06 SATURDAY
May 2023 Week 18 / 126-239

MAY 2023

S	M	T	W	T	F	S	S	M	T	W	T	F	S
	1	2	3	4	5	6	7	8	9	10	11	12	13
14	15	16	17	18	19	20	21	22	23	24	25	26	27
28	29	30	31										

Unit-II

08.00 • A Java Bean is a Java class that should follow the following conventions:

09.00

- It should have a no-arg constructor.
- It should be Seriallizable (enables is to ~~create~~ save the state of an object and re-create the object in a manner to code)
- It should provide methods to set and get the values of the properties, known as getter and setter methods

12.00

• Why use JavaBean?

01.00

According to Java ~~white paper~~, it is a reusable software component. A bean encapsulates many objects into one object so that we can access this object from multiple places. Moreover, it provides easy maintainance

04.00

→Simple example of JavaBean class

05.00

```
//Employee.java
package mypack;
public class Employee implements java.io.Serializable{
    private int id;
    private String name;
    public Employee(){}
    public void setId(int id){this.id=id;}
    public int getId(){return id;}
    public void setName(String name)
    {this.name=name;}
    public String getName(){return name;}
}
```

1st

2nd

3rd

06.00

Sunday 07

EVE

Notes

• How to access the JavaBean class?

08.00 To access the JavaBean class, we should use getter and setter methods

09.00

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

10.00

11.00

12.00

01.00

02.00

NOTE: There are two ways to provide values to the object. One way is by constructor and second is by setter method.

03.00

04.00

JavaBean Properties

05.00

A JavaBean property is a named feature that can be accessed by the user of the object. The feature can be of any Java data type, containing the classes that you define.

06.00

EVE

A JavaBean property may be read, write, read-only, or write-only. JavaBean features are accessed through two methods in the JavaBean's implementation class:

Notes

(1). getPropertyName()

For example, if the property name is firstName, then method name would be getFirstName() to read that property.

This method is called the accessor.

(2) set(propertyName)
For example, if the property name is firstName, then method name would be setFirstName() to write that property. This method is called the mutator.

- Advantages of JavaBeans

(i) JavaBean properties and methods can be exposed to another application.
(ii) It provides an easiness to reuse the software components.

- Disadvantages of JavaBean

(i) JavaBeans are mutable. So it can't take advantages of immutable objects.
(ii) Creating the setter and getter method for each property separately may lead to the boilerplate code.

NOTE: → A mutable object can be changed after it's created, and an ~~object~~ immutable object can't.
→ Own class objects can be immutable by making all fields final and private
→ Strings are immutable in Java.
→ Boilerplate code is computer language text that you can reuse with little or no alteration in several different contexts. (This term originates from document management, where you reuse document templates or boilerplate with minimal changes for different situations.)

→ When using language that are considered verbose, the programmer must write a lot of boilerplate
08.00 code to accomplish only minor functionality.

09.00 JavaBeans and Enterprise JavaBeans (EJBs)

• JavaBeans :→ JavaBeans are recyclable software components
10.00 for Java which by using a builder tool can be manipulated visually in like to a specific convention
11.00 → Java Beans create Java Components that can be composed together into applets and applications.
12.00 → For passing as a single bean object instead of as multiple respective objects, they're used to encapsulate
01.00 many objects in a single object (the bean)

02.00 • Enterprise JavaBeans (EJBs) :→ EJBs are server-side programs which generally implement middle layer business performance.
03.00 → EJBs are intended to handle such frequent concerns as perseverance, transactional integrity and security in a 3-way
04.00 model, which gives programmer freedom to focus on the specific problem in hand.
05.00

• Difference b/w JavaBeans & EJBs.
06.00 - JavaBeans is a component technology to create universal Java components.
EVE - EJB is a component technology, it neither reconstructs nor enhances the original JavaBean specification

EJB :→ It is a server-side software element. It encapsulates the business logic of an application. It is a specification
Notes for developing a distributed business application on the Java platform.
→ There are 3 types of EJBs: (i) Session Bean (ii) Entity Bean, and

Persistent object → one that continues to exist after the program that created it has been unloaded.

11 THURSDAY
May 2023 Week 19 / 131-234

MAY 2023

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

(iii) Message Driven Bean.

Coding

08.00 Session Bean

09.00 The bean sprout business idea that can be systematically used by customer, remote, and web service. When a
10.00 client wants to use an appl. distributed on a server, then the client uses (session bean methods) This session
11.00 bean provides services to customers, protecting them from complex problems by performing business operations within
12.00 the server.

To get to an application that is conveyed to the worker, the client summons the meeting beans strategies. The
01.00 meeting bean performs work for its client, safeguarding it
02.00 from intricacy by executing business errands inside the worker. Sessions are beans are not persistence.

03.00

There are two types of session beans:

04.00 (i) Stateless Session Beans
(ii) Stateful Session Beans
05.00 (iii) Singleton Session Beans

06.00 (i) Stateless Session Beans.

A stateless session bean doesn't keep a conversational state with the customer or client. When the client invokes the methods for a stateless bean, the beans instance variable may contain a state explicitly to that client, however only for the span of the invocation.

EVE

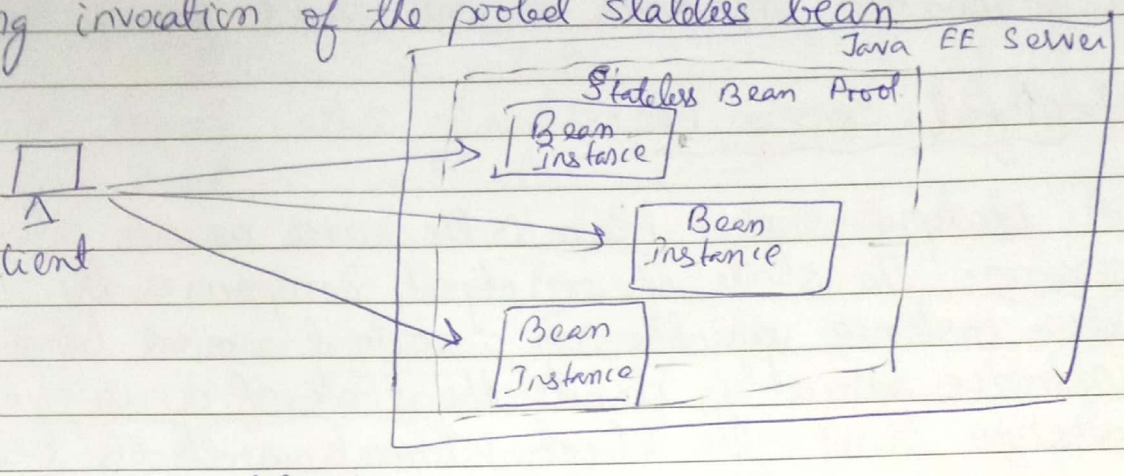
Notes

At the point when the method has finished, the client explicit state should not be held. However, the client may change the state of instance variable presented in pooled stateless beans and this state is held over to the

To err is human, to forgive divine

Http → a stateless session protocol, bcoz it doesn't keeptrack of the state

following invocation of the pooled stateless bean



08.00
09.00
10.00
11.00

Ans: Stateless Session Beans

12.00

Besides this, during the invocation of the method, all instances of a stateless bean are the same, permitting the EJB container to assign an instance to any client. Therefore, the state of a stateless session bean should apply across all clients. It can be implemented in a web services

01.00
02.00
03.00

Since they can maintain various customers stateless session beans can offer better adaptability for applications that require large number of clients. Ordinarily, an application requires fewer stateless session beans than stateful sessin beans to support the same number of clients.

04.00
05.00
06.00

EVE

To improve execution, we may choose a stateless session bean in the event that it has any of these characteristics.

Notes

- The state of the bean has no information or data for a particular client.
- In a private method invocation, the bean performs a generic task for all clients.

- The bean implements a web service.

(a) Stateful Session Beans

- A stateful session bean is the same as an interactive session. The state of an object comprises the value of its instance variables. In a stateful session bean, the instance variables denote the state of a unique client/bean session. Since, the client interacts with its bean, the state is usually called the conversational state.
It maintained the state of a client across multiple requests. So, we cannot implement it in the web service because it cannot be shared. When the client terminates the session, it is no longer associated with the client.

- The state is held for the span of the client/bean session. In case of the client eliminates the bean, the session closes and the state is omitted. This transient nature of the state isn't an issue, because the interaction between the client and the beans ends. So, it is not required to hold the state.

