selsoft
build better, deliver faster

Developing RESTful Web Services with Java

Chapter 2: Introduction to Java Web Applications

Eğitmen:

Akın Kaldıroğlu

Çevik Yazılım Geliştirme ve Java Uzmanı

Topics

- **Servlets**
  - Servlet Registration
- **JSPs**
  - Servlet vs. JSPs
- **MVC**

Servlets

selsoft
build better, deliver faster

Servlet

- **Servlet** is the most fundamental block of web applications in Java.
- It has been a part of Java EE from the beginning.
- Servlets are Java counterpart of CGI or ASP.NET, etc., technologies for to handle HTTP requests coming to web applications.
- So they are used to create HTML pages dynamically.

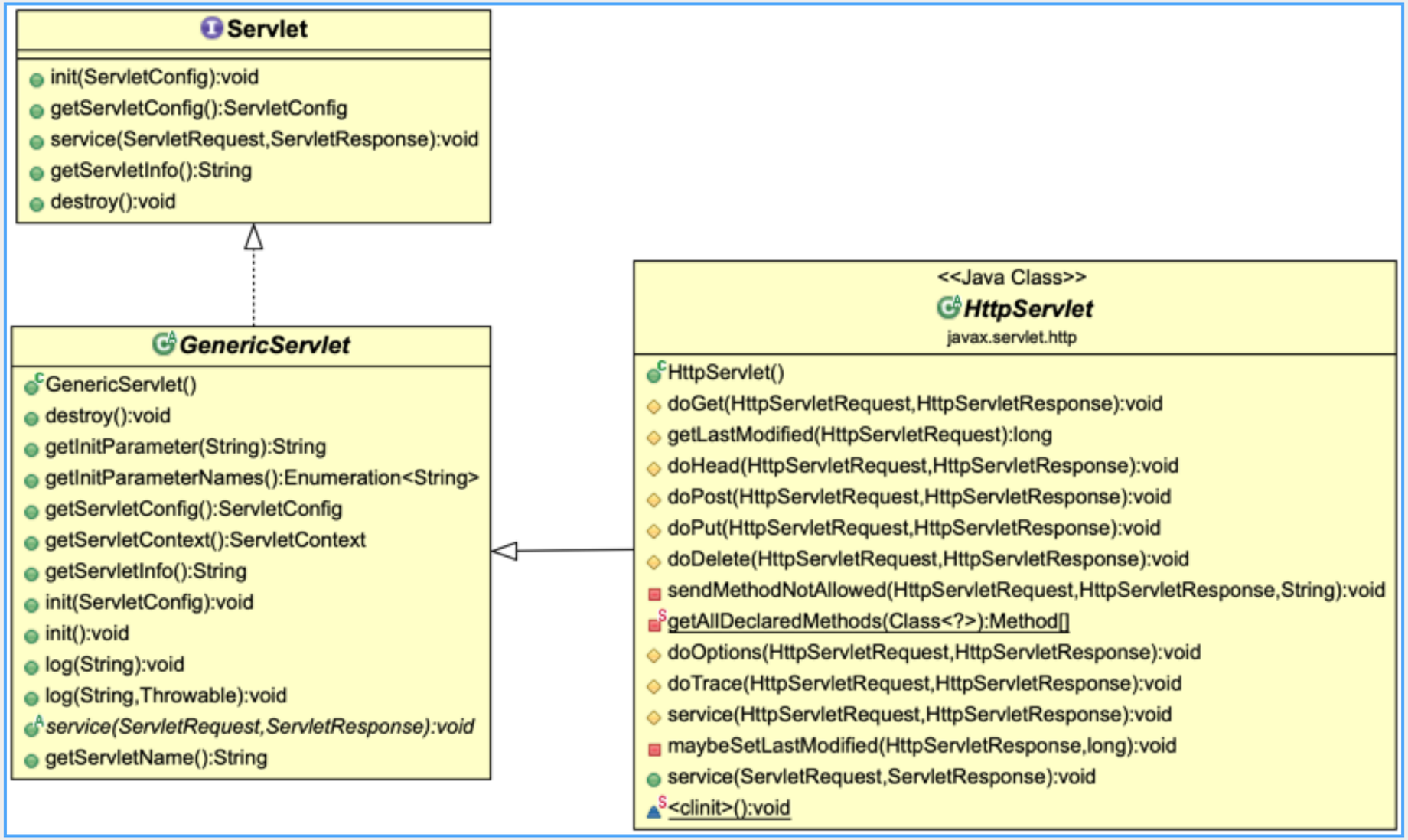
Servlet vs. Applet

- Applet and servlet are two terms coined in the early days of Java.
- Applet is a program that runs inside the browser, i.e. client side while servlet is a program that runs inside the server, i.e. server side.

HttpServlet - I

- **`javax.servlet.http.HttpServlet`** is an abstract class as the main abstraction to create servlets that handle HTTP requests.
- It is a subclass of another abstract class **`javax.servlet.GenericServlet`**, which in turn implements **`javax.servlet.Servlet`** interface.
- **`Servlet`** interface and **`GenericServlet`** abstract class provide underlying framework to write server side components working in a request-response paradigm.

Servlet Hierarchy



HttpServlet - II

- `HttpServlet` is just a customized component created within that framework to specifically work with HTTP clients.
- Thus it handles HTTP requests and produces HTTP responses.

