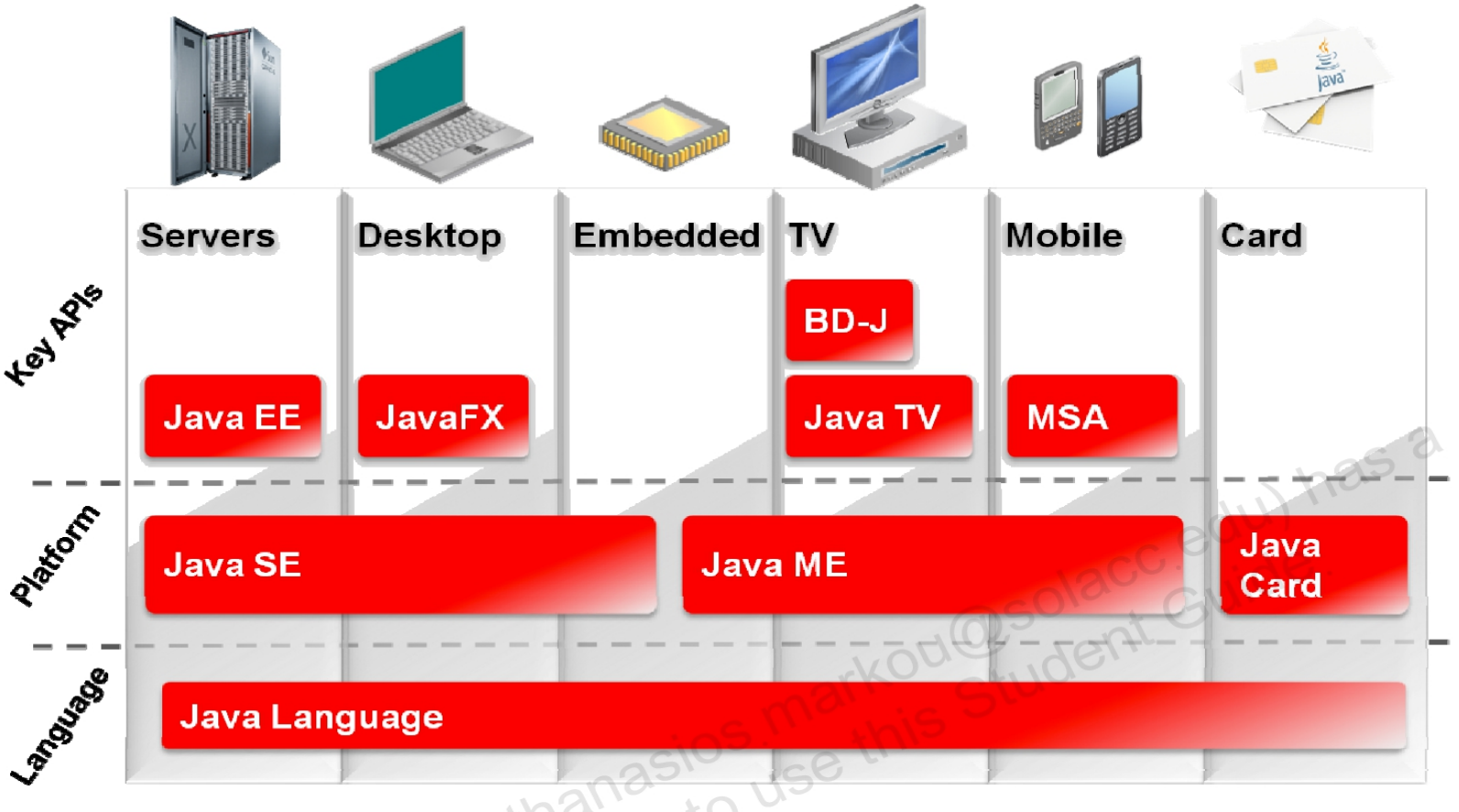
Introduction to Web App Dev Using Java EE

