Hugbúnaðarverkefni 1 / Software Project 1

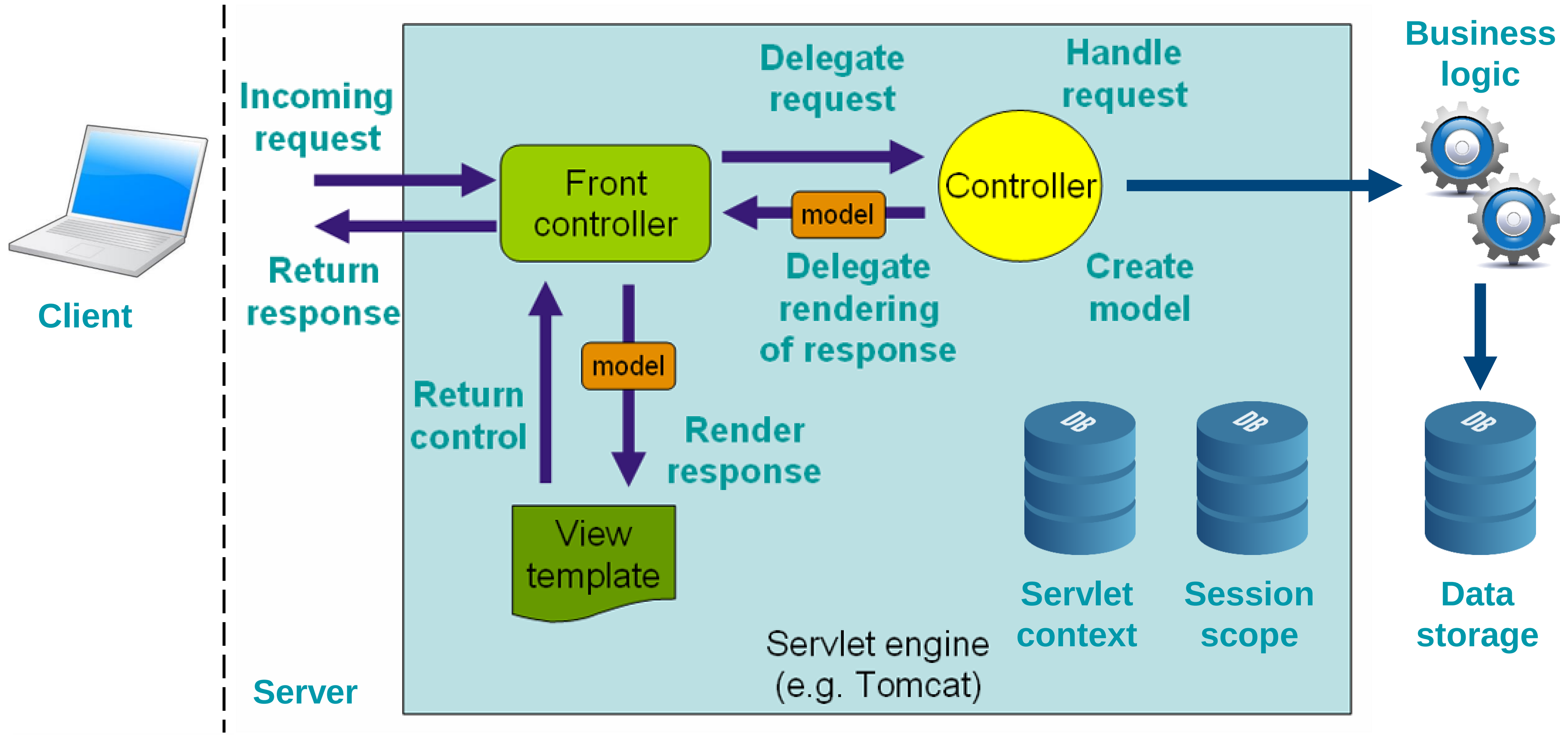
7. Presentation Layer

HBV501G – Fall 2018

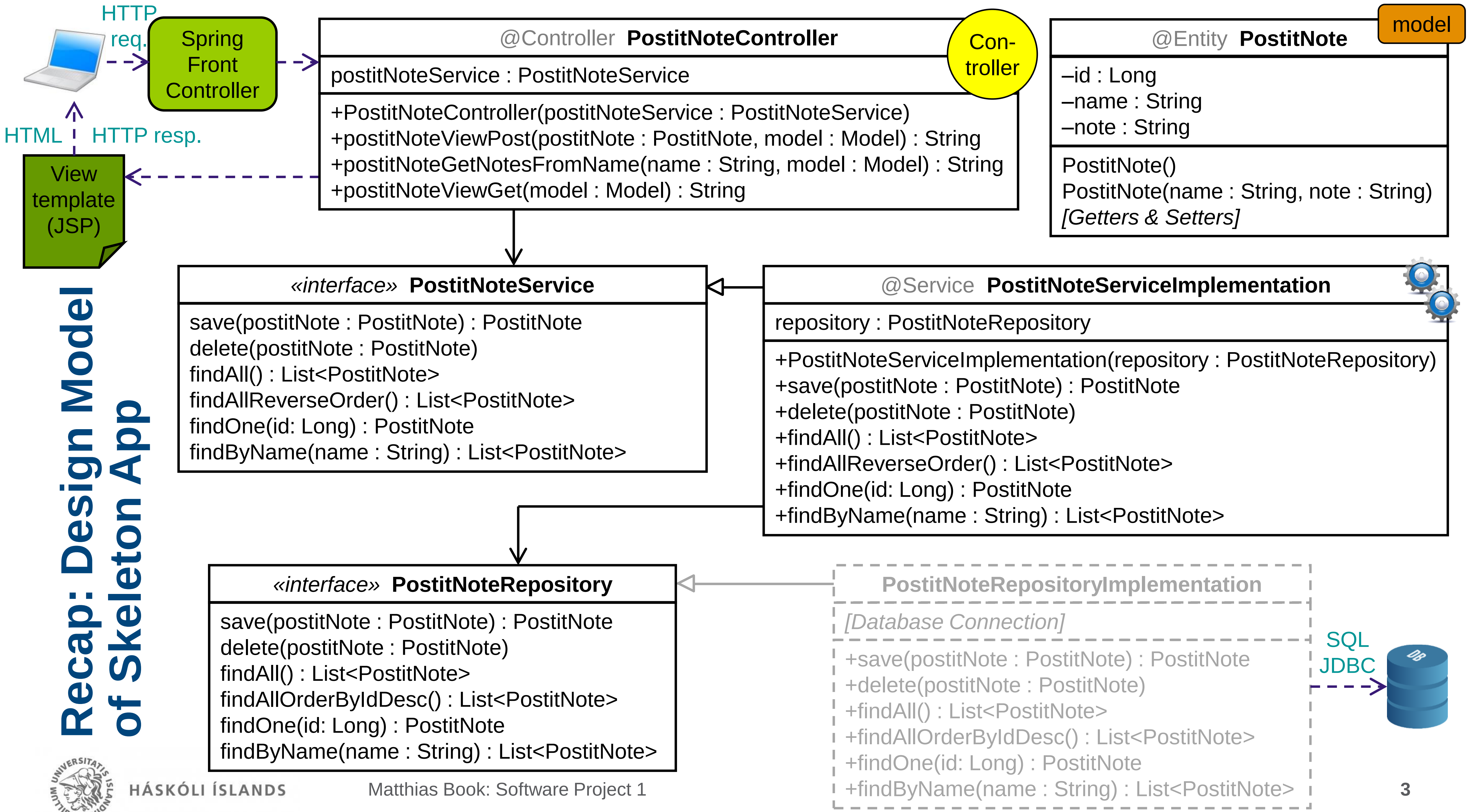
Matthias Book

HÁSKÓLI ÍSLANDS
VERKFRÆÐI- OG NÁTTÚRUVÍSINDASVIÐ
IÐNAÐARVERKFRÆÐI-, VÉLAVERKFRÆÐI-
OG TÖLVUNARFRÆÐIDEILD

Recap: Spring Web MVC Framework



Recap: Design Model of Skeleton App


HTTP req.
Spring Front Controller
HTML
HTTP resp.
View template (JSP)
@Controller PostitNoteController
Con-troller
postitNoteService : PostitNoteService
+PostitNoteController(postitNoteService : PostitNoteService)
+postitNoteViewPost(postitNote : PostitNote, model : Model) : String
+postitNoteGetNotesFromName(name : String, model : Model) : String
+postitNoteViewGet(model : Model) : String
model
@Entity PostitNote
–id : Long
–name : String
–note : String
PostitNote()
PostitNote(name : String, note : String)
[Getters & Setters]
«interface» PostitNoteService
save(postitNote : PostitNote) : PostitNote
delete(postitNote : PostitNote)
findAll() : List<PostitNote>
findAllReverseOrder() : List<PostitNote>
findOne(id: Long) : PostitNote
findByName(name : String) : List<PostitNote>
@Service PostitNoteServiceImplementation
repository : PostitNoteRepository
+PostitNoteServiceImplementation(repository : PostitNoteRepository)
+save(postitNote : PostitNote) : PostitNote
+delete(postitNote : PostitNote)
+findAll() : List<PostitNote>
+findAllReverseOrder() : List<PostitNote>
+findOne(id: Long) : PostitNote
+findByName(name : String) : List<PostitNote>
«interface» PostitNoteRepository
save(postitNote : PostitNote) : PostitNote
delete(postitNote : PostitNote)
findAll() : List<PostitNote>
findAllOrderByIdDesc() : List<PostitNote>
findOne(id: Long) : PostitNote
findByName(name : String) : List<PostitNote>
PostitNoteRepositoryImplementation
[Database Connection]
+save(postitNote : PostitNote) : PostitNote
+delete(postitNote : PostitNote)
+findAll() : List<PostitNote>
+findAllOrderByIdDesc() : List<PostitNote>
+findOne(id: Long) : PostitNote
+findByName(name : String) : List<PostitNote>
SQL
JDBC
DB

