BBM490

ENTERPRISE WEB ARCHITECTURE

Mert ÇALIŞKAN

Hacettepe University Spring Semester ’14

Recap of Week 01

- We defined Internet and World Wide Web and also talked about the history of them.
- We defined HTML, URI and HTTP.
- We mentioned the logic behind Web Browsers.
- We talked about HTML tags. html | body | head | title | link | style | script and others.

Recap of Week 01

- We mentioned some about JavaScript and CSS.
- We defined HTTP methods and did some requests with GET / POST and others.
- We talked about HTTP codes from 1xx to 5xx.
- We talked about history of Java, companies Sun and Oracle and versions of Java SE and package structure of Java EE.

Resources of BBM490

- Lecture Notes are available at :
  http://bit.ly/BBM490Notes

- Sample Source Codes are available at :
  http://bit.ly/BBM490Codes

Week 2

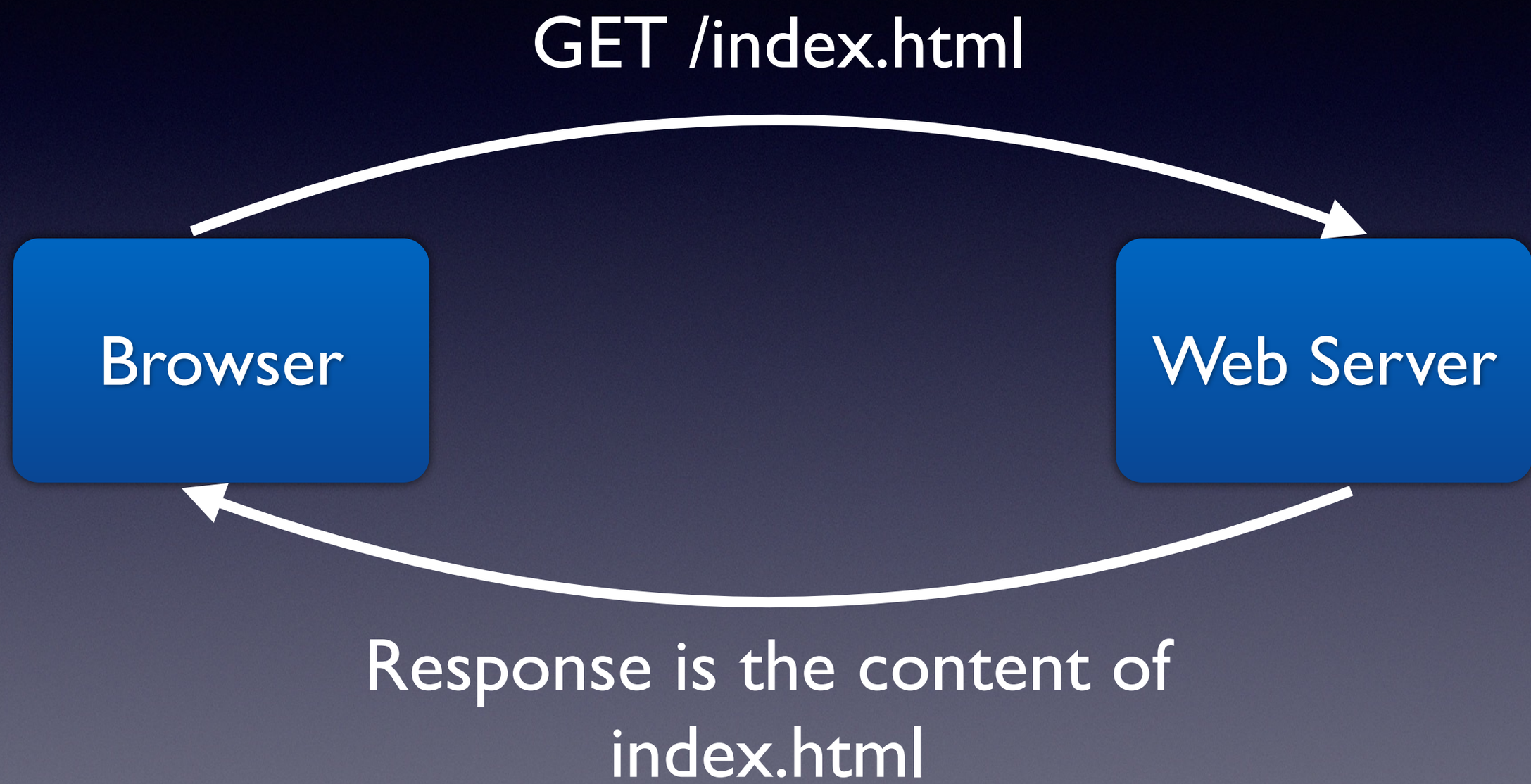
Anatomy of a WebApp

We'll examine the anatomy of Java based Web Applications structured by the Enterprise Edition

Static Applications

- Static HTML files.
- All content would be HTML, Styles, Scripts and Images.
- Not interacting with User, no data input.
- But this would be pretty simple and will not ease anyone's life…

Static Applications



Let's create one…

- We'll be using Eclipse Kepler. You can download it through:

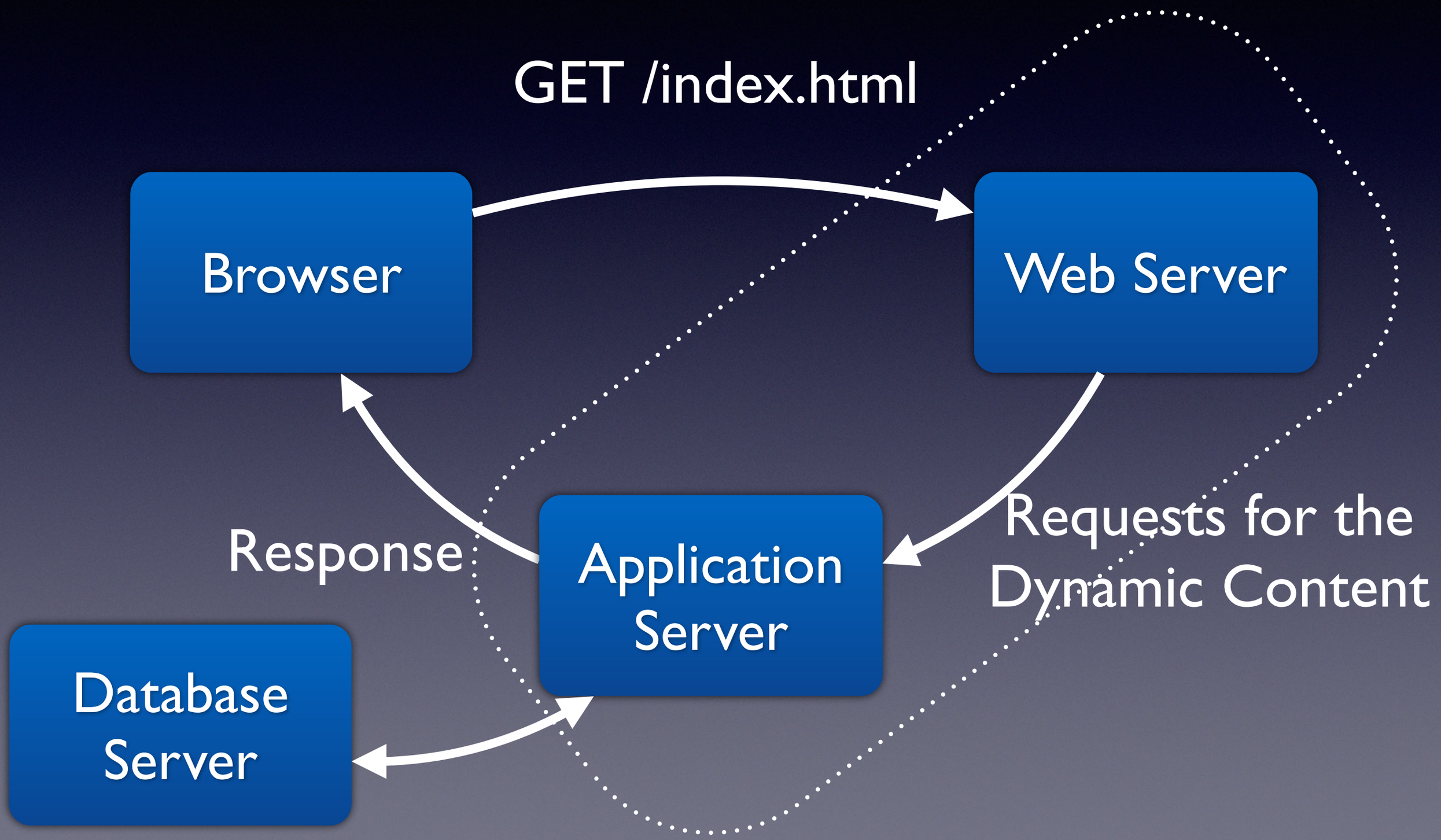
  Eclipse IDE for Java EE Developers
  (http://www.eclipse.org/downloads/)

  - We'll create a "Static Web Project"
  - We'll deploy it at HTTP Preview Server.

Dynamic Applications

- Interacting with a user: login, authorisation & etc.
- We're using it everyday's life:
  - Doing search (google.com | bing.com | yahoo.com)
  - Handling a session, shopping w/ basket (amazon.com)
  - Web Mails. (gmail | yahoo | hotmail | etc.)
  - Banking
  - and many more….

Dynamic Applications



JSP - Making it Dynamic

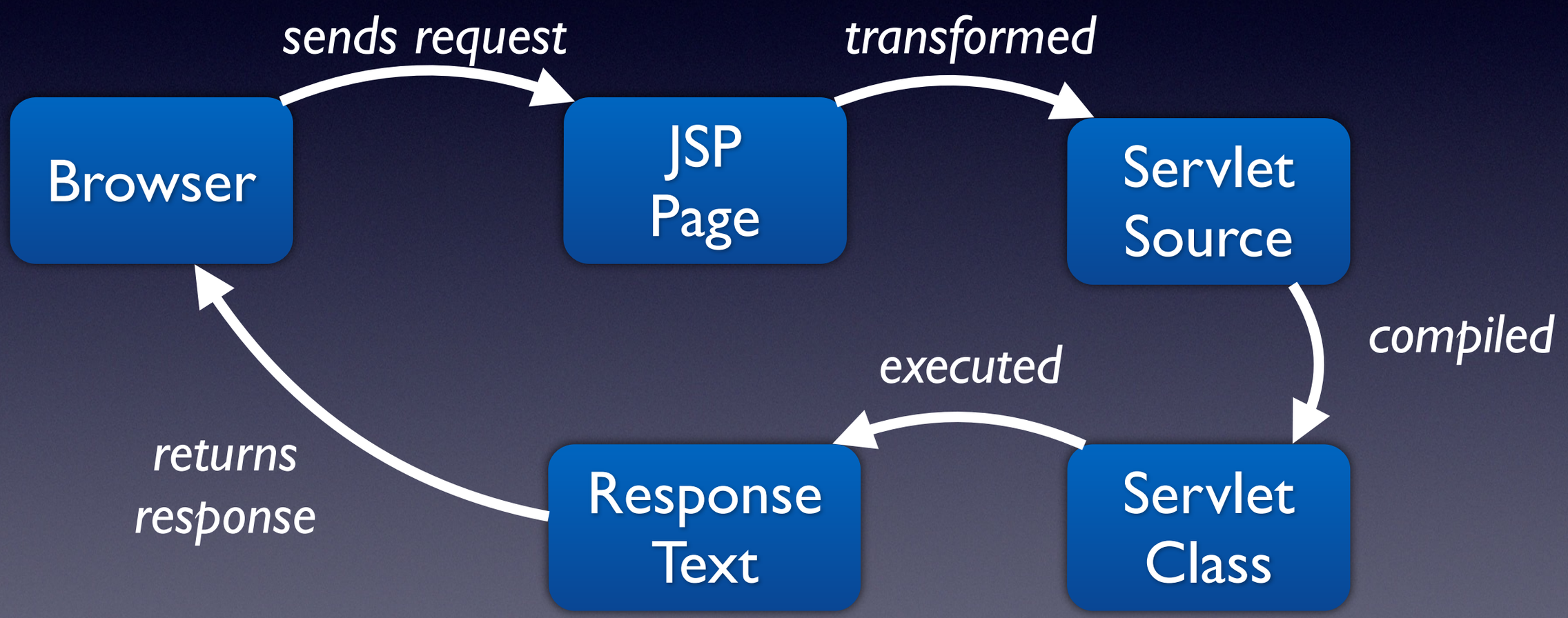
- JSP stands for Java Server Pages.
- Create dynamic Web Pages based on HTML.
- Each JSP gets translated into a Servlet. We'll cover servlets/filters soon.
- It enables to mix up Java code with HTML.
- So, to run JSP pages we need a Servlet container. We'll detail them also.

Simple JSP

```
<html>
    <body>
        <% out.write("Hello World!”);  %>
    </body>
</html>
```

- Here, JSP is the *Java inside HTML.*
- The Java statements are enclosed by <% ... %> (called JSP scriptlet).
- We can also use: <%= ... %> (called JSP expression).