Extending HttpServlet

- A servlet should extend **`HttpServlet`** and override its methods according to its needs.
- **`HttpServlet`** class provides one method to respond each of HTTP request methods
- Thus it has 7 methods for 7 HTTP request methods.

doXXX() Methods - I

- `HttpServlet` has, among others, following methods:
  - `doGet` for GET method
  - `doPost` for POST method
  - `doHead` for HEAD method
  - `doPut` for PUT method
  - `doOptions` for OPTIONS method
  - `doDelete` for DELETE method
  - `doTrace` for TRACE method

doXXX() Methods - II

- All of these seven methods have the same interface except their names:

```
protected void doXxx(HttpServletRequest, HttpServletResponse)
```

- They all receive **`HttpServletRequest`** & **`HttpServletResponse`**
- They are both interfaces in **`javax.servlet.http`** package and encapsulates HTTP request and response objects, respectively.
- They are created and passed or injected into all **`doXXX()`** methods by the servlet container.

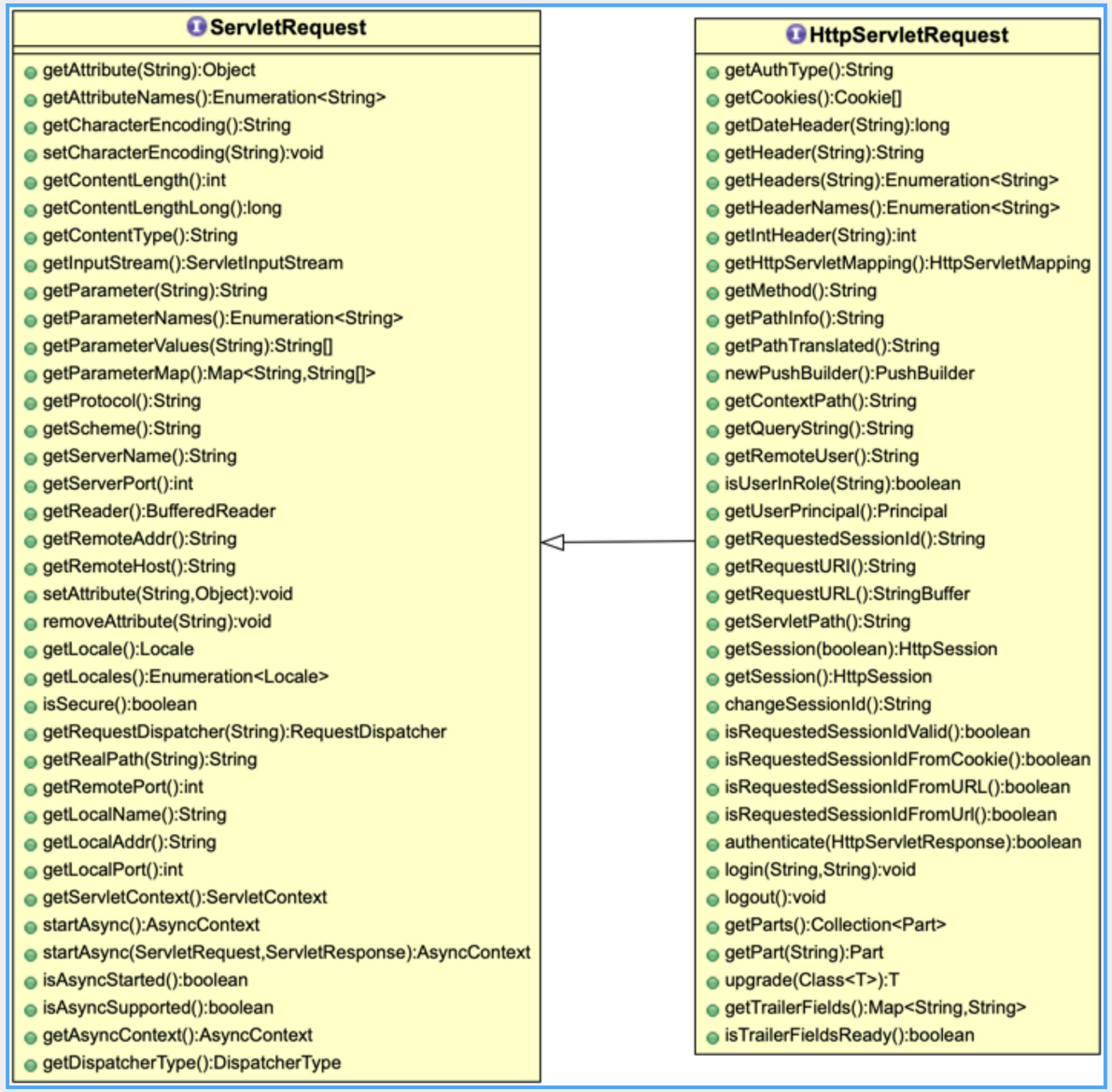
doXXX() Methods - III

- So in `doXXX()` methods, request is processed and response is constructed.
- That's, the information in and about the request is processed in business logic and outcome is loaded onto the response.

HttpServletRequest

- **`HttpServletRequest`** represents HTTP request.
  - So it has all of the information in and about the HTTP request.
  - It is a subinterface of **`javax.servlet.ServletRequest`**.
- **`HttpServletRequest`** represents the HTTP request object and has methods by which you can find out about incoming information such as HTTP request headers or parameters.

