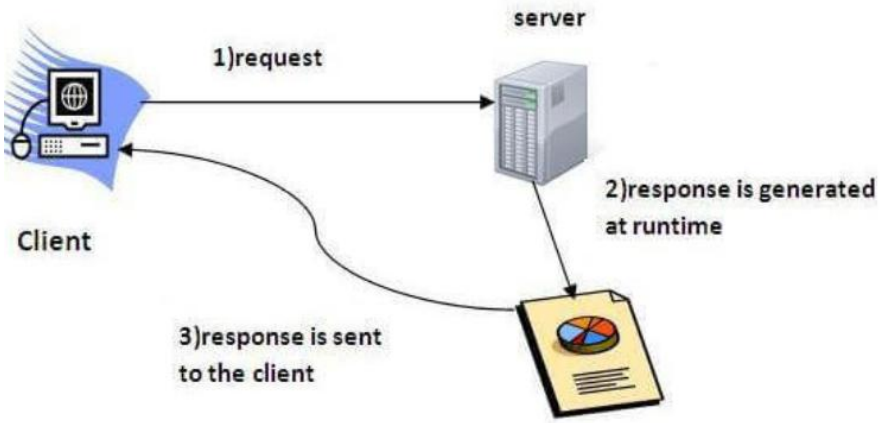
MODULE – IV Server-Side Programming: Java Servlets - Model-View-Controller Paradigm - Servlet Architecture Overview - Servlets Generating Dynamic Content - Servlet Life Cycle - Parameter Data - JSP/Servlet (J2EE-Java 2 Enterprise Edition) /Advanced Java

Servlets Technology

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

What is a Servlet?

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



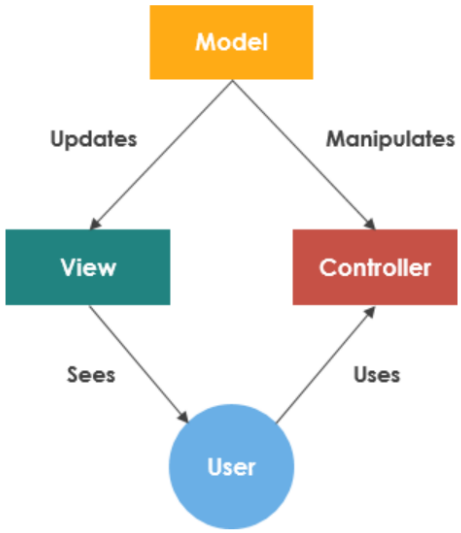
What is a web application?

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

What is Model-View and Control?

MVC (Model-View-Controller) is an architectural design pattern that encourages improved application organization through a separation of concerns. It divides an interactive application into three components: Model / View and Controller. It enforces the isolation of business data (Models) from user interfaces (Views), with a third component (Controllers) traditionally managing logic, user-input and coordinating both the models and views. The goal of MVC is to help structure the separate the concerns of an application into three parts:

- **Model** is responsible for managing the data of the application. It receives user input from the controller.
- **View** means the presentation of the model in a particular format.
- **Controller** responds to the user input and performs interactions on the data model objects. The controller receives the input, optionally validates it and then passes the input to the model.



The Purpose of MVC Framework

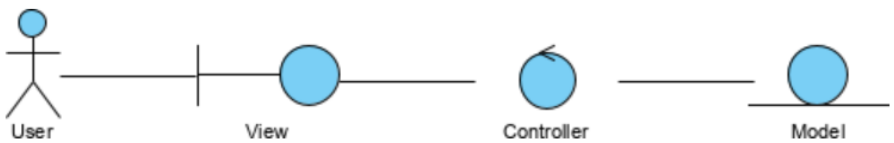
As mentioned above, MVC software framework helps us to separate the different aspects of the application (input logic, business logic, and GUI), while providing a loose coupling between these elements. Thus, the information (most reusable) logic belongs in the model, the GUI belongs in the view. Input logic belongs in the controller. This separation helps you manage complexity when you build an application because it enables you to focus on one aspect of the implementation at a time. MVC Framework is a good idea for a number of reasons, including:

- Simultaneous development – Because MVC decouples the various components of an application, developers are able to work in parallel on different components without affecting or blocking one another.
- Reusability – The same (or similar) view for one application can be refactored for another application with different data because the view is simply handling how the data is being displayed to the user.
- improved scalability – if your application begins experiencing performance issues because database access is slow, you can upgrade the hardware running the database without other components being affected
- Low coupling — the very nature of the MVC framework is such that there is low coupling among models, views or controllers.
- better extendibility – as the components have a low dependency on each other, making changes to one (to fix bugs or change functionality) does not affect another

