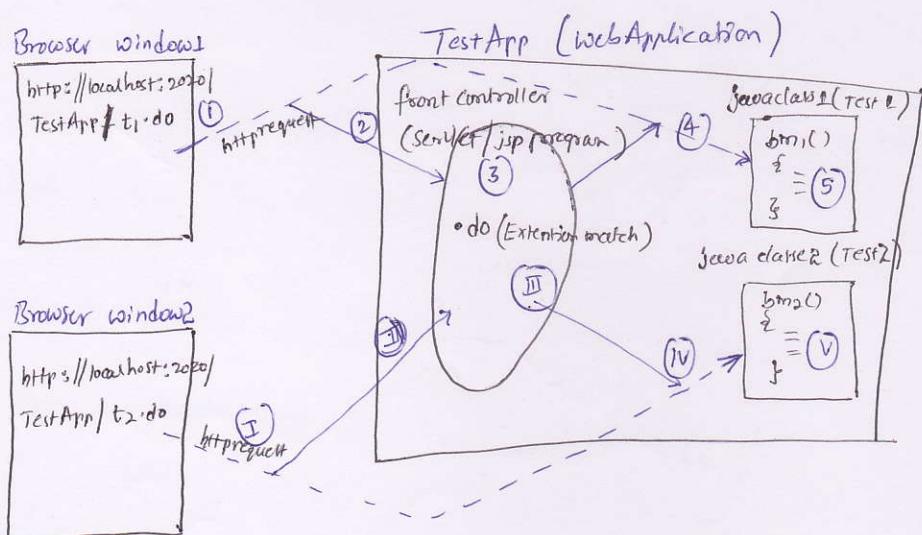
③ front controller Design pattern :-

Problem:- Ordinary java classes that are acting as web resource programs of web application can't take the client generated httprequest directly.

NOTE:- only Servlet/JSP programs of Java webapplication can take httprequest directly.

Solution:- Develop special Servlet / JSP programs as front controller trap all httprequest given by clients and to pass them to appropriate java classes

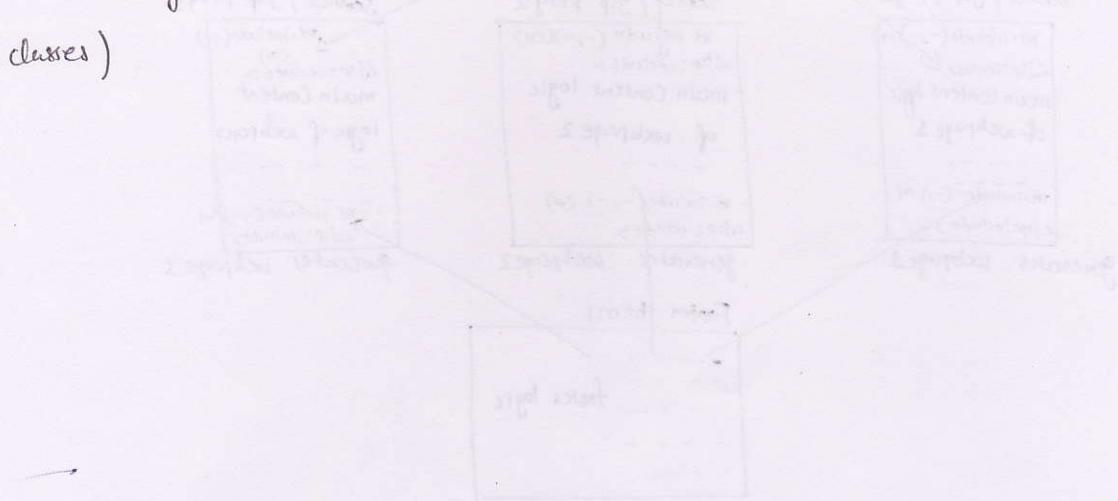
NOTE:- Front Controller Servlet / JSP program must be configured web.xml either by using extension match (or) directly match url-pattern.



NOTE:- Front Control is a Special web resource program (Servlet program / JSP program) which acts as Entry and Exit point for all request that are given to other web resource programs of

web application which are java classes.

Eg:- In Struts 1.x applications ActionServlet is front controller Servlet for Struts Action Classes (Java classes). In Spring web MVC Dispatcher Servlet is front controller for Command Controller classes (Java classes)

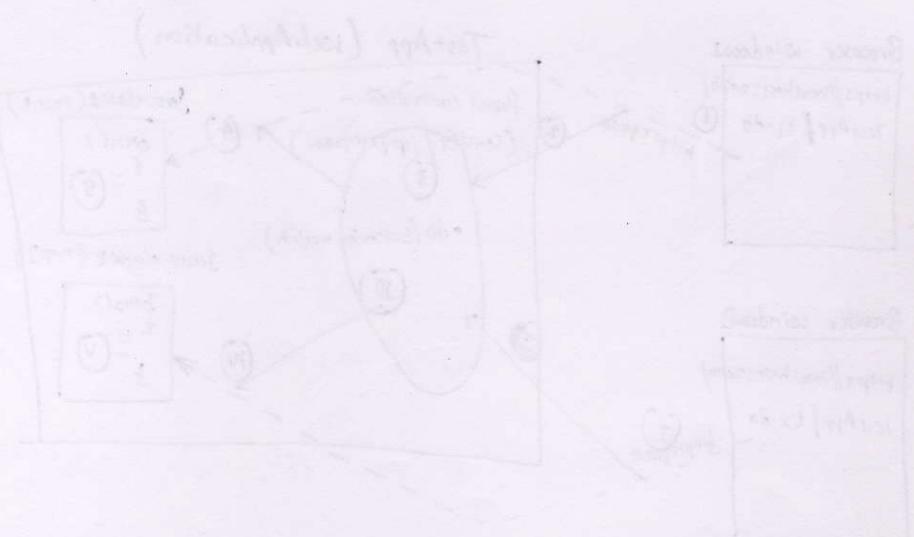


(single element no single point has point) single names and same

* another request reflected from

then controller does its mapping, controller do get the info from clients and provide

• forward temporary objects, controller does its mapping to temporary objects also
• copy temporary objects to new controller then to mapping temporary objects
• copy temporary to new controller then to mapping the first mapping to new controller
• controller mapping at each step of this steps of
• pass of this (forward temporary) and then mapping the first controller work is done
• anticipates, direct plumb (n) action patterns



So the below (mapping and mapping file) mapping does not look so good and

↳ mapping introduced with respect to the target do not reflecting the fact

Web.xml

Configure

Front Srv program in web.xml file with

*.do

*.do url pattern

NOTE:- Front Controller servlet / JSP program must be configured in web.xml file either
 with directory match url pattern or with extension match url pattern.