Java Technology Product Groups

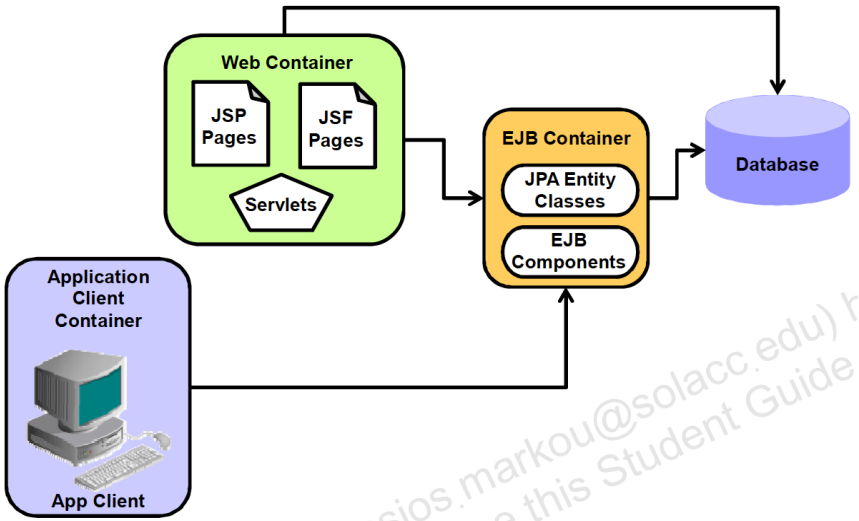


The Java EE platform

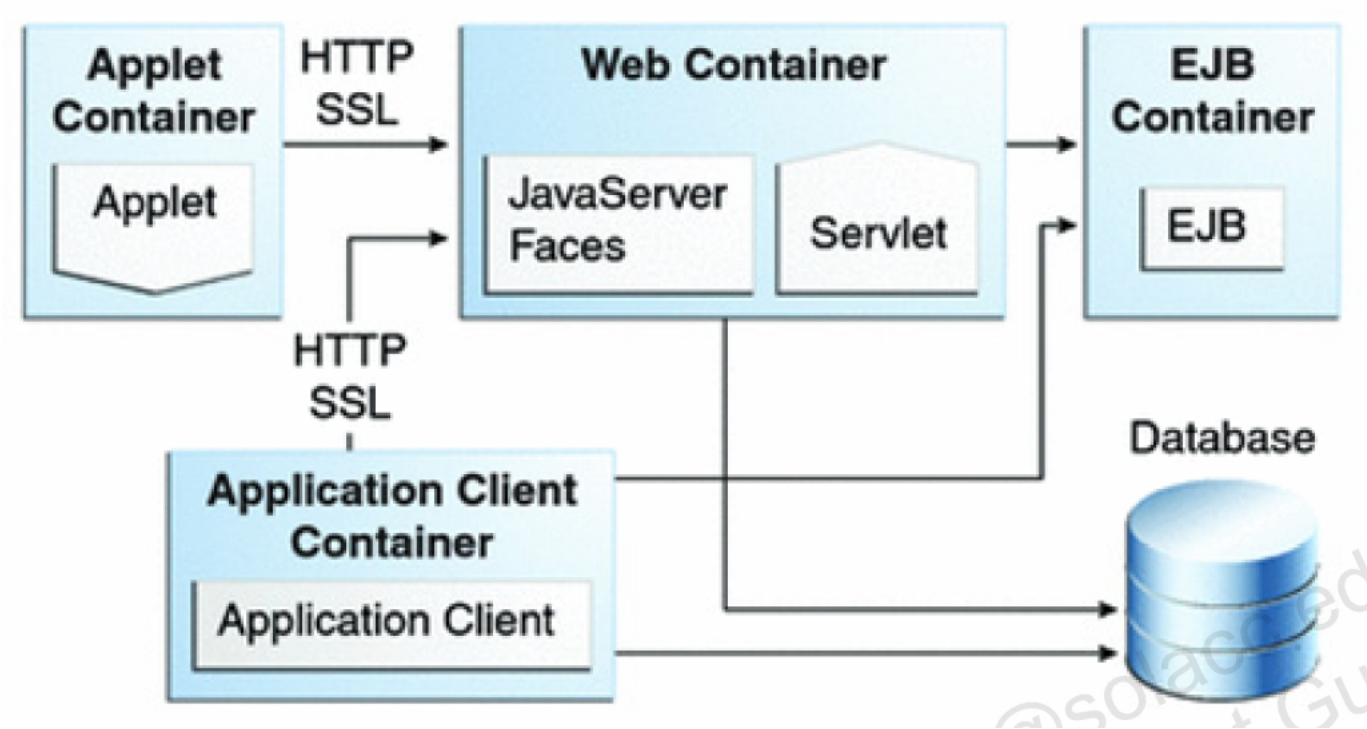
- Is an architecture for implementing enterprise-class applications
- Uses Java and Internet technology
- Has a primary goal of simplifying the development of enterprise-class applications through an application model that is:
  - Vendor neutral
  - Component based