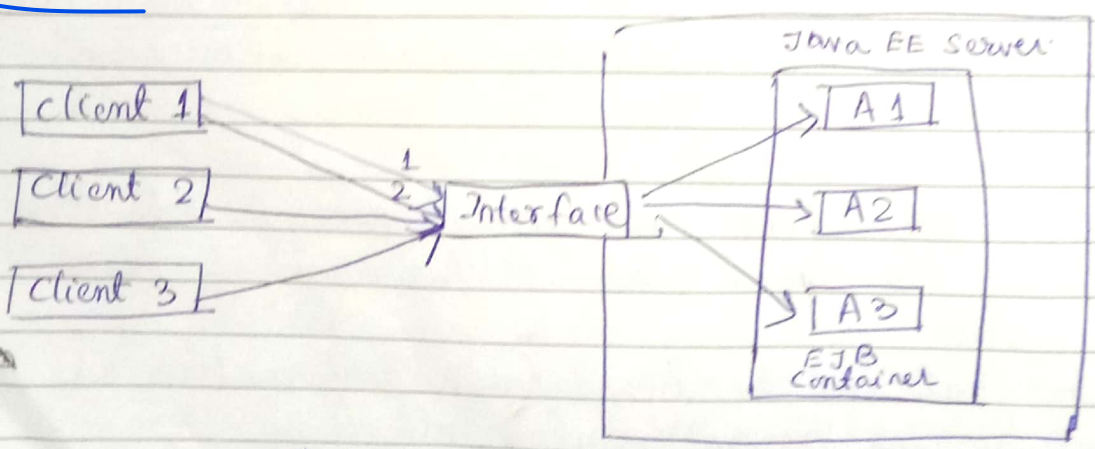


Fig: Stateful Session Beans State Management

Sunday 14

Notes

To accept good advice is but to increase one's ability

- Stateful session beans should be used, if any of the following conditions are valid:

08.00 * (i) The bean's state denotes the alliance between the bean and a particular client.

09.00 * (ii) The bean needs to hold data about the customer across method invocations.

10.00 * (iii) The bean interferes between the customer and and different segments of the application, introducing an improved visible to the client.

11.00 * (iv) In the background, the bean deals with the workstream of a few enterprise beans.

12.00

01.00 Note: Stateless and stateful both session beans are not persistent → Data

02.00

(iii) Singleton Session Beans

03.00 A singleton session bean maintains one instance per application and the instance exists for the lifecycle of the application. It offers the same functionality as the stateless session beans. But the only difference is that there is only one singleton session bean per application while in the stateless

04.00

05.00

06.00 session bean a pool of bean is used.

EVE From that pool, any of the session beans may respond to the client. We can implement it in the web-service endpoints. It takes care of the state but does not hold the state if unexpected crashes or shutdown occur. It can be used if we want to perform cleanup tasks on closing the application or shut down as well. It is because it operates throughout the life cycle of the application.

Notes

16 TUESDAY
May 2023 Week 20 / 136-229

MAY 2023
S M T W T F S S M T W T F S
1 2 3 4 5 6 7 8 9 10 11 12 13
14 15 16 17 18 19 20 21 22 23 24 25 26 27
28 29 30 31

08.00
09.00
10.00
11.00
12.00
01.00
02.00
03.00
04.00
05.00
06.00
EVE

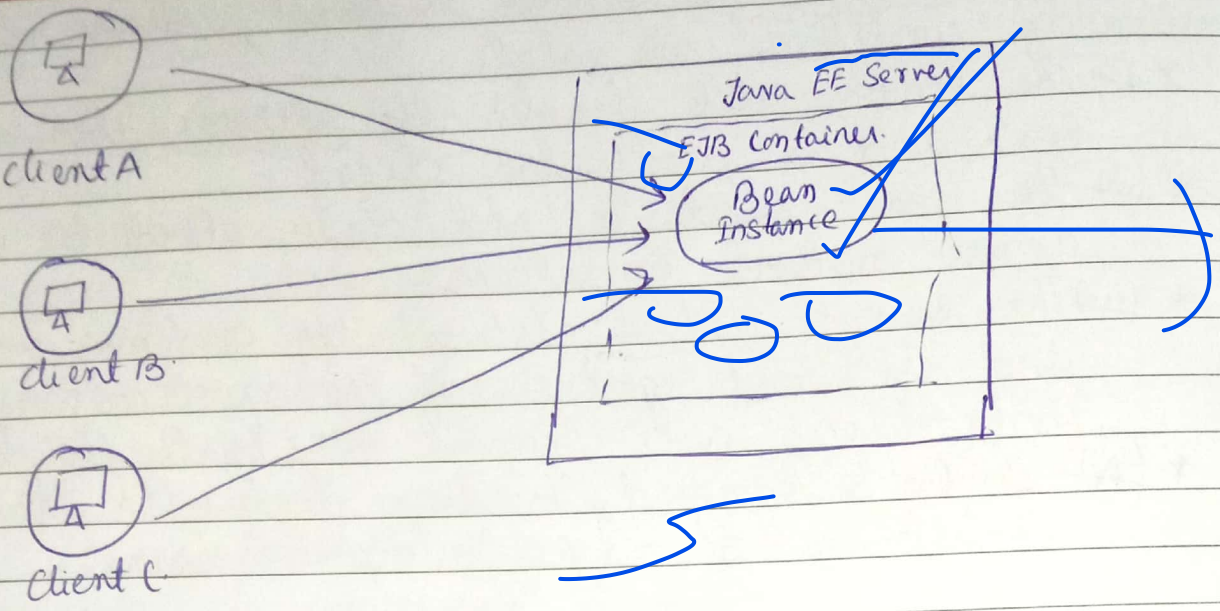


FIG: Singleton Session Bean

- Singleton Sessin beans are suitable for the following conditions:
  * The state of the bean must be shared accross the application
  * A singl enterprise bean should be accessed by multiple threads concurrently.
  * The application needs an enterprise bean to perform undertakings upon application startup and shutdown.
  * The bean implements a web service

Entity Bean (It is a remote object that manages persistent data, performs complex business logic, potentially uses several dependent Java objects, and can be uniquely identified by a primary key).

→ An entity bean is an unpredictable business substance. It models a business substance or models different activities inside a business interaction.
→ It is used to encourage business benefits that includes data and calculations on that data.

Notes

diligences → careful and persistent work or effort | endure → suffer
diligent → constant in effort to accomplish something

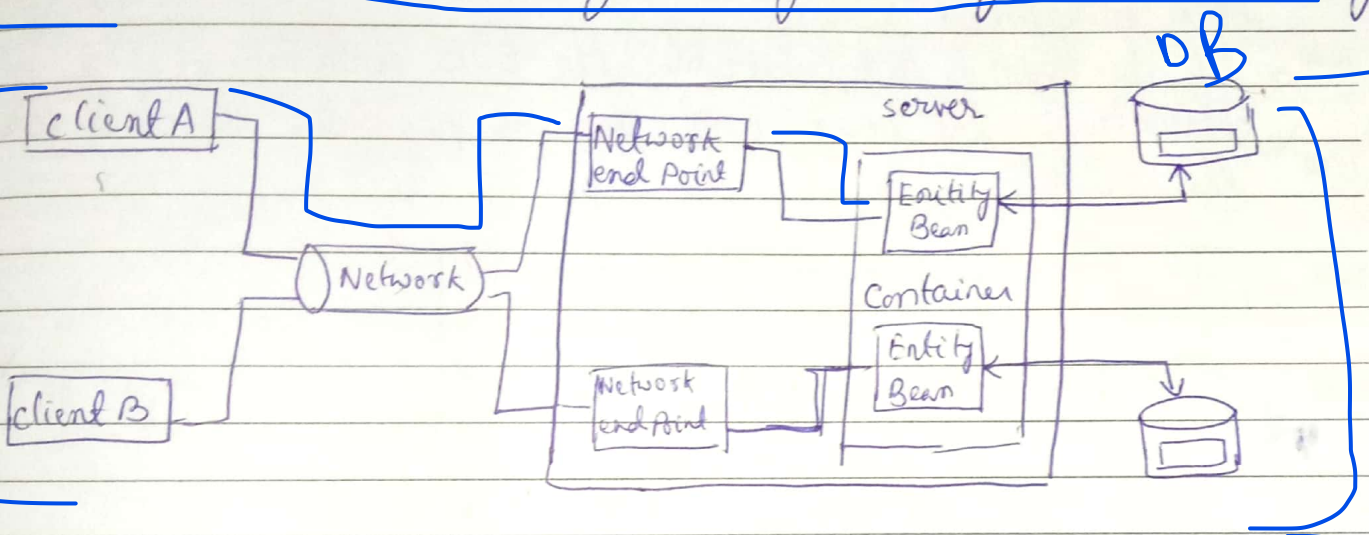
JUNE

WEDNESDAY 17

Week 20 / 137-228 May 2023

→ It can deal with various, needy, diligent articles in playing out its essential assignments.

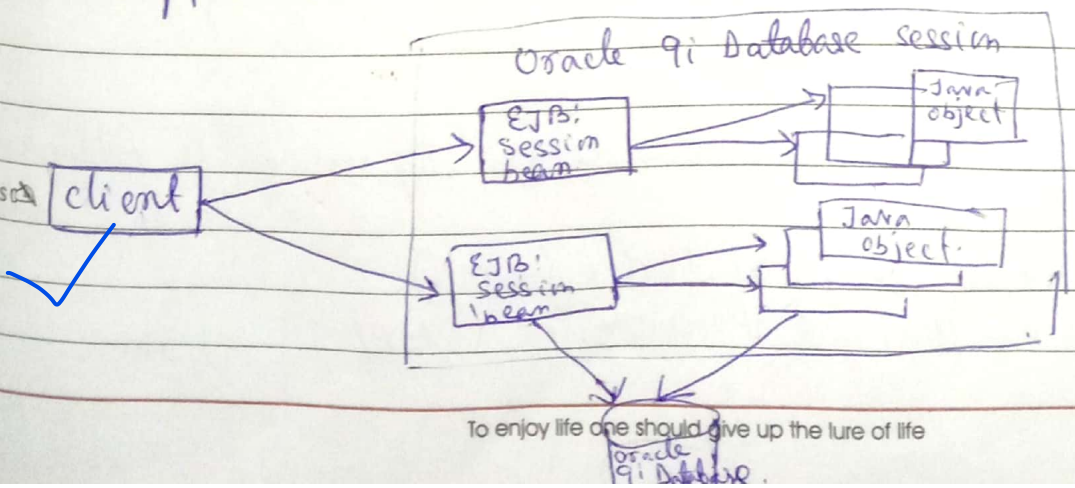
→ A substance bean is a far-off object that oversees steady information and performs complex business rationale. It can be extraordinarily distinguished by an essential key.