Web.xml

```
<web-app>
```

```
  <servlet>
```

```
    <servlet-name> F </servlet-name>
```

```
    <servlet-class> FrontSrv </servlet-class>
```

```
  </servlet>
```

```
  <servlet-mapping>
```

```
    <servlet-name> F </servlet-name>
```

```
    <url-pattern> *.do </url-pattern>
```

```
  </servlet-mapping>
```

```
</web-app>
```

// FrontSrv.java

```
import javax.servlet.*;
```

```
import javax.servlet.http.*;
```

```
import java.io.*;
```

```
public class FrontSrv extends HttpServlet
```

```
{
```

```
  public void doGet(HttpServletRequest req, HttpServletResponse res) throws
```

```
{
```

```
  // generate Settings
```

```
  PrintWriter pw = res.getWriter();
```

```
  res.setContentType("text/html");
```

```
// take the request and pass the request to appropriate java class
```

```
  int result = 0;
```

```
  if (req.getServletPath().equals("/t1.do"))
```

```
{
```

```

Test1 t1 = new Test1();
result = t1.bm1(10, 20);

if
else if (req.getServletPath().equals("/t2/do"))
{
    Test2 t2 = new Test2();
    result = t2.bm2(10, 20);
}

// display the result
pw.println("<br><b>The result is : " + result);

// doGet()
public void doGet(HttpServletRequest req, HttpServletResponse res)
{
    doGet(req, res);
}

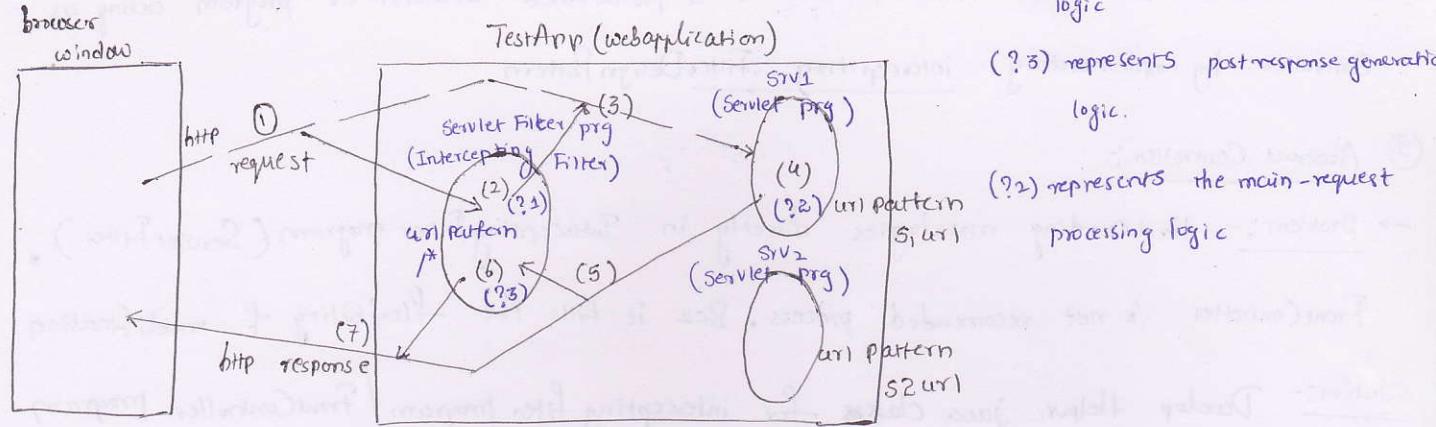
```

Date: 10/09/2012

(4) Intercepting Filter :- Intercepting Filter is the special web resource program of web application that is capable of trapping the request & response of other webresource programs. is called as "Intercepting Filter".

Problem :- keeping the common and global pre-request processing logic and post response generation logic in every main webresource program of webapplication kills the reusability of the code.

Solution:- Implement Intercepting filter Designed pattern which is nothing but Servlet filter program. → Servlet filter program execute Common pre-request processing logic by trapping request. Similarly execute Common post response generation logic by trapping the response of other webresource programs.



→ Servlet Filter is nothing but implementing intercepting Filter Design pattern. Generally we place the following pre-request logic in ServletFilter program.

- ① Authentication logic
- ② Authorization logic
- ③ Logging logic

→ We place the following post-response generation logics

- ① Decompression logic (It takes the output of compression logic.)
- ② Transformation logic and etc....

(NOTE:-) FrontController Traps the request to pass them to ordinary java classes

Intercepting Filter Traps the request going to Servlet programs, Jsp programs and Java classes.

A Sample Servlet Filter program Development :-

MyFilter.java

```

public class MyFilter implements Filter
{
    Public void init(FilterConfig fg)
    {
        ----- //Initialization logic
    }

    public void doFilter(ServletRequest req, ServletResponse res, FilterChain fc)
    {
        ----- //pre-request processing logic
        fc.doFilter(req,res);
        ----- //post-response generation logic
    }

    public void destroy()
    {
        ----- //uninitialization logic
    }
}

```

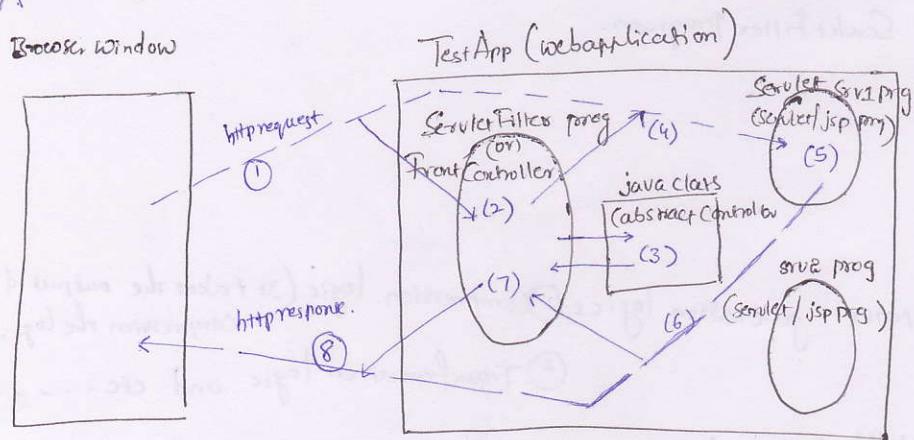
- (?1) represents pre-request processing logic
- (?3) represents post-response generation logic.
- (?2) represents the main-request processing logic

→ In Struts 2.x application FilterDispatcher is a pre-defined ServletFilter program acting as Controller. by implementing intercepting Filter Design Pattern.

(5) Abstract Controller :-

→ Problem :- Hard coding main logics directly in Intercepting Filter programs (ServletFilter), FrontController is not recommended process. Bcz it kills the flexibility of modification.

Solution :- Develop helper Java class for intercepting filter program / FrontController program having Main logics get the flexibility of modification. This Helper class is nothing but Abstract Controller.



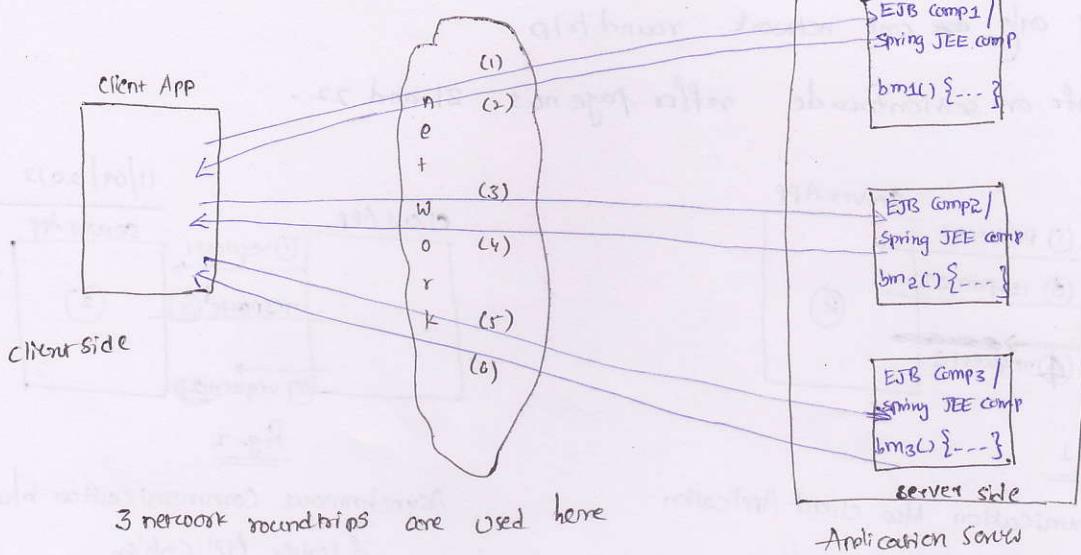
→ Abstract Controller class is very useful to customize the logics of FrontController program (as) InterceptingFilter program being from outside of these programs.