Execution Flow



Predefined objects in JSP

- They are also called as implicit objects, which are provided by the JSP container.
- request (HttpServletRequest)
- response (HttpServletResponse)
- out (PrintWriter)
- application (ServletContext)
- config (ServletConfig)
- session (HttpSession)
- page (synonym for “this”, to call methods defined with the JSP page)

Let's create one…

- We'll be using Eclipse Kepler again.
- We'll create a "Dynamic Web Project".
- For Target Runtime we need to install a Web Server, it'll be Tomcat.
- You can download tomcat @ Tomcat 7.0.52 (http://tomcat.apache.org/download-70.cgi)
- Tomcat 7 aligns with Servlet 3.0 and 8.0 version aligns with Servlet 3.1. It's all about versions.

jsp:useBean

- The useBean action declares a JavaBean for use in a JSP. Once declared, the bean becomes a scripting variable that can be accessed by both scripting elements and other custom tags used in the JSP.

```
<jsp:useBean id="date" class="java.util.Date" />
<p>The date/time is <%= date %>
```

- useBean has the attribute scope. Values for the scope attribute could be <u>request</u>, <u>page</u>, <u>session</u> or <u>application.</u>

jsp:getProperty / jsp:setProperty

```
<jsp:useBean id="students"
                class="bbm490.StudentsBean">
   <jsp:setProperty name="students" property="firstName" value="Mert"/>
   <jsp:setProperty name="students" property="lastName"  value="Caliskan"/>
   <jsp:setProperty name="students" property="age" value="33"/>
</jsp:useBean>

<p>Student First Name:
    <jsp:getProperty name="students" property="firstName"/>
</p>

<p>Student Last Name:
    <jsp:getProperty name="students" property="lastName"/>
</p>

<p>Student Age:
    <jsp:getProperty name="students" property="age"/>
</p>
```

Custom Tag Library

<bbm:hello />

Custom Tag Library

- place taglib document under WEB-INF

```
<taglib>
   <tlib-version>1.0</tlib-version>
   <jsp-version>2.0</jsp-version>
   <short-name>Example TLD</short-name>
   <tag>
      <name>hello</name>
      <tag-class>bbm.HelloTag</tag-class>
      <body-content>empty</body-content>
   </tag>
</taglib>
```

Tag definition

```
public class HelloTag extends SimpleTagSupport {
    public void doTag() throws JspException, IOException {
        JspWriter out = getJspContext().getOut();
        out.println("Hello Custom Tag!");
    }
}
```

Using the tag

```
<%@ taglib prefix="bbm" uri="WEB-INF/myTaglib.tld"%>
<html>
    <body>
        <bbm:hello />
    </body>
</html>
```

Servlet

- They are server side components written in Java to extend the server side functionality.
- 1.0 version finalised in 1997 and now JavaEE7 brings the Servlet 3.1.
- Servlets are *HTML inside Java*.
- Once an instance gets created it is used for multiple requests.

Servlet Lifecycle

- It's the entire process from its creation till the destruction.
- Servlet initialised with by calling the `init()` method, gets called only once.
- `service()` method executed to process a client's request. It executes `doGet()` or `doPost()` methods or other http methods, `doXXX()`.
- The servlet is terminated by calling the `destroy()` method.

Defining a Servlet

```
@WebServlet(value="/myServlet")
public class MyServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
        resp.getWriter().write("Welcome to the land of Servlets");
    }

    @Override
    public void init() throws ServletException {
        System.out.print("Servlet gets initialised");
    }
}
```

Defining a Servlet with web.xml

```
<servlet>
   <servlet-name>MyServlet</servlet-name>
   <servlet-class>bbm490.MyServlet</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>MyServlet</servlet-name>
   <url-pattern>/myServlet</url-pattern>
</servlet-mapping>
```

- There could be multiple mappings. With annotation this could be also done with urlPatterns attribute.
- XML overrides the annotation definition.

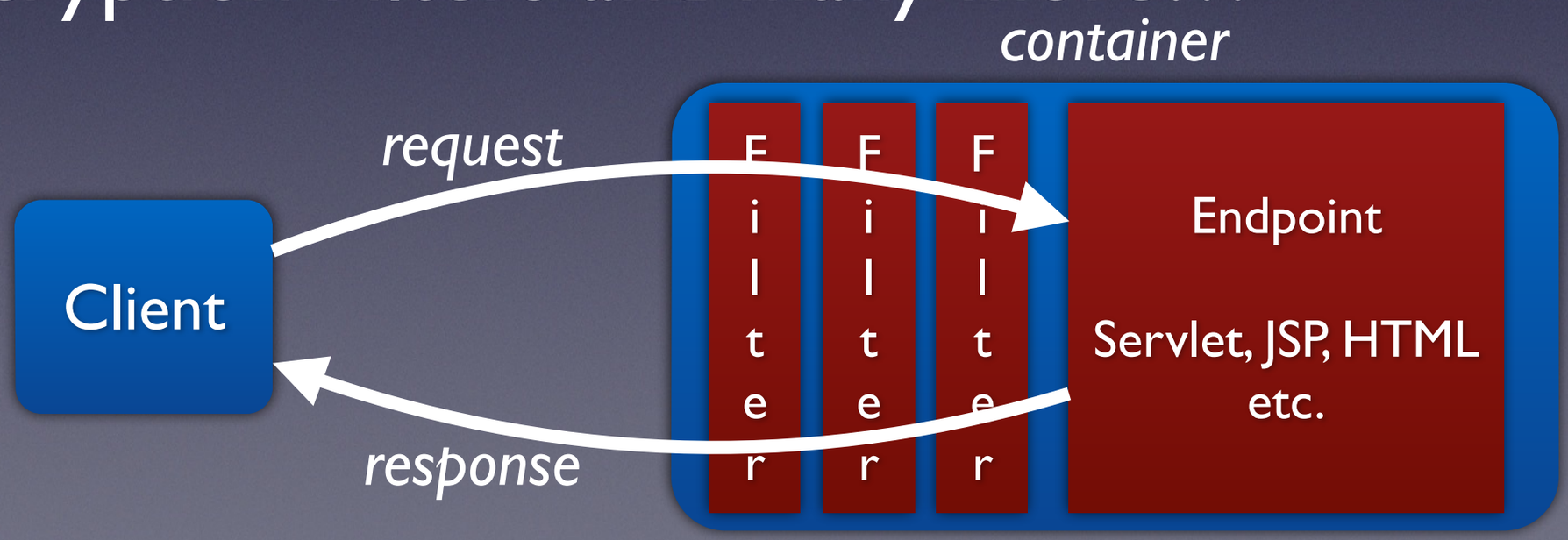
Reading form data in a Servlet

- Servlet provides the request.getParameter() method to get the value of a form parameter.
- Sample request: http://localhost:8080/DynamicWeb/myServlet?first=mert&last=caliskan

```
@Override
protected void doGet(HttpServletRequest req,
HttpServletResponse resp)
throws ServletException, IOException {
    resp.getWriter().write("Hello " + req.getParameter("first") + " "
+ req.getParameter("last"));
}
```