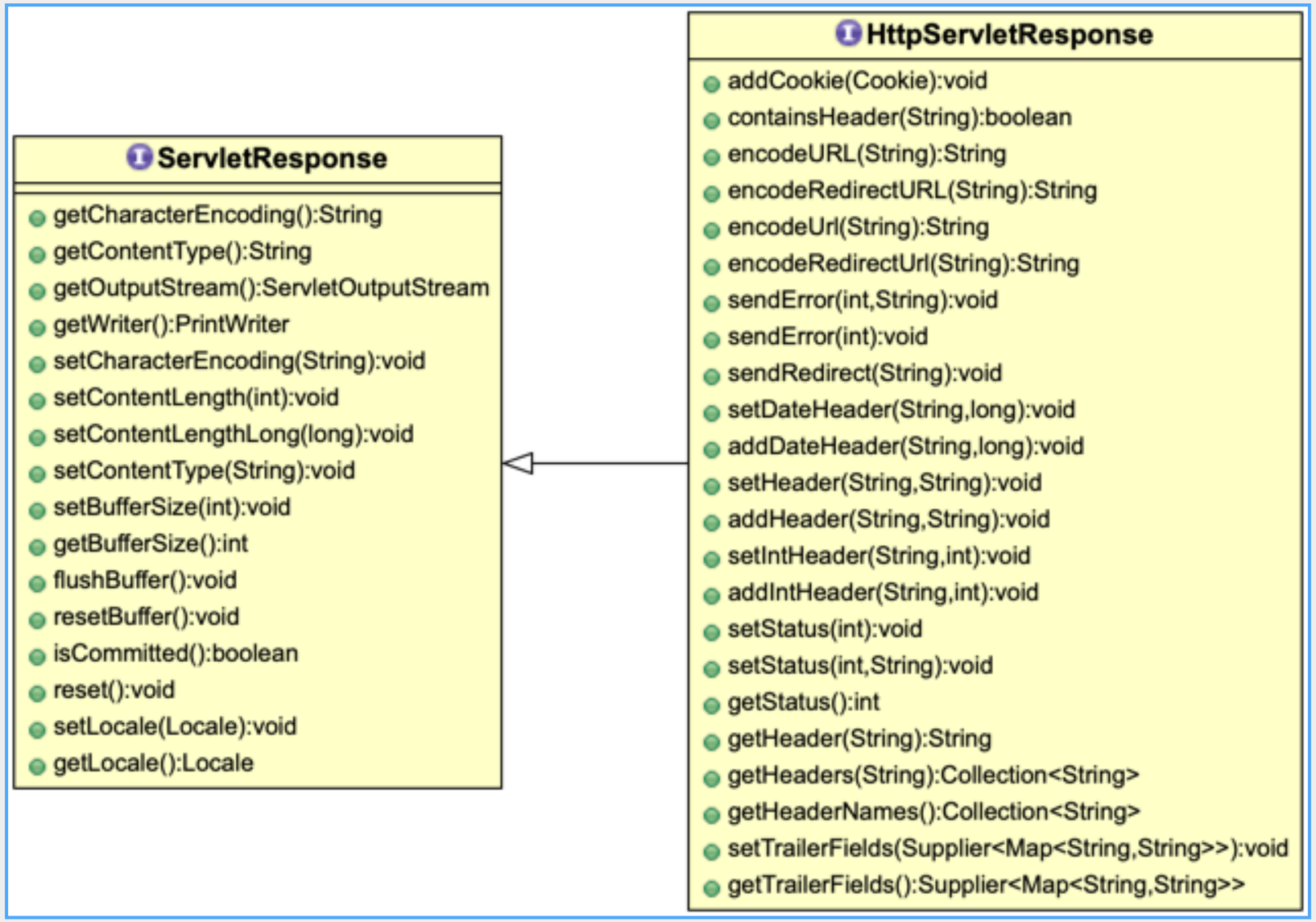
ServletRequest Hierarchy



HttpServletResponse

- `HttpServletResponse` represents the response object to be sent by the server to the client.
- It lets the server specify outgoing information as its response.
- It also lets set headers for the response.
- It is a subinterface of `javax.servlet.ServletResponse`.

ServletResponse Hierarchy



Exceptions

- They all throw two exceptions:
  - `javax.servlet.ServletException`
  - `java.io.IOException`

```
protected void doXxx(HttpServletRequest, HttpServletResponse)
                     throws ServletException, IOException
```

Overriding doXXX() Methods

- A servlet should override any of these methods that correspond to HTTP request methods for which it wants to produce a response.
- A servlet may choose to override any combination of these methods according to its functionality.
- Servlets mostly overrides one or two methods and they are mostly for GET and POST methods.

```java
@WebServlet(name = "SelamServlet3", urlPatterns = { "/SelamServlet3", "/selam3" })
public class SelamServlet3 extends HttpServlet {
     public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
          response.setContentType("text/html");
          PrintWriter out = response.getWriter();
          String docType = "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 Transitional//EN\">\n";
          out.println(docType);
          out.println("<HTML>");
          out.println("<HEAD><TITLE>SelamServlet3</TITLE></HEAD>");
          out.println("<BODY>");
          out.println("<h1 align=\"center\">SelamServlet3</h1>");
          out.println("<H1>Selam3 via GET!</H1>");
          out.println("</BODY></HTML>");
          out.close();
     }

     public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
          response.setContentType("text/html");
          PrintWriter out = response.getWriter();
          String docType = "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 Transitional//EN\">\n";
          out.println(docType);
          out.println("<HTML>");
          out.println("<HEAD><TITLE>SelamServlet3</TITLE></HEAD>");
          out.println("<BODY>");
          out.println("<h1 align=\"center\">SelamServlet3</h1>");
          out.println("<H1>Selam3 via POST!</H1>");
          out.println("</BODY></HTML>");
          out.close();
     }
}
```