→ in Struts 1.x environment RequestProcessor class is Abstract Controller for the FrontController ActionServlet. Similarly In Struts 2.x environment Interceptors are the abstract controllers for InterceptingFilter Controller called FilterDispatcher.

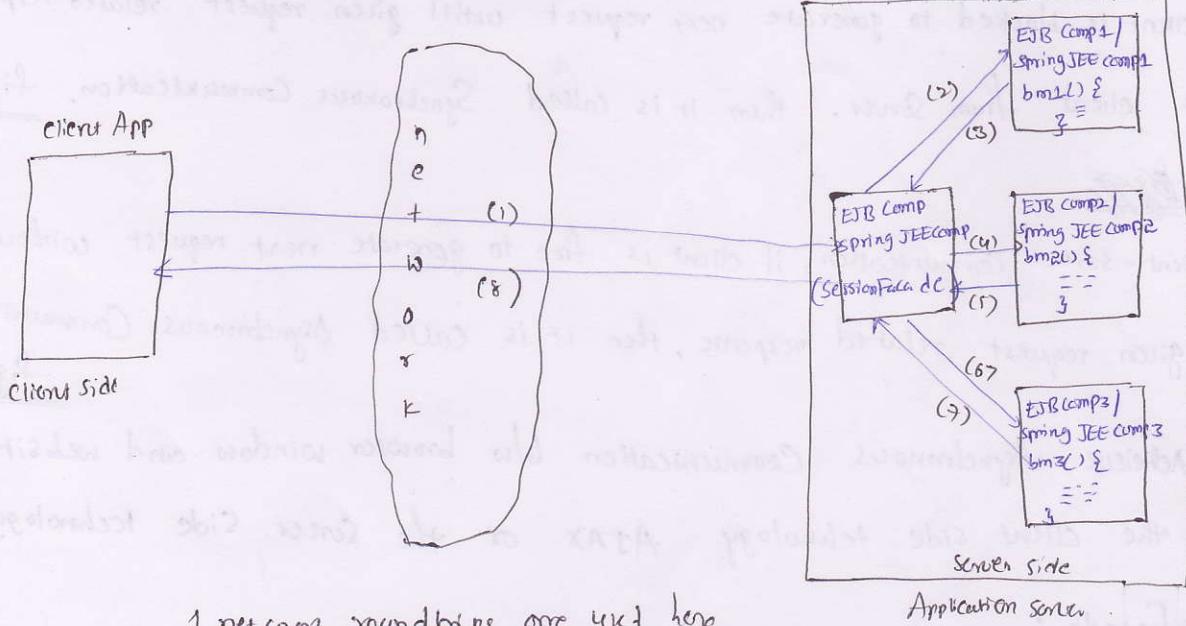
III) Integration layers Design Pattern *)

① Session Facade Design Pattern :-

(I)



(II)



1 network roundtrips are used here.

Problem:- If Remote client directly talks with multiple business Components there is a chance of getting multiple network roundtrips b/w client and ~~Business Component~~ as shown above (I) diagram

Solution:- To reduce the network roundtrips the above scenario take one dummy Business Component at Server Side to receive request from client and make that Component interacting with Other Business Component as shown above (II). here the dummy business component is called as "Session Facade". Session Facade doesn't contain any serious business logic. but it reduce

network round trips. When client wants to interact with multiple Business Component.

- In the above (I) diagram client application talking with multiple Business Component by just utilizing only one network round trip.
- For related info on session facade refer page no's 21 and 22.

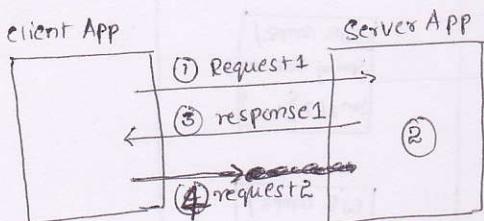


Fig-1

Synchronous Communication b/w client Application and server Application

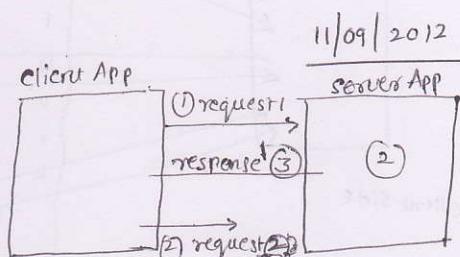


Fig-2

Asynchronous Communication b/w client App & server Application.

- If client is blocked to generate next request until given request related response comes back to client from server. then it is called Synchronous Communication. fig-I

~~Sync~~

- In client-server communication, if client is free to generate next request without waiting for given request related response, then it is called Asynchronous Communication. fig-II

- To Achieve Asynchronous Communication b/w browser window and website use either the client side technology AJAX or the server side technology portlets.

* Message Facade :-

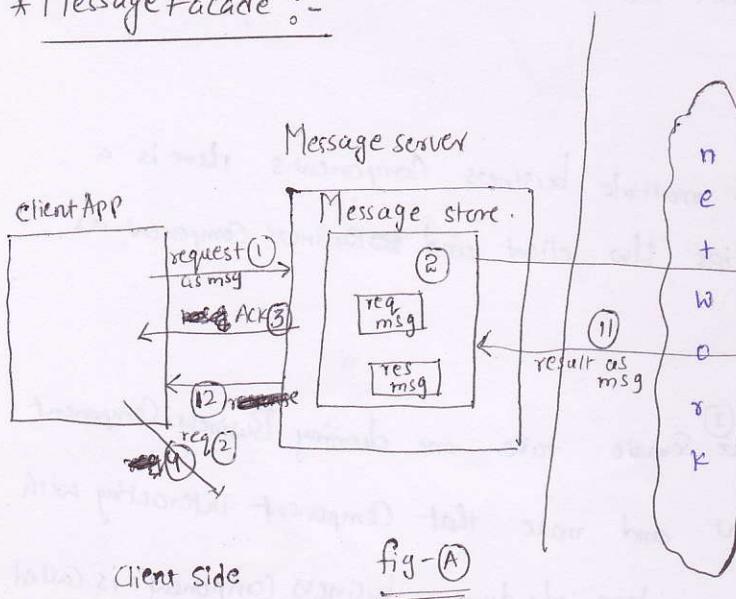
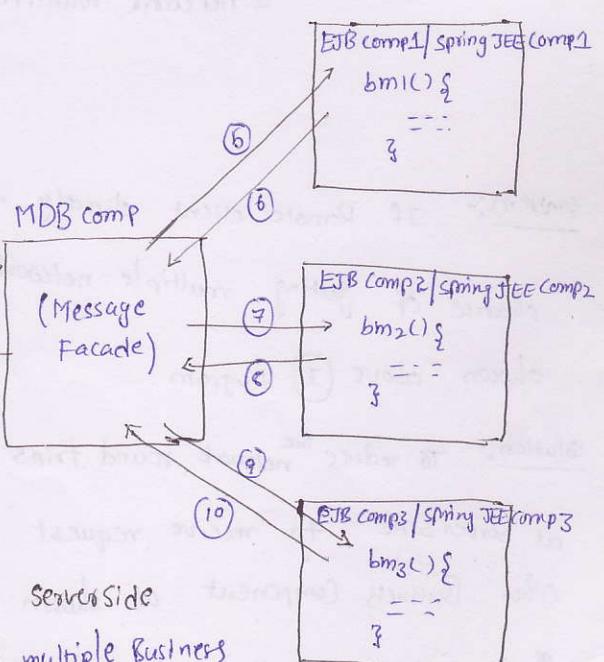


fig-(A)

Asynchronous Communication b/w Client Application and multiple Business Components



Problem:- The implementation of SessionFacade design pattern reduces network roundtrips b/w client and business Components, but that communication is Synchronous Communication. So, the client is has to wait to generate next request untill the result is received from multiple business Components.

