
UNIT 1 INTRODUCTION TO J2EE ARCHITECTURE AND DESIGN PATTERN

Structure

- 1.0 Introduction
- 1.1 Objectives
- 1.2 Web Server and Web Container
- 1.3 Introduction to J2EE
- 1.4 Design Patterns
 - 1.4.1 MVC
 - 1.4.2 Repository Design Pattern
 - 1.4.3 Singleton
 - 1.4.4 Factory
- 1.5 Building Java Application JAR and WAR and deployment in Tomcat
- 1.6 Summary
- 1.7 Solutions/Answer to Check Your Progress
- 1.8 References/Further Reading

1.0 INTRODUCTION

You are well aware of the java programming language, which is an object oriented programming language. Java technology provides the specific environment in which Java programming language applications run. The J2EE platform provides runtime environment for developing and running large-scale, multi-tiered online/web and internet applications. This unit will give you an introduction of J2EE and its architecture. Also in this unit you will be introduced to some well-known design patterns. Design patterns give readymade explanations/solutions or templates to commonly occurring problems in the programming language. Design patterns are the best solutions that are tested and verified prototypes that can speed up your development process. There are many design patterns but it is not possible to cover all patterns in this course. This unit covers MVC, Repository, Singleton, Factory design patterns and some important terms, including Web server and Web Container. This unit will also provide you with the skill and process to package Java Application project to JAR/WAR and deploy your application in Tomcat.

The next Unit 2 and 3 of this block will give you detailed discussions on Servlet programming and session management in web applications. Unit 4 of this block will provide you with a basic understanding of Java Server Pages (JSP) components that make an entire JSP page, Java Bean, Custom tag, and many other concepts used in JSP. After studying this block, you will be able to use design patterns and build your java application using data retrieval and data storage in Servlets/JSP programming and their deployment in Tomcat.

1.1 OBJECTIVES

After going through this unit, you should be able to:

- ... differentiate between web server and web container,
- ... know about design patterns and different types of design patterns,

- ... explain where and why singleton pattern is used,
- ... explain factory method design pattern and how to implement it,
- ... know the difference between jar and war files and packaging of java application to JAR/WAR file, and
- ... deploy java application as war file.

1.2 WEB SERVER AND WEB CONTAINER

You are well aware about web applications and you have opened many web applications on the internet. Assume that you have filled a form on this and received a response from them. This communication between you and web application are fulfilled using HTTP request and HTTP response with HTTP Protocol and method (You can study much more about these in the next Unit of this Block). When you write your programme in a programming language such as Servlet and JSP (this will be covered in detail in Unit 2, 3 and Unit 4 of this Block), you will be using some other terms like Web Server and Web Container in addition to HTTP request and response. This section describes you about the Web Server and Web Container.

Web Server is a server software that handles HTTP requests and responses to deliver web pages or web content to clients (i.e. web browser) using HTTP protocol. Web browser communicates with web server using the Hypertext Transfer Protocol (HTTP). Hypertext Transfer Protocol (HTTP) is specially meant to communicate between Client and Server using Web (or Internet). To successfully execute web application, the number of server side technologies (such as JSP, Servlets and PHP) and their libraries are installed on the web server. Without these libraries, a web server cannot execute those server technologies based applications. In other words, we may say that the web server creates an execution infrastructure for the server technologies. An example of web server is Apache HTTP Server.

Web Container is a web server component that handles Servlets, Java Server Pages (JSP) files, and other Web-tier components. Web container is also called a Servlet Container or Servlet Engine. It is the responsibility of the Web container to map a URL to a particular servlet. Also, Web container ensures that the mapped URL requester has the correct access rights. It means that it provides the run time environment to web applications. The most common web containers are Glassfish, Eclipse, JBOSS, Apache Tomcat, WebSphere and Web Logic.

For deployment and running the JSPs/Servlets we need a compatible web server with a servlet container.

You can know more about the Java 2 Enterprise Edition (J2EE) and its uses in java programming in the next section.

1.3 INTRODUCTION TO J2EE

In the previous section, you have learned about the two most important terms web server and web container which are required for Java programming. Through this section you know about the Java 2 Enterprise Edition (J2EE).

You are well aware about core java or Java Standard Edition (Java SE). Java SE provides all core functionalities of java programming language. Java technology is used as a programming language as well as a platform. A Java platform provides a specific environment in which Java applications run. The Java Enterprise Edition platform resides on top of the Java Standard Edition platform.

The J2EE platform is a set of services, application programming interfaces (APIs) and protocols. J2EE is used to develop and deploy multi-tier web-based enterprise applications using a series of protocols and application programming interfaces (APIs). J2EE contains several APIs such as Java Servlets, Java Server Pages (JSP), Enterprise Java Beans (EJB), Java Database Connectivity (JDBC), Java Message Service (JMS), Java Naming and Directory Interface (JNDI) and so on.

The J2EE application model divides applications into three basic parts like components, containers and connectors. The application developers work on the components part, whereas system vendors are responsible for implementing containers and connectors. Containers act as a mediator between clients and components by providing services like transaction support and resource pooling. Connectors provide bidirectional communication between J2EE components and enterprise systems. Below in figure 1, you can depict the functioning of J2EE model:

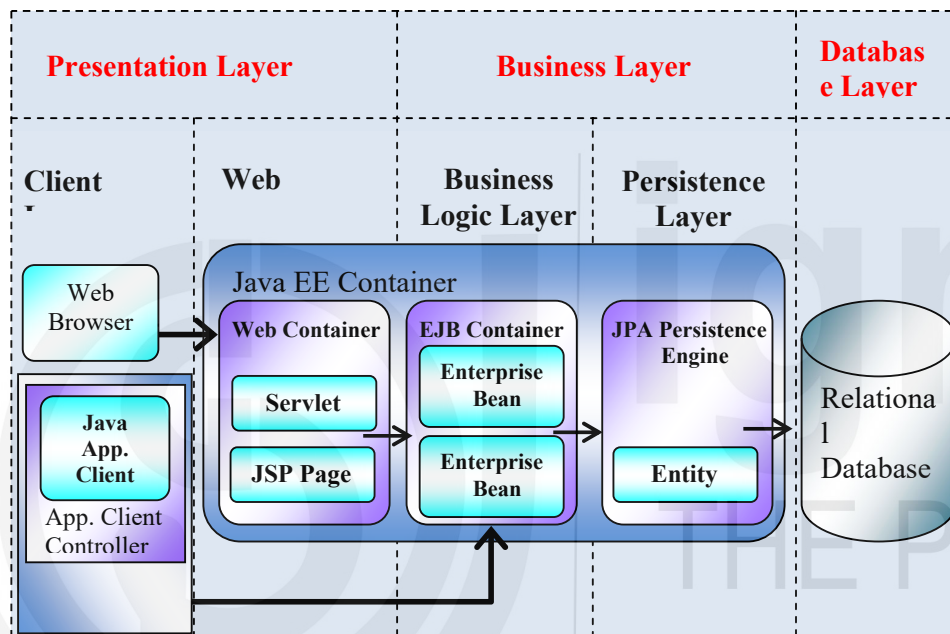


Figure 1: J2EE Architecture

A J2EE application contains four components or tiers: Presentation, Application, Business, and Resource adapter components. The presentation component is the client side component that is visible to the client and runs on the client's server. The Application component is web side layer that runs on the J2EE server. The business component is the business layer which includes server-side business logic such as JavaBeans, and it is also run on the J2EE server. The resource adaptor component comprises an enterprise information system.

When you develop J2EE application, you may find J2EE clients such as a web client, an application client, wireless clients or Java Web Start-enabled clients. For running J2EE application, you need a J2EE container which is a server platform, Java component can be run on this container using APIs provided through the Web container and EJB container. The EJB container is a server platform used for controlling the execution of Enterprise Bean. Also, the EJB container job is to provide local and remote access to enterprise beans.

For creating complex applications, you can use J2EE frameworks like Struts, Spring and Hibernate, which will you read in block-2 of this course.

☛ Check Your Progress 1

1. What is a web server, and how does it differ from a web container? Also, name any four web containers.

.....

.....

.....

.....

2. What is J2EE? What are the components of J2EE applications? What technologies are included in the J2EE platform?

.....

.....

.....

.....

3. Define the term module in J2EE. What are the four specific modules used in J2EE applications?

.....

.....

.....

.....

1.4 DESIGN PATTERNS

The previous section has given you an introductory lesson on the Java 2 Enterprise Edition (J2EE). This section gives a detailed description of the Design Patterns and their types and uses in Java.

Design patterns are the best solutions that are tested, verified developed prototypes that can speed up your development process. When you may face problems during software development, the design patterns gives you explanations for those problems. Experienced developers have developed these explanations to provide the finest solutions to the problems. It is also beneficial for un-experienced software developers to learn software design in a simpler manner. The concept of design pattern has been initiated by four authors collectively known as GOF (Gang of Four), and they have published a book (Design Patterns - Elements of Reusable Object-Oriented Software) on this concept. Design Patterns provides a general reusable solution to commonly occurring problems. It is not a complete design that can be directly mapped into your

code and solve your purpose. It is just like a template or explanation for how to solve your problems. It offers a common platform for all software developers.

There are 23 classic design patterns and they are categorized into three groups: Creational, Structural and Behavioral. The Creational design patterns deal with the creation of an object. Structural design patterns deal with the class structure, and the behavioral design patterns define the interaction between objects.

Creational design patterns

The Creational design patterns provide a way to deal with the creation of an object and define 5 design patterns such as Abstract Factory, Builder, Factory, Prototype and Singleton pattern from which two like Singleton and Factory design pattern are to be discussed below. The **Abstract Factory** design pattern allows to create families of related objects. This pattern permits us to create a Factory for factory classes. The **Builder** design pattern provides a valuable solution to many object creation problems in object-oriented programming. The objective of this pattern is to separate the creation of a complex object from its representation. The **Prototype** design pattern allows us to produce new objects by cloning an existing object using a clone() method. This pattern is useful when a particular resource is costly to create or when the abstract factory pattern is not preferred.

Structural design patterns

The Structural design patterns are related to the creation of class and object structure. These patterns define seven design patterns: Adapter, Bridge, Composite, Decorator, Façade, flyweight, and Proxy. The **Adapter** design pattern provides a way to work as a link between two unrelated interfaces. The object that links these incompatible or unrelated interfaces is called an Adapter. So, the adapter design pattern permits the interface of an existing class to be used as another interface. The adapter pattern is generally used to make available the existing classes for others without changing their source code. The **Composite** design pattern defines a pool of objects that are to be treated as a single object. The **Decorator** design pattern is used to change the functionality of an object at runtime. The **Façade** design pattern is used to generating wrapper interfaces on top of existing interfaces, and it hides the complications of the system and arranges a simpler interface to the client application. The **Flyweight** design pattern moderates the cost of generating and controlling a large number of related objects. The **Proxy** design pattern is used as a procurator or placeholder for another object to regulate the access and reduce cost as well as reduce complexity.

Behavioral design patterns

The third design patterns category is Behavioral, which defines 11 design patterns: Chain of responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, and Visitor. They are specifically concerned with the interaction between objects. As the name recommends, the **chain of responsibility** design pattern builds a chain of receiver objects for a request. This pattern is used where the request sender is disassociated from its receiver based on the type of request. The **Command** design pattern is used to create objects containing information about an action, such as method name, owner of the method, and Parameter values necessary to be produced later. As the name suggests, the **Interpreter** design pattern provides an interpreter to deal with language grammar and it is used to interpret sentences in a language. The **Iterator** design pattern provides a way to access the collection object without revealing its underlying representation.

This pattern is frequently used in Java and .Net programming languages. **Mediator** design pattern provides a communication medium or a mediator class that handles all the communications between different classes. The capability of **Memento** design pattern is used to restore an object to its earlier state. The **Observer** design pattern is beneficial when an object is changed; then, you can get notified about the state of the object through this pattern. The **State** design pattern permits an object to modify its behaviour when its inner state is modified. The **Strategy** pattern or policy design pattern allows selecting an algorithm during runtime. The **Template Method** design pattern provides a way to create a superclass template method and permit its child classes to provide concrete behaviour. The **Visitor** design pattern gives a way to separate an operation from the object structure on which it operates. The basic purpose of this pattern is to define a new operation without modifying the original classes.

Besides these design patterns some more design patterns are also available. The MVC, Repository, Singleton and Factory design patterns are discussed below:

1.4.1 MVC Design Pattern

Model View Controller (MVC) design pattern is an architectural pattern for web applications. It can be used across many frameworks with many programming languages such as Java, PHP, Python, Ruby, etc. This design pattern is extensively used to design enterprise web applications and mobile applications. The MVC design pattern comprises three layers such as data, presentation and control information. This pattern requires each of these layers to act independently.

The MVC design pattern described three layers such as Model, View and Controller. In MVC pattern, M (Model) denotes only the pure application data and does not contain any logic for representing data to a user, whereas V (View) is responsible for presenting data to the user. The C (Controller) comes between the view and the model layer and it is responsible for accepting a request from the user, performs interactions on the data model objects, and sends it back to the view layer.

The UML diagram of MVC design pattern is depicted in figure 2. The following example demonstrates to you how MVC design pattern will work. The example contains total of four java files, from which three java files for MVC layers and one java file containing the main() method. For defining the MVC pattern, the first java file is created as 'UniversityModel', which acts as a model, 'univView' as a view that can display university details and 'UnivController' file is responsible for storing the data in univ object and updateView() method updates the data in a 'univView' class method. The 'MVCEExample' file will use 'UnivController' file to illustrate the MVC pattern.

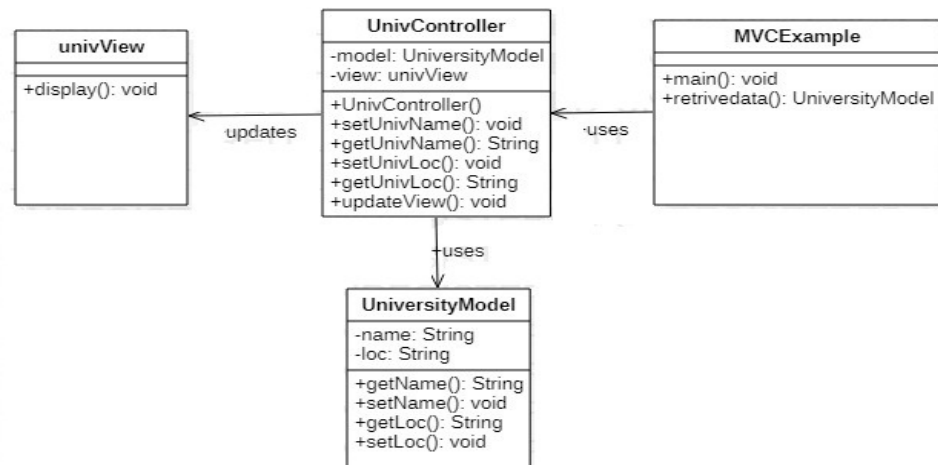


Figure 2: UML class diagram of MVC design pattern is

All files of MVC pattern example containing java source code are listed below.

1. You can create a UniversityModel.java file using the following code:

```
// UniversityModel.java
public class UniversityModel
{
    private String name;
    private String loc;

    public String getName()
    {return name; }

    public void setName(String name)
    {this.name = name; }

    public String getLoc()
    {return loc; }

    public void setLoc(String loc)
    {this.loc = loc; }
}
```

2. You can create a univView.java file using the following code:

```
// univView.java
public class univView
{
    public void display(String univName, String univLoc)
    {
        System.out.println("University Details: ");
        System.out.println("Name: " + univName);
        System.out.println("Location: " + univLoc);
    }
}
```

3. You can create a UnivController.java file using the following code:

```
// UnivController.java
public class UnivController
{
    private UniversityModel model;
    private univView view;

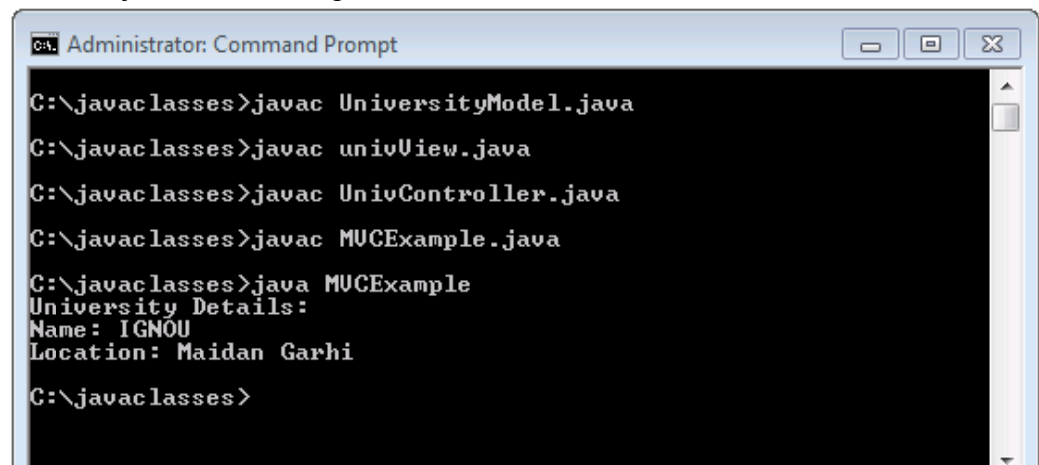
    public UnivController(UniversityModel model, univView view)
    {
        this.model = model;
        this.view = view;
    }
    public void setUnivName(String name)
    {
        model.setName(name);
    }
    public String getUnivName()
    {
        return model.getName();
    }
}
```

```
    }  
    public void setUnivLoc(String loc)  
    {  
        model.setLoc(loc);  
    }  
    public String getLoc()  
    {  
        return model.getLoc();  
    }  
    public void updateView()  
    {  
        view.display(model.getName(), model.getLoc());  
    }  
}
```

4. Now, create a MVCEXample.java file using the following code

```
//MVCEXample.java  
public class MVCEXample  
{  
    public static void main(String[] args)  
    {  
        UniversityModel model = retrivedata();  
  
        //Create a view : to write student details on console  
        univView view = new univView();  
        UnivController controller = new UnivController(model, view);  
        controller.updateView();  
    }  
    private static UniversityModel retrivedata()  
    {  
        UniversityModel univ = new UniversityModel();  
        univ.setName("IGNOU");  
        univ.setLoc("Maidan Garhi");  
        return univ;  
    }  
}
```

5. Now, compile all the files using javac command and run MVCEXample file with java as shown in figure-3:



```
C:\javaclasses>javac UniversityModel.java  
C:\javaclasses>javac univView.java  
C:\javaclasses>javac UnivController.java  
C:\javaclasses>javac MVCEXample.java  
C:\javaclasses>java MVCEXample  
University Details:  
Name: IGNOU  
Location: Maidan Garhi  
C:\javaclasses>
```

Figure 2: Output Screen for MVC pattern Example

1.4.2 Repository Design Pattern

In plain words, repository means something is deposited in storage. The repository meaning is same as in terms of design pattern, it is related to the data. The repository design pattern is used in data-centric applications. The Repository pattern is used to isolate the business logic layer and the data source layers in your application. The repository layer comes between the domain and data mapping layers. As shown in figure 4, the repository resides in the mid of the client business logic and data source layer. At the data source layer, a database can be kept/used.

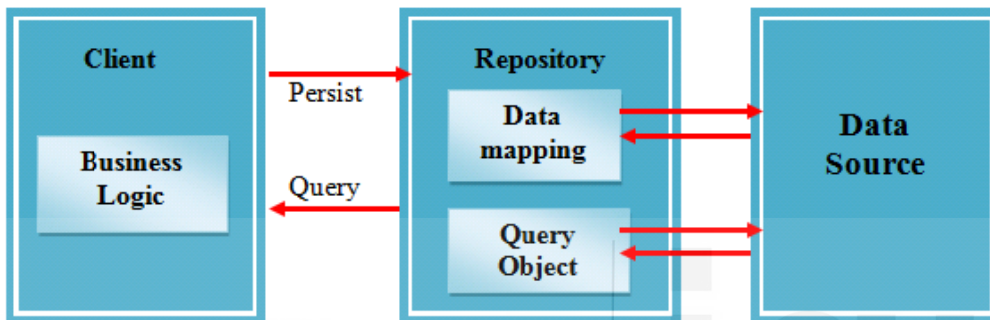


Figure 4: Block diagram for Repository Design Pattern

Suppose client business logic wishes a query from data source then it first goes to repository layer thereafter repository will send the query to the database, and then repository maps the data to the business entity. Similarly, when client business logic wants to save data in the database, data will not directly go to the database but will go through the repository layer.

While using repository pattern, your application's business logic layer need not have

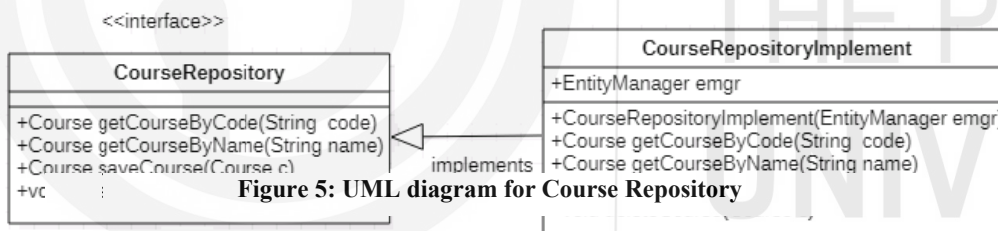


Figure 5: UML diagram for Course Repository

any knowledge on data persistence, it means the repository layer provides a way to hide the internal implementation of how data persistence happens. This way it empowers you to replace your data source without changing your business logic. This pattern is useful when you have several entities, and complex queries are applied on those entities.

This pattern is one of the most popular Java persistence patterns. This pattern is used with JPA and other frameworks. A JPA (Java Persistence API) is a java specification and defined in `javax.persistence` package. JPA is used to access and manage persist data between Java object and relational database. JPA can act as a link between object-oriented domain models and relational database systems. Various ORM (Object Relational Mapping) tools such as Hibernate, Spring are used by the JPA for implementing data persistence.

Assume that you want to develop a course repository for a university database application. For this, you need persistent data storage for courses where you may add and remove courses and search for them. For using repository design pattern, you can create an interface that defines read and write operations for a specific entity and this

interface is implemented by data store related classes. The UML diagram for such repository is as follows:

In the UML diagram, you have seen that there are four methods in 'CourseRepository' interface that you can use to find a course by code and name; save and delete course entity. You can create a course repository interface like the following code and then write a class for implementing 'CourseRepository' interface. If you are using any framework such as Spring Data JPA and Apache DeltaSpike Data then you can only define the repository interfaces, and these frameworks can generate standard repository implementations for you. Also, you can write your own implementation as per the requirements of the implementation complexity of your problem. For running this pattern, you need database connectivity with J2EE framework. The repository design pattern is stronger to you when you will read J2EE frameworks in the next Block-2 of this course.

```
public interface CourseRepository
{
    Course getCourseByCode(String code);
    Course getCourseByName(String name);
    Course saveCourse(Course c);
    void deleteCourse(Course c);
}
```

1.4.3 Singleton Design Pattern

The singleton pattern is a simplest software design pattern amongst the 23 well-known "Gang of Four" design patterns and it comes under the creational design pattern category, which deals with the creation of an object. As the name suggests, Singleton design pattern is used to create only one instance of a class, and all other classes of the application can use this instance. This type of pattern is mostly used in database connection and multi-threaded applications. Singleton pattern is used in logging, caching, thread pools, configuration settings etc. You can use Singleton pattern in other design patterns like Abstract Factory, Builder, Facade, Prototype etc., while developing your own solutions to a specific problem.

The UML class diagram of Singleton pattern is as under:

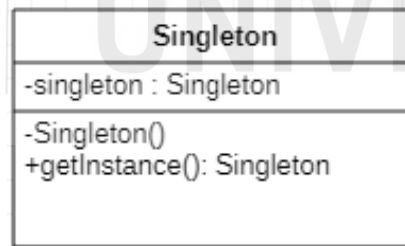


Figure 3: UML class diagram of Singleton pattern

An implementation of this pattern must ensure that only one instance of the singleton class will exist and that instance provides global access to others. While implementing singleton pattern, constructors of the class must be declared as private, and method is defined as public static that returns the instance of the class. The instance is typically stored as a private static variable. The static variable is initialized at some point before the static method is first called.

Java core libraries provide many classes which are written using singleton pattern. A few of them are listed below:

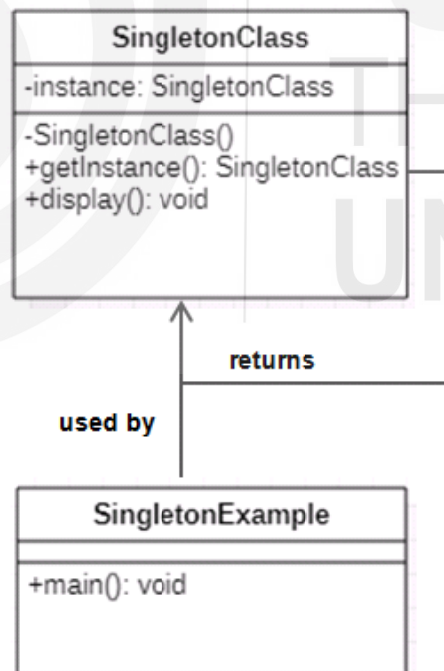
... Java.lang.Runtime with getRuntime() method

... Java.awt.Toolkit with getDefaultToolkit()
... Java.awt.Desktop with getDesktop()

A sample implementation of Singleton design pattern in Java is as follows:

```
public final class Singleton
{
    //A singleton class should have public visibility
    //static instance of class globally accessible
    private static final Singleton instance = new Singleton();
    private Singleton()
    {
        /* private constructor, so that class cannot be instantiated from outside this class
        */
    }
    public static Singleton getInstance()
    {
        /*declaration of the method as public static, so that instance can be used by
        other classes */
        return instance;
    }
}
```

The following example illustrates to you how the Singleton design pattern works. For this example, UML class diagram is as shown in figure 7:



This Singleton Design Pattern example contains two classes. One is 'SingletonClass', which behaves as Singleton class. This class have private constructor and provides a static method to get its static instance to other classes. The second class is 'SingletonExample', which use an instance of the 'SingletonClass' class. The implementation of this example is underneath.

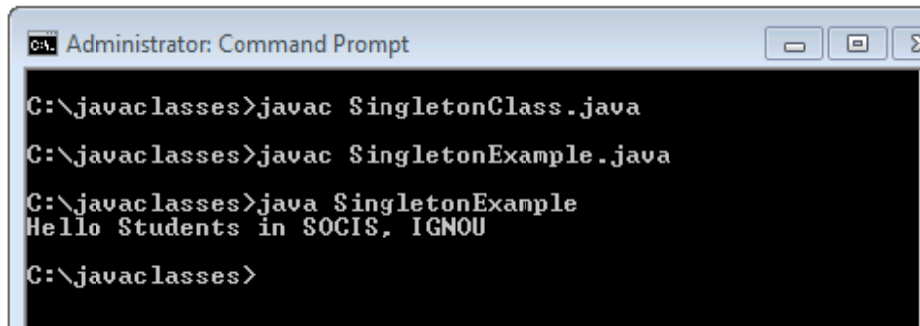
... First, you can create a Singleton Class as the name of 'SingletonClass' using the following source code:

```
// SingletonClass.java
public class SingletonClass
{
    //create an object of SingletonClass
    private static SingletonClass instance = new SingletonClass();
    private SingletonClass(){}
    public static SingletonClass getInstance()
    {
        return instance;
    }
    public void display()
    {
        System.out.println("Hello Students in SOCIS, IGNOU");
    }
}
```

... Create another class called 'SingletonExample' to get the only instance from the singleton class. The code of this class is listed below:

```
// SingletonExample.java
public class SingletonExample
{
    public static void main(String[] args)
    {
        //Get the instance created by Singleton class
        SingletonClass instance = SingletonClass.getInstance();
        instance.display();    //display data
    }
}
```

... Now compile the both classes and run the 'SingleExample'. The output of the this example is displayed in following figure-8:



```
C:\javaclasses>javac SingletonClass.java
C:\javaclasses>javac SingletonExample.java
C:\javaclasses>java SingletonExample
Hello Students in SOCIS, IGNOU
C:\javaclasses>
```

Figure 4: Output Screen for Singleton example

1.4.4 Factory Design Pattern

In the previous section, the Singleton design pattern has been explained. The Singleton design pattern belongs to the Creational pattern category. As you know, the

creational pattern is related to the creation of the object. In this section, you will learn one more design pattern named Factory, which also comes in the same category.