→ It is a far-off object that oversees steady information, performs complex business rationale, possibly utilizes a few ward Java protests, and can be remarkably distinguished by an essential key.

→ It ordinarily coarse-grained determined items since they use stably information put away inside a few fine-grained relentless Java objects.

→ Element beans are diligent on the grounds that they do endure a worker crash or an organizations disappointment



instantiates → to represent or be an example of something.

18 THURSDAY
May 2023 Week 20 / 138-227

MAY 2023

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

→ Every entity bean has a persistance identity that is associated with it. It means that it comprises a unique identity that can be fetched if we have a primary key.

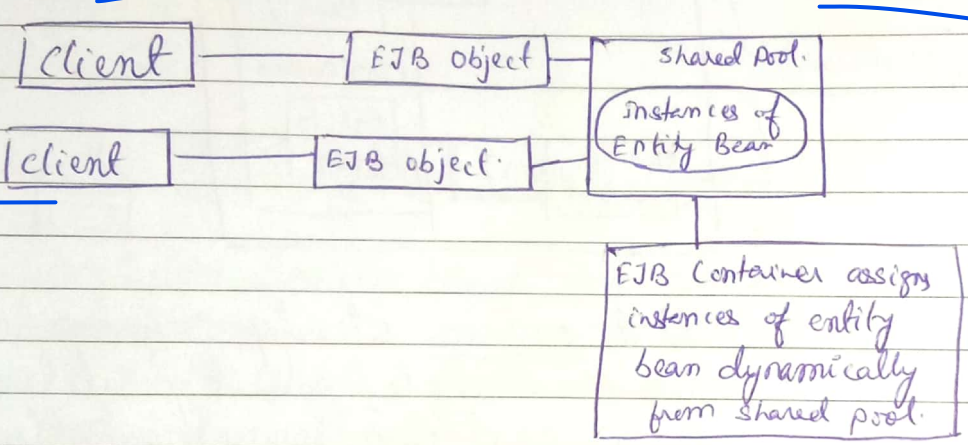
08.00 → The type for the unique key is defined by the bean provider.

09.00 → A client can retrieve the entity bean if the primary has been misplaced.

10.00 → If the bean is not available, the EJB container first, instantiates the bean and then re-populates the data for the client.

11.00

12.00



01.00

02.00

03.00

04.00 FIG: Pooling and Instances of Entity Bean

05.00 → The diligence for entity bean information is given both to saving state when the bean is passivated and for improving the state when a failover has detected.

06.00 → It can endure in light of the fact that the information is put away determinedly by the holder in some type of information stockpiling framework, like a data set.

EJB

→ Entity beans persist business data by using the following two methods:

Notes

- Bean-Managed Persistence (BMP)
- Container-Managed Persistence (CMP)

• Bean Managed Persistence (BMP)

- BMP is more complicated in comparison to container-managed persistence, since it allows us to write the persistence logic into the bean class, explicitly.
- In order to write the persistence handling code into the bean class, we must know the type of database that is being used and how fields of the bean class are mapped to the database. Therefore, it provides more flexibility between the database and the bean instance.
- It can be used as an alternative to CMP, if the deployment tools are incompatible for mapping the bean instance's state to the database.
- The disadvantages of BMP is that more work is required to define the bean and it ties the bean to a specific database type and structure.

• Container-Managed Persistence

- In CMP, the EJB container transparently and implicitly handles the relationship between the bean and the database.
- Bean developers focus on the data and the business process. The principal constraint is that the EJB component can most likely not produce information base access articulations with the proficiency of a programmer.

→ Unlike BMP, (CMP) does not allow us to write database access calls in the methods of the entity bean class.
→ It is because the persistence is handled by the container at (run-time).
→ The following two things are required to support the CMP.

• Mapping : It denotes that how to map an

entity bean to a resource.

- Runtime Environment: A CMP runtime environment that uses the mapping information to perform persistence operations on each bean.

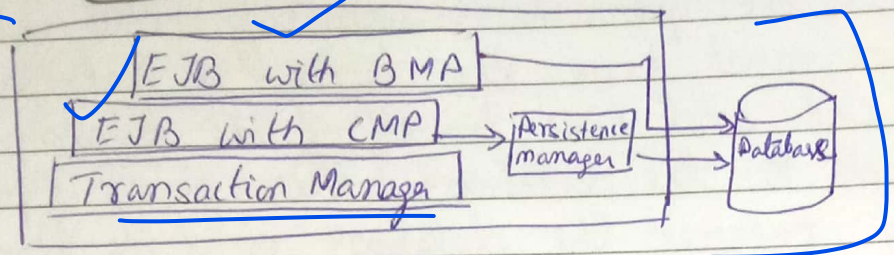


FIG: Entity Bean Flow

Note: Entity bean has been replaced with Java persistence API

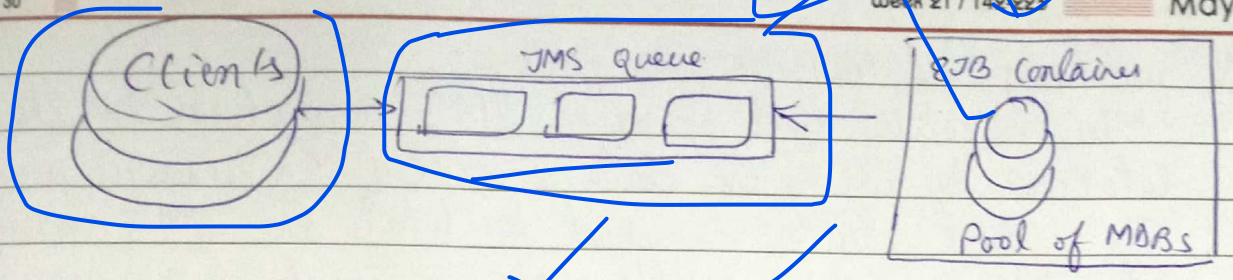
Message-Driven Bean (MDB)

→ MDB is a Java Messaging Service (message listener).
→ It consumes messages from a queue or subscription of a topic.
→ It provides an easy way to create or implement asynchronous communication using a JMS message listener.
→ An advantage of using MDB is that it allows us to use the asynchronous nature of a JMS listener.
→ We can implement message-drive beans with Oracle JMS

Sunday 21

→ There is an EJB container (contains MDBs pool) that handles the JMS queue and topic
→ Each incoming message is handled by a bean that is invoked by the container
→ No object invokes an MDB directly.
→ All invocation for MDB originates from the container.
→ When the container invokes the MDB it can invoke other EJBs or Java objects to continue the request.

Truth is always strange- stranger than fiction



It is quite similar to stateless session bean. Because it does not save informal state, also used to handle multiple incoming requests.

Java Message Server

EJB has the following benefits over the JMS are as follows:

- It creates a consumer for the listener. It means that the container creates QueueReceiver or TopicSubscriber
- MDB is registered with the consumer. It means that at deployment time QueueReceiver, TopicSubscriber and its factory are registered by the container.
- The message ACK mode is specified

The primary function of MDB is to read (receive) or write (send) incoming from a JMS destination (i.e. queue or topic).

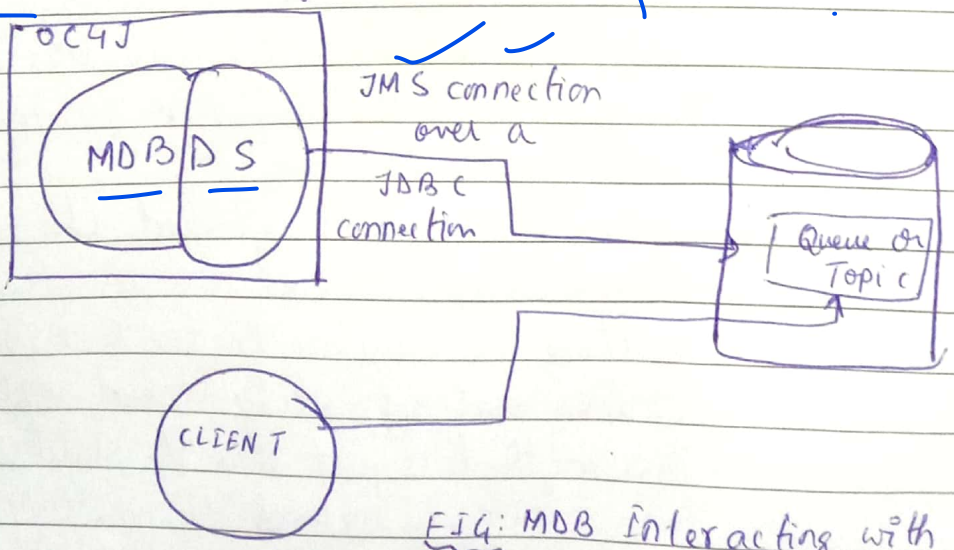


Fig: MDB Interacting with JMS Destination

23 TUESDAY
May 2023 Week 21 / 143-222

MAY 2023
S M T W T F S S M T W T F S
1 2 3 4 5 6 7 8 9 10 11 12 13
14 15 16 17 18 19 20 21 22 23 24 25 26 27
28 29 30 31

→ The working of MDB is as follows:
- MDB creates and opens a JMS connection to the database by using the data source (JMS resource provider). The JDBC driver is used facilitate the JMS connection.
- A JMS session over the JMS connection is opened by the MDB.
- If any message for the MDB is routed to the (onMessage()) method of the MDB from the queue. Other clients may also access the same queue and topic to put the message for the MDB.