Filter

- A filter is a Java implementation that performs filtering tasks on either the request or the response.
- Introduced in Servlet 2.3 in 2001.
  Authentication Filters
  Logging and Auditing Filters
  Encryption Filters and many more…



Defining a Filter

```
@WebFilter(value="/myFilter")
public class MyFilter implements Filter {

    @Override
    public void destroy() {
    }

    @Override
    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain)
throws IOException, ServletException {
        resp.getWriter().write("in request");
        chain.doFilter(req, resp);
        resp.getWriter().write("in response");
    }

    @Override
    public void init(FilterConfig arg0) throws ServletException {
    }
}
```

Defining a Filter with web.xml

```
<filter>
   <filter-name>MyFilter</filter-name>
   <filter-class>bbm490.MyFilter</filter-class>
</filter>

<filter-mapping>
   <filter-name>MyFilter</filter-name
   <url-pattern>/myServlet</url-pattern>
</filter-mapping>
```

- XML definition overrides the annotation definition.

Order of Filters

```
<filter>
    <filter-name>filter1</filter-name>
    <filter-class>bbm490.FirstFilter</filter-class>
</filter>
<filter>
    <filter-name>filter2</filter-name>
    <filter-class>bbm490.SecondFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>filter2</filter-name>
    <url-pattern>/url/*</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>filter1</filter-name>
    <url-pattern>/url/*</url-pattern>
</filter-mapping>
```

Which Filter will get executed first for a request sent to /url?

Order of Filters

```
@Override
public void doFilter(ServletRequest req, ServletResponse resp,
FilterChain chain) throws IOException, ServletException {
    resp.getWriter().write("First Filter Request");
    chain.doFilter(req, resp);
    resp.getWriter().write("First Filter Response");
}
```

Filter1

```
@Override
public void doFilter(ServletRequest req, ServletResponse resp,
FilterChain chain) throws IOException, ServletException {
    resp.getWriter().write("Second Filter Request");
    chain.doFilter(req, resp);
    resp.getWriter().write("Second Filter Response");
}
```

Filter2

What will be the output???

Order of Filters

- The execution order of the doFilter methods will be as follows,
  - request - doFilter - filter2
  - request - doFilter - filter1
  - response - doFilter - filter1
  - response - doFilter - filter2

- We'll see the output like,

Second Filter Request / First Filter Request / First Filter Response / Second Filter Response

How can we define the order of the filters with @WebFilter definition?

Listener

- With Listeners we can get notified ourselves by,

  - Servlet Context initialised / destroyed.
  - Context attribute added / removed.
  - Request initialised / destroyed.
  - Request attribute added / removed.
  - Session created / destroyed.
  - Session attribute added / removed.

Listener

- Context Listener
  (javax.servlet.ServletContextListener)
- Context Attribute Listener
  (javax.servlet.ServletContextAttributeListener)
- Servlet Request Listener
  (javax.servlet.ServletRequestListener)
- Servlet Request Attribute Listener
  (javax.servlet.ServletRequestAttributeListener)
- Http Session Listener
  (javax.servlet.http.HttpSessionListener)
- Http Session Attribute Listener
  (javax.servlet.http.HttpSessionAttributeListener)

But first let's define some of the terms given above.. What's ServletContext for instance?

ServletContext

- When a web application gets started it will parse the web.xml file and every Servlet, Filter and Listener found in web.xml will be created for once and kept in server's memory.
- That’s where web application also creates an instance of ServletContext. There is one context per "web application" per JVM.
- ServletContext is passed to Servlets via ServletConfig instance.

ServletContext Attribute Listener

- You may want to store some attribute within the application scope that will be shared to all servlets via context.

```
getServletContext().setAttribute("myAttribute", myObject);
getServletContext().getAttribute("myAttribute");
```

There are also context parameters which are static. We'll get to that.

ServletRequest Listener

- Listener to track a request made to the web application by the client.

HttpSession Listener

- Listener to track the sessions created by the application.

Defining a Listener

```
@WebListener
public class ApplicationLifeCycleListener implements ServletContextListener {
    @Override
    public void contextDestroyed(ServletContextEvent arg0) {
        System.out.println("Context Destroyed");
    }

    @Override
    public void contextInitialized(ServletContextEvent arg0) {
        System.out.println("Context Created");
    }
}
```

Defining a Listener with web.xml

```
<listener>
    <listener-class>
        bbm490.ApplicationLifeCycleListener
    </listener-class>
</listener>
```

web.xml

- We used web.xml to define servlets, filters or listeners. What is it about? What else we can have?
- web.xml is the deployment descriptor for Servlet-based Web Applications.
- It's an XML file so where are the tags defined? any guesses?
- It resides under WEB-INF / web.xml.
- WEB-INF is the folder to store special configuration files like web.xml, tag library definitions and other resource files.

<context-param>

```
<context-param>
    <param-name>myParam</param-name>
    <param-value>paramValue</param-value>
</context-param>
```

to get the parameter from ServletContext:

```
String val =
    getServletContext().getInitParameter("myParam");
```

<display-name>

```
<display-name>
    Servlet 3.0 Web Application
</display-name>
```

- The optional display-name element specifies the Web application display name, a short name that can be displayed by GUI tools.

<welcome-file-list>

```
<welcome-file-list>
    <welcome-file>index.html</welcome-file>
</welcome-file-list>
```

- The welcome files mechanism allows you to specify a list of files that the web container will use for appending to a request for a URL that is not mapped to a web component.