A Factory Design Pattern or Factory Method Pattern is one of the most used design pattern in java. It is a creational design pattern which deals with the creation of an object. As per the GOF, this pattern defines an interface or abstract class for creating an object, but subclasses are responsible for creating the instance of the class. The advantage of Factory Pattern is that it allows the sub-classes to choose the type of objects to create. In other words, this pattern is used to create an object where object creation logic is hidden to the client and refers to the newly created object using a common interface. This design pattern is also known as 'Virtual Constructor'.

To implement the Factory Design Pattern, first, you define a factory method inside an interface or abstract class and let the subclass use the above factory method and decide which object to create.

Factory Design Pattern implementation

The following example demonstrates to you how to work Factory Design Pattern. Assume that IGNOU wants to send announcement details of new courses to users

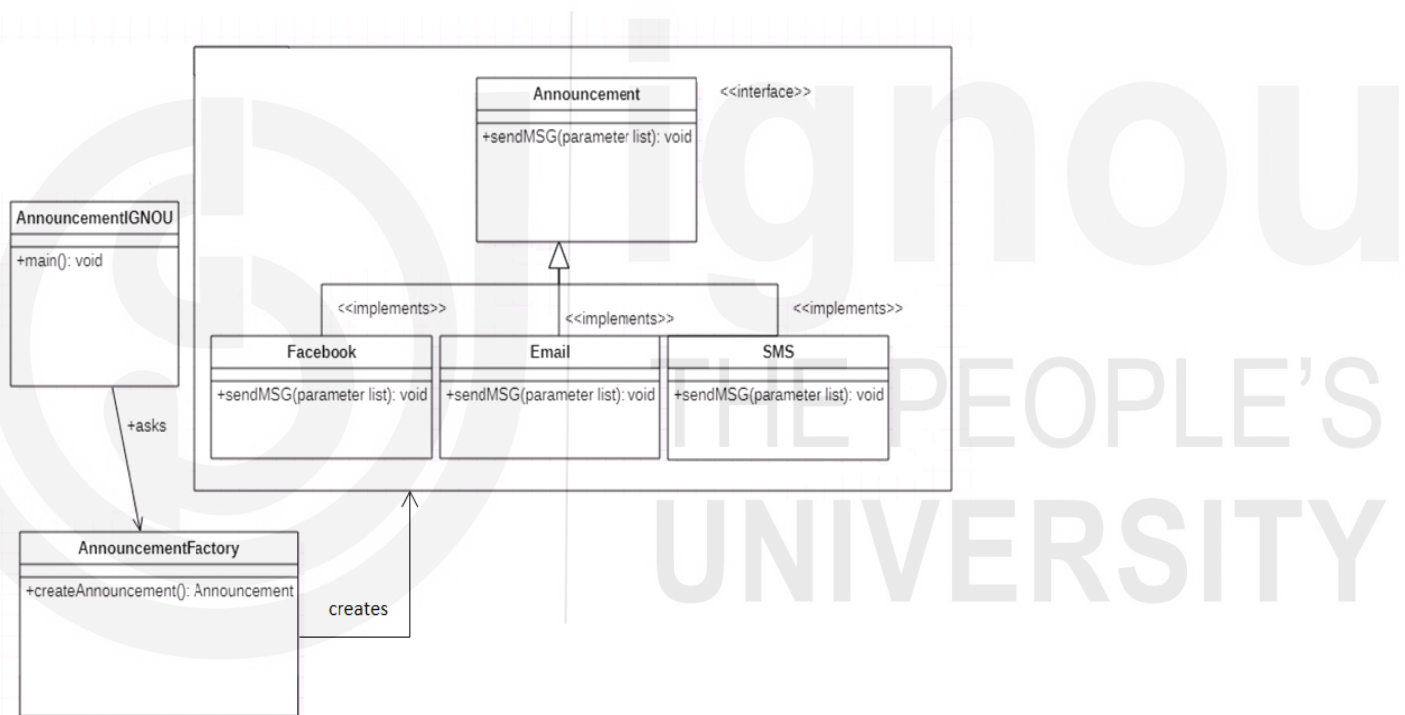


Figure 5: UML diagram for Factory Design Pattern

through Facebook, Email and SMS. Let's implement this example with the help of the Factory design pattern. For implementing this, first, you can design a UML class diagram, and then you can transform this class diagram into source code implementation. UML class diagram for this example using the Factory design pattern is shown in figure 9.

For implementing the above class diagram, you can create an interface as **Announcement** and three concrete classes such as **Facebook.java**, **Email.java** and **SMS.java**. These three classes can implement 'Announcement' interface. Besides these, you can also create two more classes. You can create a factory class 'AnnouncementFactory.java' to get an **Announcement** object. Creation of 'AnnouncementIGNOU.java' class is to use factory class and get an object of a concrete class.

Now you can follow the steps for the creation and running the example of Factory design pattern:

... First, you can create Announcement interface.

```
public interface Announcement
{
    void sendMSG();
}
```

... Now, you can create all three classes implementing the same interface as per the following codes:

```
//Facebook.java
public class Facebook implements Announcement
{
    public void sendMSG()
    {
        System.out.println("Sending announcement through Facebook");
    }
}
```

```
//Email.java
public class Email implements Announcement
{
    public void sendMSG()
    {
        System.out.println("Sending announcement as an e-mail");
    }
}
```

```
//SMS.java
public class SMS implements Announcement
{
    public void sendMSG()
    {
        System.out.println("Sending announcement as an SMS");
    }
}
```

... Now, you can create a Factory Class 'AnnouncementFactory.java' as code listed below:

```
// AnnouncementFactory.java
public class AnnouncementFactory
{
    public Announcement createAnnouncement(String media)
    {
        if (media == null || media.isEmpty())
            return null;
        if ("Facebook".equals(media))
        {
            return new Facebook();
        }
        else if ("EMAIL".equals(media))
        {

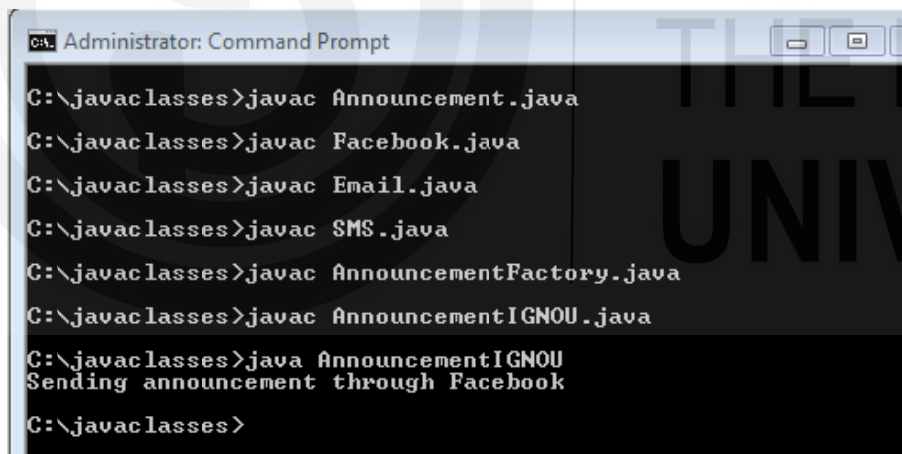
```

```
        return new Email();
    }
    else if ("SMS".equals(media))
    {
        return new SMS();
    }
    return null;
}
}
```

... You can create 'AnnouncementIGNOU.java'. Using this file, you can use 'AnnouncementFactory' class to create and get an object of concrete class. Here you are passing some information as "Facebook".

```
// AnnouncementIGNOU.java
public class AnnouncementIGNOU
{
    public static void main(String[] args)
    {
        AnnouncementFactory anFactory = new AnnouncementFactory()
        Announcement announcement =
        anFactory.createAnnouncement("Facebook");
        announcement.sendMessage();
    }
}
```

... Now, you can compile all the java files and run 'AnnouncementIGNOU' as per the following figure 10. This figure is also shown output at the command prompt.



```
C:\javaclasses>javac Announcement.java
C:\javaclasses>javac Facebook.java
C:\javaclasses>javac Email.java
C:\javaclasses>javac SMS.java
C:\javaclasses>javac AnnouncementFactory.java
C:\javaclasses>javac AnnouncementIGNOU.java
C:\javaclasses>java AnnouncementIGNOU
Sending announcement through Facebook
C:\javaclasses>
```

Figure 6: Compiling and output screen for Factory pattern example

Up to now, you have learned about the design patterns, including MVC, Repository, Singleton and Factory, and how to implement these patterns. In the next section, we will learn about the building and deployment of java/J2EE applications

☛ Check Your Progress 2

1. What do you mean by Design Patterns? Can you name some of the design patterns used in JDK core libraries?

.....

.....

.....

.....

2. What is Singleton pattern? Name one singleton class in Java. How can you create Singleton class in java? Can we create a clone of a singleton object? If yes, then how to prevent cloning of a singleton object?

.....

.....

.....

.....

3. Explain the benefits of Factory pattern.

.....

.....

.....

.....

1.5 BUILDING JAVA APPLICATION JAR AND WAR AND DEPLOYMENT IN TOMCAT

This section will help you to understand about building java applications to JAR and WAR files and deployment in Tomcat. The following sections described to you how to create WAR file, how to deploy WAR file and how to extract WAR file. You will find more about the installation process of Apache's Tomcat and running and deployment of your Servlets and JSP programs in section 2.7 of Unit 2 of this Block-1. Before an understanding of this description, you will know about the JAR and WAR packaging in Java.

JAR Packaging

The JAR (Java Archive) is a package file format. The file extension of JAR files is .jar and may contain libraries, resources, and metadata files. Basically, it is a zipped file containing the compressed versions of .class files, compiled Java libraries and applications. You can create a JAR file using the jar command like the following. You

can also create JAR files using IDEs. You can learn more about jar files in ‘Reference Section’ of this Unit.

```
jar cf name-jar-file input-file(s)
```

Where c option is used to create a JAR file, f option specifies the outputs which go to a file. You can define any filename for a JAR file. Using the input-file(s) argument, you can specify a space-separated list of one or more files that you wishes to include in your JAR file.

WAR Packaging

WAR (Web Archive) is used to package web applications. You can deploy on any Servlet/JSP container. It may contain JSP, Servlet, XML, images, HTML pages, CSS files and other resources. WAR package combines all the files into a single unit. It takes a smaller amount of time while transferring file from client to server. The extension of WAR files is .war needs a server to execute a WAR file.

The following sections describe to you how to create, deploy and extract WAR file.

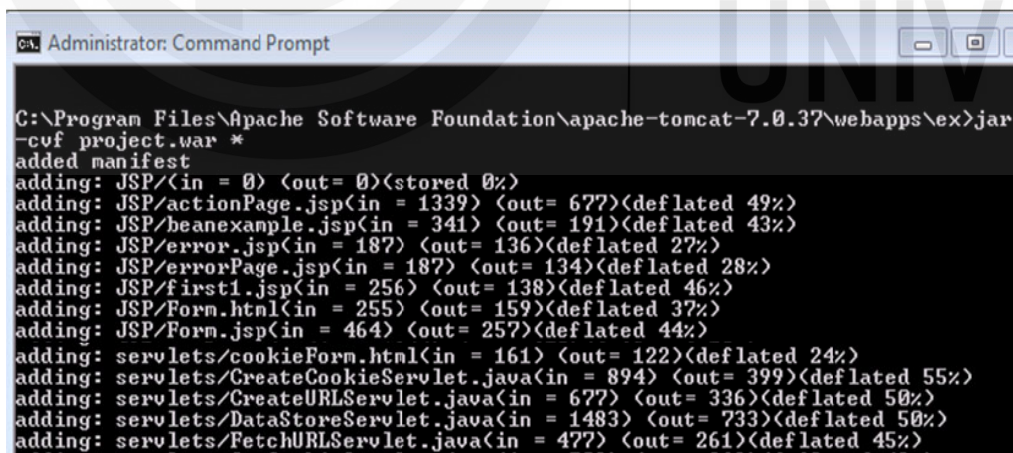
Create WAR file

You can create war file using jar tool of JDK or IDEs. You can use -c option of jar to create the war file. For creating war file, you can go inside the project application directory (outside the WEB-INF folder), then write command like the following:

```
jar -cvf projectname.war *
```

Where -c option is used to create file, -v to generate the verbose output and -f to specify the archive file name. The * (asterisk) symbol indicates that all the files of this directory (including sub directory).

The following figure-11 is displayed creation process of WAR file:



```
C:\Program Files\Apache Software Foundation\apache-tomcat-7.0.37\webapps\ex>jar
-cvf project.war *
added manifest
adding: JSP/(in = 0) (out= 0)(stored 0%)
adding: JSP/actionPage.jsp(in = 1339) (out= 677)(deflated 49%)
adding: JSP/beanexample.jsp(in = 341) (out= 191)(deflated 43%)
adding: JSP/error.jsp(in = 187) (out= 136)(deflated 27%)
adding: JSP/errorPage.jsp(in = 187) (out= 134)(deflated 28%)
adding: JSP/first1.jsp(in = 256) (out= 138)(deflated 46%)
adding: JSP/Form.html(in = 255) (out= 159)(deflated 37%)
adding: JSP/Form.jsp(in = 464) (out= 257)(deflated 44%)
adding: servlets/cookieForm.html(in = 161) (out= 122)(deflated 24%)
adding: servlets/CreateCookieServlet.java(in = 894) (out= 399)(deflated 55%)
adding: servlets/CreateURLServlet.java(in = 677) (out= 336)(deflated 50%)
adding: servlets/DataStoreServlet.java(in = 1483) (out= 733)(deflated 50%)
adding: servlets/FetchURLServlet.java(in = 477) (out= 261)(deflated 45%)
```

Figure 7: Command prompt showing creation process of WAR file

Deploy the WAR file

You can deploy war file by placing the war file in a specific folder of the server. If

Figure 7: UML class diagram for Singleton Design Pattern Example

you are using the tomcat server and want to deploy this file manually, go to the 'webapps' directory of apache tomcat and paste the war file. The server will extract the war file internally. Now, you are able to access the web project through a browser.

... Suppose, you are deploying project.war file. First go to tomcat webapps folder and paste it.

... Go to tomcat->bin folder and start tomcat by clicking startup.bat file

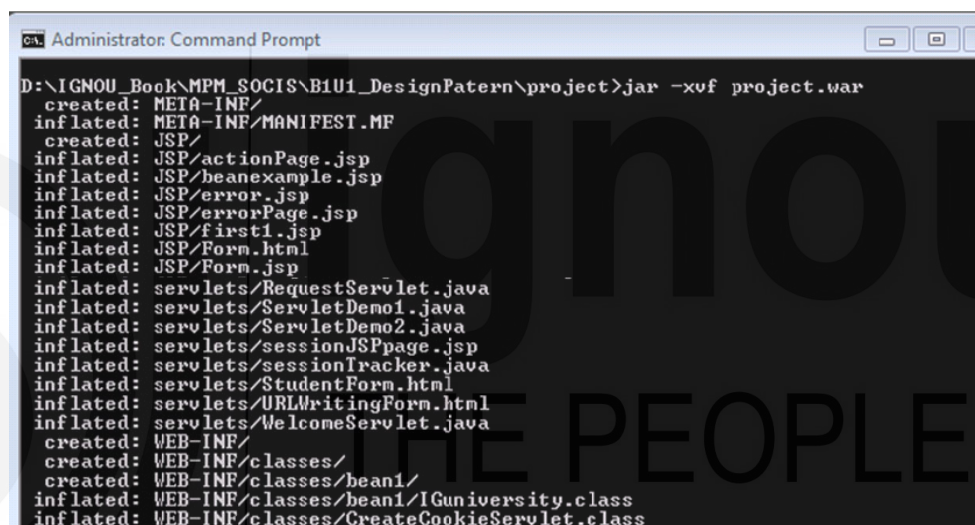
... Now, you can access your application from the browser. Open the browser and write in the address bar as localhost:port/projectname eg. localhost:8080/project

Extract war file manually

If you wish to extract the war file manually, then you need to use -x switch of jar tool of JDK. The following command is used to extract the war file.

```
jar -xvf projectname.war
```

The following figure-12 shows you extraction process of WAR file:

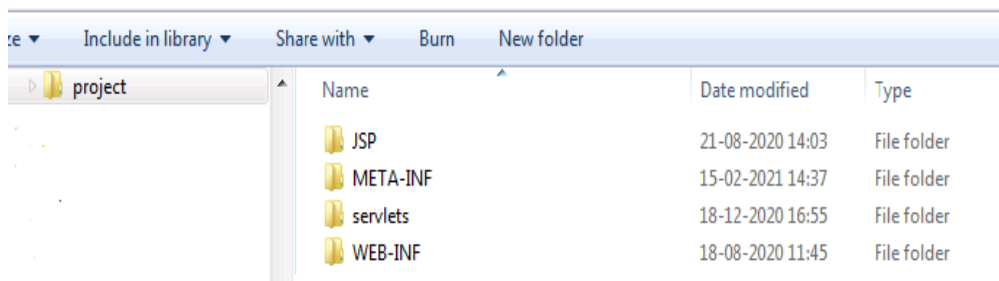


```

Administrator: Command Prompt
D:\IGNOU_Book\MPM_SOCIS\B1U1_DesignPattern\project>jar -xvf project.war
created: META-INF/
inflated: META-INF/MANIFEST.MF
created: JSP/
inflated: JSP/actionPage.jsp
inflated: JSP/beanexample.jsp
inflated: JSP/error.jsp
inflated: JSP/errorPage.jsp
inflated: JSP/first1.jsp
inflated: JSP/Form.html
inflated: JSP/Form.jsp
inflated: servlets/RequestServlet.java
inflated: servlets/ServletDemo1.java
inflated: servlets/ServletDemo2.java
inflated: servlets/sessionJSPpage.jsp
inflated: servlets/sessionTracker.java
inflated: servlets/StudentForm.html
inflated: servlets/URLWritingForm.html
inflated: servlets/WelcomeServlet.java
created: WEB-INF/
created: WEB-INF/classes/
created: WEB-INF/classes/bean1/
inflated: WEB-INF/classes/bean1/IGUniversity.class
inflated: WEB-INF/classes/CreateCookieServlet.class
  
```

Figure 12: Screen showing extraction process of WAR file

The figure-13 shows you the file structure after extraction of WAR file. Now, you have learned about how to create, deploy and extract WAR files. The next unit will tell you how to create servlet and deploy in Tomcat.



Name	Date modified	Type
JSP	21-08-2020 14:03	File folder
META-INF	15-02-2021 14:37	File folder
servlets	18-12-2020 16:55	File folder
WEB-INF	18-08-2020 11:45	File folder

Figure 13: File structure after extraction process of WAR file

1. What is the difference between JAR and WAR files?

.....

.....

.....

.....

2. Describe the process of creation, deployment and extraction of WAR files.

.....

.....

.....

.....

1.6 SUMMARY

This unit focussed on the J2EE, Architecture and Design Patterns. You have learned about the J2EE and its architecture. By the services of Java 2 Enterprise Edition, you can develop large scale, component based, distributed, multi-tier applications. J2EE provides many APIs that can be used to build applications. This unit also covered 23 well-known design patterns which are categorized into three groups such as Creational, Structural and Behavioral. The design pattern has given you prior developed descriptions or template as solutions to commonly occurring problems in programming language. Design patterns are the best solutions that are tested, verified prototypes that can speed up your software development process. You have learned about the MVC, Singleton, Factory and Repository design patterns with examples. The MVC pattern described three layers such as Model, View and Controller. These three layers have separate functions like Model represent data, View presents data to user and Controller accepts the request from the user performs interactions on the data model objects and send it back to the view layer. The repository design pattern is used in data-centric applications. The repository design pattern provides a way to hide the internal implementation of how data persistence happens, and it empowers you to replace your data source without changing your business logic.

The singleton pattern deals with the creation of an object and is used to create only one instance of a class and all other classes of the application can use this instance. A Factory Design Pattern or Factory Method Pattern deals with creating an object and defines an interface or abstract class for creating an object, but subclasses are responsible for creating the instance of the class. At the end of this unit, you have learned about how java application can be built as a war file and their deployment in tomcat.

1.7 SOLUTIONS/ANSWERS TO CHECK YOUR PROGRESS

Check your Progress 1

- 1) **Web Server** is server software that handles HTTP requests and responses to deliver web pages or web content to clients (i.e. web browser) using HTTP protocol. Web server creates an execution infrastructure for the server technologies. An example of web server is Apache HTTP Server. A compatible web server with a servlet container is always necessary for deployment and running the JSPs/Servlets.

Web Container is the web server component that handles Servlets, Java Server Pages (JSP) files, and other Web-tier components. Web container is also called as Servlet Container or Servlet Engine. It is the responsibility of Web container to map a URL to a particular servlet and ensure that the URL requester has the correct access rights. It means that it provides the run time environment to web applications.

The most common web containers are Glassfish, Eclipse, JBOSS, Apache Tomcat, Websphere and Web Logic.

- 2) J2EE is used to develop and deploy multi-tier web-based enterprise applications using a series of protocols and application programming interfaces (APIs). J2EE contains many APIs such as Java Servlets, Java Server Pages (JSP), Enterprise JavaBeans (EJB), Java Database Connectivity (JDBC), Java Message Service (JMS), Java Naming and Directory Interface (JNDI) and so on. The J2EE platform consist of set of services, application programming interfaces (APIs), and protocols.

A J2EE application comprises four components or tiers such as Presentation, Application, Business and Resource adapter components. The presentation component is the client side component which is visible to the client and runs on the client's server. The Application component is a web side layer that runs on the J2EE server. The business component is the business layer which includes server-side business logic such as JavaBeans, and it also runs on the J2EE server. The resource adaptor component includes enterprise information system.

J2EE contains several API technologies such as Java Servlets, Java Server Pages (JSP), Enterprise Java Beans (EJB), Java Database Connectivity (JDBC), Java Message Service (JMS), Java Transaction API (JTA), Java Naming and Directory Interface (JNDI), JDBC data access API and so on. The J2EE platform is a set of services, application programming interfaces (APIs) and protocols.

- 3) A module in J2EE is a group of one or more components of the same container type and one component deployment descriptor of the same type. There are four different modules used in J2EE: application client module, web, EJB, and resource adapter module. The Application Client module is bundled as a JAR file which contains class files and client deployment descriptor (web.xml) file. The WEB module is bundled as a JAR file that comprises class files (servlets), web deployment descriptor file, JSP pages, images, and HTML files. The Enterprise Java Beans (EJB) module is wrapped as a JAR file, collections of class files (ejb) and EJB deployment descriptor. The fourth module is Resource Adapter which is packaged as a JAR file consisting of classes, a resource adapter deployment descriptor, Java interfaces and native libraries.

Check your Progress 2

- 1) Design patterns are the best solutions that are tested, verified developed prototypes that can speed up your development process. When you may face problems during software development, the design patterns give you explanations for those problems. Experienced developers have developed these explanations to offer the finest solutions to the problems. It is also beneficial for inexperienced software developers to learn software design in a simpler manner.

The name of the design patterns used in JDK core libraries are Decorator pattern (BufferedInputStream can decorate other streams such as FilterInputStream), Singleton pattern (which is used by Runtime, Calendar classes), Factory pattern (used by Wrapper class like Integer.valueOf) and Observer pattern (used by event handling frameworks like swing).

- 2) The singleton pattern is a simplest software design pattern amongst the 23 well-known "Gang of Four" design patterns. It comes under the creational design pattern category which deals with the creation of an object. Singleton design pattern is used to create only one instance of a class and all other classes of the application can use this instance. This type of pattern is mostly used in database connection and multi-threaded applications.

The name of singleton class in JDK is java.lang.Runtime. The implementation of Singleton design pattern is as under:

```
public final class Singleton
{
    //A singleton class should have public visibility
    //static instance of class globally accessible
    private static final Singleton instance = new Singleton();
    private Singleton()
    {
        /* private constructor, so that class cannot be instantiated from outside
        this class */
    }
    public static Singleton getInstance()
    {
        /*declaration of method as public static, so that instance can be used
        by other classes */
        return instance;
    }
}
```

You can create a clone of a singleton object. Java provide clone() method of Object class for cloning. This method is protected method. In java, by default, every class extends Object class, the object of any class, including Singleton class can call clone() method. If you want to create a clone of a singleton object, then class must be implemented by java.lang.Cloneable interface.

You can use throw exception within the body of clone() method to prevent cloning of a singleton object.

- 3) A Factory Design Pattern or Factory Method Pattern is mostly used design pattern in java. It is a creational design pattern which deals with the creation of an object. As per the GOF, this pattern defines an interface or abstract class for creating an object, but subclasses are responsible for creating the class's instance. In other words, this pattern is used to create objects where object creation logic is hidden to the client and refers to newly created objects using

a common interface. This design pattern is also known as ‘Virtual Constructor’. The advantage of Factory Pattern is that it allows the subclasses to choose the type of objects they create.

Check Your Progress 3

- 1) The JAR (Java Archive) is a package file format. The file extension of JAR files is .jar and may contain libraries, resources and metadata files. Basically, it is a zipped file containing the compressed versions of .class files, compiled Java libraries and applications. WAR (Web Archive) is used to package web applications. You can deploy any Servlet/JSP container. It may contain JSP, Servlet, XML, images, HTML pages, CSS files and other resources. It combines all the files into a single unit and takes a smaller amount of time while transferring file from client to server. The extension of WAR files is .war and needs a server to execute a WAR file.
- 2) You can create war file using jar tool of JDK. You can use -c switch of jar to create the war file. For creating war file, you can go inside the project application directory (outside the WEB-INF folder) then write command as `jar -cvf projectname.war *` on the command prompt. For the deployment of WAR file, you can place war file in specific folder of server. If you are using the tomcat server and want to deploy the war file manually, go to the ‘webapps’ directory of apache tomcat and paste the war file. The server will extract the war file internally. Now, you are able to access the web project through the browser. If you wish to extract the war file manually then you need to use -x switch of jar tool of JDK.

1.8 REFERENCES/FURTHER READING

- ... Kathy Sierra, Bryan Basham, and Bert Bates, “Head First Servlets and JSP”, O’Reilly Media, Inc., 2008.
 - ... Budi Kurniawan, “Java for the Web with Servlets, JSP, and EJB: A Developer’s Guide to J2EE Solutions: A Developer’s Guide to Scalable Solutions”, Techmedia, 2002.
 - ... <https://www.oracle.com/java/technologies/appmodel.html>
 - ... <https://docs.oracle.com/javase/8/docs/technotes/guides/javaws/>
 - ... https://docs.oracle.com/cd/B10018_07/migrate.902/a95110/overview.htm
 - ... Design Patterns: Elements of Reusable Object-Oriented Software – Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides.
 - ... <https://docs.oracle.com/javase/tutorial/deployment/jar/build.html>
 - ... <https://introcs.cs.princeton.edu/java/85application/jar/jar.html>
-

UNIT 2 **BASICS OF SERVLET**

Structure

- 2.0 Introduction
- 2.1 Objective
- 2.2 Introduction of Servlets
- 2.3 HTTP Protocol and Http Methods
 - 2.3.1 HTTP Protocol Overview
 - 2.3.2 HTTP Request
 - 2.3.3 HTTP Response
 - 2.3.4 HTTP Method
- 2.4 Servlet Architecture
- 2.5 Servlet Life Cycle
 - 2.5.1 The init Method
 - 2.5.2 The service Method
 - 2.5.3 The destroy Method
- 2.6 Creating a Servlet
- 2.7 Running and Deployment of Servlet
- 2.8 Summary
- 2.9 Solutions/Answers
- 2.10 Further Readings

2.0 **INTRODUCTION**

In this new era of technology, the Internet is just like a bottomless ocean where you will find anything from a website to a web application. As you know very well, a website is a collection of related web pages that contains static (HTML pages, images, graphics) as well as dynamic files, whereas a Web application is a piece of software with dynamic functionality on the server. Google, Facebook, Twitter are some popular examples of web applications.

Today, you are aware of the need of creating dynamic web pages. For this, you can use server-side programming languages such as Java Servlets, Java Server Pages, ASP.Net and PHP etc. This Unit and the next unit will provide you with more details on Java Servlets. Next th Java Server Pages will be discussed in unit 4 of this block.

You are also aware of core Java concepts and compiling and running of java classes. In this unit, you will learn the basics of Servlet, Servlet-API and life-cycle of the servlet. Servlets are java classes that run on the Java-enabled web server and are widely used for web processing as well as are capable of handling complex requests obtained from the web server. Servlets need some protocol and methods for communicating with the server. This unit will also cover HTTP protocol and methods and guides you on how to write, run and deploy a servlet.

2.1 **OBJECTIVES**

After completion of this unit, you will be able to:

- ... Describe basics of Servlet,
- ... Difference between HTTP request and HTTP response,
- ... Use of HTTP methods,

- ... Use different classes and interface included in Servlet API and some of its methods,
- ... Describe how servlet container maintains the Servlet Life Cycle, and
- ... Write simple servlet for the web applications.