* Note that it does not handle the requests that are requested by the clients instead it handles requests that are placed into a queue.

→ MDB implements the (javax.ejb.MessageDriverBean) interface and inherits the (javax.jms.MessageListener) class that provides the following methods:

Method	Description
(i) onMessage(msg)	- The container dequeues a message from the JMS queue associated with this MDB and gives it to this instance by invoking this method
(ii) setMessageDrivenContext(ctx)	- After the bean is created, the setMessageDrivenContext method is invoked. This method is similar to the EJB setSessionContext and setEntityContext methods.
(iii) ejbCreate()	- This method is used just like the stateless session bean ejbCreate method. No initialization should be done in this method.

Notes

There is no instinct like that of the heart

(iv) ejbRemove()

However, any resources that you allocate within this method will exist for this object.
- Delete any resources allocated within the ejbCreate method.

Difference Between Session and Entity Bean

	Session Bean	Entity Bean
(1) Primary Key	It has no primary key. It used to identify and retriere specific bean instances.	It has a primary key that allows us to find instances and can be shared by more than one client.
(2) Stateless/ Stateful	It may be stateful or stateless	It is stateful
(3) Span	It is relatively short-lived	It is relatively long-lived.
(4) Performance	It represents a single conversation with a client.	It encapsulates persistent business data.
(5) Accessibility	It is created and used by a single client.	It may be shared by multiple clients.
(6) Recovery	It is not recoverable in case if EJB server fails. So, it may be destroyed.	It is recoverable if any failure has occured.
(7) Persistence of Data	It persists data only for the span of the conversation with the client.	Persistence beyond the life of a client instance. Persistence can be container-managed or bean-managed.
(8) Remote Interface	It extends javax.ejb.EJBObject	It extends javax.ejb.EJBObject
(9) Home Interface	It extends javax.ejb.EJBHome	It extends javax.ejb.EJBHome
(10) Bean Class	It extends javax.ejb.EJBSessionBean	It extends javax.ejb.EJBEntityBean

25 THURSDAY
May 2023 Week 21 / 145-220

Code

Stateless Session Bean

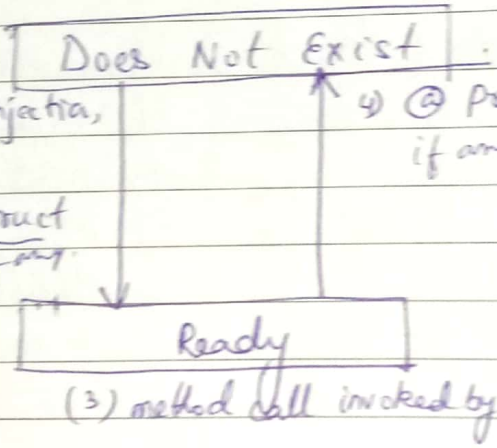
→ Annotations used in stateless Session Bean

3 imp. annotations used in stateless Session Bean:

(i) @Stateless
(ii) @PostConstruct
(iii) @PreDestroy.

→ Life cycle of stateless Session Bean:

2 states → Does Not Exist
→ Ready.



→ Example of Stateless Session Bean

Notes: To create EJB appl., need to create bean component and bean client.

(i) Create stateless bean component

To create the stateless bean component, need to create a remote interface and a bean class.

* File: AdderImplRemote.java

```
package com.javatpoint;
import javax.ejb.Remote;

@Remote
public interface AdderImplRemote {
  int add(int a, int b);
}
```

* File: AdderImpl.java

```
package com.javatpoint;
import javax.ejb.Stateless;

@Stateless(mappedName = "st1")
public class AdderImpl implements AdderImplRemote {
  public int add(int a, int b) {
    return a+b;
  }
}
```

(ii) Create stateless bean client

The stateless bean client may be local, remote or webservice client. Here, we are going to create remote client. It is console based application. Here we are not using dependency injection. The

dependency injection can be used with web based client only

08.00 File: AdderImpl.java

```
09.00 package com.javatpoint;
      import javax.naming.Context;
10.00 import javax.naming.InitialContext;

11.00 public class Test{
      public static void main(String[] args) throws Exception{
12.00
        Context context = new InitialContext();
01.00   AdderImplRemote remote = (AdderImplRemote)context.
                                  lookup("st1");
02.00   System.out.println(remote.add(32,32));
      }
03.00 }
```

04.00 Output: 64

05.00

06.00 Stateful Session Bean

Sunday 28

EVE It is a business object that represents business logic like stateless session bean. But, it maintains state (data). In other words, conversational state between multiple method calls is maintained by the container in stateful session bean.

Notes

Annotations used in Stateful Session Bean.

1) @Stateful

(2) @PostConstruct
(3) @PreDestroy
(4) @PrePassivate
(5) @PostActivate

→ Note: Annotations in Java provide additional information to the compiler and JVM

Tag representations
Metadata about Class, Interface variables

Example of Stateful Session Bean

We need to create bean component and bean client for creating session bean application.

(1) Create stateful bean component

Let's create a remote interface and a bean class for developing a stateful bean component.

File: BankRemote.java

```
package com.javatpoint;                    // user-defined package
import javax.ejb.Remote;

public interface BankRemote {
boolean withdraw(int amount);
void deposit(int amount);
int getBalance();
}
```

File: Bank.java

```
package com.javatpoint;
import javax.ejb.Stateful;

@Stateful(mappedName = "stateful123")
public class Bank implements BankRemote {
```

```
private int amount = 0
public boolean withdraw(int amount) {
    if(amount <= this.amount){
        this.amount -= amount;
        return true;
    }else{
        return false;
    }
}
public void deposit (int amount){
    this.amount += amount;
}
public int getBalance(){
    return amount;
}
}
```

(2) Create stateful bean client

The stateful bean client may be local, remote or web service client. Here, we are going to create web client and not using dependency injection.

- File: index.jsp

```
<a href = "OpenAccount" > Open Account </a>
```

* File: operation.jsp

```
<form action = "operationprocess.jsp">
```

Enter Amount <input type="text" name="amount" />

Choose Operation:
Deposit <input type="radio" name="operation" value="deposit" />
Withdraw <input type="radio" name="operation" value="withdraw" />
Check balance <input type="radio" name="operation" value="checkbalance" />

<input type="submit" value="submit">
</form>

* File: operationprocess.jsp

<%@page import="com.javatpoint.*" %>
<%
BankRemote remote=(BankRemote) session.getAttribute("remote");
String operation = request.getParameter("operation");
String amount = request.getParameter("amount");

if(operation!=null){
if(operation.equals("deposit")){
remote.deposit(Integer.parseInt(amount));
out.print("Amount successfully deposited!");
}else
if(operation.equals("withdraw")){
boolean status = remote.withdraw(Integer.parseInt(amount));
if(status){
out.print("Amount successfully withdrawn!")
}

```
else{
    out.println("Enter less amount");
}
}else{
    out.println("Current Amount: "+remote.getBalance());
}
}
%>
<hr/>
<jsp:include page="operation.jsp"></jsp:include>
```

* File: OpenAccount.java

```
package com.javatpoint;
import java.io.IOException;
import javax.ejb.EJB;
import javax.naming.InitialContext;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/OpenAccount")
public class OpenAccount extends HttpServlet{
//@EJB(mappedName="stateful123")
//BankRemote b;
protected void doGet(HttpServletRequest request, HttpServletResponse response)
```

Notes

```
throws ServletException, IOException{
try{
InitialContext context = new InitialContext();
BankRemote b = (BankRemote) context.
lookup("stateful123");

request.getSession().setAttribute("remote", b);
request.getRequestDispatcher("/operation.jsp").forward
(request, response);

}catch(Exception e){System.out.println(e);}
}
}
```

08.00
09.00
10.00
11.00
12.00
01.00
02.00
03.00
04.00
05.00
06.00
EVE

Notes

- POJO (full form in Java: Plain Old Java Object) defines an object in Java that doesn't require a classpath. It doesn't have any references to any particular framework.

An Entity Bean Example:-

A Java Persistence Example.

Java Persistence API is the standard API used for the management of the persistent data and object/relational mapping. Java Persistence API is added in Java EE 5 platform. This API is a lightweight framework based on POJO for object-relational mapping.

Entity:

- Entity.
  - An entity can be considered as a lightweight persistence domain object.
  - An entity defines a table in a relational database and each instance of an entity corresponds to a row in that table.
  - An entity refers to a logical collection of data that can be stored or retrieved as a whole.
  - For example, in a banking application, Customer and BankAccount can be treated as entities.
  - Customer name, customer address etc. can be logically grouped together for representing a Customer entity. Similarly account number, total balance etc may be logically grouped under BankAccount entity.