Solution:- Use JMS and MDB support to make Communication b/w client and business Components more asynchronous as shown above diagram fig-(A)

For related information on MessageFacade design pattern, refer Page no :- 22, 23

DAO : (Data Access Object) :-

Problem:- Mixing up persistence logic with other logics of Application (Specially with business logic) does not give flexibility of modification for persistence logic when database structure is changed or its details are changed.

Solution:- Implement DAO design pattern. it is a java class that Separates persistence logic from other logics of the application development and provides flexibility of modification.

DAO class contains the following logics -

- ① Logic to establish the connection
- ② Logic to release the Connection

③ Logic to perform CURD operations on table as per project requirement

Note:- In one project we can have either one DAO class (or) multiple DAO classes

Note:- To develop persistence logics in DAO class we can use any persistence technology like JDBC, hibernate & etc....

* DAO Factory :-

It is extension of DAO, Factory, patterns which contains the ability to return one database specific DAO class object based on the data that is supplied. This is useful when project deals with multiple database softwares performing some persistence operations on multiple database softwares.

```
public class MyDAOFactory
```

```
{
```

```
//Factory method
```

```
public static DAO getDAO (String name)
```

```
{
```

```
if (name.equals ("oracle"))
```

```
return OracleDAO();
```

```
else if (name.equals ("Mysql"))
```

```
return MySqlDAO();
```

```
else
```

```
return null;
```

```
}
```

```
abstract class DAO
```

```
{
```

```
public abstract void makeConnection();
```

```
public abstract void releaseConnection();
```

```
public abstract int insertInfo (-,-,-,-);
```

```
public abstract int updateInfo (-,-,-,-);
```

```
}
```

```
//DAO class for oracle
```

```
class OracleDAO extends DAO
```

```
{
```

```
public void makeConnection()
```

```
{  
--- logic to establish connection with oracle db/s/w.
```

```
}
```

```
public void releaseConnection()
```

```
{  
--- logic to release connection with oracle db/s/w.
```

```
}
```

```
public int insertInfo (-,-,-,-)
```

```
{  
--- logic to insert record into Oracle table
```

```
}
```

```
public int updateInfo (-,-,-,-)
```

```
{  
--- logic to update record in Oracle table.
```

```
}
```

```

// DAO class for MySql
class MySqlDAO extends DAO
{
    public void makeConnection()
    {
        --- logic to establish Connection with MySql db slw.
    }

    public void releaseConnection()
    {
        --- logic to release Connection with MySql db slw
    }

    public int insertInfo (-,-,-,-)
    {
        --- logic to insert record into MySql db slw
    }

    public int updateInfo (-,-,-,-)
    {
        --- logic to update record in MySql db slw.
    }
}

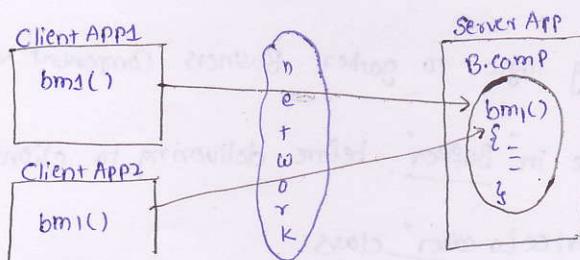
```

→ DAO Factory is a factory returning one or other DAO class object based on the data that is supplied. This Factory class will be used in other resources of the project to get their choice DAO class object and to perform persistence operations by using that DAO class object.

Business Delegate :-

12/09/2012

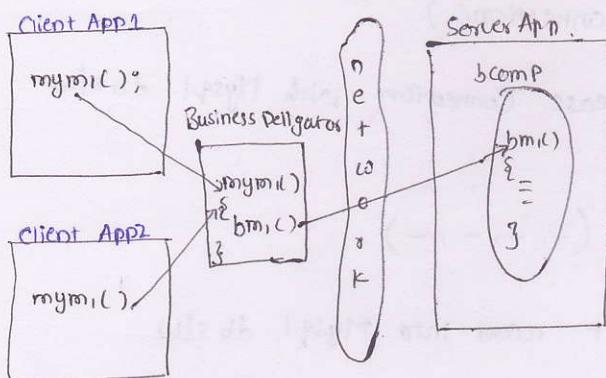
Problem:- Business In client applications calls business methods of business components available in Server application directly we don't get flexibility of modification If business method details are changed in future.



Solution:- Developed helper class for Client Application having the logics to call the business methods of Server Application. so, Any modifications are there in the details of

Business method can perform them Business Delegation class and there is no necessity of disturbing client Applications.

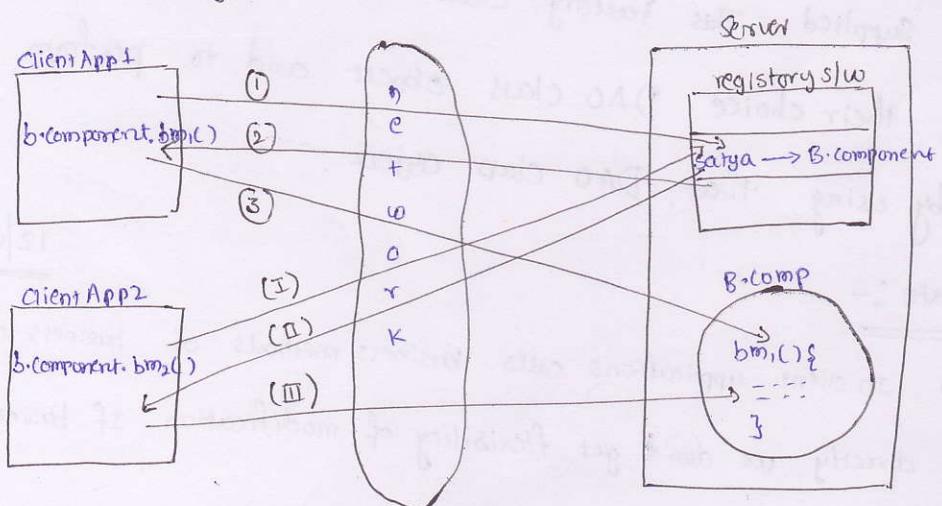
→ In one angle it calls struts ActionClass as Business Delegate when it is having logic calling the business methods of Model Layer EJB components (or) Spring applications.



Service Locator :-

Problem :- To call Business methods of Business Components Belonging to Server the client Applications must gather Business Component references from registry/s/w using JNDI code

If Every client application separately gathers this Business Component reference from registry that ~~may increase~~ more network round trips.