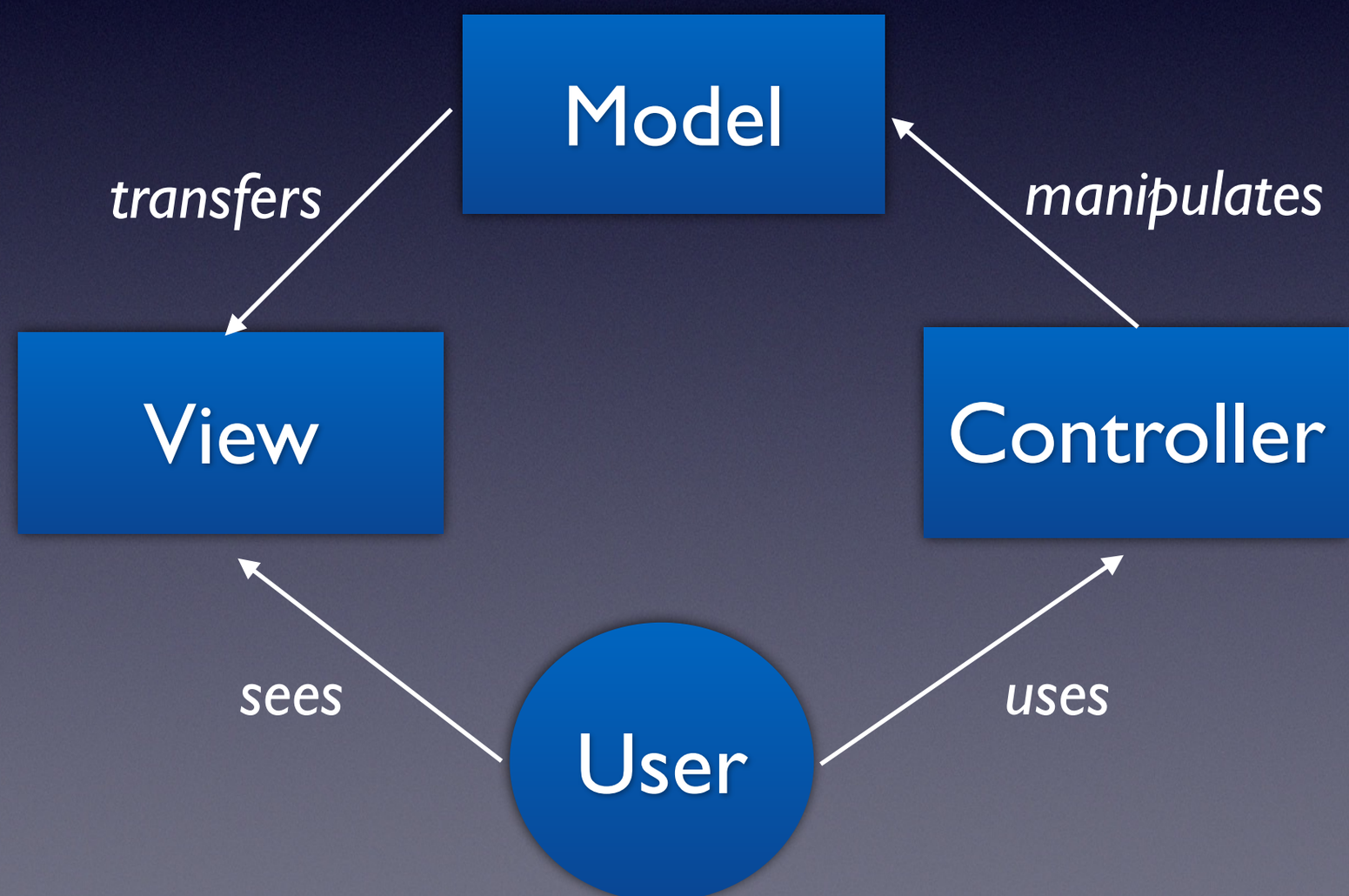
<session-config>

```
<session-config>
    <session-timeout>15</session-timeout>
</session-config>
```

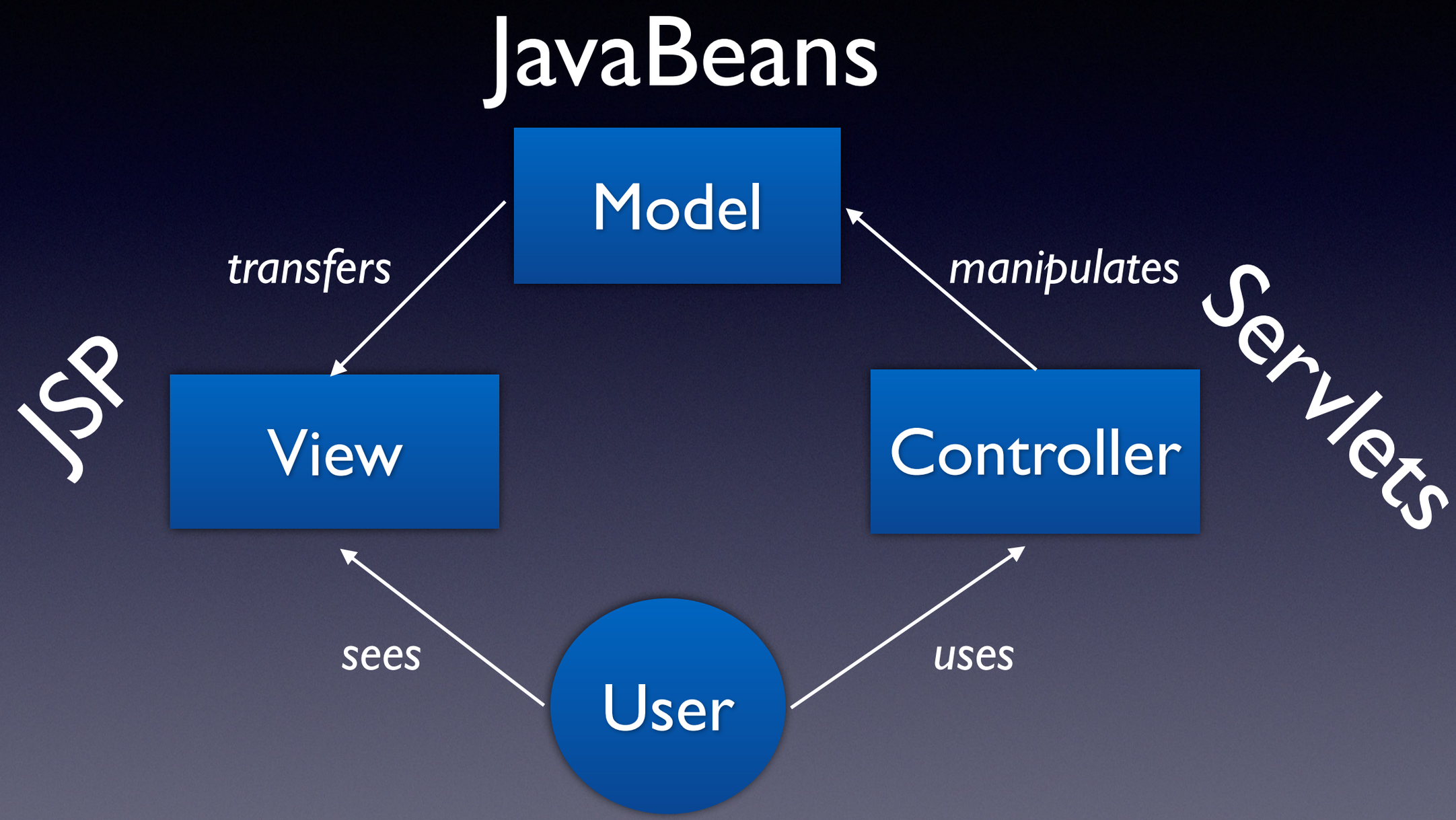
- Defines the session attributes for this Web application. session-timeout defines the number of minutes after which sessions in this Web application expire.