Java EE Containers

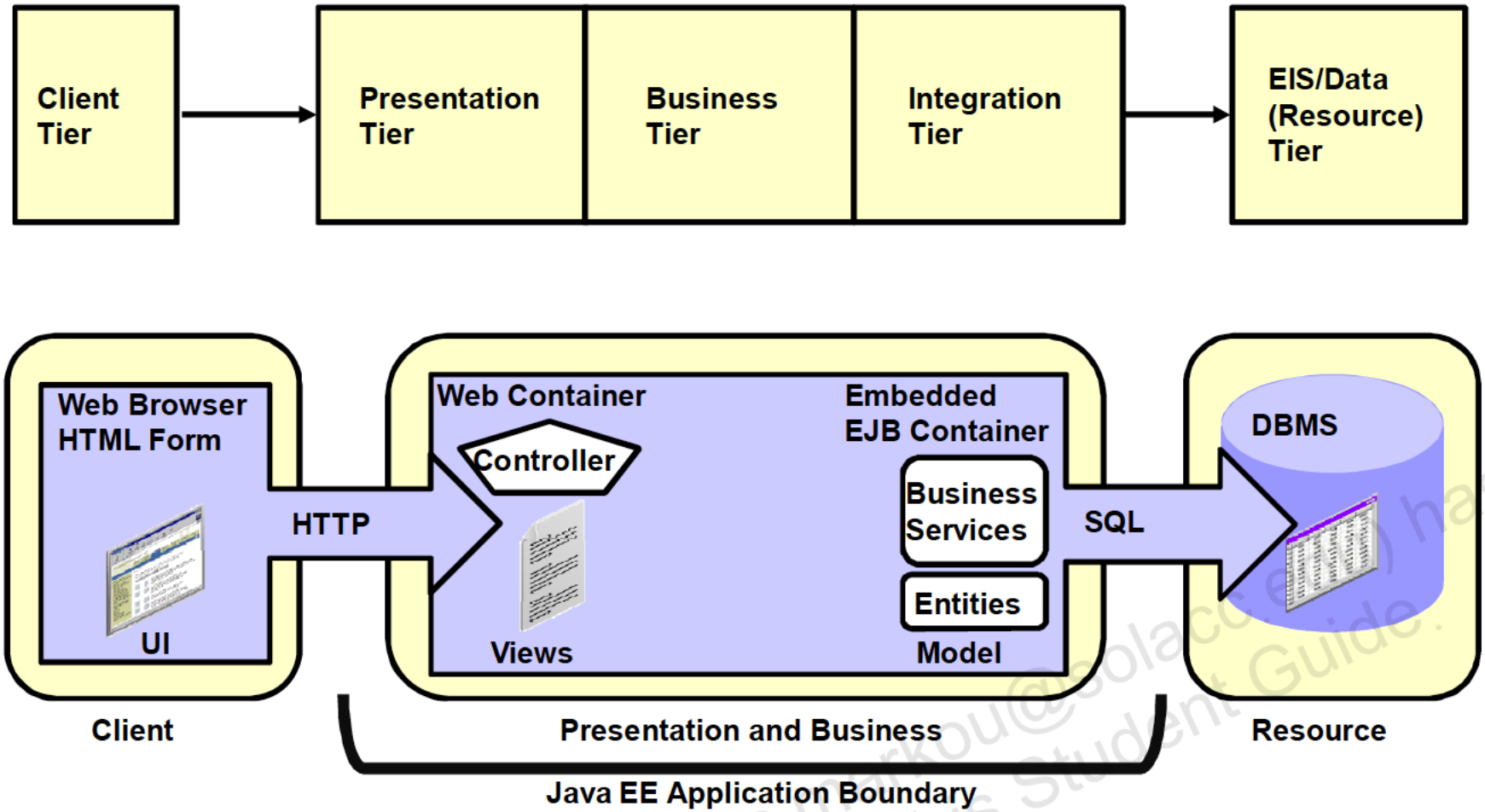
- Java EE application components (JSF) never interact directly with other Java EE application components but use methods and services of the **container** to interact.
- Java EE services allows the container to transparently inject the services required by the component, such as declarative transaction management, security checks, resource pooling, and state management.