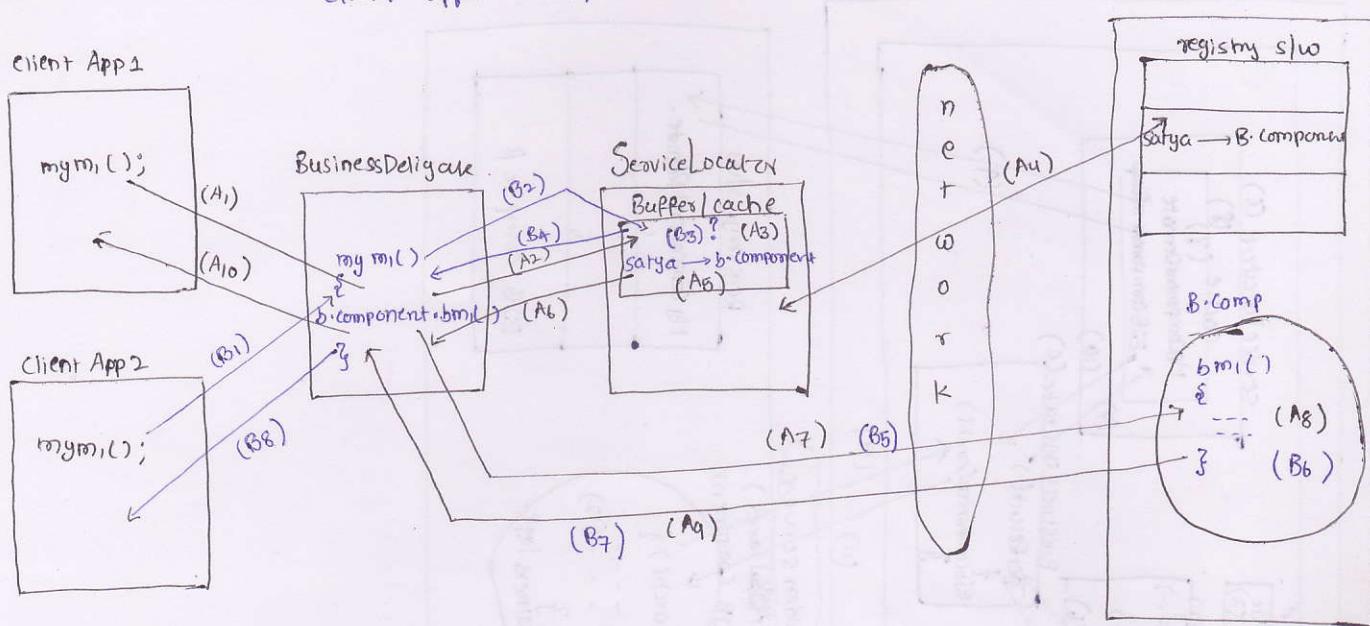
Solution :- Develop Helper class having logic to gather Business Component reference from registry of server and to keep that one in Buffer, before delivering to client Applications.

This helper class is called "Service Locator" class.

→ we generally develop this Service Locator class as Singleton Java class to avoid multiple buffers to creations.

We generally used ServiceLocator class along with BusinessDelegate class.

→ here Service Locator responsible to gather and give business object / component reference and Business Delegate responsible to call Business methods by using Business object reference and to pass the results to client application.

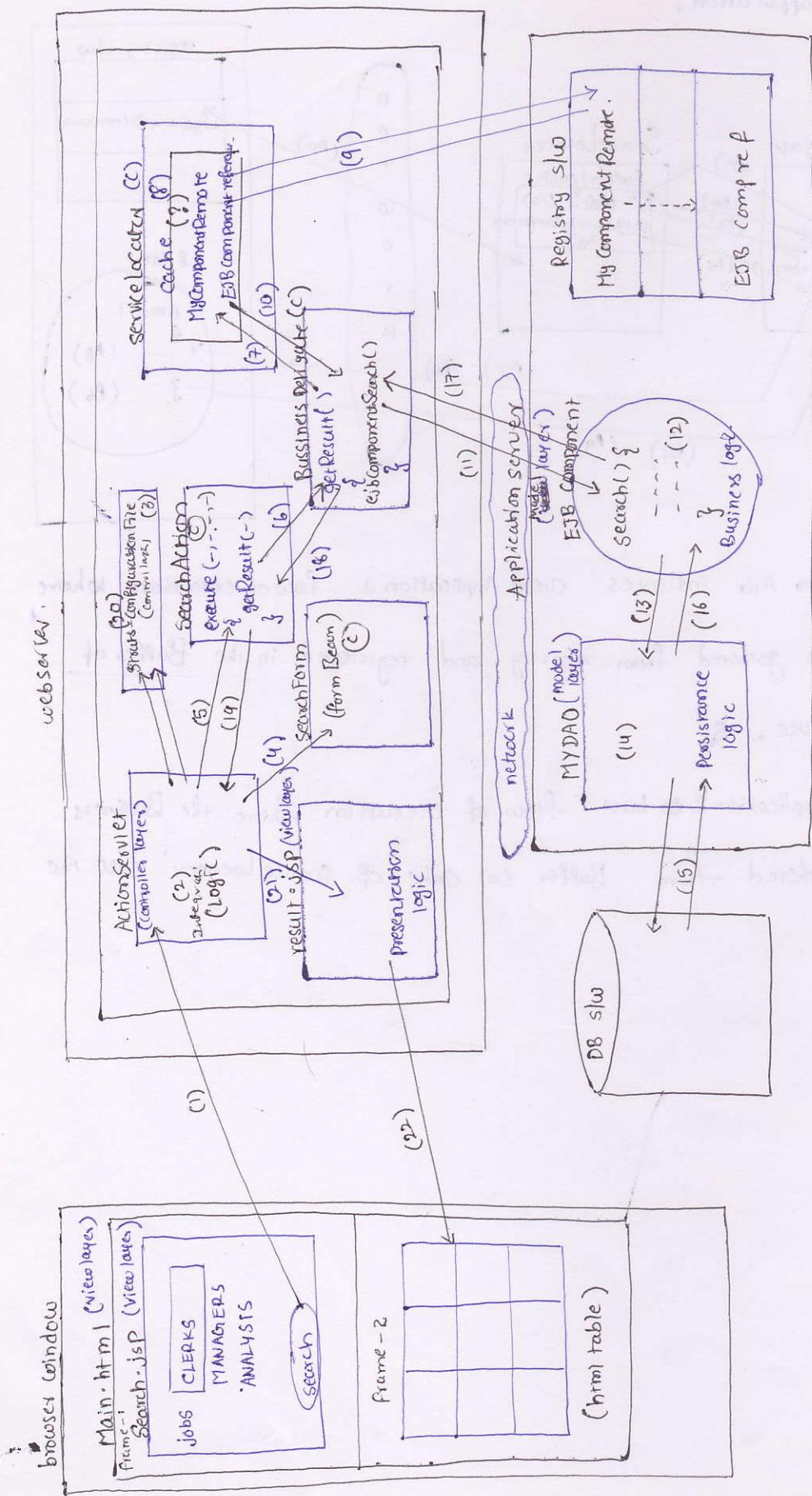


In the above diagram A₁ to A₁₀ indicates Client Application 1 flow of execution where Business object reference is gathered from registry and registered in the Buffer of Service Locator before getting use.

B₁ to B₈ indicates Client Application 2 flow of execution where the Business Component reference is gathered from Buffer (or) cache of Service locator call the Business methods.

MINI project by using Struts, EJB and JDBC technologies having the implementation

of multiple Design Patterns :-



W.r.t. to the diagram

- ① Form page submits the request by selecting one item of select box
- ② As a controller ActionServlet traps and takes the request
- ③ ActionServlet uses the entries of Struts-configuration file decide form bean and Action class to process the request.
- ④ ActionServlet writes the formdata to FormBean class object.
- ⑤ ActionServlet calls the execute() of ActionClass.
- ⑥ This execute() passes the request to BusinessDelegate class
- ⑦ & ⑧ BusinessDelegate uses ServiceLocator class and checks the availability of EJB Component reference in Buffer (or) Cache
- ⑨ Since EJB Component reference is not available in Buffer it will be gathered from the registry slo of Application Server and will be registered with the cache (or) Buffer of ServiceLocator.
- ⑩ ServiceLocator passes the EJB Component reference to BusinessDelegate class
- ⑪ BusinessDelegate class uses that EJB component reference and calls the Business method of EJB Component.
- ⑫ & ⑬ & ⑭ & ⑮ & ⑯ This Business method takes the support of DAO class to send and execute SQL select query DataBase table and gets the records into ArrayList object.
- ⑯ SQL select query DataBase table and gets the records into ArrayList object to BusinessDelegate class
- ⑰ Business method of EJB Component passes the result (ArrayList object) to ActionServlet
- ⑱ Business Delegate class parses the result to execute()
- ⑲ execute() keep the result in request attribute and transfer the control to ActionServlet
- ⑳ & ㉑ ActionServlet uses the entries of Struts-configuration file get the result page and pass the control to Result.jsp.
- ㉒ Result.jsp formats the results by using presentation logic and sends that result to 2nd frame of webpage as HTML table Content-

