What is a Java Bean?

- Simply a class wrapping some data...
  ...and meets certain restrictions
- Three components:
  - Default constructor (no arguments)
  - Private properties (instance variables) only, no public ones!
  - For each field, provide a getter method to retrieve it and a setter method to modify it
**Why Java Beans?**

- **Reusable and Portable Component Model**
  - No need to repeat the same code/queries
  - Code maintenance becomes much easier
  - Better use of resources
- Managed by container (Apache Tomcat)
  - Bean scope can be page, request, session or application
- Easy to assign HTTP request parameters to bean properties
- Easy to share beans among multiple HTTP requests, servlets, actions and JSPs
- Can be introspected, persisted and customized

---

**Currency Converter Example**

[Diagram showing the currency converter example with app server, form.html, submit, ConverterBean, and result.jsp in session scope.]
form.html

```html
<html>
<head>
    <title>Currency Conversion Form</title>
</head>
<body>
    <h1>Currency Conversion Form</h1>
    <p>Enter an amount to convert:</p>
    <form action="result.jsp" method="GET">
        <input type="text" name="usdAmount" size="25"><br/>
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

result.jsp (Without Java Bean)

```html
<html>
<head><title>Currency Conversion Result</title></head>
<body>
    <h1>Currency Conversion Result</h1>
    <%
        String usdAmount = request.getParameter("usdAmount");
        BigDecimal yenRate = new BigDecimal("88.75");
        BigDecimal yenAmount =
            new BigDecimal(usdAmount).multiply(yenRate);
        yenAmount = yenAmount.setScale(2, BigDecimal.ROUND_UP);
    %>
    <p>% usdAmount %> USD are % yenAmount %> Yen</p>
</body>
</html>
```
Converter Bean

```java
package converter;
public class ConverterBean {
    // private properties
    private String usdAmount = "0.0";
    private String yenRate = "88.75";
    private BigDecimal yenAmount;
    // Getters and setters automatically generated by Eclipse
    // Menu Source -> Generate Getters and Setters...
    public String getUsdAmount() { return usdAmount; }
    public void setUsdAmount(String usdAmountStr) {
        usdAmount = usdAmountStr;
        ...
    }
    ...
}
```

Naming Convention

- Connect property names with getter/setter method names
- Capitalize the first letter of the property and add the word “get” (“set” respectively)
  - Property: `usdAmount`
  - Getter: `getUsdAmount`
  - Setter: `setUsdAmount`
How to Use a Java Bean in a JSP

- Compile bean and place under WEB-INF/classes

```xml
<jsp:useBean id="<beanName>" class="<beanClass>"
    scope="<scope>">
    </jsp:useBean>
- Equivalent to
  <% <beanClass> <beanName> = new <beanClass>(); %>

```xml
<jsp:getProperty name="<beanName>"
    property="<propertyName>">
- Equivalent to <%= <beanName>.getPropertyName() %>

```xml
<jsp:setProperty name="<beanName>"
    property="<propertyName>"
    value="<value>">
- Can be initialized when subelement of jsp:useBean
```

result.jsp (With Java Bean)

```jsp
<%@ page import="converter.*"%>
<jsp:useBean id="conv" scope="session"
    class="converter.ConverterBean"/>
<jsp:setProperty name="conv" property="*"/>
<html>
<head><title>Currency Conversion Result</title></head>
<body>
  <h1>Currency Conversion Result</h1>
  <p>
  <%=
      <jsp:getProperty name="conv" property="usdAmount"/>
  %>
  USD are
  <%=
      <jsp:getProperty name="conv" property="yenAmount"/>
  %>
  Yen
  </p>
</body>
</html>
```
resultInitRate.jsp

- Initialize bean properties

```jsp
<jsp:useBean id="conv" scope="request"
class="converter.ConverterBean">
    <jsp:setProperty name="conv" property="yenRate"
    value="100" />
</jsp:useBean>
<jsp:setProperty name="conv" property="*"/>
<html>
<head><title>Currency Conversion Result</title></head>
<body>
    ...
</body>
</html>
```

Why not session?

Data Access Object (DAO)
Java Design Pattern

- A Data Access Object (DAO) is a bean encapsulating database access code
- Completely separates DB access code from application logic and presentation code
- Improves code maintenance and portability
- Improves database server performance
**StudentBean**

```java
public class StudentBean {
    private Integer id = null;
    private String first = null, middle = null, last = null;

    private String selectStr = "SELECT * FROM Students";
    private String insertStr = "INSERT INTO Students VALUES (?,?,?,?)";
    private String updateStr = "UPDATE Students SET first_name = ?, "
        + "middle_name = ?, last_name = ? WHERE id = ? ";
    private String deleteStr = "DELETE FROM Students WHERE id = ?";

    public Integer getId() { return id; }
    public void setId(Integer id) { this.id = id; }
    
    public ResultSet getAllStudents() {...}
    public void insertStudent() {...}
    public void updateStudent() {...}
    public void deleteStudent() {...}
    public void close() { ... }
}
```

**Data Entry Example – 1st Attempt**

```java
public class StudentBean {
    ....
    public void insertStudent() {...}
    ....
}
```
Data Entry Example – 1st Attempt

Advantages
• Consolidated SQL statements and connection code in StudentBean
  – Improves code maintenance and portability
• StudentBean makes it easy to process request parameters

Disadvantages
• StudentBean mixes DB access and session data
• Still need to manually close connection

Data Entry Example – 2nd Attempt

```java
public class StudentBean {
    private Integer id = null;
    private String first = null;
    private String middle = null;
    private String last = null;

    public Integer getId() { return id; }
    public void setId(Integer id) { this.id = id; }

    public void clear() {
        id = null;
        first = null;
        middle = null;
        last = null;
    }
}
```
Data Entry Example – 2nd Attempt

**StudentModel**

```java
public class StudentModel {
    private static String selectStr = "...;"
    private static String insertStr = "...;"
    private static String updateStr = "...;"
    private static String deleteStr = "...;"

    public static CachedRowSet getAllStudents() {...}
    public static void insertStudent(StudentBean student) {...}
    public static void updateStudent(StudentBean student) {...}
    public static void deleteStudent(StudentBean student) {...}
}
```

Data Entry Example – 2nd Attempt

**StudentModel**

```java
public static void insertStudent(StudentBean student)
    throws SQLException, NamingException {
    Connection conn = DBConnectionPool.getConnection();
    PreparedStatement pStmt = conn.prepareStatement(insertStr);

    pStmt.setString(1, student.getFirst());
    ...
    pStmt.executeUpdate();
    conn.commit();
    pStmt.close();
    conn.close();
}
```
StudentModel

public static CachedRowSet getAllStudents() throws SQLException, NamingException {
    Connection conn = DBConnectionPool.getConnection();
    PreparedStatement pStmt = conn.prepareStatement(selectStr);
    ResultSet allStudents = pStmt.executeQuery();

    CachedRowSet crsStudents = new CachedRowSetImpl();
    crsStudents.populate(allStudents);

    allStudents.close();
    pStmt.close();
    conn.close();
    return crsStudents;
}

Advantages over 1st Attempt

- StudentBean stores session data only
- StudentModel executes DB access code only
- No need to manually close any connection