Quiz #5 Solution: JPA Query Methods

Which JPA query method (1-8) will generate which SQL clause (a-h)?

(Consider the questions as independent – the type of `bar` *varies between questions.)*

`SELECT f FROM Foo f WHERE...`	`List<Foo> ...`
a) `f.bar = ?1`	2. `findByBar(Bar bar)`
b) `f.bar = ?1 AND f.baz = ?2`	6. `findByBarAndBaz(Bar bar, Baz baz)`
c) `f.bar BETWEEN ?1 AND ?2`	8. `findByBarBetween(Bar bar1, Bar bar2)`
d) `f.bar = ?1 ORDER BY f.baz DESC`	5. `findByBarOrderByBazDesc(Bar bar)`
e) `f.bar <= ?1`	4. `findByBarLessThanEqual(Bar bar)`
f) `f.bar IN ?1`	7. `findByBarIn(Collection<Bar> bars)`
g) `UPPER(f.bar) = UPPER(?1)`	3. `findByBarIgnoreCase(Bar bar)`
h) `f.bar = TRUE`	1. `findByBarTrue()`

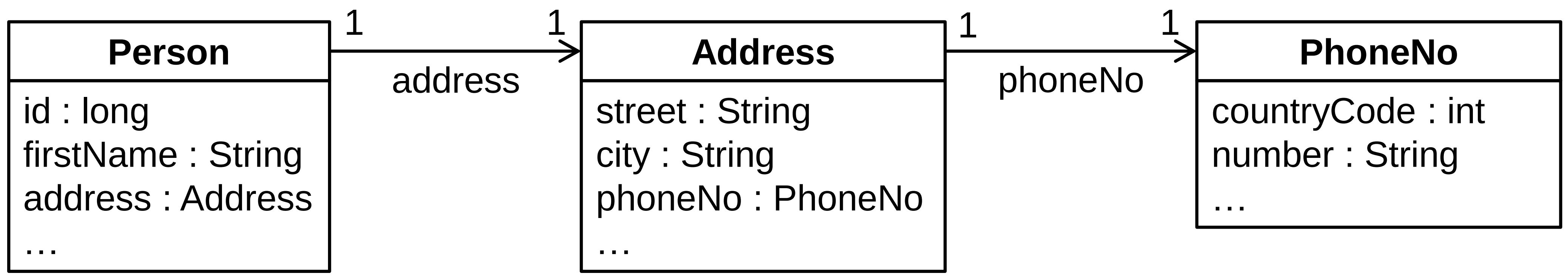
Beyond the Skeleton App: **Mapping Complex Data Types**

```
@Entity
public class PostitNote {
  @Id
  @GeneratedValue(strategy =
    GenerationType.IDENTITY)
  private Long id;
  private String name;
  private String note;

  public String getName() { return name; }
  public void setName(String name) {
    this.name = name; }
  // [...other getters & setters...]
}
```

- Recap: Automatic mappings to SQL types exist for common Java types:
  - Primitive types (`int`, `float`, `boolean`, …)
  - Primitive type wrappers (`Integer`, …)
  - Strings and arrays (`char`, `byte[]`, …)
  - `Date`, `Time`, `Calendar`
  - Enumerated properties (`enum`)
  - Any class implementing `Serializable`
- **What about complex attributes?**
  - Complex properties of the entity, e.g. a structured `PhoneNumber`
    - ➢ Embedded properties
  - References to other entities, e.g. the `LineItems` of a `Receipt`
    - ➢ Entity relationships

Composite One-to-One Relationships in UML

- Example: Composition of personal information



Composite One-to-One Relationships as Embedded Properties using JPA

```
@Entity
public class Person {
  private long id;
  private String firstName;
  private Address address;
  @Id
  @GeneratedValue(strategy =
    GenerationType.IDENTITY)
  public long getID() {...}
  public void setID(long id) {...}
  @Embedded
  public Address getAddress() {...}
  public void setAddress(Address addr) {...}
  // ...
}
```

Indicates the fields of Address shall be incorporated directly into the Person table

```
@Embeddable
public class Address {
  private String street;
  private String city;
  private PhoneNo phoneNo;
  public String getStreet() {...}
  public void setStreet(String street) {...}
  public String getStreet() {...}
  public void setStreet(String street) {...}
  @Embedded
  public PhoneNo getPhoneNo() {...}
  public void setPhoneNo(PhoneNo phNo) {...}
  // ...
}
```

Indicates that Address data is not stored in a table of its own, but embedded into another entity's table

Note: Therefore, no @Id here!