- Entity Beans
  - Entity Beans are enterprises beans, which represent persistent data stored in a storage medium, such as relational database. An entity bean persists across multiple session and can be accessed by multiple clients.
  - An entity bean acts as an intermediary between a client

A POJO class must contain @Id annotation and each object must have a unique Id.

JULY 2023

S	M	T	W	T	F	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

(24)

THURSDAY 15

Week 24 / 166-199 Jun 2023

and a database.

- When a client wants to perform a transaction, the information regarding their specific account is loaded into an entity bean instance from the database.
- Operations are performed on the data present in the instance and updated in the bank's database at regular intervals.

- **Primary key generation:**

- EJB entities are POJOs.
- In EJB 3.0, a primary key is used with @Id annotation. Depending upon the application requirement, Id annotation can be used with diff. primary key generation strategies defined by GeneratorType enum. The GeneratorTypes are TABLE, SEQUENCE, IDENTITY, AUTO, and NONE.

- **Declaring an entity bean:**

Every bound persistent POJO class is an entity bean and is declared using the @Entity annotation (at the class level):

```
@Entity
public class Book implements
Serializable { Long empid;
@Id
@GeneratedValue
public Long getId()
{ return id; }
public void setId(Long id)
```

→

```
{ this.id = id; }
}
```

Wisdom is better than strength

@Entity declares the class as an entity bean (i.e a persistent POJO class), which tells the EJB3 container that this class needs to be mapped to a relational database table.

@Id declares the identifier property of this entity bean.

@GeneratedValue annotation indicates that the server automatically generates the primary key value.

The class Book is mapped to the Book table, using the column id as its primary key column.

Defining the table:

```
@Entity
@Table(name = "book").
@SequenceGenerator(name= "book_sequence",
    sequenceName = "book_id_seq") public
class Book implements Serializable {
```

@Table -> defines the table name.

@SequenceGenerator -> defines a sequence generator. A sequence is a database feature which returns the next Integer or long value each time it is called.

Notes

Managing Entities:

08.00 - The entity manager manages entities.
- It is represented by javax.persistence.EntityManager

09.00 The BookBank e

10.00

Example of developing a BookBank JEE application

11.00

There are following steps that you have to follow to develop
12.00 a book bank JEE application.

01.00 1. Create Remote business interface:
2. BookCatalogInterface
02.00 2. Implement the Annotated Session Bean:
BookCatalogBean
03.00 3. Create the Entity bean: BookBank
4. Create the web client: WebClient
04.00 5. Deploy book on the server.
6. Run web client on the web browser.

05.00

The BookBank entity bean class:

06.00 Sunday 18

In the Book catalog example, we define a Book entity
EVE bean class. The bean has three properties (title, author and price) to model a Book product. The id property is used to uniquely identify the Book bean instance by the EJB3 container. The id value is automatically generated
Notes when the bean is saved to the database.

The code for the Book Bean:

(21)

```
package entity library;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import java.util.Collection;
import javax.persistence.*;
import java.io.Serializable;

@Entity
@Table(name = "bookbank")
public class BookBank implements Serializable {

    long id;
    String title;
    String author;
    double price;

    public BookBank() {
    super();
    }

public BookBank(String title, String author, double price){
    super();
    this.title = title;
    this.author = author;
    this.price = price;
    }
```

Notes

@Id.
@GeneratedValue(strategy = GenerationType.AUTO)

08.00

// Getter and setter methods for the defined properties.

09.00

```
public long getId() {
    return id;
}
```

10.00

11.00

12.00

```
public void setId(long id) {
    this.id = id;
}
```

01.00

02.00

```
public String getTitle() {
    return title;
}
```

03.00

04.00

```
public void setTitle(String title) {
    this.title = title;
}
```

05.00

06.00

```
public String getAuthor() {
    return author;
}
```

EVE

```
public void setAuthor(String author) {
    this.author = author;
}
public double getPrice() {
    return price;
}
```

Notes

21 WEDNESDAY

Jun 2023 Week 25 / 172-193

JUNE 2023

S	M	T	W	T	F	S	S	M	T	W	T	F	S
				1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20	21	22	23	24
25	26	27	28	29	30								

```
public void setPrice(double price) {
    this.price = price;
}
```

08.00

09.00

10.00

11.00

12 00

01.00

02.00

03.00

04.00

05.00

06.00

EVE

Notes

You are judged by the company you keep

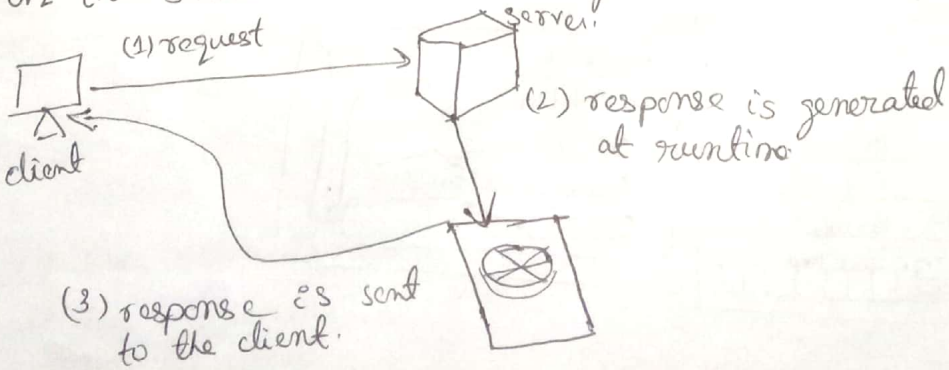
Servlets (small Java program that runs within a web server)

- Servlet technology is used to create a web application (resides at server side and generates a dynamic web page).
- Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language. However, there were many disadvantages to this technology. We have discussed these disadvantages ~~before~~ later.
- There are many interfaces and classes in the servlet API such as servlet such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse etc.

What is a Servlet?

Servlet can be described in many ways, depending on the context.

- Servlet is a technology which is used to create a web application.
- Servlet is an API that provides many interfaces and classes including documentation.
- Servlet is an interface that must be implemented for creating any ~~Servlet~~.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.

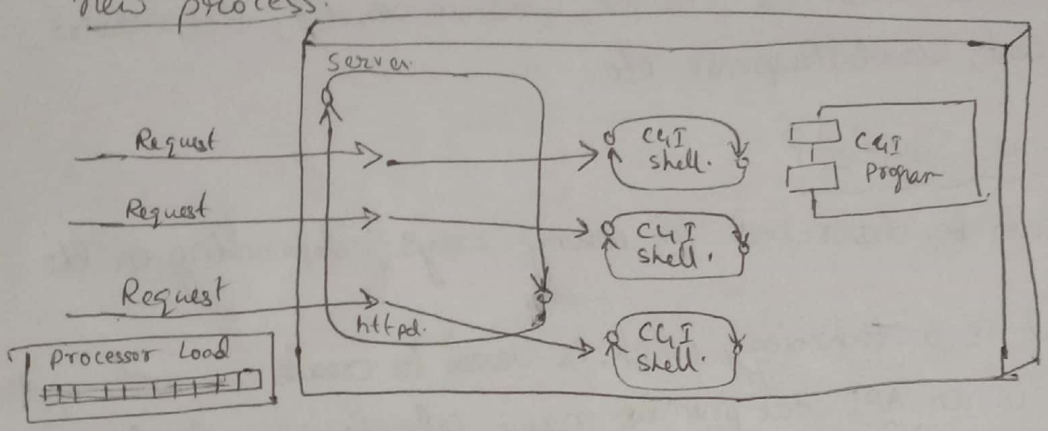


What is a web application?

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request

-CGI(Common Gateway Interface)

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.



-Disadvantages of CGI

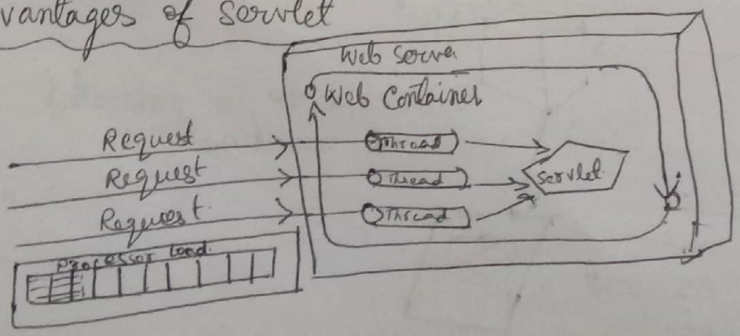
There are many problems in CGI technology:

(i) If the number of clients increases, it takes more time for sending the response.

(ii) For each request, it starts a process, and the web server is limited to start processes.

(iii) It use platform dependent language e.g. C, C++, perl.