2.2 INTRODUCTION OF SERVLETS

You all are aware of java classes. A servlet is also a java class that is descended from the class `javax.servlet.http.HttpServlet`. So, servlet is a java class, but not all java classes are servlets. In this section, you will learn more about Java Servlet.

Java Servlets are small, platform-independent java programs that run on the Java-enabled web server. It extends from a Java class or rather interface and requires you to implement certain methods so that web container or servlet container (Tomcat, etc.) is able to send execution to your servlet. Basically it creates a class that extends either `GenericServlet` or `HttpServlet`, overriding the appropriate methods, so it handles requests.

Web containers job is to handle the request of the web server; process these requests, produce the response and send it back to the web server. Servlets are executed within the address space of a Web Server.

As Servlet technology is based on java language, it is robust and scalable. It is used to create a program for developing web applications. These programs reside on server side. Java Servlet is the foundation technology for Java server-side programming. JSP (Java Server Pages), JSF (Java Server Faces), Struts, Spring, Hibernate and others are extensions of the servlet technology.

Servlet extends the capabilities of web servers that host applications accessed by means of a request-response programming model. For developing web applications, Java Servlet technology defines HTTP-specific servlet classes. A HTTP Servlet runs under the HTTP protocol. This protocol is an asymmetrical request-response protocol where the client sends a request message to the server, and the server returns a response for requested data. You will get more details on HTTP protocol and methods in the next section of this unit.

There are many interfaces and classes in the Servlet API for creating servlets, such as `GenericServlet`, `HttpServlet`, `ServletRequest`, `ServletResponse`, and `Servlet`, which will be discussed in section 2.4 of this unit.

A java servlet program has a life cycle that defines:

- i. how the servlet is loaded and initiated,
- ii. how a Servlet receives and responds to requests, and
- iii. how it is taken out of service.

The Servlet life cycle will be discussed in detail in section 2.5 of this unit.

2.3 HTTP PROTOCOL AND HTTP METHODS

In the previous section, you have been explained about the servlets which are a server-side scripting language. It follows client-server architecture in which client (browser) sends data to server and it requires some protocol (set of rules) for communication between the two. In this section, you will learn some of the basic protocol and methods that make this communication possible.

Hypertext Transfer Protocol (HTTP) is specially meant to communicate between Client and Server using Web (or Internet). The HTTP is a stateless protocol, it supports only one request per connection. It can be further simplified as; HTTP client connects to the server to send one request only and then it is disconnected. This mechanism facilitates connecting more users to a given server over a period of time.

To run web applications, web browsers communicate with the web servers using the HTTP. When you type any URL in the browser, the browser sends an HTTP Request to the server. The server receives the request and sends back the requested data to the browser. In the following sections, you will learn more about the HTTP Protocol and HTTP methods.

2.3.1 HTTP Protocol Overview

HTTP Protocol is a network protocol used for transferring files on the Internet. HTTP is the primary protocol used for most web applications on the Internet. Whether the application are written in Java, ASP.Net or PHP; all web applications use HTTP. This protocol is the foundation of any data exchange on the Web. HTTP works based on request-response model between a client and the server. An HTTP client sends a request message to a HTTP server. The server returns a response message to the client. As HTTP is a stateless protocol, the current request does not know what has been done in the previous requests. For communication between the different web pages, you need to establish sessions or create cookies. These will be discussed in Unit 3 of this block.

HTTP is also called connectionless protocol, which means that the client establishes a connection with the server before sending a request, and the server sends back response to the client over this connection only. When a response is delivered, the connection between the client and the server is destroyed. If the same client wants to connect to same server again, the client should establish a new connection. So, HTTP is designed as connectionless for the above said reason; the server should share resources equally to the clients who exist all over the world. If one client is connected all the time with the server then the server cannot allocate the time to other clients.

HTTP was initially developed by Tim Berners-Lee and his team in early 1990. HTTP/1.0 defines the basic protocol with some methods such as GET, POST and HEAD, and this version did not support informational status codes. The HTTP/1.1 defines seven HTTP methods GET, POST, HEAD, OPTIONS, TRACE, PUT, DELETE, and also add many headers and facilities to the original HTTP/1.0. Subsequently, in HTTP/2.0 there has been a major revision of the HTTP network protocol. HTTP/2.0 is the first new version of HTTP since HTTP 1.1. HTTP/3 is the third major version of the HTTP used to exchange information on the World Wide Web. When a web page link is clicked on a web page to look for a search or submit a form from your browser, the browser translates your requested URL into a **request message** according to the specified protocol and sends it to the HTTP server. The

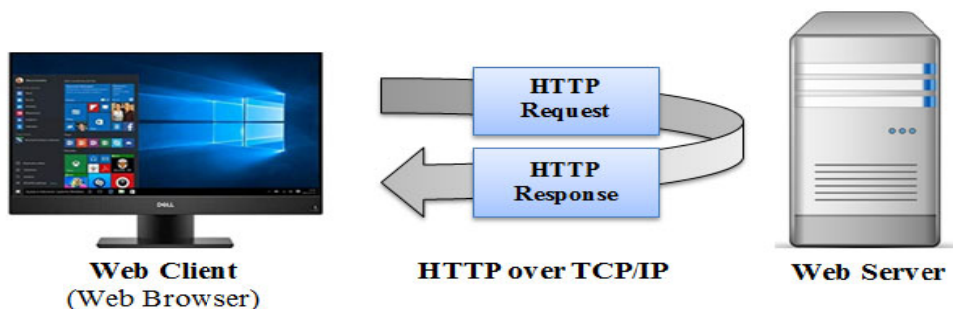


Figure 1: Process of communication between Web Client and Web Server

HTTP Server (Web Server) interprets the request message received, maps the request into a *program* kept in the server, executes the program and returns you an appropriate **response message**. in the form of the resource has been requested or as an error message. In figure 1 this request and response process is displayed.

HTTP messages manage the information exchange between a server and a client. There are two types of HTTP messages: **HTTP requests** sent by the client to the server, and **HTTP responses**, the answer from the server. Both the messages are described below:

2.3.2 HTTP Requests

Whenever a client sends a message to a server it is an HTTP request. As you have learned until now, when the client sends a request to the server, the server returns a response to the client. Web Client sends a request specifying one of the seven HTTP request methods (you can read more about these methods in the next section), the location of the resource to be invoked, protocol version, a set of optional headers and an optional message body.

In a simple terminology, as a client, when you type a URL of the website in the browser, for example, you want to download your Hall-ticket from the IGNOU website and open the website by typing the URL as `http://www.ignou.ac.in/studentzone/hallticket.jsp`; a request is initiated from your computer to the web server. This process sets up a connection from your computer to the host computer where the IGNOU web server is hosted. The domain name from your URL is converted into a machine-readable state known as an IP address by the Domain Name System (DNS) server during the connection setup. The rest part of the URL are path i.e. `/studentzone/` and resource file (`hallticket.jsp`). The client computer, after knowing the IP address of web server, path, resource file, your request goes to the web server using HTTP methods such as GET. Some more technical operations are needed to establish a connection between client and server. A detailed discussion on this is beyond the scope of this unit.

For the requested IGNOU page, your request may look like the following line: method name, path, resource name, and protocol version.

```
GET /studentzone/hallticket.jsp HTTP/1.1
```

You can also find these details using the following example. The 'RequestServlet' example illustrates some of the resources available in the request object. The detailed running procedure is given in section 2.7 of this unit.

Source Code 'RequestServlet' example

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

public class RequestServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        java.util.Date date = new java.util.Date();
        out.println("Current Date & Time: " + date.toString());
        out.println("<h3>Request Information Example</h3>");
        out.println("Method: " + request.getMethod()+"<br>");
    }
}
```

```

        out.println("Request URI: " + request.getRequestURI()+"<br>");
        out.println("Protocol: " + request.getProtocol()+"<br>");
        out.println("PathInfo: " + request.getPathInfo()+"<br>");
        out.println("Remote Address: " + request.getRemoteAddr());
    }
    //doGet
}

```

The key methods included in the above program are described in the table below:

Table 1: Some HttpServletRequest Methods

Method	Description
getMethod()	Returns the HTTP request method
getRequestURI()	Returns the complete requested URI
getProtocol()	Returns the protocol used by the browser
getPathInfo()	only returns the path passed to the servlet
getRemoteAddr()	Returns the IP address of the client

The output of this Servlet should look like this (figure-2):

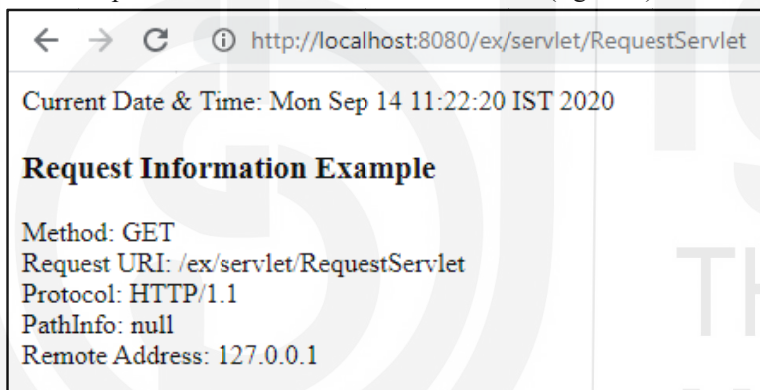


Figure 2: Output Screen for 'RequestServlet' example

As you know that the request message shows the method, Request URI, Protocol, PathInfo and the remote address of client machine. Here, the method is GET, Request URI is the path of your 'webapps' and the Protocol version is HTTP 1.1 (it may be different on your server). This servlet is running on the local machine. So, the address is 127.0.0.1. These details depend on your web server.

You can also check your logs under the local web server. For example, the above program shows the following line in the logs folder under the Tomcat Server:

```
127.0.0.1 - - [14/Sep/2020:11:22:20 +0530] "GET /ex/servlet/RequestServlet HTTP/1.1"
```

2.3.3 HTTP Response

Once a server receives a request, it interprets the request message and responds with an HTTP response message. The server sends back a response to the client containing the version of HTTP we are using, a response or status code, a description of the response code, a set of optional headers, and an optional message body.

HTTP response status codes is used to indicate whether a specific HTTP request has been successfully completed or not. Responses are grouped in five classes:

Table 1: HTTP Response Classes

S.No.	Status Code	Description
1	Informational responses (100–199)	Request has been received and the process is continuing.
2	Successful responses (200–299)	Action was successfully received, understood, and accepted.
3	Redirects (300–399)	Action must be taken to complete the request.
4	Client errors (400–499)	Request contains incorrect syntax or cannot be fulfilled - It means the server failed to fulfil an apparently valid request.
5	Server errors (500–599)	Server failed to fulfil a valid request.

It is not necessary for HTTP applications to understand the meaning of all registered status codes. More details of the status code can be explored at: <https://www.w3.org>

For example, if the response line of HTTP applications is:

HTTP/1.1 200 222

Then it can be interpreted as: HTTP/1.1 is the HTTP version; The HTTP Status 200 indicates the successful processing of the request on the server.

2.3.4 HTTP Method

In the previous section, you read the term method in the context of the request message. In this section, you will learn some frequently used methods for getting response from the server. You are already aware of the HTTP request which is a message that a client sends to a server. To send these requests, clients can use various HTTP methods. These request methods are case-sensitive and should always be noted in upper case. There are various HTTP request methods such as GET, POST, PUT, HEAD, DELETE, OPTIONS and PATCH and each one is assigned a specific purpose.

GET Method

GET method is used to retrieve requested data from a specified resource in the server. GET is one of the most popular and commonly used HTTP request methods. GET request is only used to request data (not modify). It means that the GET method is used to retrieve information and does not make any change in the server. These types of methods are called 'safe' methods.

The following simple HTML programme will demonstrate you how the 'GET' method will work:

```
<html><body>
<form name="InfoForm" action="Info.jsp" method="GET">
  <div> <label>Name</label> <input name="name1" value=""> </div>
  <div> <label>Course</label> <input name="course" value=""> </div>
  <div> <input type="Submit"> </div>
</form>
</body>
</html>
```

The above programme consists of two input boxes i.e. name and course. When you run the above program by using the procedure defined in the section 2.7, the result screen will display as shown in the figure 3.

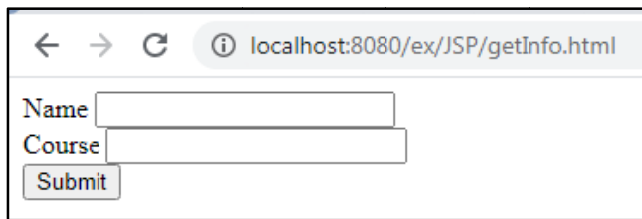


Figure 3: Simple Form for 'GET' method Example

When you put some values in the input boxes i.e. name & course, and press the 'Submit' button as displayed in Figure 3, then action control will transfer on the following HTML code line in the form:

```
<form name="InfoForm" action="Info.jsp" method="GET">
```

So, action will be taken by the resource file:Info.jsp and the method will be GET. Here, GET method is used to retrieve values from the server. Note that the query string (name/value pairs) is sent in the URL of a GET request. After submitting the form, the name/value is visible as like the following:



Here, 'localhost:8080' indicates that you are referring to the local web server on this machine at a 8080 port number. The web page uses the path /ex/JSP of the web server and accesses the resource file 'Info.jsp' and after the resource file name, there is a text ?name1=Poonam&course=MCA which indicates passing additional information to the server in the form of the parameter. In this case, name1 is defined as the name of 'Name' input box (<input name="name1" >) and value of this box is 'Poonam'. In the similar manner, course (<input name="course" >) is the name of 'Course' input box and value of this box is MCA. This example shows how the GET method works. You can use this type of query string to find the data from the database or simply get the parameter value to display on the web page. In the above example, server may use the operation which is defined in the 'Info.jsp' file to display data.

HEAD Method

The HEAD method is similar to GET method but doesn't have a message-body in the response. In a web application, the server replies with a response line and headers section only.

POST Method

Another popular HTTP request method is POST. In web communication, this method is used to send data to a server to create or update a resource. The information submitted using POST request method to the server is archived in the request body of the HTTP request. The information /data sent through the POST method will not be visible in the URL as parameters are not sent along with the URL.

For example, you can use the same form defined in GET method, only change the name of method as POST. So, form contains the following HTML code line for <form> element:

```
<form name="InfoForm" action="Info.jsp" method="POST">
```

When you submit the form by pressing 'Submit' button, information must be sent to the server. In this example, parameters like name and course will be put inside the request body. Like the GET method, parameter and their values are not visible in the URL.

Both the Get and Post method of HTTP protocol is frequently used methods in web programming. Using a GET request, information is sent to the server via URL parameters. On the other hand with a POST method, some additional data is also supplied from the client to the server in the message body of the HTTP request. An advantage of the POST over the GET request is that it is more secure - it cannot be bookmarked, and it is difficult to hack. Also it is not stored in the browser history. This method is, therefore, more commonly used when sensitive information is involved.

There are several other request methods. You may refer to more of these methods from the 'Further Readings' section at the end of this unit.

HTTP and Java

Servlets can be seen as java components that can respond to an HTTP request. There are methods in Java Servlets corresponding to each of the HTTP methods. The seven HTTP methods map on to seven servlet methods of the same name with a "do" in front of the method. For example, GET maps on to doGet(). Mostly you can use either GET or POST method while developing a web page.

☛ Check Your Progress 1

1. What is a Servlet? Define the workflow of a servlet,

2. What are the basic Features of HTTP?

3. What is HTTP Message? What are request and response in the context of HTTP?

4. How is GET method different to POST method?

2.4 SERVLET ARCHITECTURE

Servlets are the Java program that runs on the Java-enabled web server. The client or web browser sends the HTTP request to the web server. The web server receives the request and passes the request to the servlet container and subsequently the servlet container forwards this request to the corresponding servlet. The servlet processes the request and generates the response in the form of output. This process may require communication to a database and may also invoke a web service, or computing the response directly. After processing, the servlet builds the response object and sends it back to the Web Container.

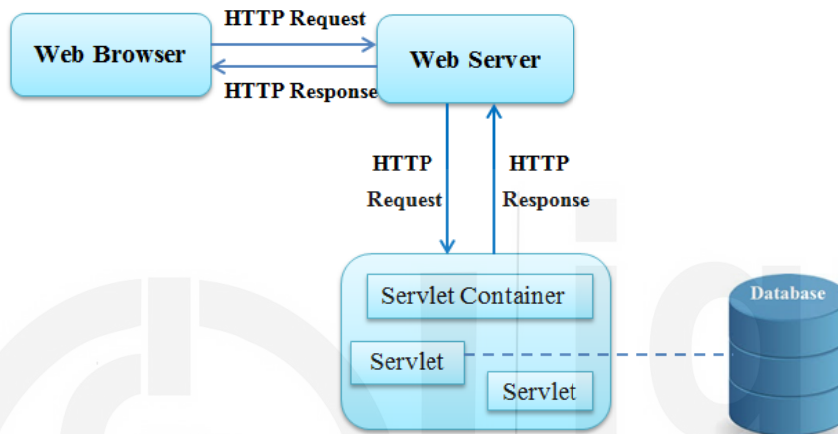


Figure 4: Servlet Architecture

The servlet container sends the response back to the web server; the web server sends the response back to the browser and the browser displays it on the screen. The diagram shows the execution of servlet in Figure 4.

You need to use Servlet API to create servlets. These APIs allow developer to build programs that can run with a web server. The Servlet APIs (Application Programming Interface) contains two packages: `javax.servlet` and `javax.servlet.http`. These two packages make up the servlet architecture. The `javax.servlet` and `javax.servlet.http` packages provide interfaces and classes for writing servlets. The `javax.servlet` package contains generic interfaces and classes that are implemented and extended by all the servlet. The `javax.servlet.http` package contains a number of classes and interfaces that are used to create HTTP protocol-specific Servlets.

GenericServlet Class

The `javax.servlet` package provides an abstract class called `GenericServlet` that implements **Servlet**, **ServletConfig** and **java.io.Serializable** interfaces. A servlet can directly extend it. The subclass of `GenericServlet` is `HttpServlet`. It can handle any type of request. You may create a generic servlet by inheriting the `GenericServlet` class and implementing the `service()` method. The prototype of `service()` method is defined as follows:

```
public abstract void service(ServletRequest req, ServletResponse res)
throws ServletException, IOException;
```

The two objects of `service()` method are `ServletRequest` and `ServletResponse`. The `ServletRequest` object holds the information that is being sent to the servlet and the `ServletResponse` object is to assist a servlet in sending a response back to the client.

The `javax.servlet.ServletException` is a general exception that a Servlet can throw. `IOException` is a checked exception and is thrown when there is any input/output file operation issues while application is performing certain tasks accessing the files.

Servlet Interface

All servlets must implement the Servlet interface either directly or indirectly. Java servlets class does not have a `main()` method, so all servlets must implement the `javax.servlet.Servlet` interface. It defines five methods, including three life-cycle methods. You may read the life-cycle methods such as `init()`, `service()` and `destroy()` in the subsequent section 2.5 and other two methods are `getServletConfig()` and `getServletInfo()`.

ServletConfig Interface

An object of `ServletConfig` is created by the servlet container for each servlet. This object is used to pass configuration related information to a servlet during start up. The `getServletConfig()` method is used to return the object of `ServletConfig`. It contains name/value pairs of initialization parameters for the servlet. It defines four methods such as `getServletContext()`, `getServletName()`, `getInitParameter()` and `getInitParameterNames()` for accessing this information.

Serializable interface

The Java Serializable interface (`java.io.Serializable`) is a marker interface. It means that it contains no methods. Therefore, a class implementing Serializable does not have to implement any specific methods.

HttpServlet Class

The `javax.servlet.http` package contains a number of classes and interfaces that are used to create HTTP protocol-specific Servlets. The abstract class `HttpServlet` is a base class for user defined HTTP Servlets which provide methods. The basic methods such as `doGet` and `doPost` are for handling HTTP-specific services. When you are developing your own servlets then you can use `HttpServlet` class which is extended from `GenericServlet`.

```
public abstract class HttpServlet
```

Unlike with `GenericServlet`, when you extend `HttpServlet`, you can skip implementing the `service()` method. The `HttpServlet` class has already implemented the `service()` method. The following are the prototype of the `service()` method:

```
protected void service(HttpServletRequest req, HttpServletResponse res)  
throws ServletException, IOException
```

When the `HttpServlet.service()` method is invoked, it first reads the method type stored in the request, and on that basis it determines the method to be invoked. For example if the method type is GET, it will call `doGet()` method, and if the method type is POST, it will call `doPost()` method. These methods have the same parameter as the `service()` method.

There are many interfaces in `javax.servlet.http` package but the three important interfaces `HttpServletRequest`, `HttpServletResponse` and `HttpSession` are described below:

Interface HttpServletRequest

The `HttpServletRequest` interface captures the functionality for a request object that is passed to a HTTP servlet. It provides access to an input stream and allows the servlet

to read data from the client. The `HttpServletRequest` interface extends the `ServletRequest` interface.

public interface `HttpServletRequest` extends `ServletRequest`

The servlet container creates an `HttpServletRequest` object and passes it as an argument to the servlet's service methods (`doGet`, `doPost` etc). The `HttpServletRequest` interface has many methods. Some of the commonly used methods of `HttpServletRequest` are:

1. `getParameter()`

It returns the value associated with a parameter sent to the servlet as a part of a GET or POST request. The name argument represents the parameter name.

public String `getParameter(String name)`

2. `getQueryString()`

It returns the query string that is contained in the request URL if any. This method returns null if the URL does not have a query string. A query string is defined as any information following a ? character in the URL.

public String `getQueryString()`

3. `getCookies()`

The `getCookies()` method returns an array of `Cookie` objects found in the client request. This method does not take any parameters and throws no exceptions. Cookies are used to uniquely identify clients to servlet. In case of no cookies in the request, an empty array is returned.

public `Cookie[]` `getCookies()`

4. `getHeader()`

It returns the value of the specified request header as a `String`. In case if the request did not include a header of the specified name, this method will return null. If there are multiple headers with the same name, this method returns the first header in the request. The header name is case insensitive.

public String `getHeader(String name)`

5. `getMethod()`

It returns the name of the HTTP method used to make the request. Typical return values are 'GET', 'POST', or 'PUT'

public String `getMethod()`

6. `getSession()`

It returns the current session associated with the request or if the request does not have a session, creates one.

```
public HttpSession getSession(boolean create)
public HttpSession getSession()
```

Interface HttpServletResponse

The HttpServletResponse extends the ServletResponse interface to provide HTTP specific functionality while sending a response to the client. It provides access to an output stream and allows the servlet to send data to the client.

```
public interface HttpServletResponse extends ServletResponse
```

This interface has many methods; some are described in the following sections.

1. **getWriter() Method**

It obtains a character-based output stream that enables text data to be sent to the client.

```
public PrintWriter getWriter()
```

2. **addCookie() Method**

This method is used to add a Cookie to the HttpServletResponse object. It returns no value and throws no exception. This method can be called many times to set more than one cookie.

```
public void addCookie(Cookie cookie)
```

3. **addHeader() Method**

This method is used to add a response header with the given name and value.

```
public void addHeader(String name, String value)
```

4. **getOutputStream() Method**

It obtains a byte-based output stream that enables binary data to be sent to the client.

```
public ServletOutputStream getOutputStream()
```

5. **sendError() Method**

This method sends an error to the client in the response object. The error consists of only the 'int' status code and returns no value.

```
public void sendError(int statusCode) throws IOException
```

The following sendError() method sends an error to the client in the response object. The error consists of an 'int' status code and a String message. It returns no value.

```
public void sendError(int statusCode, String message) throws IOException
```

6. **sendRedirect() Method**

This method redirects the client to the passed-in URL, which must be an absolute URL. It returns no value.

Note- You can reference the Servlet-API from your Tomcat installation at <https://tomcat.apache.org/tomcat-9.0-doc/servletapi/index.html>

Interface HttpSession

This is an important interface used to identify a user across more than one page request or visits to a website and stores information about that user. Servlet Container uses this interface for the creation of session between HTTP client and HTTP server. Using HttpSession, you can maintain state (data) between transactions. The HttpServletRequest interface provides two methods such as getSession() and getSession(boolean create) to get the object of HttpSession. Both the methods getSession() and getSession(boolean create) are explained in the “HttpServletRequest interface” section of this Unit. The signature of this interface is as under:

```
public interface HttpSession
```

The commonly used methods of HttpSession interface are defined below. You can find an example of these methods in the next unit of this course.

1. getId() Method

The getId() method returns a string containing a unique identifier assigned to the current HttpSession.

```
public String getId()
```

2. getCreationTime() Method

The getCreationTime() method returns the time in which the session was created.

```
public long getCreationTime()
```

3. getLastAccessedTime() Method

This method returns the last time the client sent a request associated with this session object.

```
public long getLastAccessedTime()
```

4. getAttribute() Method

This method is used to return the value of given parameter from the session.

```
public Object getAttribute(String name)
```

5. setAttribute() Method

This method is used to set the attribute in session.

```
public void setAttribute(String name, String value)
```

6. invalidate() Method

This method is used to destroy the session.

```
public void invalidate()
```

For example, session.invalidate();

Note: You can find more methods of HttpSession interface in the following link:
<https://docs.oracle.com/javaee/5/api/javax/servlet/http/HttpSession.html>

☛ Check Your Progress 2

1. What is Servlet interface and what is the use of it?

2. Write the difference between GenericServlet and HTTPServlet?

3. What is ServletConfig?

4. What is ServletContext?

2.5 SERVLET LIFE CYCLE

A Java Servlet has a life cycle that defines the steps in the whole processing of a servlet. This includes:

1. The Servlet loading and initialization,
2. How servlet receives and responds to requests, and
3. How a servlet is taken out of service.

The servlet life cycle is very simple object oriented design. A servlet life cycle includes the entire process from its creation to the destruction. A servlet is first initialized and then it serves to zero or more service requests until the service ends and shuts down. Servlet is loaded only once, and it stays resident in the memory till it is servicing a request. **Servlet container manages the** entire life cycle of a Servlet, using the **javax.servlet.Servlet** interface. Every servlet you write must implement the **javax.servlet.Servlet** interface either directly or indirectly. The Servlet interface defines all the methods of servlet life cycle such as `init()`, `service()` and `destroy()`.

Now let us discuss the life cycle methods in detail.

2.5.1 The `init()` Method

The `init()` method is where the servlet's life cycle begins. The servlet container calls this method after the servlet class has been instantiated. This method is called exactly once by the servlet container to indicate to the servlet that the servlet is being placed into service. In this method, servlet creates and initializes the resources, including the data members that it will be using while handling requests.

The signature of this method is defined as follows:

This method takes a ServletConfig object as a parameter. The ServletConfig object contains the servlet's configuration and initialization parameters. This init() method can also throw a ServletException. The ServletException is the most important exception in servlet programming. If the servlet cannot initialize the resource to handle the request, then the method will throw an execution.

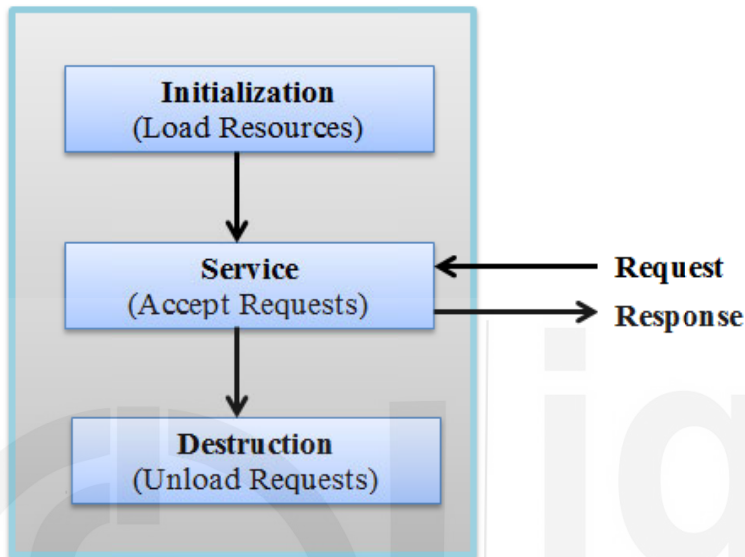


Figure 5: Servlet life cycle

2.5.2 The service() Method

The service() method is used to handle incoming requests and generate a response. This method is called by the servlet container and cannot start servicing the request until the servlet's init() method has been executed. The most common use of this method is in the HttpServlet class. The HttpServlet class provides http specific methods such as doGet, doPost, doHead, doTrace etc. In order to generate a response you should override the doGet or doPost methods as per your requirement.

This method has the following signature:

```
public void service(ServletRequest req, ServletResponse res) throws  
ServletException, java.io.IOException
```

This method implements a request and response paradigm. The servlet container passes two objects, ServletRequest object and ServletResponse object. The ServletRequest object contains the client's request and ServletResponse object contains the servlet's response. Both objects are important because they enable you to write code that determines how the servlet fulfils the client request. The service() method may throw the ServletException and IOException while processing the request. This method throws a ServletException if any exception occurs that interferes with the servlet's normal operation and also throw a java.io.IOException if an input or output exception occurs during the execution of service() method.

2.5.3 The destroy() Method

The destroy() method is called only once when all the processing of the servlet is over and it is at the end of its life cycle. When your application is stopped or Servlet Container shuts down, this method will be called to close down those shared resources and other clean up required before the servlet is taken out of service. This is the place where any resources that were created in init() method will be cleared up.

The signature of this method is defined as follows:

```
public void destroy( )
```

2.6 CREATING A SERVLET

In this section, we will try to create a basic servlet. This will help you in becoming familiar with the basic parts of the servlets. Here is a simple servlet program code for printing a text message such as 'Welcome to the IGNOU Family!'.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
public class WelcomeServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Servlet Testing</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("Welcome to the IGNOU Family!!");
        out.println("</BODY></HTML>");
    }
}
```

Write the above code in a notepad and save it as WelcomeServlet.java on your PC. You will find running procedure for the servlet in the next section.