MVC

- States for Model View Controller
- It's a pattern.



MVC


JavaBeans
Model
transfers
manipulates
JSP
View
Controller
Servlets
sees
uses
User

MVC

- The Model is the part of the application that handles the logic for the application data.
  Often model objects retrieve data (and store data) from a database.

- The View is the parts of the application that handles the display of the data.
  Most often the views are created from the model data.

MVC

- The Controller is the part of the application that handles user interaction.
  Typically controllers read data from a view, control user input, and send input data to the model.

- MVC approach provides clean layered architecture.

Form Elements

<form>

- The <form> tag is used to create an HTML form for user input.
- All form elements must be inside a form element.
- `action` attribute specifies the URI for submitting.
- `method` attribute specifies the method for submitting the form values (method="GET|POST")

<form>

- Form can contain regular HTML markup and form elements.
- The regular HTML markup creates the form's layout (table, list, texts)
- Each form should have a submit button. When pressing this button, the form values are sent to the action URI. Without such a button, the form values cannot be submitted.

Let’s go through what can be defined within a <form>

<input> types

- Specifies an input field that user can enter data.
- Input elements should be defined in <form>
- Different types of input elements can be defined with the type attribute.
- With HTML4.0.1 there are 10 types. Let's go through them.

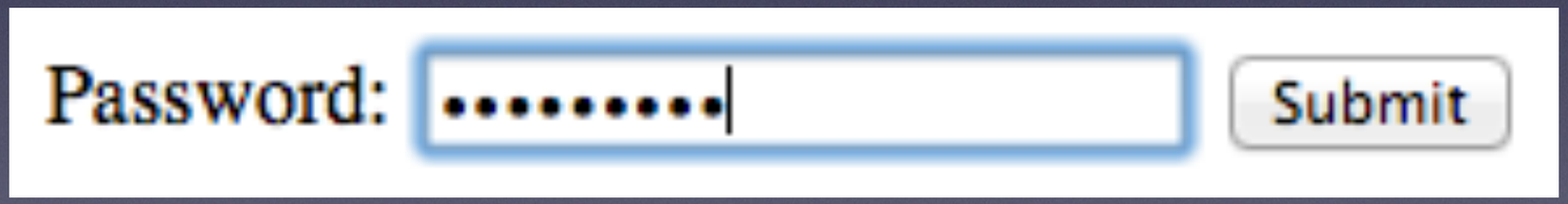
TEXT | PASSWORD | CHECKBOX | RADIO | SUBMIT | RESET | FILE | HIDDEN | IMAGE | BUTTON

<input type=“text” />

```
<form action=“/myForm“>
Username: <input type="text" name="username" />
<input type="submit" />
</form>
```

<input type=“password” />

```
<form action=“/myForm“>
  Password: <input type="password" name="password" />
  <input type="submit" />
</form>
```



<input type="checkbox" />

- checkboxes are on/off switches that may be toggled by the user.
- checked="true" attribute will make it selected.

```
<form action="/myForm">
  Selection:
  <input type="checkbox" name="attending" value="Y">Attending</input>
  <input type="submit" />
</form>
```

Selection: ☐ Attending Submit

<input type=“radio” />

- Same as checkbox. User toggles on and off.

```
<form action=“/myForm">
  Selection:
  <input type="radio" name="dummy" value="D">Dummy</input>
  <input type="submit" />
</form>
```

Selection: Dummy Submit

<input type=“radio” />

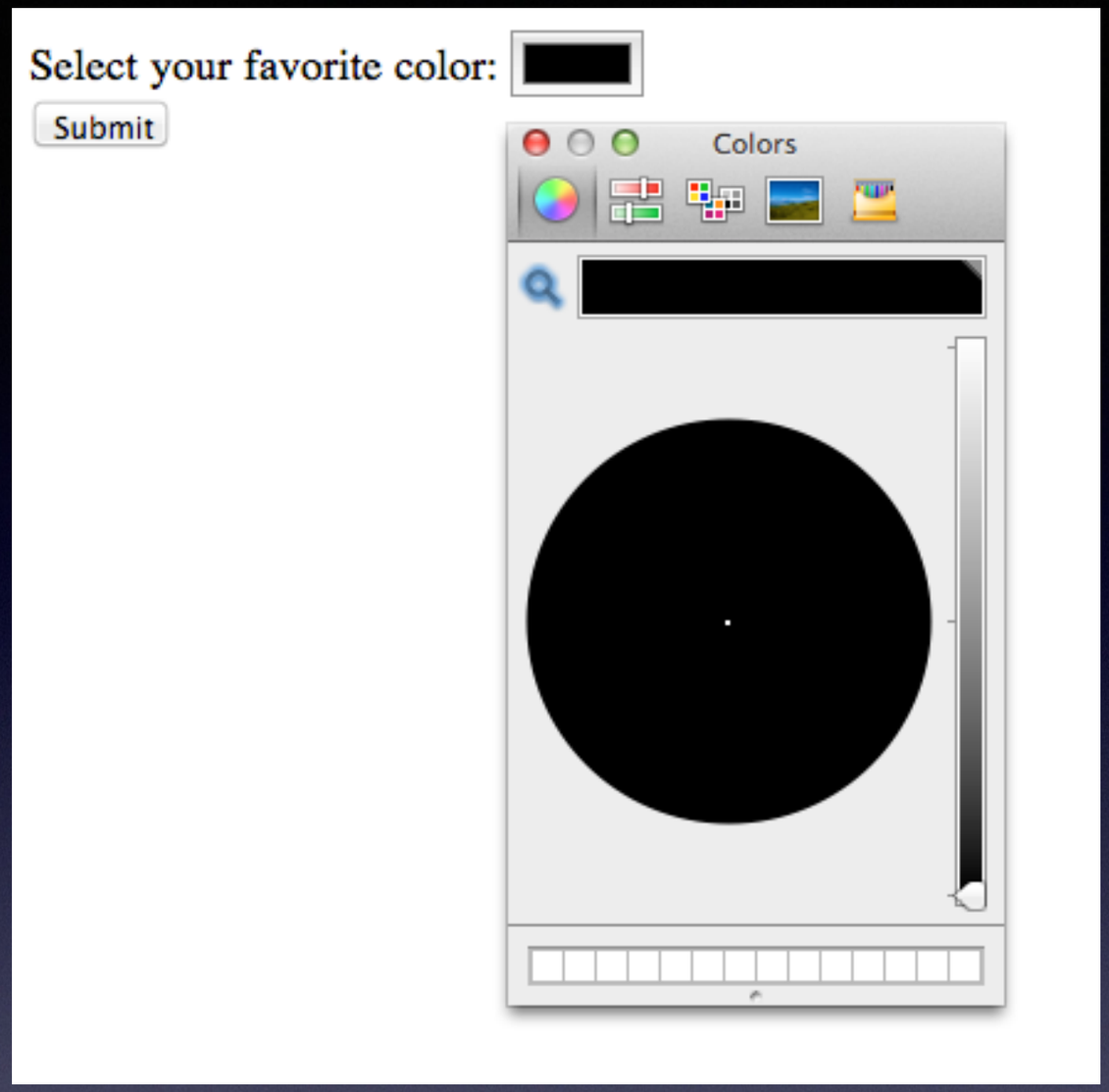
- While they are sharing the same name, when one is switched “on", all others with the same name are switched “off”.

```
<form action=“/myForm">
  Selection:
  <input type="radio" name=“selection" value=“F”>Female</input>
  <input type="radio" name=“selection" value=“M”>Male</input>
  <input type="submit" />
</form>
```