SelamServlet

- In **WAP4.0** `org.javaturk.wap.ch04.SelamServlet`

Several Different Servlets

- In **WAP**
  - `org.javaturk.wap.ch05.ClientInformationServlet`
  - `org.javaturk.wap.ch05.RequestHeadersServlet`
  - `org.javaturk.wap.ch05.BrowserCheckerServlet`

SelamServlet

- In **WAP**
  - `org.javaturk.wap.ch04.HelloJapanServlet`
  - `org.javaturk.wap.ch04.HelloArabicServlet`
  - `org.javaturk.wap.ch06.StatusCodesServlet`

service() Methods

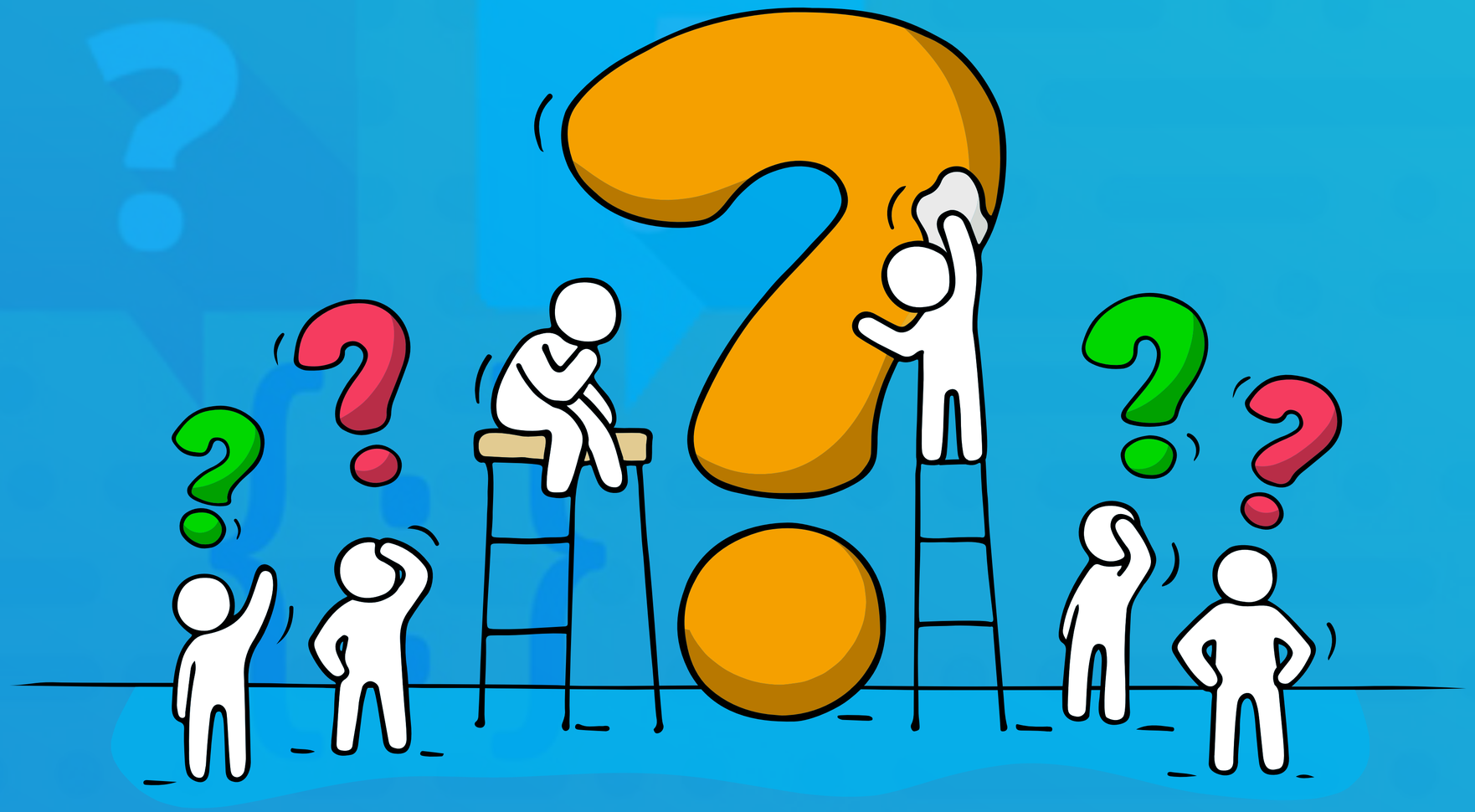
- **`HttpServlet`** class has several other methods:

```
public void service(ServletRequest, ServletResponse)
```

- Is inherited from **`Servlet`** interface and dispatches the request to following method, which dispatches it to the appropriate **`doXxx()`** method

```
public void service(HttpServletRequest, HttpServletResponse)
```

Time for questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

selsoft

build better, deliver faster

Servlet Registration

web.xml - I

- **`web.xml`** is the file for settings of the web application.
- It should be in **`WEB-INF`** directory under the root.
- Traditionally all web application properties such as servlet, filter etc. registrations, initialization paramters, welcome pages, etc. are defined in **`web.xml`**.
- That's why **`web.xml`** was used to be mandatory.
- It was the only way to all these registrations.