Explicitly implemented Design Patterns of the above project :-

- (1) MVC2 M - ejb component V - jsp programs C - ActionServlet
- (2) DAO
- (3) D.T.O class / V.O class
- (4) Business Delegate
- (5) Service Locator
- (6) Singleton java class } required while implementing ServiceLocator.
- (7) Factory method }
- (8) View Helper Design Pattern

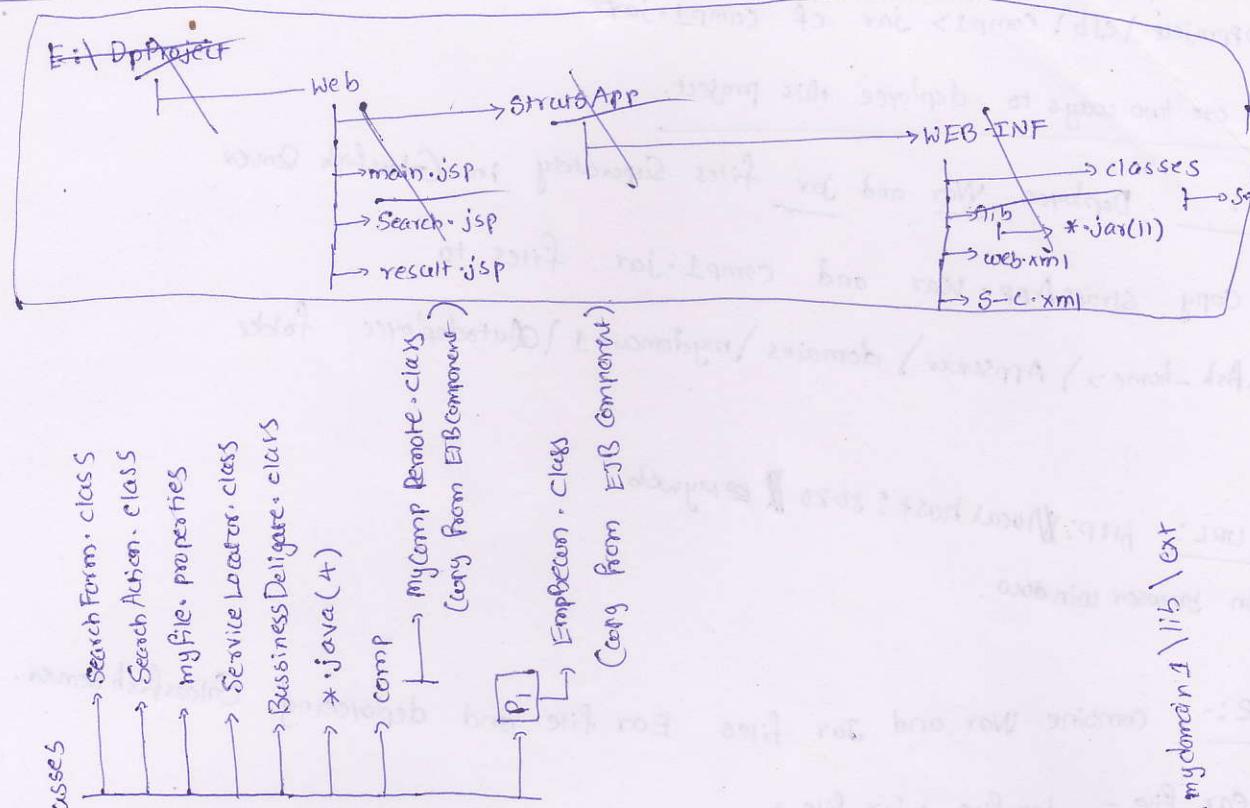
Some Implicit Design Patterns of the above project :-

- (1) Front Controller (ActionServlet)
- (2) Dependency Injection / IOC (Form Data is written to FormBean class obj)
(The way ActionServlet writes FormData to FormBean class obj)
- (3) AbstractController (The pre-defined helper class to ActionServlet which is nothing but Request Processor class)
- (4) D.T.O class / V.O class (FormBean class)
- (5) Template Method (The process(-) of Request Processor)

For the above diagram base Mini Project having the implementation of Design Pattern

refer the Supplementary Handout given on 13/09/2012

procedure to deploye and execute design project based mini project of handout



EPIPHYSIS

→ web

WEB-INF

```

graph TD
    main[main.jsp] --> lib[lib]
    main --> search[search.jsp]
    main --> result[result.jsp]
    main --> S["S-c-xm1"]
    lib --> jars[jars(10)]
  
```

Soviet-*apirojek*

Shuts - Core - 1.3.8. Jctd

... in WEB-INF\lib folder

files in a
to regular structures jax files

an **XML** file representing sheets Application

在命令行中输入以下命令，将启动文件打包成jar包

e:\DPR

E:\! Dprj\etc

```

graph LR
    object --> obj
    obj -- 1 --> comp1
    obj -- 2 --> comp1
    VITA_INF[VITA - INF]
  
```

```

classDiagram
    package com {
        MyCompremote <|-- MyCompremote.java
        myComppbean <|-- myComppbean.java
        my DAO - class <|-- myDAO - class.java
        EmpBean - class <|-- EmpBean.java
    }

```

in files in class path

javaee-jax
Odbc14-jax

Prepare JAR files representing EJB Component :-

E:\DpProject\ejb\compi> jar cf comp1.jar

There are two ways to deploy this project.

Approach 1:- Deploy War and Jar files Separately in Glassfish Server

Copy strutsApp.war and comp1.jar files to

(Glassfish-home) > Appserver > domains > mydomain1 > Autodeploy folder

Request URL: HTTP://localhost:8080/MyWeb

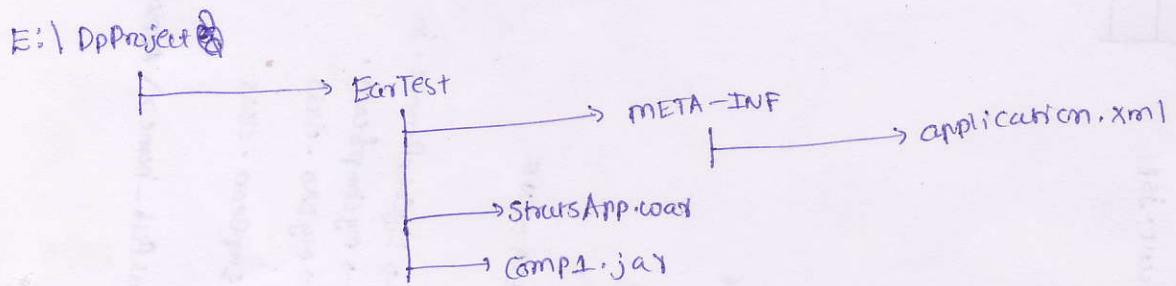
Open browser window

Approach 2:- Combine War and Jar files EAR file and deploying Glassfish server.

ear file = jar file + war file +

ear file = jar file + war file +

ear file = war file + war file +



To prepared ear file.