2.7 RUNNING AND DEPLOYMENT OF SERVLET

In this section, you will find the installation process of Apache's Tomcat. You can use it for running and deployment of your Servlets as well as JSP programs. You will study JSP in Unit 4 of this course. It is the best practice to use IDE for web development. Here, you can go through the **simple steps for running your servlet and JSP program using Apache Tomcat** without any IDE. Apache Tomcat is an open source web server for testing servlets and JSP programs. You can use Notepad to write your program. Also, there are some open source Integrated Development Environment (IDE) for developing J2EE applications. NetBeans IDE supports the development of all Java application types, including Java SE (including JavaFX), and J2EE. More about the IDEs you will learn in MCSL-222 course.

The following steps are based on Windows Operating System(Windows XP/8/10):

Step 1 - Download and install Java Development Kit.

Step 2 - Download the latest version of **Tomcat Server** and install it on your machine.

Step 3 - After successful installation of Java and Tomcat, you can set the environment variables by using the Environment Variables option. For this, do right click on System icon → properties → Advanced System Setting → environment variables. Now, click on 'New' button and enter the variable name as JAVA_HOME and in variable value write the path of Java installation directory and click on the 'ok' button. The following screen comes after the selection of the step3.

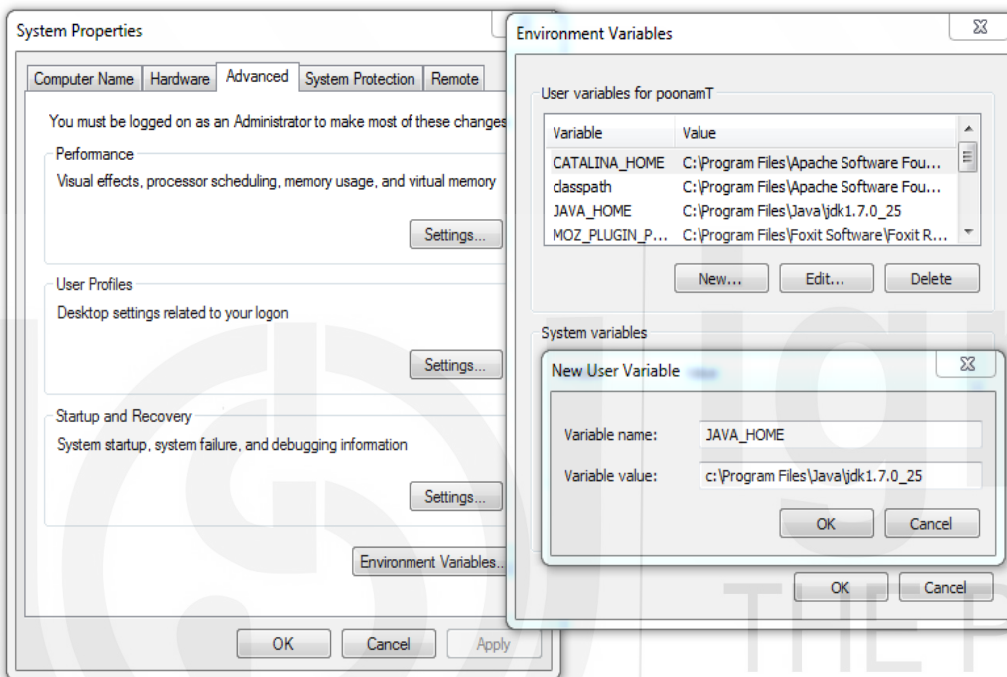


Figure 6: Screen for setting the environment variable.

In a similar manner, you can set the variable name and variable value for the following environment variable by using step 3:

variable name	variable value:
classpath	C:\Program Files\Apache Software Foundation\apache-tomcat-7.0.37\lib\servlet-api.jar;C:\Program Files\Apache Software Foundation\apache-tomcat-7.0.37\lib\jsp-api.jar;C:\Program Files\Apache Software Foundation\apache-tomcat-7.0.37\lib\msbase.jar;C:\Program Files\Apache Software Foundation\apache-tomcat-7.0.37\lib\msutil.jar;C:\Program Files\Apache Software Foundation\apache-tomcat-7.0.37\lib\mysqlserver.jar;
Path	C:\ProgramFiles\Java\jdk1.7.0_25\bin;C:\ProgramFiles\Java\jdk1.7.0_25\lib;
CATALINA_HOME	C:\Program Files\Apache Software Foundation\apache-tomcat-7.0.37
JRE_HOME	C:\Program Files\Java\jdk1.7.0_25\jre

tep 4 - After installation, you will find the tomcat folder, which contains the following folders:

- ... **bin** (binary files for Tomcat and friends),
- ... **conf** (configuration files),
- ... **lib** (library JAR files),
- ... **logs** (server log files),
- ... **temp** (temporary files),
- ... **webapps** (area Tomcat looks for web application, it contains JSP files, Servlets and other content), and
- ... **work** (working space for holding translated JSP files).

You can see these folders under your web application like the following figure 7.

Step 5 - Create directory “ex” under the ‘webapps’ folder as per the following figure 7:

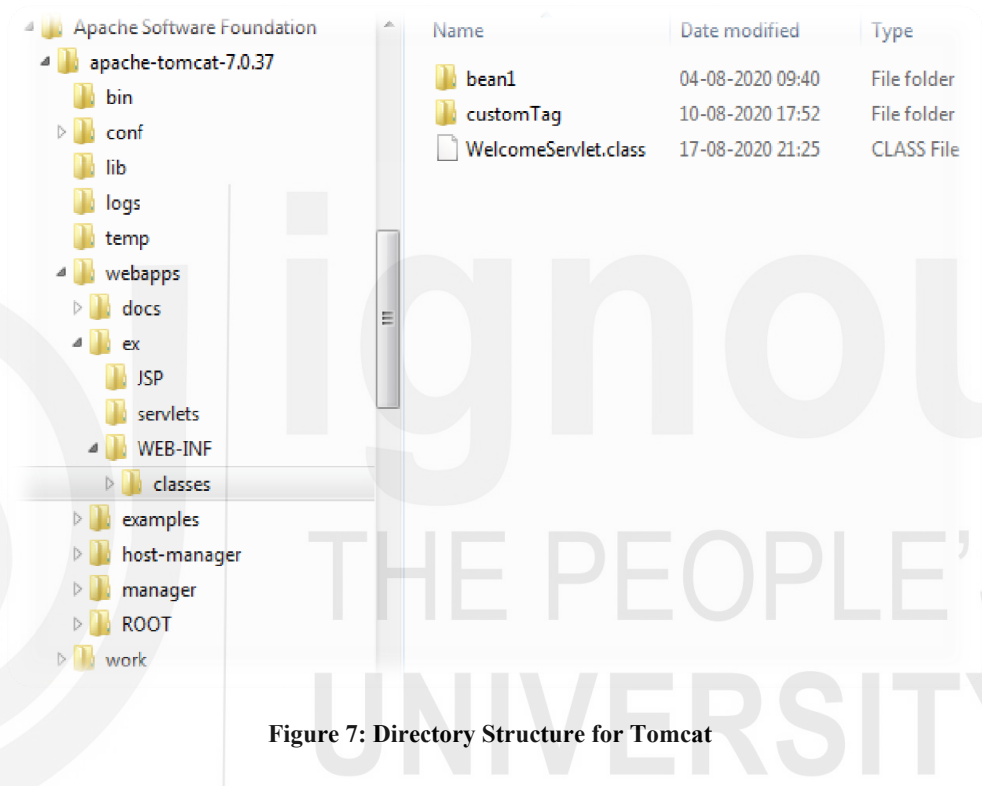


Figure 7: Directory Structure for Tomcat

Step 6 – Create one ‘servlets’ folder for source file and one “JSP” folder (for JSP program) under ‘ex’ folder. Also create a directory ‘WEB-INF’ under the ‘ex’ folder. Under WEB-INF folder create a folder named ‘classes’. All your java classes used in your web application should be placed in ‘classes’ folder.

Step 7 - Copy web.xml file from ROOT directory and paste into “WEB-INF” folder under the “ex” folder.

Now, the Java and Tomcat installation process is completed.

Compiling and Running the Servlet

Step 8 – Under the ‘servlets’ folder, place your servlet source file. Now, you will compile this servlet by using the following command at the Command Prompt:

```
C:\Program Files\Apache Software Foundation\apache-tomcat-7.0.37\webapps\ex\servlets>javac WelcomeServlet.java
```




Figure 8: Command Prompt for Step 8

Step 9- After successful compilation, 'WelcomeServlet.class' file will be created. Place this file in the location: C: / Program Files/.../webapps/ex/Web-INF/classes folder.

Step 10 – In the deployment descriptor (web.xml) file under the WEB-INF folder, write the following code as the sub-element of <webapp> element.

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

<servlet-mapping>
  <servlet-name>WelcomeServlet</servlet-name>
  <url-pattern>/servlet/WelcomeServlet/*</url-pattern>
</servlet-mapping>
```

Step 11- Start Tomcat Server. In any case, if Tomcat Server is not running from your program menu or shortcut, run the 'startup' command from Command Prompt like the following way:

C:\Program Files\Apache Software Foundation\apache-tomcat-7.0.37\bin>startup



Step 12- Open new tab in the browser or open new window and type the following URL to execute your servlet.

http://localhost:8080/ex/servlet/WelcomeServlet

Congratulations!! Your first servlet is successfully created and run at Tomcat Server. Now, you are able to create a servlet. This is a simple servlet example. In the next Unit of this course, you will learn some other complex servlets with database connectivity.

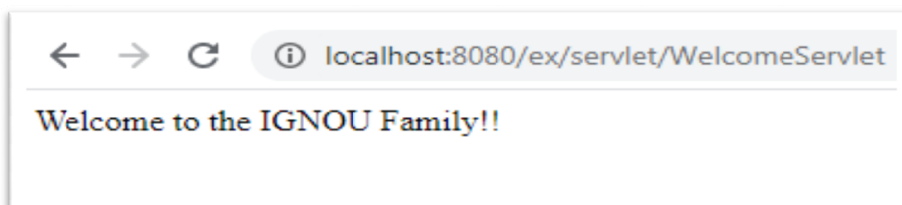


Figure 9: Output Screen for 'WelcomeServlet' program

Steps to Create Servlet Application in Netbeans IDE

In the above section, you have created your first Servlet Application but without using any Integrated Development Environment (IDE). Using IDE, you can easily create Servlet Applications. Here are steps to create Servlet Application in Netbeans IDE. Installation process are given in the lab manual.

Now, start Netbeans and performs the following steps:

Select File -> New Project (CTRL+SHIFT+N) from Menu. In this window you can select Java Web category for your new Servlet/JSP project and select the Next button.

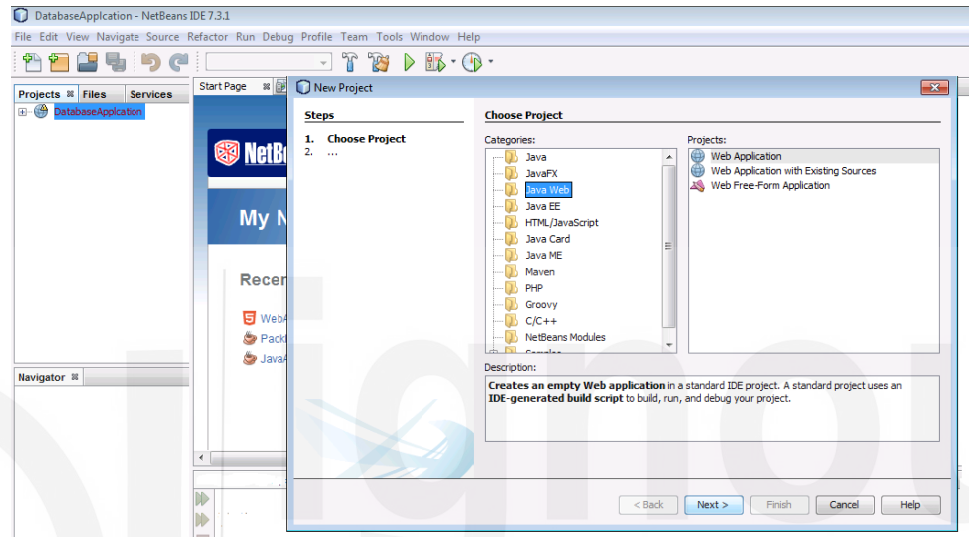


Figure 10: New Project Window

In the next window you can enter project name and location of your choice. For example, you have given the project name as MyProject and selected a Folder C:\Projects for location as shown in the figure-11 and select the Next button.

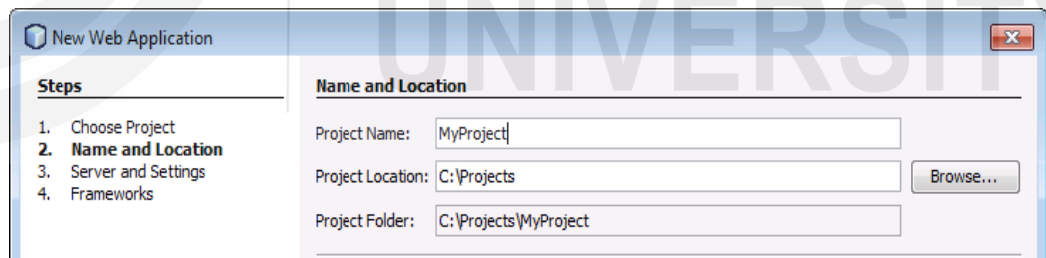
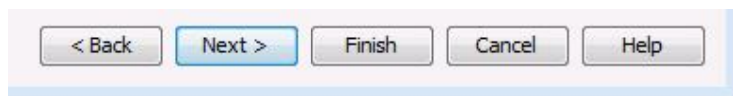


Figure 11: Selection of Name and Location of Project

Now a window appears with Server settings, you can select Finish button from the button Panel.



The complete directory structure required for the Servlet Application will be created automatically by the IDE.

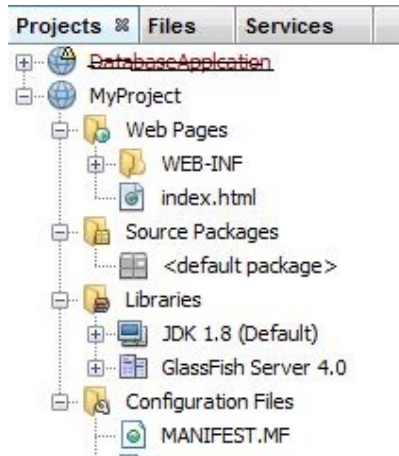


Figure 12: Directory Structure

Now you are ready to create your web application. To create a Servlet, open Source Package, right click on default packages -> New -> Servlet. Give a Name to your Servlet class file and click on the Next button.

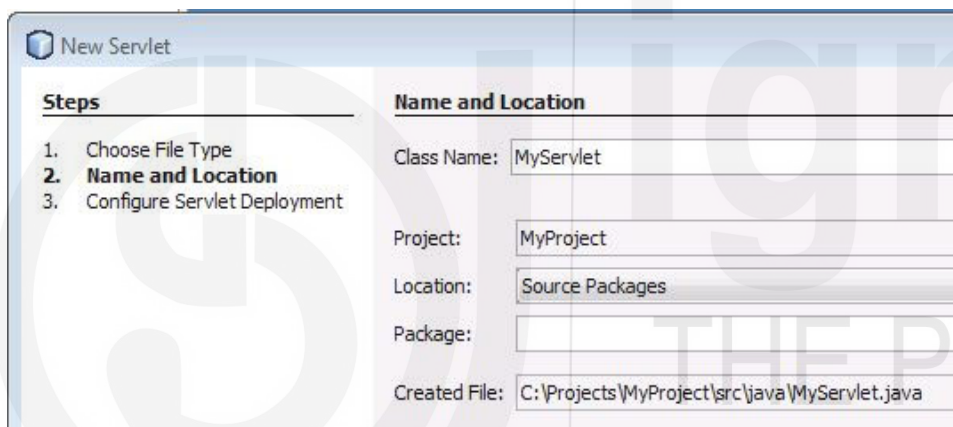


Figure 13: Name and Location window for new Servlet

In the next window you can change servlet name and patterns and finally click on finish button.

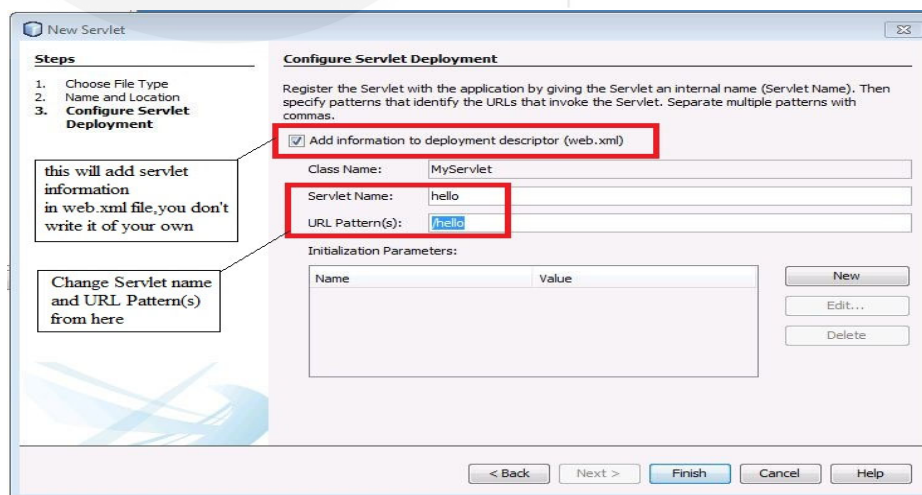


Figure 14: Configure Servlet Deployment Window

Now that Servlet class is ready, you can change the code as per your requirement . Once this is completed, edit the index.html file and add the following code in the index.html. This file is read as the first page of the web application.

```
<h1>Click here to go <a href="hello" >MyServlet Program</a></h1>
```

Edit web.xml file. You can see here the specified url-pattern and servlet-name, this means when hello url is accessed your Servlet file will be executed. If the index.html is not listed as welcome file in web.xml and then you can add this file.

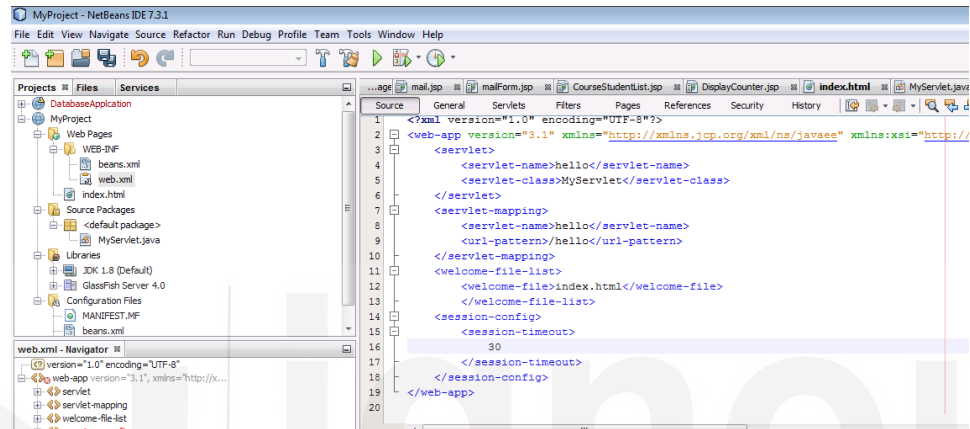


Figure 15: Web.xml file

Now Run your application, right click on your Project name and select **Run**. Click on the link created, to open your Servlet and your first servlet in NetBeans will run. The output of this servlet is shown in Figure-16.



Figure 16:Output of first Servlet program in NetBeans

Check Your Progress 3

1. Which interface contains Servlet life cycle methods?

.....

.....

.....

2. Explain servlet life cycle methods.

.....

.....

.....

3. What is the difference between init() and destroy() methods in servlet?

4. Write a servlet program to display “We are student of SOCIS, IGNOU!!”.

2.8 SUMMARY

This unit introduced you to the basics of Servlet. Servlet is a Java class that creates dynamic content. It takes user request which was sent from the web browser, for processing. Java Servlet is an established technology used for creating dynamic web applications.

After defining the basics of servlet, the unit described the HTTP protocol and HTTP methods. HTTP refers to Hyper Text Transfer Protocol. It is a protocol which governs any data exchange on the Web. HTTP is a client-server protocol i.e. each request is sent by the client to a server that handles it and provides an answer, called the response. While communicating with the Server, clients can use various requests methods. Two of them i.e. GET, and POST are widely used methods that take part in requesting the data in applications.

This Unit also introduced you two packages i.e. javax.servlet and javax.servlet.http that make up the servlet architecture. The two main classes GenericServlet and HttpServlet are defined in Servlet-API. The GenericServlet class provides implementations for all methods; most of them are blank. You can extend GenericServlet and override only methods that you need to use. If you are going to create a web application, then you can extend the HttpServlet class. In the life cycle of a servlet, you have learned three methods such as init(), service(), and destroy() methods. At the end of the unit, you are able to write a simple servlet.

2.9 SOLUTIONS/ANSWERS

Check Your Progress 1

1. A servlet is a small Java program that runs on a web server. Servlets are often run when the user clicks a web link, submits a form or performs some other type of action on a website. Servlets receive and respond to requests of the clients/users of web applications. Using servlets, dynamic web pages are created. It can be seen as a mediator between the incoming HTTP request from the browser and the database. Servlet is a robust server-side programming language.

When a web client sends an HTTP request to web server, the webserver receives the request and passes the request to the servlet container. The servlet container is responsible for instantiating the servlet or creating a new thread to handle the request. The servlet container forwards this request to the corresponding servlet. The servlet processes the request and generates the response in the form of output. During processing, servlet follows all the phases of life cycle. When servlet container shuts down, it unloads all the servlets and calls destroy() method for each initialized servlets.

2. The Hypertext Transfer Protocol (HTTP) is an application-level protocol for developing distributed applications. This protocol is the foundation for data communication for the WWW. The basic features of HTTP are as follows:
 - ... HTTP is a request and response protocol.
 - ... HTTP is a media independent protocol.
 - ... HTTP is a stateless protocol.
3. HTTP messages consist of textual information encoded in ASCII format. It is used to show how data is exchanged between client and server. There are two types of messages: requests sent by the client for action to the server and responses or the answer from the server. It is based on client-server architecture. An HTTP client is a program that establishes a connection to a server to send one or more HTTP request messages. An HTTP server is a program that accepts connections to serve HTTP requests sent by the clients and send HTTP response messages to the client.
4. Both GET, and POST method are used for transfer of data from client to server in HTTP protocol. The main difference between POST and GET method is that GET carries request parameter appended in URL string while POST carries request parameter in message body which makes it a more secure way of transferring data from client to server in HTTP protocol. GET method can send a limited amount of data to the server, while POST method can send data in bulk to the server.

Check Your Progress 2

1. Servlet interface is an API for servlets. Every Servlet should either implement the servlet interface or extends the class which already implements the interface. `javax.servlet.GenericServlet` and `javax.servlet.http.HttpServlet` are the Servlet classes that implement the Servlet interface; hence, every servlet should either implement the Servlet interface directly or by extending any of these classes.
2. `GenericServlet` is an abstract class that implements Servlet interface while `HttpServlet` abstract class extends the `GenericServlet` class. `GenericServlet` class is a base class for `HttpServlet`. `GenericServlet` does not support any protocol while `HttpServlet` support HTTP and HTTPS protocol. `HttpServlet` can handle cookies and session but `GenericServlet` cannot handle them.
3. `ServletConfig` interface belongs to the package `javax.servlet.ServletConfig`. It is used for passing the configuration parameters to the servlet. Each Servlet has a separate `ServletConfig` object. Parameters of `ServletConfig` are defined under `<init-param>` tags in `web.xml` file.
4. Each web application has a common `ServletContext`. All the servlets in the web application can access the `ServletContext`. It has the web-application information and resources, which are common and accessible to all the servlets present in the web application. `ServletContext` is common for all the servlets in the web application.

Check Your Progress 3

1. Servlet interface contains the common methods for all servlets i.e. provides the common behaviour for all servlets. All the three lifecycle methods of a Servlet are defined in Servlet interface, `javax.servlet.Servlet`. The Servlet Container calls the `init()` after instantiating the servlet and indicates that the servlet is ready for service. The `service()` is the method that accepts the

- request and response only after `init()` has been completed. The servlet is terminated by calling `destroy()` method.
2. A servlet life cycle includes the entire process from its creation till the destruction. The servlet is initialized by calling the `init()` method. The servlet calls `service()` method to process a client's request. `Destroy()` method is used to destroy the servlet. It is called only once at the end of the life cycle of a servlet.
 3. The `init()` method is used to create or load objects used by the servlet to handle its requests while the server calls a servlet's `destroy()` method when the servlet is about to be unloaded. In the `destroy()` method, a servlet should free any resources allocated during initialization and shutdown gracefully. Hence this acts as an excellent place to deallocate resources such as an open file or open database connection.
 - 4.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
public class studentServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Servlet program</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("We are student of SOCIS, IGNOU!!");
        out.println("</BODY></HTML>");
    }
}
```

2.10 FURTHER READINGS

- ... Jason Hunter, and William Crawford “Java Servlet Programming”, O'Reilly Media, Inc., 2001.
- ... Kathy Sierra, Bryan Basham, and Bert Bates, “Head First Servlets and JSP”, O'Reilly Media, Inc., 2008.
- ... <https://www.w3.org/>
- ... <https://www.liquidweb.com/kb/installing-tomcat-9-on-windows/>
- ... <https://docs.oracle.com/cd/E19857-01/820-0261/abxbh/index.html>
- ... <https://tomcat.apache.org/tomcat-9.0-doc/servletapi/index.html>
- ... <https://www.w3adda.com/servlet-tutorial>
- ... https://www.w3schools.com/tags/ref_httpmethods.asp
- ... <https://docs.oracle.com/javaee/5/api/javax/servlet/http/HttpSession.html>

UNIT 3 SESSION MANAGEMENT AND DATABASE CONNECTIVITY IN SERVLET

Structure

- 3.0 Introduction
- 3.1 Objectives
- 3.2 Session Management
 - 3.2.1 HttpSession
 - 3.2.2 Cookies
 - 3.2.3 URL Writing
 - 3.2.4 Hidden Fields
- 3.3 Servlet Collaboration
 - 3.3.1 Using RequestDispatcher Interface
 - 3.3.1.1 forward() Method
 - 3.3.1.2 include() Method
 - 3.3.2 Using HttpServletResponse Interface
 - 3.3.2.1 sendRedirect() Method
 - 3.3.3 Using ServletContext Interface
 - 3.3.3.1 setAttribute() Method
 - 3.3.3.2 getAttribute() Method
- 3.4 Database Connectivity
 - 3.4.1 Insert data into Database
 - 3.4.2 Retrieve Data from Database
- 3.5 Summary
- 3.6 Solutions/Answers to Check Your Progress
- 3.7 References/Further Reading

3.0 INTRODUCTION

In the previous unit, you have already gone through the basics of Servlets in detail which are a server side programming language. As you know that the servlets are used for dynamic web application. Several users interact with such web applications simultaneously. Do you know how to manage this dynamic application among the users? A strategy called session management is applied in these applications. In this unit, you will learn more about session management. In Java Servlet, session is managed through different techniques such as HttpSession object, Cookies, URL rewriting and Hidden Form field. For example, when you check your result on the IGNOU website and put your roll number in the input field, it shows that some roll numbers might appear for your selection. It is all possible by the cookies. You will find explanations and examples of these techniques used in session management.