Java EE Component Containers



Java EE Tiered Architecture

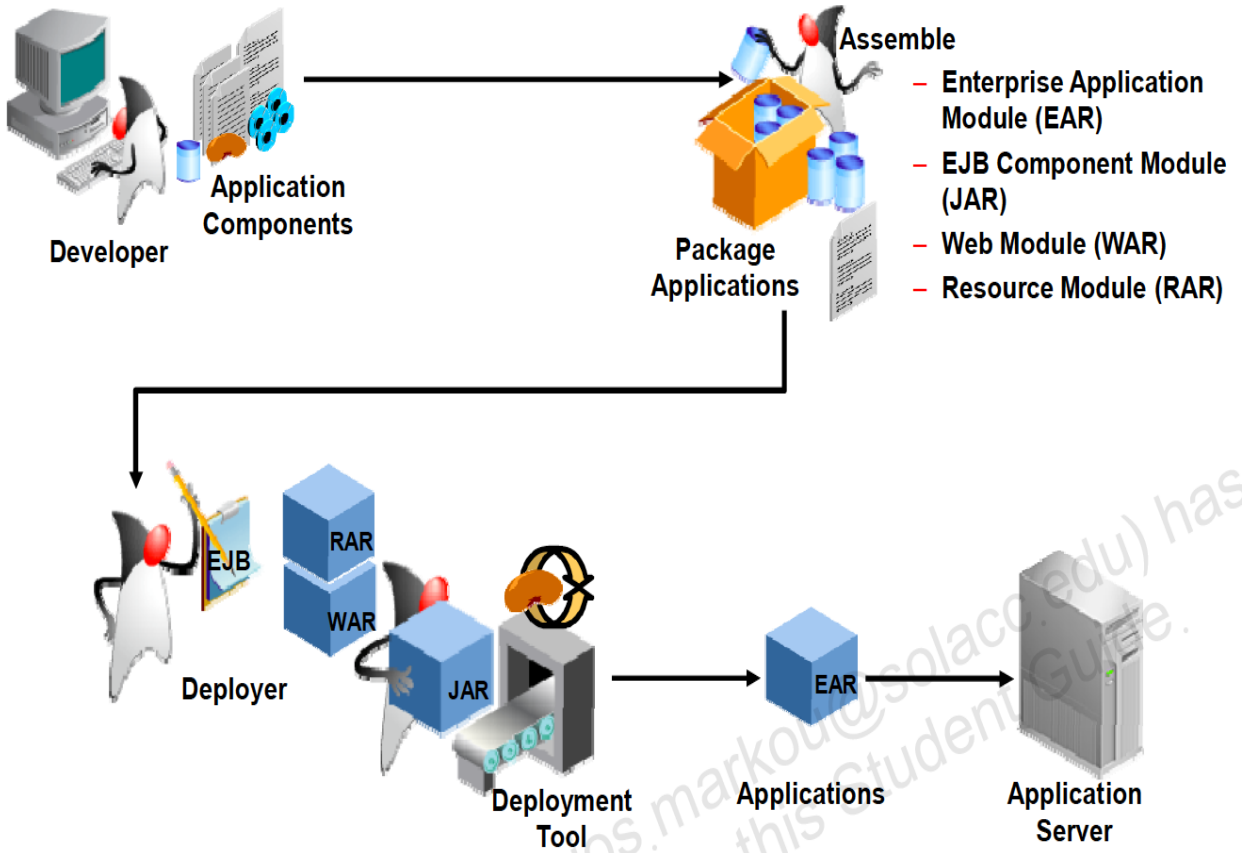


•Java EE Application Servers

- Java EE application server implementations.
  - Oracle Fusion Middleware
  - GlassFish (Oracle)
  - WebLogic (Oracle)
  - IBM WebSphere
  - Apache TomEE
  - JBoss Application Server
  - and many more…