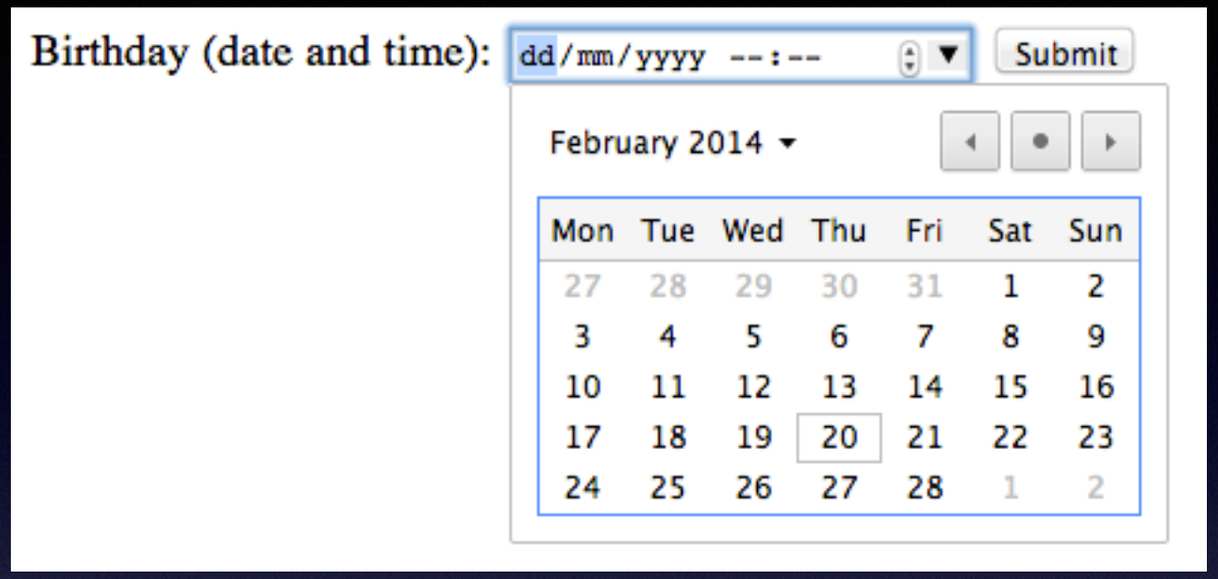
HTML5 - <input> types

- HTML5 introduces new input types like,

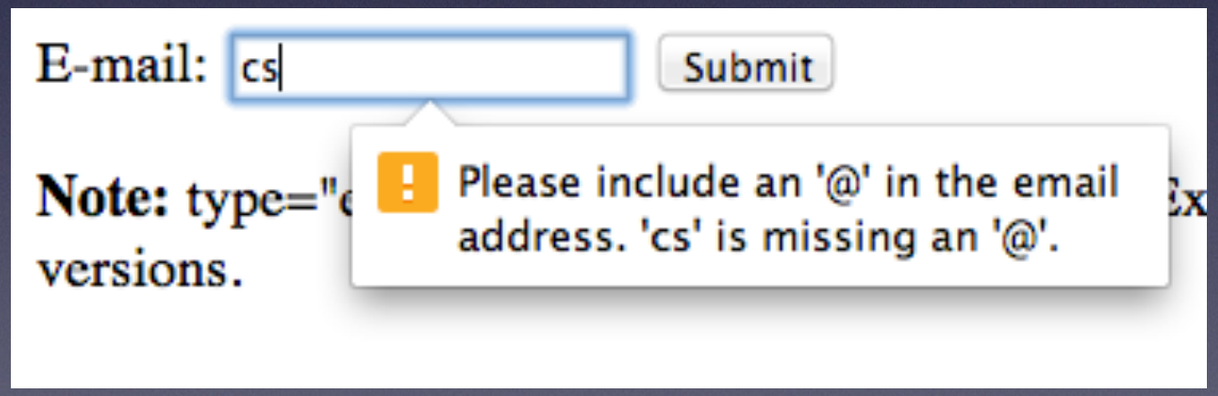
color
date
datetime
datetime-local
email
month
number
range
search
tel
time
url
week



type="color"



type="datetime-local"



type="email"

type="range"

<textarea>

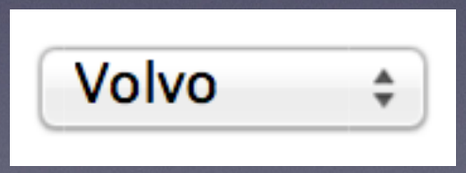
- The <textarea> tag defines a multi-line text input control.

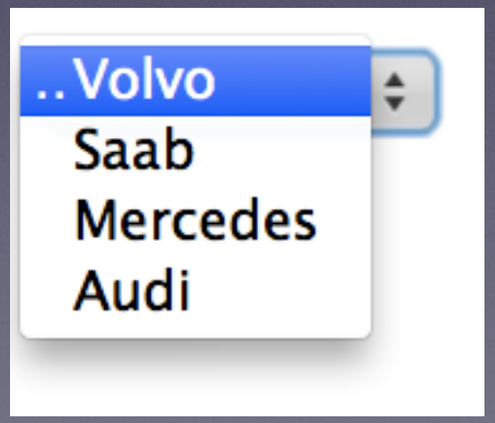
```
<textarea rows="4" cols="50">
    BBM490 - Enterprise Web Architecture
</textarea>
```

<select>

- The <select> element is used to create a drop-down list.

```
<select name="car">
   <option value="volvo">Volvo</option>
   <option value="saab">Saab</option>
   <option value="mercedes">Mercedes</option>
   <option value="audi">Audi</option>
</select>
```





<button>

- Buttons created with the <button> element function just like buttons created with the <input> element, but they offer richer rendering possibilities: the <button> element may have content, like HTML.

```
<button>
   <img src="/images/icons/tick.png" alt=""/>
    Save
</button>
```

Form - GET

- Parameters remain in browser history because they are part of request URL.
- Can be bookmarked.
- Not suitable for uploading files.
- Only ASCII characters allowed for data type.
- Less secure - data sent through URL.
- Restriction on data length since length of URL is limited to browser and server.
- Suitable for idempotent forms, like read from DB.