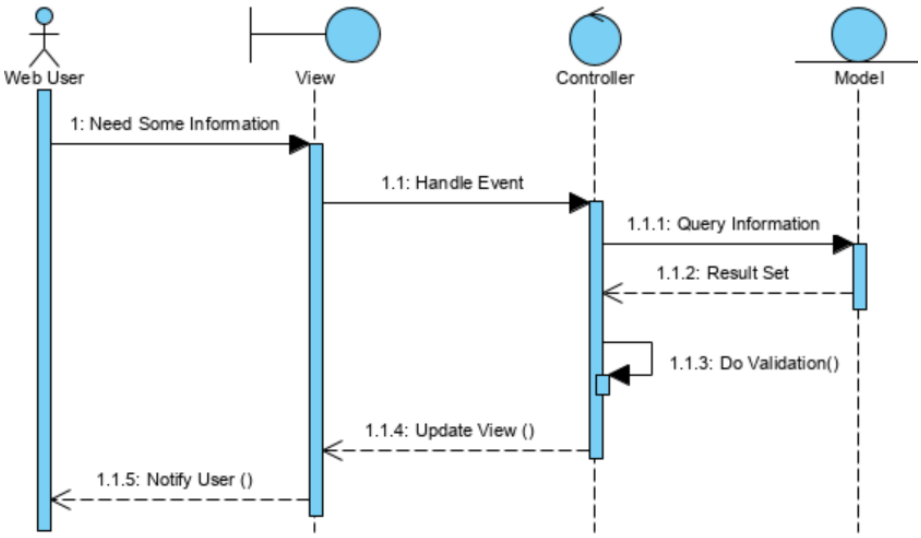
Develop Use Case Scenario Using MVC Sequence Diagrams

You can use stereotypes for the lifeline in the MVC sequence diagram to make visually clear what type of objects you are using in the MVC. An MVC Sequence diagram has interface objects, controller objects and entity objects:

- Entities are objects representing system data: Customer, Product, Transaction, Cart, etc.
- Boundaries are objects that interface with system actors: UserInterface, DataBaseGateway, ServerProxy, etc.
- Controls are objects that mediate between boundaries and entities. A controller object often correspond to use cases scenario and often represented by a sequence diagram.



And here is the simplistic and hypothetical sequence diagram for MVC. What you see in this diagram, a web-user initiated a query and an event is generated that is handled by the controller and gets information that is needed from the model, validates the information and passes back the result set to the view.