Steps for Developing a Java EE Application

1. Designing
2. Coding
3. Creating deployment descriptors
4. Packaging
5. Assembly
6. Deployment



Why Create Web Applications

Desktop Applications

- AWT, Swing, Java FX
- Downloaded or installed on the client
- Run on the client
- Requires a JRE
- 3D and 2D better
- Updates must be installed on all clients

Web Applications

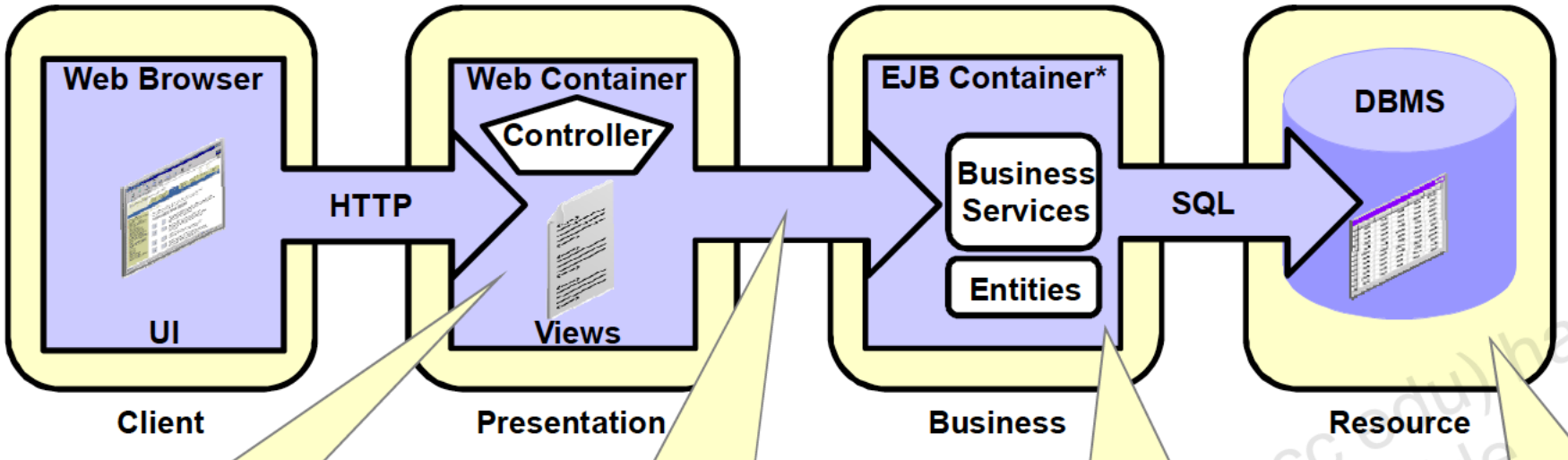
- Servlets, JSP, JSF
- Viewed (rendered) on the client
- Run on a server
- Requires a web browser
- 3D and 2D limited graphic support
- Requires a network connection

•Java standards for developing web applications

1. JavaServer Faces (JSF)
2. JavaServer Pages (JSP)
3. Servlets

- Require a web server, known as a web container, which supports running Java code

The Role of Web Components in a Java EE Application



Web Components focus on processing HTTP requests and responses. Business logic should not be implemented here.

Web Components can talk to the business layer using local method calls, distributed calls for remote EJBs, or HTTP for web services.

Business logic can be implemented using EJBs or POJOs.

Changes here, such as switching from file-based storage to a relational DB only require changes in the adjacent layer.