E:\DpProject\EartTest> jar cf FinalApp.ear

Application.xml → Jar file for ear file
<Doctype! application PUBLIC
|| ENW & WRT: || scwa & such: Combind: application - I - J dI & " >

<application>

<display-name> EartTest </d-n>

<description> Struts EJB project </description>

<module>

<jar> comp1.jar </j&b>

</module>

Mini Project implementing Design Pattern

13/09/2012

①

```

1 =====
2 App2 http://localhost:8080/StrutsApp → ①
3 =====
4 -----main.jsp----- Welcome Page (V) → ③
5 <frameset rows="30%,70%">
6   <frame name="f1" src="Search.jsp">
7   <frame name="f2"/>
8 </frameset>
9 -----Search.jsp----- FormBased (View)
10 <%@taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
11 <%@taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
12 <html:form action="searchpath" target="f2"> ④ *1 (ref 7)
13   (5) Action Path of Action class (refer line no: 69)
14     <bean:message key="my.lbl"/>
15       (refer: 31)
16     <html:select property="job" multiple="yes">
17       <html:option value="CLERK">CLERKS</html:option>
18       <html:option value="MANAGER">MANAGERS</html:option>
19       <html:option value="ANALYST">ANALYSTS</html:option>
20       <html:option value="SALESMAN">SALES MEN</html:option>
21     </html:select>
22
23     <html:submit>
24       <bean:message key="btn.cap"/>
25     </html:submit>
26
27 </html:form>
28 -----myfile.properties-----
29 # To change this template, choose Tools | Templates
30 # and open the template in the editor.
31 my.lbl=Select job(s)
32 btn.cap=Search
33 -----web.xml-----
34 <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
35   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
36   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
37   http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
38   <servlet>
39     <servlet-name>action</servlet-name>
40     <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
41     <init-param>
42       <param-name>config</param-name>
43       <param-value>/WEB-INF/struts-config.xml</param-value>
44     </init-param>
45     <load-on-startup>2</load-on-startup>
46   </servlet>
47
48   <servlet-mapping>
49     <servlet-name>action</servlet-name>
50     <url-pattern>*.do</url-pattern>
51   </servlet-mapping>
52
53   <welcome-file-list>
54     <welcome-file>Main.jsp</welcome-file>
55   </welcome-file-list>
56 </web-app>
57 -----struts-config.xml----- Struts Configuration file (Controller)
58 <!DOCTYPE struts-config PUBLIC
59   "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
60   "http://jakarta.apache.org/struts/dtds/struts-config_1_3.dtd">
61
62   <struts-config>
63     <form-beans>
64       <form-bean name="sf" type="SearchForm"/>
65     </form-beans>
66
67
68     <action-mappings>
69       <action name="sf" path="/searchpath" type="SearchAction">

```

69 to 7# Struts Action class cfg

⑦

Page 1 of 4

1

x) when form is submitted to web application the output generated by the web resource program of webapp will be displayed if frame whose name is "f2"

Deployment Descriptor File

38-51 ⑥

Selectbox
allowing us
to select
multiple
items
at a time

⑥

Form bean
Configuration

```

26 70      <forward name="success" path="/result.jsp"/> (26)
27 71      </action>
28 72  </action-mappings>
29 73
30 74  <message-resources parameter="myfile"/> → properties file configuration
31 75
32 76  </struts-config>
33 77 -----SearchForm.java----- FormBeans (Controller layer)
34 78 import javax.servlet.http.HttpServletRequest;
35 79 import org.apache.struts.action.ActionMapping;
36 80 import org.apache.struts.action.ActionForm;
37 81
38 82 public class SearchForm extends ActionForm {
39 83     private String job[]; to hold multiple values collected from list box
40 84
41 85     public void reset(ActionMapping mapping, HttpServletRequest req) {
42 86         job=new String[1];
43 87         job[0]="no item selected"; logic to handle no item selected state of list box while working with session scoped FormBean
44 88     }
45 89
46 90
47 91     public String[] getJob() {
48 92         return job;
49 93     }
50 94
51 95     public void setJob(String[] job) {
52 96         this.job = job;
53 97     }
54 98
55 99 }

```

-----SearchAction.java----- Action Class (Controller layer)

```

100 101 import java.util.ArrayList;
102 103 import javax.servlet.http.*;
103 104 import org.apache.struts.action.*;
104
105 105 public class SearchAction extends Action {
106 106     @Override
107 107     public ActionForward execute(ActionMapping mapping, ActionForm form,
108 108         HttpServletRequest request, HttpServletResponse response)
109 109         throws Exception {
110 110         //read form data
111 111         SearchForm sf=(SearchForm)form; → typecasting
112 112         String jobs[]=sf.getJob();
113 113         System.out.println("In execute(-,-,-):SearchAction");
114 114         System.out.println("before calling B.method");
115 115
116 116         // call B.method
117 117         BusinessDelegate bd=new BusinessDelegate(); → calling a method of Business Delegate class
118 118         ArrayList al=bd.getResult(jobs); → gather the result from EJB component Business method
119 119         System.out.println("after calling B.method"); (refer line no: 196 to 206 )
120 120
121 121
122 122         // send result to result page as req attribute
123 123         request.setAttribute("result",al);
124 124         //forward control to result page
125 125         return mapping.findForward("success");
126 126         //execute(-,-,-)
127 127     } //class
128
129 -----ServiceLocator.java----- Class implementing Service Locator Designed Pattern
130 130 import java.util.*;
131 131 import javax.naming.*;
132
133 133 public class ServiceLocator { It is singleton class
134 134     {
135 135         private Hashtable Cache; → maintained in local cache or buffer having EJB Component reference gathered from registry.
136 136         private InitialContext ic; Connectivity with registry.
137
138 138     private static ServiceLocator sl=null;

```

for supporting Singleton

```

139
140     private ServiceLocator() //private constructor (to support Singleton)
141     {
142         try
143         {
144             Cache = new Hashtable();
145
146             // jndi properties
147             Hashtable ht=new Hashtable();
148             ht.put(Context.INITIAL_CONTEXT_FACTORY,
149                     "com.sun.enterprise.naming.SerialInitContextFactory");
150             ht.put(Context.PROVIDER_URL,"iiop://localhost:4848");
151             ic=new InitialContext(ht);
152         } //try
153         catch(Exception e)
154         {
155             e.printStackTrace();
156         }
157     } //constructor
158
159     // Factory method having singleton logic
160     public static ServiceLocator getLocator() //factory method.
161     {
162         if(sl==null)
163             sl=new ServiceLocator();
164
165         return sl;
166     } //getLocator()
167
168     // method having ServiceLocator Impl logic
169     public Object getService(String jndiName) throws Exception
170     {
171         System.out.println("ServiceLocator:getService(-)");
172         try
173         {
174             if(!Cache.containsKey(jndiName)) // from Hash table registry
175                 Cache.put(jndiName,ic.lookup(jndiName)); // if
176             } //if
177             } //try
178             catch(Exception e) // gathers EJB component reference
179             { // from registry.
180                 e.printStackTrace();
181             } //catch
182
183             // Return object from cache
184             return Cache.get(jndiName); // The EJB Component reference of given JNDI name will be returned by gathering it from local & cache.
185         } //getService()
186     } //class
187
188     -----BusinessDelegate.java----- class implementing Business Delegate Design Pattern (control layer)
189     import java.util.*;
190     import comp.MyCompRemote;
191
192     public class BusinessDelegate
193     {
194
195         public ArrayList getResult(String job[]) throws Exception
196         {
197             System.out.println("getResult method of BusinessDelegate class");
198
199             ServiceLocator sl= ServiceLocator.getLocator(); //use the singleton object of Service locator.
200             Object obj =sl.getService("comp.MyCompRemote"); //logic to get EJB component reference
201             MyCompRemote wor=(MyCompRemote)obj; //from the cache maintained by Service locator
202
203             // Calling B.method
204             ArrayList al=wor.search(job); // calls the business method of EJB component
205             return al;
206         }
207     }