In addition, this unit introduced you to Servlet Collaboration which is all about sharing information among the servlets. The Servlet-API provides the RequestDispatcher, HttpServletResponse and ServletContext interface to achieve Servlet Collaboration. RequestDispatcher interface is useful for forwarding the request to another web resource and including the resource in the current servlet. HttpServletResponse interface makes available a method sendRedirect to communicate with others. A servlet can also share information among multiple

servlets by using `setAttribute` and `getAttribute` method of `ServletContext`. This unit explains to you how the servlets communicate with each other using the methods defined in these three interfaces. Apart from these, this unit also enlightens you about database access using Java Database Connectivity (JDBC) technology.

In the previous unit, you have already gone through the procedure of compiling as well as after compiling servlet made entries in deployment descriptor (`web.xml`) file and invoked them from a web browser. The previous Unit also defined the creating and running procedure of servlet in NetBeans. This Unit is also devoted to the servlets. So, there is no need to define the same procedures again.

3.1 OBJECTIVES

After going through this unit, you will be able to:

- ... Describe how to manage session between servlets,
- ... Connect servlet with database,
- ... Use `forward()` and `include()` method,
- ... Share information between servlets,
- ... Use `setAttribute()` and `getAttribute()` in servlets, and
- ... Insert and retrieve data to/from database.

3.2 SESSION MANAGEMENT

In the previous unit, you studied the Java Servlets used to create dynamic content-based web pages or web applications. Dynamic web pages are different from static web pages in which web server creates a web page when a web client or user requests it. For example, when you check your online results on IGNOU website, different pages are generated for different students by the IGNOU web server depending on your enrolment number.

We all know that HTTP is a stateless protocol which is explained in the previous Unit. It means that the HTTP protocol does not remember information when client or user communicates with the server. Whenever you send a request to the server through HTTP, the HTTP treats each request as a new request. But sometimes in web applications, clients are doing some important work such as online shopping, online banking etc. In that situation, it is necessary to know about the client and remember the client's request accordingly. For example, an online banking application system enables many clients to do their different activities like checking account balance(s), obtaining statements and making financial transactions simultaneously. For maintaining each client's session, you can use session management.

Before going into session management details, you should know about the two important terms, i.e. session and state, which are necessary for implementing business transactions across multiple clients. These are as under:

Session: The server should recognize a series of requests from the same user which form a single working 'session'. For example, in net banking application, each client can be differentiated from another client by associating a unique identifier in request and response within a specific working session.

State: The server should be able to remember information related to the previous request and other business transaction that are made for that request. For example, in net banking application, state comprises client information such as account number and amount transaction made within the particular session.

Do remember one thing-- session management does not change the nature of HTTP protocol i.e., stateless feature provides a way to remember the client information. Session management is also known as session tracking, which permits Servlet/JSP to maintain information about a series of request from the same client. It is a mechanism to store session information for each client. For example, When a user visits any web application, unique identification information about the user is stored in an object available during the particular session until the user quits the web application.

There are four ways to manage sessions: HttpSession object, Cookies, URL rewriting, and hidden fields.

3.2.1 HttpSession Object

Any dynamic website or web application uses the concept of sessions and Session object to store data for a particular user. For example, when you visit any web application for the first time, you have entered login name and password. This information might be stored in session variable and maintained by the servlet container during your visit to web application so that it can be accessed when needed. When your session is started, the requesting browser is allotted a unique id for each of the clients for identifies the client. This Session object is denoted by javax.http.HttpSession interface.

You have learned about the HttpSession interface under the Servlet API in the previous Unit. Servlet API provides session management through HttpSession interface. This HttpSession interface provides a mechanism to identify a user to examine who is visiting a web application and to store the user information. Servlet Container creates a session id for each user. You can maintain state between the transactions by using methods of HttpSession interface.

Example -1

The following example will explain to you about session management using the methods defined in the HttpSession interface.

Following servletSession.java is servlet program that uses session tracking to keep track of how many times a particular user has accessed a page and to display some details of the current session such as Session identifier, Creation Time and Last Access Time. The source code of this program is listed below:

servletSession.java

```
import java.io.*;
import java.util.Date;
import javax.servlet.*;
import javax.servlet.http.*;
public class servletSession extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws
    ServletException, IOException
    {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        HttpSession session = req.getSession(true);
        Integer ct = (Integer) session.getAttribute("ct");
        if (ct == null)
        { ct = new Integer(1); }
```

```
else { ct = new Integer(ct.intValue()+1); }  
session.setAttribute("count", ct);  
out.println("Session Details: ");  
out.println("<br/>");  
out.println("You have visited this page : " + ct + ((ct.intValue() ==1)? " time" : " times")  
);  
out.println("<br/>");  
out.println("Session ID : " + session.getId());  
out.println("<br/>");  
out.println("New Session : " + session.isNew() );  
out.println("<br/>");  
out.println("creation Time : " + new Date(session.getCreationTime()));  
out.println("<br/>");  
out.println("Last Access Time : " + new Date(session.getLastAccessedTime()));  
}  
}
```

You already have learnt about the running procedure of servlet and methods included in the above servlet program in Unit 2 Block 1 of this course. Compile the above servlet, put the class file in the classes folder, and create appropriate entry in the web.xml file under the WEB-INF folder in your 'webapps' folder. Now, you can start your web server and run the program from your browser.

When you access this program for the first time, the visiting counter will be one and the new Session will 'true'. When visiting counter increases in numbers the new session will be 'false'. This program will also display some details of the current session such as Session identifier, Creation Time and Last Access Time.

Output of the above servlet program is displayed in the following figure-1. Creating and running procedures of Servlet are given in the previous Unit of this block.

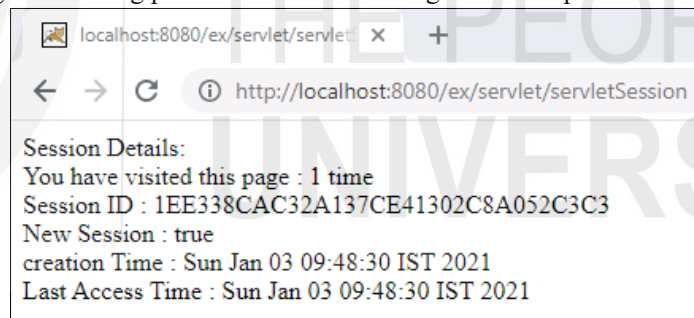


Figure 1: Output of Servlet by using HttpSession Object

In the similar way, you can create above servlet program and add the code in index.html file like the following:

```
<h1>Click here to go <a href="servletSession" >Session Servlet Program</a></h1>
```

When you will run your Project in NetBeans it is displayed as output like the figure-2:

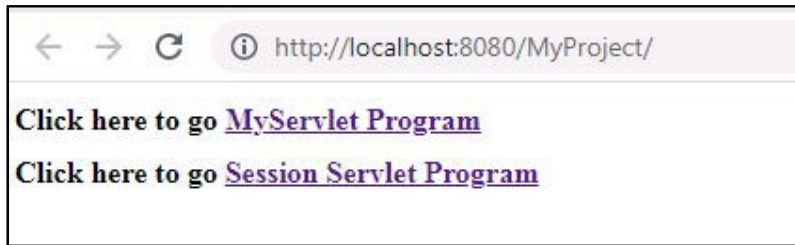


Figure 2: Output Screen of Welcome Index Page

When you will click on ‘Session Servlet Program’ link, then servlet will run and give output as Figure-3:

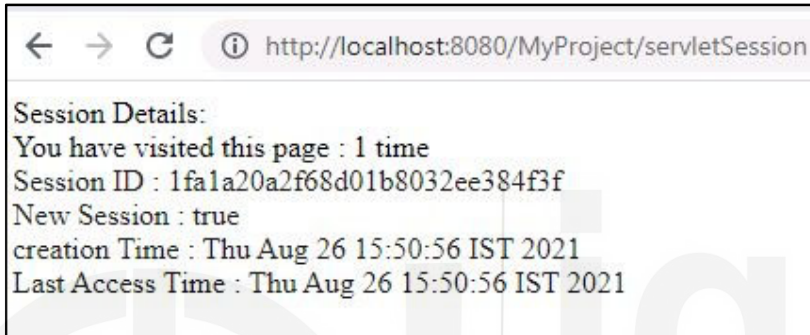


Figure 3: Output Screen of Session Servlet program

3.2.2 Cookies

In the last section, you have studied session management through HttpSession object in Servlet. Here you will learn about Cookies. This is another session management technique. Cookies are a small part of information like a name, a single value and optional attributes such as comment, path and domain qualifiers, a maximum age, and a version number. A servlet sends this cookie information to a web browser. It is saved by the browser and later sent back to the server. The browser probably supports 20 cookies for each Web server, 300 cookies total and may limit cookie size to 4 KB each. You can uniquely identify a client through cookie's value, so cookies are generally used for session management. If the client disables the cookies then it won't work and you cannot maintain a session with cookies.

There are two types of cookies such as Non-persistent cookie and Persistent cookie in servlets. The **Non-persistent cookie** is effective for a single session only and removed each time when the user closes the browser. On the contrary, **Persistent cookie** is effective for multiple sessions, and it is removed only when user logouts.

A cookie is indicated by the Cookie class in the javax.servlet.http package. You can create a cookie by calling Cookie class like the following:

```
Cookie c = new Cookie("userid", "socs");
```

You can send the cookie to the client browser by using addCookie() method of the HttpServletResponse interface like the following:

```
response.addCookie(c);
```

You can retrieve cookies from request using `getCookie()` of the `HttpServletRequest` interface like the following:

```
request.getCookie(c);
```

Example -2

The following example demonstrates to you how to create cookies and how these cookies are transferred to another servlet. This example contains one HTML form (`cookieForm.html`) and two servlet programs (`CreateCookieServlet.java` and `GetCookieServlet.java`).

The source codes of the programs are given below:

cookieForm.html

```
<form action="../servlet/CreateCookieServlet" method="post">
Name:<input type="text" name="uname"/><br/>
<input type="submit" value="Submit"/>
</form>
```

CreateCookieServlet.java

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

public class CreateCookieServlet extends HttpServlet
{
    public void doPost(HttpServletRequest request, HttpServletResponse response)
    {
        try
        {
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            String name=request.getParameter("uname");
            out.print("Welcome "+name);
            out.print(", Submit your data for GetCookieServlet");
            //create cookie object
            Cookie c=new Cookie("uname",name);
            //add cookie in the response
            response.addCookie(c);
            out.println("<form action='../servlet/GetCookieServlet' method='post'>");
            out.println("<input type='submit' value='Submit data'>");
            out.println("</form>");
        }
        catch(Exception e){System.out.println(e);}
    }
}
```

GetCookieServlet.java

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

public class GetCookieServlet extends HttpServlet
{
```

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
{
    try
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        Cookie c[] = request.getCookies();
        out.println("Welcome in SOCIS "+c[0].getValue());
    }
    catch(Exception e){System.out.println(e);}
}
```

Now, you can compile both the servlets program and place class files in the classes folder of the your web application. Also, make an entry in the deployment descriptor file and first run the HTML form program from your browser. When you submit your data by clicking submit button of the HTML program (see figure-4), the control is transferred to CreateCookieServlet program (see figure-5).

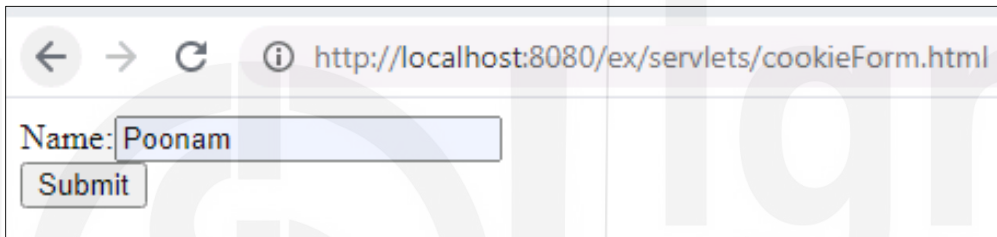


Figure 4: Cookie HTML Form

The process of creation and sending of cookies are included in CreateCookieServlet program. Now at this stage, form data is fetched and displayed on screen. When you click on 'Submit data' button, the control goes to the GetCookieServlet program (see Figure-5)

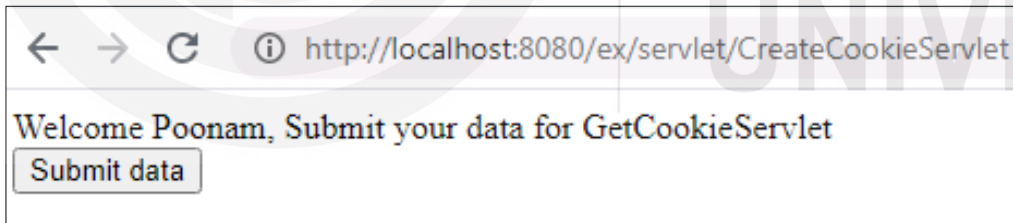


Figure 5: Intermediate output of CreateCookieServlet program

Now, the GetCookieServlet program fetches your data from CreateCookieServlet program and displays this as an output on your monitor screen like the following figure-6.

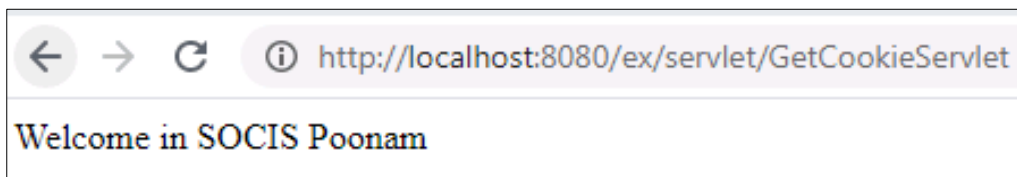


Figure 6: Final Output through Cookie techniques in session management

3.2.3 URL Writing

In the previous section, you have learnt two session management techniques i.e. HttpSession Object and Cookies. The third technique that you can use to maintain user sessions is by using URL writing. In this approach, the token (parameter) is embedded in each URL. When client submits request using such URLs, the token is retransmitted to the server. In each dynamically generated page, server embeds an extra query parameter or extra path information. If a browser does not support cookies then in that case URL rewriting technique is the best alternative for session management.

Using URL Writing technique, you can send parameter name/value pairs like the following:

`http://myserver.com?name=xyz&age=20`

When you click on URL, the parameter name/value pair will transfer to the servlet. The servlet will fetch this parameter by using `getParameter()` method of `HttpServletRequest` interface from the requested URL and use it for session management.

Example -3

The following example will show you how to work URL Writing technique with session management. This example comprises the 3 programs such as html form (`URLWritingForm.html`) and two Servlet program (`CreateURLServlet.java` and `FetchURLServlet.java`). The source codes of programs are listed below:

URLWritingForm.html

```
<html>
<head><title>URL Writing Session FORM</title></head>
<body>
<form action="../servlet/CreateURLServlet" method="post">
<fieldset>
<legend>Student Details</legend>
Student Name :<input type="text" name="uname"> <br>
Password: <input type="password" name="pwd"> <br>
<input type="submit" value="Submit">
</fieldset>
</form>
</body></html>
```

CreateURLServlet.java

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

public class CreateURLServlet extends HttpServlet
{
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String name = request.getParameter("uname");
        String password = request.getParameter("pwd");
```

```

if(password.equals("socis"))
{
    response.sendRedirect("FetchURLServlet?username="+ name);
}
else
{out.println("Password is incorrect");}
}
}

```

FetchURLServlet.java

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FetchURLServlet extends HttpServlet
{
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String name = request.getParameter("username");
        out.println("Welcome in SOCIS, "+name);
    }
}

```

Now, you can compile both the servlet programs and place class file in 'classes' folder. Also make entry in web.xml file and start web server. Finally, you can run URLWritingForm.html file from your browser. Here, you can submit your data such as student name, password and click on 'Submit' button (see Figure-7)

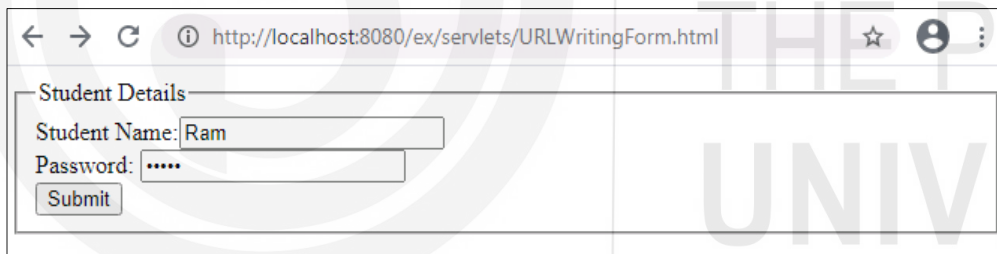


Figure 7: URL Writing HTML Form

After submitting your data, control goes to the CreateURLServlet program. This servlet program fetches your username and password from html form and compares your password statically with given 'socis' password in the program. If a match is done then control is transferred to the FetchURLServlet program else it displays an error message on the server.

Here, you can see parameters value in URL of the FetchURLServlet program in address bar like the following figure-8.



Figure 8: URL Parameter value in address bar

After fetching the parameter value, the FetchURLServlet program will display output

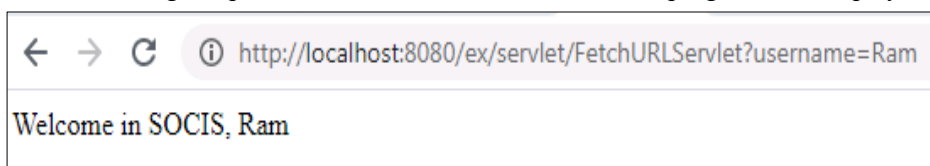


Figure 9: Example Output screen for URL Writing technique

as like following Figure-9.

3.2.4 Hidden Fields

Another technique for managing user sessions is by passing a token as the value for an HTML hidden field. You have seen web-form many times on the website. When client submits the form, additional fields will also be sent in the request in the form of hidden fields to keep track of the session. This method gives you the advantage to use this without depending on the browser whether the cookie is disabled or not. It has a disadvantage also as it is not secure because anyone can view the hidden form field value from the HTML file and use it to hack the session. Another disadvantage of using this method is that it needs extra form submission on each page.

You can create a unique hidden field in the HTML form to keep track of the session like the following format:

```
<input type="hidden" name="userid" value="socis">
```

You can get this hidden field in Java Servlet using the `getParameter()` method of `HttpServletRequest` interface.

```
String param1 = request.getParameter("userid");
```

☛ Check Your Progress 1

1. What is session management? What are the different techniques of session management in servlets? Why is session management needed for HTTP protocol? Explain how cookies can be used for session management.

2. Write a servlet program by using `HttpSession` Object of session management. Servlet program will display creation time when user will visit web page for the first time else it displays last access time. Also display session ID in both the conditions.

3. What is the difference between a session and cookie?

3.3 SERVLET COLLABORATION

In the previous section, you have gone through the session tracking techniques. Using these techniques, you can track a series of user requests. This section describes to you how servlets share information between two or more servlets.

Servlets running together under the same server, need to communicate with each other for exchanging data information. When two or more servlets can communicate or share common information, you can call it Servlet Communication or Servlet Collaboration.

Servlet collaboration means sharing information among the servlets. This tactic requires each servlet to know the other servlet with which it collaborates. Sometimes, a situation arises in program coding when you may require passing the request from one servlet to another. Also, you may want to include the content from HTML, JSP or Servlet into your servlet. Java provides Servlet-API to accomplish Servlet Collaboration and Servlet-API covers two interfaces namely: `javax.servlet.RequestDispatcher` and `javax.servlet.http.HttpServletResponse`. Both interfaces have various methods which are used for sharing information between servlets.

3.3.1 Using RequestDispatcher Interface

`RequestDispatcher` is an interface which is found in `javax.servlet` package. There are two methods defined in the `RequestDispatcher` interface: `forward()` and `include()` methods for dispatching requests to/from web resources. Both the methods accept a `javax.servlet.ServletRequest` and a `javax.servlet.ServletResponse` object as arguments. The client or browser is not involved in request dispatching.

3.3.1.1 The forward() Method

This method is used to forward a request from a servlet to another web resource like servlet, JSP file or HTML file on the server. When this method is called, control is transferred from the current to the next resource called.

The signature of this method is as follows:

```
public void forward(ServletRequest request, ServletResponse response)
throws ServletException, java.io.IOException
```

3.3.1.2 The include() method

This method is used to include the content of another servlet, JSP page, HTML file in the servlet response. After calling this method, the response of another resource is included in the called resource.

The signature of this method is as follows:

```
public void include(ServletRequest request, ServletResponse response)
throws ServletException, java.io.IOException
```

For using a servlet `forward()` or `include()` method, you first need to obtain a `RequestDispatcher` object. You can obtain a `RequestDispatcher` object like the following:

```
RequestDispatcher rd = request.getRequestDispatcher(String resource);
```

Example-4

The following example explains how to use both the forward() and include() method of RequestDispatcher interface to achieve Servlet Collaboration. This example comprises one HTML program (loginForm.html) and two servlets programs (RequestDispatcherServlet.java and WelcomeStudentServlet.java). The LoginForm contains two input fields i.e., name and password. RequestDispatcherServlet program received data from LoginForm and compares statically with given data in the servlet program. If a match is done, control is transferred to the WelcomeStudentServlet program, else control gets back to LoginForm again to re-enter the data.

The source codes of all three programs are listed below:

LoginForm.html

```
<html>
<head></head>
<body>
<form action="../servlet/RequestDispatcherServlet" method="post">
<fieldset>
<legend>Student Details</legend>
Student Name: <input type="text" name="user">
<br>
Password: <input type="password" name="pwd">
<br>
<input type="submit" value="Submit">
</fieldset>
</form>
</body>
</html>
```

RequestDispatcherServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class RequestDispatcherServlet extends HttpServlet
{
    public void doPost(HttpServletRequest req, HttpServletResponse res) throws
    ServletException, IOException
    {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        String name=req.getParameter("user");
        String pass=req.getParameter("pwd");
        if(name.equals("Ram") && pass.equals("socs"))
        {
            RequestDispatcher rd=req.getRequestDispatcher("WelcomeStudentServlet");
            rd.forward(req, res);
        }
        else
        {
            out.print("User name or password is incorrect!");
            RequestDispatcher rd=req.getRequestDispatcher("../servlets/LoginForm.html");
        }
    }
}
```

```

        rd.include(req, res);
    }
}

```

WelcomeStudentServlet.java

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class WelcomeStudentServlet extends HttpServlet
{
    public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String name=request.getParameter("user");
        out.print("Welcome in IGNOU, "+name+"!");
    }
}

```

You can compile both the servlet programs and make a suitable entry in web.xml file. Now you can start the web server and run HTML form through the browser.

When you run your LoginForm.html, it will display like the following Figure-10 with two input fields. When you enter student name and Password and click on submit button then action control goes to the RequestDispatcherServlet program.

Figure10: Login Form for RequestDispatcher servlet example

When you input the Student Name as Ram and Password as socis then it will welcome you (see figure-11); otherwise, it will ask to re-enter values (see figure-12)

Figure 11: Request Dispatcher example, Welcome Screen

When condition is false, it will display a message “User name or password is incorrect” and LoginForm will display again for re-enter input values (see figure-12).

Figure 12 Request Dispatcher example, Login Form Screen

3.3.2 Using HttpServletResponse Interface

The HttpServletResponse Interface captures the functionality of a response object that is returned to the client from an HTTP servlet. The interface HttpServletResponse offers many protocol-specific methods which are not available in ServletResponse interface. The interface HttpServletResponse extends the javax.servlet.ServletResponse interface. You have already explored two of the methods in HttpServletResponse at the time of servlet writing in this unit as well as Unit 2. These two methods setContentType() and getWriter() are used when sending output to the browser. The setContentType(java.lang.String type) method is used to set the type of the response data e.g. text, HTML etc. The getWriter() method returns the PrintWriter object for sending text data to the client.

```
res.setContentType("text/html");  
PrintWriter out = res.getWriter();
```

Another method of HttpServletResponse interface is addCookie() method which you have already studied in section 3.2.2 of this Unit. Another most important method in HttpServletResponse interface is sendRedirect() method. In the next section, you will learn about this method. You can use this method to redirect the user to another web page.

3.3.2.1 The sendRedirect() Method

The sendRedirect() method of HttpServletResponse interface is used to redirect response to another web resource such as HTML, servlet or jsp file. The sendRedirect() method works at the client side. This method always sends a new request. It can be used within and outside the application. For some applications, you want to send the request outside your web application and then it becomes most useful because it takes more time to fetch resources.

The syntax of sendRedirect() method is as under:

```
public void sendRedirect(String URL) throws IOException;
```

The following example illustrates the simplicity of sendRedirect() method:

```
response.sendRedirect("http://www.ignou.ac.in");
```

In the previous section, you have studied forward() method. Function of both the methods forward() method of RequestDispatcher interface and sendRedirect() of RequestDispatcher interface are almost similar but they differ also.

Difference between forward() and sendRedirect() method

Both methods bring the user to a new web resource. Both are similar, but knowing the fundamental difference between the two can help you write a servlet more efficiently. The difference between the two methods is as follows:

- ... The forward method can be used to redirect the request without any help of client browser, and control transfer remains within-applications resources only, whereas sendRedirect method can redirect the request to client browser and control transfer goes outside the current domain.
- ... In forward method, HttpServletRequest object and HttpServletResponse object are passed to the new resource, whereas in sendRedirect method, previous HttpServletRequest object is lost. For passing information between the original and new request, you can pass the information as a query string appended to the destination URL.

3.3.3 Using ServletContext Interface

In servlet programming, servlet context is the environment where the servlet runs. The ServletContext's object is created by the web container at the time of deploying the project. Using this you can use to access all the resources available within the application and to store attributes which other servlets with the same context can use. You can use one context per "web application" per Java Virtual Machine. The ServletContext object is used to get configuration information from deployment descriptor file (web.xml), which will be available to any servlet or JSPs that are component of the 'webapps'.

The ServletContext Interface defines various methods which are used by servlets to communicate with its servlet container. For example, dispatch requests or writes to a log file to get the MIME type of a file. The object of ServletContext can be added and retrieved from the context using the setAttribute and getAttribute methods.

How to get the Object of ServletContext

```
ServletContext app = getServletContext();  
//This is convenient way to get the ServletContext object  
OR  
ServletContext app = getServletConfig().getServletContext();  
//You can get the ServletContext object from ServletConfig object
```

When you have defined ServletContext object, you can set attributes of ServletContext object by using the setAttribute() method. From now, ServletContext object is available to all the servlets of the web application. By using the getAttribute() method, other servlets can get the attribute from the ServletContext object.

3.3.3.1 setAttribute() Method

This method stores an object and binds the object with the given attribute name in the application scope. It means that the said object is accessible from any servlet within the same web application. If attribute name already exists in the ServletContext, the old bound object will be replaced by the object passed to this method.

```
void setAttribute(java.lang.String name, java.lang.Object obj)
```

This method has two parameters:

String name: By providing the value of this argument, you can specify the name of attribute.

Object obj: By providing the value of this argument, you can specify the value of the named attribute.

3.3.3.2 getAttribute() Method

This method is used to get attribute with the given name from context. It returns the value of named attribute as an Object or NULL if there is no attribute by that name.