Regular POJO

Embeddable properties can contain embedded properties themselves

HÁSKÓLI ÍSLANDS

Composite One-to-One Relationships in RDBMS

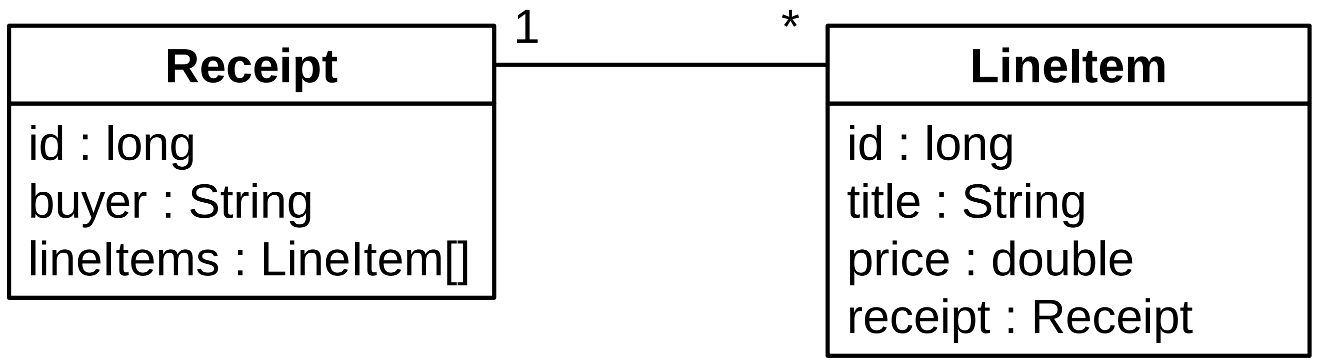
- Example: Resulting database table

Person

id	firstName	street	city	countryCode	number	…
1	Matthias	Dunhagi	Reykjavík	354	525-4603	…
2	Christy	Nathan Road	Hong Kong	852	3107-0566	…
…	…	…	…	…	…	…

One-to-Many Entity Relationships in UML

- Example: Aggregation of line items in a sales receipt



One-to-Many Entity Relationship using JPA

```
@Entity
public class Receipt {
  private long id;
  private String buyer;
  private Set<LineItem> lineItems =
    new HashSet<>();

  @Id
  @ColumnName(name = "ReceiptId")
  @GeneratedValue(strategy =
    GenerationType.IDENTITY)
  public long getId() {...}
  public void setId(long id) {...}

  public String getBuyer() {...}
  public void setBuyer(String buyer) {...}

  @OneToMany(mappedBy = "receipt",
    fetch = FetchType.LAZY,
    cascade = CascadeType.ALL,
    orphanRemoval = true)
  public Set<LineItem> getLineItems()
    {...}
  public void setLineItems(
    Set<LineItem> lineItems) {...}
}
```

Name of related entity's attribute that references this entity

Don't retrieve related entities from DB until someone accesses them in Java

Action on related entities when this one is deleted

Object-oriented declaration of the One-to-Many relation: A reference to a Set of the "many" objects

One-to-Many Entity Relationship in RDBMS (“One” Side)

- Example: Resulting database tables

Receipt

ReceiptId	buyer
1	Matthias
2	Christy
…	…

Many-to-One Entity Relationship using JPA

```
@Entity
@Table(name = "Receipt_LineItem")
public class LineItem {
  private long id;
  private String title;
  private double price;
  private Receipt receipt;
  @Id
  @ColumnName(name = "LineItemId")
  @GeneratedValue(strategy =
    GenerationType.IDENTITY)
  public long getId() {...}
  public void setId(long id) {...}
  public String getTitle() {...}
  public void setTitle(String title) {...}
  public double getPrice() {...}
  public void setPrice(double price) {...}

  @ManyToOne(fetch = FetchType.EAGER,
    optional = false)
  @JoinColumn(name = "ReceiptId")
  public Receipt getReceipt() {...}
  public void setReceipt(Receipt receipt)
    {...}
}
```

Reference to related entity

Retrieve related entity from DB immediately

There must be a related entity

Name of column in this table that will contain the other table's primary key

Object-oriented declaration of the Many-to-One relation: A reference to the "one" object

One-to-Many Entity Relationship in RDBMS (“Many” Side)