```

logic to
 Establish
 connection
 with Glass
 Fish registry
 by using its
 JNDI properties

Service locator contains some
 member variable "cache"
 Cache contains Buffer

(16)

(18)

(14)

(15)

(19)

(24)

```

208 -----result.jsp----- result Page
209 <%@page import="java.util.* , java.math.* , p1.EmpBean" %>
210
211 <%
212     ArrayList al=(ArrayList)request.getAttribute("result");
213 %>
214 <center>
215     <table border width="100%" bgcolor="#FFFFFF">
216         <tr>
217             <th>EMPNO</th>
218             <th>ENAME</th>
219             <th>JOB</th>
220             <th>SAL</th>
221         </tr>
222
223 <%
224     for(int i=0;i<al.size();i++)
225     {
226         EmpBean eb=(EmpBean)al.get(i);
227         BigDecimal bd=new BigDecimal(eb.getSal());
228     %>
229         <tr>
230             <td><b><%=eb.getEmpno()%></b></td>
231             <td><b><%=eb.getEname()%></b></td>
232             <td><b><%=eb.getJob()%></b></td>
233             <td><b><%=bd.setScale(2,BigDecimal.ROUND_HALF_UP)%></b></td>
234         </tr>
235     <%
236     }
237 %>
238 </table>
239 </center>
240
241 <%--<%@taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
242 <%@taglib uri="http://struts.apache.org/tags-bean" prefix="bean"%>
243 <%@page import="java.util.* , p1.EmpBean" %>
244
245 <% System.out.println("In result.jsp"); %>
246
247 <table border=1>
248 <%
249 ArrayList al=(ArrayList)request.getAttribute("result"); → reads the request attribute value
250 %>
251
252 <logic:notEmpty name="result" scope="request"> → checks whether the request attribute is empty or not.
253     <tr>
254         <td>ENO</td>
255         <td>ENAME</td>
256         <td>DESG</td>
257         <td>SALARY</td>
258     </tr>
259
260
261     <logic:iterate id="id1" collection="<%> al %>"> → logic to display the records of
262         <tr> → ArrayList as Html table content
263             <td> <bean:write name="id1" property="empno" /></td>
264             <td> <bean:write name="id1" property="ename" /></td>
265             <td> <bean:write name="id1" property="job" /></td>
266             <td> <bean:write name="id1" property="sal" /></td>
267         </tr>
268     </logic:iterate>
269 </logic:notEmpty>
270 </table> --%>

```

logics to go through the elements of ArrayList. In each iteration, id1 points to one element of ArrayList i.e. EmpBean class object

the request attribute is empty or not.

```

1 =====
2 App1
3 -----
4 -----MyCompRemote----- Business Interface of EJB Component
5 package comp;
6 import java.util.ArrayList;
7 import javax.ejb.Remote;
8 @Remote → makes Business interface as Remote Business Interface
9 public interface MyCompRemote {
10     public ArrayList search(String job[]);
11 }
12 -----MyCompBean----- bean class of EJB component (Model layer)
13 package comp;
14 import java.util.ArrayList;
15 import javax.ejb.Stateless;
16 import p1.EmpBean;
17
18 L.D.O. class Visibility
19 @Stateless → makes the Bean as SLSB (Stateless session Bean)
20 public class MyCompBean implements MyCompRemote {
21     (20) Business method
22     public ArrayList search(String[] jobs) {
23         System.out.println("MyCompBean:search(-) method");
24
25         // write b.logic to frame condition required for sql query
26         StringBuffer sb=new StringBuffer();
27         sb.append("(");
28         for(int i=0;i<jobs.length;i++)
29         {
30             if(i == jobs.length-1)
31                 sb.append("'" + jobs[i] + "'");
32             else
33                 sb.append("'" + jobs[i] + "',");
34         }
35         sb.append(")");
36         String cond=sb.toString(); → gives condition required for the sql query like ('CLERK', 'MANAGER')
37
38         // use the persistency logic of DAO class
39         MyDAO dao=new MyDAO(); → refer line no's 54 to 93
40         ArrayList al=dao.findEmployees(cond);
41         return al;
42     } // search → returns to BusinessDelegate class.
43
44 } // class
45 -----MyDAO----- (model layer)
46 package comp;
47 import java.sql.*;
48 import java.util.*;
49 import p1.EmpBean;
50
51 public class MyDAO {
52
53
54     public ArrayList findEmployees(String cond) → method having persistency logic
55     {
56
57         System.out.println("findEmployees(-):MyDAO");
58
59         Connection con=null;
60         Statement st=null;
61         ResultSet rs=null;
62         ArrayList al=new ArrayList();
63
64         try
65         {
66             Class.forName("oracle.jdbc.driver.OracleDriver");
67             con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:satya","scott","ti");
68             st=con.createStatement();
69             rs=st.executeQuery("select empno,ename,job,sal from emp where job in "

```

Job[]	0
CLERK	0
MANAGER	1

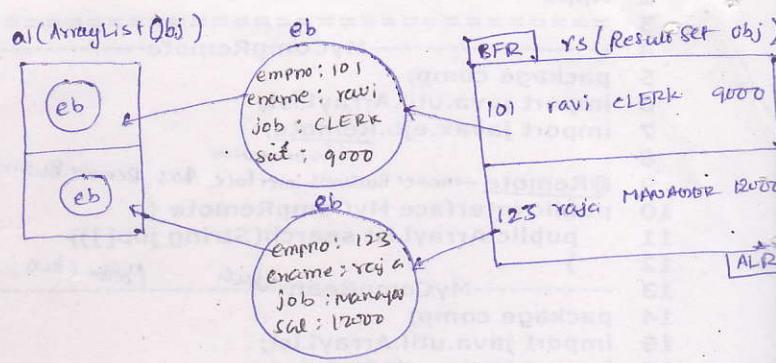
Refer line nos
54 to 93

A-93
(22)

```

70           + cond+" order by job");
71     while(rs.next())
72     {
73       System.out.println("In while");
74       EmpBean eb=new EmpBean();
75
76       eb.setEmpno(rs.getInt(1));
77       eb.setEname(rs.getString(2));
78       eb.setJob(rs.getString(3));
79       eb.setSal(rs.getFloat(4));
80
81       al.add(eb);
82     }//while
83     rs.close();
84     st.close();
85     con.close();
86   } //try
87   catch(Exception ee)
88   {
89     System.out.println(ee.toString());
90   }
91   return al; // returns to the search method of EJB component Bean class.
92 } //findEmployees
93 } //class
94
95 -----EmpBean----- D.T.O class | V.O. class
96 package p1;
97 import java.io.*;
98
99 public class EmpBean implements Serializable
100 {
101   private int empno; //mandatory
102   private String ename;
103   private String job;
104   private float sal;
105
106   public int getEmpno() {
107     return empno;
108   }
109
110   public void setEmpno(int empno) {
111     this.empno = empno;
112   }
113
114   public String getEname() {
115     return ename;
116   }
117
118   public void setEname(String ename) {
119     this.ename = ename;
120   }
121
122   public String getJob() {
123     return job;
124   }
125
126   public void setJob(String job) {
127     this.job = job;
128   }
129
130   public float getSal() {
131     return sal;
132   }
133
134   public void setSal(float sal) {
135     this.sal = sal;
136   }
137 }

```



When we deployee EJB3.x component in GlassFish server then the Business interface name of that component automatically becomes JNDI name but EJB component reference that is placed in the registry.

For the above component Comp_MyCompRemote is **6** the Jndi name.

I would try to update our site JavaEra.com everyday with various interesting facts, scenarios and interview questions. Keep visiting regularly.....

Thanks and I wish all the readers all the best in the interviews.

www.JavaEra.com

A Perfect Place for All **Java Resources**