MVC Architecture

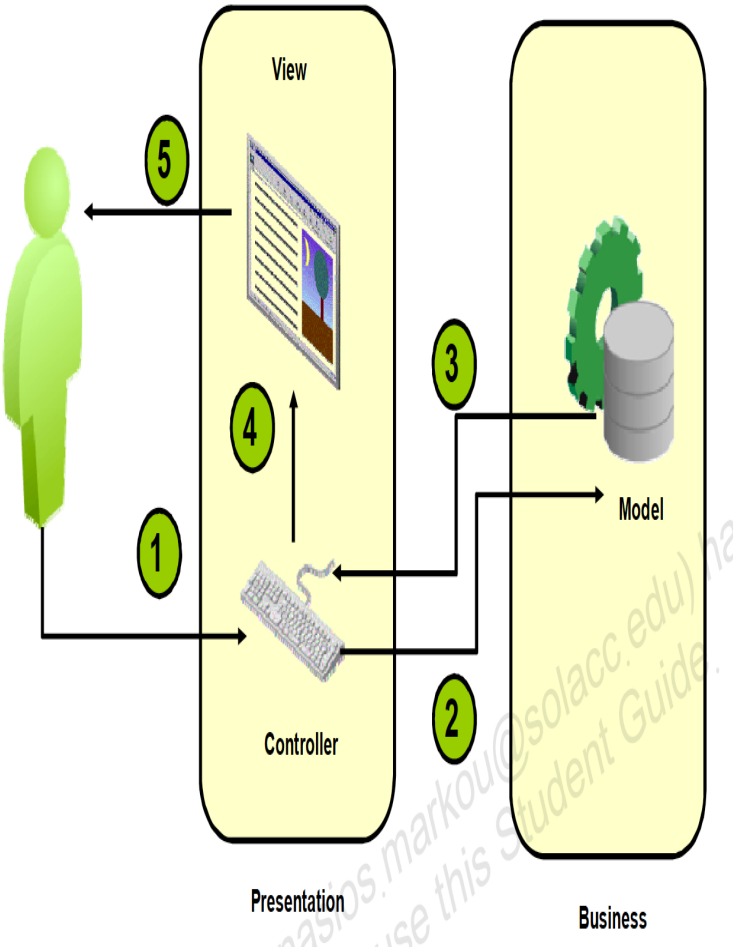
1. A **controller** component processes user input.

2. After performing input validation and conversion, a controller invokes business methods in a service component. The **controller** converts user interactions, such as a button click, into business method calls.

3. If a user is attempting to look at some type of information (for example, view a bank account balance), the service method called by the **controller** will return data objects from the model.

4. Based on the user input received in step 1 and the data returned in step 3, the **controller** will decide what screen or view the user should be shown.

5. The **view component** takes any available model data and creates an organized (visual) representation of it to display to the user.