- Example: Resulting database tables

Receipt

ReceiptId	buyer
1	Matthias
2	Christy
…	…

Receipt_LineItem

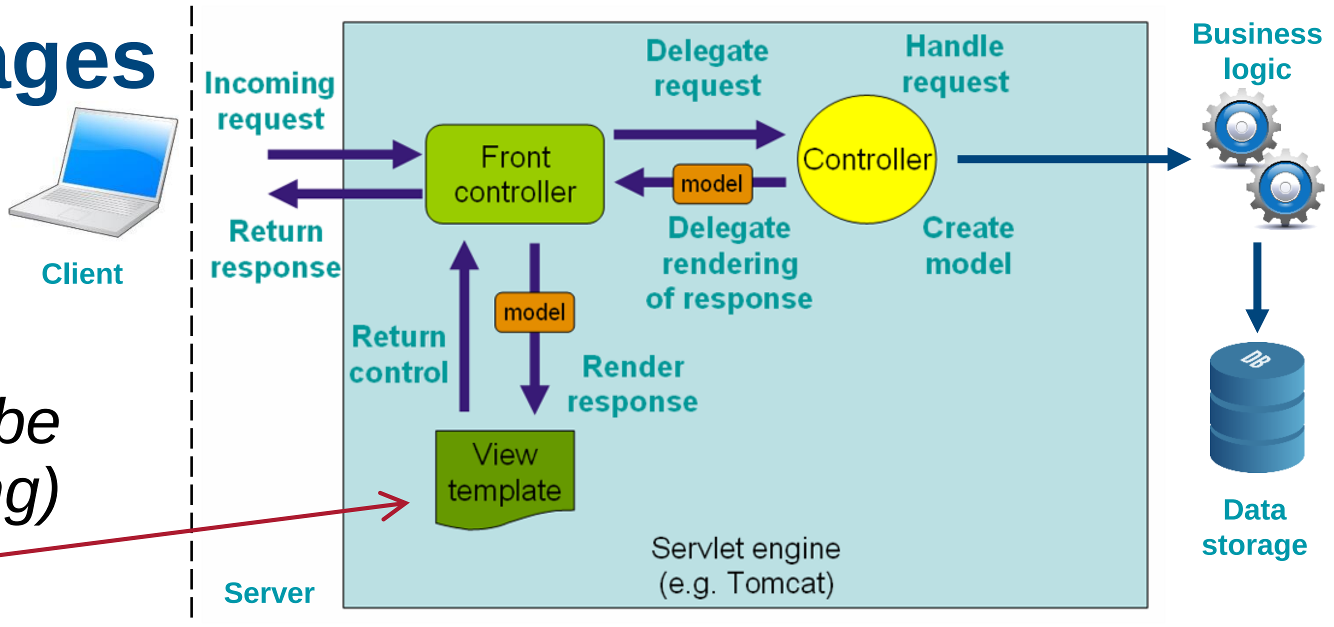
LineItemId	title	Price	ReceiptID
1	Textbook	7000	1
2	Headset	3500	1
3	Raincoat	21000	2
4	Movie	3500	2
…	…	…	…

JavaServer Pages

see also:

- Williams: Professional Java for Web Applications, Ch. 4 & 6

Motivation: JavaServer Pages



- Constructing a server-side web application's user interface *(i.e. constructing the web pages to be returned to the browser for rendering)* is the job of the View Templates

- Necessary design compromise:
  - On one hand, the implementation of the view templates should be structurally close to the end product, i.e. HTML pages
  - On the other hand, the view templates need to be dynamically populated with data and may have to visually adapt to that data
    - e.g. accommodate different volumes of data, including no data
  - ➢ Need to embed control elements into HTML structure

➢ Solution: JavaServer Pages (JSP), Expression Language (EL) and Tag Libraries

Example: Time-of-Day JSP (`showtime.jsp`)

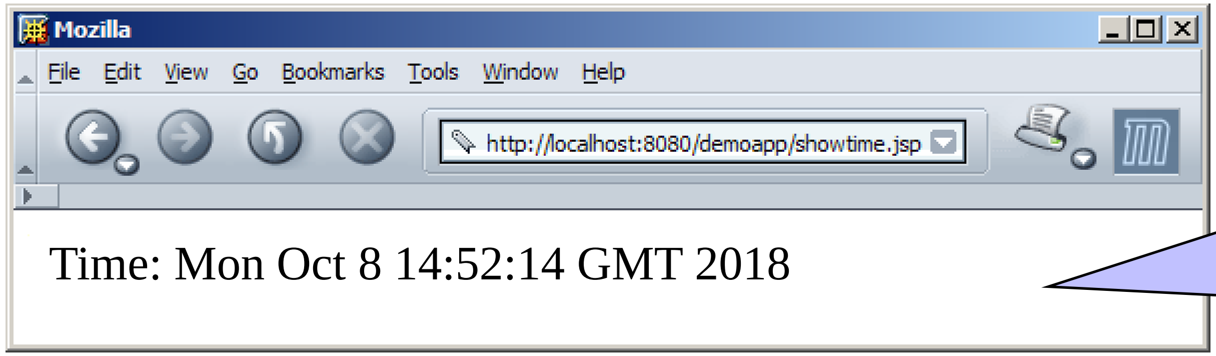
```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page import="java.util.Date" %>

<html>
  <body>
    <p>Time: <%= new Date() %></p>
  </body>
</html>
```