```
Object getAttribute(String name)
```

Example- 5

The following example explains to you how `setAttribute()` and `getAttribute()` method works with the `ServletContext`.

The example uses two servlets: `ServletDemo1` and `ServletDemo2` and one HTML Form (`Login.html`). This form comprises two input fields such as name and percentage of student. A student must have a percentage of more than 80. When the user submits the form, control will transfer to the `ServletDemo1` servlet.

The `ServletDemo1` servlet received two input values i.e. name and percentage by using `getParameter()` method from html form. This servlet uses `setAttribute()` method to bind name attribute and does this by first obtaining the `ServletContext` object. Here, this servlet checks percentage if percentage is more than 80, then the control will pass to the `ServletDemo2` servlet, otherwise control will transfer to Login form to fill data again.

In `ServletDemo2` servlet, after getting the `ServletContext` object, you can use `getAttribute()` method to get name attribute.

The code source of the three programs is listed below:

Login.html

```
<form action="..servlet/ServletDemo1" method="post">
<fieldset>
<legend>Student Details</legend>

Student Name: <input type="text" name="username">
<br>
Percentage: <input type="text" name="percent">
<br>

<input type="submit" value="Submit">
</fieldset>
</form>
```

ServletDemo1.java

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

public class ServletDemo1 extends GenericServlet
{
    public void service(ServletRequest request, ServletResponse response) throws
        ServletException, IOException
    {
        //Creating ServletContext object
        ServletContext sc = getServletContext();
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("Using ServletContext object to set and read attributes");
        out.println("<br/>");
        String uname = request.getParameter("username");
        int Per = Integer.parseInt(request.getParameter("percent"));
        //Setting name attribute to be shared between multiple servlets
        sc.setAttribute("Name", uname);
        if( Per > 80)
        {
```

```

        RequestDispatcher rd = sc.getRequestDispatcher("/servlet/ServletDemo2");
        rd.forward(request,response);
    }
    else
    {
        out.print("Data is incorrect!");
        out.println("<br/>");
        out.print("Fill your data again");
        RequestDispatcher rd = sc.getRequestDispatcher("/servlets/Login.html");
        rd.include(request,response);
    }
}
}
}

ServletDemo2.java
import java.io.*;
import javax.servlet.*;
public class ServletDemo2 extends GenericServlet
{
    public void service(ServletRequest request, ServletResponse response) throws
    ServletException, IOException
    {
        ServletContext sc = getServletContext();
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<b>" + "Congratulations! " + sc.getAttribute("Name") + "</b>");
    }
}
}

```

Now, you can compile both servlets and create appropriate entry in web.xml file. Finally, you can start server and run the Login form from your browser. The output of the html program is displayed like the following figure13.

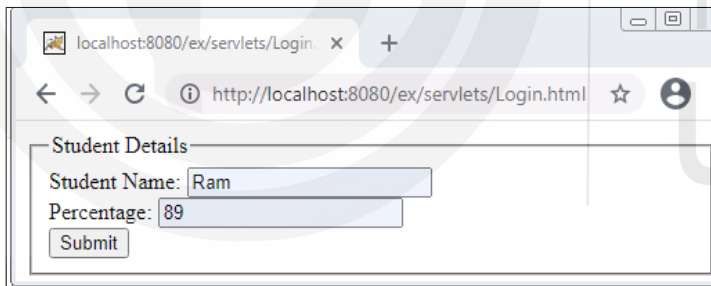


Figure 13: Login form

Here you can submit your data. The flow controls of servlets depends on your percentage. If you submitted more than 80, you will get congratulations (see figure-14), this procedure is defined in the ServletDemo2 servlet otherwise you will get back to login again to re-enter data (see figure- 15)

When condition will be true, you will get following figure-14:

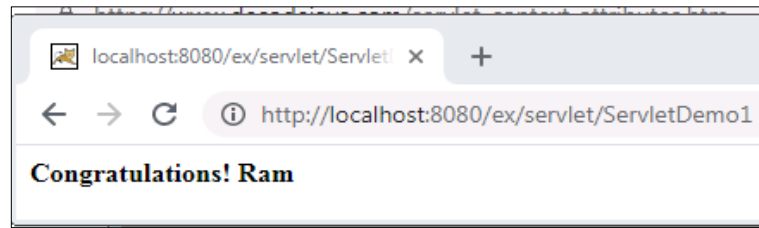


Figure 14: Output of setting and getting attribute example

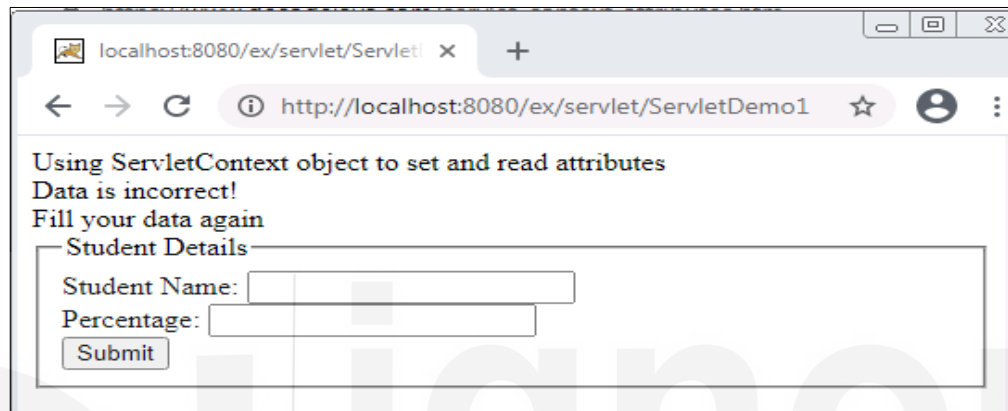


Figure 15: Output Screen for re-enter data

When the condition is false, you will get the following to re-enter data:

☛ Check Your Progress 2

1. What is servlet collaboration? How many ways can servlet communicate with each other? What is the purpose of RequestDispatcher Interface?

.....

.....

.....

2. What is the difference between forward() and sendRedirect() method?

.....

.....

.....

3. Write a servlet program for opening the IGNOU website by using sendRedirect() method.

.....

.....

.....

3.4 DATABASE CONNECTIVITY

In the previous section, you have studied more about the advanced features of servlets with examples, but all this work has been done statically without any database connectivity. This section will provide you in-depth knowledge of data access, specifically insertion and retrieval of data to/from a database.

Database access is one of the important features of Web application. Java has its own technology for database connectivity called JDBC (Java Database Connectivity). JDBC provides a standard library for accessing a wide range of relational databases like MS-Access, Oracle and Microsoft SQL Server. Using JDBC API, you can access a wide variety of different SQL databases. The core functionality of JDBC is found in java.sql package.

Consider a table named as Student created in Microsoft SQL Server database with Roll No, Name and Program name. This table is used in both the following sections. This section defines example(s) for storing/retrieving data into/from a Microsoft SQL Server using type-4 JDBC driver. You can run these programs on any web server such as Tomcat. For running these programs, you need a JDBC driver in .jar form and placing them in lib directory under the web server.

3.4.1 Insert data in Database

For inserting data into a database, you can use the following example.

Example - 6

The following example will demonstrate to you how to store data in a database using servlet. For this, create a HTML Student Form and one Servlet for storing data into a database.

The student form contains student's details like student enrolment no., student name and Programme name. You will have to create a student table with related fields of student form. When you submit these details, access 'DataStoreServlet' to store data into the database.

The DataStoreServlet program is written similar to the above servlet programs with database query and 'try' and 'catch' clause of standard Java mechanism.

The source codes of both the files are listed below:

StudentForm.html

```
<html>
<body>
<form action="..servlet/DataStoreServlet" method="post">
<h3>Indira Gandhi Open University </h3>

<fieldset>
<legend><b>Student Details </b></legend>

Enrolment No: <input name="sid" type="text"/><br>
Student Name : <input name="sname" type="text" value="" ><br>
Programme : <input type="text" name="prg" value=""> <br>
<input type="submit" value="Submit" />

</fieldset>
</form>
```

```
</body>  
</html>
```

In the following servlet program, the first step is to get the data from 'StudentForm.html' program using request.getParameter() method. After connecting to database, an insert query is executed using executeUpdate() method.

DataStoreServlet.java

```
import java.io.*;  
import java.sql.*;  
import java.util.*;  
import javax.servlet.*;  
import javax.servlet.http.*;  
public class DataStoreServlet extends HttpServlet  
{  
    public void doPost(HttpServletRequest req, HttpServletResponse res) throws  
        ServletException, IOException  
    {  
        res.setContentType("text/html");  
        PrintWriter out = res.getWriter();  
        String rollNo = req.getParameter("sid");  
        String StuName = req.getParameter("sname");  
        String prgName = req.getParameter("prg");  
        //create connection object  
        Connection con = null;  
        // create statement object  
        Statement stmt = null;  
        // connection string using Type-4 Microsoft SQL Server driver  
        // you can also change the next line with your own environment  
        String url=  
            "jdbc:microsoft:sqlserver://127.0.0.1:1433;user=sa;password=poonam;DatabaseName=SOCIS";  
        try  
        {  
            // load JDBC type-4 SQL Server driver  
            Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");  
            con = DriverManager.getConnection(url);  
            if (con != null)  
            {  
                stmt = con.createStatement();  
                //insert query  
                String rsq1="insert into student  
                    values("+rollNo+", '"+StuName+"', '"+prgName+"')";  
                //execute query  
                stmt.executeUpdate(rsq1);  
                out.println("Your data is successfully stored in database");  
            }  
            if(con == null)  
            { con.close(); // release connection }  
        }  
        // end of try clause catch(SQLException se)  
        { out.print("SQL:"+se.getMessage());}  
        catch(Exception e)  
        { out.print("Exception:"+e.getMessage());}  
    }  
}
```

As earlier, you can compile the above servlet and make related entry in deployment descriptor file (web.xml) then start your web server and run your StudentForm.html file from your browser. When you run your form, it will display output like the following figure-16:

Figure 16: Student data Form for storing data into database

In the above screen, when you will enter values then the following screen (figure-17) will show you a message for successful data storage.

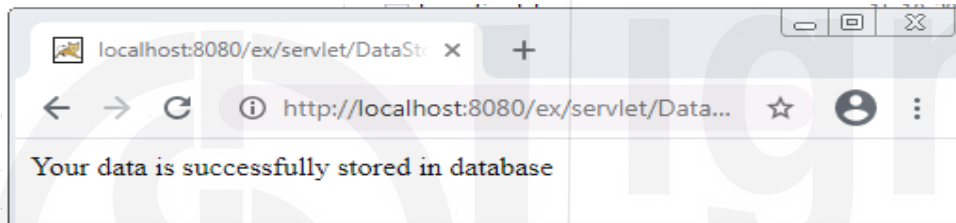


Figure 17: Data stored in persistent storage

Example-7

The following example gives you an illustration about how to retrieve data from the database. After execution of the above DataStoreServlet program, you have stored sufficient data into the database. Now, you will execute the following code for retrieving the data from the database. In this program, one additional ResultSet object is used for retrieving data from the select query. The data is retrieved from ResultSet object by using getXXX() method such as getInt() and getString(). Note that if the column name is an integer type, then you should use getInt() method instead of getString() method. The following RetDataServlet program is listed for retrieving data from database.

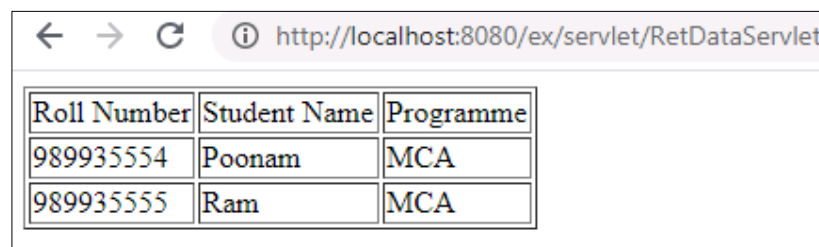
```
import java.io.*;
import java.util.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class RetDataServlet extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws
    ServletException, IOException
    {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        Connection con = null;           //create connection object
        Statement stmt = null;           // create statement object
        ResultSet rs = null;              // create ResultSet object
    }
}
```

```
// connection string using Type-4 Microsoft SQL Server driver
// you can also change the next line with your own environment
String url=
"jdbc:microsoft:sqlserver://127.0.0.1:1433;user=sa;password=poonam;DatabaseName=SOCIS";
try
{
    // load JDBC type-4 SQL Server driver
    Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
    con = DriverManager.getConnection(url);
    if (con != null)
    {
        stmt = con.createStatement();    // select SQL statement
        String rsq1 ="select * from Student";
        rs = stmt.executeQuery(rsq1);    //execute query
        out.println("<table border=1><tr><td>Roll Number</td><td>Student Name</td><td>Programme</td></tr>");
        while( rs.next() )
        {
            out.println("<tr><td>" + rs.getInt("RollNo") + "</td>");
            out.println("<td>" + rs.getString("Student_Name") + "</td>");
            out.println("<td>" + rs.getString("Programme") + "</td></tr>");
        }
        out.println("</table>");
    }
    if(con == null) {con.close();}
}
catch(SQLException se){ out.println("SQL:"+se.getMessage());}
catch(Exception e){ out.println("Exception:"+e.getMessage());}
}
```

Now, you can compile the above servlet and make entry in deployment descriptor file (web.xml), start your web server and run your servlet program from your browser. After running the RetDataServlet program, the data will be displayed on your output screen like following figure-18:

If you want to build your project using Oracle or other database as back end then you can change only port no. and relevant JDBC driver name in above servlet source code.



The screenshot shows a web browser window with the address bar displaying 'http://localhost:8080/ex/servlet/RetDataServlet'. The main content area contains a table with three columns: 'Roll Number', 'Student Name', and 'Programme'. The table has two data rows. The first row contains the values '989935554', 'Poonam', and 'MCA'. The second row contains the values '989935555', 'Ram', and 'MCA'.

Roll Number	Student Name	Programme
989935554	Poonam	MCA
989935555	Ram	MCA

Figure 18: Display data from database

3.5 SUMMARY

In this unit, you have learned how to use session management using four techniques: HttpSession object, Cookie, URL Writing, and Hidden form field. You have been shown a number of examples to demonstrate the use of session management in the

servlets. In addition, this Unit introduced you to Servlet collaboration which is all about sharing information among the servlets. The Servlet-API provides the RequestDispatcher interface to achieve Servlet Collaboration. This interface is useful for forwarding the request to another web resource and including the resource in the current servlet. Apart from these, this unit also discussed database access using Java Database Connectivity (JDBC) technology.

3.6 SOLUTIONS/ANSWERS TO CHECK YOUR PROGRESS

☛ Check Your Progress 1

- 1) Session management allows servlets to maintain information about a series of requests from the same user. Session in Java Servlet is managed through four techniques such as HttpSession API, Cookies, URL rewriting and Hidden Field.

HTTP is a stateless protocol. It means a HTTP server does not remember any state information. So one needs to maintain state using session management techniques.

For this cookie can be used in the following way for managing sessions in java servlets:

... You can create a cookie by calling Cookie class like the following:

```
Cookie c = new Cookie("userid", "socis");
```

... You can send the cookie to the client browser by using addCookie() method of the HttpServletResponse interface like the following:

```
response.addCookie(c);
```

... You can retrieve cookies from the request by using getCookie() of the HttpServletRequest interface like the following:

```
request.getCookie(c);
```

- 2) The source code of servlet program by using HttpSession object is as under:

```
import java.io.*;
import java.util.Date;
import javax.servlet.*;
import javax.servlet.http.*;
public class sessionServlet extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws
    ServletException, IOException
    {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        HttpSession session = req.getSession(true);
        if ( session.isNew())
        {
            out.println("New Session : " + session.isNew() );
            out.println("<br/>");
            out.println("Session ID : " + session.getId());
            out.println("<br/>");
        }
    }
}
```

```
        out.println("creation Time : " + new Date(session.getCreationTime()));  
    }  
    else  
    {  
        out.println("New Session : " + session.isNew() );  
        out.println("<br/>");  
        out.println("Session ID : " + session.getId());  
        out.println("<br/>");  
        out.println("Last Access Time : " + new  
            Date(session.getLastAccessedTime()));  
    }  
}  
}
```

- 3) Both the Session and Cookie are used to store information. The session is stored in server-side machine, whereas Cookies are stored on the client-side machine. Session should work regardless of the settings on the client browser. Session data will be available to all pages on the site during the particular session of visit. Session can store objects, and cookies can store only strings. Cookie expires depending on the lifetime you set for it whereas a Session ends when a user closes the browser or after leaving the site.

☛ Check Your Progress 2

- 1) Servlet collaboration means sharing information among the servlets. This tactic requires each servlet needs to know the other servlet with which it collaborates. The Servlet API provides the following interfaces and their methods which are responsible for sharing information between servlets:

- ... RequestDispatcher Interface : include() and forward() method;
- ... HttpServletResponse Interface : sendRedirect() method;
- ... ServletContext Interface : setAttribute() and getAttribute() method;

RequestDispatcher interface is found in javax.servlet package. There are two methods defined in the RequestDispatcher interface: forward () and include() methods for dispatching requests to/from web resources.

- 2) Both the methods bring the user to a new web resource. Both are similar but there is a fundamental difference between the two. The difference between two methods is as follows:

- ... The forward method can be used to redirect the request without the help of the client browser and control transfer remains within-applications resources only. The sendRedirect method can redirect the request to the client browser and control transfer goes outside the current domain.

- ... In forward method, HttpServletRequest object and HttpServletResponse object are passed to the new resource whereas in sendRedirect method, previous HttpServletRequest object is lost. For passing information between the original and new request, you can pass the information as a query string appended to the destination URL.

- 3) The source code of servlet by using sendRedirect() method is given below:

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

```
public class sendRedirectServlet extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws
        ServletException, IOException
    {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        res.sendRedirect("http://www.ignou.ac.in/");
    }
}
```

3.7 REFERENCES/FURTHER READING

- ...Jason Hunter, and William Crawford “Java Servlet Programming”, O'Reilly Media, Inc., 2001.
- ...Kathy Sierra, Bryan Basham, and Bert Bates, “Head First Servlets and JSP”, O'Reilly Media, Inc., 2008.
- ...Budi Kurniawan, “Java for the Web with Servlets, JSP, and EJB: A Developer's Guide to J2EE Solutions: A Developer's Guide to Scalable Solutions”, *Techmedia*, 2002.
- ...Pratik Patel and Karl Moss, “Java Database Programming with JDBC: Discover the Essentials for Developing Databases for Internet or Intranet Applications”, Coriolis Group, 1996.
- ... <https://www.w3adda.com/servlet-tutorial>
- ... <https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/ServletContext.html>
- ... <https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/http/HttpSession.html>
- ... list of drivers: <http://www.devx.com/tips/Tip/28818>

UNIT 4 JAVA SERVER PAGES

Structure

- 4.0 Introduction
- 4.1 Objectives
- 4.2 JSP Overview
- 4.3 JSP Life Cycle
- 4.4 JSP API
- 4.5 Components of JSP
 - 4.5.1 Directives
 - 4.5.2 Scripting Elements
 - 4.5.3 Action Elements
- 4.6 JSP Implicit Objects
 - 4.6.1 out Object
 - 4.6.2 request Object
 - 4.6.3 response Object
 - 4.6.4 session Object
 - 4.6.5 application Object
 - 4.6.6 page Object
 - 4.6.7 pageContext Object
 - 4.6.8 config Object
 - 4.6.9 exception Object
- 4.7 An Introduction to JSP Standard Tag Library (JSTL)
- 4.8 Exception handling in JSP
 - 4.8.1 Using `errorPage` and `isErrorPage` attribute of `page` directive
 - 4.8.2 Using `try` and `catch` block in `Scriptlets`
 - 4.8.3 Using `<error-page>` element in `Deployment Descriptor`
- 4.9 Database Connectivity
 - 4.9.1 Insert data in Database using JSP
 - 4.9.2 Retrieve Data from Database using JSP
- 4.10 Summary
- 4.11 Solutions/Answers to Check Your Progress
- 4.12 References/Further Readings

4.0 INTRODUCTION

Servlet is a server-side programming language. In unit 2 and 3 of this course, you have already gone through the concept of Servlets in detail. In this Unit, you will learn another server-side language i.e. Java Server Pages (JSP). Both the JSP and Servlets are correlated. JSP uses a component-based approach that allows web developers to easily combine static HTML for look-and-feel with Java components for dynamic features.

JSP is a specification of Sun Microsystems which first appeared in 1999. The current specification of JSP is 2.3. The updates between JSP 2.2 and 2.3 are relatively minor. JSP is a technology for developing dynamic web pages. It follows the characteristics of Java 'write once and run anywhere'. A JSP document contains HTML tags as well as JSP elements. JSP page comprises different components such as directives, scripting elements, standard actions and implicit objects.

This unit covers how to create a JSP page. It also provides a basic understanding of JSP components that make an entire JSP page, Java Bean, Custom tag, and life cycle of Java Server Pages. This unit will also introduce you to the exception handling in

JSP as well as database connectivity which are a necessary feature for any web application.

4.1 OBJECTIVES

After going through this unit, you should be able to:

- ... use JSP to create dynamic web pages,
- ... use directives, scripting and action elements in a JSP page,
- ... write java code within the JSP page,
- ... create a custom tag,
- ... forward request from JSP pages to other resources,
- ... write program for handling exceptions at page and application level in JSP, and
- ... develop database applications using JDBC and JSP.

4.2 JSP OVERVIEW

Java Server Pages (JSP) is powerful web technology. Using this technology, you can create dynamic content based web pages. Dynamic web pages are different from static web pages in which web server creates a web page when a web client or user requests it. For example, your online results on IGNOU website, the page for every student instead IGNOU web server dynamically creates a page depending on your enrolment number.

As you know very well, an HTML page is a static page; it contains static content that always remains the same. When you insert some dynamic content or java code inside the HTML page, it becomes JSP page. A JSP page encompasses a very simple structure that makes it easy for developers to write JSP code as well as for servlet engine to translate the page into a corresponding servlet. You can change content dynamically with the help of Java Bean and JSP elements. The JSP elements are the basic building blocks of the page. JSP page consists of directives, scripting elements and action elements. Each of these elements can use either JSP syntax or be expressed in XML syntax, but you cannot use both syntaxes simultaneously. For this problem, you can use the include mechanism to insert files that may use a different syntax.

Both JSP and Servlets are very closely related to each other. You have already studied the Servlets in Unit 2 and Unit 3 of this course. The Java Server Pages have more advantages over the servlets, which are as follows:

- ... JSP allows programmers to insert the Java code directly into the HTML file, making the JSP document and development process easier, while Servlets use plenty of 'println' statements for printing an HTML document that is usually very difficult to use. The JSP has no such tedious task to perform.
- ... JSP supports element based dynamic content that allows programmers to develop custom tags libraries to satisfy application needs.
- ... In a JSP page, visual content and logic are separated, which is not possible in the servlets.
- ... JSP pages can be used in conjunction with servlets that handle business logic.
- ... Major advantage of JSP page is that in case JSP page is modified, there is no need to recompile and redeploy the project. But the Servlet code needs to be

updated and recompiled if we have to change the look and feel of the application.

To create the first JSP page, write some HTML code and java code, similar to code given below, and save it with .jsp extension. This is a simple example of JSP where you are using the scriptlets tag to put Java code in the JSP page. You will learn scriptlets tag in section 4.5 of this unit.

```
<html><body>
```

```
<% out.println("Welcome to the IGNOU family"); %>
```

```
</body></html>
```

For running the JSP program, you need Apache's Tomcat. You have already installed this software for servlets using the procedure defined in section 2.7 of Block 1 Unit 2 of this course. You should only create a 'JSP' folder under the 'ex' folder, and this folder exists in the 'webapps' directory in Apache Tomcat. For this, you can see the following figure-1. Place your JSP document in the 'JSP' folder to run the JSP page.

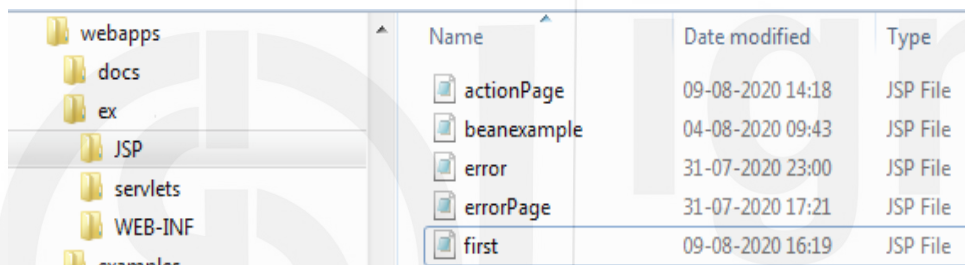


Figure 1: Directory structure for your program in Tomcat

Start your Tomcat Server, open a browser, and type the following URL to execute your first JSP program.

<http://localhost:8080/ex/JSP/first.jsp>

The output of the program is displayed in figure-2:

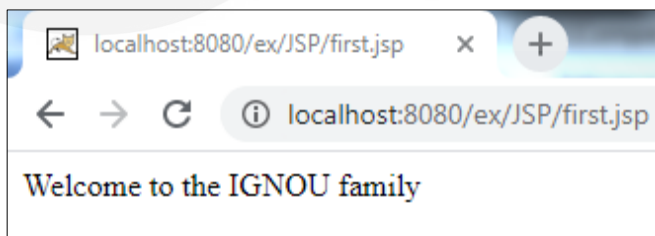


Figure 2: First Program in JSP

You have seen the above program, which looks like an HTML document with some added tags containing java code that allows the server to insert dynamic content in the page. When a client sends a request for a JSP page, the server executes a JSP page elements merges with static contents and sends the dynamically page back to the client browser.

Steps for Creating and running procedure of JSP in NetBeans

Creating and running procedure of servlet in NetBeans is defined in Unit-2 of this block. Installation process is given in Lab manual. This Unit is devoted to JSP. The creating and running procedure of servlet as well as JSP are similar. To create a JSP, open 'Source Packages' in NetBeans, right click on default package -> New -> JSP.

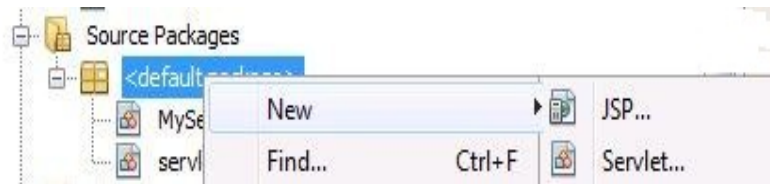


Figure 3: Creation window of JSP

Give a name and select location to JSP file as you have given a name MyJSP.jsp under MyProject in Figure-4. Also select JSP File (Standard Syntax) option.

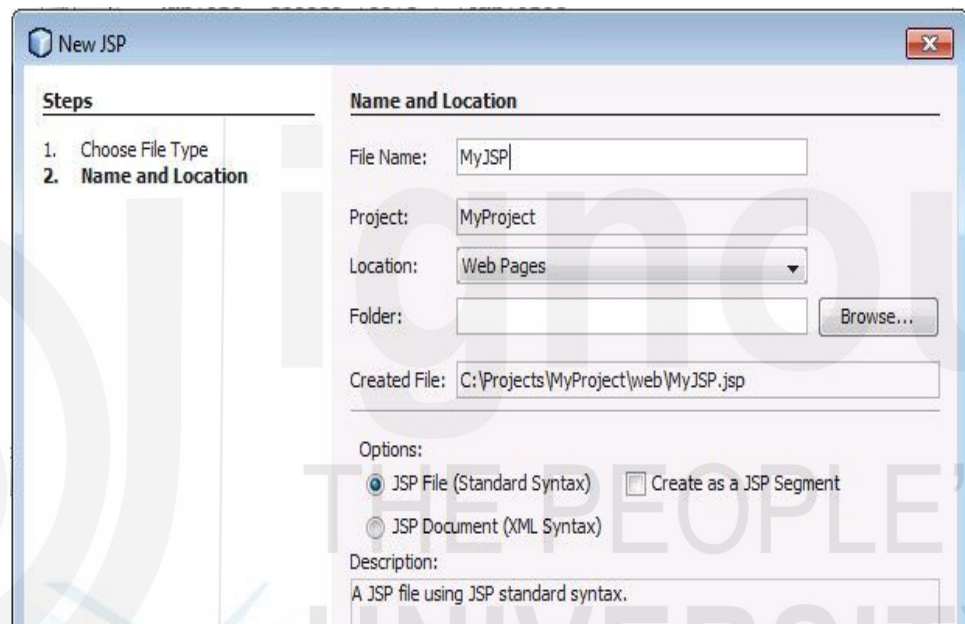


Figure 4: Name and Location window for JSP file

Finally select Finish button from button Panel.



Figure 5: Button Panel

Figure 6:Source code of MyJSP file

Now JSP file is ready, you can change the code as per your requirement .

Once this is completed. For running the particular JSP file, right-click on the file and select 'Run File' (Shift+F6). The output of the MyJSP file is displayed as shown in figure-7:

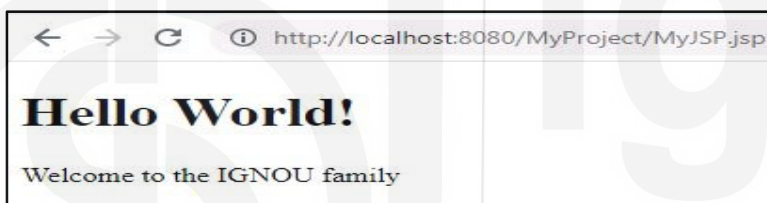


Figure 7: Output Screen of MyJSP file

JSP uses server-side scripting that is actually translated into servlets and compiled before they are run. You can study more about the life cycle of JSP in subsequent section 4.3.