-Advantages of Servlet



- There are many advantages of servlet over CGI. The web container creates threads for handling the multiple requests to the servlet. Threads have many benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The advantages of servlet are as follows:

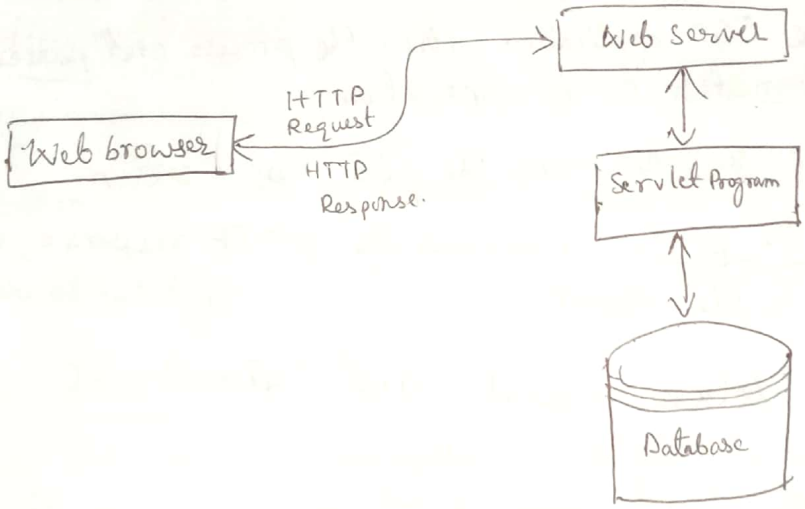
(i) Better performance: because it creates a thread for each request, not process.

(ii) Portability: because it uses Java language.

(iii) Robust: JVM manages servlets, so we don't need to worry about the memory leak, garbage collection, etc.

(iv) Secure: because it uses Java language.

* Servlet Architecture



* Execution of servlets basically involves 6 basic steps:-

(i) The client send the request to the webserver

(ii) The web server receives the request.

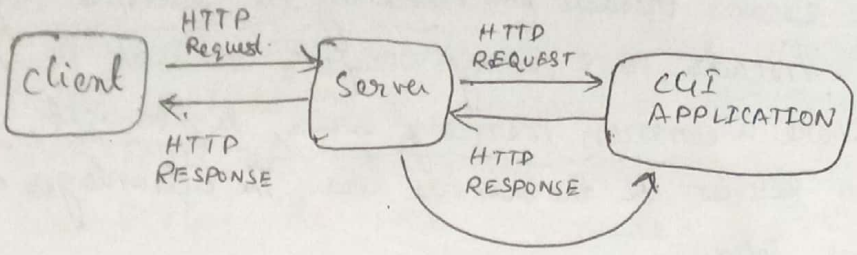
(iii) The web server passes the request to the corresponding servlet.

(iv) The servlet processes the request and generates the response in the form of output.

(v) The servlet sends the response back to the webserver

(vi) The web server sends the response back to the client and the client browser displays it on the screen.

* CGI processing a clients request



CGI is actually an external application, which is responsible for processing client requests and generating dynamic content In CGI application, when a client makes a request to access dynamic Web pages, the web server performs the following operations.

(i) It first locates the requested web page i.e. the required CGI application using URL.

(ii) It then creates a new process to service the clients request.

(iii) Invokes the CGI application within the process and passes the request information to the application.

(iv) Collects the response from the CGI application.

(v) Destroys the process, prepares the HTTP response, and sends it to the client.

* Difference between Servlet and CGI.

Servlet	CGI
(i) Servlets are portable and efficient	(i) CGI is not portable.
(ii) In servlets, sharing data is possible	(ii) In CGI, sharing data is not possible
(iii) Servlets can directly communicate with the webserver	(iii) CGI can't directly communicate with the webserver
(iv) Servlets are less expensive than CGI	(iv) CGI is more expensive than servlets.
(v) Servlets can handle the cookies	(v) CGI cannot handle the cookies. (cookies are the textual information that is stored in key-value pair format to the clients browsers)

* Servlets API's: (You need to use servlet API to create servlets)

Servlets are built from two packages:
-> javax.servlet (Basic) - (contains interfaces & classes that are used by the servlet or web container)
-> javax.servlet.http (Advance) - (contain interfaces & classes that are responsible for http requests only).

-> Various classes and interfaces present in these packages are:-

Component	Type	Package
(i) Servlet	Interface	javax.servlet.*
(ii) ServletRequest	Interface	javax.servlet.*
(iii) ServletResponse	Interface	javax.servlet.*
(iv) GenericServlet	class	javax.servlet.*
(v) Httpservlet	Class	javax.servlet.http.*
(vi) HttpServletRequest	Interface	javax.servlet.http.*
(vii) HttpServletResponse	Interface	javax.servlet.http.*
(viii) Filter	Interface	javax.servlet.*
(ix) ServletConfig	Interface	javax.servlet.*

* Servlet Interface

- Servlet Interface provides common behaviour to all the servlets.
- Servlet interface defines methods that all servlets must implement.
- Servlets interface needs to be implemented for creating any servlet (either directly or indirectly)
- It provides 3 life cycle methods :-
  (i) that are used to initialize the servlet
  (ii) to service the requests
  (iii) to destroy the servlet
- It provides 2 non-life cycle methods.
  (i) returns the object
  (ii) returns information about servlet.

* Methods of Servlet Interface

- There are 5 method in Servlet interface. Init, service and destroy are the life cycle methods of servlet. These are invoked by the web container.

Method	Description
- public void init(ServletConfig config)	- initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
- public void service(ServletRequest request, ServletResponse response).	- provides response for the incoming request. It is invoked at each request by the web container.
- public void destroy()	- is invoked only once and indicates that servlet is being destroyed.
- public ServletConfig getServletConfig()	- returns the object of ServletConfig.
- public String getServletInfo()	- returns information about servlet such as writer, copyright, version etc.

* Servlet example by implementing Servlet Interface

```
import java.io.*;
import javax.servlet.*;
```

```
public class First implements Servlet{
   ServletConfig config=null;

public void init(ServletConfig config){
this.config=config;
System.out.println("Servlet is initialized");
}

public void service(ServletRequest req, ServletResponse res)
throws IOException, ServletException{

res.setContentType("text/html");

PrintWriter out=res.getWriter();
 out.print("<html><body>");
 out.print("<b>hello simple servlet</b>");
 out.print("</body></html>");

}
public void destroy()
{System.out.println("servlet is destroyed");}

public ServletConfig getServletConfig()
 {return config;}
public String getServletInfo()
 {return "copyright 2007-1010";}

}
```

Lifecycle of servlets

Web container

* A web container is responsible for managing the lifecycle of servlets, mapping a URL to a particular servlet and ensuring that the URL requester has the correct access rights.
  - ↳ It handles requests to servlets, ~~Jakarta Server~~ JSP files, and other types of files that include server-side code.
  - ↳ Examples of web containers include Tomcat, Glassfish, JBoss Application Server, or Wildfly.
  - ↳ The benefit of using a web container is that there are fewer applications to maintain and configure.

* Containers are the interface between a component and the low-level, platform specific functionality that supports the component.

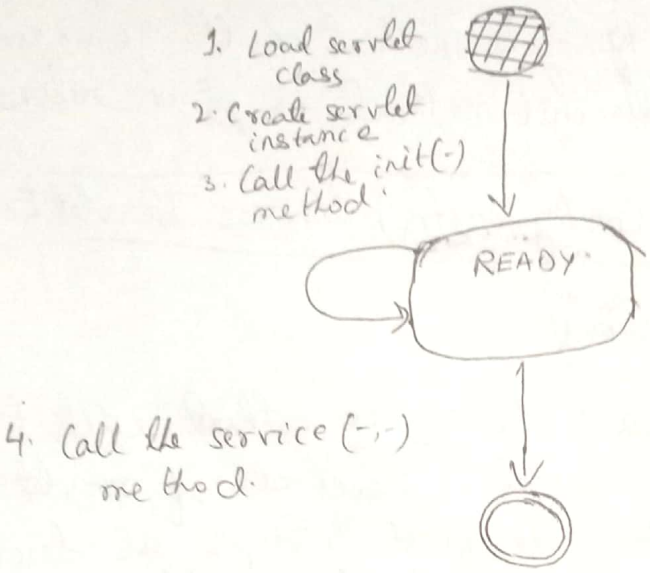
* Is Tomcat an Application Server or a Web Server?
  Tomcat is considered a web server instead of an application server because it functions as a web server and Servlet Container.

* When to use container in Java?
  - A container can store instances of any Java class. For eg: we can store a String, a Date, and an Integer in the same container. ~~When you retrieve an~~
  - When you retrieve an object, you must cast the object back to the required type before applying a type-specific method.

* EJBs can be deployed only on Application servers, but not on web containers.

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet.

(1) Servlet class is loaded
(2) servlet instance is created.
(3) init method is invoked
(4) service method is invoked.
(5) destroy method is invoked.



As displayed in the above diagram, there are 3 states of a servlet: new, ready and end.
The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

(1) Servlet class is loaded

The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

(2) Servlet instance is created

The web container creates the instance of a servlet after loading the servlet class. The servlet is created only once in the servlet life cycle.

(3) init method is invoked

The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:

```
public void init(ServletConfig config) throws ServletException
```

(4) service method is invoked

The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first 3 steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

```
public void service(ServletRequest request, ServletResponse response)
    throws ServletException, IOException.
```

(5) destroy method is invoked

The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the servlet interface is given below:

```
public void destroy()
```

Deployment Directory

`http://localhost:8082/DisplayServlet`

↳ Tomcat server running on port no. 8082

input.html (part of your web content in eclipse)

```
<form action="xyz" method="post">
   Name <input type="text"
               name="name">
<input type="submit"          ="Display">
</form>
```

(1) control goes to web.xml

[form has two attributes as (i) action (ii) method]

[(i) action is on URL]

[After we click "submit", then control goes to "action"]

DisplayServlet.java (From Java src (source code) ↳ test (package test; public class DisplayServlet) ↳ DisplayServlet)

(3) control goes to web.xml

```
public void doPost(... req, ... res)
   ---
   ---
   ---
```

(4) Now the control goes to DisplayServlet

web.xml (part of your WEB-INF in eclipse)

```
<web.app>
   <servlet>
      <Servlet name> ds </Servlet name>
      <servlet.class> test.DisplayServlet </servlet class>
   </servlet>
   <servlet.mapping>
      <servlet name> ds </servlet.name>
      <url-pattern> /xyz </url-pattern>
   </servlet mapping>
```

3 servlet name is matched with servlet tag

2 after the control comes here

(2) control goes to input.html

```
   <welcome.file list>
      <welcome.file> input.html </welcome.file>
   </welcome.file.list>
</web.app>
```

(4) Whenever we call DisplayServlet.java file, then it is loaded for execution (and also automatically instantiated which is part of the Servlet life cycle

- Two types of request from web browser
  - (i) GET request
  - (ii) POST request

So after the post request is invoked on input.html, the control goes to DisplayServlet.java file and instantiation starts.

Life cycle of servlet

(i) Loading
(ii) Instantiation
(iii) Initialization process
(iv) Request processing
(v) Destroy process

`http: localhost/8082/DisplayServlet` ①

web.xml

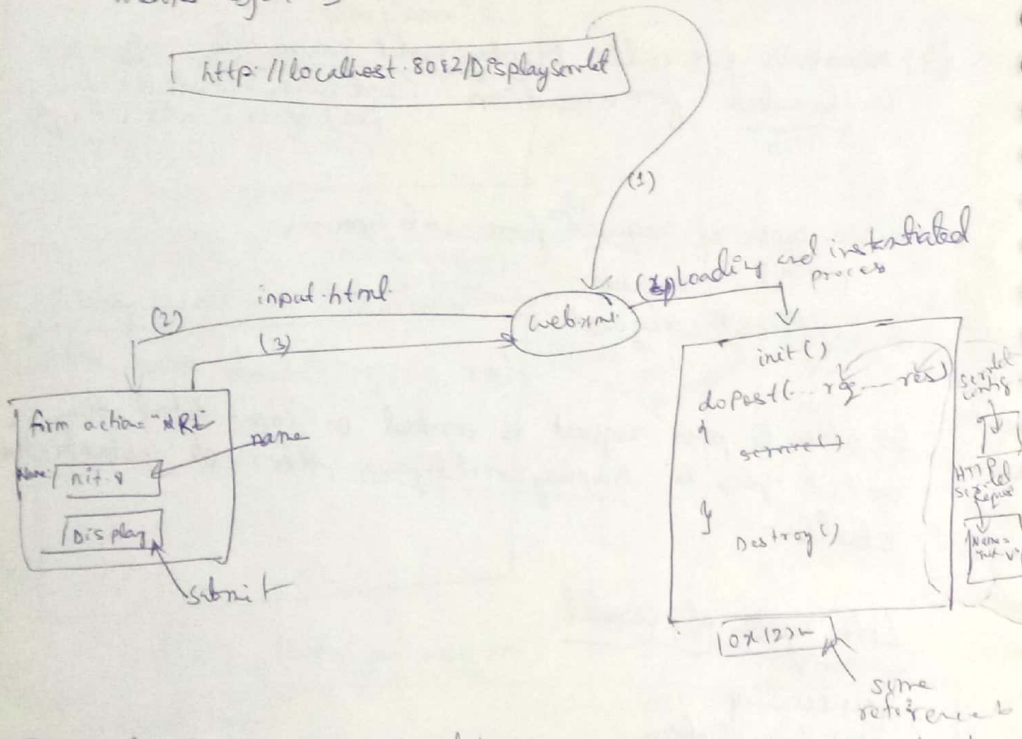
② form action = "xyz"
Name: [ ] name
[Display] ← submit

④ So the control comes here and we enter something in the 'Name' box

⑤ So after we submit, using this url the control goes to web.xml

③ from web.xml, input.html file is identified through welcome file list, in that input.html is invoked and it is displayed

③ continued -- In web.xml servlet mapping is there from where URL pattern will is identified, so servlet is delected, so servlet program is loaded and this is known as loading process. ④ After the servlet program is loaded automatically instantiated so instantiation process means object is created



④ Continued -- This 3 init(), service(), & destroy() are called as life cycle methods.

So, while instantiation service is going on one servlet config object is created to hold configuration details but in the place of service we're using "doPost" method, That "doPost" method is passed with 2 parameters, one is HTTPServletRequest and HTTPServletResponse

javax.servlet.Servlet (Interface)
↓ implemented by
javax.servlet.GenericServlet (Abstract class)
↓ extended to
javax.servlet.HttpServlet
↓ javax.servlet.http

```
package test;
import javax.servlet.*;
import javax.servlet.http.*;
public class DisplayServlet extends HttpServlet{
    public void doPost(){
        (continued in next to next page)
}
```

doPost → javax.servlet.http

⇒ Through the URL given in the input.html we're going to trigger the execution of servlet program i.e attribute assigned with the method 'post', so the web browser whenever passes request to servlet program, two types of request are processed i.e GET & POST request

→ If we don't use 'POST' method then by default it becomes 'GET' request

This references are passed as parameters.

So, whenever it is instantiated the process executes, then first it calls init() and initializing any variables or instance variables, so initialization process is completed

(A) After that control goes to doPost() method in the process of accepting the request. So whenever this doPost() method is executed we called it as "request processing"

(ii) So when doPost() method will be executed "init.v" (data from form) is written in HTTPServletRequest object. So whenever we need to get that data then "req.getParameter" and display it using "PrintWriter pw = res.getWriter()"

SERVLETS- FORM DATA

You must have come across many situations when you need to pass some information from your browser to web server and ultimately to your backend program. The browser uses two methods to pass this information to web server. These methods are **GET Method** and **POST Method.**

GET Method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? (question mark) symbol as follows –

```
http://www.test.com/hello?key1=value1&key2=value2
```

The GET method is the default method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use the GET method if you have password or other sensitive information to pass to the server. The GET method has size limitation: only 1024 characters can be used in a request string.

This information is passed using QUERY_STRING header and will be accessible through QUERY_STRING environment variable and Servlet handles this type of requests using **doGet()** method.

POST Method

A generally more reliable method of passing information to a backend program is the POST method. This packages the information in exactly the same way as GET method, but instead of sending it as a text string after a ? (question mark) in the URL it sends it as a separate message. This message comes to the backend program in the form of the standard input which you can parse and use for your processing. Servlet handles this type of requests using doPost() method.

Reading Form Data using Servlet

Servlets handles form data parsing automatically using the following methods depending on the situation –

getParameter() – You call request.getParameter() method to get the value of a form parameter.

getParameterValues() – Call this method if the parameter appears more than once and returns multiple values, for example checkbox.

getParameterNames() – Call this method if you want a complete list of all parameters in the current request.

GET Method Example using URL

Here is a simple URL which will pass two values to **HelloForm** program using GET method.

```
http://localhost:8080/HelloForm?first_name = ZARA&last_name = ALI
```

Given below is the **HelloForm.java** servlet program to handle input given by web browser. We are going to use **getParameter()** method which makes it very easy to access passed information –

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloForm extends HttpServlet {

  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Set response content type
    response.setContentType("text/html");

    PrintWriter out = response.getWriter();
```

```
    String title = "Using GET Method to Read Form Data";
    String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";

    out.println(docType +
      "<html>\n" +
        "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor = \"#f0f0f0\">\n" +
          "<h1 align = \"center\">" + title + "</h1>\n" +
          "<ul>\n" +
            "  <li><b>First Name</b>: "
            + request.getParameter("first_name") + "\n" +
            "  <li><b>Last Name</b>: "
            + request.getParameter("last_name") + "\n" +
          "</ul>\n" +
        "</body>" +
      "</html>"
    );
  }
}
```

Assuming your environment is set up properly, compile **HelloForm.java** as follows −

```
$ javac HelloForm.java
```

If everything goes fine, above compilation would produce **HelloForm.class** file. Next you would have to copy this class file in <Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes and create following entries in **web.xml** file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

```
<servlet>
  <servlet-name>HelloForm</servlet-name>
  <servlet-class>HelloForm</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>HelloForm</servlet-name>
```

```
  <url-pattern>/HelloForm</url-pattern>
</servlet-mapping>
```

Now type http://localhost:8080/HelloForm?first_name=ZARA&last_name=ALI in your browser's Location:box and make sure you already started tomcat server, before firing above command in the browser. This would generate following result –

Using GET Method to Read Form Data

First Name: ZARA
Last Name: ALI

GET Method Example Using Form

Here is a simple example which passes two values using HTML FORM and submit button. We are going to use same Servlet **HelloForm** to handle this input.