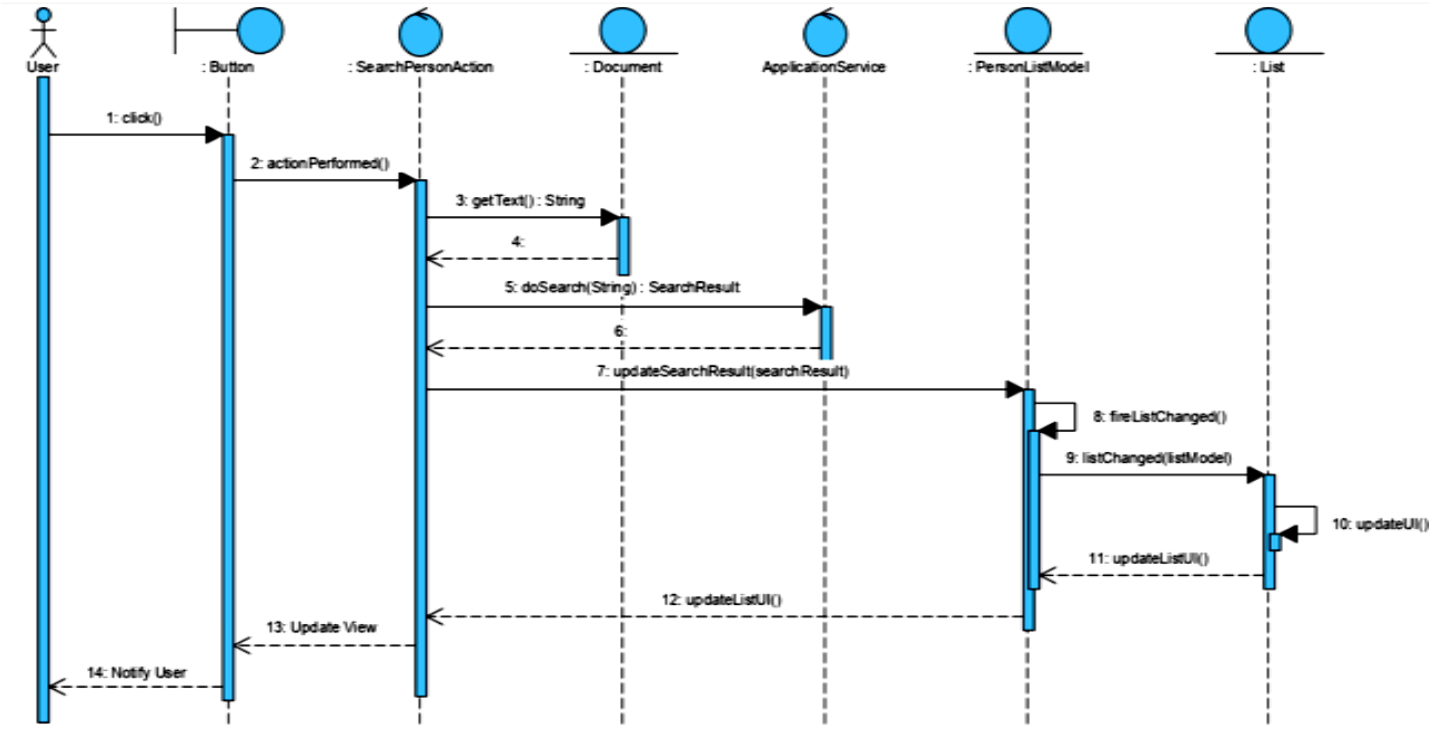


Example MVC Sequence Diagram

Suppose an application that let you search for persons. The UI must have a text field where the user can enter a search string and it might have a button to start the search. Finally it must have an area where the search results are displayed. In our case, it is implemented with a list component.

The “Search for Persons” use case Scenario is:

- The user enters a search string in the text field
- The user clicks the search button.
- The search result is displayed in the result list.



Interfaces in javax.servlet package

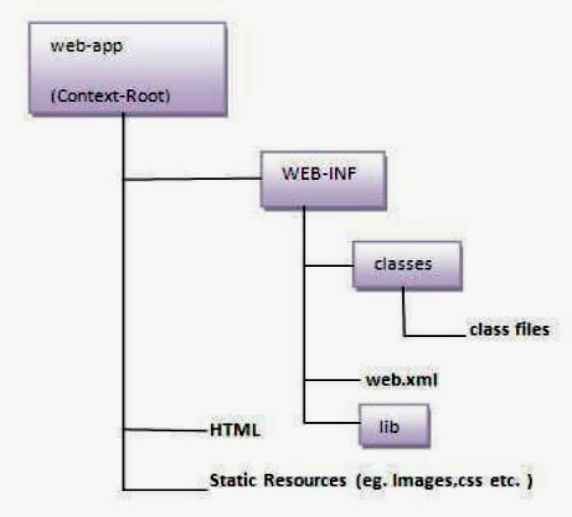
There are many interfaces in javax.servlet package. They are as follows:

1. Servlet
2. ServletRequest
3. ServletResponse
4. RequestDispatcher
5. ServletConfig
6. ServletContext

1)Create a directory structures

The directory structure defines that where to put the different types of files so that web container may get the information and respond to the client.

The Sun Microsystem defines a unique standard to be followed by all the server vendors. Let's see the directory structure that must be followed to create the servlet.



2)Create a Servlet

There are three ways to create the servlet.

1. By implementing the Servlet interface
2. By inheriting the GenericServlet class
3. By inheriting the HttpServlet class

The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost, doHead() etc.

In this example we are going to create a servlet that extends the HttpServlet class. In this example, we are inheriting the HttpServlet class and providing the implementation of the doGet() method. Notice

that get request is the default request.