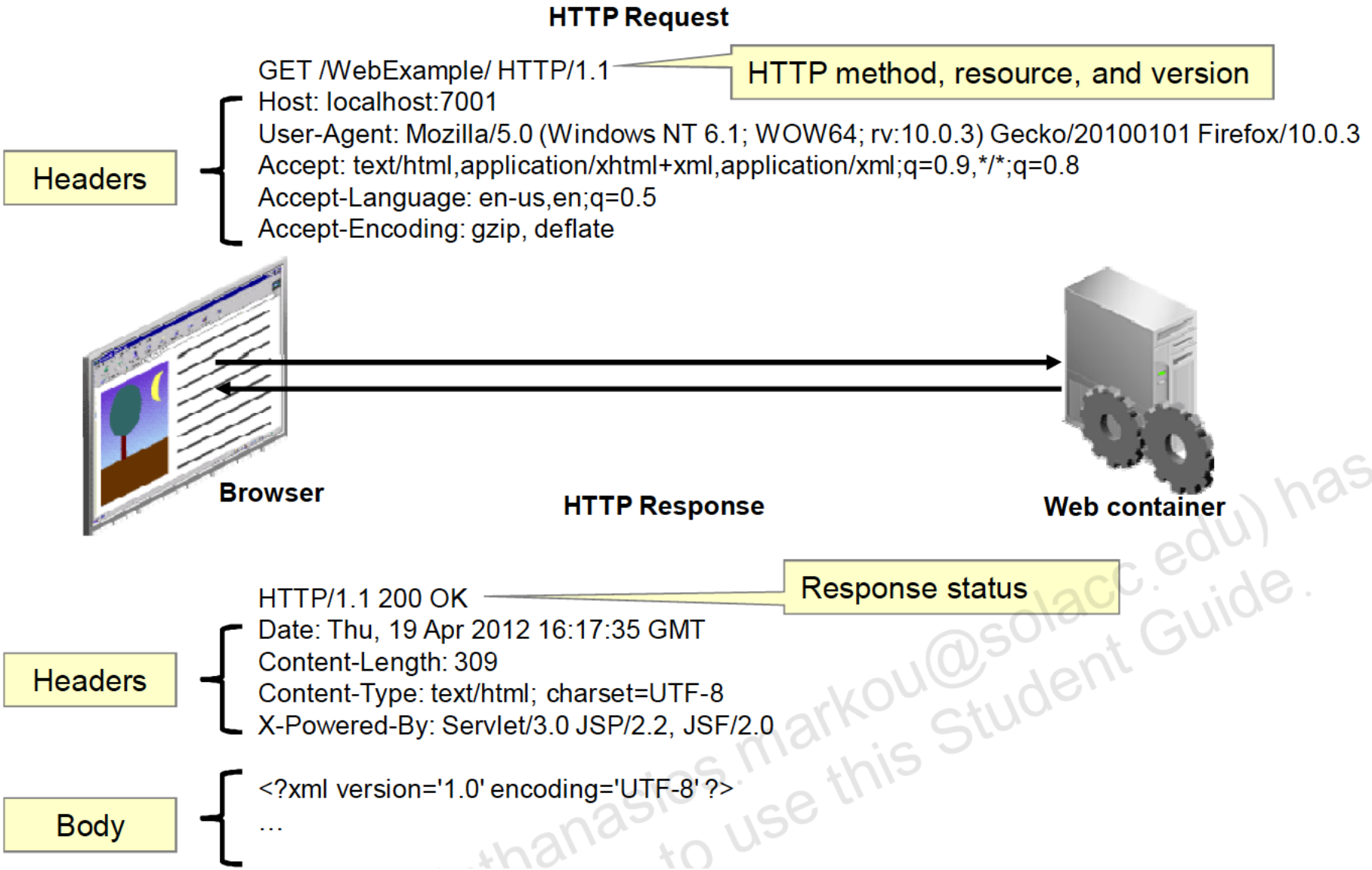
MVC in a Java EE Web Architecture

- Controller
  - Servlets
  - JSF Backing Beans
- View
  - JSPs
  - JSF Facelet Pages
- Model
  - POJOs
  - EJBs

Overview of Web Components

	JSPs & Servlets	JavaServer Faces
Description	Multi-threaded request response–oriented components	UI Component model similar to Swing
Characteristics	Direct manipulation of HTTP headers and HTML content. You must manually handle all form data (all parameters are strings). HTML is directly produced by Servlet or JSP components.	Little to no HTTP manipulation by developers. HTTP requests are converted to higher-level actions. Form parameters are automatically read by JSF, converted, and passed into setter methods of your choice. UI Components use a RenderKit to produce HTML.
Benefits	Greater control over HTML content, requires more knowledge of HTML; works with WYSIWYG HTML tools	HTML is generated on behalf of a component requiring less HTML manipulation. UIs can be rapidly created when using the built-in components.

HTTP Request-Response Model



GET and POST Requests

The HTTP specification defines 7 types of requests:
GET, POST, PUT, TRACE, DELETE, HEAD, OPTIONS.

In practice, most web applications use only the GET and POST requests.

	GET Request	POST Request
Type of Use	Default	Form submission
Method of Sending Form Data	• Sent with the URI • Size limited (8K)	• Sent in the request body • Size unlimited (2G)
Benefits and Drawbacks	• Form data viewable in the browser's address bar • Form can be resubmitted with a bookmark	• Form data is not displayed in the browser's address bar. • Form cannot be resubmitted with a bookmark

What is JavaServer Faces?

JavaServer Faces define:

- a component architecture
- a standard set of UI widgets
- an application infrastructure

They keep your UI components in sync with Java objects which are called **backing beans** that collect user input values and respond to events.

Difference of Faces Programs versus Desktop Java Programs

JavaServer Faces and desktop UI frameworks like Swing, or JavaFX is that **Faces run on a server** such as GlassFish, Apache TomCat, WebSphere etc.