4.3 JSP LIFE CYCLE

In this section, you will go through the life cycle of JSP and see how a JSP page is displayed. When the JSP is first accessed, it is translated into a corresponding servlet (i.e. java class) and compiled, then JSP page services request as a servlet. The

translation of JSP page is done by the JSP engine of the underlying web container/servlet container (e.g. Tomcat). Figure-8 shows how a JSP page is processed.

The life cycle of JSP page is controlled by three methods i.e. `jspInit()`, `_jspService()` and `jspDestroy()`.

jspInit() - The `jspInit()` method is called only once during life cycle of a JSP. Similarly, servlet also has an `init()` method whose purpose is the same as that of `jspInit()`. `jspInit()` method is used to initialize objects and variables that are used throughout the life cycle of JSP. This method is defined in `JspPage` interface. This method is invoked when the JSP page is initialized. It has no parameters, returns no value and thrown no exceptions.

The signature of the method is as follows:

```
public void jspInit() { // Initialization code }
```

_jspService() – `_jspService()` is the method that is called every time the JSP page is requested to serve a request. This method is defined in the `javax.servlet.jsp.HttpJspPage` interface. This method takes `HttpServletRequest` and `HttpServletResponse` objects as an argument. The `_jspService()` method corresponds to the body of the JSP page. It is defined automatically by the processor and should never be redefined by you. It returns no value. The underscore (‘_’) signifies that you cannot override this method. The signature of the method is as follows:

```
public void _jspService  
(  
    javax.servlet.http.HttpServletRequest request,  
    javax.servlet.http.HttpServletResponse response)  
    throws javax.servlet.ServletException, java.io.IOException  
{  
    // services handling code  
}
```

jspDestroy()- The `jspDestroy()` is invoked only once when the JSP page is about to be terminated. It is synonymous to the `destroy()` method of a servlet. You have to override `jspDestroy()` if you need to perform any cleanup, such as releasing database connections or closing open files. The signature of the method is as follows:

```
public void jspDestroy(){ // cleanup code }
```

Following figure-9 shows the life cycle of JSP page:

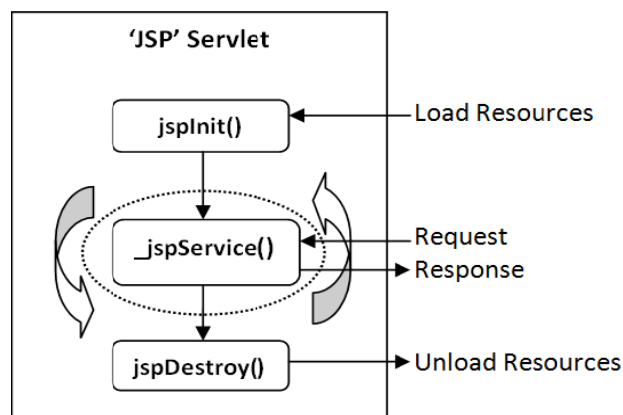


Figure 9: life cycle of JSP Page

4.4 JSP API

The JSP technology is based on JSP API (Application Programming Interface) that contains two packages such as **javax.servlet.jsp** and **javax.servlet.jsp.tagext**. In addition to these two packages, JSP also needs two packages of servlets such as **javax.servlet** and **javax.servlet.http**. Both packages provide interfaces and classes for writing servlets. You have already studied this in Block 1 Unit 2 of this course.

Package javax.servlet.jsp

The javax.servlet.jsp package has two interfaces such as `HttpJspPage` and `JspPage` and four classes such as `JspEngineInfo`, `JspFactory`, `JspWriter` and `PageContext`. These are as follows:

Interface	Description
<code>JspPage</code>	The <code>JspPage</code> is the interface that all JSP servlet classes must implement. The <code>JspPage</code> interface has two methods, <code>JspInit</code> and <code>JspDestroy</code> . The <code>JspInit</code> method is similar to the 'init' method in the <code>javax.servlet.Servlet</code> interface. The <code>JspDestroy</code> method is analogous with destroy method of the <code>javax.servlet.Servlet</code> interface. You have studied both the methods in section 4.3 of this Unit. JSP developers rarely make full use of these two methods.
<code>HttpJspPage</code>	The <code>HttpJspPage</code> interface directly extends the <code>JspPage</code> interface. Each JSP page implements this interface. It supports only one method <code>_jspService()</code> . The <code>_jspService()</code> method is already described in life cycle of JSP in section 4.3 of this Unit.
Class	Description
<code>JspEngineInfo</code>	The <code>JspEngineInfo</code> is an abstract class that provides information of the current JSP engine.
<code>JspFactory</code>	The <code>JspFactory</code> is an abstract class that defines a number of factory methods available to a JSP page at runtime for the purposes of creating instances of various interfaces and classes used to support the JSP implementation.
<code>JspWriter</code>	The <code>JspWriter</code> class is derived from the <code>java.io.Writer</code> class. This is the normal mechanism for JSP pages to produce output to the response. No. of methods are defined in this class such as <code>print()</code> and <code>println()</code> .
<code>PageContext</code>	The <code>PageContext</code> class is an abstract class. It extends <code>JspContext</code> to provide useful context information when JSP technology is used in a servlet environment. This class provides methods that are used to create other objects. For example, <code>getServletConfig()</code> returns a <code>ServletConfig</code> object and <code>getServletContext()</code> returns a <code>ServletContext</code> object.

Apart from these interfaces and classes, the two exception classes: `JspException` and `JspError` are also defined in `javax.servlet.jsp` package of JSP API. `JspException` is the base class for all JSP exceptions. For details, please refer to the following link:
https://docs.oracle.com/cd/E17802_01/products/products/jsp/2.1/docs/jsp-2_1-pfd2/javax/servlet/jsp/package-summary.html

Package javax.servlet.jsp.tagext

The `javax.servlet.jsp.tagext` encompasses classes and interfaces for the definition of JSP Tag Libraries. The use of this package is defined in the creation of custom tag under the section 4.5.1(taglib directives) of this unit. The interfaces in the

javax.servlet.jsp.tagext package are BodyTag, IterationTag, Tag, and TryCatchFinally. The classes are defined in this package such as BodyContent, BodyTagSupport, PageData, TagAttributeInfo, TagData, TagExtraInfo, TagInfo, TagLibraryInfo, TagLibraryValidator, TagSupport, TagvariableInfo and VariableInfo. You know more about the package details; please refer to the following link:
<https://docs.oracle.com/javaee/5/api/javax/servlet/jsp/tagext/package-summary.html>

4.5 COMPONENTS OF JSP

In this section, we are going to learn the components of JSP that make up a JSP page. There are three types of JSP components such as **Directives**, **Scripting** and **Action**. All the components are to be discussed in detail in the following sections:

4.5.1 Directives

Directives have great use as it guides the JSP container for translating and compilation of JSP page. Directives control the processing of an entire JSP page. It appears at the top of the page. Using directives, the container translates a JSP page into the corresponding servlet. They do not directly produce any output. A directive component comprises one or more attributes name/value pairs. Directives are defined by using `<%@` and `%>` tags. The syntax of Directive is as under:

`<%@ directive attribute = "value" %>`

There are three types of directives used in JSP documents: **page**, **include** and **taglib**. Each one of these directives and their attributes is defined in the following sections:

The *page* Directive

The page directive defines attributes that apply to an entire JSP page, such as the size of the allocated buffer, imported packages and classes/interfaces, and the name of the page that should be used to report run time errors. The page Directive is a JSP element that provides global information about an entire JSP page. This information will directly affect the compilation of the JSP document. The syntax of JSP page directive is as under:

`<%@ page attribute = "value" %>`

The following are the different properties that can be defined by using page directive:

Attribute	Description
Language= "scripting language"	This attribute outlines the language that will be used to compile the JSP document. By default, it is java language.
import= "import list"	This attribute defines the names of packages.
session= "true false"	It specifies whether or not the JSP document participates in HTTP session. The default is <i>true</i> .
extends= "classname"	This attribute states the name of the parent class that will be inherited by the generated servlet. It is rarely used.
buffer= "none size in kb"	The default value is 8kb. It specifies the size of 'out' buffer.
autoFlush= "true false"	The default is <i>true</i> . It means that the <i>out</i> buffer will be automatically flushed when full. If it is <i>false</i> , it will be raised as an exception when the buffer is full.
Info= "text"	If this attribute is used, the servlets will override the <code>getServletInfo()</code> method.
errorPage= "error_page_url"	This attribute defines the relative URL to JSP document that will handle the exception.
isErrorPage= "true false"	This attribute indicates that the current page can act as an error page for another JSP page. The default value is <i>false</i> .
isThreadSafe= "true false"	The default value is <i>true</i> . It indicates that the page can service more than a request at a time. When it is <i>false</i> , the <i>SingleThreadModel</i> is used.

An example of the use of page directive is as follows:

```
<%@ page import="java.io.*, java.util.Date" buffer="16k" autoFlush="false" %>
<%@ page errorPage="error.jsp" %>
```

You can specify one or more page directives in your JSP document. In the above example, first page directive statement instructs the web container to import java.io package and java.util.Date class. It also instructs the web container to set buffer size to 16k and turn off autoflushing. The second page directive statement defines a name of error page which is used for handling errors.

The include Directive

JSP include directive is used to include files such as HTML, JSP into the current JSP document at the translation time. It means that it enables you to import the content of another static file into a current JSP page. The advantage of using an include directive is to take advantage of code re-usability. This directive can appear anywhere in a JSP document. The syntax of include directive is as follows:

```
<%@ include file = "relative URL" %>
```

Note: Relative URL only specifies the filename or resource name, while absolute URL specifies the protocol, host, path, and name of the resource name.

Consider the following example for include Directive. In this example, there are two files: an HTML file, and another is a JSP file. You can create both files but run only JSP file and the result is displayed like the figure-5.

Source code for header.html:

```
<html> <body>
<h2>Indira Gandhi National Open University</h2>
<h4>The text is from Header File</h4>
</body></html>
```

Source code for index.jsp:

```
<html><body>
<h3>Example of include directive</h3>
<%@ include file= "header.html" %>
</body></html>
```

The output of the above program is shown in the following figure-10:

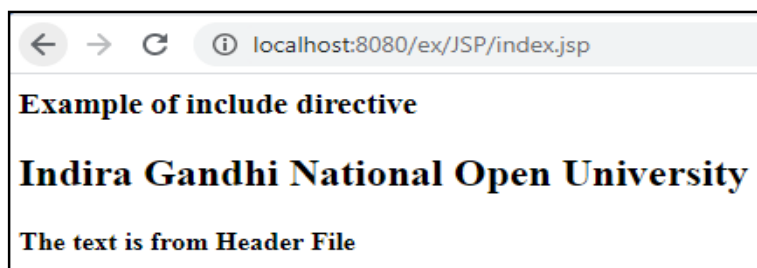


Figure 10: Example of Include Directive

The taglib Directive

This directive allows users to use Custom tags in JSP. A custom tag is user-defined tag. The custom tag eliminates the need for scriptlet tag. In this section, you will learn

about the creation of custom tag libraries in Java Server Pages. It is a reusable code in a JSP page and tag library is a collection of custom tags. The taglib directive has the following syntax:

```
<%@ taglib uri="tagLibraryURI" prefix="tagPrefix" %>
```

The uri (Uniform Resource Identifier) attribute defines an absolute or relative uri of tag library descriptor (TLD) file, and the prefix attribute defines the string that will identify a custom tag instance.

There are four components which you need to use customs tags in a JSP page:

- 1) The **Java file** that does the processing or Tag handler class
- 2) **Tag Library Descriptor (TLD)** file that points from JSP to Java file
- 3) **JSP** file that contains the tag being used
- 4) **Deployment descriptor** file (web.xml) that states the server where to find the TLD file

Tag Handler Class

It is a java class that defines the behaviour of the tags. This class must implement the javax.servlet.jsp.tagext package.

Tag Library Descriptor (TLD) file

A tag library descriptor file is an xml document. It defines a tag library and its tags. This file must be saved with a .tld extension to the file name. It contains <taglib> root element and <tlib-version>, <jsp-version>, <short-name>, <tag> which all are sub elements of the taglib element. The <tag> is the most important element in the TLD file because it specifies the tag's name and class name. You can define more than one <tag> element in the same TLD file.

Deployment Descriptor file (web.xml)

The deployment descriptor is an xml file that specifies the configuration details of the tag. The most essential element for custom tag in web.xml file is <taglib-location>. Using the web.xml, the JSP container can find the name and location of the TLD file.

JSP file

Once you have created tag handler java class, tag descriptor file, and defined configuration details in deployment descriptor file, you have to write a JSP file that uses the custom tag.

The following are five easy steps for building a JSP application that uses custom tags:

Step-1: Write, compile and deploy a java class called MyCustomTag.java, which is given in the following source program. The class file must be placed in the directory, say 'customTag' under the WEB-INF/classes directory like the figure 11. The directory 'customTag' is user defined directory.

Source Code for simple CustomTag

```
package customTag;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class MyCustomTag extends TagSupport
{
    public int doEndTag() throws JspException
```

```

{
    JspWriter out = pageContext.getOut();
    try
    { out.println("Hello!! Creating a First Custom tag"); }
    catch(Exception e) {}
    return super.doEndTag();
}
//doEndTag()
}
//main class

```

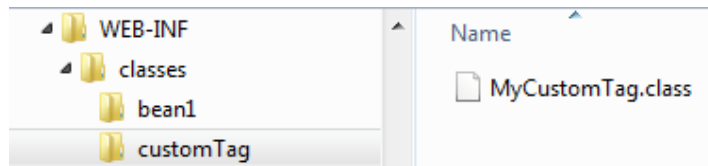


Figure 11: classes directory structure for Custom Tag in Tomcat

Step-2: Create a TLD file named taglib.tld as shown in following source program and save it in WEB-INF directory.

```

<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library
1.2//EN" "http://java.sun.com/j2ee/dtd/web-jsptaglibrary_1_2.dtd">
<taglib>
    <tlib-version>1.0</tlib-version>
    <jsp-version></jsp-version>
    <short-name></short-name>
    <tag>
        <name>myTag</name>
        <tagclass>customTag.MyCustomTag</tagclass>
    </tag>
</taglib>

```

Step-3: Create a JSP file named SimpleCustomTag.jsp from the following source code and save it in the directory say 'JSP' like the figure 12 under your 'webapps' folder.

```

<html><body>
<%@ taglib uri="/myTLD" prefix="easy" %>
<easy:myTag />
</body></html>

```

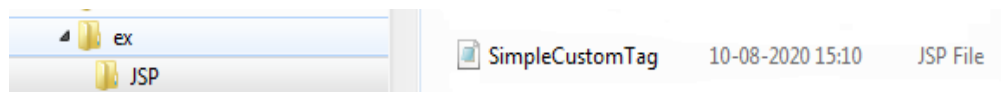


Figure 12: File structure for Custom tag in JSP

Step-4: Edit your deployment descriptor (web.xml) file. To use custom tags, you must specify a <taglib> element in your deployment descriptor file. It is a large file; you can place the following code under the <web-app> root element.

```
<jsp-config>
<taglib> <taglib-uri> /myTLD </taglib-uri>
        <taglib-location>/WEB-INF/taglib.tld </taglib-location>
</taglib>
</jsp-config>
```

Step-5: Start server let us say Tomcat (if you are using this). Open web browser and run the JSP page using the URL as <http://localhost:8080/ex/JSP/SimpleCustomTag.jsp>. The following screen comes as an output for a simple custom tag.

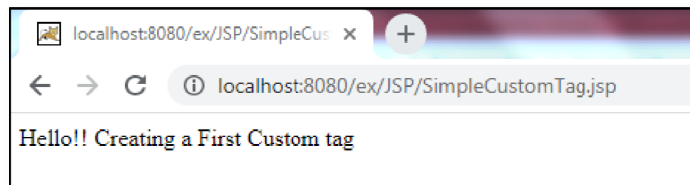


Figure 13: A simple JSP Custom Tag page

When a user requests to a JSP page, the JSP container first checks the taglib directive and gets the taglib 'uri' attribute. On the basis of this, JSP container looks into the web.xml file to find taglib location and continues the processing and gets the name of tag. After getting the name and location of TLD file, it obtains the java class. Now, the JSP container loads the class for custom tag and continues the processing.

4.5.2 Scripting elements

JSP scripting elements is a mechanism for embedding java code fragments directly into a HTML page. There are three scripting language elements such as **declarations**, **expressions**, **scriptlets** involved in JSP scripting. Each of these scripting elements has an appropriate location in the generated servlet. In this section, you will look at these elements and how together they will result in a complete servlet.

JSP Scripting Element	Description
Declarations	To declare the variables and methods for the page.
Expressions	To return a value from the scripting code as a <i>String</i> to the page.
Scriptlets	To execute java source code

Declarations

Declarations are used to declare the variables and methods that you can use in the JSP document. The declaration part is initialized when the JSP document is initialized. After the declarations have been initialized, they are available to other expressions, declarations and scriptlets. A declaration starts with `<%!` and ends with `%>`. It has the following syntax:

```
<%! declaration %>
```

Following is an example for JSP Declarations:

```
<%! x=0; %> // x is an integer type variable
```

```
<%! int x, y, z; %> // x, y, z is an integer type variable
```

```
<%! String name = new String("JSP"); %> // string type variable declaration
```

```
<%! public String getName() { return name; } %> //method declaration
```

Expressions

The code placed within JSP expression element is written to the output stream of the response. You need not write `out.print()` to write data. Expressions are evaluated at request time, and the result is inserted where the expression appears in the JSP file. The syntax of the expression is as follows:

```
<%= expression %>
```

Note: *Do not end your statement with a semicolon in case of expression tag.*

Here is an example of expressions:

```
<%! int x, y, z; %> // x, y, z is an integer type variable
<%
    x=5, y=5;
    z = x/y;
%>
<%=z %>
```

For example, to show the today date and time:

```
<%= new java.util.Date() %>
```

Scriptlets

A scriptlet element is used to execute java source code in JSP. It is executed at the request time and makes use of declarations, expressions and Java Beans. You can write scriptlets anywhere in a page. It contains a valid java statement within the `<%` and `%>` tag and gets inserted into the `_jspService()` method of generated servlet. The syntax of the scriptlet is as follows:

```
<% // java code %>
```

Note: *Semicolon at the end of scriptlet is necessary.*

The following example will demonstrates how to add two numbers and print this result on output screen using Scriptlets.

```
<html><body>
<% int num1 = 5, num2 = 15, sum;
    sum= num1 + num2;
    out.println("sum= "+sum);
%>
<footer><hr>&copy;2020 SOCIS, IGNOU</footer>
</body></html>
```

Output of the above program is as follows:

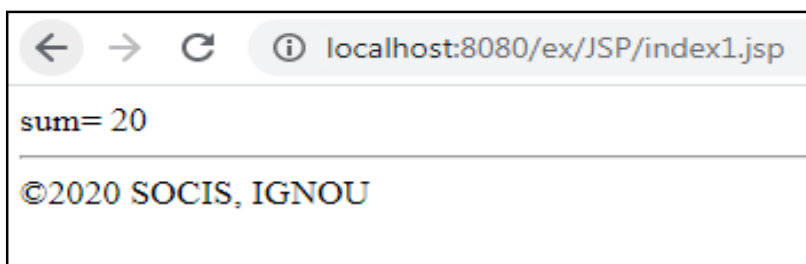


Figure 14: Output screen for Scriptlets example

Check Your Progress 1

1. Write the basic steps for processing JSP request.

2. Write a JSP program for Fibonacci Series.

3. Write a JSP program using Scriptlets that add numbers from 1 to 10 and prints this result.

4.5.3 Action Elements

In the previous section, you have already learnt about two JSP components i.e. Directives and Scripting elements. Now, you will learn about the third JSP component i.e. Action elements or Standard actions tags that can be embedded in a JSP program. JSP provides many standard action tags that you can use for specific tasks such as add java bean objects in JSP document, forward the request to another resource or include other resources in current resource, etc. At the compile time, they are replaced by java code. The list of some important standard JSP action elements are given below:

JSP Action	Description
<code><jsp:useBean></code>	To get the java bean object from given scope or to create a new object of java bean.
<code><jsp:getProperty></code>	To get the property of a java bean, used with <code>jsp:useBean</code> action.
<code><jsp:setProperty></code>	To set the property of a java bean object, used with <code>jsp:useBean</code> action.
<code><jsp:param></code>	To define the name/value pair of parameter to be passed to <code><jsp:useBean></code> , <code><jsp:forward></code> , <code><jsp:include></code> and <code><jsp:plugin></code> tag.
<code><jsp:forward></code>	To forward the request to another resource.
<code><jsp:include></code>	To include a resource at runtime, the resource may be HTML, JSP or any other file
<code><jsp:plugin></code>	To embed another component such as applet in JSP

Before you start learning about how you can add Java Bean in the JSP page, you must look at what a bean is. A Java Bean is a java class that maintains some data called properties and has a no-arguments constructor. It is reusable component that work on any Java Virtual Machine. To create Java Bean, you must create a java class that implements `java.io.Serializable` interface and uses public `get/set` methods to show its properties.

`<jsp:useBean>` Action

This action is used to include an instance of java bean within JSP document. The syntax of first JSP standard action is as follows:

```

<jsp:useBean id="name"
             scope="page | request | session | application"
             class="className" >
             body
</jsp:useBean>

```

In the above syntax, 'id' represents the name of the object and this attribute is necessary. The '**class**' attribute represents the fully qualified class name of the object. The class name is case sensitive. The '**scope**' attribute represents the life of the object. It may be page, request, session or application. It means how long the object is available for a single user or all application users. JSP provides different scope for sharing data between web pages. These are:

- ... **Page** - 'page' scope means the JSP object can be accessed only from within the same page where it is created. By default, it is page. JSP implicit objects out, exception, response, pageContext, config and page have 'page' scope.
- ... **Request** – Beans with request scope are accessible only within pages that are processing the same request for that the object was created in. Objects that have request scope are most often used when you need to share information between resources that is pertinent for the current request only. Only Implicit object request has the 'request' scope.
- ... **Session** – This object is accessible from any JSP within the same session. Implicit object session has the 'session' scope.
- ... **Application** - This object is accessible from any JSP within the same web application. Implicit object application has the 'application' scope.

<jsp:setProperty> Action

The second JSP standard action is used to set the Java Beans property value. The syntax for this action is as follows:

```

<jsp:setProperty name="beanName" property="propertyName" />

```

In the above syntax, '**name**' attribute represents the name of the bean instance defined by <jsp:useBean> action and '**property**' attribute represents the bean property for which you want to set a value.

<jsp:getProperty> Action

This final bean handling action is <jsp:getProperty> which gets the named property and outputs its value for inclusion in the page as a *String*. The syntax for this action is as follows:

```

<jsp:getProperty name="name" property="propertyName" />

```

In the above syntax, '**name**' attribute represents the name of the bean instance defined by <jsp:useBean> action and '**property**' attribute represents the bean property for which you want to get a value.

The following is a simple Java Bean example that stores University details:

Let's create a Java Bean named university.java and place class file under WEB-INF/classes/bean1 directory.

Source code for university.java

```
package bean1;
public class Iguniversity
{
    public IGuniversity()
    {
    }
    private String uname;           //define university name
    private int year;               //define year variable
    private String school;
    /* ----- getMethod and setmethod for university name--- */
    public String getUname()
    {
        return uname;
    }
    public void setUname(String uname)
    {
        this.uname = uname;
    }

    /* ----- getMethod and setmethod for Year--- */
    public int getYear()
    {
        return year;
    }
    public void setYear(int year)
    {
        this.year= year;
    }

    /* ----- getMethod and setmethod for School--- */
    public String getSchool()
    {
        return school;
    }
    public void setSchool(String school)
    {
        this.school = school;
    }
}
```

You can access the properties of the Java Bean from the following JSP file named as universitydetails.jsp.

```
<jsp:useBean id="universityinfo" class="bean1.IGuniversity" scope="session" />
<jsp:setProperty property="*" name="universityinfo"/>
```

You have entered following university details:


```
<jsp:getProperty property="uname" name="universityinfo"/><br>
<jsp:getProperty property="year" name="universityinfo" /><br>
<jsp:getProperty property="school" name="universityinfo"/><br>
```

Now, you can create a JSP file named 'beanexample.jsp' for putting values to the bean.


```

<html><head><title>useBean, getProperty and setProperty example </title></head>
<form action="universitydetails.jsp" method="post">
University Name: <input type="text" name="uname"><br>
Year <input type="text" name="year"><br>
School: <input type="text" name="school"><br>
<input type="submit" value="Go"> </form> </html>

```

Now, you can run 'beanexample.jsp' file in your browser using the URL as <http://localhost:8080/ex/JSP/beanexample.jsp> and get the following screen for the above programs.

Figure 15: Input Screen for Bean Example

The following output screen is as follows for the University details program:

Figure 16: Output Screen for a Java Bean Example

<jsp:param>

The <jsp:param> action is used within the body of <jsp:include>, <jsp:forward> and <jsp:plugin> tag to supply extra name/value pair of parameter. Following is the syntax of the <jsp:param> action :

```
<jsp:param name="paramName" value="paramValue" />
```

In the above syntax, name attribute defines the name of parameter being referenced and value attribute represents the value of named parameter.

<jsp:include>

The <jsp:include> action is used to include additional static or dynamic resources in current JSP document at run-time. The static resource is inserted at the translation time and the dynamic resource at the request time. In the previous section, you have already studied the static include i.e. include directive.

The syntax for this action is as follows:

```
<jsp:include page="relativeURL" flush="true" />  
OR  
<jsp:include page="relativeURL" flush="true" >  
    <jsp:param ...../>  
</jsp:include>
```

In the above syntax, **page** attribute defines the path of the resource to be included and **flush** attribute indicates whether the buffer is flushed. It is an optional parameter. The first syntax is used when `<jsp:include>` does not have a parameter name/value pair. If you want to pass the parameter to the included resource, use the second syntax. The `<jsp:param>` tag allows parameter to be appended to the original request.

<jsp:forward>

The `<jsp:forward>` action is used to terminate the current execution of JSP page and forwarding the client request to another URL, whether it can be an HTML file, JSP or Servlet within the same application at the run time. The `<jsp:param>` tag is used with `<jsp:forward>` action element to passing parameter. The syntax for `<jsp:forward>` action tag is as follows:

```
<jsp: forward page="relative_url" flush="true" />  
OR  
<jsp: forward page="relative_url" flush="true" >  
    <jsp:param ...../>  
</jsp: forward >
```

<jsp:plugin>

The `<jsp:plugin>` action element enables the JSP to include a bean or a applet in the client page. The `<jsp:param>` is also used with `<jsp:plugin>` action element to send parameters to Applet or Bean. It has the following syntax:

```
<jsp:plugin type="pluginType" code="classFile"codebase="relativeURLpath">  
<jsp:param ...../>  
</jsp: plugin >
```

In the above syntax, **type** attribute indicates the type of plugin to include in JSP page, **code** attribute represents the name of class file and **codebase** attribute is the path where the code attribute can be found.

4.6 JSP IMPLICIT OBJECT

JSP Implicit objects or predefined variables are the java objects that you can use directly without being declared first in scriptlets of JSP document. JSP implicit objects are created during the translation phase of JSP to the servlet. These variables are automatically available for the JSP page developer to use. These nine implicit objects are summarized in the following table:

Table: JSP Implicit Objects

S.No.	Implicit Object	Type	Scope
1	out	javax.servlet.jsp.JspWriter	Page
2	request	javax.servlet.HttpServletRequest	Request
3	response	javax.servlet.HttpServletResponse	Page
4	session	javax.servlet.http.HttpSession	Session
5	application	javax.servlet.ServletContext	Application
6	page	javax.servlet.jsp.HttpJspPage	Page
7	pageContext	javax.servlet.jsp.pageContext	Page
8	config	javax.servlet.http.servletConfig	Page
9	exception	java.lang.throwable	Page

4.6.1 out Object

This implicit object is a simple and frequently used in scriptlet of JSP page. You call either its `print()` or `println()` method to send output to the client browser. For example,

```
<% out.println(" JSP is an easy language"); %>
```

In the above code, you use the implicit out object that represents `javax.servlet.jsp.JspWriter` class.

4.6.2 request Object

The request object is an instance of `HttpServletRequest` interface. This object is used to retrieve the values that the client browser passed to the server during HTTP request, such as cookies, session, headers information, or parameters associated with the request. The most common use of request object is to access queries by calling the `getParameter()` and `getQueryString()` method.

For example: `<% String name=request.getQueryString(); %>`

You can find another example of `request.getParameter()` method in Database Connectivity section of this Unit.

4.6.3 response Object

The response object is an instance of `HttpServletResponse` interface. It is used to add or manipulate response such as redirect response to another resource, send error etc. Using response object, a reply is sent back to the client browser.

```
<% response.sendRedirect("http://www.ignou.ac.in"); %>
```

For example, when the above code is executed, the `sendRedirect()` method of the `javax.servlet.HttpServletResponse` to redirect the user to IGNOU website.