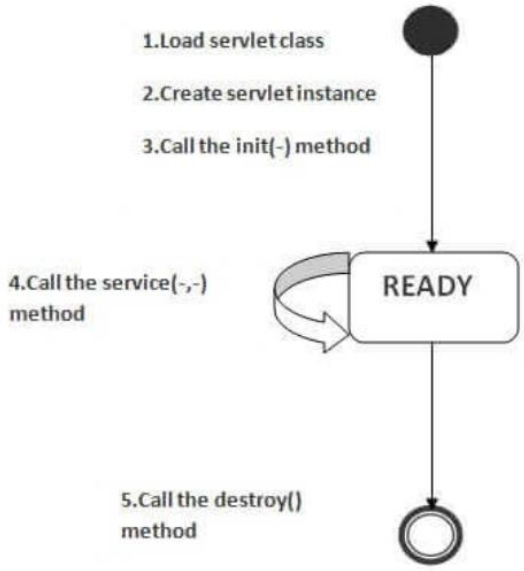
DemoServlet.java

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class DemoServlet extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res)
                                        throws ServletException,IOException
{
        res.setContentType("text/html");       //setting the content type
        PrintWriter pw=res.getWriter();        //get the stream to write the data
                        //writing html in the stream
        pw.println("<html><body>");
        pw.println("Welcome to servlet");
        pw.println("</body></html>");
        pw.close();                            //closing the stream
}
}
```

Life Cycle of a Servlet (Servlet Life Cycle)

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

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



The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

1) Servlet class is loaded

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

2) Servlet instance is created

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

3) init method is invoked

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

public void init(ServletConfig config) throws ServletException

4) service method is invoked

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

public void service(ServletRequest request, ServletResponse response)

throws ServletException, IOException

5) destroy method is invoked

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

public void destroy()

Advanced Java (Java Enterprise Edition)

Core Java

Core Java (J2SE) and **Advanced Java (JEE)**. The core Java part covers the fundamentals (data types, functions, operators, loops, thread, exception handling, etc.) of the Java programming language. It is used to develop general purpose applications.

Advance Java

It is a part of Java programming language. It is an advanced technology or advance version of Java specially designed to develop web-based, network-centric or enterprise applications. It includes the concepts like **Servlet, JSP, JDBC, RMI, Socket programming, etc.** It is a specialization in specific domain.

- It simplifies the complexity of a building n-tier application.
- Standardizes and API between components and application sever container.
- JEE application Server and Containers provides the framework services.

Benefits of Advance Java

The four major benefits of advance Java that are, network centric, process simplification, and futuristic imaging standard.

- JEE (advance Java) provides libraries to understand the concept of **Client-Server architecture** for web- based applications.
- We can also work with web and application servers such as **Apache Tomcat** and **Glassfish** Using these servers, we can understand the working of HTTP protocol. It cannot be done in core Java.
- It is also important understand the advance Java if you are dealing with trading technologies like **Hadoop, cloud-native** and **data science**.
- It provides a set of services, **API** and **protocols**, that provides the functionality which is necessary for developing **multi-tiered** application, web-based application.
- There is a number of advance Java frameworks like, **Spring, Hibernate, Struts,** that enables us to develop secure **transaction-based** web applications such as banking application, inventory management application.

JDBC
- JDBC Architecture
- Steps to create JDBC Application
- JDBC Driver Types & Connections

Java Servlets
- Introduction to Java Servlets
- Servlet Life Cycle
- Steps to create Servlet
- Session Tracking in Servlets

JSP
- Introduction to JSP
- Life Cycle of JSP

JSP Scripting Elements

Advance Java Topics

1. Basics of a Web application

- What is a web application?
- What is a web client and web server?
- How do client and server communicate?
- HTTP protocol basics
- HTML language basics
- What is a TCP/IP port, URL?
- Need for a Web Container

2. Web Container and Web Application Project Set up

- To set up Tomcat Container on a machine
- To set up a Servlets JSP project in Eclipse
- To configure dependency of Servlet JSP APIs
- Web application project structure

3. Servlets

- What are Servlets?
- What can they do? Why are they needed?
- How do Servlets look in code?
- HTTP Methods; GET, POST, PUT, DELETE, TRACE, OPTIONS
- GET/POST request; differences between the two
- Servlet Lifecycle
- Servlet Context and Servlet Config
- Forwarding and Redirection of requests

4. Session Management

- What is a session?
- Why is it required?
- How to get a session?
- Session information passing between client and server
- Session information passing mechanisms - Cookies, Rewriting
- How to destroy a session

5. JSPs

- Introduction to JSP and need for JSPs