```
<html>
  <body>
    <form action = "HelloForm" method = "GET">
      First Name: <input type = "text" name = "first_name">
      <br />
      Last Name: <input type = "text" name = "last_name" />
      <input type = "submit" value = "Submit" />
    </form>
  </body>
</html>
```

Keep this HTML in a file Hello.htm and put it in <Tomcat-installationdirectory>/webapps/ROOT directory. When you would access http://localhost:8080/Hello.htm, here is the actual output of the above form.

First Name:

Last Name:

SUBMIT

Enter First Name and Last Name and then click submit button to see the result on your local machine where tomcat is running. Based on the input provided, it will generate similar result as mentioned in the above example.

POST Method Example Using Form

Let us do little modification in the above servlet, so that it can handle GET as well as POST methods. Below is HelloForm.java servlet program to handle input given by web browser using GET or POST methods.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloForm extends HttpServlet {

  // Method to handle GET method request.
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Set response content type
    response.setContentType("text/html");

    PrintWriter out = response.getWriter();
    String title = "Using GET Method to Read Form Data";
    String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";

    out.println(docType +
      "<html>\n" +
        "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor = \"#f0f0f0\">\n" +
          "<h1 align = \"center\">" + title + "</h1>\n" +
```

```
            "<ul>\n" +
              "  <li><b>First Name</b>: "
              + request.getParameter("first_name") + "\n" +
              "  <li><b>Last Name</b>: "
              + request.getParameter("last_name") + "\n" +
            "</ul>\n" +
          "</body>"
        "</html>"
    );
  }

  // Method to handle POST method request.
  public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    doGet(request, response);
  }
}
```

Now compile and deploy the above Servlet and test it using Hello.htm with the POST method as follows −

```
<html>
  <body>
    <form action = "HelloForm" method = "POST">
      First Name: <input type = "text" name = "first_name">
      <br />
      Last Name: <input type = "text" name = "last_name" />
      <input type = "submit" value = "Submit" />
    </form>
  </body>
</html>
```

Here is the actual output of the above form, Try to enter First and Last Name and then click submit button to see the result on your local machine where tomcat is running.

First Name:

Last Name:

SUBMIT

Based on the input provided, it would generate similar result as mentioned in the above examples.

Passing Checkbox Data to Servlet Program

Checkboxes are used when more than one option is required to be selected.

Here is example HTML code, CheckBox.htm, for a form with two checkboxes

```
<html>
  <body>
    <form action = "CheckBox" method = "POST" target = "_blank">
      <input type = "checkbox" name = "maths" checked = "checked" /> Maths
      <input type = "checkbox" name = "physics"  /> Physics
      <input type = "checkbox" name = "chemistry" checked = "checked" />
                          Chemistry
      <input type = "submit" value = "Select Subject" />
    </form>
  </body>
</html>
```

The result of this code is the following form

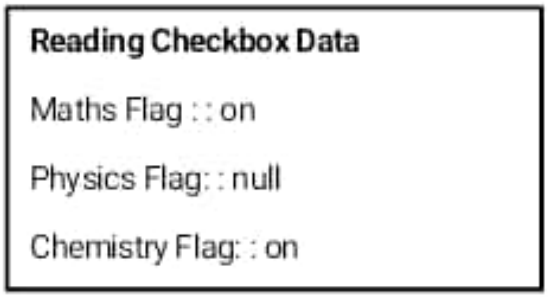
Given below is the CheckBox.java servlet program to handle input given by web browser for checkbox button.

```
// Import required java libraries
import java.io.*;
```

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

// Extend HttpServlet class
public class CheckBox extends HttpServlet {

  // Method to handle GET method request.
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Set response content type
    response.setContentType("text/html");

    PrintWriter out = response.getWriter();
    String title = "Reading Checkbox Data";
    String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";

    out.println(docType +
      "<html>\n" +
        "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor = \"#f0f0f0\">\n" +
          "<h1 align = \"center\">" + title + "</h1>\n" +
          "<ul>\n" +
            "  <li><b>Maths Flag : </b>: "
            + request.getParameter("maths") + "\n" +
            "  <li><b>Physics Flag: </b>: "
            + request.getParameter("physics") + "\n" +
            "  <li><b>Chemistry Flag: </b>: "
            + request.getParameter("chemistry") + "\n" +
          "</ul>\n" +
        "</body>"
      "</html>"
    );
  }

  // Method to handle POST method request.
  public void doPost(HttpServletRequest request, HttpServletResponse response)
```

```
    throws ServletException, IOException {

    doGet(request, response);
  }
}
```

For the above example, it would display following result −



Reading All Form Parameters

Following is the generic example which uses **getParameterNames()** method of HttpServletRequest to read all the available form parameters. This method returns an Enumeration that contains the parameter names in an unspecified order

Once we have an Enumeration, we can loop down the Enumeration in standard way by, using hasMoreElements() method to determine when to stop and using nextElement() method to get each parameter name.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// Extend HttpServlet class
public class ReadParams extends HttpServlet {

  // Method to handle GET method request.
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Set response content type
```

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

PrintWriter out = response.getWriter();
String title = "Reading All Form Parameters";
String docType =
  "<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";

out.println(docType +
  "<html>\n" +
  "<head><title>" + title + "</title></head>\n" +
  "<body bgcolor = \"#f0f0f0\">\n" +
  "<h1 align = \"center\">" + title + "</h1>\n" +
  "<table width = \"100%\" border = \"1\" align = \"center\">\n" +

  "<tr bgcolor = \"#949494\">\n" +
    "<th>Param Name</th>"
    "<th>Param Value(s)</th>\n"+
  "</tr>\n"
);
Enumeration paramNames = request.getParameterNames();

while(paramNames.hasMoreElements()) {
  String paramName = (String)paramNames.nextElement();
  out.print("<tr><td>" + paramName + "</td>\n<td>");
  String[] paramValues = request.getParameterValues(paramName);

  // Read single valued data
  if (paramValues.length == 1) {
    String paramValue = paramValues[0];
    if (paramValue.length() == 0)
      out.println("<i>No Value</i>");
      else
      out.println(paramValue);
  } else {
    // Read multiple valued data
    out.println("<ul>");

    for(int i = 0; i < paramValues.length; i++) {
      out.println("<li>" + paramValues[i]);
    }
```

```
            out.println("</ul>");
          }
        }
        out.println("</tr>\n</table>\n</body></html>");
      }

      // Method to handle POST method request.
      public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        doGet(request, response);
      }
    }
```

Now, try the above servlet with the following form −

```
<html>
  <body>
    <form action = "ReadParams" method = "POST" target = "_blank">
      <input type = "checkbox" name = "maths" checked = "checked" /> Maths
      <input type = "checkbox" name = "physics"  /> Physics
      <input type = "checkbox" name = "chemistry" checked = "checked" /> Chem
      <input type = "submit" value = "Select Subject" />
    </form>
  </body>
</html>
```

Now calling servlet using the above form would generate the following result −

Reading All Form Parameters

Param Name	Param Value(s)
maths	on
chemistry	on

The above servlet code can be tried to read any other form's data having other objects like text box, radio button or drop-down box etc.

SERVLETS- COOKIES

Cookies are the textual information that is stored in **key-value pair** format to the client' s browser during multiple requests.
It is one of the **state management techniques** in session tracking.
Basically, the server **treats every client request as a new one** so to avoid this situation cookies are used.
When the client **generates a request, the server gives the response with cookies** having an id which are then stored in the client' s browser.
Thus, if the client generates **a second request, a cookie with the matched id is also sent** to the server.

The server will **fetch the cookie id, if found it will treat it as an old request** otherwise the request is considered new.

Using Cookies in Java

In order to use cookies in java, use a Cookie class that is present in javax.servlet.http package.
To make a cookie, create an object of Cookie class and pass a name and its value.
To add cookie in response, use addCookie(Cookie) method of HttpServletResponse interface.
To fetch the cookie, getCookies() method of Request Interface is used.

Methods in Cookies

clone(): Overrides the standard java.lang.Object.clone method to return a copy of this Cookie.
getComment(): Returns the comment describing the purpose of this cookie, or null if the cookie has no comment.
getDomain(): Gets the domain name of this Cookie.
getMaxAge(): Gets the maximum age in seconds of this Cookie.
getName(): Returns the name of the cookie.
getPath(): Returns the path on the server to which the browser returns this cookie.
getSecure(): Returns true if the browser is sending cookies only over a secure protocol, or false if the browser can send cookies using any protocol.

getValue(): Gets the current value of this Cookie.
getVersion(): Returns the version of the protocol this cookie complies with.
setValue(String newValue): Assigns a new value to this Cookie.
setVersion(int v): Sets the version of the cookie protocol that this Cookie complies with.

Example

The name of the Institute is passed to Servlet 2 from Servlet 1 using Cookies.

```
<!DOCTYPE html>
<html>
	<head>
		<title>TODO supply a title</title>
		<!-- css-->
		<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"

	integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJl
SAwiGgFAW/dAiS6JXm"
                                                crossorigin="anonymous">
		<link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.
min.css">

		<meta charset="UTF-8">
		<meta name="viewport" content="width=device-width,
initial-scale=1.0">
	</head>
	<body>
		<form action="servlet1" method="POST">
			<div class="container-fluid ">

			<div class="jumbotron">
				<div class="container col-sm-4">
			<h2>Enter your institute's name</h2>
			<input type="text" name="name" style="font-size:30px;">
			<br>
			<br>
			<!-- button to redirect to servlet1 -->
```

```
                    <button type="submit" style="font-size:20px;" class="center">
                        Go !
                    </button>
                    <br><br>
                    </div>
                    </div>
                    </div>
                </form>
        </body>
</html>
```