4.6.4 session Object

The session object is represented by the `javax.servlet.http.HttpSession` interface. This object is most often used when there is a need to share information between requests for a single user. It is used to store the user's data to make it available on other JSP pages until the user session is active. This object is used to track the user information in the same session.

4.6.5 application Object

The application object is an instance of `javax.servlet.ServletContext`. The `ServletContext` object is created only once by the web container when application or project is deployed on the server. This object is used to get initialization parameter from configuration file. There is one `ServletContext` object for each web application per Java Virtual Machine.

4.6.6 page Object

The page object is an instance of `javax.servlet.jsp.HttpJspPage` interface. The page object is just as it sounds, a reference to the current instance of the JSP page. The page object is a synonym for 'this' reference and is not useful for programming language.

4.6.7 pageContext Object

The pageContext object is an instance of javax.servlet.jsp.PageContext. It is used to represent the entire JSP page. The pageContext object is used to set, get and remove attribute of the JSP page.

It provides a single point of access to many page attributes such as directives information, buffering information, errorPageURL and page scope. It is also provide a convenient place to store shared data. This object stores references to the request and response objects for each request. The PageContext class defines several fields, including PAGE_SCOPE, REQUEST_SCOPE, SESSION_SCOPE, and APPLICATION_SCOPE, for identifying the four scopes of attributes.

4.6.8 config Object

The config object is an instance of javax.servlet.http.servletConfig. The container for each jsp page creates it. This object is used to get the configuration information of the particular JSP page by using the getServletConfig method. This can be useful in retrieving standard global information such as the paths or file locations.

4.6.9 exception Object

This implicit object only exists in a defined errorPage. It contains reference to uncaught exception that caused the error page to be invoked. This object is assigned to the Throwable class, which is the super class of all errors and exceptions in the java language. You can find a complete description of errorPage mechanism including the use of this exception object in section 4.8 (exception handling in JSP) of this unit.

☛ Check Your Progress 2

1. What is the purpose of action elements in JSP?

2. Write and explain the various bean scopes.

3. Explain the application implicit object with example.

4.7 AN INTRODUCTION TO JSP STANDARD TAG LIBRARY

In the section 4.5.2 (Scripting elements), you have learnt about the scriptlets in JSP pages to generate dynamic content. Sometimes it creates readability issues and made it difficult to maintain the JSP pages. To overcome this problem, Custom tags have been introduced. Although custom tags are better choice than scriptlets but web

developers need to be consumed a lot of time in coding and testing such tags. A new feature named JSP Standard Tag Library (JSTL) has been introduced for web developers to develop web pages in a better manner.

The Java Server Pages Standard Tag Library is a collection of useful JSP tags to simplify the JSP development. The JSTL tags can be classified as per their features such as Core Tags, Formatting tags, SQL tags, XML tags and JSTL Functions. For JSTL 1.1 Tag details, refer to the link: <https://docs.oracle.com/javaee/5/jstl/1.1/docs/tlddocs/>. To begin working with JSP JSTL tags, you need to first install the JSTL library. If you are using the Apache Tomcat, then use this link:

<https://tomcat.apache.org/taglibs/index.html> to download the library.

There are different types of JSTL tags to include in JSP document as per your requirement.

JSTL Core Tags

The core group of tags are the most commonly used JSTL tags. To use these tags in JSP, you should have used the following taglib:

```
<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>
```

JSTL Formatting Tags

The Formatting tags provide support for message formatting, number and date formatting. To use these tags in JSP, you should have used the following taglib:

```
<%@ taglib prefix = "fmt" uri = "http://java.sun.com/jsp/jstl/fmt" %>
```

JSTL XML tags

These tags provide support for creating and manipulating the XML documents. The JSTL XML tag library has custom tags used for interacting with the XML data. This includes parsing the XML, transforming the XML data and the flow control based on the XPath expressions. To use these tags in JSP, you should have used the following taglib:

```
<%@ taglib prefix = "x" uri = "http://java.sun.com/jsp/jstl/xml" %>
```

JSTL Functions

JSTL includes a number of standard functions; most of them are common string manipulation functions. To use these tags in JSP, you should have used the following taglib:

```
<%@ taglib prefix = "fn" uri = "http://java.sun.com/jsp/jstl/functions" %>
```

JSTL SQL Tags

The JSTL SQL tag library provides tags to access the database such as Oracle, MySQL, or Microsoft SQL Server. To use these tags in JSP, you should be used the following taglib:

```
<%@ taglib prefix = "sql" uri = "http://java.sun.com/jsp/jstl/sql" %>
```

There are two important JSTL SQL tag such as `<sql:setDataSource>` and `<sql:query>` which are described below:

JSTL <sql:setDataSource> tag

This tag is used to create a DataSource configuration that may be stored into a variable that can be used as an input to another JSTL database-action. You can use this tag like the following:

```
<sql:setDataSource var="name_of_variable" driver="name_of_driver"
    url="jdbc_url" user=" " password=" " />
```

In the <sql:setDataSource> tag, **var** attribute is optional and it specifies the variable's name, which stores a value of the specified data source. The **dataSource** attribute is optional that specifies the data source. The **driver** attribute defines the JDBC driver class name and **url** attribute specifies the JDBC URL associated with the Database. Both the **user** and **password** are the optional attributes that specify the JDBC database user and password name.

JSTL <sql:query> tag

This tag is used for executing the SQL query written into its body or through the sql attribute. For example:

```
<sql:query dataSource="${db}" var="name_of_variable">
    SELECT ProgCode, Pname from programme;
</sql:query>
```

In the <sql:query> tag, **var** is a required attribute that specifies the name of the variable to stores a value of the query result, **sql** attribute specifies the statement of SQL query to execute and **dataSource** is an optional attribute specifies the data source.

4.8 EXCEPTION HANDLING IN JSP

When you are writing a JSP code, there is a chance that you might make coding errors that can occur in any part of the code. In Java Server Pages (JSP), there are two types of errors. The first type of error comes at translation or compilation time when JSP page is translated from JSP source file to Java Servlet class file. The second type of JSP error which is called a request time error occurs during run time or request time. These run time errors result in the form of exception. Exception Handling is the process to handle runtime errors.

There are three ways to perform exception handling in JSP. They are:

- ... **errorPage** and **isErrorPage** attributes of page directive
- ... **<error-page>** tag in Deployment Descriptor file
- ... **try** and **catch** block in Scriptlets

4.8.1 Using page directive attributes

You have already learned about the page directive in section 4.5.1 of this Unit. The page directive defines attributes that apply to an entire JSP page. The two attributes of the page directive, such as 'errorPage' and 'isErrorPage' and one implicit object 'exception' are useful in JSP exception handling.

The errorPage attribute

The `errorPage` attribute of page directive is used to specify the name of error page that handles the exception. The error page contains the exception handling code description for a particular page. The syntax of this attribute is as follows:

```
<%@ page errorPage="relative URL" %>
```

The `isErrorPage` attribute

The `isErrorPage` attribute of page directive indicates whether or not the JSP page is an errorPage. The default value of this attribute is false.

```
<%@ page isErrorPage="true" %>
```

For exception handling, you need to create a simple JSP page and write only one additional **errorPage** attribute of page directive at the top of the JSP page to make an error page and set its value equal to the location of your JSP error page. In a similar way, you can create another JSP page where an exception may occur. Here, you can put first line of `isErrorPage` attribute of page directive at the top of the JSP page and set its value equal to true and write the second line of code designates where the thrown exception is being used.

For example, you have to create an 'error.jsp' named file which contains source code for handling exception and the other page named 'processPage.jsp' file, where exception may occur and define the `errorPage` attribute of page directive. The third file `inputPage.jsp` is used for input values. This example is for dividing two values and displaying the result.

Source code for `inputPage.jsp`

```
<html>
<head> <title>InputPage.jsp</title> </head>
<body>
<form action="processPage.jsp">
Enter Number 1<input type="text" name="num1"><br>
Enter Number 2 <input type="text" name="num2"><br>
<input type="submit" value="submit">
</form>
</body>
</html>
```

Source code for `processPage.jsp`

```
<%@ page errorPage="error.jsp" %>
<html>
<head> <title> procesPage.jsp </title> </head>
<body>
<%
String n1 = request.getParameter("num1");
String n2 = request.getParameter("num2");

int Var1 = Integer.parseInt(n1);
int Var2 = Integer.parseInt(n2);
int Var3 = Var1 / Var2;

out.println("First number = "+ Var1);
out.println("Second number = "+ Var2);
out.println("Division of two numbers are "+ Var3);

%>
</body></html>
```

Source code for error.jsp.

```
<%@ page isErrorPage="true" %>
<html>
<head>
<title>error.jsp</title>
</head>
<body>
Your page generate an Exception : <br>
<%= exception.getMessage() %>
</body>
</html>
```

Output of the above programs

When you run the above program code, it shows the following two screens one after another. In the first screen (figure-17), there are two input box where you will input two integer values and click on submit button. In the second screen (figure-18), the browser displays the result after the division of two numbers.

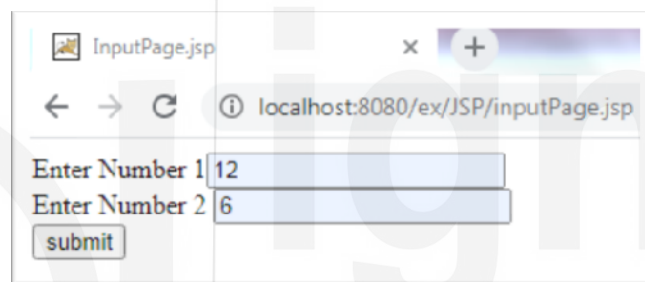


Figure 17: Input Screen for entering values

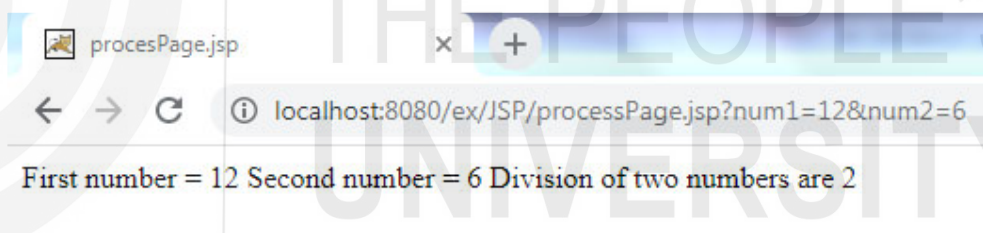


Figure 18: Output Screen for division of two numbers

Now, you will run the above same program again and input any integer value in first text box and zero in second text box and click on submit button. The program will generate an Arithmetic Exception: division by zero. When you will input any float value in any text box, then it will also generate exception.

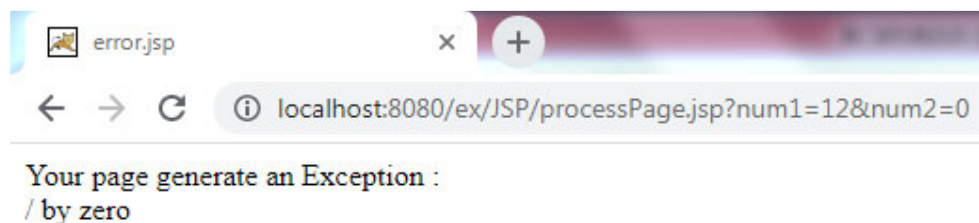


Figure 19: Exception handling Screen

4.8.2 Using try and catch block in Scriptlets

If you want to handle exception within the current JSP page, you can also use the standard java exception handling mechanism in the Scriptlets component of the JSP page. For this, you can use try and catch clause in your source code. In try block, you can write the normal code that might throw an exception. The catch block is used to handle the exception. Both the try and catch clauses are written inside the scriptlets. Unlike page directive, there is no need to write an error page for this mechanism.

You can use the try and catch block like the following manner:

```
<%
    try { // code that thrown an exception }
    catch (exception e) { // exception handler for exception occur in try block}
```

```
%>
```

The following program code is for handling exception using try and catch clause of Java. In this program, normal coding is defined in try block and exception handling code is in catch block. In try block, there are three integer variable x, y and z defined. The variable x contains a value zero and y contains value 10. When the program tries to divide the value of y by x then exception occurs. The program control flow is preceded and displays only those states which are included in catch block.

Source Code for Exception handling through try and catch clause

```
<html>
<head><title> Exception handling through try and catch clause </title></head>
<body>
<%
    try
    {
        int x = 10;
        x = x/ 0;
        out.println("Output = " + x);
    }
    catch (Exception e)
    { out.println("Error caught by Catch block : " + e.getMessage()); }
%>
</body></html>
```

Output of the program:

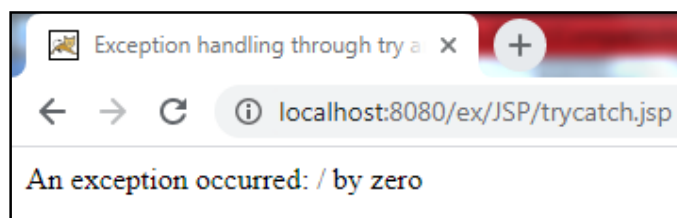


Figure 20: Output Screen for Exception handling through try and catch clause

4.8.3 Using <error-page> tag in Deployment Descriptor file

In the above section, you have learned about the page level exception handling in JSP through page directive and try-catch mechanism in scriptlets. If you want to describe the exception handling in the JSP page at the application level, you can use the <error-

page> tag element in the Deployment Descriptor file. The deployment descriptor is a file named web.xml. It resides in the web applications under the WEB-INF/ directory.

This approach is better as you don't need to use the page directive to declare the error page in each JSP file. Making a single entry in the web.xml file serves the purpose. You can specify either an <exception-type> element with the expected exception's class name, such as java.lang.ArithmeticException or <error-code> element with an HTTP error code value such as 500 with <location> element. The <location> element states JSP container for path of the resource to show when the error occurs. If you want to handle all the exceptions you need to specify the java.lang.Exception in the exception type element.

To include a generic error page for all exceptions at application level in the following way in web.xml file:

```
<error-page>
  <exception-type>Exception</exception-type>
  <location>/error.jsp</location>
</error-page>
```

If you want to handle exception using any specific error status code such as File not found error 404, Server error 500, then you can specify <error-code> element instead of <exception-type>. This is the best way to declare error page by using error-code element in web.xml file. It is used as under:

```
<error-page>
  <error-code>500</error-code>
  <location>/jsp/error.jsp</location>
</error-page>
```

Here is the example for exception handling at the application level using <error-page> element. The example code is very similar to the above-used example in page directive option. For this example, four files are needed to run the program:

- ... **inputPage.jsp** file for input values (The code is same as described in the example of page directive section 4.8.1)
- ... **processPage.jsp** for dividing two numbers and displaying the result. (You can use the same code which is described in the example of page directive section 4.8.1 except the line which contains errorPage attribute of page directive).
- ... **error.jsp** file for displaying the exception (which is also the same as the example defined in page directive option).
- ... **web.xml** file for specifying the <error-page> element.

Source code for web.xml: You can include the following code in your web application under the WEB-INF/web.xml file. In this example, error.jsp file is placed under the jsp/error.jsp folder in web application.

```
<web-app>
.....
.....
<error-page>
  <error-code>500</error-code>
  <location>/jsp/error.jsp</location>
</error-page>
.....
</web-app>
```

Now, you can run the program as similar to the example of page directive section 4.8.1. In this way, you can handle the exceptions at page as well as application level.

Note: The page directive declaration overrides any matching error page configurations in the deployment descriptor. Suppose the JSP page throws `java.lang.ArithmeticException` and deployment descriptor has an exception-type attribute for that exception, the web container invokes the page directive instead of any URI specified in deployment descriptor (web.xml) configuration.

Check Your Progress 3

1. What is the use of JSTL tags in JSP?

2. What is JSP error page?

3. Explain the use of deployment descriptor in JSP.

4.9 DATABASE CONNECTIVITY

Database access is one of the important features of Web application. Java has its own technology for database connectivity called JDBC (Java Database Connectivity). JDBC provides a standard library for accessing a wide range of relational databases like MS-Access, Oracle and Microsoft SQL Server. Using JDBC API, you can access a wide variety of different SQL databases. The core functionality of JDBC is found in `java.sql` package.

In the previous sections, you have seen examples on Scriptlets and Java Bean where data is fetched from HTML form and displayed on JSP document in a static manner. This section will provide you in-depth knowledge of data access, specifically insertion and retrieval of data to/from a database.

Consider a table named as Student is created in Microsoft SQL Server database with Roll No, Name and Course name. This table is used in both the following sections. This section defines example for storing/retrieving data into/from a Microsoft SQL Server using type-4 JDBC driver. You can run these programs on any web server such as Tomcat. For running these programs, you need a JDBC driver in .jar form and place them in lib directory under the web application.

4.9.1 Insert data in Database using JSP

Let us take an example: a JSP form named “Form.jsp” contains student’s details like student enrolment no., student name and Programme name. When you submit these details it access ‘actionPage.jsp’ document to store data into the Database.

Source Code for Form.jsp

```
<html>
<body>
<form name="form1" method="post" action="actionPage.jsp" >
<h3>Indira Gandhi Open University </h3>
<fieldset>
<legend><b>Student Details </b></legend>
Enrolment No.:<input name="srno" type="text" value="" size=9><br>
Student Name : <input name="sname" type="text" value="" size=15><br>
Programme : <input type="text" name="prg" value="" size=15> <br>
<input type="submit" value="Submit" />
</fieldset>
</form>
</body>
</html>
```

In the following source code, the first step is to get the data from 'Form.jsp' program using request.getParameter() method. After connecting to Database, an insert query is executed using executeUpdate() method. The program is written by using try and catch clause of standard Java mechanism within JSP scriptlets and data connectivity through the Type-4 driver of SQL Server.

Source Code for actionPage.jsp

```
<%@ page import="java.util.*" %>
<%@ page import="java.sql.*;" %>
<html>
<head><title>Insert data into database</title></head>
<body>
<h3>Insert data in Database using JSP</h3>
<table border=1>
<%
String rollNo = request.getParameter("srno");
String StuName = request.getParameter("sname");
String prgName = request.getParameter("prg");

Connection con = null; //create connection object
Statement stmt = null; // create statement object

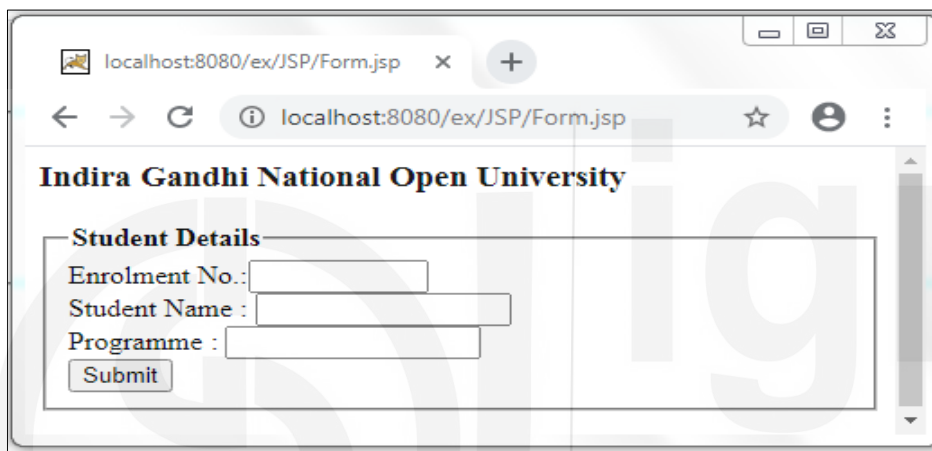
// connection string using Type-4 Microsoft SQL Server driver
// you can also change the next line with your own environment
String url=
"jdbc:microsoft:sqlserver://127.0.0.1:1433;user=sa;password=poonam;DatabaseName=SOCI
S";
try
{
// load JDBC type-4 SQL Server driver
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
con = DriverManager.getConnection(url);
if (con != null)
{
stmt = con.createStatement();
//insert query
String rsq1="insert into student values("+rollNo+", "+StuName+", "+prgName+"")";
//execute query
```

```

        stmt.executeUpdate(rsq);
        out.println("Your data is successfully stored in database");
    }
    if(con == null)
    {
        con.close(); // release connection
    }
}
// end of try clause
catch(SQLException se){ out.print("SQL:"+se.getMessage());}
catch(Exception e){ out.print("Exception:"+e.getMessage());}
%>

```

When you run the 'Form.jsp' program, you will see the following screen (figure-21):



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/ex/JSP/Form.jsp'. The page title is 'Indira Gandhi National Open University'. Below the title, there is a section titled 'Student Details' containing three input fields: 'Enrolment No.', 'Student Name', and 'Programme'. A 'Submit' button is located below these fields.

Figure 21: Input Form for storing data into database

In the above screen, when you will enter values then the following screen (figure-22) will show you a message for data storage



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/ex/JSP/actionPage.jsp'. The page title is 'Insert data into database'. Below the title, there is a section titled 'Insert data in Database using JSP' followed by the message 'Your data is successfully stored in database'.

Figure-22 : Data stored in persistent storage

4.9.2 Retrieve Data from Database using JSP

The following example gives you an illustration of how to retrieve data from the Database. After execution of the above program, you have stored sufficient data into Database. Now, you will execute the following code for retrieving the data from Database. In this program, one additional ResultSet object is used for retrieving data from select query. The data is retrieved from ResultSet object by using getXXX() method such as getInt() and getString(). Note that if the column name is an integer type, you should use the getInt() method instead of the getString() method.

Source Code for getData.jsp

```
<%@ page import="java.util.*" %>
<%@ page import="java.sql.*" %>
<html>
<head><title>Retrieved data from database</title></head>
<body>
<h3> Retrieve Data from Database using JSP</h3>
<table border=1>
<%
Connection con = null; //create connection object
Statement stmt = null; // create statement object
ResultSet rs = null;    // create ResultSet object

// connection string using Type-4 Microsoft SQL Server driver
// you can change the next line with your own environment
String url=
"jdbc:microsoft:sqlserver://127.0.0.1:1433;user=sa;password=poonam;DatabaseName
=
SOCIS";
Try
{
    // load sql server JDBC type-4 driver
    Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");

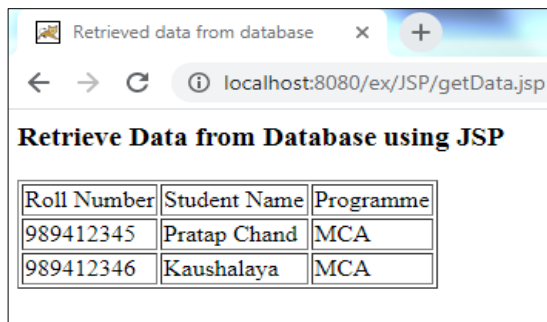
    con = DriverManager.getConnection(url);

    if (con != null)
    {
        stmt = con.createStatement();

        // select SQL statement
        String rsq1 ="select * from Student";

        //execute query
        rs = stmt.executeQuery(rsq1);
        %>
<tr><td>Roll Number</td><td>Student Name</td><td>Programme</td></tr>
<%
        while( rs.next() )
        {
            %><tr>
            <td><%= rs.getInt("RollNo") %></td>
            <td><%= rs.getString("Student_Name") %></td>
            <td><%= rs.getString("Programme") %></td>
            </tr>
            <%
        }
    }
}
if(con == null) {con.close();}
}
catch(SQLException se){ out.print("SQL:"+se.getMessage());}
catch(Exception e){ out.print("Exception:"+e.getMessage());}
%>
```

After running the above program, the following output screen (figure-23) is displayed:



Roll Number	Student Name	Programme
989412345	Pratap Chand	MCA
989412346	Kaushalaya	MCA

Figure 23: Display data from database

If you want to build your project using Oracle Database as back end, for this you can change only port no and Oracle JDBC driver name in the above source code.

4.10 SUMMARY

In this unit you have gone through the components of 'Java Server Pages' which makes the entire JSP document as directives, scripting and action elements. A 'Java Server Pages' component is a type of Java servlet that is designed to fulfil the role of a user interface for a Java web application. This Unit also introduced you the JSP API on which the entire JSP relies. JSP API is a set of classes and interfaces that is used for making JSP pages. JSP supports nine automatically defined variables which are called implicit objects. When you write a JSP program, you might face a run time error(s). For this JSP provides exception handling through the page directive element, core java mechanism and deployment descriptor file.

Now, you are able to create a JSP document that can be used for a variety of purposes such as retrieving information from a database, passing control between pages and accessing Java Beans components.

4.11 SOLUTIONS/ANSWERS TO CHECK YOUR PROGRESS

☛ Check Your Progress 1

- 1) When a client requests a JSP page, the browser sends a request to web server which is forwarded to the JSP engine. If it is a new request from the client then it is translated and compiled by the JSP engine into a servlet. Now, servlet is ready for servicing the client request and generates response which returns to the browser via web server. For the second time the same request from the client (including browser and web server request) to JSP engine, the JSP engine determines the request that the JSP-Servlet is up-to-date, if yes then it immediately passes control to the respective JSP-Servlet for fulfilling the request
- 2)

```
<html><body>
<%! int n = 10, num1 = 0, num2 = 1; %>
<% for (int i = 1; i <= n; i++)
{
    out.print(num1 + " , ");
```

```
        int sum = num1 + num2;
        num1 = num2;
        num2 = sum;
    }
    %>
</body></html>
```

```
3)  <html><body>
    <% int sum=0;
    for (int i = 1; i <=10; i++)
        { sum = sum+i ; }
    out.print("sum =" + sum);

    %>
    </body></html>
```

☛ Check your Progress 2

- 1) Standard actions are tags that affect the runtime behaviour of the JSP. These JSP action tag is used to perform some specific tasks such as insert a file dynamically, reuse external JavaBean components, forward the request to the other page and generate HTML for Java Applet Plugin.
- 2) JSP provides different scope for sharing data between web pages. These are:
 - ... Page - 'page' scope means the JSP object can be accessed only from within the same page where it is created. By default, it is page. JSP implicit objects out, exception, response, pageContext, config and page have 'page' scope.
 - ... Request – Beans with request scope are accessible only within pages processing the same request that the object was created in. Objects that have request scope are most often used when you need to share information between resources that is pertinent for the current request only. Only Implicit object request has the 'request' scope.
 - ... Session – This object is accessible from any JSP within the same session. Implicit object session has the 'session' scope.
 - ... Application - This object is accessible from any JSP within the same web application. Implicit object application has the 'application' scope.
- 3) It is used for getting initialization parameters and for sharing the attributes and their values across the entire JSP application. For example, `<%=application.getServerInfo()%>`, it returns the name and version of the servlet container on which the servlet is running.

☛ Check your Progress 3

- 1) The 'Java Server Pages' Standard Tag Library (JSTL) contains a set of tags to simplify the JSP development. JSTL provides tags to control the JSP page behaviour and common tasks such as xml data processing, conditionals execution, iteration and SQL tags.
- 2) A JSP error page is designed to handle runtime errors and display a customized view of the exception. You can include an error page in your application at page or application level. At page level, you can use page directive or standard java mechanism options. At the application level, you can only use an `<error page>` element of the deployment descriptor.

- 3) This file is an xml file whose root element is `<web-app>`. It is reside in the web applications under the `WEB-INF/` directory. You can configure JSP tag libraries, welcome files, customizing HTTP error code or exception type. You can use the `<error-page>` element in the deployment descriptor to specify exception type or HTTP error code and location of the error page. The JSP tag libraries can be defined using the `<tag-lib>` element of the deployment descriptor.

4.12 REFERENCES/FURTHER READINGS

- ... Kathy Sierra, Bryan Basham, anddd Bert Bates ,“Head First Servlets and JSP”, O'Reilly Media, Inc., 2008.
- ... Brown-Burdick et al., “Professional JSP”, Apress,2001.
- ... Budi Kurniawan , “Java for the Web with Servlets, JSP, and EJB: A Developer's Guide to J2EE Solutions: A Developer's Guide to Scalable Solutions” *Techmedia* , 2002.
- ... James Goodwill , “Pure JSP”, Techmedia, 2017.
- ... https://docs.oracle.com/cd/E17802_01/products/products/jsp/2.1/docs/jsp-2_1-pfd2/javax/servlet/jsp/package-summary.html
- ... <https://docs.oracle.com/javaee/5/api/javax/servlet/jsp/tagext/package-summary.html>
- ... <https://docs.oracle.com/javaee/5/jstl/1.1/docs/tlddocs/>
- ... <https://tomcat.apache.org/taglibs/index.html>
- ... list of drivers is available at : <http://www.devx.com/tips/Tip/28818>