- Basic HTML tags
- JSP Lifecycle

6. JSP Elements

- Scriptlets
- Expressions
- Declarations
- Significance of above elements and fitment into the JSP Lifecycle
- What are Directives in JSP?
- Page Directive
- Include Directives
- Taglib Directive

7. JSP Tag library

- JSP Standard Actions
- Expression Language
- JSTL basics and it's usage
- Need for Custom Tag Library
- Custom Tag Library implementation

Struts Framework (version 2.x)

1. Basics of MVC

- What is MVC?
- MVC Type1 and Type2 architecture
- Why Struts framework?
- Struts 1 overview

- Struts 1 and Struts 2 comparison

2. Struts 2 Architecture

- Architecture Diagram explanation of following components:
- Components of Model, Views and Controller in Struts Framework
- Interceptors
- Model/Action classes
- Value Stack
- OGNL
- Introduction to configurations; framework and application architecture
- Declarative and Annotations configuration approaches

3. Struts 2 set up and first Action class

- Download JAR files
- Struts 2 project build up and Configuration files
- To build Action class
- To intercept an HTTP request via Struts2 framework using Action class
- Defining data and business logic in Action class
- Preparing and Forwarding control to Views

4. Struts 2 Interceptors

- What are Interceptors
- Responsibilities of an Interceptor
- Mechanism of Interceptor calling in Struts 2
- Defining Interceptors
- Defining Interceptor stacks
- Defining Custom Interceptors

5. Struts 2 Tag Library

- Introduction to tag library of Struts 2 and it's usage

6. Struts 2 Validations

- Validations using Validateable interface
- Workflow interceptor mechanism for validations
- Validations using Validateable interface
- Validation Framework introduction and architecture
- Validating user input with above two mechanisms

7. Struts 2 Tiles Frameworks

- Introduction to Tiles in a page
- Struts2 Tiles framework introduction
- Defining tiles.xml file
- Configuring pages for tiles
- A complete Tiles example with Struts2

Hibernate Framework (version 3.x)

1. Introduction

- What is ORM principle?
- Why ORM?
- ORM implementations

2. Hibernate Architecture

- Introduction to Hibernate
- Hibernate Architecture
- What are Persistent classes?

3. Hibernate CRUD

- Setting up Hibernate project
- Configuring all JARs and XML files
- Setting up connection to DB using Hibernate
- Performing basic CRUD operations using Hibernate API
- Object Identity; Generator type classes
- Using SQL with Hibernate
- Using HQL
- Using Criteria queries

4. Mapping Collections and Associations

- To define sets, mas, lists in Hibernate
- Association Mappings:
    1. One to one
    2. One to many
    3. Many to one
    4. Many to many
- Hibernate Caching
- What is caching?
- What are the types of caching in Hibernate?
- Explanation of various caching mechanisms in Hibernate

5. Using Hibernate Annotations

- Sample example of using Hibernate Annotations

Spring Framework (version 3.x)

1. Introduction to spring

- What is Spring?
- Spring Architecture explanation and all it's components

2. Introduction to all modules of Spring

- Spring Bean Factory
- Spring Application Context
- Spring DI
- Spring Integration; Spring messaging, Spring JMS
- Spring MVC
- Spring DAO

3. Setting up spring

- Setting up of Spring framework
- Download JARs
- Configure XML files

4. Dependency Injection

- What is Dependency Injection?
- How is it implemented using Spring Framework?
- Bean Wiring mechanisms in Spring

5. Spring AOP

- What is Spring AOP?
- Implementation of Spring AOP

Spring Boot Framework (Version 2.x)

1. Introduction

- Spring Boot Introduction
- Spring Boot Version
- Spring vs Spring Boot vs Spring MVC
- Spring Boot Architecture

2. Creating Project

- Spring Initializr
- Download & Install STS IDE
- Spring Boot Example
- Spring Boot CLI
- Spring Boot Example-STS

3. Project Components

- Annotations
- Dependency Management
- Application Properties
- Starters
- Starter Parent
- Starter Web
- Starter Data JPA
- Starter Actuator
- Starter Test
- Devtools

- Multi Module Project
- Packaging
- Auto-Configuration

4. Tool Suite

- Hello World Example
- Project Deployment Using Tomcat

5. Spring Boot AOP

- What is AOP?
- AOP Before Advice
- AOP After Advice
- AOP Around Advice
- After Returning Advice
- After Throwing Advice

6. Spring Boot Database

- JPA
- JDBC
- H2 Database
- Crud Operations

7. Spring Boot View

- Thymeleaf View

8. Spring Boot Caching

- What is Caching?
- Cache Provider
- EhCaching

9. Spring Boot Misc

- Run Spring Boot Application
- Changing Port
- Spring Boot Rest Example

Web Services: REST and SOAP

- Logging Framework: Splunk, Log4J, SLF4j
- Version-control system + repository hosting service: Git + Github