Form - POST

- Parameters are not saved in browser history.
- Can not be bookmarked.
- Also suitable for uploading files.
- No restriction on data type, could be ASCII or binary.
- No restriction on data length.
- Much safer than GET.
- Suitable for forms that modifies data, like from DB.

Let's create an App

- A JSP page with a form.
- Form will submit to a Servlet.
- And Servlet will render the output according to the given values.

So question is:

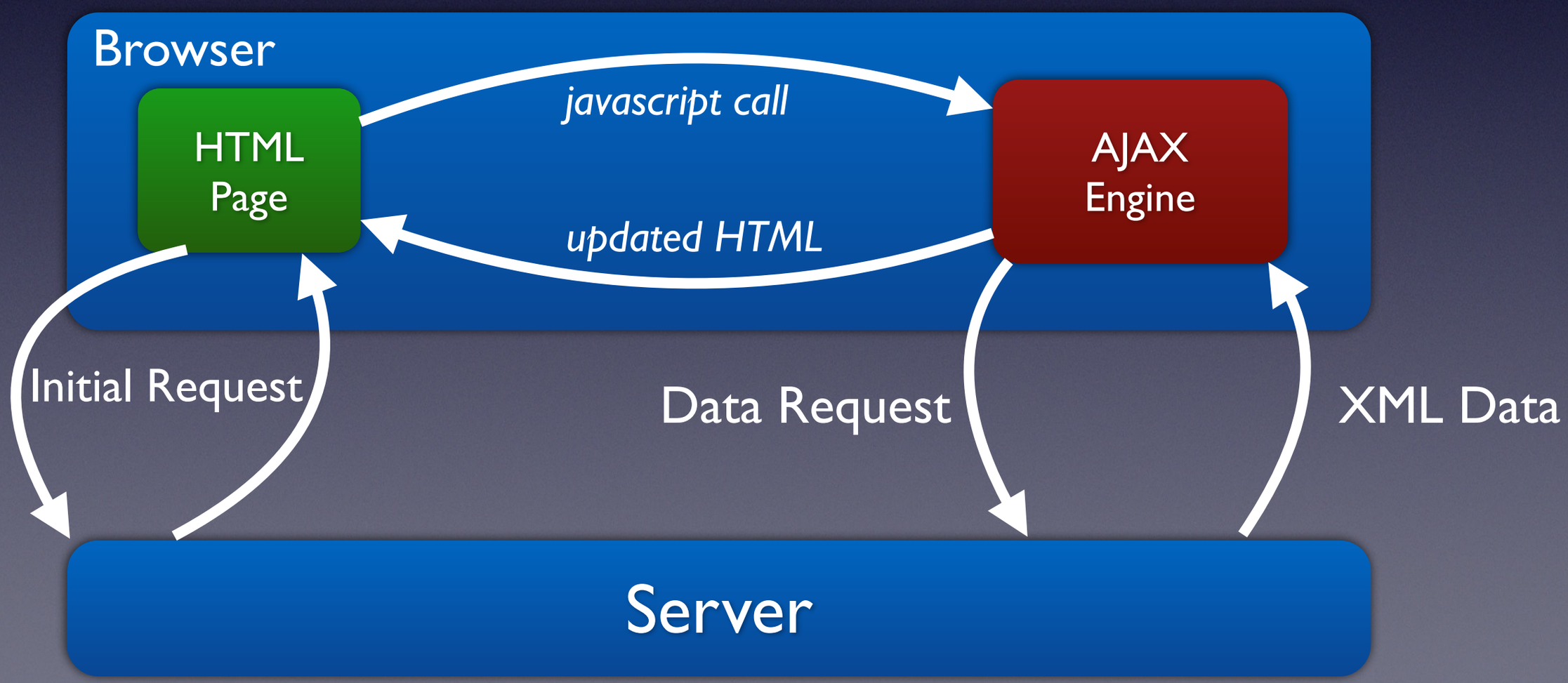
What is AJAX?

AJAX

- States for Asynchronous JavaScript and XML.
- It's a web development technique used on the client-side to create asynchronous web applications.
- Before Ajax, whole pages get submitted to the server and gets back from. This was an inefficient way.
- <iframe> introduced by IE to do async calls.

AJAX

- Client side make server side calls by partially submitting the page (specific form elements) and gets partial data from server to update the page partially.



Drawbacks of AJAX

- Pre HTML5 browsers do not support back button. So browser history will be problematic.
- Bookmarking gets difficult since pages get dynamically created.
- Ajax pages are mostly not search engine friendly since indexing javascript is hard for engines.
- Users can interact with bad UI experience on autocompletes and other fields when on slow internet connection.

AJAX

- For the last years AJAX is: Asynchronous JavaScript and JSON.
- Instead of XML, JSON is going back and forth between client and Server.
- JSON stands for JavaScript Object Notation.
  - It's human readable, attribute - value pairs.

Sample JSON

```
{
    "firstName": "John",
    "lastName": "Smith",
    "age": 25,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumbers": [
        {
            "type": "home",
            "number": "212 555-1234"
        },
        {
            "type": "fax",
            "number": "646 555-4567"
        }
    ]
}
```

Let's create an AJAXified App

- There will be an HTML page, simple one. One button and one .
- On click of the button javascript will get executed and make a call to a servlet.
- Servlet will just write the current date to the output.
- The javascript will get the result and put it to the content of the div.

```
<!DOCTYPE html>
<html>
    <head>
        <script src="http://code.jquery.com/jquery-latest.min.js"></script>
        <script>
            $(document).ready(function() {
                $('#myButton').click(function() {
                    $.get('myServlet', function(responseText) {
                        $(‘#myDiv').text(responseText);
                    });
                });
            });
        </script>
    </head>
    <body>
        <input type="button" id="myButton">Press</input>
        <div id="myDiv"></div>
    </body>
</html>
```

```
@WebServlet(value="/myServlet")
public class AjaxServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req,
        HttpServletResponse resp) throws ServletException, IOException {
        resp.setContentType("text/plain");
        resp.getWriter().write(new Date().toString());
    }
}
```

jQuery is…

- Write Less, Do More. That's what jQuery is…
- jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.

jQuery offers…

- Dom Traversal and Manipulation.
  Find the with id ‘myDiv’ and change its text.
  $(‘#myDiv’).text(“Hello World”);
- Event Handling.
  Execute alert javascript on click of the button that has the id ‘myButton’

```
$('#myButton').click(function() {
   alert(“ok”);
});
```

- Provides mechanisms for Ajax and event handling.