Static text: included directly in HTML page

Script element: Content is evaluated at runtime and integrated into HTML page

Invoking the JSP

```
http://localhost:8080/demoapp/showtime.jsp
```


Mozilla
File Edit View Go Bookmarks Tools Window Help
http://localhost:8080/demoapp/showtime.jsp

Time: Mon Oct 8 14:52:14 GMT 2018

```
<html>
  <body>
    <p>Time: Mon Oct 8 14:52:14 GMT 2018</p>
  </body>
</html>
```

- When the Front Controller delegates a request to a JSP, its script elements, expressions and dynamic tags are executed / evaluated and create static HTML elements that are embedded into the rest of the JSP’s static HTML code
- The resulting HTML code is returned in the HTTP response to the web browser, where it is rendered as a web page.

Integration of Java Statements into JSPs *(discouraged)*

- Importing classes with the `page` directive:

  `<%@page import="`*`package`*$_1$*`.class`*`, `*`package`*$_2$`.*, ..." %>`

  - makes given classes from given packages available in JSP
- Integrating Java statements with Scriptlets:

  `<% `*`Java statements`*` %>`

  - Executes Java code in brackets at runtime
  - Java control structures can be applied to static HTML code:

```
<% if (request.getParameter("passwd").equals("mySecretPassword")) { %>
  <p>Password correct!</p>
<% } else { %>
  <p>Password incorrect!</p>
<% } %>
```

- Integrating Java expressions into JSPs: `<%= `*`Java expression`*` %>`
  - Evaluates expression in brackets
  - Converts the result to a string
  - Includes the string into the HTML code

Implicit Objects in Java Fragments *(discouraged)*

- The following objects are available in Java fragments (statements or expressions) in any JSP, without having to be explicitly imported:
  - `HttpServletRequest request`: current request
  - `HttpServletResponse response`: current response
  - `PageContext pageContext`: page-specific data store
  - `HttpSession session`: current user's session (data store)
  - `ServletContext application`: application-wide data store
  - `JspWriter out`: text output stream to client
  - `ServletConfig config`: web application configuration (data store)

- Problem with Java code built into JSP:
  No separation of application logic and presentation
  - ➢ **Avoid – use Expression Language and Custom Tags instead!**

Excursion: JavaBeans

- Accessing data model is easiest when entities follow **JavaBeans** convention:
- JavaBeans are regular classes adhering to this structure:
  - Have a constructor without paramters:

    `public BeanName()`
  - Have a `get` method for each readable `property`:

    `public PropertyType getProperty()`
  - Have a `set` method for each writable `property`:

    `public void setProperty(PropertyType property)`
- Rules for properties:
  - usually attributes of the JavaBean class, but
  - can also be calculated in `get` methods
  - can also be combined with other values in `set` methods
  - *`PropertyType`* can be a primitive type, reference type or array

Expression Language (EL)

- Usage scenarios
  - in static HTML code (will be evaluated and incorporated as string into HTML code)
  - in attributes of JSP elements (such as actions and custom tags)
- Syntax: `${`*`expression`*`}`
  - Operators: common operators for comparison, arithmetics etc.
  - Operands: Literals, JavaBeans, implicit objects, methods
  - Expression will be evaluated to a typed value (primitive or reference type)
  - When used in static text, automatic conversion to `String`
- Accessing structured objects:

EL syntax	Type of *a*	Semantics of *b*	equiv. Java syntax
a`.`*b*	Array	Index	*a*`[`*b*`]` or *a*`["`*b*`"]`
a`[`*b*`]`	List	Index	*a*`.get(`*b*`)`
a`[`*b*`]`	Map	Key	*a*`.get(`*b*`)`
a`[`*b*`]`	JavaBean	Property name	*a*`.get`*B*`()`

Accessing JavaBean Properties

`${beanInstance.property}`

- looks for the JavaBean *`beanInstance`* in the page, request, session and application scope (in this order) and returns the value of that bean's *`property`*

```
${pageScope.beanInstance.property}
${requestScope.beanInstance.property}
${sessionScope.beanInstance.property}
${applicationScope.beanInstance.property}
```