Data Scopes

Web applications support storing data in memory based on the lifetime and isolation of the data. Common scopes are:

- Request
- Session
- Application

All web components exist or access data in these scopes:

- Servlets use method calls such as request.getAttribute.
- JSPs use the Expression Language.
- JSF Facelets use the Expression Language.
- JSF Managed Beans are placed in these scopes using annotations.

Facelets

What is Facelets

- Facelets is a powerful but lightweight page declaration language that is used to build JavaServer Faces views using HTML style templates and to build component trees. Facelets features include the following:
  - Use of XHTML for creating web pages
  - Support for Facelets tag libraries in addition to JavaServer Faces and JSTL tag libraries
  - Support for the Expression Language (EL)
  - Templating for components and pages

Tag Libraries Supported by Facelets

Table 8-1 Tag Libraries Supported by Facelets

Tag Library	URI	Prefix	Example	Contents
JavaServer Faces Facelets Tag Library	`http://xmlns.jcp.org/jsf/facelets`	`ui:`	`ui:component` `ui:insert`	Tags for templating
JavaServer Faces HTML Tag Library	`http://xmlns.jcp.org/jsf/html`	`h:`	`h:head` `h:body` `h:outputText` `h:inputText`	JavaServer Faces component tags for all `UIComponent` objects
JavaServer Faces Core Tag Library	`http://xmlns.jcp.org/jsf/core`	`f:`	`f:actionListener` `f:attribute`	Tags for JavaServer Faces custom actions that are independent of any particular render kit
Pass-through Elements Tag Library	`http://xmlns.jcp.org/jsf`	`jsf:`	`jsf:id`	Tags to support HTML5-friendly markup
Pass-through Attributes Tag Library	`http://xmlns.jcp.org/jsf/passthrough`	`p:`	`p:type`	Tags to support HTML5-friendly markup
Composite Component Tag Library	`http://xmlns.jcp.org/jsf/composite`	`cc:`	`cc:interface`	Tags to support composite components
JSTL Core Tag Library	`http://xmlns.jcp.org/jsp/jstl/core`	`c:`	`c:forEach` `c:catch`	JSTL 1.2 Core Tags
JSTL Functions Tag Library	`http://xmlns.jcp.org/jsp/jstl/functions`	`fn:`	`fn:toUpperCase` `fn:toLowerCase`	JSTL 1.2 Functions Tags

The Lifecycle of a Facelets Application

1. When a client, such as a browser, makes a new request to a page that is created using Facelets, a new component tree or **javax.faces.component.UIViewRoot** is created and placed in the FacesContext.
2. The UIViewRoot is applied to the Facelets, and the view is populated with components for rendering.
3. The newly built view is **rendered** back as a response to the client.
4. On rendering, the **state of this view is stored** for the next request. The state of input components and form data is stored.
5. The client may **interact** with the view and request another view or change, from the JavaServer Faces application. At this time, the saved view is restored from the stored state.
6. The restored view is once again passed through the JavaServer Faces lifecycle, which eventually will either **generate a new view or re-render the current view** if there were no validation problems and no action was triggered.
    1. If the same view is requested, the stored view is rendered once again.
    2. If a new view is requested, then the process described in Step 2 is continued.

What EL is used for?

- To dynamically read application data stored in JavaBeans components, various data structures, and implicit objects
- To dynamically write data, such as user input into forms, to JavaBeans components
- To invoke any static or public methods
- To dynamically perform arithmetic, boolean and string operations
- To dynamically construct collection objects and perform operations on collections

EL immediate evaluation

- All expressions using the ${} syntax are evaluated immediately.

  <h:outputText value="${catalog.bookQuantity}" />

- The JavaServer Faces implementation evaluates the expression ${catalog.bookQuantity}, converts it, and passes the returned value to the tag handler. The value is updated on the page.

EL Deferred evaluation

- #{expr} can be evaluated at later phases of a page lifecycle

  <h:inputText id="name" value="#{customer.name}" />

- **For an initial request** the JavaServer Faces evaluates the #{customer.name} expression during the render-response phase. It accesses the value of **name** from the customer bean, as is done in immediate evaluation.

- **For a postback request**, evaluates the expression at different phases of the lifecycle, during which the value is retrieved from the request, validated, and propagated to the customer bean.