Servlet Registration in web.xml

- A servlet is registered using its class name and url patterns in **`web.xml`**.

```
<servlet>
    <servlet-name>SelamServlet</servlet-name>
    <servlet-class>org.javaturk.wap.ch04.SelamServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>SelamServlet</servlet-name>
    <url-pattern>/SelamServlet</url-pattern>
    <url-pattern>/selam</url-pattern>
    <url-pattern>/selam/*</url-pattern>
</servlet-mapping>
```

Annotations

- But starting servlet version 3.0 in Java EE 6, annotations to register servlets, filters and specify their properties have been introduced.

- `HandlesTypes`
- `HttpConstraint`
- `HttpMethodConstraint`
- `MultipartConfig`
- `ServletSecurity`
- `WebFilter`
- `WebInitParam`
- `WebListener`
- `WebServlet`

Servlet Registration with Annotations

- **`@WebServler`** is used to register servlets,
- **`@WebInitParam`** is used to register initialization parameters for a servlet.

```
@WebServlet(urlPatterns = { “/InitialParameterServlet3”, , “/ips" }, initParams = {
          @WebInitParam(name = "ilAdi", value = "Balikesir", description = "Name of the providence."),
          @WebInitParam(name = "ilceAdi", value = "Ayvalik", description = "Name of the city."),
          @WebInitParam(name = "ulkeAdi", value = "Turkiye", description = "Name of the country.") })
public class InitialParameterServlet3 extends InitialParameterServlet {
   ...
}
```

SelamServlet3

- In **WAP** `org.javaturk.wap.ch04.SelamServlet3`

InitialParameterServlet3

- In **WAP** `org.javaturk.wap.ch05.InitialParameterServlet3`

selsoft

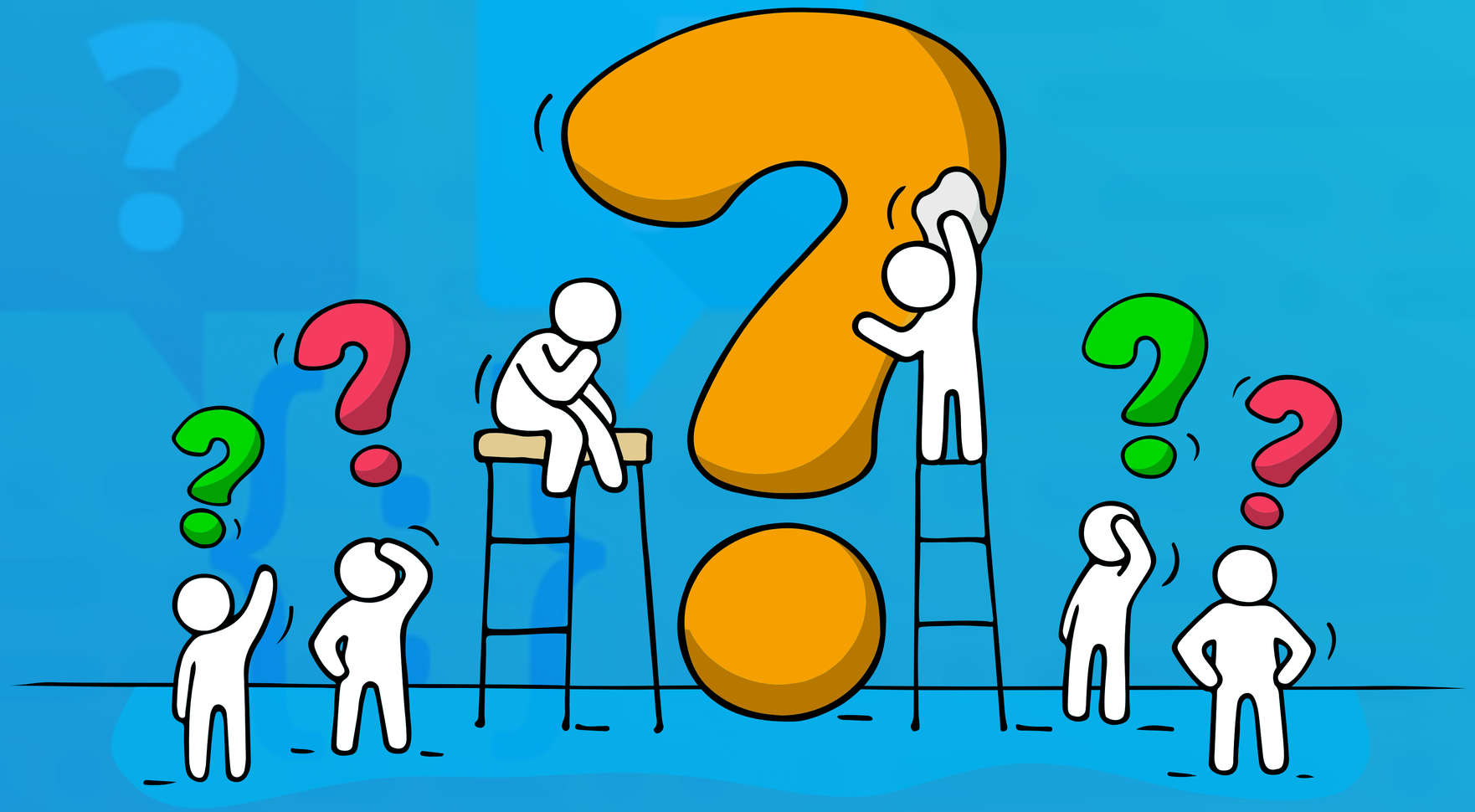
build better, deliver faster

Exercise

Exercise

- Create a servlet that prints date and time to the page.
- You can use **`web.xml`** or annotations for registration.