- retrieves the JavaBean *`beanInstance`* from the given scope and returns the value of that bean's *`property`*

Implicit Objects in EL

The following objects are available in EL expressions in any JSP, without having to be imported explicitly:

- `pageScope.`*`obj`*, `requestScope.`*`obj`*, `sessionScope.`*`obj`*, `applicationScope.`*`obj`*
  - delivers the object *`obj`* in the page, request, session or application scope
- `param.`*`parameter`*
  - delivers the value of the first request parameter called *`parameter`*
- `paramValues.`*`parameter`*
  - delivers an array with the values of all request parameters called *`parameter`*
- `cookie.`*`cookieName`*
  - delivers the value stored in the cookie *`cookieName`*

Example

```
<html>
  <body>
    <p>Welcome, ${sessionScope.user.username}!</p>
    <p>Your password is ${user.passwd}.</p>
    <p>Chosen language: ${param.lang} </p>
  </body>
</html>
```

Get `username` property of `user` bean from session

Access `passwd` property without specifying scope of `user` bean

Read request parameter `lang`

Tag Libraries

see also:

- Williams: Professional Java for Web Applications, Ch. 7

JSP Standard Tag Library and Custom Tags

- Tag libraries define additional tags that can be incorporated into the HTML code to generate markup or implement control structures
  - Syntax: `<`*`prefix`*`:`*`tag`*` ...>...</`*`prefix`*`:`*`tag`*`>`
  - Developers can build “taglibs” with individual custom tags and use them in their JSPs
  - Many template engines are based on custom tags

- The **JSP Standard Tag Library (JSTL)** contains a variety of tags for control flow, XML processing, internationalization, SQL queries, string manipulation etc.
  - in the libraries `core`, `xml`, `fmt`, `sql`, `functions`
  - Imported into JSP with the `taglib` directive:

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

    - Imports the JSTL Core taglib
    - Tags of this library have the prefix `c`

Conditional Evaluation: `if` Tag

- Syntax:

```
<c:if test="${expression}">
  Element content
</c:if>
```

  - If *`expression`* is evaluated to `true`, the *`Element content`* is evaluated and incorporated into the output.

- Example:

```
<c:if test="${user.passwd == 'mySecretPassword'}">
  Password correct!
</c:if>
```

  - Will integrate the string `Password correct!` into the output if the property `passwd` of the `user` object has the value `mySecretPassword`; otherwise, no output is created.

Conditional Evaluation: choose Tag

```
<c:choose>
  <c:when test="${expr1}"}>
    Element Content1
  </c:when>
  <c:when test="${expr2}">
    Element Content2
  </c:when>
  <c:otherwise>
    Element Content3
  </c:otherwise>
</c:choose>
```

- the *`Element Content`* of the first `when` block whose *`expr`* can be evaluated to `true` will be evaluated and incorporated into the output
- If no *`expr`* can be evaluated to `true`, the *`Element Content`* of the `otherwise` block will be evaluated and incorporated into the output

Iteration: `forEach` Tag

- Syntax:

```
<c:forEach var="element" items="${collection}">
  Content using ${element}
</c:forEach>
```

  - Evaluates the *`Content`* for each element of the *`collection`*
  - The current element of each iteration is available in the variable *`element`*
- Treatment of various types of collections:
  - Array of primitive or reference types
    - primitive types are packed in instances of wrapper classes (`double` → `Double` etc.)
  - Implementations of `Collection` or `Map` interfaces
    - `Map` elements are placed in `Map.Entry` instances
  - Implementations of `Enumeration` or`Iterator` interfaces
    - possible only once per JSP due to lack of access to reset mechanism
  - `String` of comma-separated substrings