Time for questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

JSPs

selsoft
build better, deliver faster

JSP

- **JSP**, **Java Server Pages** is a technology based on servlets.
- It allows to write HTML pages by using special tags inside files with an extension of .jsp that run to produce HTML content.
- So it simplifies the process for developing dynamic HTML pages.
- JSPs can do whatever servlets do in a different way.

Servlet vs. JSP

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

        out.println("<HTML>");
        out.println("<HEAD><TITLE>SelamServlet</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<H1 align=\"center\">SelamServlet</H1>");
        out.println("<H1>Selam via GET!</H1>”);
        out.println("<H2>" + new Date() + "</H2>");
        out.println("</BODY></HTML>");
        out.close();
    }
}
```

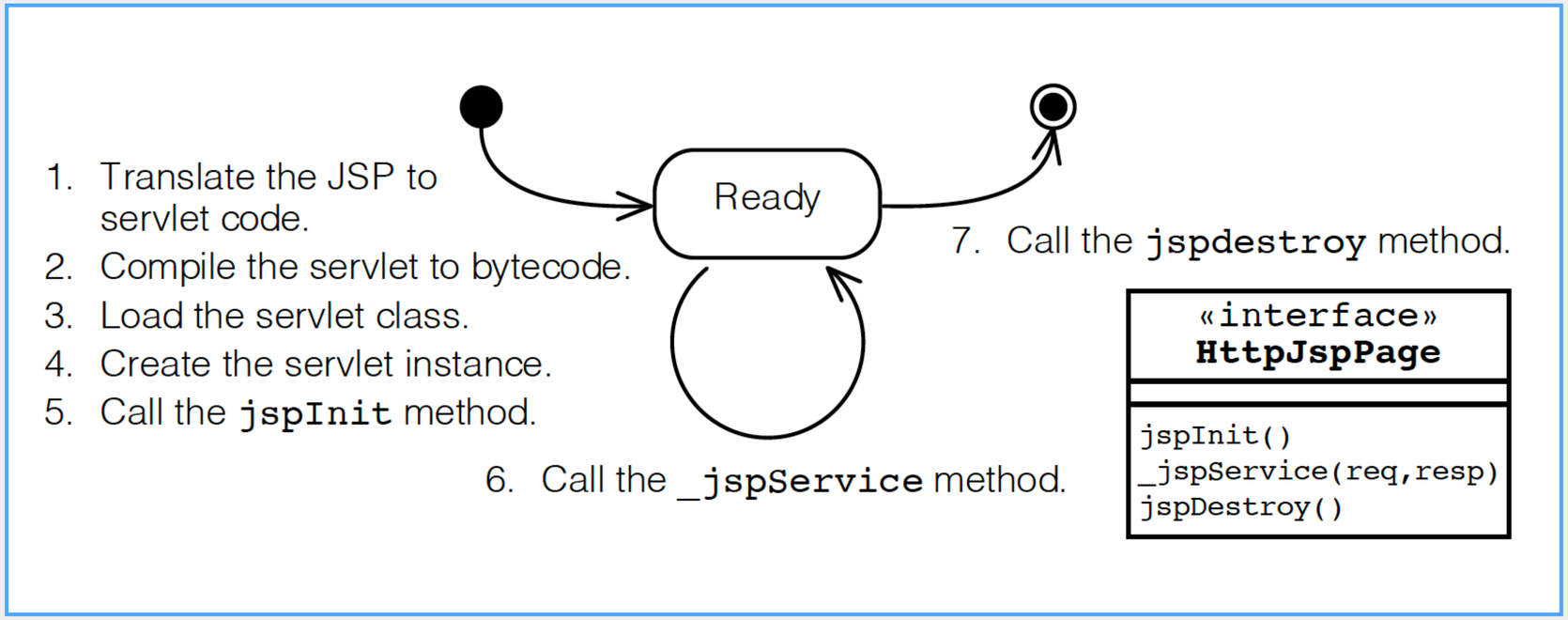
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"%>
<html>
<head>
<title>selam</title>
</head>
<body>
     <h1 align="center">Selam</h1>
     <h2>
          Selam!
          <%="Selam!"%>
          <p>
          <%= new Date()%>
          </p>
     </h2>
</body>
</html>
```

SelamServlet vs. selam.jsp

- In **WAP** `org.javaturk.wap.ch04.SelamServlet` and `ch01/selam.jsp`

JSP Lifecycle

- When it is reached for the first time a JSP is translated into a servlet which eventually handles the HTTP requests.
- So all JSPs are translated into servlet instances and live as servlets.



selsoft

build better, deliver faster

Servlet & JSP

Servlet vs. JSP - I

- What is wrong with the approaches to handling HTTP request and producing HTML pages dynamially in servlets and JSPs?

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

        out.println("<HTML>");
        out.println("<HEAD><TITLE>SelamServlet</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<H1 align=\"center\">SelamServlet</H1>");
        out.println("<H1>Selam via GET!</H1>”);
        out.println("<H2>" + new Date() + "</H2>");
        out.println("</BODY></HTML>");
        out.close();
    }
}
```

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"%>
<html>
<head>
<title>selam</title>
</head>
<body>
      <h1 align="center">Selam</h1>
      <h2>
            <p>Selam!</p>
            <p><%="Selam!"%></p>
            <p><%= new Date()%></p>
      </h2>
</body>
</html>
```

Servlet vs. JSP - II

- Main problem of servlets and JSPs is the fact that they don’t apply the principle of separation of concerns.
  - Processing HTTP request requires Java coding while producing HTTP response requires presentation skills.
- Servlets can only be developed by Java developers but they eventually produce HTML pages which requires use of many design tools for layout, images, color, CSS, etc..
  - HTML pages are mostly developed visually by designers and have strong aesthetic issues.

Servlet vs. JSP - III

- Problem of JSPs is the fact that while they can be developed by web designers they still require Java knowledge.
  - JSPs can have scriplets that in fact are pure Java code.

```
<%@page contentType="text/html"%>
<html>
<head><title>yaziTura</title></head>
<body>
     <h1 align="center">YaziTura</h1>
     <p>
     <h2>
          Your virtual coin has landed on:
          <% if (Math.random() < 0.5) { %>
          Yazi
          <% } else { %>
          Tura
          <% } %>
     </h2>
     </p>
</body>
</html>
```

Solution - I

- So here is the solution that applies separation of concerns:
  - Everybody should do what it is created for:
    - Servlets should do coding and JSPs should be responsible for presenting!
- Servlets should be responsible for Java code and handle only parts that requires programming skills.
- JSPs should be responsible only for look-and-feel part and they don’t require writing any Java code.

Solution - II

- To avoid Java code in JSPs several technologies were developed.
- The most notable ones are **`<jsp:useBean>`** tag and **Java standard tag library**, **JSTL**.
- **`<jsp:useBean>`** tag allows to use Java beans by setting and getting properties.
- JSTL is a set of tags to do some processing required when building the presentation of the page.

<jsp:useBean> & JSTL

- **`<jsp:useBean>`** tag and **JSTL** together satisfy the need for Java code inside JSPs.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1” pageEncoding="ISO-8859-1" session="true"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<jsp:useBean id="user" class="org.javaturk.wap.jsp.ch07.User" scope="session" />