Example

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
  <body>
    <c:forEach var="user" items="${list}">
      <p>User ${user.username} has password ${user.passwd}.</p>
    </c:forEach>
    ...
    <p>Chosen language:
    <c:choose>
      <c:when test="${param.lang == 'is'}">Íslenska</c:when>
      <c:when test="${param.lang == 'en'}">English</c:when>
      <c:otherwise>undefined</c:otherwise>
    </c:choose>
    </p>
  </body>
</html>
```

Elements of `list` are subsequently read into `user`, and tag content built with that variable value

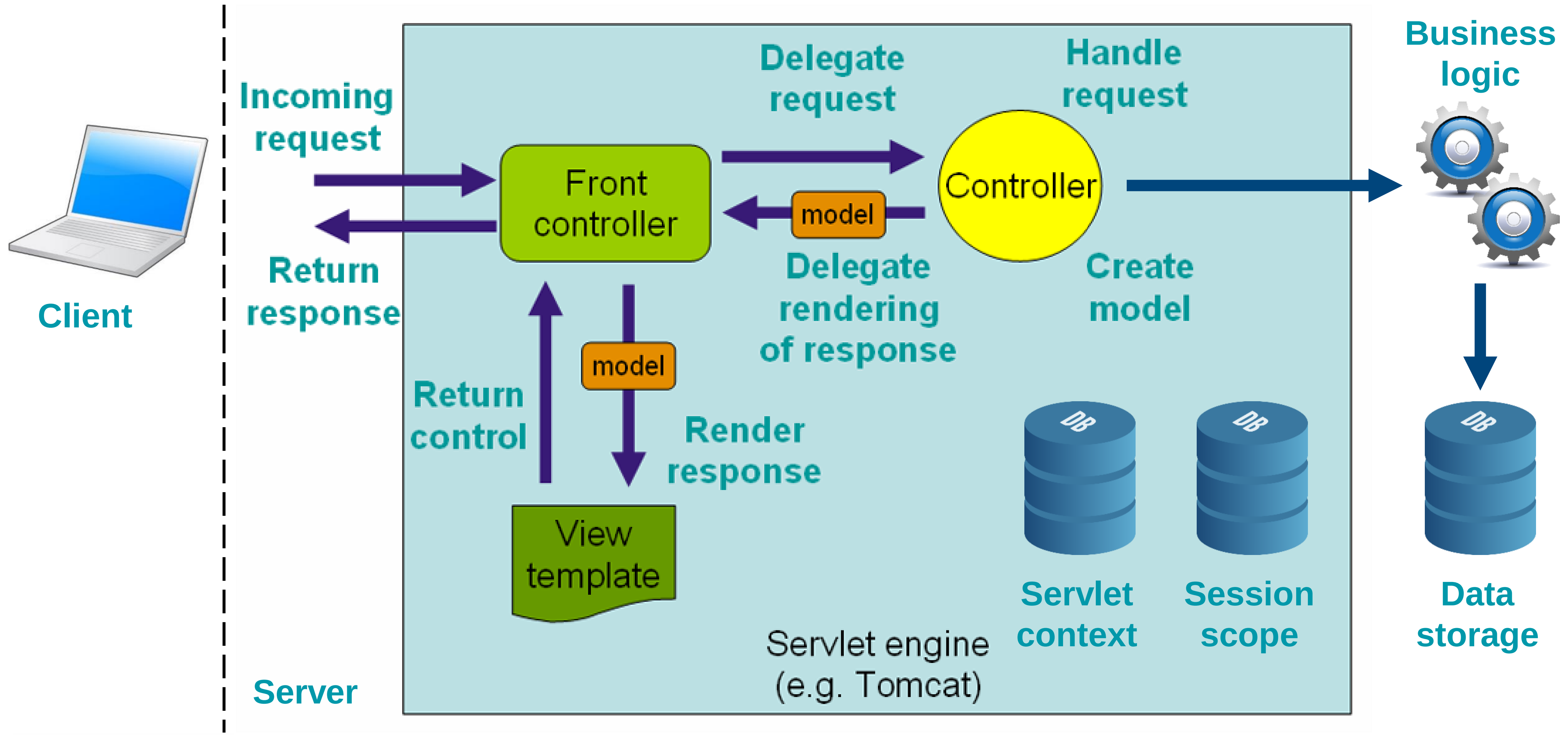
Evaluation depends on value of request parameter `lang`

Summary: Elements of JSPs

- Static text
  - Text with HTML markup that is sent to output (HTTP response) unchanged
- Directives: `<%@ ... %>`
  - Commands that are interpreted at the time of generation of the servlet from the JSP, but which do not create any output
- Scripting Elements: `<% ... %>`, `<%= ... %>`
  - Java source code fragments that are executed at runtime, and whose output is incorporated into the response *(discouraged)*
- Expressions: `${...}`
  - Expressions enabling simple access to objects' properties
- Actions: `<jsp:...>...</jsp:...>`
  - Statements that influence the behavior of the JSP at runtime
- Tags: `<`*`prefix`*`:`*`tag`*` ...>...</`*`prefix`*`:`*`tag`*`>`
  - Statements (self-defined or provided by the JSTL or frameworks) that influence the behavior of the JSP at runtime and may create output sent to the response

Note: Any JSP "behavior" occurs at the time of creating the HTTP response (i.e. usually creating an HTML page) on the server. It is **not** behavior executed in the client's browser!

Recap: Spring MVC Web Application Structure


Client
Server
Incoming request
Return response
Front controller
Delegate request
Handle request
Controller
model
Delegate rendering of response
Create model
Return control
model
Render response
View template
Servlet engine (e.g. Tomcat)
Servlet context
Session scope
Business logic
Data storage
DB
DB
DB