<jsp:setProperty name="user" property="*" />
…
<H2 ALIGN="CENTER">User Info with JSTL</H2>
<c:choose>
     <c:when test="${user.role == ‘manager'}"> <p>Welcome, Manager!</p> </c:when>
     <c:when test="${user.role == ‘poweruser'}"> <p>Welcome, Power User!</p> </c:when>
     <c:otherwise> <p>Welcome, User!</p> </c:otherwise>
</c:choose>
<c:out value="Hello ${user.firstName} ${user.lastName}" />
…
```

user.jsp

- In **WAP** `ch08.userForm.jsp` & `ch08.user.jsp`

leaguesAndCoffees.jsp

- In **WAP** `ch08.leaguesAndCoffees.jsp`

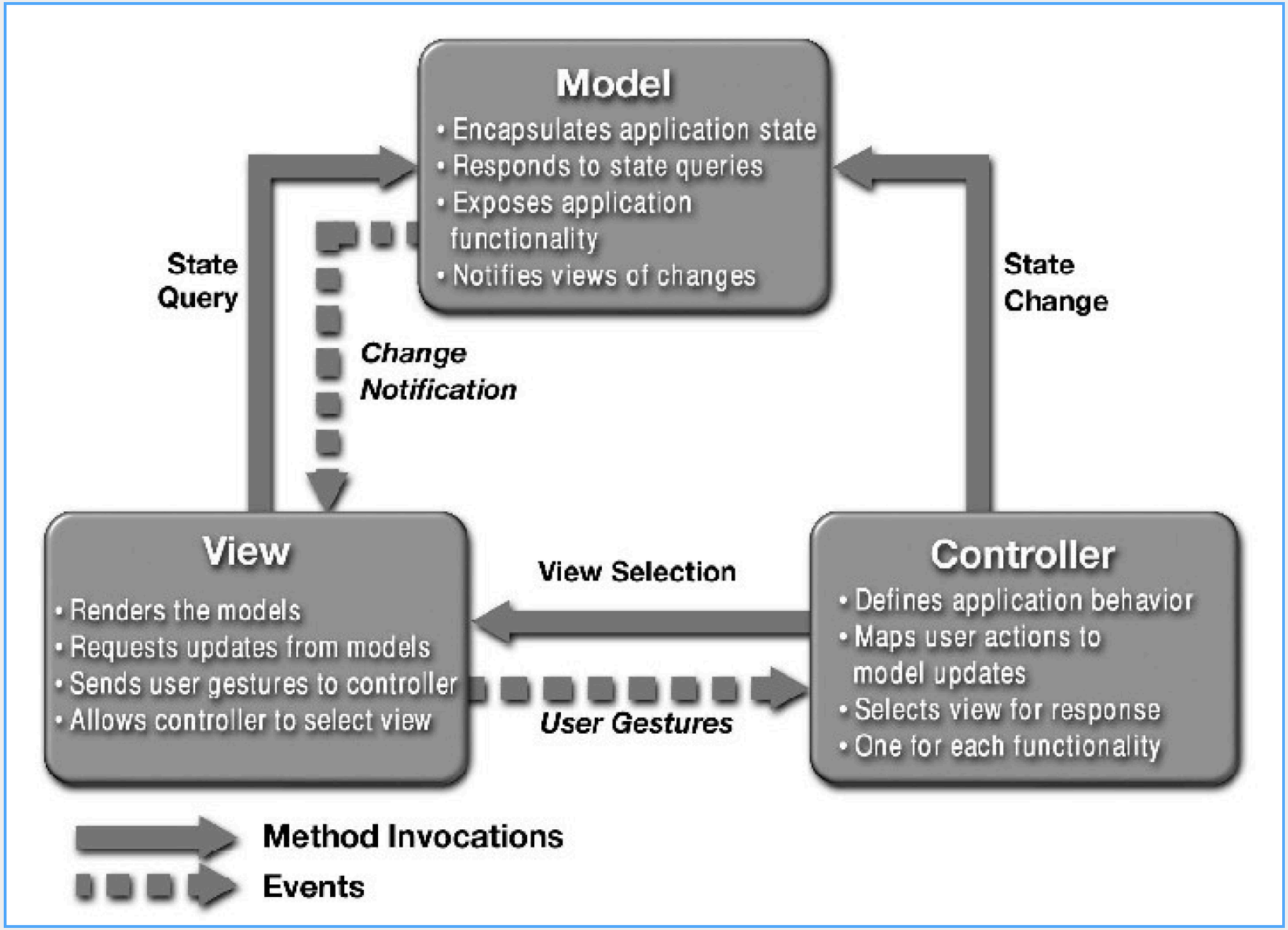
MVC

selsoft

build better, deliver faster

MVC

- **Model-View-Controller** or **MVC** is an architectural pattern commonly used in presentation layers.
  - Model represent business process
  - View represents presentation or user interface
  - Controller represents the logical connection between Model and View.
- It provides different roles for servlets and JSPs.


Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
Change Notification
State Change
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

MVC Frameworks

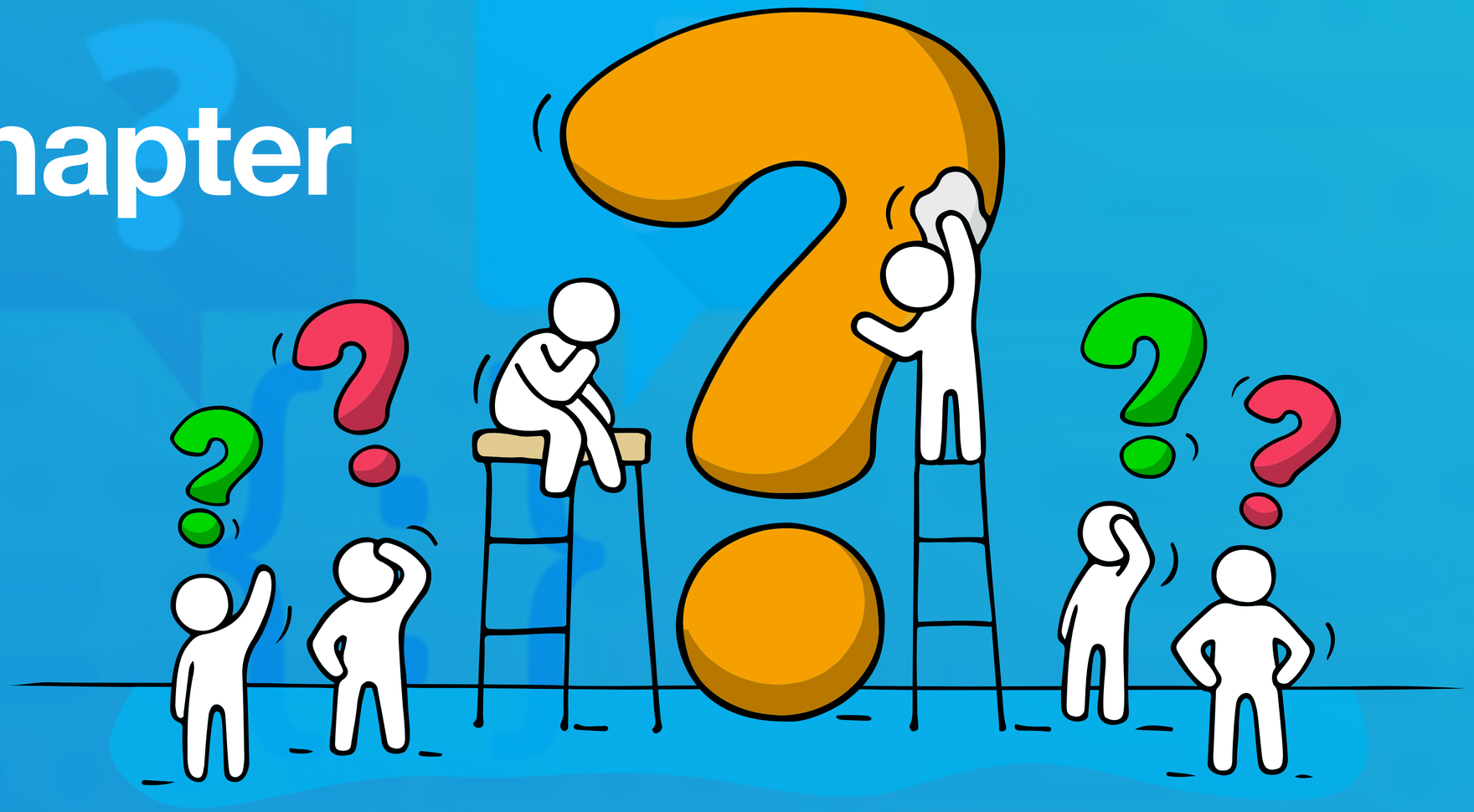
- In Java world there are tons of web frameworks that are based on MVC architecture:
  - Struts was the first non-standard web framework in Java world.
  - Spring MVC is the most used web framework in Java world.
  - JSF is the standard MVC framework in Java EE.
- And overwhelming majority of those frameworks use servlets in their backgrounds.

WAP MVC

- Run application **WAP MVC.**

End of Chapter

Time for Questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr