

Разработка Servlet и JSP в соответствии со стандартом Java EE 6

Практические работы

Оглавление

Оглавление	3
Упражнение 1. Пример простого Servlet	6
Раздел 1. Создание Enterprise приложения и Web проекта	7
Раздел 2. Создание Servlet	16
Раздел 3. Создание бизнес логики.	21
Раздел 4. Создание класса LibraryIdGenerator и метода generateId().	24
Раздел 5. Тестирование сервлета	26
Упражнение 2. Создание базы данных библиотеки	29
Раздел 1. Импорт проекта и создание базы.....	30
Упражнение 3. Сервlet с параметрами.....	33
Раздел 1. Импорт проекта с логикой для взаимодействия с базой данных	34
Раздел 2. Обеспечение взаимодействия Web приложения с классами проекта LibraryJava	38
Раздел 3. Импорт страницы Register.html	41
Раздел 4. Изменение логики сервлета	42
Раздел 5. Создание источника данных (data source) и тестирование сервлета	51
Упражнение 4. Простая JSP страница	63
Раздел 1. Создание JSP Register.....	64
Раздел 2. Создание JSP RegistrationSuccess.....	68
Упражнение 5. Вызов JSP страниц из сервлета.....	73
Раздел 1. Изменение сервлета RegisterPatron	74
Раздел 2. Передача параметров для JSP страниц.....	79
Раздел 3. Сохранение введенных значений	83
Упражнение 6. Управление сессиями	86
Раздел 1. Создание сервлета для аутентификации	87
Раздел 2. Создание страницы Login JSP	89
Раздел 3. Создание сервлета ProcessLogin.....	98

Раздел 4. Создание сервлета ProcessListItems	101
Раздел 5. Создание страницы ListItems JSP	104
Раздел 6. Тестирование сервлетов и JSP	106
Раздел 7. Сохранение Patron ID в Cookie.....	108
Упражнение 7. Создание JavaBean.....	113
Раздел 1. Создание LoanedCopyListBean JavaBean.....	114
Раздел 2. Создание LoanedCopyListBean JavaBean.....	116
Упражнение 8. Совместное использование сервлетов, страниц JSP и JavaBeans...	119
Раздел 1. Обновление сервлета ProcessListItems.....	120
Раздел 2. Обновление JSP ListItems	123
Раздел 3. Создание Error.jsp	126
Раздел 4. Тестирование системы	129
Раздел 5. Обновление JSP страниц для использования тегов <useBean>	133
Упражнение 9. Использование JSP Expression Language	139
Раздел 1. Изменение JSP страниц для системы регистрации.....	140
Раздел 2. Изменение JSP страниц системы входа и просмотра списка книг.....	142
Упражнение 10. Новые JSP теги.....	145
Раздел 1. Использование тега JSTL для цикла	146
Раздел 2. Создание своего тега JSP	147
Раздел 3. Создание дескриптора для тега собственной разработки .	151
Раздел 4. Изменение ListItems JSP для использования нового тега..	155
Раздел 5. Создание сервлета RenewItems.....	157
Раздел 6. Тестирование новых функций	158
Раздел 7. Отображение статуса продления с помощью View Bean ...	160
Раздел 8. Отображение количества продлений для книги с помощью Tool tip.....	162
Упражнение 11. Сервлетные фильтры	167
Раздел 1. Создание и настройка фильтра таймера	168
Раздел 2. Добавление логики фильтра	170

Раздел 3. Тестирование фильтра	171
Раздел 4. Создание Helper класса	171
Раздел 5. Создание и настройка фильтра Appender	173
Раздел 6. Создание логики фильтра Appender	176
Раздел 7. Тестирование фильтра	177
Упражнение 12. Создание JSF модулей	178
Раздел 1. Создание проектов	179
Раздел 2. Создание шаблонов и страниц для JSF проекта	182
Раздел 3. Добавление JavaBean и правила навигации	188
Раздел 4. Тестирование разработанного функционала	194

Упражнение 1. Пример простого Servlet

О чём это упражнение:

В этой лабораторной работе вы создадите сервлет и протестируете его. Сервлет создает документ и возвращает его в браузер пользователя. Документ подтверждает, что клиент был создан, и демонстрирует его ID.

Что вы должны будете сделать:

- Создать Enterprise и Web проекты
- Создать сервлет
- Протестировать сервлет в среде WebSphere Application Server
- Проверить сервлет с помощью браузера

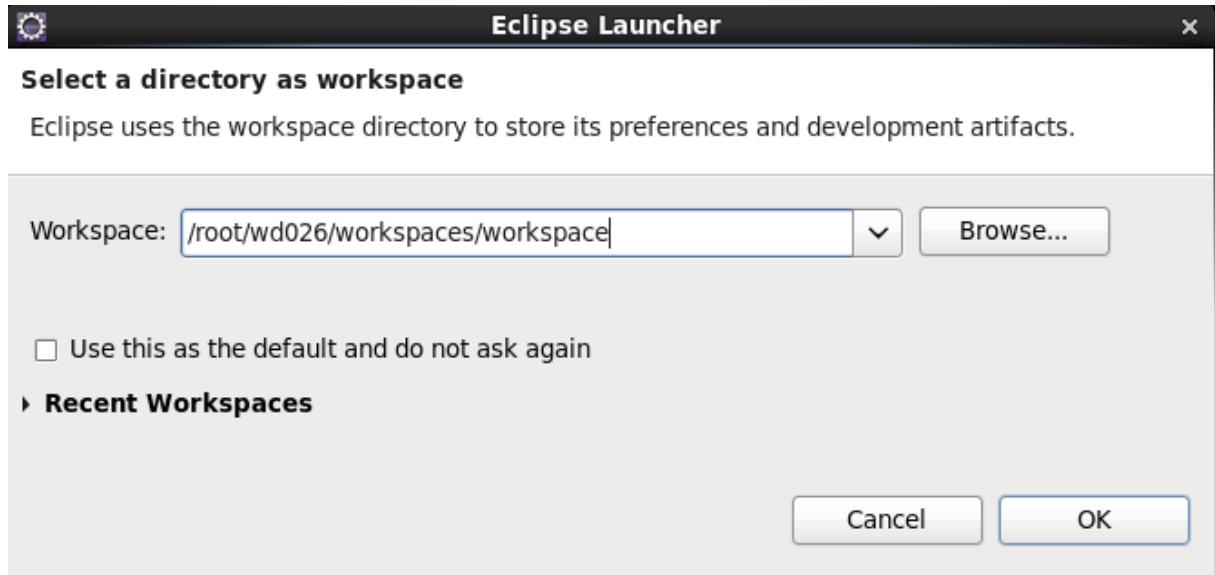
Раздел 1. Создание Enterprise приложения и Web проекта

С помощью среды Eclipse вы имеете возможность создавать любые Java EE модули и собирать их в единые приложения. Ключевой термин – Enterprise приложение (корпоративное приложение) – это верхний уровень иерархии с точки зрения группировки отдельных модулей. Физически эти приложения представляют собой архивы с расширением EAR. Эти архивы разворачиваются и устанавливаются на сервера приложений (например, WebSphere Application Server). В Eclipse для создания такого приложений нужно завести проект соответствующего типа (Enterprise Application Project). Затем внутри этого приложения создается Web модуль, который содержит сервлет. Web модуль хранится в виде архива формата WAR. В среде разработки для таких модулей создаются проекты типа Web Module Project. Это проект связывается с ранееенным корпоративным приложением. В данном разделе создаются необходимые проекты для системы библиотеки, работу которой мы собираемся автоматизировать в этом лабораторном практикуме.

1. Запустите Eclipse, если он еще не запущен.
 - a. Реквизиты для доступа к виртуальной машине:
Учетная запись: **root**
Пароль: **web1sphere**
 - b. Воспользуйтесь ярлыком для запуска среды разработки на рабочем столе: **Eclipse**.

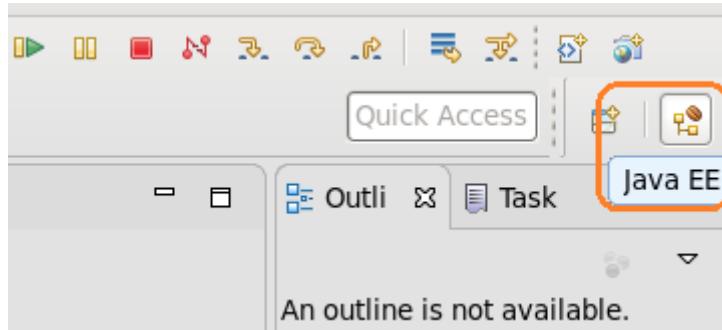
с. Укажите workspace (рабочее пространство):

/root/wd026/workspaces/workspace



д. После того как среда загрузится, закройте страницу **Welcome**.

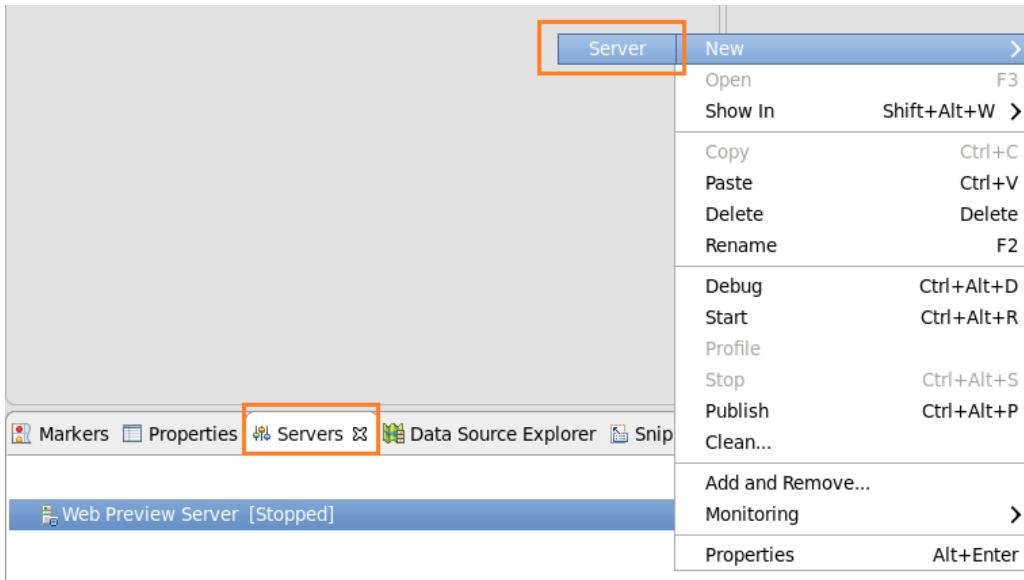
е. По умолчанию открывается перспектива Java EE, в чем можно убедиться по пиктограмме в правом верхнем углу:



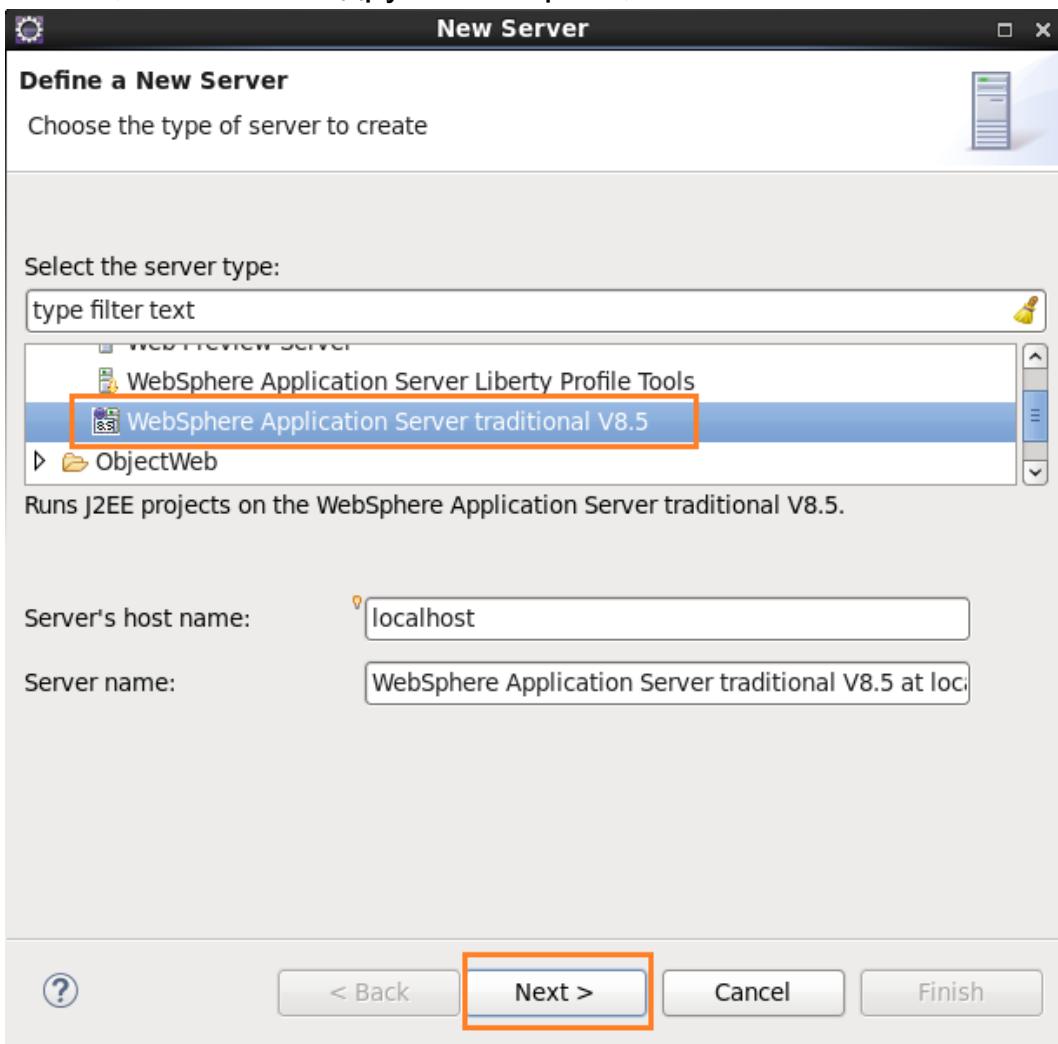
2. Настройте сервер для развертывания и отладки приложений.

а. Перейдите в представление **Servers**.

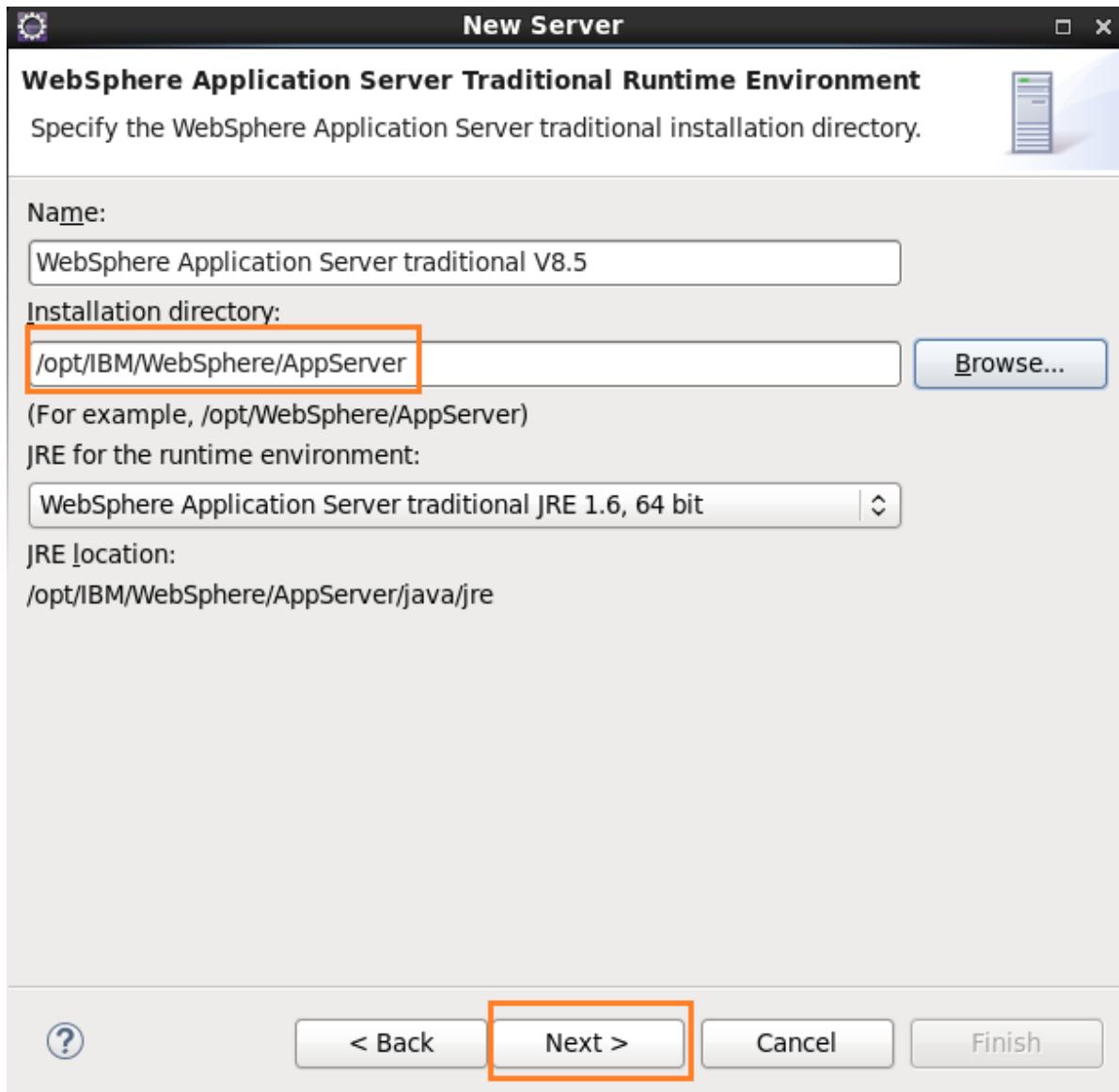
b. Нажмите правой кнопкой мыши в этом представлении и выберите **New -> Server**.



c. Выберите тип сервера **WebSphere Application Server traditional V8.5** и, не изменяя других настроек, нажмите **Next**.

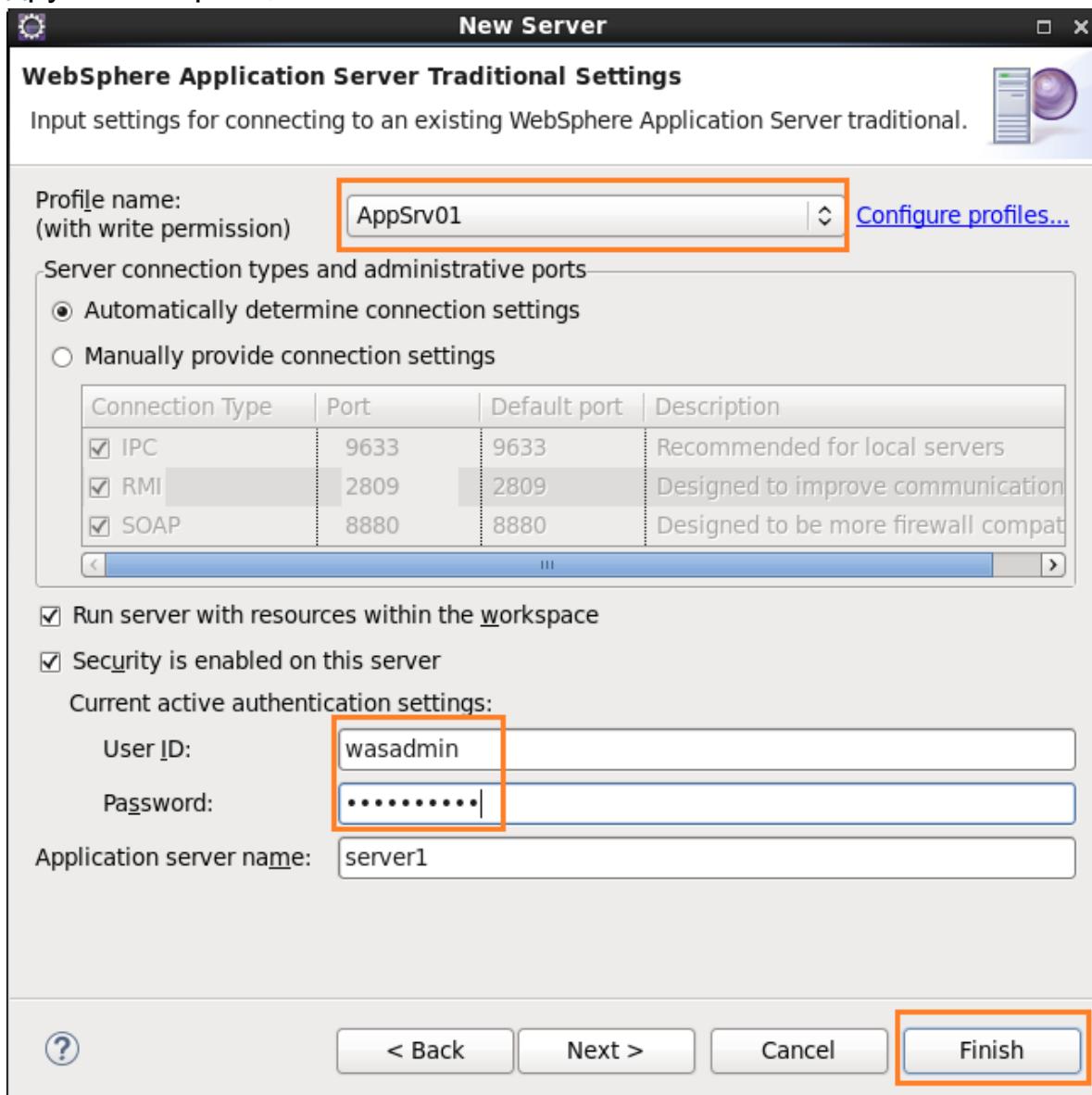


- d. Укажите каталог, в который установлен сервер приложений:
/opt/IBM/WebSphere/AppServer, после чего нажмите **Next**.

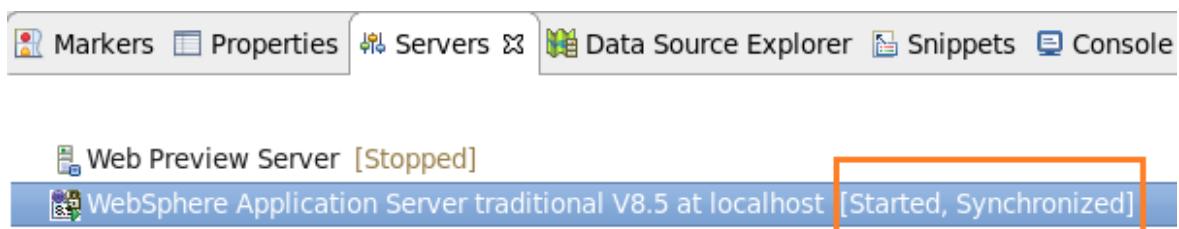


- e. При появлении всплывающего окна о потенциальных сложностях при подключении сервера (если это окно появится) нажмите **OK**, но обратите на это внимание вашего инструктора.
f. Убедитесь, что в настройках профиля указан **AppSrv01**, укажите пароль для пользователя **wasadmin – web1sphere**. Не меняя

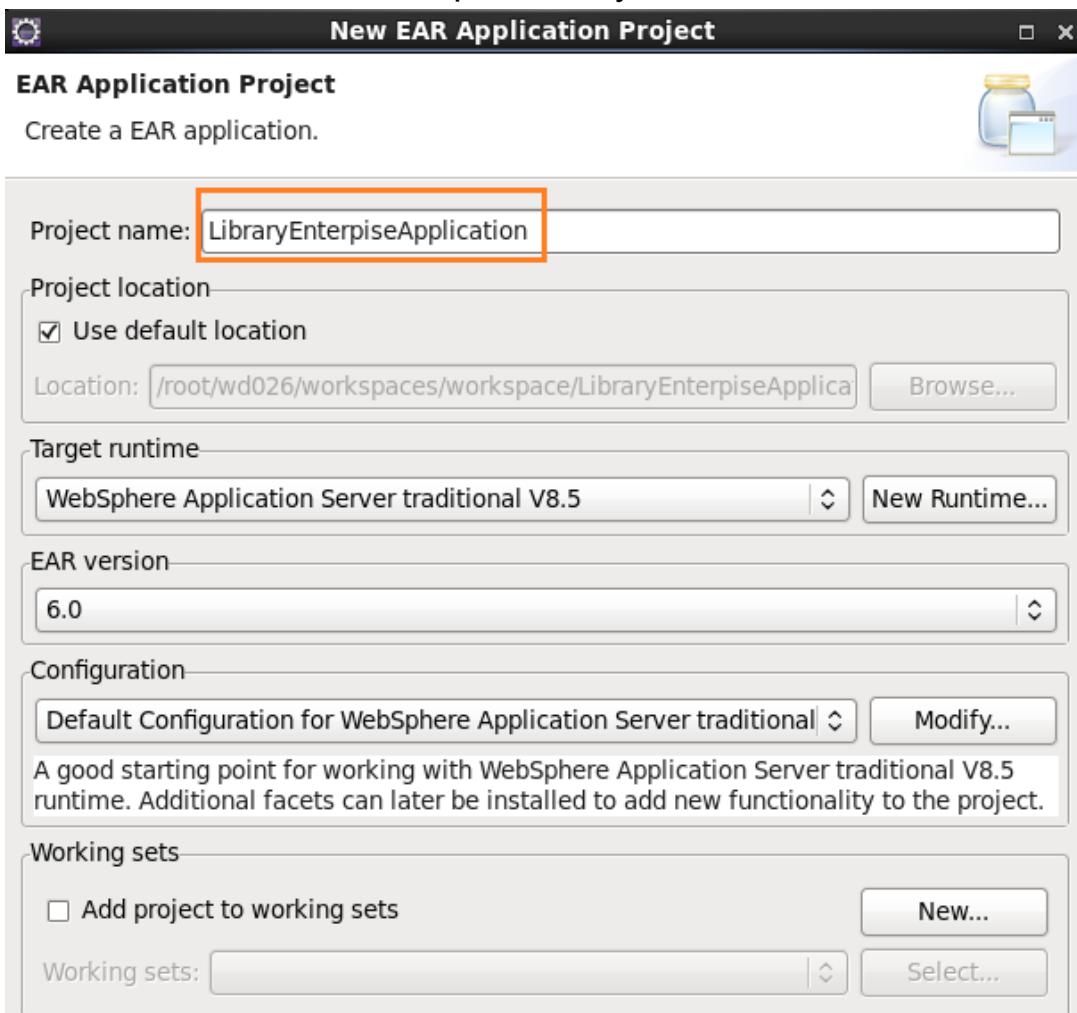
других настроек, нажмите **Finish**.



- g. В представлении **Servers** выберите только что созданный сервер, нажмите по нему правой кнопкой мыши, выполните действие **Start**.
- h. Дождитесь появления в логе сервера (представление **Console**) сообщения **Server server1 open for e-business** и вернитесь в представление Servers. Убедитесь, что сервер перешел в состояние **Started**.

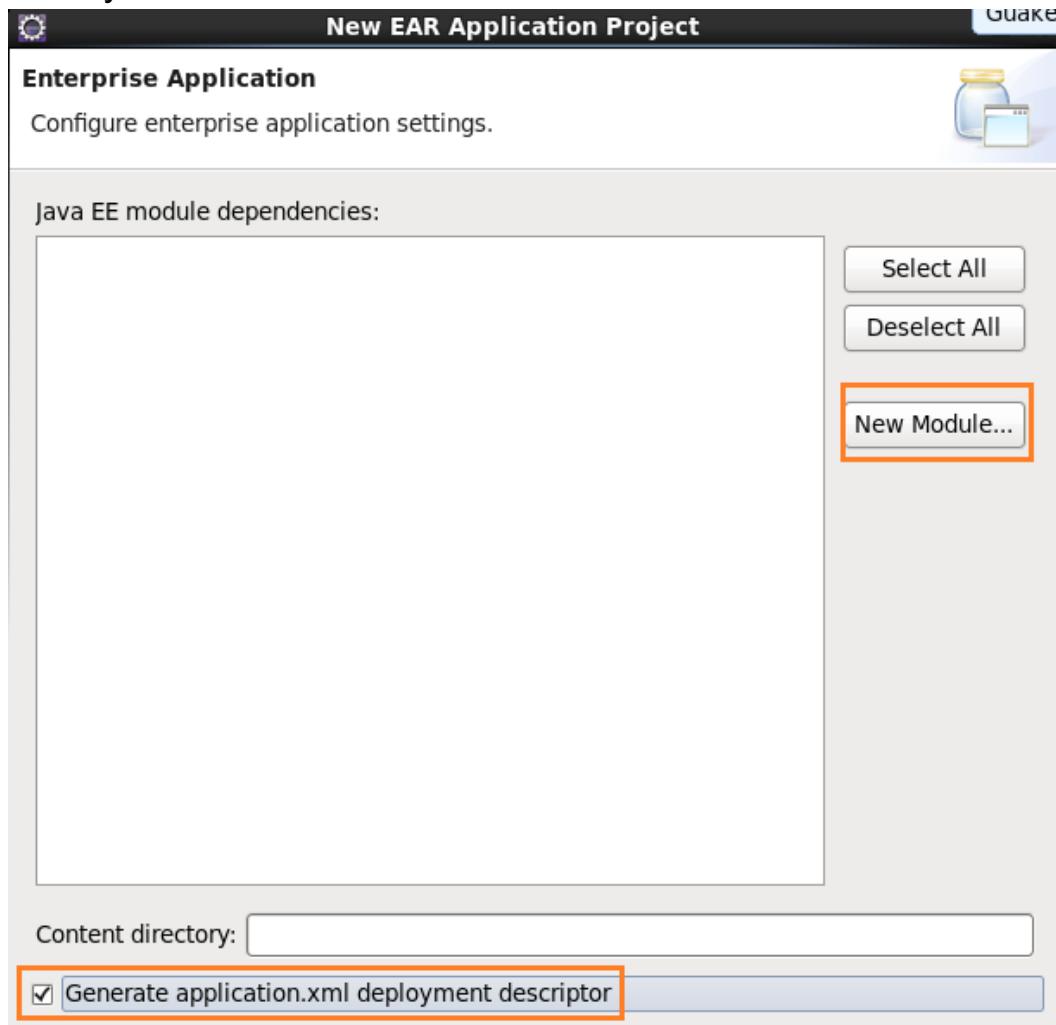


3. Создайте Enterprise приложение под названием LibraryEnterpriseApplication. Это приложение должно включать в себя Web модуль LibraryWebProject.
- a. Перейдите в меню **File -> New -> Enterprise Application Project**.
Откроется мастер по созданию соответствующего проекта.
- b. Укажите **LibraryEnterpriseApplication** в качестве имени проекта.
- c. Укажите **WebSphere Application Server traditional V8.5** в качестве среды для запуска приложения.
- d. Выберите формат EAR файла версии **6.0**.
- e. Оставьте остальные настройки по умолчанию и нажмите **Next**.



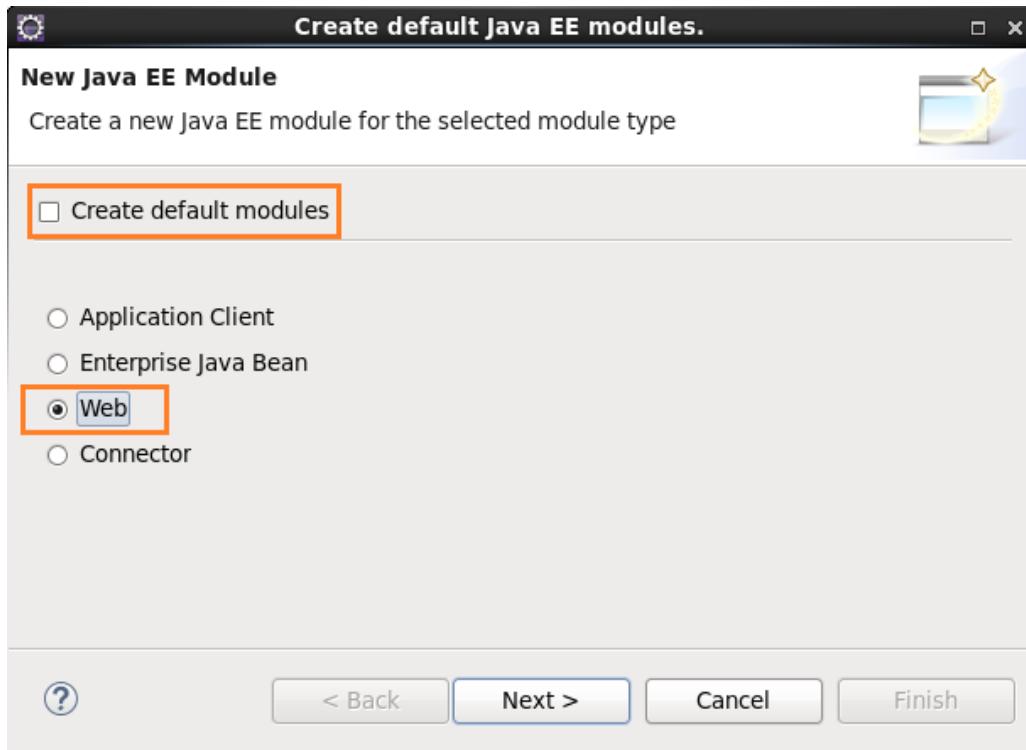
- f. В следующем интерфейсе установите опцию **Generate application.xml deployment descriptor**. После чего нажмите

кнопку **New module...**

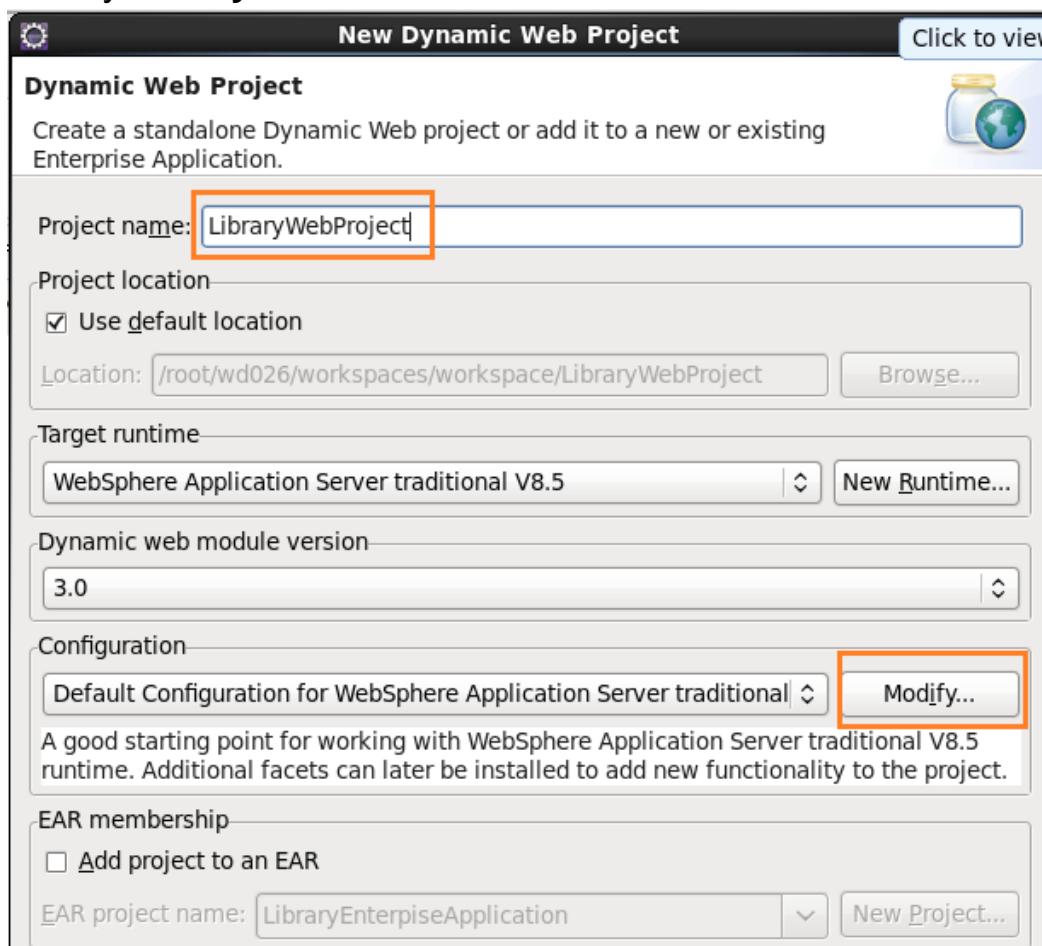


- g. Отключите опцию **Create default modules**, так как она предполагает создание нескольких модулей разных типов, которые сейчас не требуются.

h. Выберите опцию **Web** и нажмите **Next**.



- i. Укажите имя Web проекта **LibraryWebProject**.
- j. Поменяйте конфигурацию для работы приложения: нажмите кнопку **Modify**.



k. Выберите **Default style sheet (CSS file)** и нажмите **OK**.

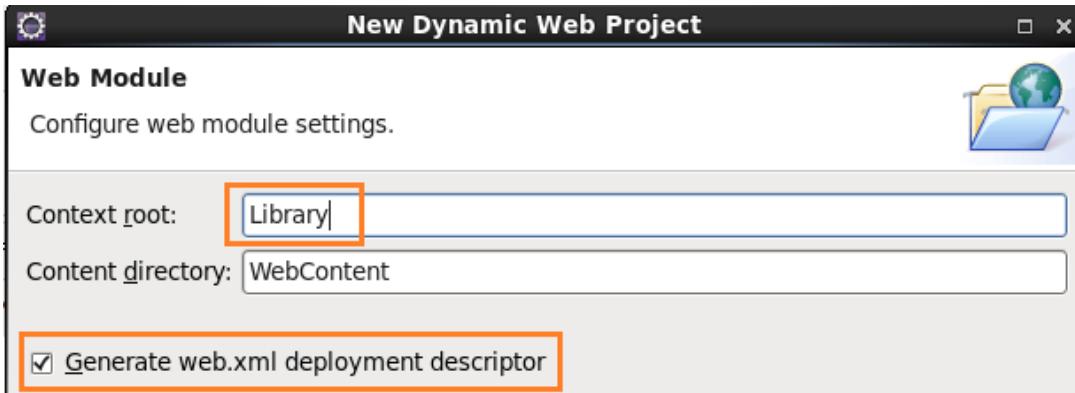
Project Facet	Version
Axis2 Web Services	
Context and dependency injection (CDI)	1.0
CXF 2.x Web Services	1.0
<input checked="" type="checkbox"/> Default style sheet (CSS file)	1.0
Default synchronization policy for CVS repository	1.0
<input checked="" type="checkbox"/> Dynamic Web Module	3.0
<input checked="" type="checkbox"/> Java	1.6
JavaScript	1.0
JavaServer Faces	2.2
JAX-RS (REST Web Services)	1.1
JAXB	2.2
JPA	2.1
OSGi Bundle	
WebDoclet (XDoclet)	1.2.3
WebSphere Web (Co-existence)	8.5
<input checked="" type="checkbox"/> WebSphere Web (Extended)	8.5

l. В интерфейсе создания динамического Web проекта нажмите **Next**.

m. Нажмите **Next**.

n. Укажите **Context Root** для приложения **Library**.

o. Выберите опцию **Generate web.xml deployment descriptor**.



p. Нажмите **Finish**. После этого среда разработки предложит переключиться в перспективу для Web разработки. Ответьте **Yes** и установите опцию **Remember my decision**.

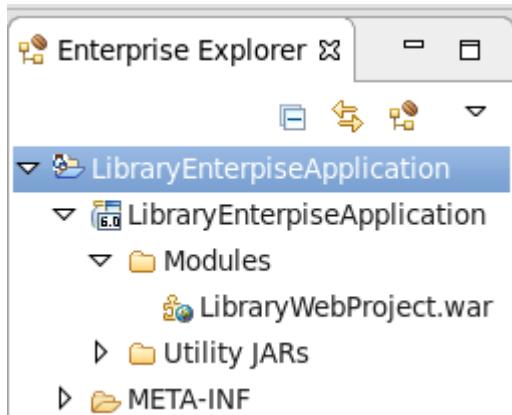
q. После возвращения в интерфейс создания приложения убедитесь, что в списке зависимостей появился новый Web модуль и нажмите **Finish**.

r. Отклоните предложение перейти в перспективу для Java EE разработки.

4. Посмотрите структуру созданных проектов.

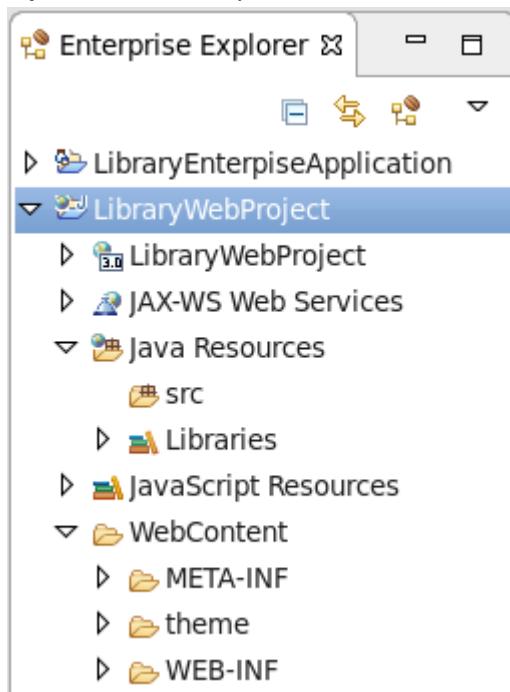
a. В представлении **Enterprise Explorer** найдите проект **LibraryEnterpriseApplication**. Артефакт с тем же названием на

следующем уровне иерархии – это Deployment Descriptor (дескриптор развертывания) приложения. Если открыть его, будет запущен редактор для работы с этим артефактом. Если же в представлении Enterprise Explorer раскрыть его, то можно посмотреть перечень модулей, входящих в приложение.



b. Найдите в этом же представлении проект **LibraryWebProject**.

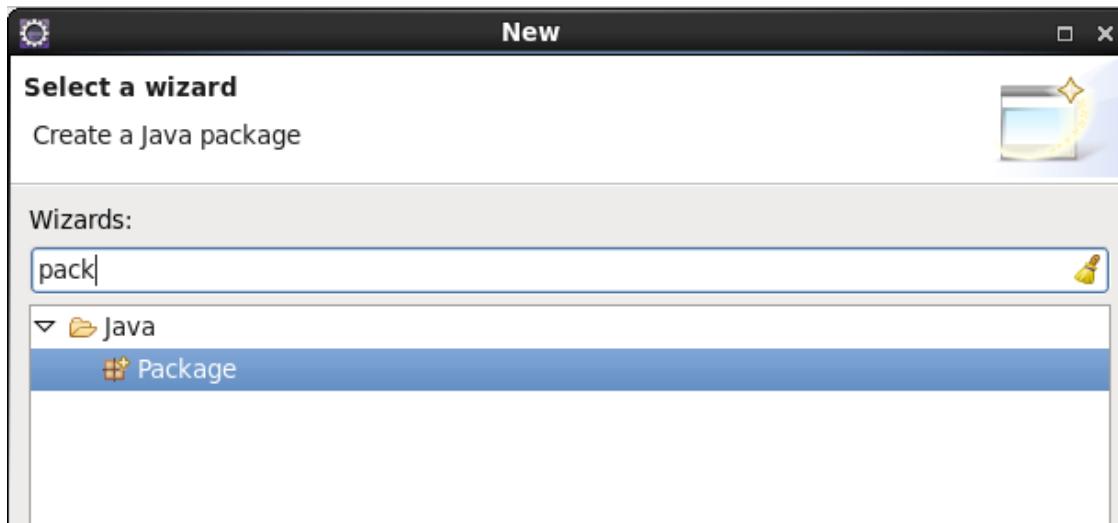
Посмотрите его структуру. В каталоге **Java Resources** располагаются создаваемые Java классы. В каталоге **WebContent** располагаются JSP страницы и статическое содержимое приложения (HTML, CSS, JavaScript, изображения и т.п.).



Раздел 2. Создание Servlet

Теперь после создания необходимых проектов, можно создавать необходимые компоненты в рамках Web проекта.

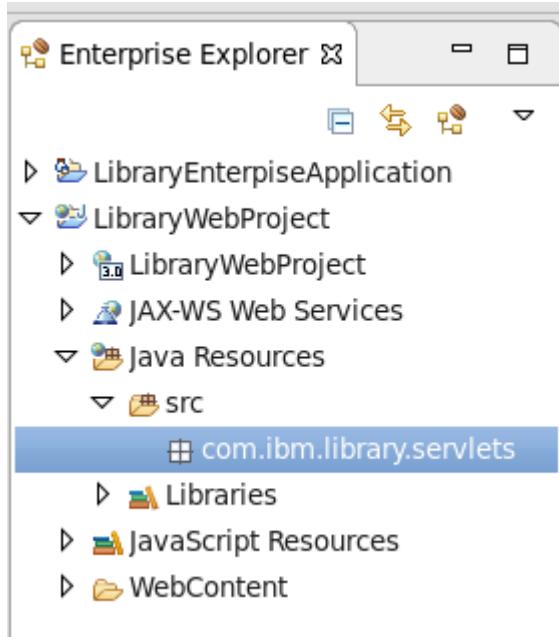
1. Перейдите в перспективу для Web разработки, если вы еще ее не активировали.
 - a. В главном меню выберите **Window -> Perspective -> Open Perspective -> Other... -> Web.**
2. Создайте Java package **com.ibm.library.servlets** в Web проекте.
 - a. В главном меню выберите **File -> New -> Other**. Откроется мастер создания нового артефакта.
 - b. Начните вводить **package** в поле фильтра, чтобы увидеть нужный тип артефакта.



- c. Выберите **Package** и нажмите **Next**.
- d. Укажите каталог для пакета **LibraryWebProject/src** и название **com.ibm.library.servlets**.

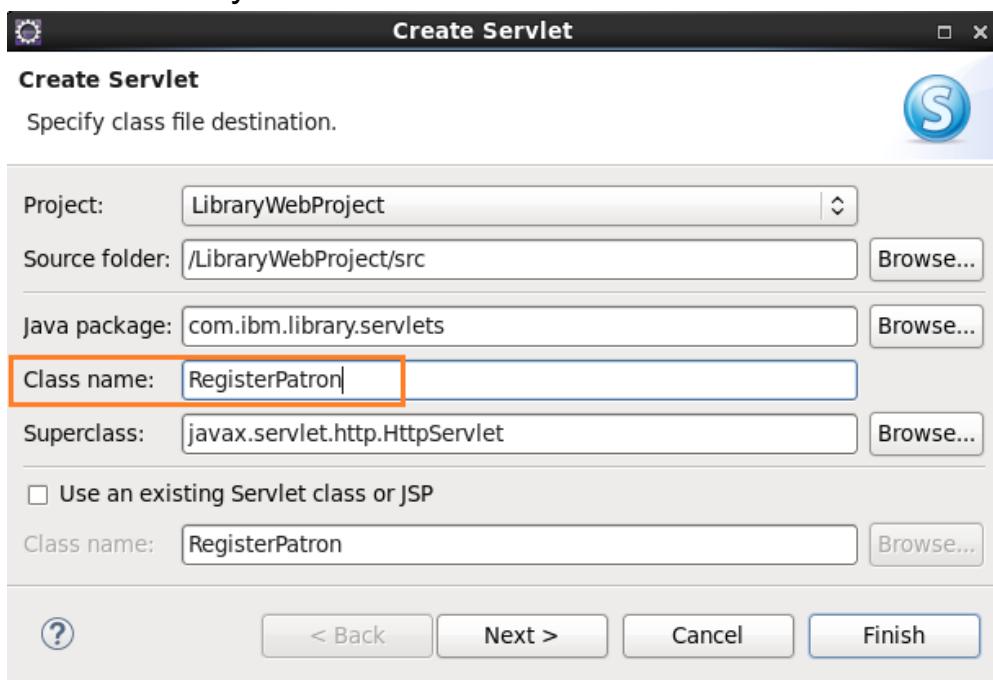


е. Нажмите **Finish**. Убедитесь, что соответствующий пакет появился в проекте.

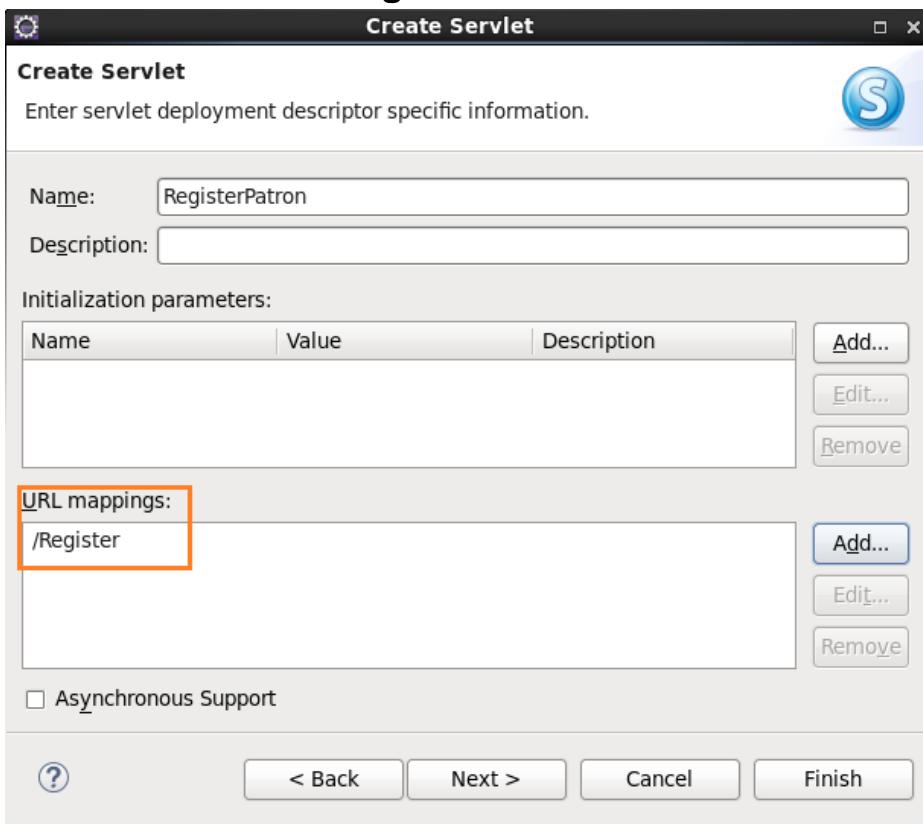


3. Создайте новый сервлет в созданном пакете. Он будет расширять класс **javax.servlet.http.HttpServlet** и реализовывать интерфейс **javax.servlet.Servlet**. Также нужно реализовать методы **doGet()** и **doPost()** и сделать сервлет доступным по URI **/Register**.

- В представлении **Enterprise Explorer** нажмите правой кнопкой мыши по пакету **com.ibm.library.servlets** и выберите **New -> Servlet**. Откроется мастер создания нового сервлета.
- В поле **Class name** укажите **RegisterPatron**. Оставьте остальные значения по умолчанию и нажмите **Next**.

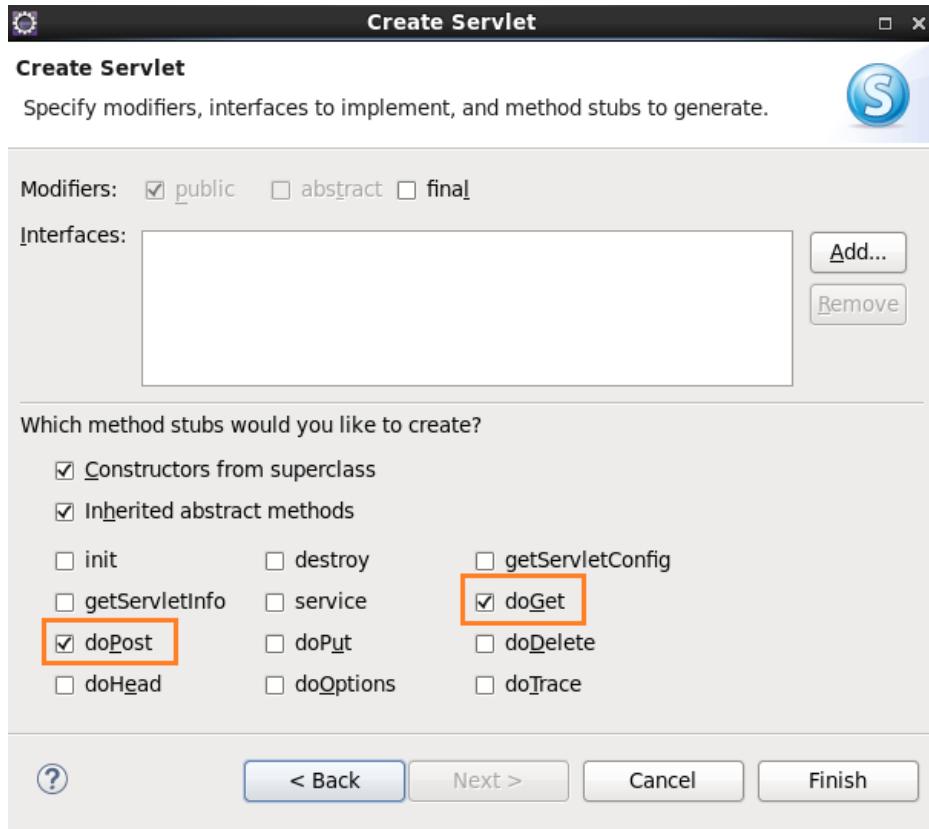


- c. В следующем интерфейсе с создаваемым сервлетом сопоставляется URL, по которому он будет принимать запросы. Значение по умолчанию необходимо поменять. Выберите **URL mapping – /RegisterPatron** и нажмите кнопку **Edit...**.
- d. Измените URL на **/Register** и нажмите **OK**.



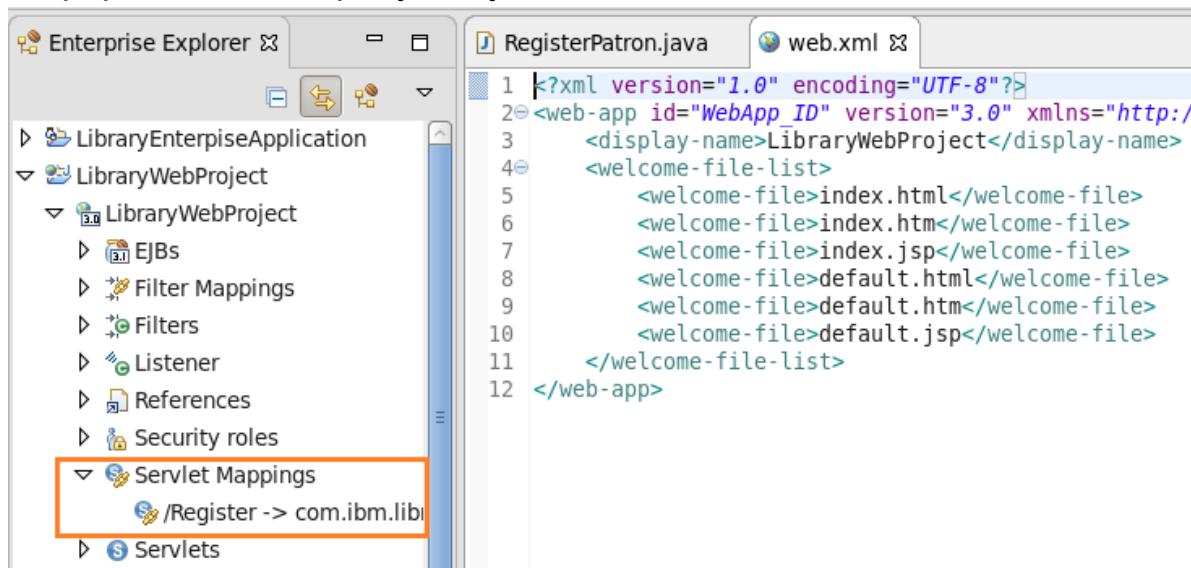
- e. Нажмите **Next**.

- f. В следующем интерфейсе убедитесь, что в новом сервлете будут определены методы **doGet** и **doPost**. Нажмите **Finish**.



4. Посмотрите deployment descriptor Web модуля.

- a. В перспективе **Enterprise Explorer** откройте дескриптор развертывания Web модуля.
- b. Обратите внимание на то, что в самом файле дескриптора нет указаний на то, что сервлету сопоставлен некий URL, по которому он должен отвечать на запросы. При этом если раскрыть иерархию deployment descriptor в Enterprise Explorer, то эта информация там присутствует.



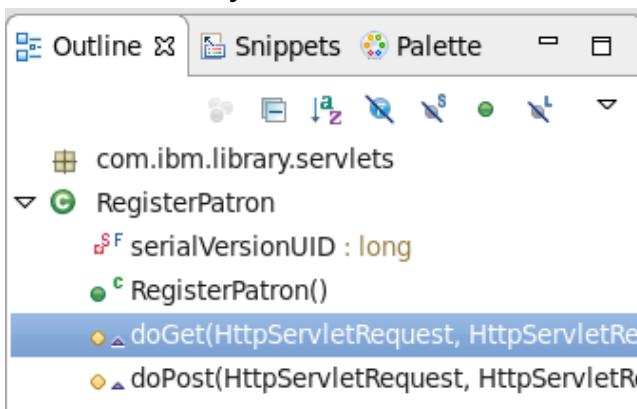
Это обусловлено тем фактом, что URL Mapping по умолчанию в Eclipse для этой (6-ой) версии Java EE сохраняется не в дескриптор развертывания, а в сам код сервлета в виде аннотации:

```
RegisterPatron.java
1 package com.ibm.library.servlets;
2
3 import java.io.IOException;
4
5 /**
6  * Servlet implementation class RegisterPatron
7  */
8 @WebServlet("/Register")
9 public class RegisterPatron extends HttpServlet {
10     private static final long serialVersionUID = 1L;
11
12     /**
13      * @param request
14      * @param response
15      * @throws ServletException
16      * @throws IOException
17  }
```

Раздел 3. Создание бизнес логики.

Теперь необходимо определить логику работы созданного сервлета. В данной реализации будет генерироваться случайное значение идентификатора читателя, которое будет включаться в сообщение для библиотекаря, сигнализирующее о создании клиента. Реальная логика для регистрации клиента будет реализована в последующих упражнениях.

1. Откройте файл **RegisterPatron.java**, если он еще не открыт для редактирования. Сделать это можно с помощью двойного щелчка мыши по названию файла в представлении **Enterprise Explorer**.
2. Реализуйте основные методы сервлета.
 - a. В представлении **Outline** выберите метод **doGet()**. Автоматически этот метод будет выделен в коде сервлета.



- b. Замените текущую реализацию метода (комментарий и вызов метода для работы с объектом response) на следующую строчку:

```
processRequest(request, response);
```

Аргументами для вызова этого метода являются объекты, описывающие исходный HTTP запрос и результирующий HTTP ответ.

- c. Аналогичным образом замените тело метода **doPost()**. Для его быстрого нахождения можно воспользоваться представлением Outline или встроенным интерфейсом Outline, доступным по нажатию **Ctrl + O**.
- d. По итогам должен получиться следующий код для переопределенных методов:

```
24
25  /**
26   * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
27  */
28  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
29      processRequest(request, response);
30 }
31
32  /**
33   * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
34  */
35  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
36      processRequest(request, response);
37 }
```

Предупреждения рядом с вызовами метода **processRequest** возникают из-за того, что этот метод еще не был реализован. Зеленые стрелки рядом с объявлением методов показывают, что эти методы переопределяют методы, описанные в классе-родителе.

3. Исправьте ошибку с неопределенным методом **processRequest**.
- Нажмите левой кнопкой мыши по лампочке с сигналом об ошибке рядом с методом **doGet()**. Появятся предложения со стороны Eclipse, таким образом эту проблему можно решить.
 - Выберите опцию **Create method ‘processRequest(...)**’ дважды кликнув по ней.
 - Соответствующий метод появляется в классе сервлета.
Сохраните изменения (**Ctrl + S**).
4. Реализуйте метод **processRequest()**. Этот метод будет использовать объект **PrintWriter** из параметра **HttpServletResponse** для отображения идентификатора клиента, который будет генерироваться с помощью метода **generateId()** класса

LibraryIdGenerator. Класс с соответствующим методом будут созданы позже. После генерации идентификатора он будет включен в HTML, который должен отобразиться в интерфейсе библиотекаря. После завершения этого шага, в коде должна остаться одна ошибка: не разрешенное имя класса **LibraryIdGenerator**.

а. В методе `processRequest()` замените его текущую реализацию на следующий код:

```
PrintWriter out = response.getWriter();
```

б. Щелкните по лампочке с сообщением об ошибке рядом с этой строчкой и выберите опцию: **Import 'PrintWriter' (java.io)** затем выберите вариант **Add throws declaration**. Это добавит соответствующую строчку к объявлению класса – для выбрасывания ошибок со вводом/выводом.

с. Предупреждение о том, что эта переменная не используется можно проигнорировать.

д. Добавьте в метод строчку для генерации идентификатора:

```
int id = LibraryIdGenerator.generateId();
```

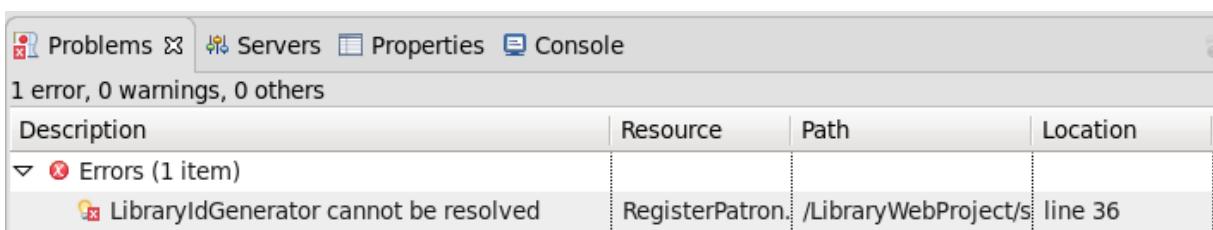
е. Добавьте следующие строчки для формирования HTML, который будет отображен в интерфейсе библиотекаря:

```
out.println("<HTML>");  
out.println("<HEAD><TITLE>Patron  
Added</TITLE></HEAD>");  
out.println("<LINK rel=\"stylesheet\""  
type=\"text/css\" "+  
"href=\"/Library/theme/master.css\"");  
out.println("<BODY>");  
out.println("<P>");  
out.println("Patron with id " + id + " has been  
added.</P>");  
out.println("</BODY>");  
out.println("</HTML>");
```

f. Итоговый код метода должен получиться таким:

```
33
34  private void processRequest(HttpServletRequest request, HttpServletResponse response) throws IOException {
35      PrintWriter out = response.getWriter();
36      int id = LibraryIdGenerator.generateId();
37
38      out.println("<HTML>");
39      out.println("<HEAD><TITLE>Patron Added</TITLE></HEAD>");
40      out.println("<LINK rel=\"stylesheet\" type=\"text/css\" " +
41                  "href=\"/Library/theme/master.css\">");
42      out.println("<BODY>");
43      out.println("<P>");
44      out.println("Patron with id " + id + " has been added.</P>");
45      out.println("</BODY>");|  
46      out.println("</HTML>");  
47
48  }
```

g. Сохраните внесенные изменения. Убедитесь, что единственная ошибка связана с неразрешенным именем класса **LibraryIdGenerator**. Сделать это можно в представлении **Problems**.



Раздел 4. Создание класса **LibraryIdGenerator** и метода **generateId()**.

1. Создайте новый класс **LibraryIdGenerator** в пакете com.ibm.libray.servlets.

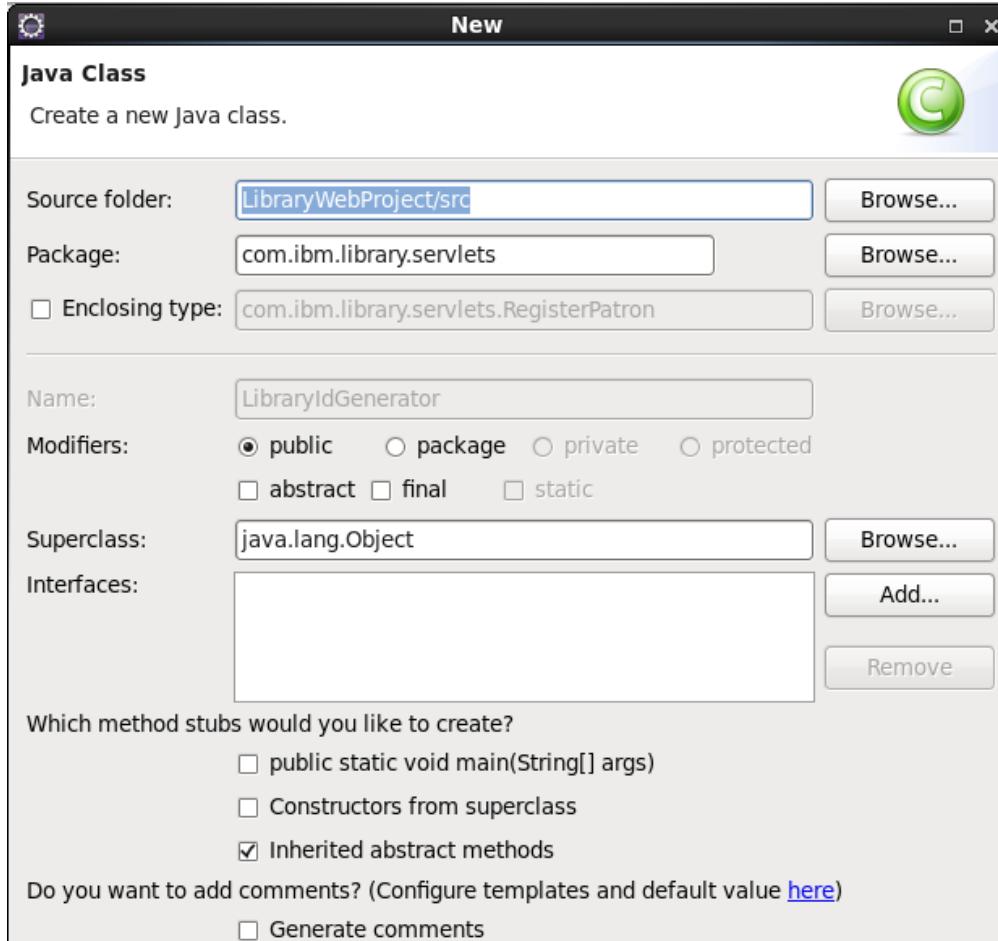
а. Нажмите по лампочке рядом с ошибкой в коде RegisterPatron.

б. Выберите опцию **Create class 'LibraryIdGenerator'**

```
33
34  private void processRequest(HttpServletRequest request, HttpServletResponse response) throws IOException {
35      PrintWriter out = response.getWriter();
36      int id = LibraryIdGenerator.generateId();  
37          ^-----|  
38          | Create class 'LibraryIdGenerator'  
39          | Create constant 'LibraryIdGenerator'  
40          | Create local variable 'LibraryIdGenerator'  
41          | Change to 'LibraryValidator' (com.ibm.websphere.validation)  
42          | Create field 'LibraryIdGenerator'  
43      out.print("id=" + id);  
44      out.print("<html>");  
45      out.print("<head>");  
46      out.print("<title>Patron Added</title>");  
47      out.print("<link href=\"/Library/theme/master.css\" type=\"text/css\" />");  
48      out.print("<body>");  
49      out.print("<p>");  
50      out.print("Patron with id " + id + " has been added.");  
51      out.print("</p>");  
52      out.print("</body>");  
53      out.print("</html>");  
54  }
```

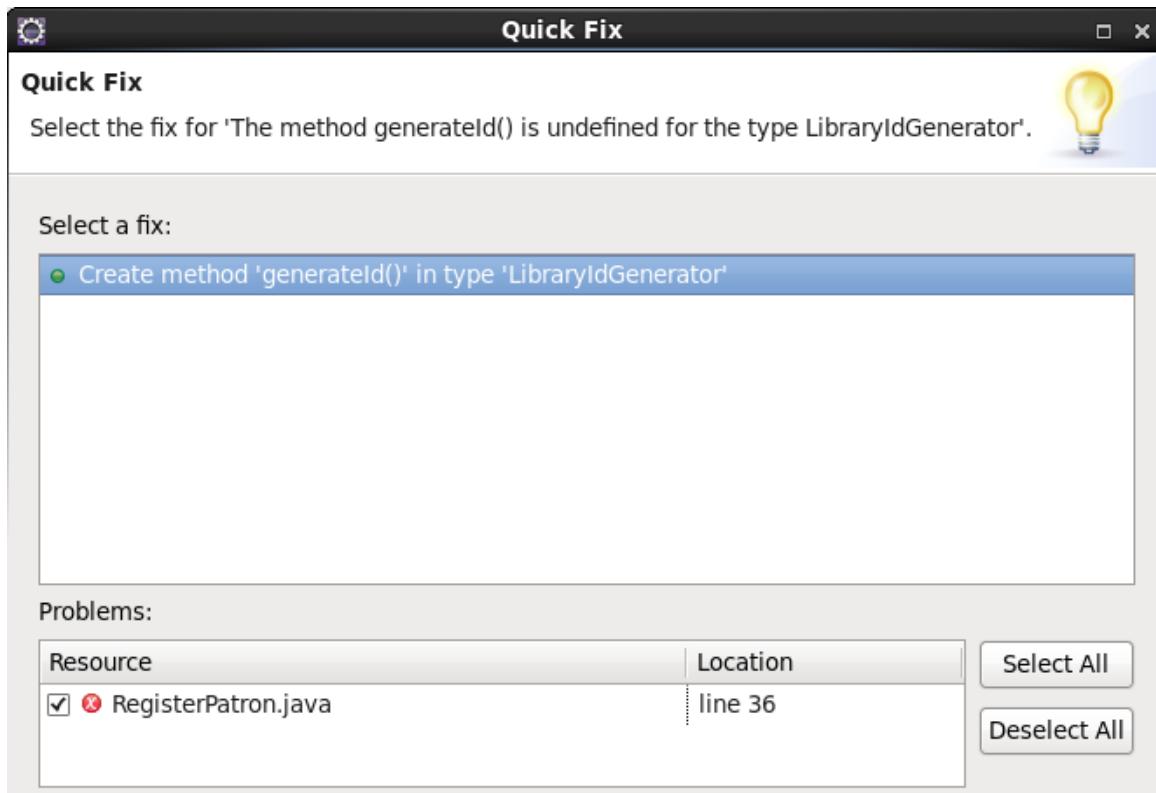
с. Открывается мастер создания нового класса. Согласитесь со всеми настройками по умолчанию и нажмите **Finish**.

Соответствующий класс будет создан и открыт в редакторе.



2. Реализуйте статический метод generateId(), возвращающий целочисленное значение.
 - a. Перейдите в представление **Problems**. Выделите единственную ошибку, которая говорит о том, что метод **generateId()** не определен (**The method generateId() is undefined for the type LibraryIdGenerator**).
 - b. Нажмите по ошибке правой кнопкой мыши и выберите **Quick Fix**.

- c. Согласитесь с вариантом, предложенным по умолчанию, и нажмите **Finish**. Это приведет к созданию необходимого метода.



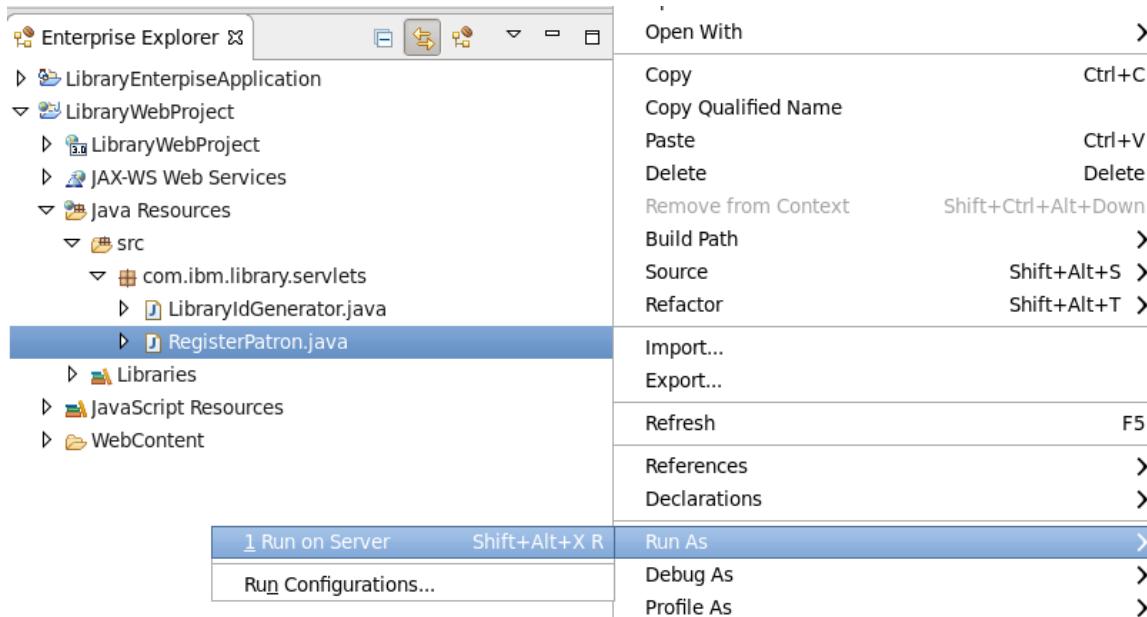
- d. Соответствующий метод добавлен в класс b. Обратите внимание на то, что он возвращает значение **0**.
- e. **Сохраните** сделанные изменения. Убедитесь, что ошибки пропали.

Раздел 5. Тестирование сервлета

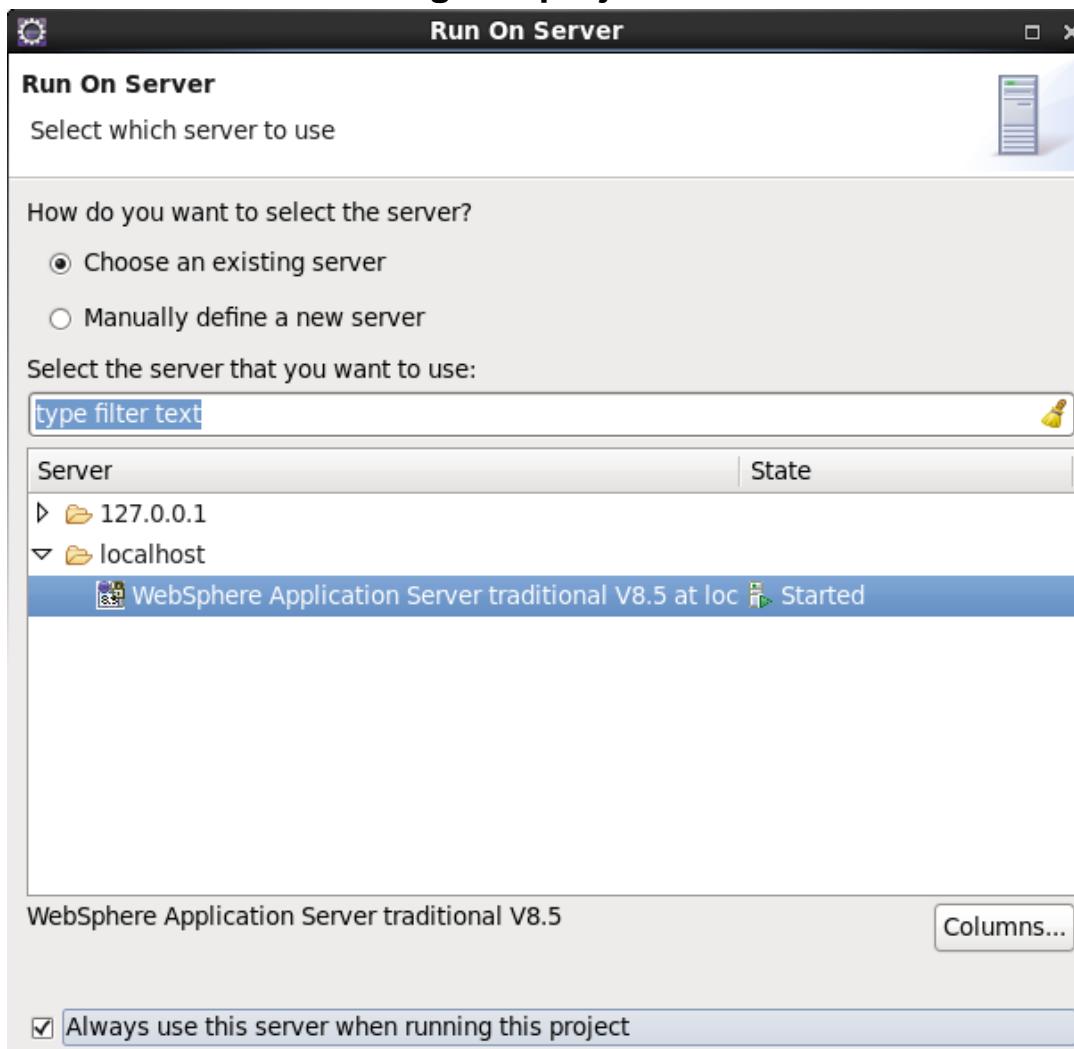
Для тестирования сервлета его нужно будет развернуть на сервере, и обратиться по выбранному адресу через выбранный адрес.

1. Запустите сервлет на сервере.
 - a. Выберите в представлении Enterprise Explorer сервлет: **LibraryWebProject -> Java Resources -> src -> com.ibm.library.servlets -> RegisterPatron.java**. Нажмите по нему

правой кнопкой мыши и выберите **Run As -> Run on server**.

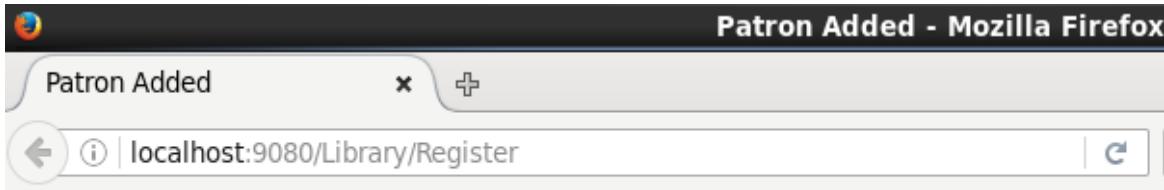


- b. На странице выбора сервера согласитесь с вариантом по умолчанию (использовать **существующий сервер WebSphere Application Server traditional V8.5**). Выберите опцию **Always use this server when running this project**.



с. Нажмите **Finish**.

д. Через некоторое время после успешного запуска приложения (об этом можно судить по представлению **Console**) автоматически будет открыт браузер, в котором будет выполнено обращение к серверу, который вернет ожидаемое значение.



Patron with id 0 has been added.

2. Измените метод **generateId()** для возврата псевдослучайного значения.

а. Замените в теле метода строчку, возвращающую константу на следующую:

```
return (new  
Double(Math.floor(1000000*Math.random()))).intValue()  
);
```

б. Сохраните сделанные изменения.

с. Проведите повторное тестирование. Для этого достаточно обновить страницу, так как приложение на сервере обновится автоматически.

Конец упражнения

Упражнение 2. Создание базы данных библиотеки

О чём это упражнение:

В этой лабораторной работе вы импортируете и запустите несколько Java классов, которые создадут и заполнят данными базу для работы библиотеки.

Что вы должны будете сделать:

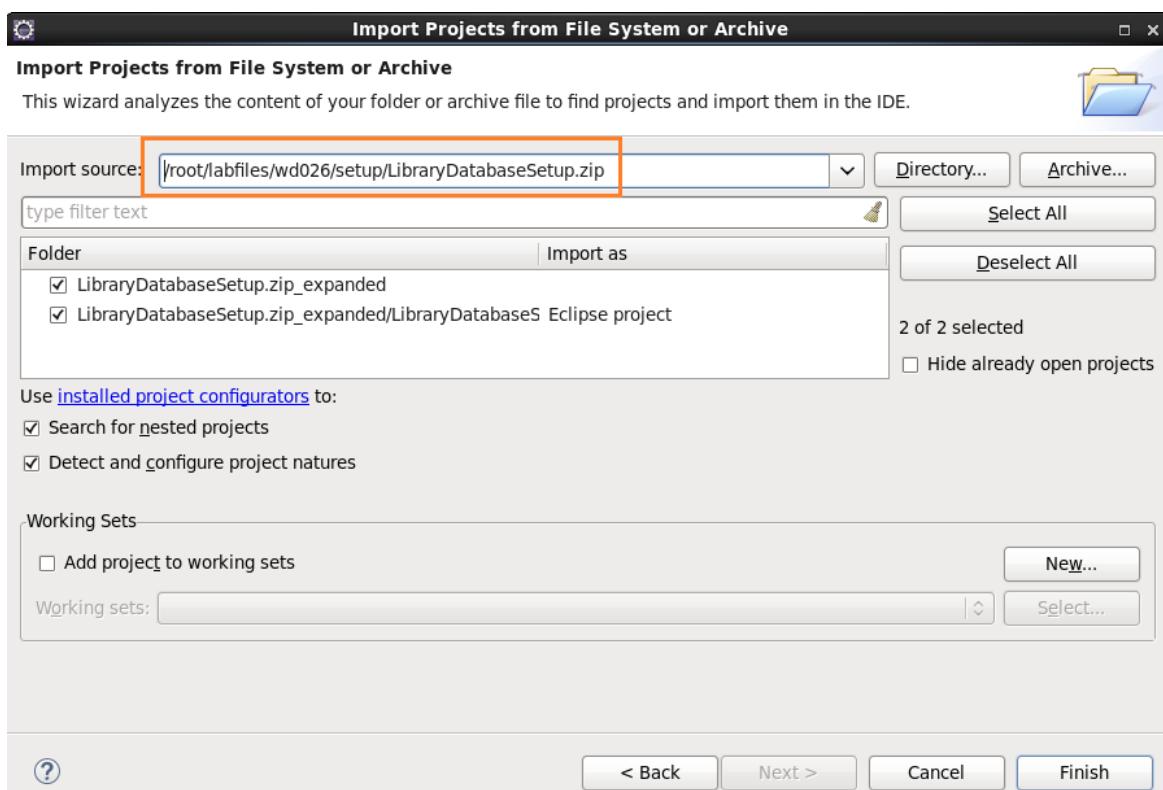
- Импортировать существующий проект в Eclipse
- Запустить приложение
- Создать и заполнить базу данных Derby. Derby – СУБД, которая сходит в комплект поставки сервера приложений или среды разработки. Подходит для учебных и демонстрационных целей.

Раздел 1. Импорт проекта и создание базы

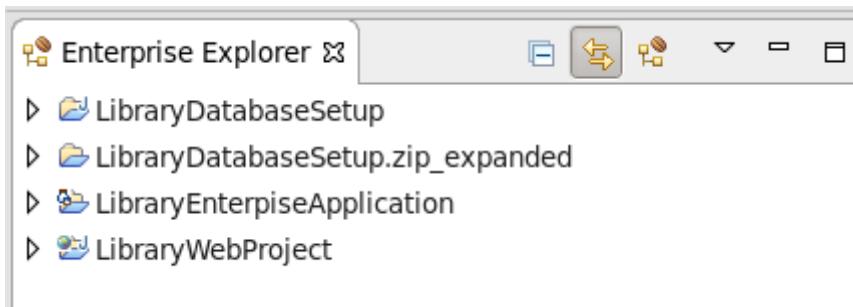
В данном разделе импортируется проект, содержащий все необходимое для создания и заполнения базы данных.

1. Импортируйте архив с проектом для создания базы.

- a. Перейдите в меню **File -> Import...** Открывается мастер для импорта ресурсов.
- b. Выберите **General -> Projects from Folder or Archive**.
- c. Нажмите **Next**.
- d. Укажите путь до архива
/root/labfiles/wd026/setup/LibraryDatabaseSetup.zip
- e. Нажмите **Finish**.



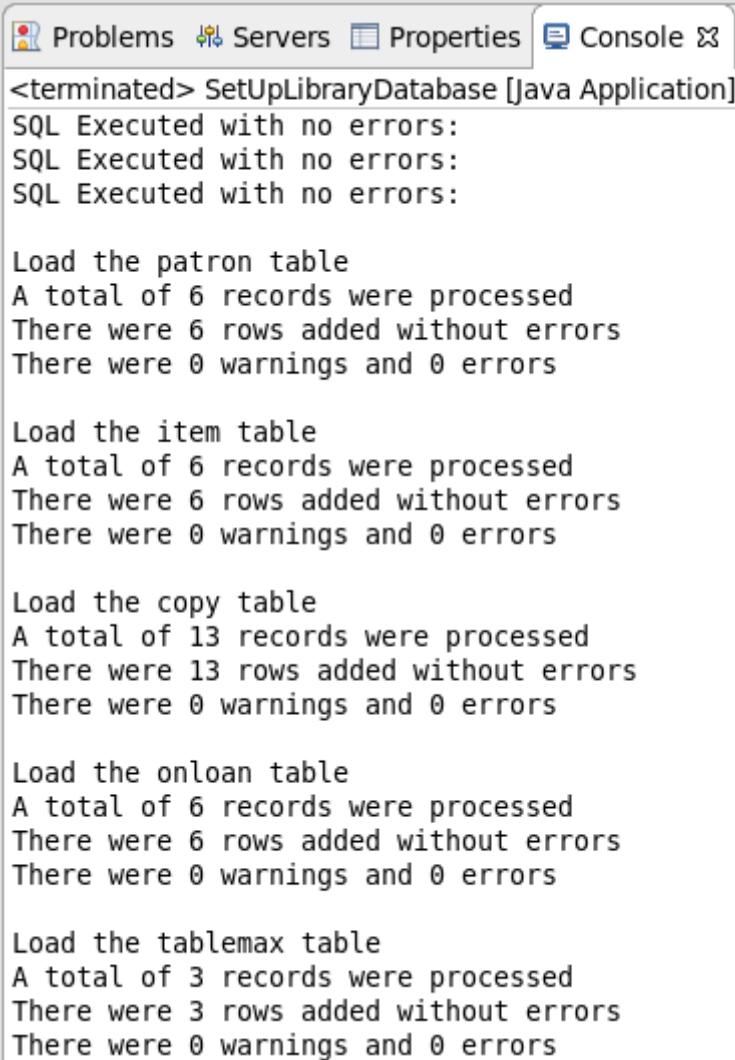
- f. Убедитесь, что в рабочем пространстве в представлении **Enterprise Explorer** появилось два новых проекта:
LibraryDatabaseSetup и **LibraryDatabaseSetup.zip_expanded**.



2. Запустите приложение для создания базы.

- a. В представлении Enterprise Explorer разверните пакет **LibraryDatabaseSetup -> driver**.
- b. Правой кнопкой мыши нажмите по файлу **SetUpLibraryDatabase.java**. Выберите **Run As -> Java Application**.
- c. Перейдите в представление **Console**. В консоли должен отобразиться перечень сообщений относительно создания и заполнения базы. Окончание листинга должно быть похоже на

фрагмент, показанный ниже:



The screenshot shows a Java application window titled "SetUpLibraryDatabase [Java Application]". The "Console" tab is selected. The output text is as follows:

```
<terminated> SetUpLibraryDatabase [Java Application]
SQL Executed with no errors:
SQL Executed with no errors:
SQL Executed with no errors:

Load the patron table
A total of 6 records were processed
There were 6 rows added without errors
There were 0 warnings and 0 errors

Load the item table
A total of 6 records were processed
There were 6 rows added without errors
There were 0 warnings and 0 errors

Load the copy table
A total of 13 records were processed
There were 13 rows added without errors
There were 0 warnings and 0 errors

Load the onloan table
A total of 6 records were processed
There were 6 rows added without errors
There were 0 warnings and 0 errors

Load the tablemax table
A total of 3 records were processed
There were 3 rows added without errors
There were 0 warnings and 0 errors
```

3. Это приложение можно запустить в любой момент времени по ходу лабораторного практикума для того чтобы пересоздать базу или вернуть ее в исходное состояние.

Конец упражнения

Упражнение 3. Сервлет с параметрами

О чём это упражнение:

В этой лабораторной работе вы доработаете сервлет RegisterPatron. Изменения позволяют регистрировать клиента в базе данных и решать следующие задачи:

- Обрабатывать входные данные: считывать значения введенных полей и проверять их.
- Обращаться к классам для создания соответствующего объекта в базе данных
- Отображать результат

Логику для взаимодействия с базой данных писать не нужно, она уже была разработана и импортируется в ходе этого упражнения в рабочее пространство. На ней не заостряется внимание, так как она не является темой для данного курса.

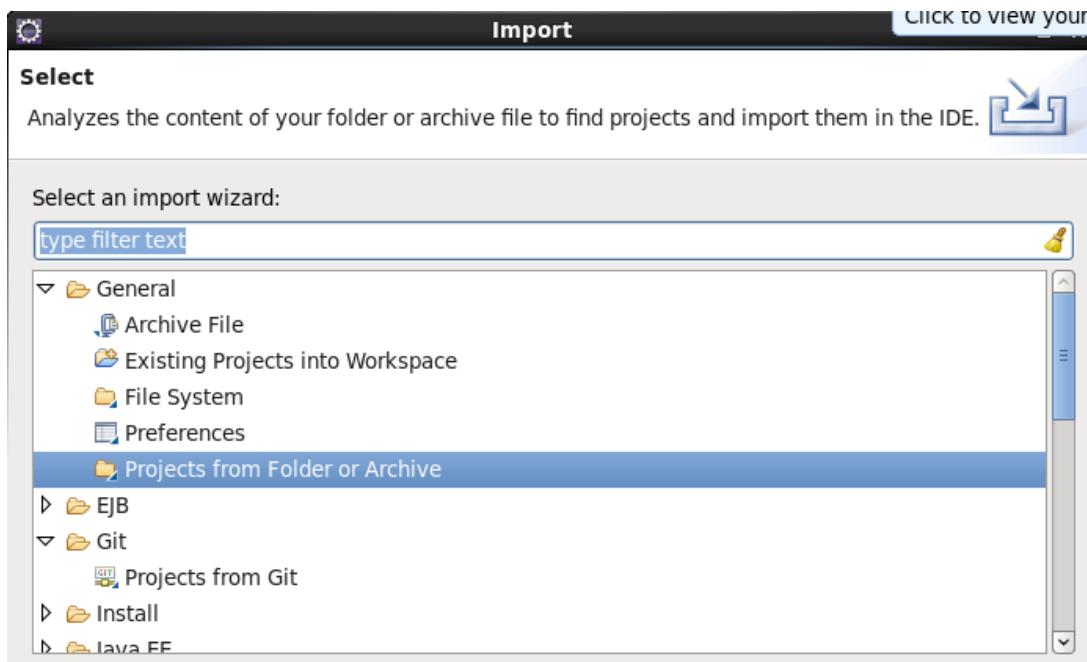
Что вы должны будете сделать:

- Импортировать проект
- Связать проект с другими проектами, находящимися в рабочем пространстве
- Переместить класс между проектами
- Импортировать HTML страницу
- Обновить логику сервлета
- Создать источник данных (data source)
- Протестировать сервлет

Раздел 1. Импорт проекта с логикой для взаимодействия с базой данных

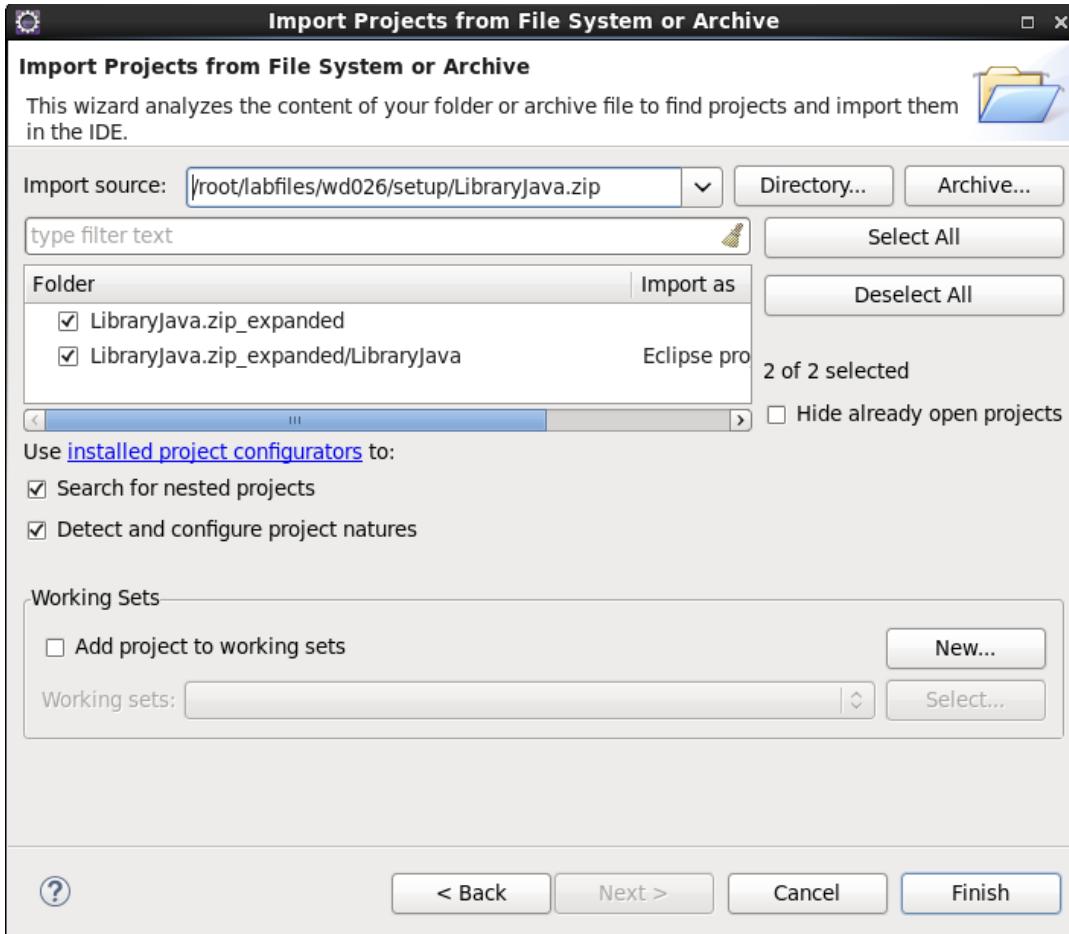
1. Импортируйте проект

- В главном меню выберите **File -> Import**. Откроется мастер импорта ресурсов.
- Выберите **General -> Projects from Folder or Archive**. Нажмите **Next**.



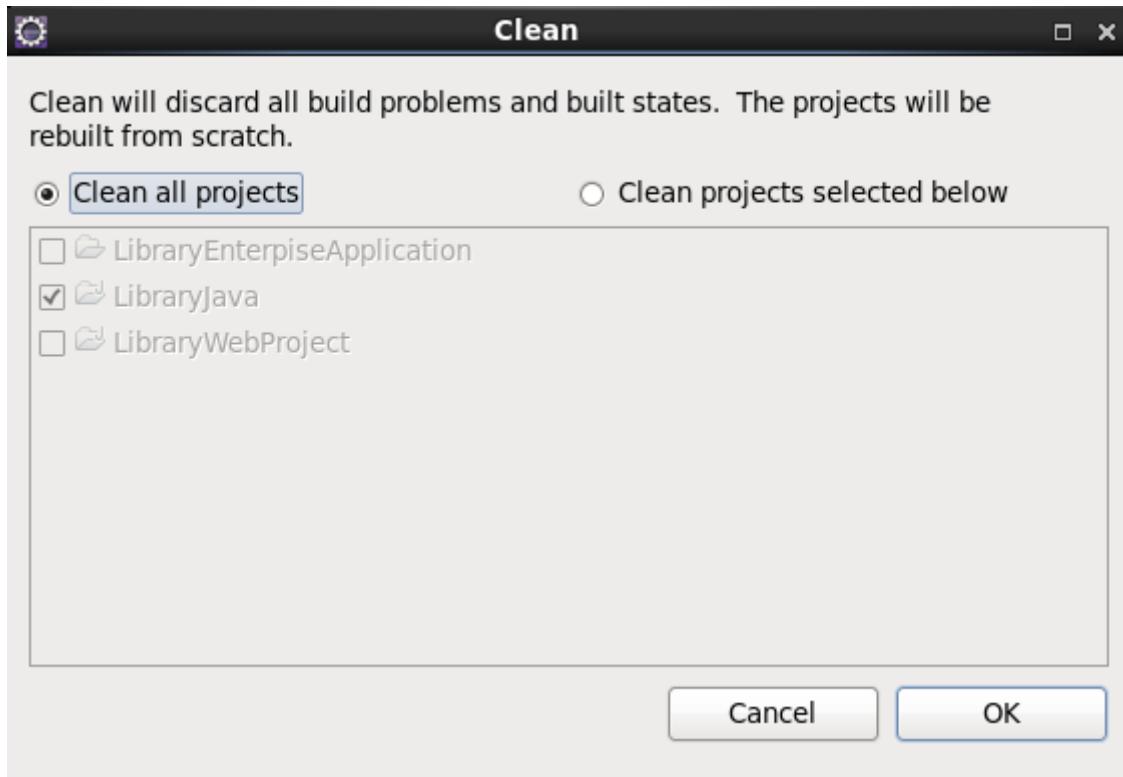
- Укажите путь до импортируемого архива (вы можете написать его самостоятельно или воспользоваться файловым проводником):
/root/labfiles/wd026/setup/LibraryJava.zip

d. Нажмите **Finish**.

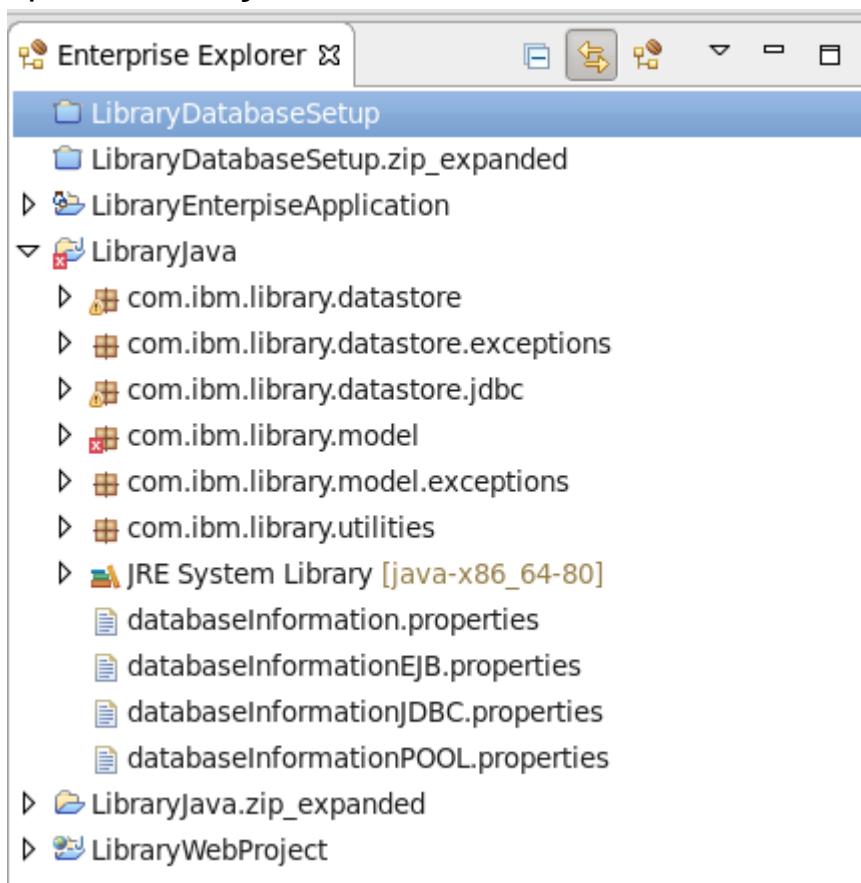


e. После успешного импорта в представлении **Problems** должна появиться 1 ошибка и несколько предупреждений. Ошибка связана с неразрешенным именем класса: **LibraryIdGenerator cannot be resolved**. Если эти ошибки не появляются в течение минуты, то следует обновить рабочее пространство, пересобрав проекты: для этого выберите в меню **Project -> Clean...** После чего почистите

все проекты, выбрав **Clean all projects** и нажав **OK**.



- f. По итогам импорта в рабочем пространстве должен появиться проект **LibraryJava** с пакетами, показанными на скриншоте.



Всего в этом проекте есть около 20 классов, разнесенных по различным пакетам. Все пакеты можно разделить на две большие

группы: **com.ibm.library.datastore** и **com.ibm.library.model**.

com.ibm.library.model содержит front-end классы API для работы с бизнес-объектами данного приложения. Именно к ним придется обращаться из создаваемых веб-приложений. В данном упражнении реально будет использоваться только класс Patron, так как один из его методов позволяет как раз зарегистрировать нового клиента. Классы этого пакета не работают напрямую с базой данных, а используют вместо этого другие классы из второй группы пакетов **com.ibm.library.datastore**. Здесь, например, есть классы для установления соединения (**ConnectionFactory**), для обработки конкретных запросов (**PatronDataStoreJDBC**) и т.п.

2. Переместите класс **LibraryIdGenerator** из проекта

LibraryWebProject в проект **LibraryJava**.

a. В представлении **Enterprise Explorer** откройте пакет:

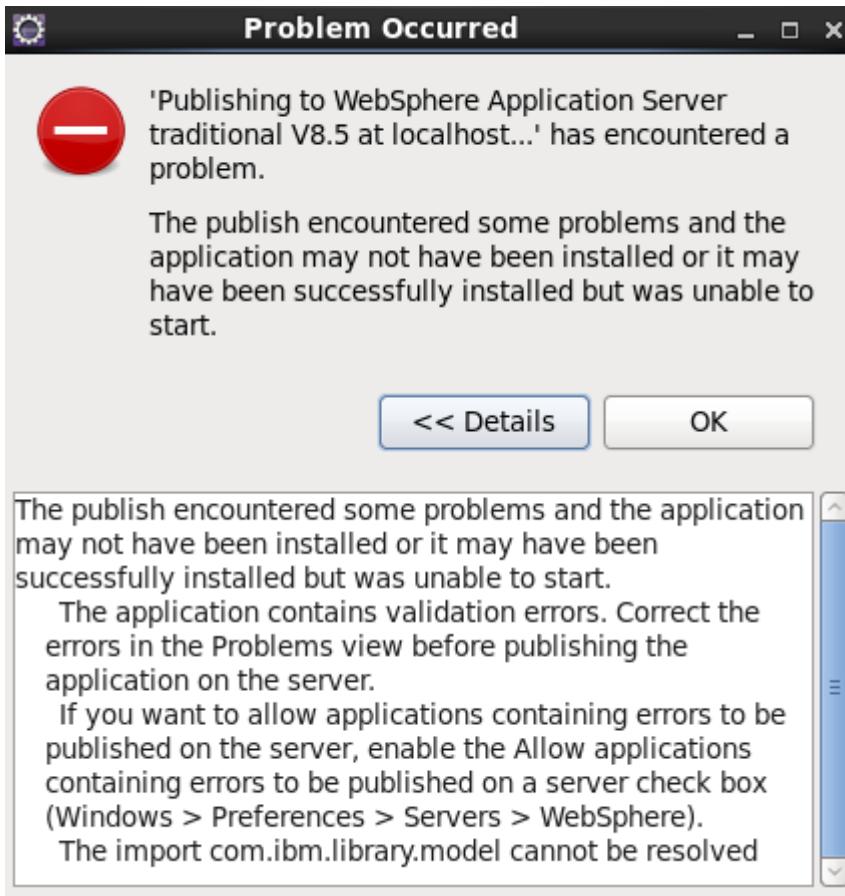
**LibraryWebProject -> Java Resources -> src ->
com.ibm.library.servlets**.

b. Методом Drag & Drop перенесите файл **LibraryIdGenerator.java** из пакета **com.ibm.library.servlets** проекта **LibraryWebProject** в пакет **com.ibm.library.model** проекта **LibraryJava**.

c. После открытия диалога **Move** согласитесь с политиками переноса по умолчанию и нажмите **OK**.

d. Эти действия приводят к тому, что ошибок в представлении **Problems** становится уже 2. Они будут исправлены позже. Также это приводит к тому, что Web проект, при попытке опубликовать его на сервере после обновления вызывает ошибку, связанную с неразрешенным классом. Эту ошибку также можно пока

проигнорировать.

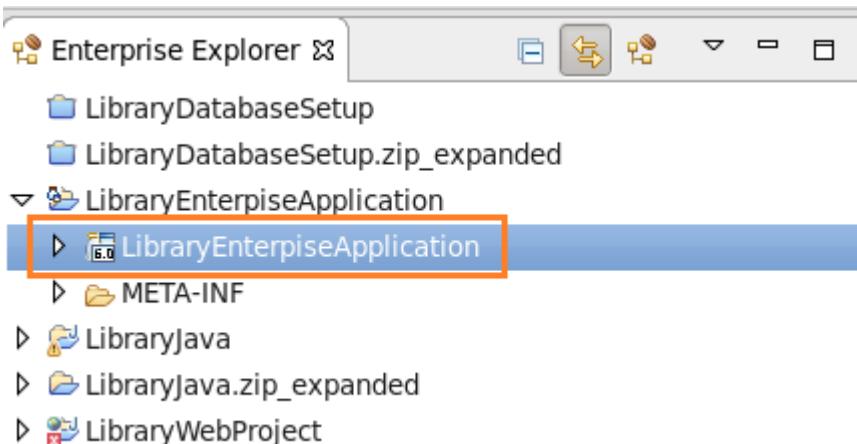


Раздел 2. Обеспечение взаимодействия Web приложения с классами проекта LibraryJava

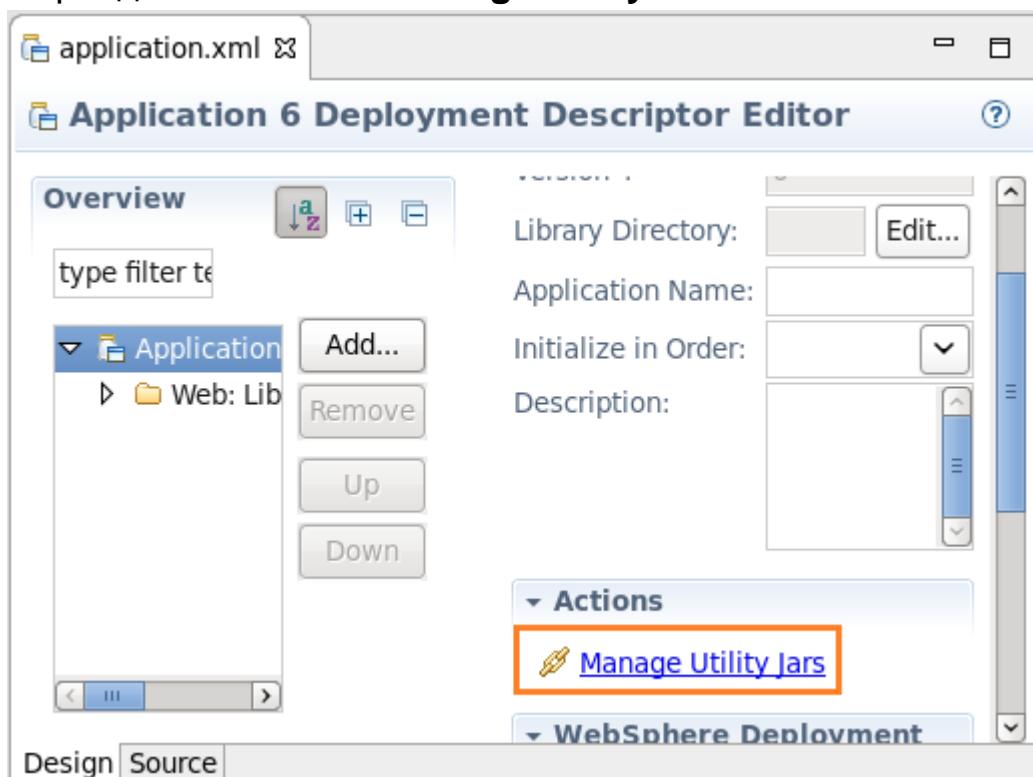
В данном разделе требуется настроить зависимость для Web модуля. При добавлении такой зависимости соответствующая строчка со ссылкой на jar появляется в файле MANIFEST.MF для модуля. При развертывания этого приложения соответствующий jar архив тоже разворачивается.

1. Откройте дескриптор развертывания приложения и дайте в нем ссылку на проект **LibraryJava**.
 - a. Переключитесь в перспективу **Web** (если вы находитесь не в ней).
 - b. В представлении **Enterprise Explorer** дважды кликните по дескриптору развертывания приложения

LibraryEnterpriseApplication.

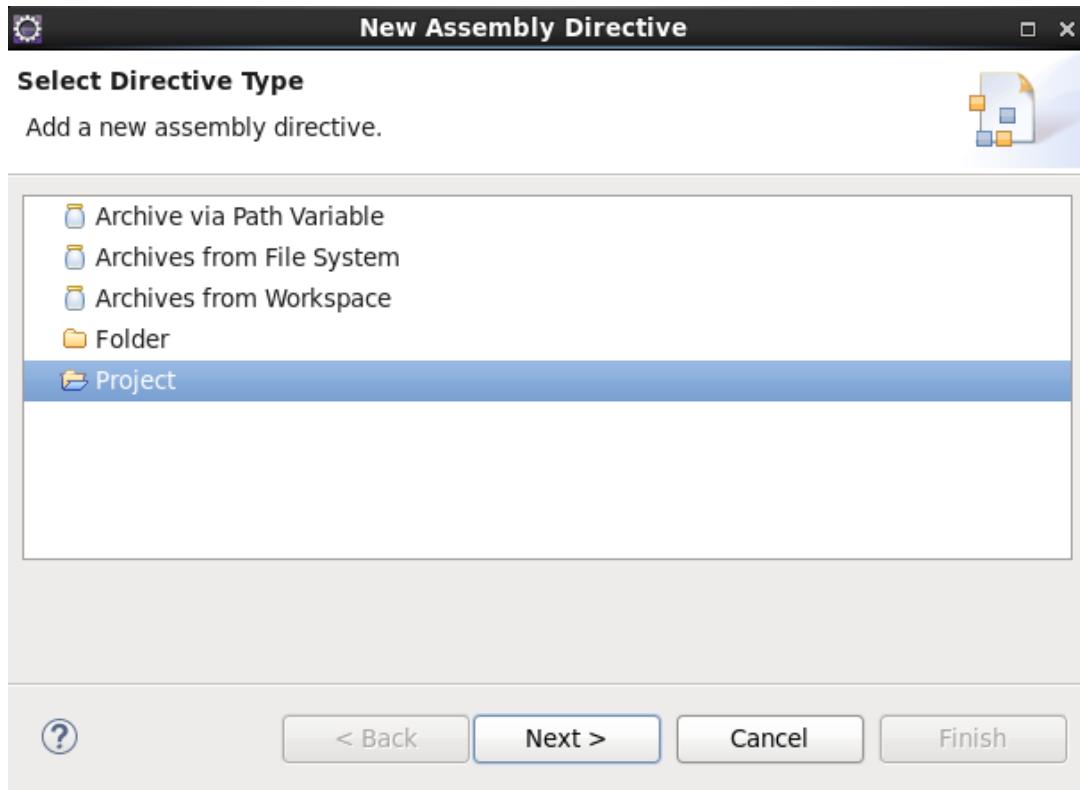


с. Перейдите по ссылке **Manage Utility Jars**.



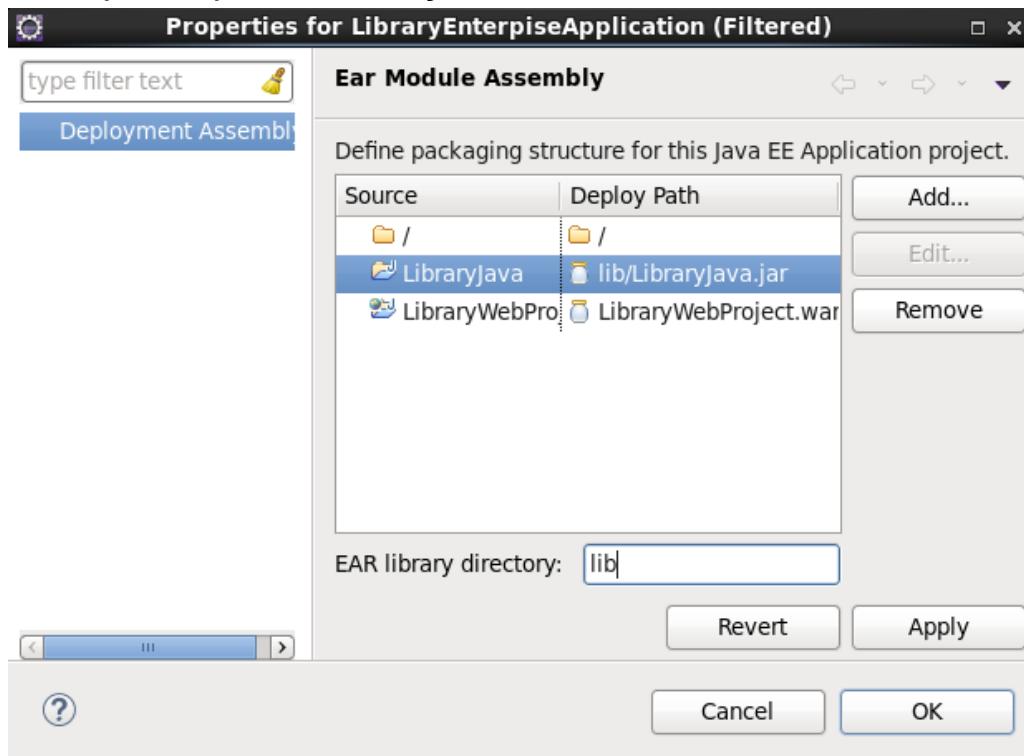
д. Открывается редактор для подключения дополнительных артефактов сборки. Нажмите **Add...**.

е. Выберите **Project**.



f. Нажмите **Next**.

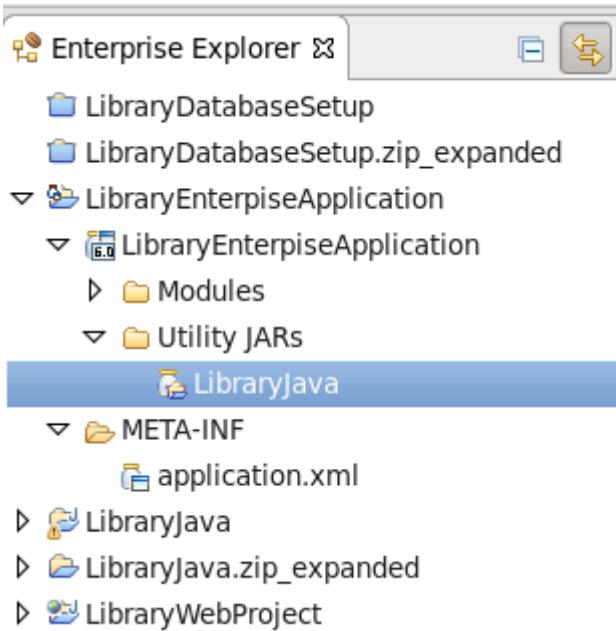
g. Выберите проект **LibraryJava**. Нажмите **Finish**.



h. Нажмите **OK**.

i. Эта цепочка действий приводит к тому, что теперь при создании поставки приложения (EAR файла), jar архив с классами из

проекта **LibraryJava** будет включен в него.



- j. Сохраните сделанные изменения и закройте дескриптор развертывания.
- k. После этого добавленная библиотека автоматически попадает в Build path для Web модуля, и соответственно из сервлета можно пользоваться функционалом проекта LibraryJava.

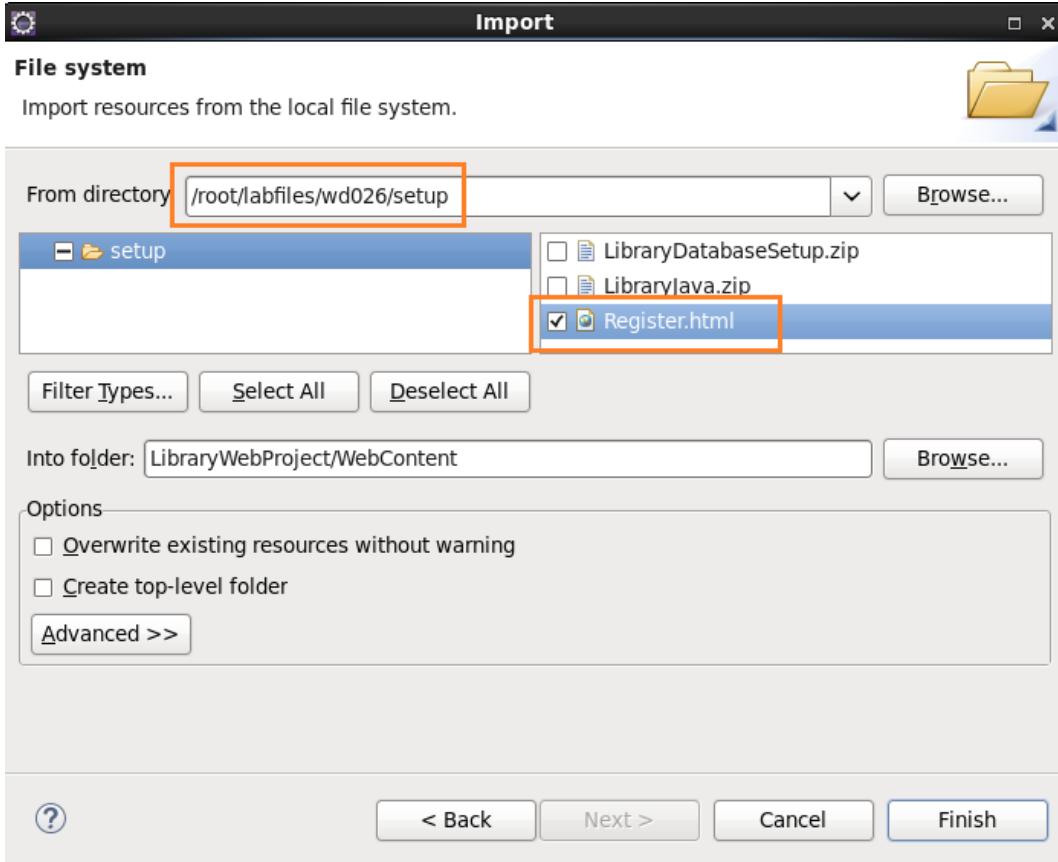
Раздел 3. Импорт страницы Register.html

Эта страница будет использоваться для тестирования нового функционала сервлета. С помощью нее можно будет отправить данные нового клиента на сервер через соответствующую форму. Эти данные будет обрабатывать сервлет. Форма состоит полей для ввода информации о клиенте, кнопки и ссылки, которая действуется при нажатии на кнопку и указывает на сервлет RegisterPatron.

1. Импортируйте страницу.

- a. В представлении **Enterprise Explorer** выберите **LibraryWebProject** -> **WebContent**. Нажмите по этому пункту правой кнопкой мыши и выберите в меню **Import**.
- b. Откроется диалог импорта ресурсов. Выберите **General -> FileSystem** и воспользуйтесь фильтром для поиска и выбора этого пункта. Нажмите **Next**.
- c. С помощью меню **Browse** укажите каталог, из которого вы хотите импортировать ресурс: **/root/labfiles/wd026/setup**.

d. Выберите файл **Register.html**. Не выбирайте никаких других ресурсов. Нажмите **Finish**.



Раздел 4. Изменение логики сервлета

Сервлет должен быть изменен для обеспечения возможности обрабатывать данные с формы. Таким образом, понадобится обновить метод **processRequest()**. Логика работы обновленного сервлета будет следующей:

- Для начала создается новый объект типа **Patron** с помощью метода **buildPatron()**;
- Проверяются значения полей **firstName**, **lastName**, **email** – они не должны быть пустыми или равны null. Это будет происходить с помощью метода **isValid()**.
- Если какие-то значения некорректны, то пользователю возвращается страницы с ошибкой. Делается это с помощью метода **displayMissingFields()**;
- Новый клиент добавляется в базу с помощью метода **add()**. В зависимости от результатов добавления клиента, пользователю отображается та или иная информация:

- Если все прошло успешно, открывается соответствующая страница. Используется метод **displayPatronAdded()**;
- При ошибке **PatronExists** открывается страница с ошибкой. Используется метод **displayDuplicatePatron()**;
- При ошибке **SystemUnavailableException** открывается страница с ошибкой. Используется метод **displaySystemError()**;
- При ошибке **InvalidPassword** открывается страница с ошибкой. Используется метод **displayInvalidPassword()**;

Все указанные методы, за исключением метода для работы с базой (**add()**) требуется разработать в процессе работы над этим упражнением.

1. Откройте файл **RegisterPatron.java** для редактирования.

а. В представлении **Enterprise Explorer** разверните **LibraryWebProject -> Java Recources -> src -> com.ibm.library.servlets**. Дважды нажмите по файлу **RegisterPatron.java**, чтобы открыть его в редакторе.

2. Реализуйте метод **buildPatron()**. Этот метод должен принимать на вход объект типа **HttpServletRequest**, а возвращать объект типа **Patron**. Для получения параметров будет использовать метод **getParameter()** объекта **HttpServletRequest**. Затем будет создаваться и возвращаться объект Patron, созданный на основе этих параметров. Названия параметров можно посмотреть в HTML, импортированном ранее.

а. Добавьте в код сервлета метод **buildPatron()**:

```
private Patron buildPatron(HttpServletRequest req) {  
    // pull off all of the fields and create a new Patron  
    Patron patron = null;  
  
    String firstName = req.getParameter("FIRST_NAME");  
  
    String lastName = req.getParameter("LAST_NAME");  
  
    String password = req.getParameter("PASSWORD");  
  
    String email = req.getParameter("EMAIL");
```

```
    patron = new Patron(firstName, lastName, password,
email);

    return patron;

}
```

- b. Нажмите по любой лампочке, сигнализирующей об ошибке в коде и выберите вариант **Import ‘Patron’ (com.ibm.library.model)** для решения этих проблем. Также можно сделать это с помощью сочетания клавиш **Ctrl + Shift + O**.
- c. Предупреждение о том, что метод не используется можно пока проигнорировать.
3. Реализуйте метод **isValid()**. Он должен принимать на вход объект **Patron**, проверять все его поля на предмет пустых и null строк, возвращать **boolean** значение. Если есть проблемы в данных, возвращается **false**, в противном случае – **true**.
- a. Добавьте в код сервлета следующий метод:

```
private boolean isValid(Patron patron) {

    String firstName = patron.getFirstName();

    String lastName = patron.getLastName();

    String email = patron.getEmail();

    boolean error = firstName == null ||
firstName.length() == 0

        || lastName == null || lastName.length() == 0

        || email == null || email.length() == 0;

    return !error;

}
```

4. Приватный метод **displayMissingFields()** принимает один параметр типа **HttpServletResponse**, возвращает **void**, выбрасывает ошибку

IOException. Логика метода в следующем: получается объект типа **PrintWriter** с помощью метода **getWriter()** объекта **HttpServletResponse**, далее выводится с помощью этого объекта сообщение об ошибке при валидации.

а. Добавьте следующий метод в класс сервлета:

```
private void displayMissingFields(HttpServletRequest
resp) throws IOException {

    PrintWriter out = resp.getWriter();

    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Missing Fields</TITLE>");
    out.println("<LINK rel=\"stylesheet\""
type="text/css\" href=\"/Library/theme/Master.css\">");

    out.println("</HEAD>");
    out.println("<P>");

    out.println("Registration failed: One or more
fields are incomplete." + " Please fill in all fields
and try again");

    out.println("</P>");
    out.println("</BODY>");
    out.println("</HTML>");

}
```

5. Приватный метод **displayDuplicatePatron()** принимает два параметра. Первый – **Patron**, второй – **HttpServletResponse**. Метод возвращает **void**, выбрасывает ошибку **IOException**. В методе используется объект **PrintWriter**, и с помощью него записывается сообщение для пользователя о том, что клиент с таким email уже существует.

а. Добавьте следующий метод в класс сервлета:

```
private void displayDuplicatePatron(Patron patron,
HttpServletResponse resp) throws IOException {

    PrintWriter out = resp.getWriter();

    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Duplicate Patron</TITLE>");
    out.println("<LINK rel=\"stylesheet\""
type="text/css\""
href="/Library/theme/Master.css\"");
    out.println("</HEAD>");
    out.println("<P>");
    out.println("Registration failed: Patron " +
patron.getEmail() +
" already exists and could not be added");
    out.println("</P>");
    out.println("</BODY>");
    out.println("</HTML>");

}
```

6. Приватный метод **displaySystemError()** принимает два параметра.

Первый – **HttpServletResponse**, второй –

SystemUnavailableException. Метод возвращает **void**, выбрасывает ошибку **IOException**. В методе используется объект **PrintWriter**, и с помощью него записывается сообщение для пользователя о том, что возникла системная ошибка, которая не позволила провести регистрацию клиента.

а. Добавьте следующий метод в класс сервлета:

```
private void displaySystemError(
```

```
HttpServletResponse resp,  
SystemUnavailableException e)  
throws IOException {  
  
    PrintWriter out = resp.getWriter();  
  
    out.println("<HTML>");  
  
    out.println("<HEAD>");  
  
    out.println("<TITLE>System Error</TITLE>");  
  
    out.println(  
        "<LINK rel=\"stylesheet\" type=\"text/css\""  
        + "  
        href=\"/Library/theme/Master.css\"");  
  
    out.println("</HEAD>");  
  
    out.println("<P>");  
  
    out.println("Registration failed: There is a  
system error. " +  
        "Please try the registration later");  
  
    out.println("</P>");  
  
    out.println("</BODY>");  
  
    out.println("</HTML>");  
}
```

7. Приватный метод **displayInvalidPassword()** принимает два параметра. Первый – **String** (с сообщением об ошибке **InvalidPassword**), второй – **HttpServletResponse**. Метод возвращает **void**, выбрасывает ошибку **IOException**. В методе используется объект **PrintWriter**, и с помощью него записывается сообщение для пользователя о том, что регистрация невозможна из-за неправильно введенного пароля.

а. Добавьте следующий метод в класс сервера:

```
private void displayInvalidPassword(
    String message,
    HttpServletResponse resp)
    throws IOException {

    PrintWriter out = resp.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Invalid password</TITLE>");
    out.println(
        "<LINK rel=\"stylesheet\" type=\"text/css\""
        +
        " href=\"/Library/theme/Master.css\"");
    out.println("</HEAD>");
    out.println("<P>");
    out.println("Registration failed: " + "Patron");
    out.println("could not be added. " + message);
    out.println("</P>");
    out.println("</BODY>");
    out.println("</HTML>");
}
```

8. Приватный метод **displayPatronAdded()** принимает два параметра. Первый – **Patron**, второй – **HttpServletResponse**. Метод возвращает **void**, выбрасывает ошибку **IOException**. В методе используется объект **PrintWriter**, и с помощью него записывается сообщение для

пользователя о том, что клиент с указанным email был зарегистрирован в базе данных под некоторым идентификатором.

а. Добавьте следующий метод в класс сервлета:

```
private void displayPatronAdded(Patron patron,
HttpServletResponse resp)

throws IOException {

PrintWriter out = resp.getWriter();

out.println("<HTML>");
out.println("<HEAD>");
out.println("<TITLE>Patron Added</TITLE>");
out.println(
"<LINK rel=\"stylesheet\" type=\"text/css\""
+ "
href=\"/Library/theme/Master.css\"");

out.println("</HEAD>");
out.println("<P>");
out.println(
"Patron "
+ patron.getEmail()
+ " with id "
+ patron.getId()
+ " has been added.");

out.println("</P>");
out.println("</BODY>");
out.println("</HTML>");

}
```

9. Замените метод **processRequest()**. Логика работы этого нового метода описана в начале данного этого раздела.

a. Удалите текущую версию метода **processRequest()**.

b. Добавьте новую версию метода в класс сервлета:

```
private void processRequest(  
    HttpServletRequest req,  
    HttpServletResponse resp)  
throws IOException {  
  
    Patron patron = null;  
  
    // Build the Patron from the HttpServletRequest  
    // fields  
    patron = buildPatron(req);  
  
    // Verify that the patron is valid. Build a  
    // message based  
    // on the results of the processing.  
    if (!isValid(patron)) {  
        // No, one or more fields are missing  
        displayMissingFields(resp);  
    } else {  
        // Try to add the new patron  
        try {  
            patron.add();  
            displayPatronAdded(patron, resp);  
        } catch (PatronExists e) {  
            displayDuplicatePatron(patron, resp);  
        }  
    }  
}
```

```
        } catch (SystemUnavailableException e) {  
            displaySystemError(resp, e);  
        } catch (InvalidPassword e) {  
            String message = e.getMessage();  
            displayInvalidPassword(message, resp);  
        }  
    }  
}
```

с. Нажмите по лампочке рядом со строчкой:

```
} catch (PatronExists e) {
```

выберите вариант **Import ‘PatronExists’**
(com.ibm.library.model.exceptions) для разрешения этой
проблемы.

d. Нажмите по лампочке рядом со строчкой:

```
} catch (InvalidPassword e) {
```

выберите вариант **Import ‘InvalidPassword’**
(com.ibm.library.model.exceptions) для разрешения этой
проблемы.

e. Для разрешения остальных проблем с импортами нажмите **Ctrl + Shift + O** или в меню **Source -> Organize Imports**.

f. Сохраните сделанные изменения

Раздел 5. Создание источника данных (data source) и тестирование сервлета

Чтобы приложение могло взаимодействовать с базой, предварительно на сервере нужно создать ресурс, который называется data source. Это ресурс указывает на конкретную базу и определяет параметры для соединения с ней. Специфика WebSphere Application Server позволяет нам для решения этой задачи не прибегать к помощи административных интерфейсов, а воспользоваться технологией расширенного EAR файла

(Enhanced EAR). Она предполагает, что определение необходимого для приложения ресурса будет размещено в специальном конфигурационном файле в самом EAR архиве приложения. При его установке сервер сам автоматически создаст такой ресурс с областью видимости для нашего разворачиваемого приложения. Но такой подход требует в обязательном порядке использование среды разработки Rational Application Developer, которая позволяет создавать такие артефакты. В Eclipse сделать это весьма затруднительно, поэтому Data Source будет создан в административной консоли сервера.

1. Зайдите в административную консоль сервера приложений.

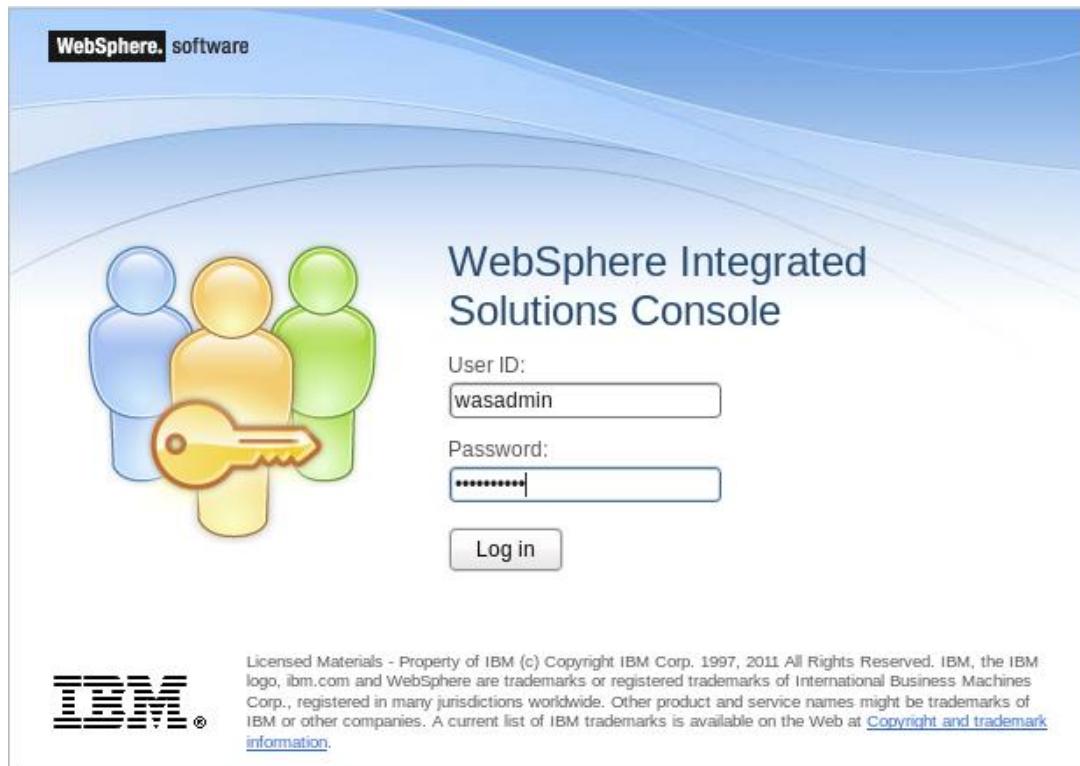
a. В **Mozilla Firefox** перейдите по адресу

http://localhost:9060/ibm/console

b. Введите данные учетной записи администратора сервера:

User ID: wasadmin

Password: web1sphere



2. Создайте Data Source, указывающий на базу в каталоге

/root/labfiles/wd026/database/library и имеющий JNDI имя
jdbc/libraryDev.

а. Перейдите по ссылке в меню: **Resources -> JDBC -> Data Sources.**



б. Выберите **Scope** (область видимости ресурса)

Node=wd026Node01, Server=server1

The screenshot shows the 'Data sources' configuration page. At the top, there is a header bar with the title 'Data sources'. Below it, there is a section titled 'Data sources' with a descriptive text about editing datasource settings. Underneath, there is a 'Scope' dropdown menu with the value 'Node=wd026Node01, Server=server1' selected, which is highlighted with a red box. Further down, there is a 'Preferences' section with buttons for 'New...', 'Delete', 'Test connection', and 'Manage state...'. Below that is a toolbar with icons for creating, deleting, and managing resources. A table follows, with columns for 'Select', 'Name', 'JNDI name', 'Scope', 'Provider', 'Description', and 'Category'. The table contains one row for a 'Default Datasource' with the following details:

Select	Name	JNDI name	Scope	Provider	Description	Category
<input type="checkbox"/>	Default Datasource	DefaultDatasource	Node=wd026Node01,Server=server1	Derby JDBC Provider	Datasource for the WebSphere Default Application	

с. Нажмите кнопку **New...**

д. Укажите следующие параметры источника данных:

Data source name: library

JNDI name: **jdbc/libraryDev**

Create a data source

→ Step 1: Enter basic data source information Step 2: Select JDBC provider Step 3: Enter database specific properties for the data source Step 4: Setup security aliases Step 5: Summary	Enter basic data source information <p>Set the basic configuration values of a datasource for association with your JDBC provider. A datasource supplies the physical connections between the application server and the database.</p> <p>Requirement: Use the Datasources (WebSphere(R) Application Server V4) console pages if your applications are based on the Enterprise JavaBeans(TM) (EJB) 1.0 specification or the Java(TM) Servlet 2.2 specification.</p> <p>Scope</p> <p>cells:nullNode01Cell:nodes:wd026Node01:servers:server1</p> <p>* Data source name library</p> <p>* JNDI name jdbc/libraryDev</p>
---	---

Next **Cancel**

e. Нажмите **Next**.

- f. Выберите существующий драйвер для работы с СУБД: **Derby JDBC Provider**. Этот ресурс драйвера создается на сервере WebSphere по умолчанию. Нажмите **Next**.
- g. Укажите путь до каталога с базой данных:
/root/labfiles/wd026/database/library.

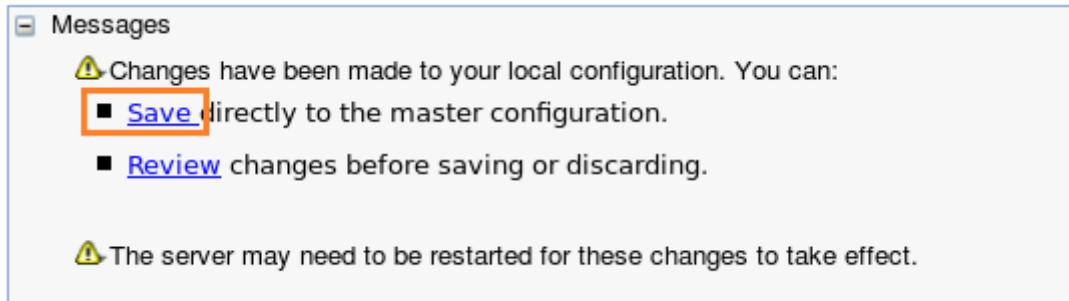
Create a data source

Step 1: Enter basic data source information Step 2: Select JDBC provider → Step 3: Enter database specific properties for the data source Step 4: Setup security aliases Step 5: Summary	Enter database specific properties for the data source <p>Set these database-specific properties, which are required by the database vendor JDBC driver to support the connections that are managed through the datasource.</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>* Database name</td> <td>/root/labfiles/wd026/database/library</td> </tr> </tbody> </table> <p><input checked="" type="checkbox"/> Use this data source in container managed persistence (CMP)</p>	Name	Value	* Database name	/root/labfiles/wd026/database/library
Name	Value				
* Database name	/root/labfiles/wd026/database/library				

Previous **Next** **Cancel**

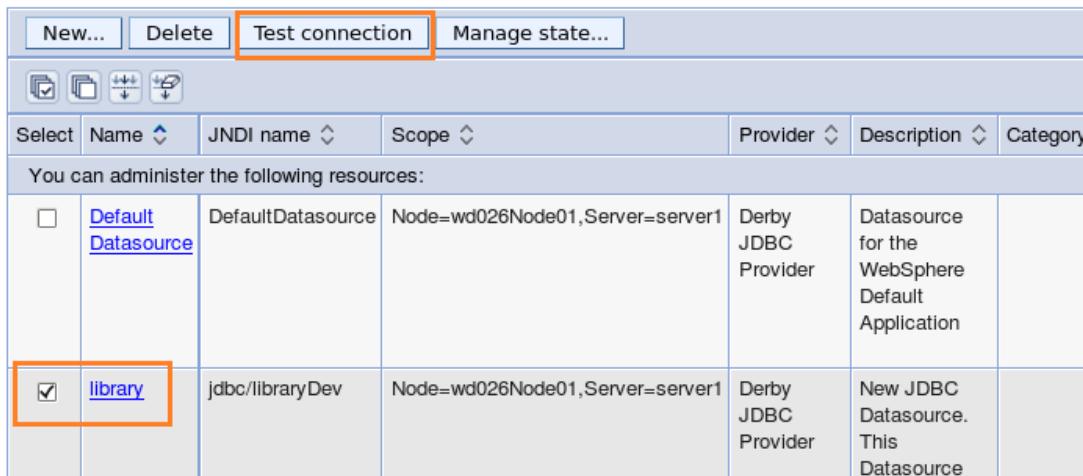
h. В настройках безопасности, ничего не меняя, нажмите **Next**.

- i. В итоговом интерфейсе проверьте все введенные/выбранные значения и нажмите **Finish**.
- j. Сохраните сделанные изменения, кликнув по ссылке **Save** в сообщение в верхней части консоли.



3. Проверьте созданный источник данных.

- a. Выберите в списке созданный источник данных **library** и нажмите кнопку **Test connection**.



- b. Если Data Source был создан корректно, то появится сообщение об успешности теста.



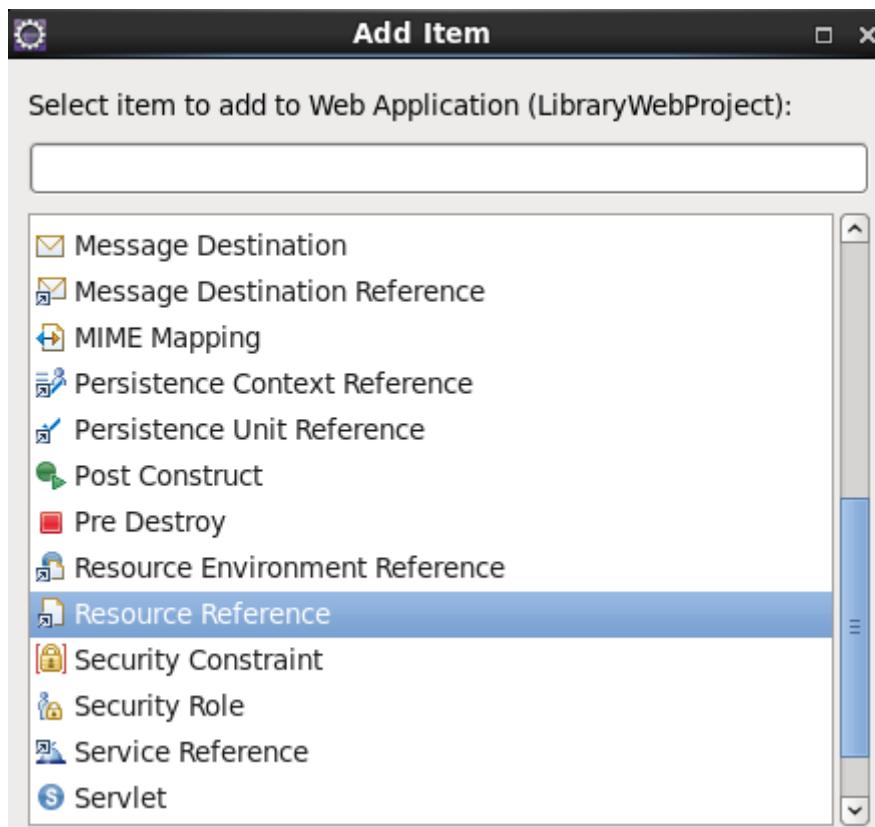
4. Остановите сервер приложений. Дальнейшие манипуляции с проектом могут привести к тому, что во время автоматического обновления кода приложения возникнут ошибки, которые можно будет исправить только с перезагрузкой сервера.

- a. Вернитесь в **Eclipse**.
- b. В представлении **Servers** выберите **WebSphere Application Server**, нажмите по нему правой кнопкой мыши и нажмите **Stop**.
- c. Подождите, пока статус сервера не изменится на **Stopped**.

5. Создайте ссылку на ресурс (resource reference) для источника данных в дескрипторе развертывания Web модуля. Ссылка на

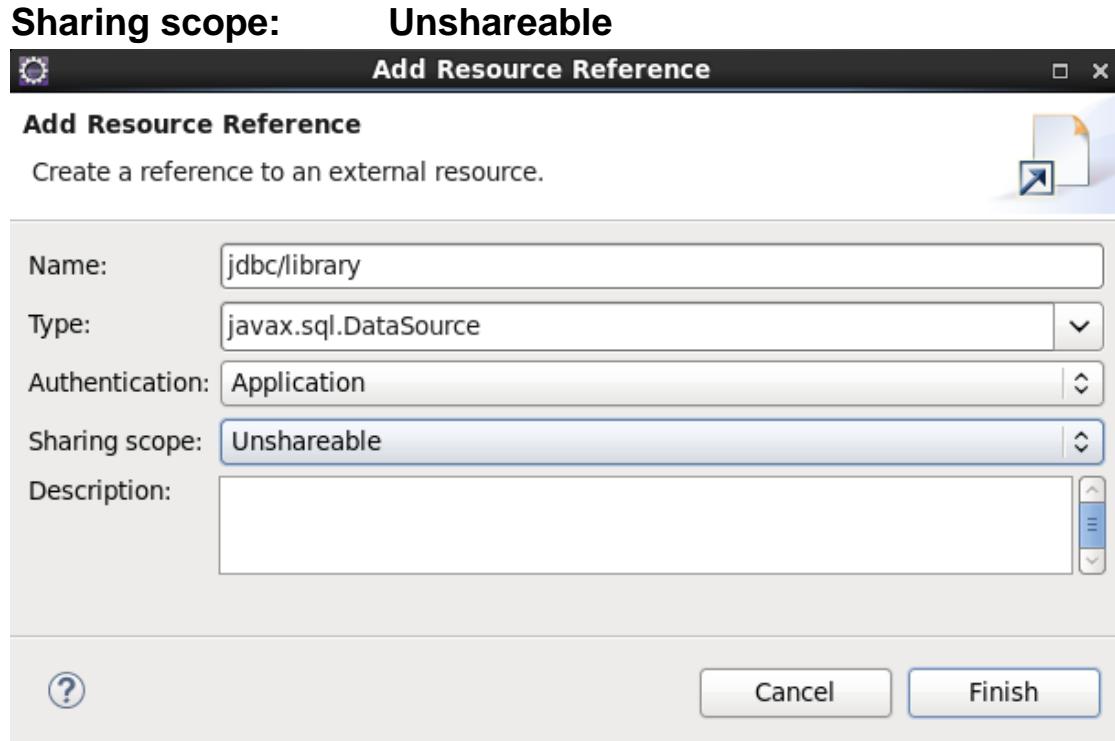
ресурс добавляет дополнительный уровень абстракции между именем источника данных, который использует разработчик, и именем источника данных, который определен на сервере. Таким образом, разработчику не обязательно знать реальное JNDI имя источника данных на сервере.

- a. В представлении **Enterprise Explorer** дважды нажмите по дескриптору развертывания Web модуля, чтобы приступить к его редактированию: **LibraryWebProject -> LibraryWebProject**.
- b. Нажмите кнопку **Add** для добавления нового элемента в дескриптор развертывания.
- c. Выберите элемент **Resource Reference** для добавления и нажмите **OK**.

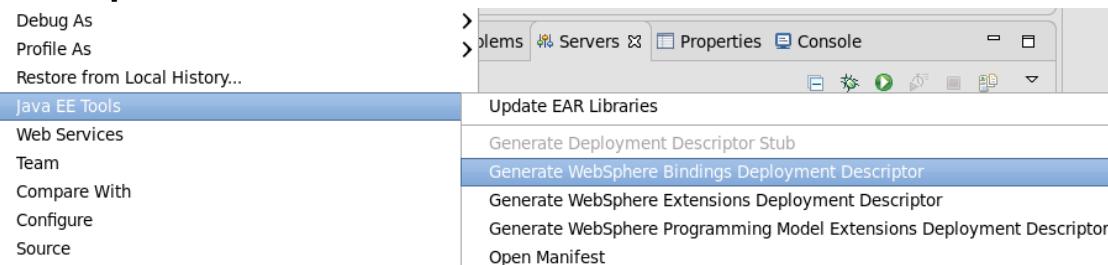


- d. Укажите следующие параметры ресурсной ссылки:

Name: **jdbc/library**
Type: **javax.sql.DataSource**
Authentication: **Application**

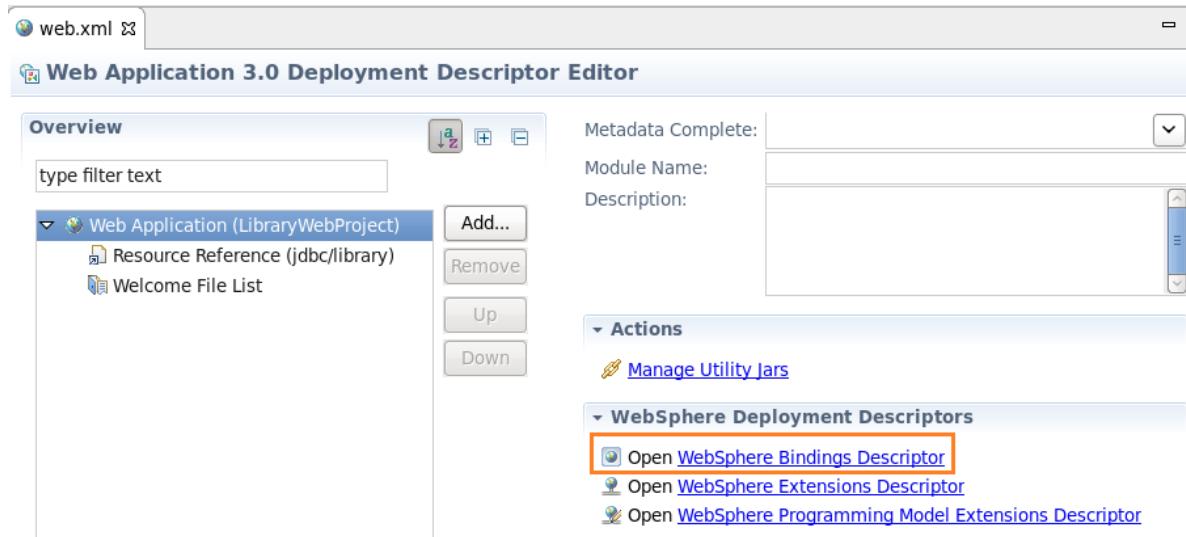


- e. Нажмите **Finish**.
- f. **Сохраните** сделанные изменения и закройте дескриптор развертывания.
6. Свяжите ресурсную ссылку **jdbc/library** с источником данных **jdbc/libraryDev**, созданном на сервере. Сделать это можно, редактируя специальный конфигурационный файл, входящий в расширенный EAR для сервера приложений WebSphere.
- a. В представлении **Enterprise Explorer** нажмите правой кнопкой мыши по проекту **LibraryEnterpriseApplication**. Выберите пункт **Java EE Tools -> Generate WebSphere Bindings Deployment Descriptor** в меню.

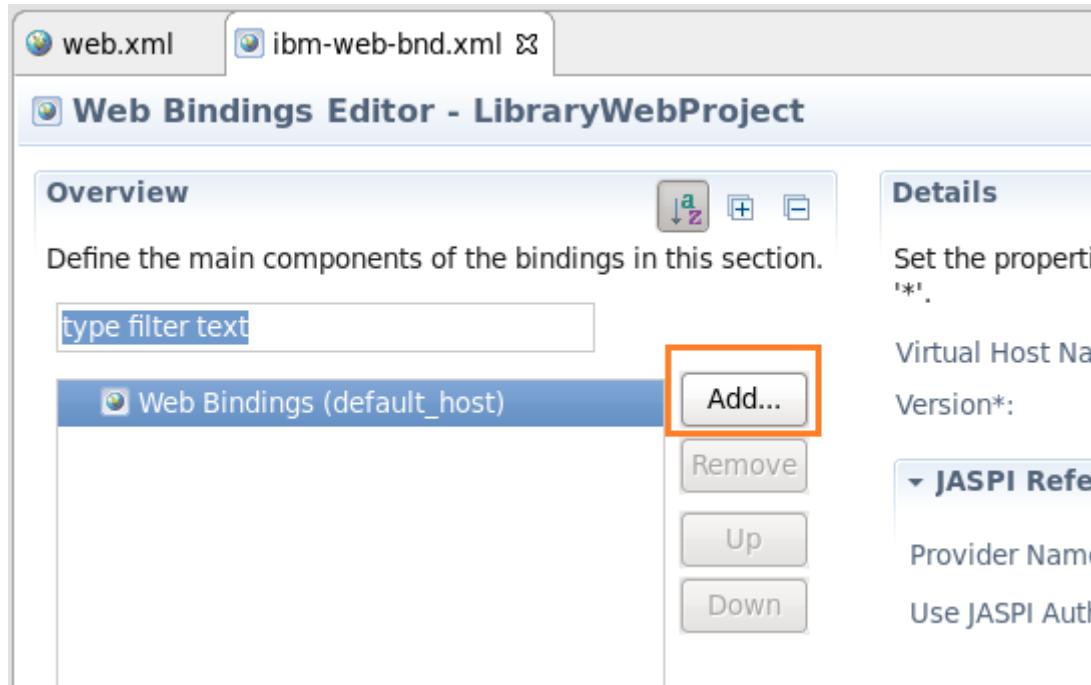


- b. **Сохраните** и закройте созданный документ.
- c. Откройте дескриптор развертывания Web модуля: **LibraryWebProject -> LibraryWebProject**.

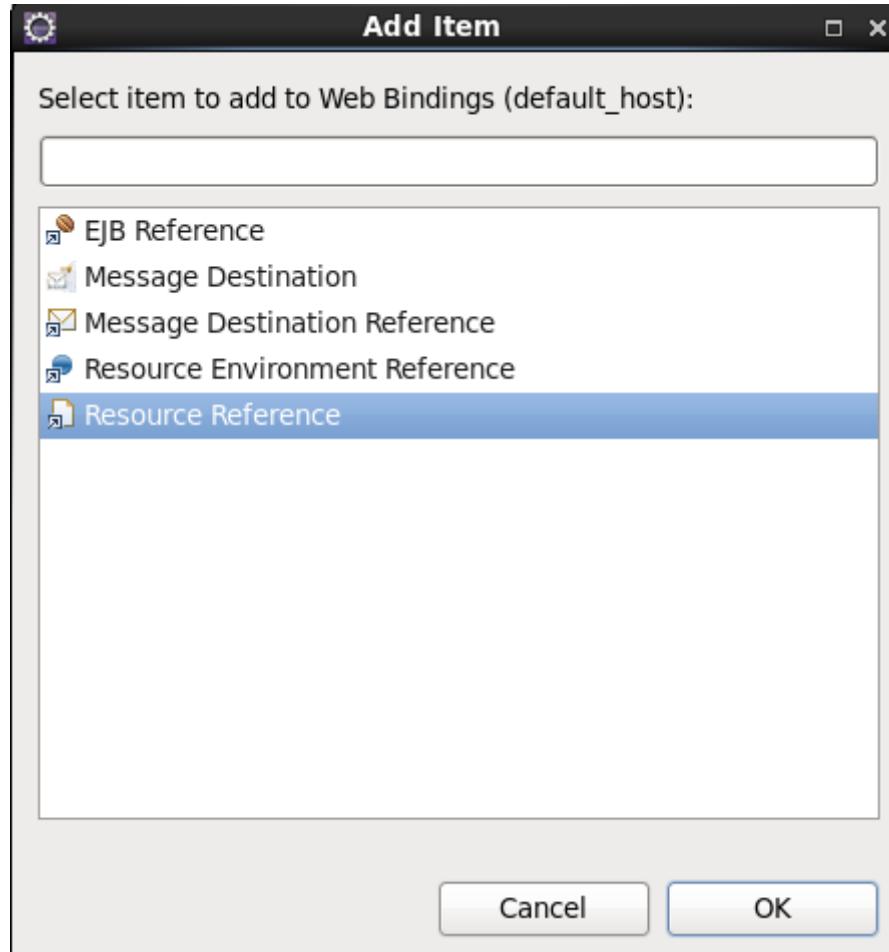
d. В редакторе работы с дескриптором перейдите по ссылке **WebSphere Bindings Descriptor**.



e. Нажмите кнопку **Add...** в редакторе для добавления нового элемента.



f. Выберите **Resource Reference**.



g. В секции **Details** укажите следующие параметры этой привязки:

Name: jdbc/library

Binding name: jdbc/libraryDev

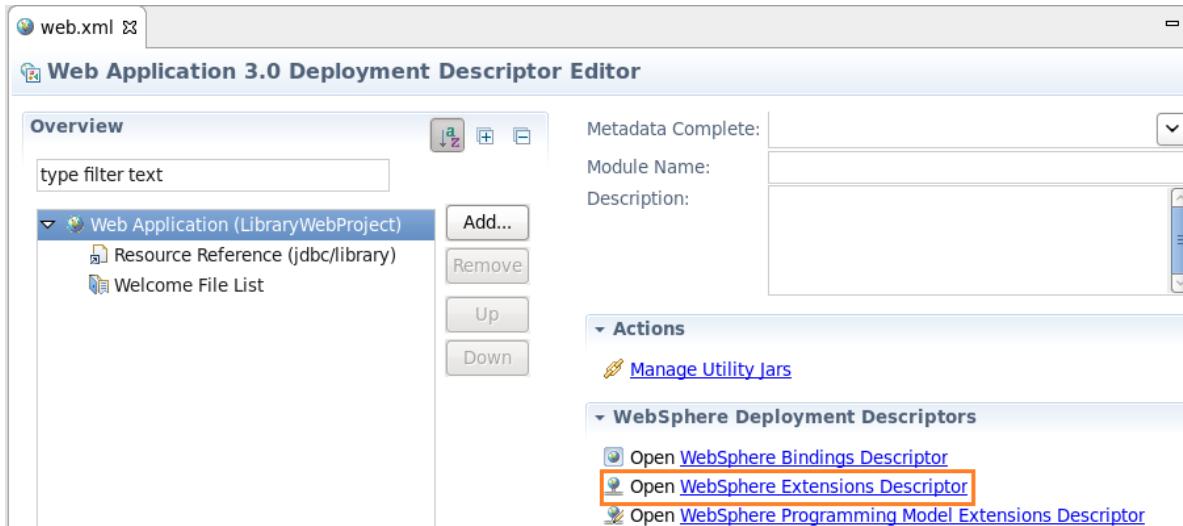
Details

Set the properties for the selected item. Required fields are denoted by '*'.

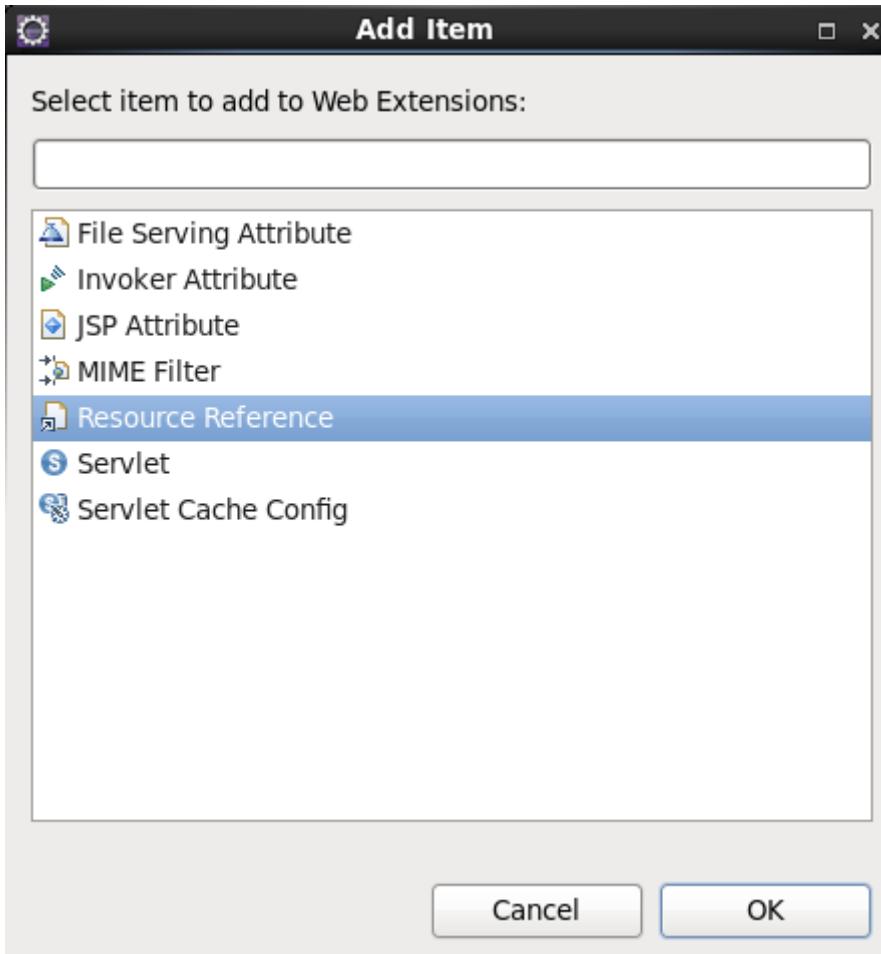
Name*:	jdbc/library
Binding Name*:	jdbc/libraryDev

h. Сохраните сделанные изменения и закройте редактор.

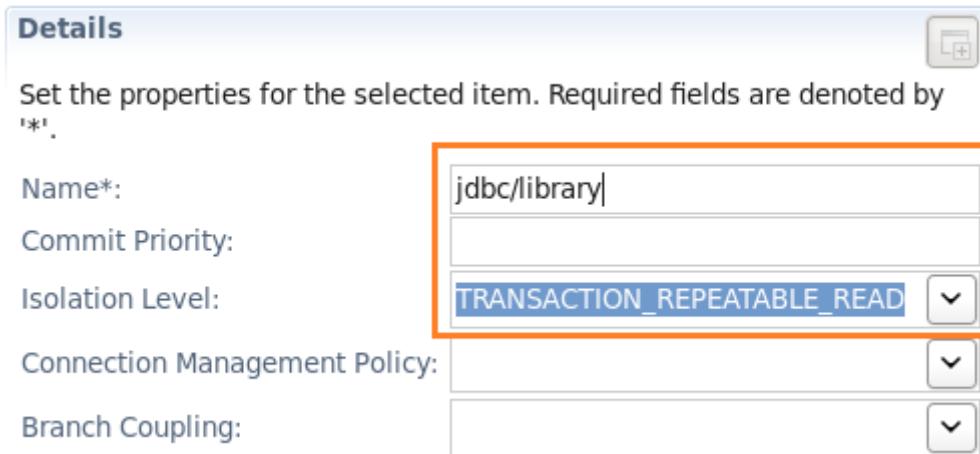
- i. В интерфейсе редактора дескриптора откройте ссылку **WebSphere Extensions Descriptor**.



- j. Нажмите **Add...** для добавления нового элемента.
k. Выберите тип расширения **Resource Reference**.



- l. В секции **Details** укажите следующие параметры:
Name: *jdbc/library*
Isolation Level: *TRANSACTION_REPEATABLE_READ*



Это определяет уровень блокировок при работе с этой базой данных. **Сохраните** изменения и закройте редактор.

7. Запустите сервер и опубликуйте приложение.

- a. В представлении **Servers** выберите **WebSphere Application Server**, нажмите по нему правой клавишей мыши и выберите **Start**.
- b. Дождитесь окончания запуска сервера – он должен перейти в состояние **Started**.
- c. Приложение должно быть опубликовано автоматически с текущими настройками рабочего пространства. В этом можно убедиться по статусу приложения в представлении **Servers**. Статус должен быть **Started, Synchronized**. Если статус какой-то иной, например Republish, то нужно выбрать **WebSphere Application Server**, нажать по нему правой клавишей мыши и выбрать **Publish**.

8. Протестируйте сервлет, смоделировав различные ситуации: успешная регистрация, ошибка валидации, попытка создания дублирующей записи, попытка использовать пароль меньше 5 символов.

- a. Перейдите в браузере по адресу:
http://localhost:9080/Library/Register.html

- b. После открытия страницы регистрации заведите нового клиента

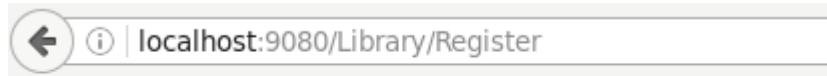


IBM Library System

Register A New Patron

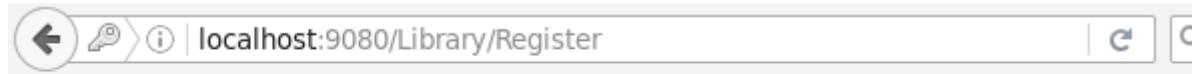
First name:	ivan
Last name:	ivanov
Email address:	ivanov@mail.ru
Password	*****
Register	

- c. Операция должна завершиться успешно.



Patron ivanov@mail.ru with id 774587 has been added.

- d. Попробуйте добавить клиента с тем же email.



Registration failed: Patron ivanov@mail.ru already exists and could not be added

- e. Попробуйте не заполнить какие-то поля в форме.



Registration failed: One or more fields are incomplete. Please fill in all fields and try again

- f. Попробуйте использовать пароль меньше 5 символов.



Registration failed: Patron could not be added. Password must be at least 5 characters long

- g. После завершения тестирования закройте браузер.

Конец упражнения

Упражнение 4. Простая JSP страница

О чём это упражнение:

В этой лабораторной создаются и тестируются две простые JSP страницы, которые будут использоваться в последующих упражнениях. Первая страница принимает параметр через URL и отображает соответствующее сообщение. Вторая страница принимает email и идентификатор в качестве параметров и отображает сообщение об успешной регистрации клиента.

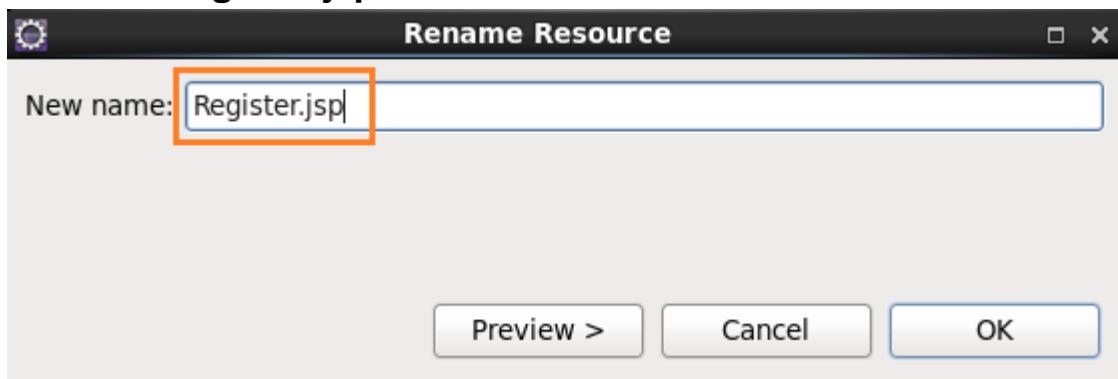
Что вы должны будете сделать:

- Создать JSP страницу
- Воспользоваться JSP Scriptlet
- Воспользоваться JSP Expressions (выражениями)
- Передать параметры для JSP Через URL

Раздел 1. Создание JSP Register

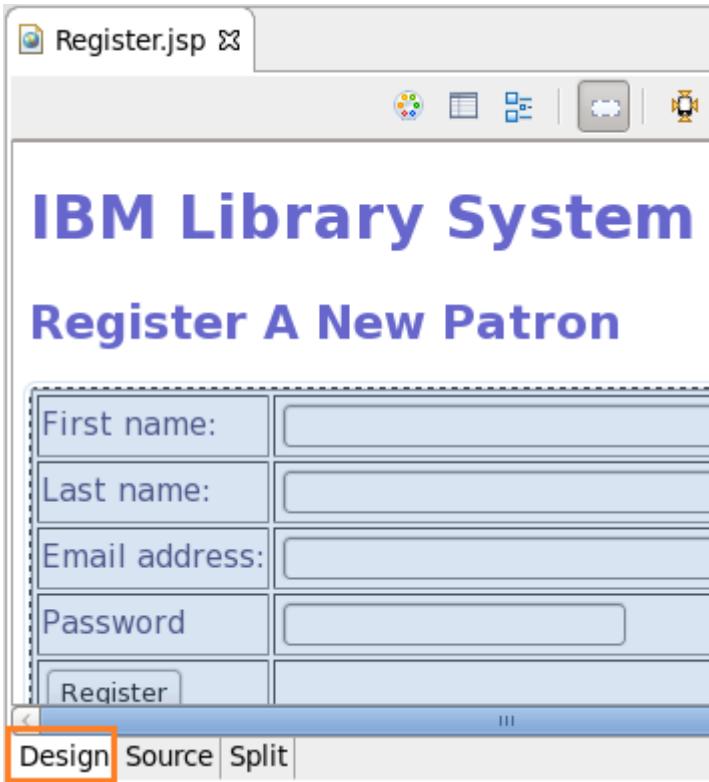
В этом разделе создается JSP страница. Создается она на основе HTML страницы **Register.html**, которая используется как шаблон для этой конвертации. После конвертации JSP не будет содержать какой-то логики, а будет, по сути, той же самой, только переименованной (из **Register.html** в **Register.jsp**) страницей. Для получения значения параметра **message** следует использовать метод **getParameter()** объекта **request**. Если параметр не установлен, то в **message** сохраняется пустая строка. Далее сообщение выводится в виде заголовка на странице, для этого можно использовать метод **println()**.

1. Переименуйте файл **Register.html** в **Register.jsp**.
 - a. Переключитесь при необходимости в перспективу **Web**.
 - b. В представлении Enterprise Explorer разверните каталог: **LibraryWebProject -> WebContent**.
 - c. Выберите файл **Register.html**. В главном меню выберите **File -> Rename** (или просто нажмите **F2**).
 - d. Укажите **Register.jsp** в качестве нового имени. Нажмите **OK**.

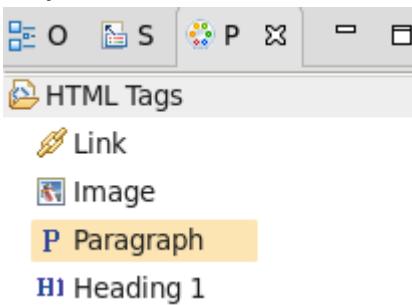


2. Добавьте в JSP логику (скриплет) для считывания параметра и его отображения на странице.
 - a. Откройте страницу **Register.jsp** в редакторе.

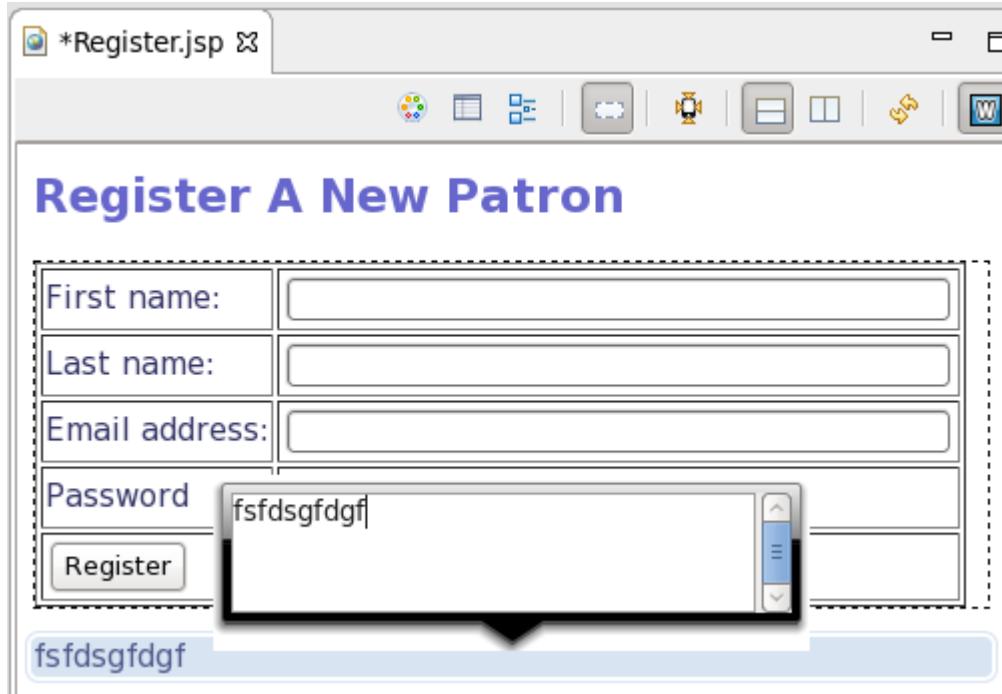
b. Перейдите на вкладку **Design**.



c. Откройте представление **Palette**. Выберите раздел **HTML Tags** и выберите из него элемент **Paragraph** и методом Drag & Drop перенести его в нижнюю часть формы.



- d. Дважды кликнув по этому элементу интерфейса, вы можете ввести текст для этого абзаца.



- e. Перейдя в представление Properties, Вы можете изменить некоторые атрибуты этого элемента, в частности используемые стили.
- f. Удалите созданный элемент, выделив его и нажав кнопку **Delete**. Если возникнет сообщение об ошибке, его можно проигнорировать, нажав **OK**. Элемент в любом случае будет удален.
- g. Перейдите на вкладку **Source** для работы с кодом JSP страницы.
- h. Добавьте в конец тела страницы до закрывающего тега </BODY> следующий фрагмент:

```
<p>
<%String message =
request.getParameter("message") ;

if (message == null) {

    message = "";

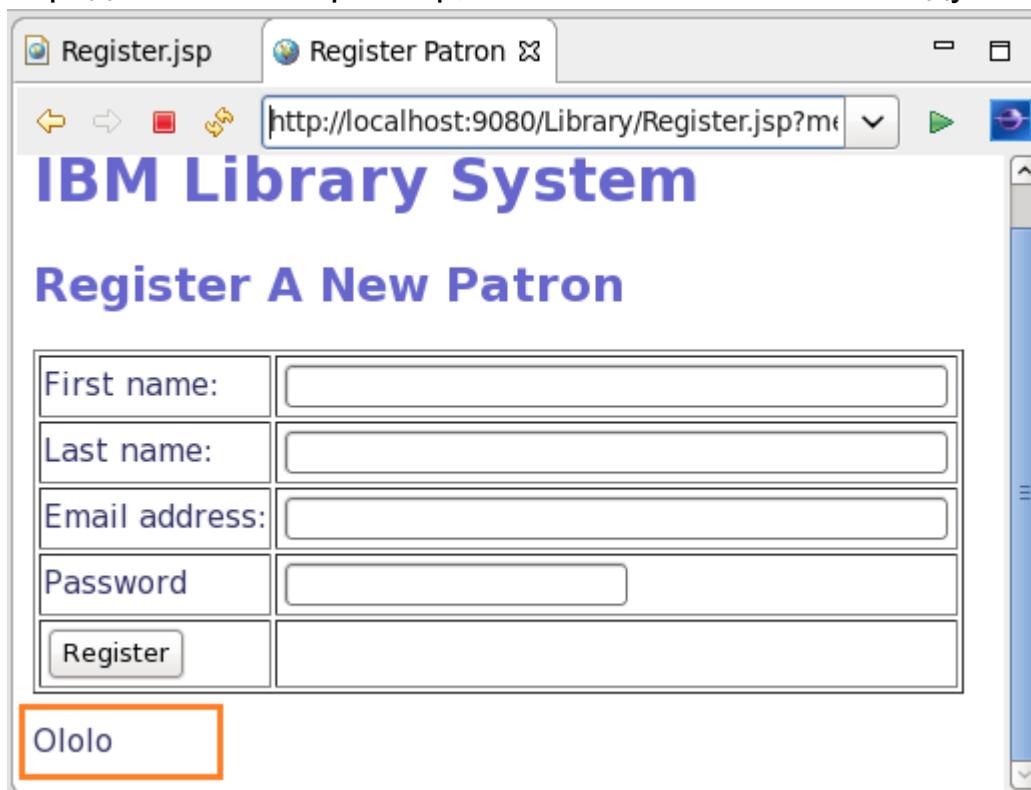
}

try {

    out.println(message) ;
```

```
        } catch (java.io.IOException e) {  
  
            System.out.println("Exception occurred in  
Register.jsp: " +  
e);  
  
} %></p>
```

- i. Сохраните сделанные изменения и закройте редактор.
3. Проверьте работы обновленной JSP.
 - a. В представлении **Enterprise Explorer** выберите страницу **Register.jsp**, нажмите по ней правой кнопкой мыши и выберите **Run As -> Run on Server**.
 - b. После открытия встроенного браузера, обратите внимание на содержание формы. Сообщение отсутствует, так как для JSP не было передано никаких параметров.
 - c. Измените URL для перехода на эту страницу, чтобы включить в него дополнительный параметр message, например, используйте такой: **http://localhost:9080/Library/Register.jsp?message=Ololo**
 - d. Теперь в нижней части страницы отображается сообщение, переданное как параметр, что и было изначально задумано.



Раздел 2. Создание JSP *RegistrationSuccess*

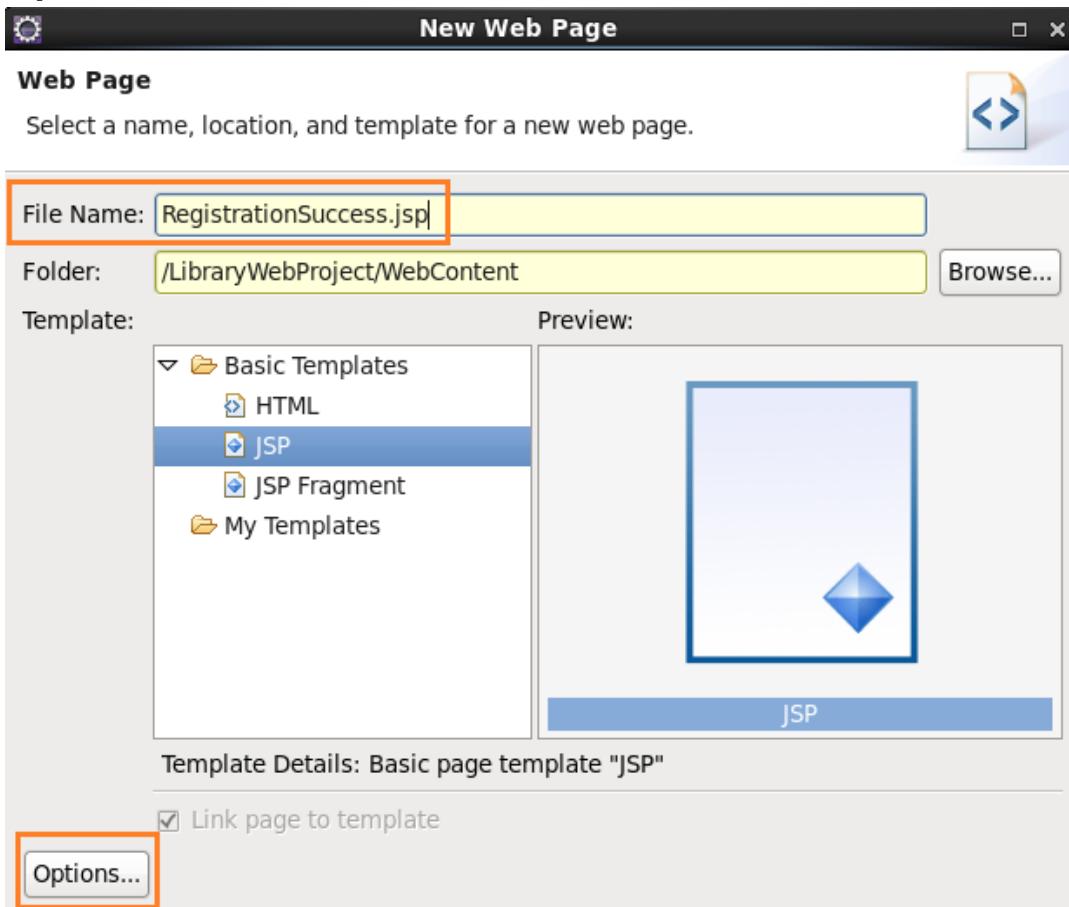
В этом разделе создается и тестируется JSP страница **RegistrationSuccess**. Эта страница будет считывать передаваемые параметры **email** и **id**, с помощью метода **getParameter()**, если эти параметры не переданы, то для них устанавливается значение “**unknown**”. После этого выводится сообщение о том, что **клиент с email** был зарегистрирован под идентификатором **id**.

1. Создайте новую JSP страницу.

a. В представлении **Enterprise Explorer** выберите **LibraryWebProject** → **WebContent**, нажмите по нему правой кнопкой мыши и

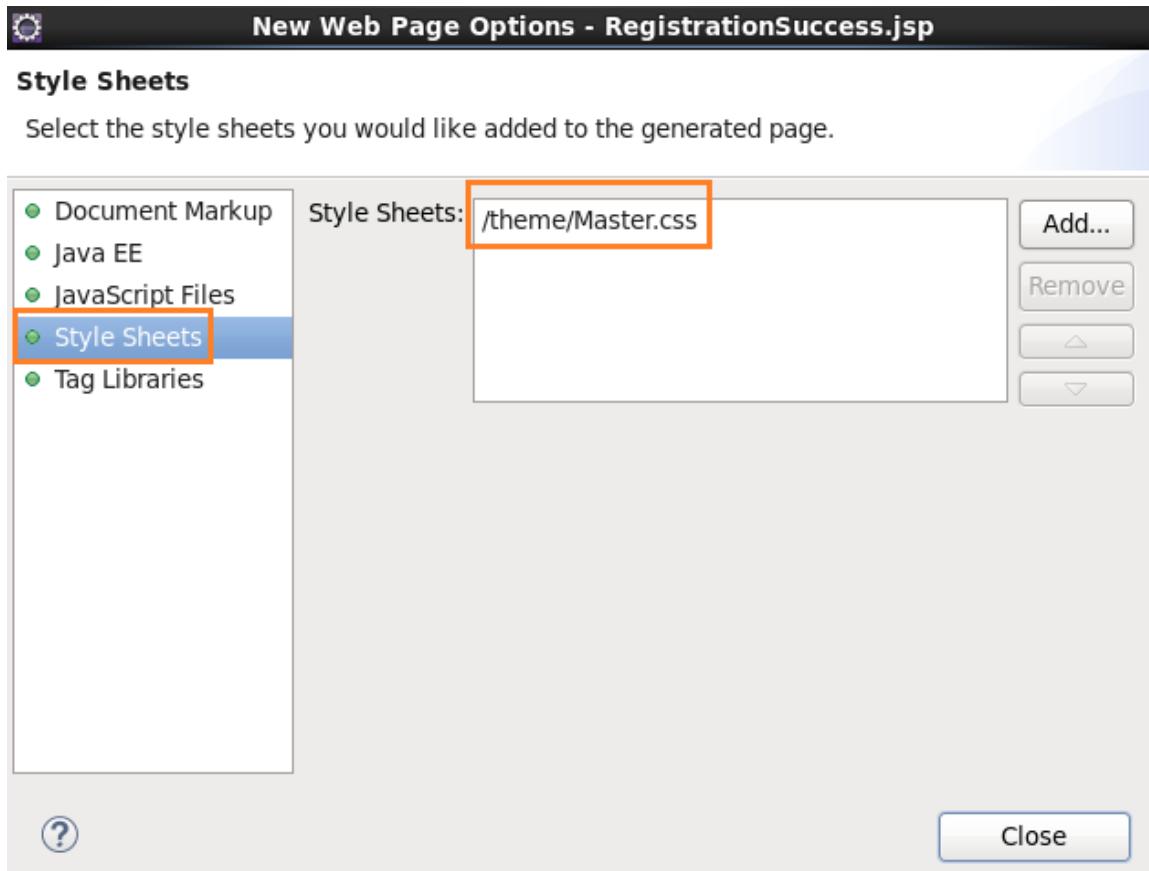
выберите в меню **New -> Web Page**. Откроется мастер создания новой страницы.

b. Укажите имя страницы **RegistrationSuccess.jsp** и нажмите кнопку **Options**.



c. Перейдите в раздел **Style Sheets** и убедитесь, что для определения стилей этой страницы уже подключен файл

Master.css.



- d. Это файл был указан в качестве стиля по умолчанию для нашего проекта и поэтому он уже выбран для создаваемой страницы. Если по какой-то причине он не указан, выберите его с помощью кнопки **Add...**.
- e. Нажмите кнопку **Close**.
- f. Нажмите кнопку **Finish** для создания новой страницы.
- g. Страница автоматически открывается в редакторе.
2. Добавьте на страницу заголовок и логику для считывания параметров.
- а. Переключитесь на вкладку **Source**.
 - б. Добавьте в тело сообщения заголовок:

```
<h1>IBM Library System</h1>
```

```
1 <!DOCTYPE HTML><%@page language="java"
2     contentType="text/html; charset=UTF-8" pageEncoding=
3%><html>
4<head>
5 <link rel="stylesheet" href="theme/Master.css" type="te
6 <title>RegistrationSuccess</title>
7 <meta http-equiv="Content-Type" content="text/html; cha
8 </head>
9<body>
10
11 <h1>IBM Library System</h1>
12
13 </body>
14 </html>
```

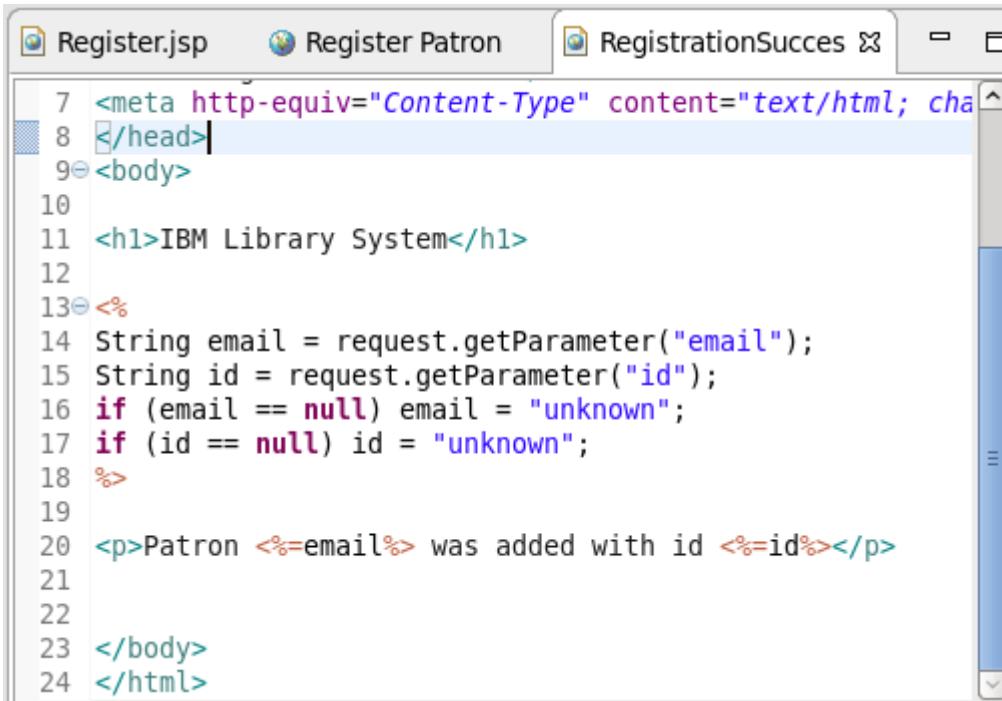
- c. Под заголовком разместите скрипlet для считывания параметров и задания значений по умолчанию:

```
<% String email = request.getParameter("email");
   String id = request.getParameter("id");
   if (email == null) email = "unknown";
   if (id == null) id = "unknown";
%>
```

3. Добавьте элемент для отображения полученных через параметры значений.

- a. Под кодом для считывания параметров добавьте следующий фрагмент:

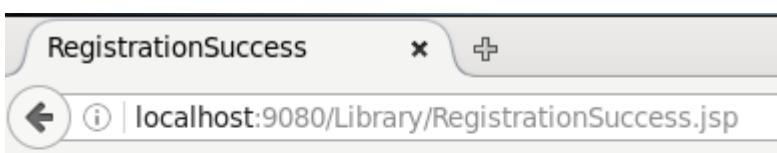
```
<p>Patron <%=email%> was added with id <%=id%></p>
```



The screenshot shows a Java IDE interface with three tabs at the top: "Register.jsp", "Register Patron", and "RegistrationSuccess". The "RegistrationSuccess" tab is active, displaying the following JSP code:

```
7 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
8 </head>
9 <body>
10 <h1>IBM Library System</h1>
11
12
13 <%>
14 String email = request.getParameter("email");
15 String id = request.getParameter("id");
16 if (email == null) email = "unknown";
17 if (id == null) id = "unknown";
18 %>
19
20 <p>Patron <%=email%> was added with id <%=id%></p>
21
22
23 </body>
24 </html>
```

- b. Сохраните сделанные изменения и закройте редактор.
4. Протестируйте работу новой страницы JSP.
- Откройте браузер **Firefox**.
 - Перейдите по адресу
http://localhost:9080/Library/RegistrationSuccess.jsp
 - Убедитесь в том, что сообщение отображается, но так как параметры не были переданы, они заменены строчками «unknown».

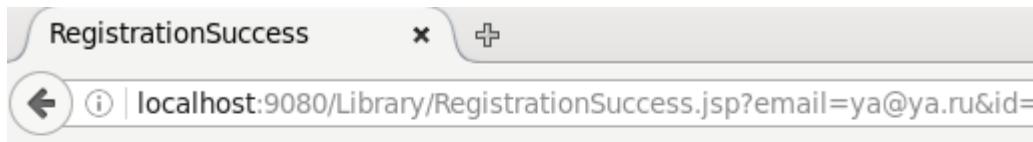


IBM Library System

Patron unknown was added with id unknown

- d. Измените URL, включив в него необходимые параметры.
Например, на такой:
http://localhost:9080/Library/RegistrationSuccess.jsp?email=ya@ya.ru&id=123456

- е. Убедитесь, что сообщение отображается, и в нем фигурируют переданные параметры.



IBM Library System

Patron ya@ya.ru was added with id 123456

Конец упражнения

Упражнение 5. Вызов JSP страниц из сервлета

О чём это упражнение:

В этой лабораторной работе сервlet **RegisterPatron** модифицируется таким образом, чтобы вызывать в нужный момент времени JSP страницы, созданные в предыдущем упражнении.

Что вы должны будете сделать:

- Воспользоваться контекстом сервлета для получения объекта типа RequestDispatcher
- Использовать RequestDispatcher для вызова JSP из сервлета
- Передать параметры в JSP
- Импортировать пакеты в JSP странице

Раздел 1. Изменение сервлета *RegisterPatron*

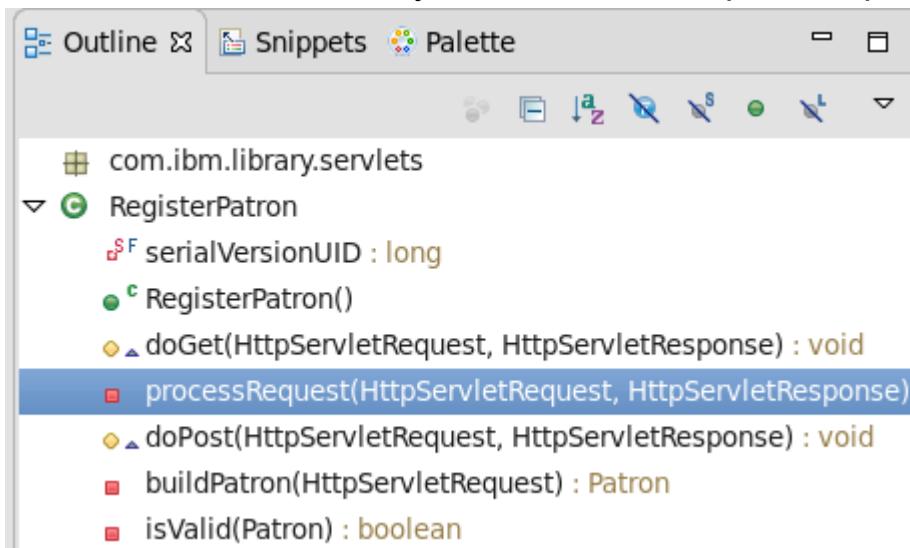
В данном разделе модифицируется метод `processRequest()`. Теперь в этом методе необходимо получить доступ к объекту `ServletContext` с помощью метода `getServletContext()`. Этот объект опосредованно (через другой объект `RequestDispatcher`) будет вызывать JSP. Логика работы сервлета также меняется в плане вызова методов для прорисовки пользовательского интерфейса: теперь вместо методов `display` будут использованы правильные JSP, которым будут передаваться соответствующие параметры.

1. Откройте файл `RegisterPatron.java`.

а. Перейдите при необходимости в перспективу **Web**.

б. Откройте файл **LibraryWebProject -> Java Resources -> src -> com.ibm.library.servlet -> RegisterPatron.java**

с. В представлении **Outline** выберите метод `processRequest()`. Этот метод автоматически будет подсвечен в редакторе.



2. Измените метод `processRequest()` таким образом, чтобы получать объект типа `ServletContext` с помощью метода `getServletContext()`.

Далее для этого объекта вызывается метод `getRequestDispatcher()`, а далее для соответствующего объекта вызывается метод `forward()`, используемый для вызова различных JSP в зависимости от состояния сервлета. Если данные клиента указаны не полностью, следует вызвать `Register.jsp` с параметром `message` (это будет строка, которая ранее выводилась с помощью метода `displayMissingFields`). Если регистрация завершена успешно, то следует вызвать `RegistrationSuccess.jsp` с параметром `message`.

(это будет строчка, которая раньше выводилась с помощью метода **displayPatronAdded**). Во всех остальных случаях вызывается **Register.jsp** с параметром **message**, который содержит пояснение по возникшей ошибке.

- Вставьте следующую строчку после строчки объявления переменной **patron**:

```
ServletContext context = getServletContext();
```

- Соответствующий класс еще не был импортирован, нажмите по лампочке рядом с новой строчкой и выберите вариант для разрешения проблемы: **Import ‘ServletContext’ (javax.servlet)**
- Замените вызов метода **displayMissingFields()** следующей строчкой:

```
context.getRequestDispatcher("/Register.jsp?message=" +  
    "Registration failed: One or more fields are " +  
    "incomplete. Please fill in all fields and " +  
    "try again.").forward(req, resp);
```

- Появившуюся ошибку можно пока проигнорировать (аналогично можно игнорировать ошибки при выполнении последующих шагов).
- Замените вызов метода **displayPatronAdded()** следующей строчкой:

```
context.getRequestDispatcher("/RegistrationSuccess.jsp?" +  
    "email=" + patron.getEmail() + "&id=" +  
    patron.getId()).forward(req, resp);
```

- Замените вызов метода **displayDuplicatePatron()** следующей строчкой:

```
context.getRequestDispatcher("/Register.jsp?message=" +  
    "Registration failed: Patron " +  
    patron.getEmail() +
```

```
" already exists and could not be " +  
"added.").forward(req, resp);
```

- g. Замените вызов метода **displaySystemError()** следующей строчкой:

```
context.getRequestDispatcher("/Register.jsp?message=" +  
"Registration failed: There's a system error. " +  
"Please try the registration again " +  
"later.").forward(req, resp);
```

- h. Замените вызов метода **displayInvalidPassword()** следующей строчкой:

```
context.getRequestDispatcher("/Register.jsp?message=" +  
"Registration failed: Patron couldn't be added. " +  
message).forward(req, resp);
```

- i. Нажмите по любой из появившихся лампочек, сигнализирующих об ошибке, и выберите вариант решения проблемы: **Add throws declaration**. В объявлении класса появляется строкка **throws ServletException**.

- j. По завершению всех манипуляций с этим методом должен получиться следующий код:

```
RegisterPatron.java
45 Patron patron = null;
46 ServletContext context = getServletContext();
47 patron = buildPatron(req);
48 if (!isValid(patron)) {
49     context.getRequestDispatcher("/Register.jsp?message=" +
50         "Registration failed: One or more fields are " +
51         "incomplete. Please fill in all fields and " +
52         "'try again.'").forward(req, resp);
53 } else {
54     try {
55         patron.add();
56         context.getRequestDispatcher("/RegistrationSuccess.jsp?" +
57             "email=" + patron.getEmail() + "&id=" +
58             patron.getId()).forward(req, resp);
59     } catch (PatronExists e) {
60         context.getRequestDispatcher("/Register.jsp?message=" +
61             "Registration failed: Patron " +
62             patron.getEmail() +
63             " already exists and could not be " +
64             "added.").forward(req, resp);
65     } catch (SystemUnavailableException e) {
66         context.getRequestDispatcher("/Register.jsp?message=" +
67             "Registration failed: There's a system error. " +
68             "Please try the registration again " +
69             "later.").forward(req, resp);
70     } catch (InvalidPassword e) {
71         String message = e.getMessage();
72         context.getRequestDispatcher("/Register.jsp?message=" +
73             "Registration failed: Patron couldn't be added. " +
74             message).forward(req, resp);
```

- k. Нажмите **Ctrl + Shift + f** для форматирования кода.
l. Удалите все неиспользуемые методы (**display...**) и неиспользуемые импорты. **Сохраните** сделанные изменения.

3. Протестируйте новую логику работы сервера.

- a. Перейдите в браузер **Firefox**.
b. Проверьте успешную регистрацию:



IBM Library System

Patron petrov@petrov.com was added with id 726472

c. Проверьте ошибку с заполнением полей:

Register Patron

localhost:9080/Library/Register

Search

IBM Library System

Register A New Patron

First name:	<input type="text"/>
Last name:	<input type="text"/>
Email address:	<input type="text"/>
Password	<input type="text"/>
Register	<input type="button" value="Register"/>

Registration failed: One or more fields are incomplete. Please fill in all fields and try again.

d. Проверьте ошибку с неправильным паролем:

Register Patron

localhost:9080/Library/Register

Search

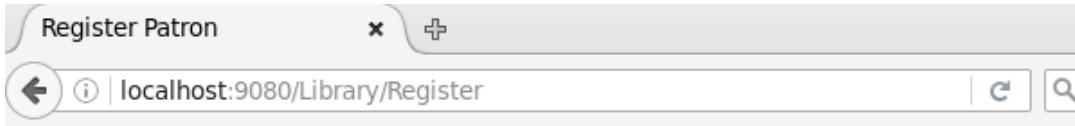
IBM Library System

Register A New Patron

First name:	<input type="text"/>
Last name:	<input type="text"/>
Email address:	<input type="text"/>
Password	<input type="text"/>
Register	<input type="button" value="Register"/>

Registration failed: One or more fields are incomplete. Please fill in all fields and try again.

е. Проверьте ошибку с дублирующей записью:



IBM Library System

Register A New Patron

First name:	<input type="text"/>
Last name:	<input type="text"/>
Email address:	<input type="text"/>
Password	<input type="password"/>
Register	<input type="button" value="Register"/>

Registration failed: Patron ivanov@mail.ru already exists and could not be added.

f. После завершения тестирования закройте браузер.

Раздел 2. Передача параметров для JSP страниц

Передача параметров через URL по нескольким причинам считается не самой лучшей практикой. Грамотный подход для решения этой задачи – это передача параметров с помощью объекта типа **HttpServletRequest**, который передается от сервлета к JSP. Метод **setAttribute()** данного объекта позволяет передать пару ключ/значение, которая может быть использована в логике JSP. Для начала, реализуем сценарий, при котором в случае успешной регистрации для JSP страницы передается объект типа **Patron**, а в случае возникновения ошибки передается объект типа **String** с текстом сообщения об ошибке.

1. Измените логику метода **processRequest()**, для реализации метода передачи данных в JSP, описанной выше.
 - а. Измените блок внутри оператора **if**, таким образом, чтобы использовать метод **setAttribute()**. Удалите передачу параметров через URL и убедитесь в том, что метод **setAttribute()** вызывается до вызова метода **forward()** объекта **RequestDispatcher()**.

```
req.setAttribute("message",
    "Registration failed: One or more fields are
    incomplete. "
```

```
+ "Please fill in all fields and try again.");  
context.getRequestDispatcher("/Register.jsp").forward(req, resp);
```

- b. Измените блок **try** таким образом, чтобы удалить передачу атрибутов **email** и **id** через URL. Вместо этого используйте метод **setAttribute()** для передачи параметра типа **Patron**. Не удаляйте метод **patron.add()**.

```
req.setAttribute("patron", patron);  
context.getRequestDispatcher("/RegistrationSuccess.jsp").forward(req, resp);
```

- c. Измените первый блок **catch** таким образом, чтобы использовать метод **setAttribute()** вместо передачи параметров через URL.

```
req.setAttribute("message",  
    "Registration failed: Patron "  
    + patron.getEmail()  
    + " already exists and could not be added.");  
context.getRequestDispatcher("/Register.jsp").forward(req, resp);
```

- d. Измените второй блок **catch** таким образом, чтобы использовать метод **setAttribute()** вместо передачи параметров через URL.

```
req.setAttribute("message",  
    "Registration failed: There is a system error. "  
    + "Please try the registration again later.");  
context.getRequestDispatcher("/Register.jsp").forward(req, resp);
```

- e. Измените третий блок **catch** таким образом, чтобы использовать метод **setAttribute()** вместо передачи параметров через URL.

```
String message = e.getMessage();  
req.setAttribute("message",  
    "Registration failed: Patron could not be added."  
    + message);
```

```
context.getRequestDispatcher("/Register.jsp").forward(req, resp);
```

- f. После внесения изменений в метод он должен выглядеть так, как показано на следующей иллюстрации (показан фрагмент кода метода):

```
RegisterPatron.java
49     if (!isValid(patron)) {
50         // no, one or more fields are missing
51         req.setAttribute("message",
52             "Registration failed: One or more fields are incomplete. "
53             + "Please fill in all fields and try again.");
54         context.getRequestDispatcher("/Register.jsp").forward(req, resp);
55     } else {
56         // try to add the new patron
57         try {
58             patron.add();
59             req.setAttribute("patron", patron);
60             context.getRequestDispatcher("/RegistrationSuccess.jsp").forward(req, resp);
61         } catch (PatronExists e) {
62             req.setAttribute("message",
63                 "Registration failed: Patron "
64                 + patron.getEmail()
65                 + " already exists and could not be added.");
66             context.getRequestDispatcher("/Register.jsp").forward(req, resp);
67         } catch (SystemUnavailableException e) {
68             req.setAttribute("message",
69                 "Registration failed: There is a system error. "
70                 + "Please try the registration again later.");
71             context.getRequestDispatcher("/Register.jsp").forward(req, resp);
72         } catch (InvalidPassword e) {
73             String message = e.getMessage();
74             req.setAttribute("message",
75                 "Registration failed: Patron could not be added. " + message);
76             context.getRequestDispatcher("/Register.jsp").forward(req, resp);
77         }
78     }
79 }
```

- g. Сохраните сделанные изменения и убедитесь в отсутствии ошибок.

2. Адаптируйте JSP страницу **RegistrationSuccess.jsp** для чтения параметров с помощью метода **getAttribute()** вместо чтения их из URL с помощью **getParameter()**.

- С помощью представления **Enterprise Explorer** откройте файл **LibraryWebProject -> WebContent -> RegistrationSuccess.jsp**.
- Перейдите на вкладку **Source**.
- Замените код по заполнению/считыванию данных для отображения на страницы на следующий:

```
Patron patron = (Patron)
request.getAttribute("patron");
String email = "unknown";
String id = "unknown";
if (patron != null) {
```

```
email = patron.getEmail();
id = String.valueOf(patron.getId());
}
```

d. Сохраните сделанные изменения.

e. Так как для этой JSP страницы не был импортирован класс Patron, возникает сообщение об ошибке. Для решения этой проблемы добавьте в директиву @page атрибут для импорта нужного класса. В итоге она должна выглядеть следующий образом:

```
<%@page language="java" contentType="text/html;
charset=UTF-8" pageEnconding="UTF-8"
import="com.ibm.library.model.Patron" %>
```

3. Измените страницу **Register.jsp** чтения параметров с помощью метода **getAttribute()** вместо считывания их из URL с помощью **getParameter()**.

- С помощью представления **Enterprise Explorer** откройте файл **LibraryWebProject -> WebContent -> Register.jsp**.
- Замените директивы для получения, заполнения и отображения параметра **message** на следующий код:

```
String message = (String)
request.getAttribute("message");
if (message == null) {
    message = "";
}
try {
    out.println(message);
} catch (IOException e) {
    System.out.println("Exception occurred in
Register.jsp: " + e);
}
```

c. В данном случае изменяется только метод для получения значения переменной **message**.

d. Добавьте директиву для импорта необходимого класса в JSP.
После открывающего тега **<HEAD>** добавьте следующую строчку:

```
<%@page import="java.io.IOException" %>
```

- e. Сохраните сделанные изменения, убедитесь в отсутствии ошибок.
4. Повторите процедуру тестирования из первого раздела этой лабораторной работы для проверки успешного сценария и всех вариантов возникновения ошибок.

Раздел 3. Сохранение введенных значений

В текущей реализации приложения осталась нерешенная проблема: после возникновения той или иной ошибки, данные, введенные пользователем, пропадают из формы, и их нужно вводить заново.

Очевидно, это не является ожидаемым и правильным поведением для такого рода приложения. В данном разделе логика сервлета RegisterPatron и Register.jsp будет изменена как раз для сохранения этих значений. Достигается это путем передачи атрибута типа Patron в JSP при каждом обращении. Это даст возможность всегда отобразить актуальные введенные данные.

1. Измените сервlet для сохранения объекта **Patron** в качестве атрибута запроса перед вызовом **Register.jsp**.
 - a. Откройте файл **RegisterPatron.java**. Найдите строчку:

```
patron = buildPatron(req);
```

После нее добавьте еще одну строчку:

```
req.setAttribute("patron", patron);
```

- b. Удалите строчку `req.setAttribute("patron", patron)` в блоке **try**. Она больше не требуется, так как раньше использовалась для передачи данных в успешном сценарии, а теперь это будет происходить каждый раз.
 - c. Сохраните сделанные изменения.
2. Измените **Register.jsp** для считывания объекта типа **Patron** в соответствующую переменную и отображения имеющих место полей в форме для ввода.
 - a. Откройте страницу **Register.jsp** в редакторе.

- b. После заголовка **<H2>**, но до формы **<FORM>** добавьте следующий блок кода для считывания объекта типа **Patron**:

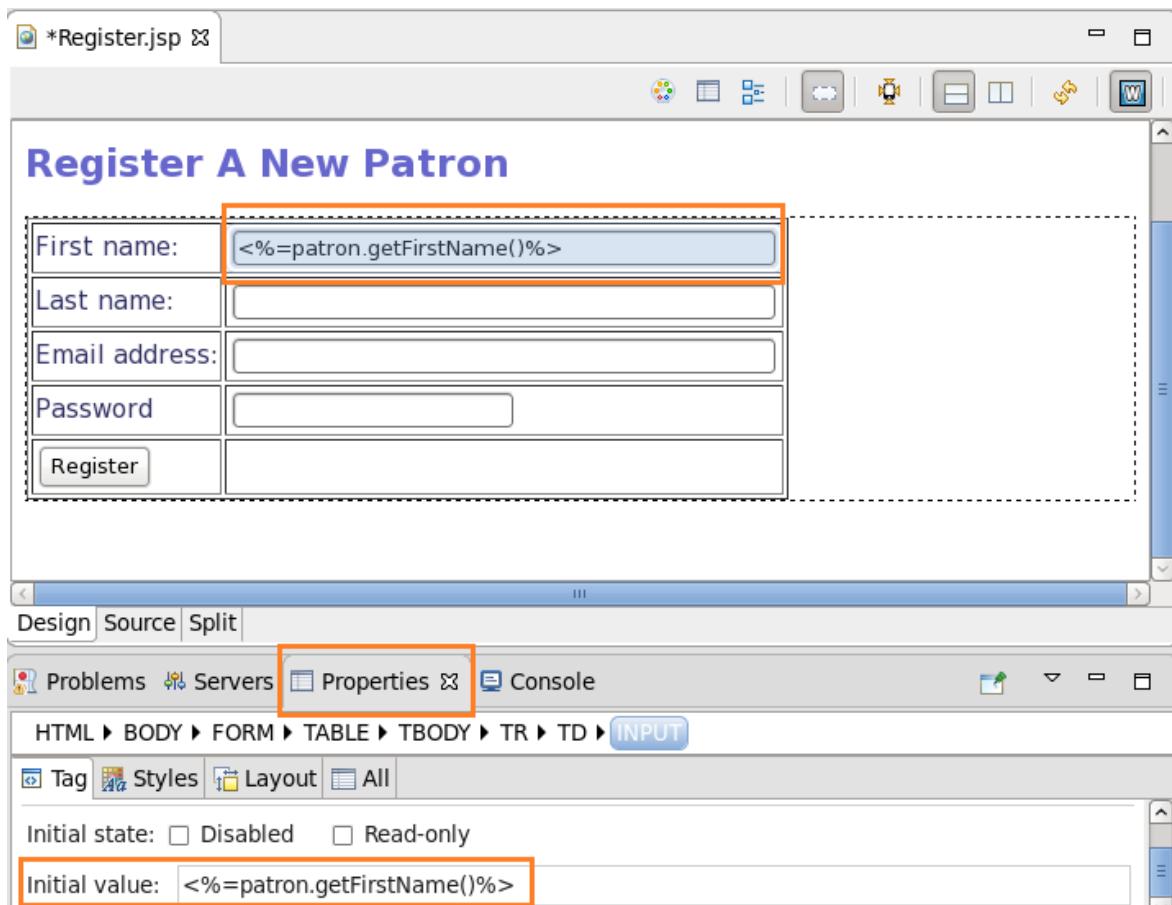
```
<% Patron patron = (Patron)
request.getAttribute("patron");
if (patron == null) {
    patron = new Patron();
}%>
```

- c. Появляется сообщение об ошибке. Для его устранения измените директиву для импорта классов, добавив туда указание на нужный класс **Patron**. Результирующая директива должна выглядеть следующим образом:

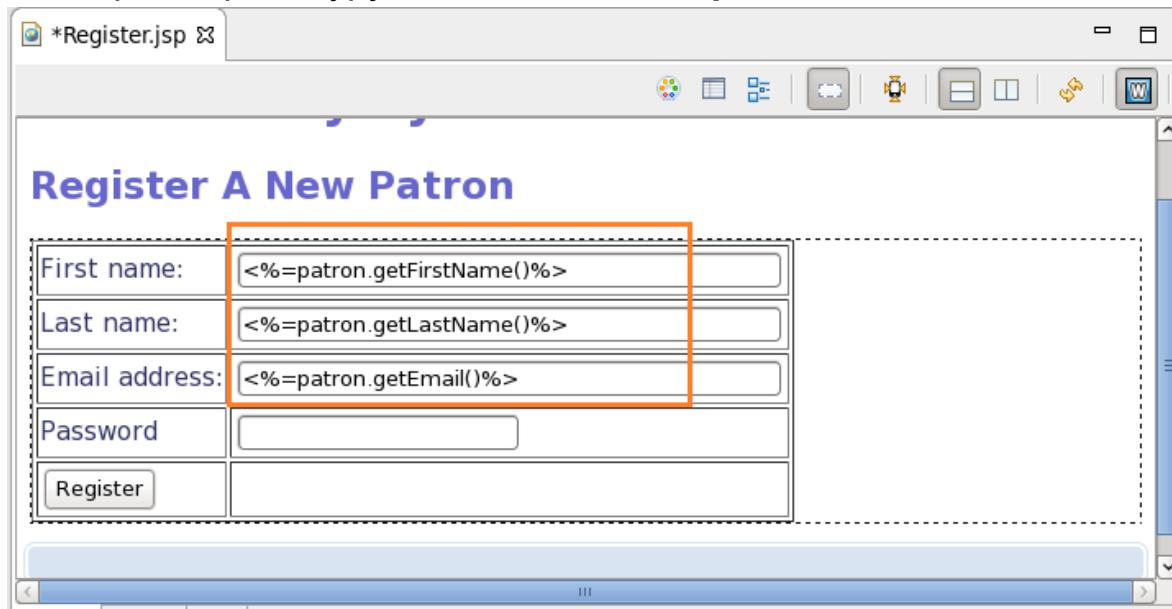
```
<%@page
import="java.io.IOException, com.ibm.library.model.Pa
tron"%>
```

- d. **Сохраните** сделанные изменения.
e. В редакторе страницы **Register.jsp** перейдите на вкладку **Design**.
f. Выберите поле для ввода, в котором указывается **имя (First name)**. Откройте представление **Properties**.

g. Введите `<%=patron.getFirstName()%>` в поле **Initial value**.



h. Повторите процедуру для полей ввода фамилии и email.



3. Проведите тестирование успешного сценария, а также кейсов с ошибками для проверки сохранения данных в форме.

Конец упражнения

Упражнение 6. Управление сессиями

О чём это упражнение:

В этой лабораторной работе разрабатывается система, которая позволяет проходить аутентификацию в системе и смотреть перечень заказанных книг. Для работы этой системы понадобится сделать следующее:

- Сервлет для прохождения аутентификации
- JSP для ввода учетной записи/пароля
- Сервлет для проверки учетных данных
- Сервлет для отображения списка заказанных книг
- JSP для отображения списка книг.

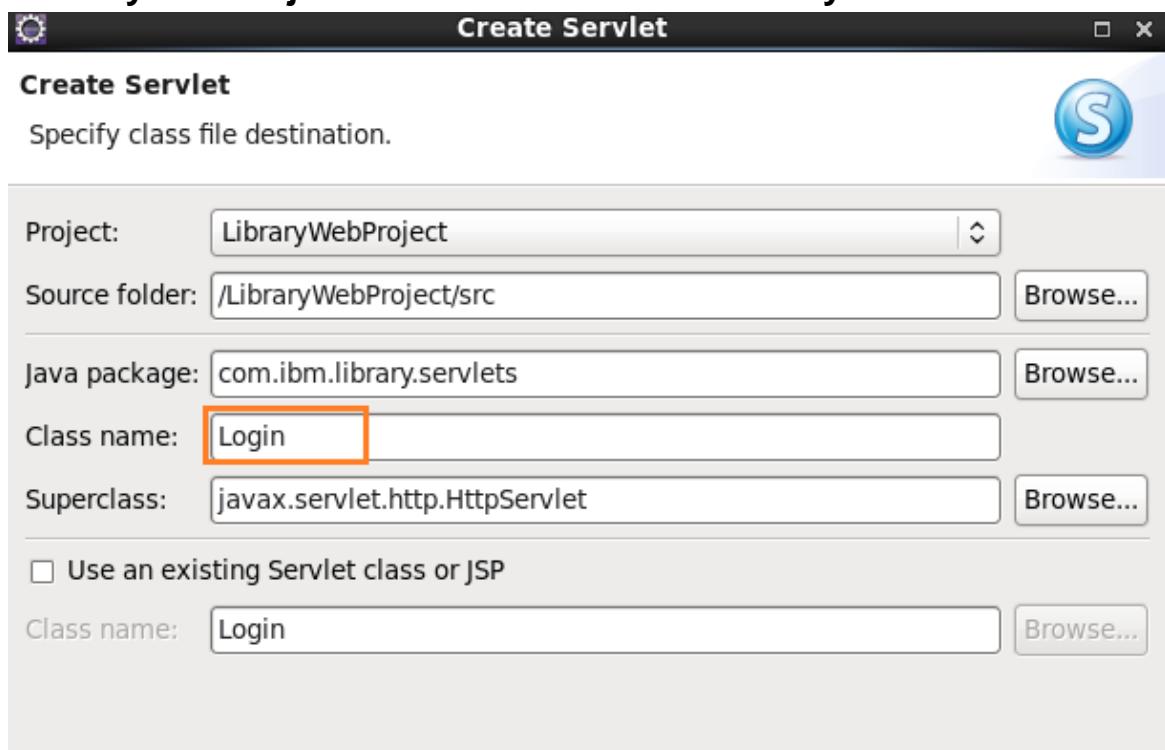
Что вы должны будете сделать:

- Научиться использовать объект HttpSession
- Реализовать собственную систему аутентификации
- Сохранять объект в разделяемой сессии
- Воспользоваться Cookies для получения данных аутентификации.

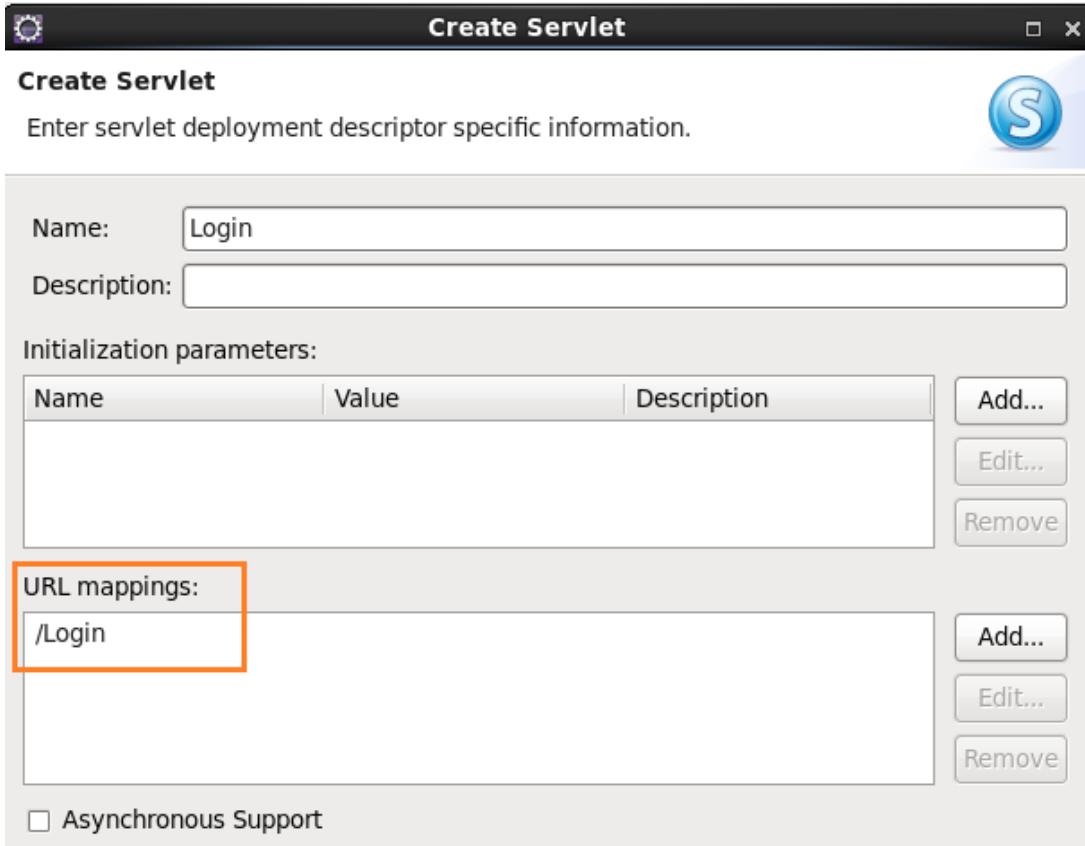
Раздел 1. Создание сервлета для аутентификации

В этой части лабораторной работы создается новый сервлет **Login**. Доступ к этому сервлету будет осуществляться по URL: **/Library/Login**. Для отображения формы сервлет передает управление JSP.

1. Создайте новый сервлет **Login.java** в пакете **com.ibm.library.servlets** в проекте **LibraryWebProject**. Необходимо связать сервлет с URL **/Login**. Реализуйте методы **doGet()** и **doPost()**, чтобы вызывать метод **processRequest()**.
 - a. Переключитесь в перспективу **Web**, если это необходимо.
 - b. В представлении **Enterprise Explorer** раскройте каталог **LibraryWebProject -> Java Resources -> src**.
 - c. Нажмите правой кнопкой мыши по пакету **com.ibm.library.servlets** и выберите **New -> Servlet**. Открывается диалог по созданию нового сервлета.
 - d. Укажите имя **Login**. Убедитесь, что выбран каталог **/LibraryWebProject/src** и пакет **com.ibm.library.servlets**.



e. Нажмите **Next**. Убедитесь, что указан URL **/Login** в списке **URL Mappings**.



f. Нажмите **Next**.

g. Нажмите **Finish**.

2. Реализуйте метод **processRequest()**, который вызывается из методов **doGet()** и **doPost()**, принимает идентичные входные параметры и выбрасывает идентичные исключения. В этом методе передается управление Login JSP (которая еще пока не создана). Для того, чтобы доступ к этой JSP нельзя было получить непосредственно по ссылке, она размещается в каталоге **WEB-INF**, что находит отражение в ссылке, которую нужно использовать в реализуемом методе.

a. Напишите следующий код метода **processRequest()**:

```
private void processRequest(HttpServletRequest request,  
                           HttpServletResponse response)  
throws ServletException, IOException {  
    ServletContext context = getServletContext();  
    context.getRequestDispatcher("/WEB-INF/jsp" +  
        "/Login.jsp").forward(request, response);
```

}

- b. Разрешите возникающую ошибку, нажав по пиктограмме лампочки, сигнализирующей об ошибке и выберите вариант решения проблемы: **Import `ServletContext` (javax.servlet)**.
3. Напишите реализацию методов **doGet()** и **doPost()**.
 - a. Замените тело обоих методов единственным оператором – вызовом метода **processRequest**:

```
processRequest(request, response);
```

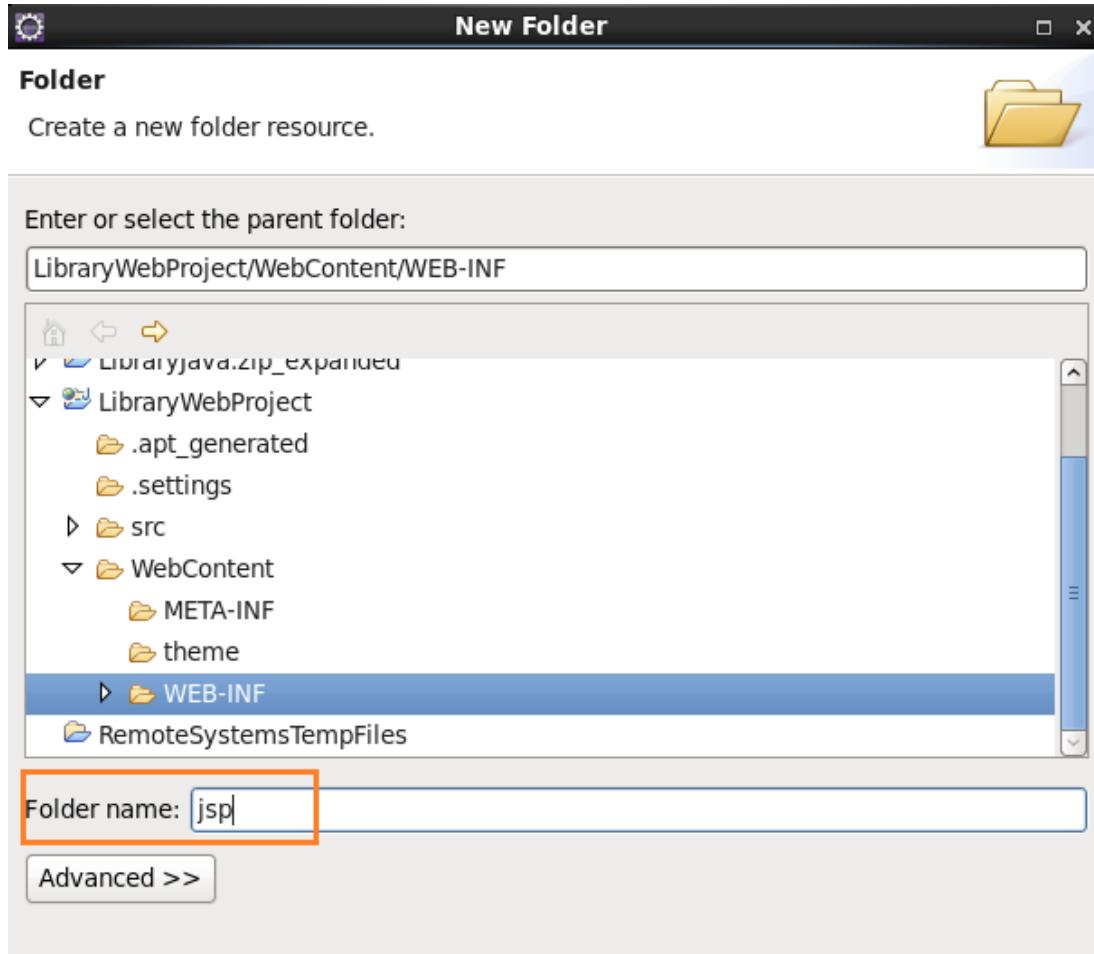
- b. Сохраните все сделанные изменения и закройте файл **Login.java**.

Раздел 2. Создание страницы Login JSP

В этой части лабораторной работы создается JSP страница **Login.jsp**. На этой странице отображается форма для ввода логина и пароля. Эта форма будет передаваться сервлету **ProcessLogin** для дальнейшей обработки. Если в сервлете возникает та или иная ошибка, но пользователь остается на форме, отображаемой с помощью **Login.jsp**. При этом сервлет передает информацию о сути произошедшей ошибки, которая должна отобразиться на странице. Для запрета непосредственного доступа на эту страницу по URL, она размещается в каталоге **WEB-INF**. В нем создается подраздел **jsp**.

1. Создайте каталог **jsp** в директории **WEB-INF**.
 - a. В представлении **Enterprise Explorer** нажмите правой кнопкой мыши по каталогу **LibraryWebProject -> WebContent -> WEB-INF**. Выберите **New -> Folder**. Открывается мастер по созданию нового каталога.

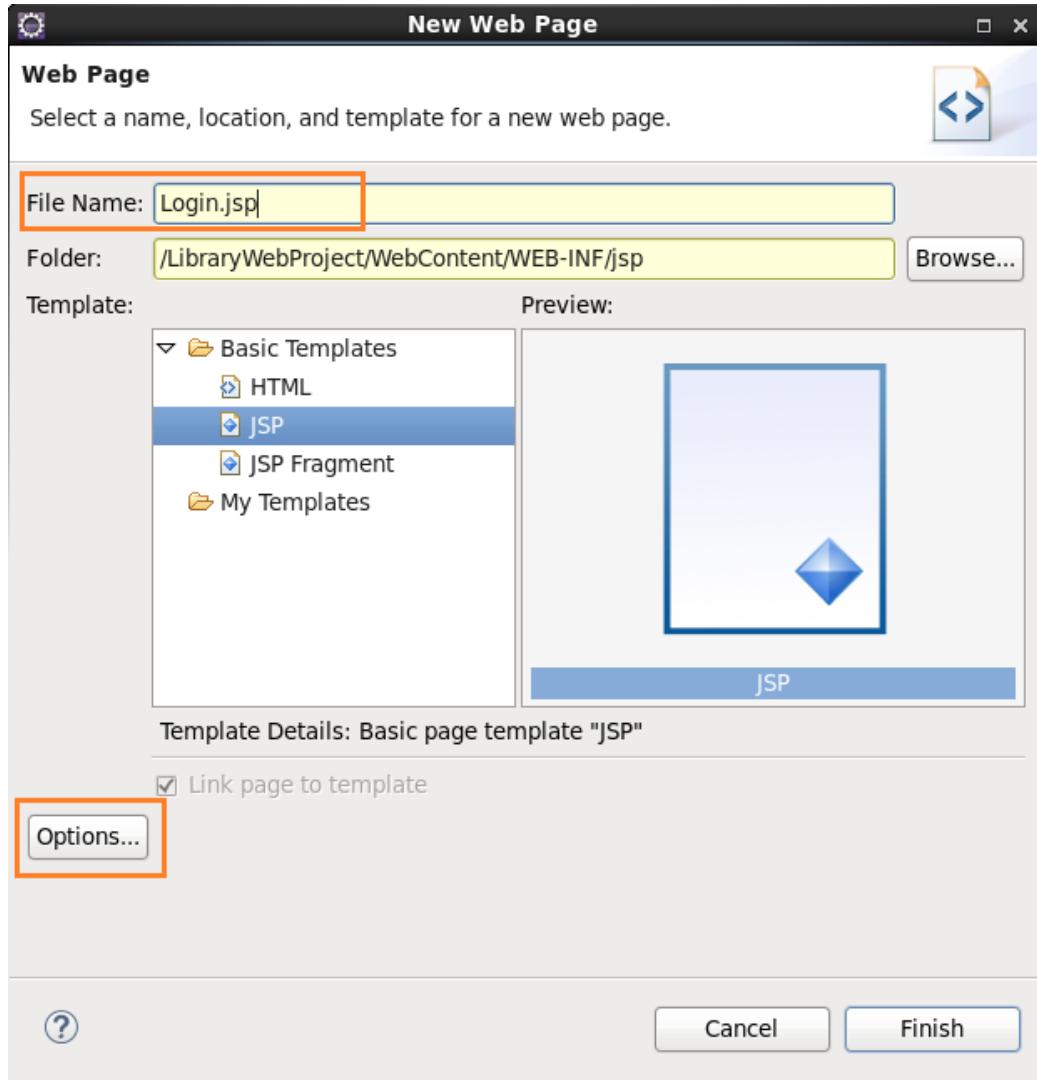
b. Укажите имя каталога **jsp** и нажмите **Finish**.



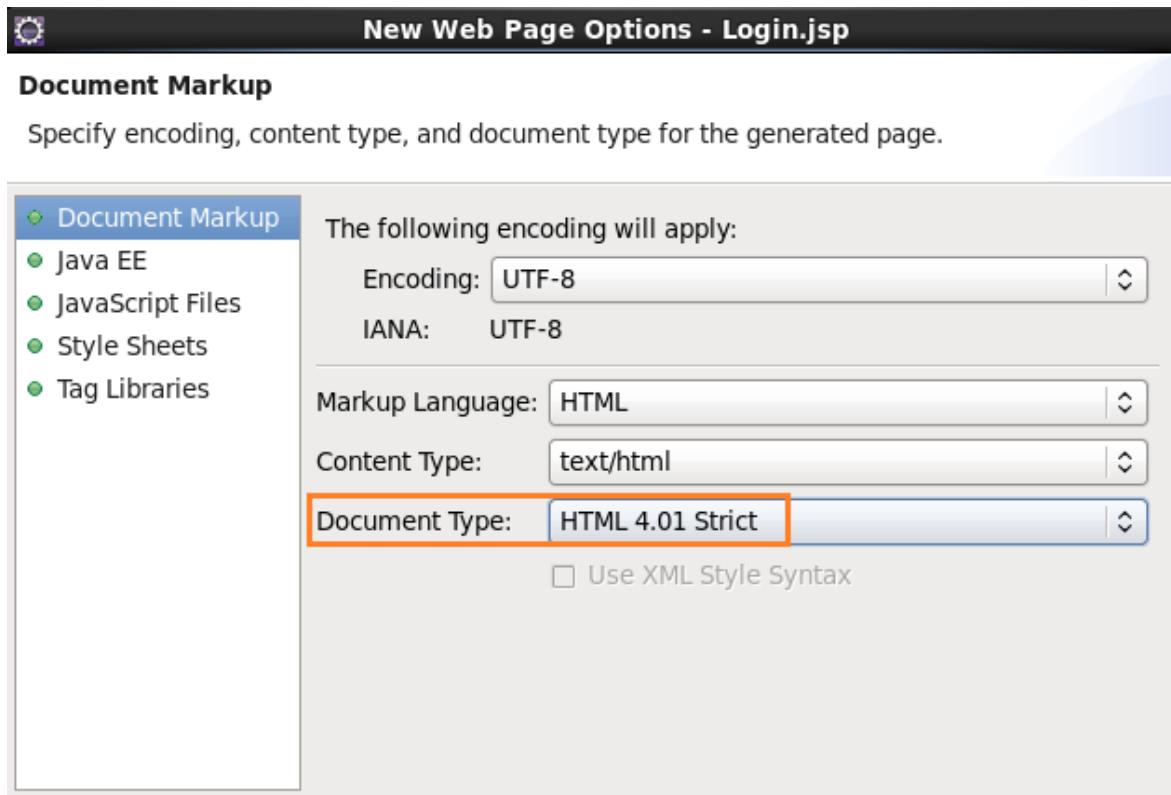
2. Создайте страницу **Login.jsp**.

а. В представлении **Enterprise Explorer** нажмите правой кнопкой мыши по только что созданному каталогу **jsp**. Выберите **New -> Web Page**.

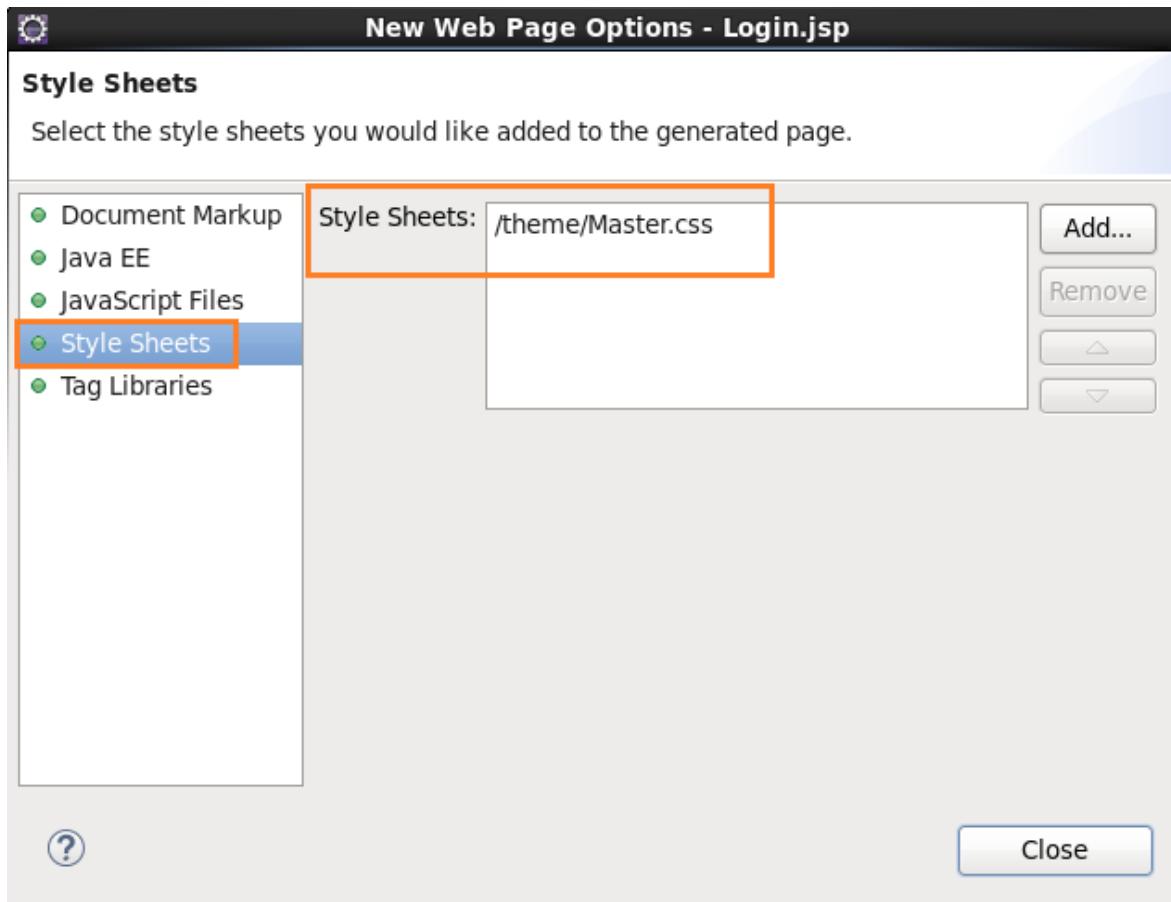
b. Укажите имя **Login.jsp**. Нажмите кнопку **Options**.



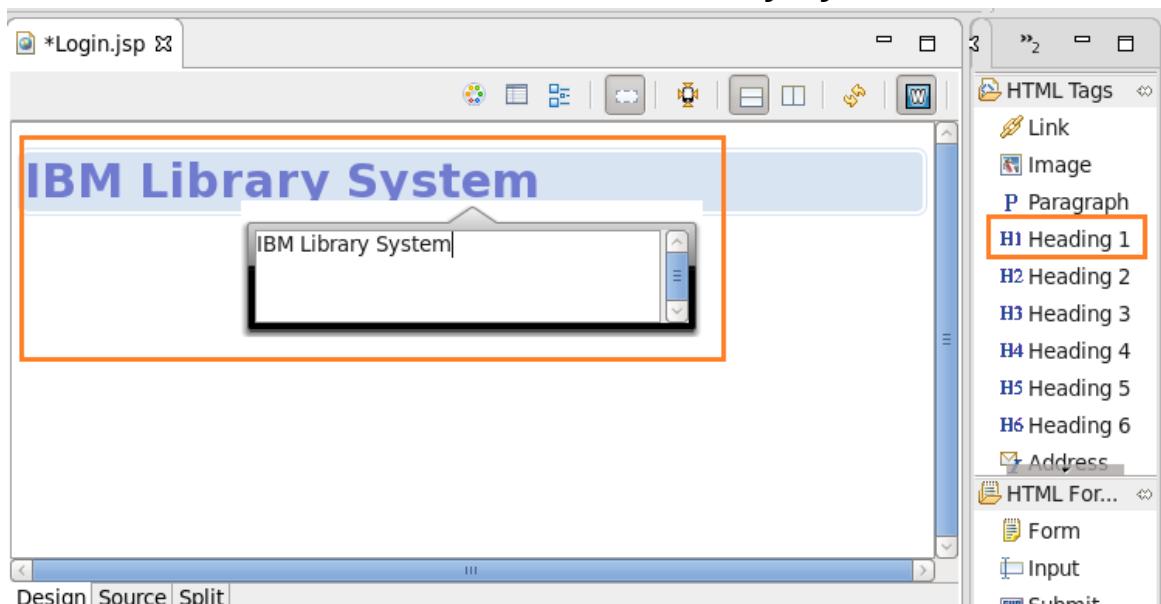
- c. На странице настроек страницы укажите **Document Type – HTML 4.01 Strict.**



- d. На вкладке **Style Sheets** убедитесь, что выбран файл **/theme/Master.css**.

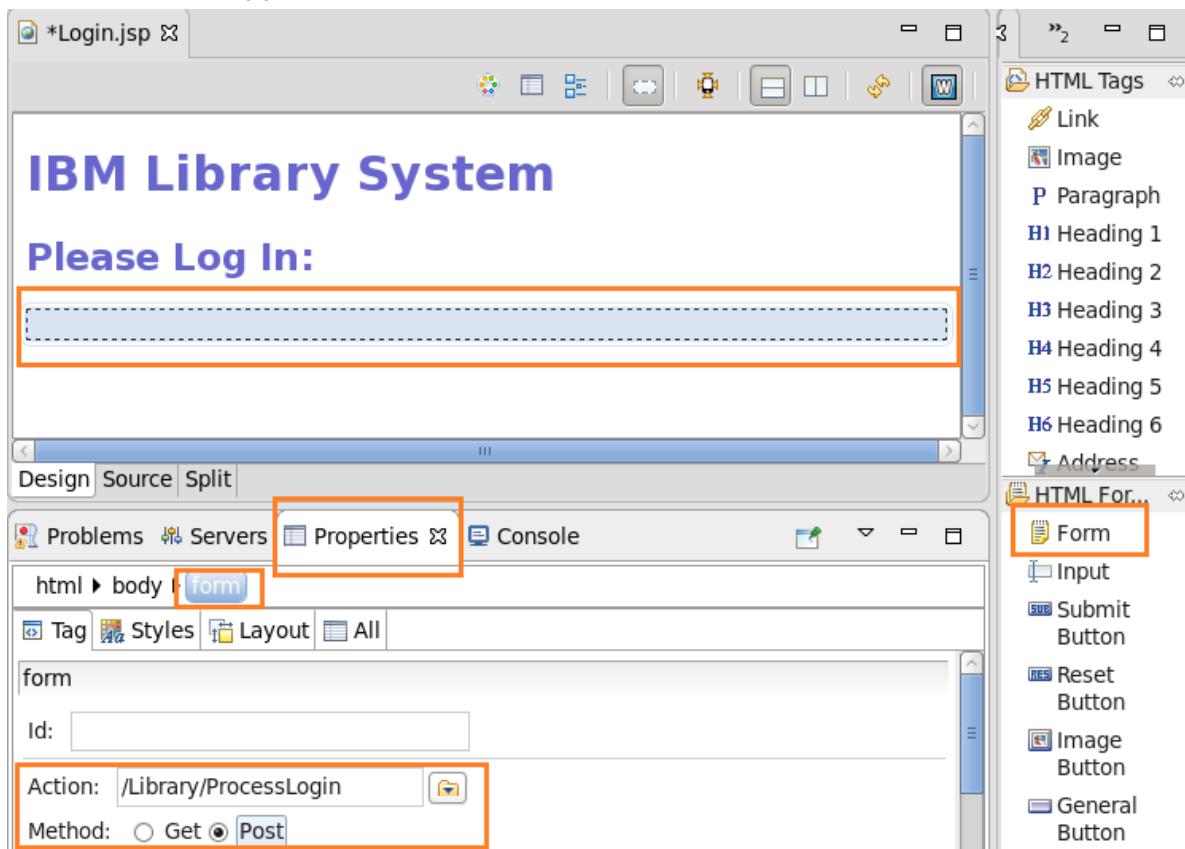


- e. Нажмите **Close**. Нажмите **Finish**, завершая работу мастера по созданию новой страницы.
 - f. Создается JSP страница и открывается в редакторе. Перейдите на вкладку **Design**.
3. Сделайте на странице форму для ввода логина и пароля, которые будут передаваться серверу **ProcessLogin** под именами **PATRON_ID** и **PASSWORD** соответственно. Можно использовать табличную верстку. Для пароля нужно использовать соответствующее поле ввода (маскирующее введенные данные). Кнопка **Submit** для этой формы будет отправлять запрос на URL **/Library/ProcessLogin**, за которым будет стоять соответствующий сервер (пока еще он не написан).
- a. Выполните перенос элемента **Heading 1** из палитры (**Palette**) в редактор для страницы **Login.jsp**. Дважды кликните по нему левой кнопкой мыши так, чтобы открылся интерфейс (форма) для ввода текста заголовка. Введите текст **IBM Library System**.

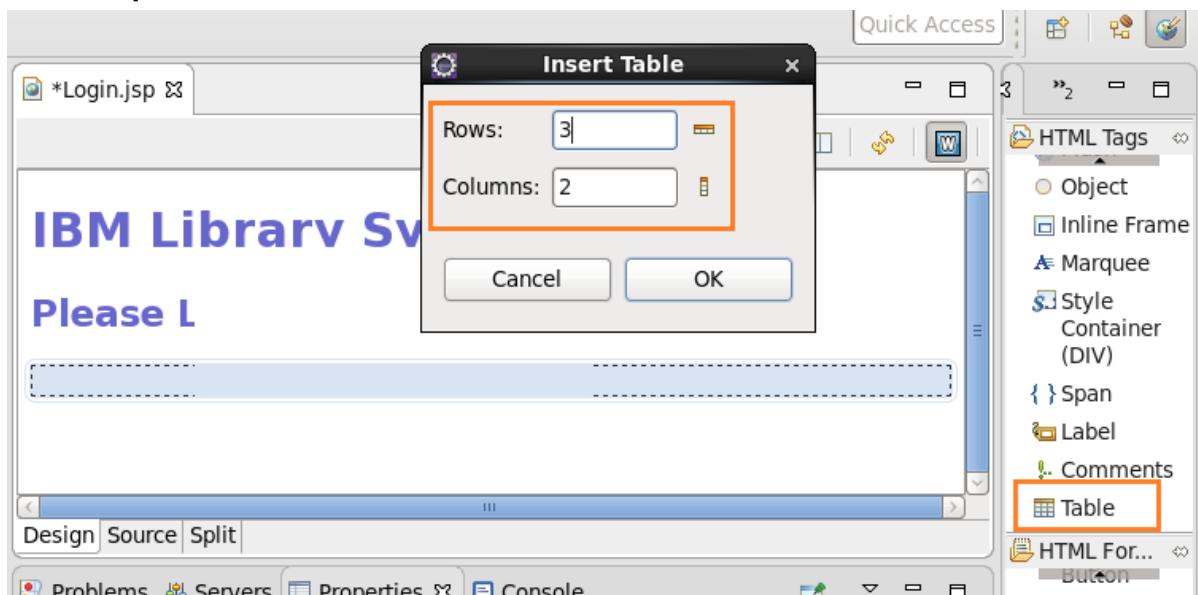


- b. По тому же сценарию разместите под первым заголовком второй типа **Heading 2** с текстом **Please Log in**:
- c. Ниже второго заголовка разместите элемент **Form**.
- d. Выберите элемент **Form** в редакторе и перейдите в представление **Properties**.
- e. В поле **Action** напишите **/Library/ProcessLogin**.

f. Укажите метод **Post**.



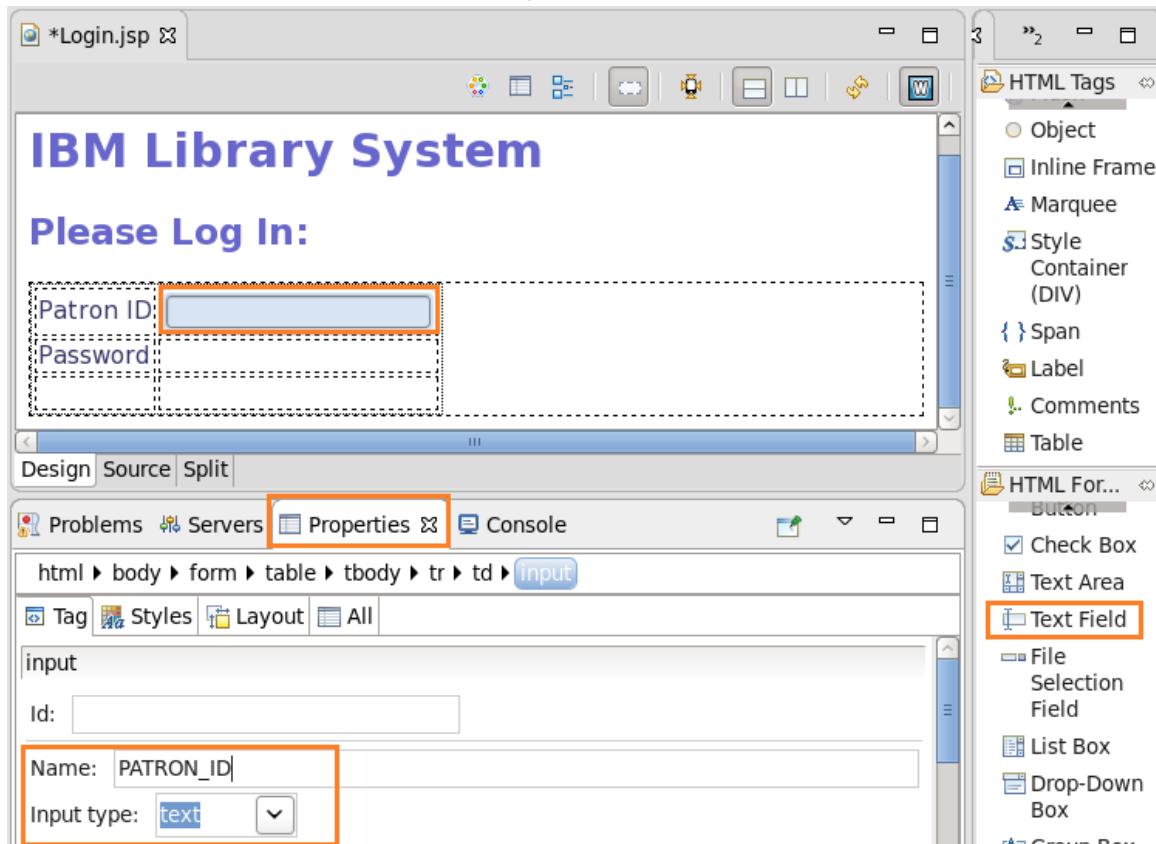
g. В редакторе еще раз выберите созданную форму. Из палитры на форму вынесите элемент **Table**. Будет задан вопрос относительно количества столбцов и строк в этой таблице. Укажите **3 строки и 2 столбца**.



h. В первой строке, первой колонке введите «**Patron ID:**». Для ввода нужно дважды кликнуть по этой ячейке и дождаться открытия формы, в которой происходит ввод.

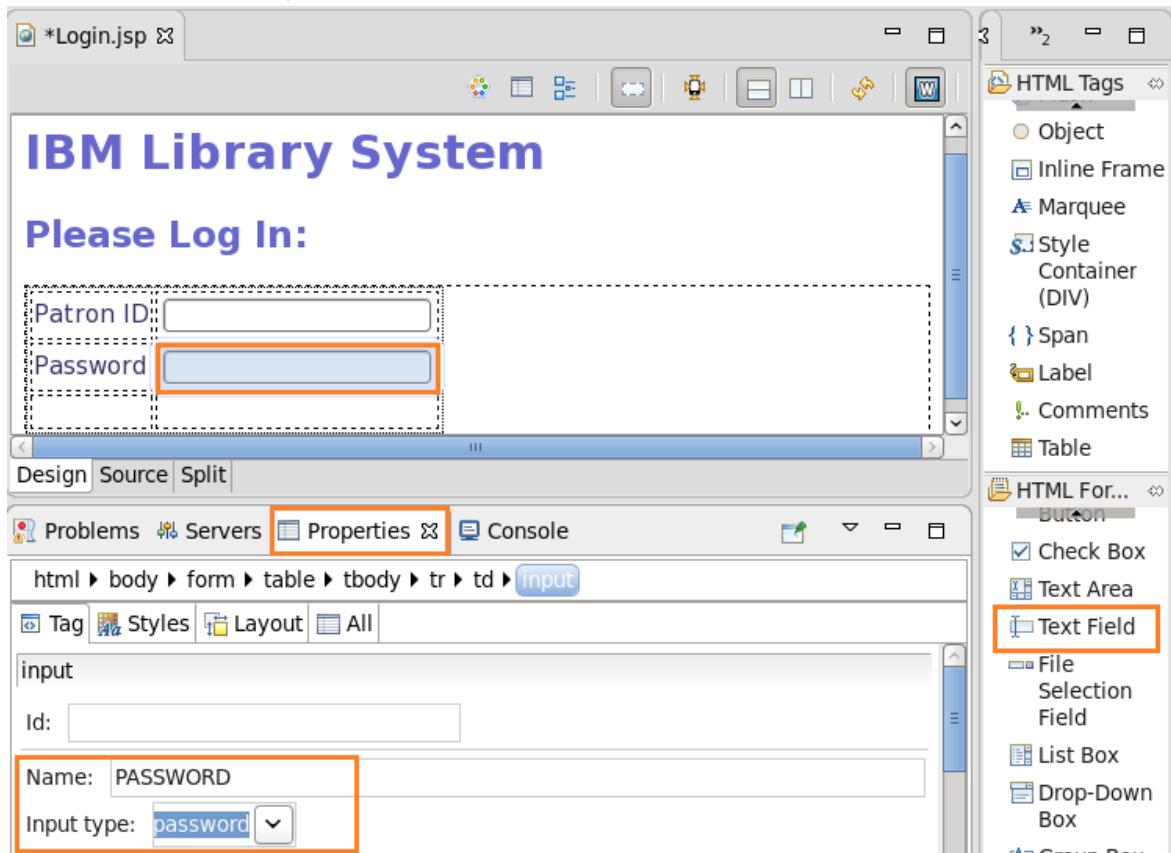
i. Во второй строке, первой колонке введите «**Password**».

- j. Вставьте элемент **Text Field** в первую строку, вторую колонку, вытащив его из палитры. Выберите его и перейдите на вкладку **Properties**. Так укажите имя **PATRON_ID** и тип **text**. (В Eclipse той версии, которая используется в лабораторном практикуме иногда добавленный на форму элемент для ввода отображается в виде трех полей. Это ошибка отображения. Если такая ситуация возникает, то для установки параметров, можно выбирать любой из них. При этом можно убедиться, что по факту поле для ввода одно, если перейти на вкладку **Source** редактора).

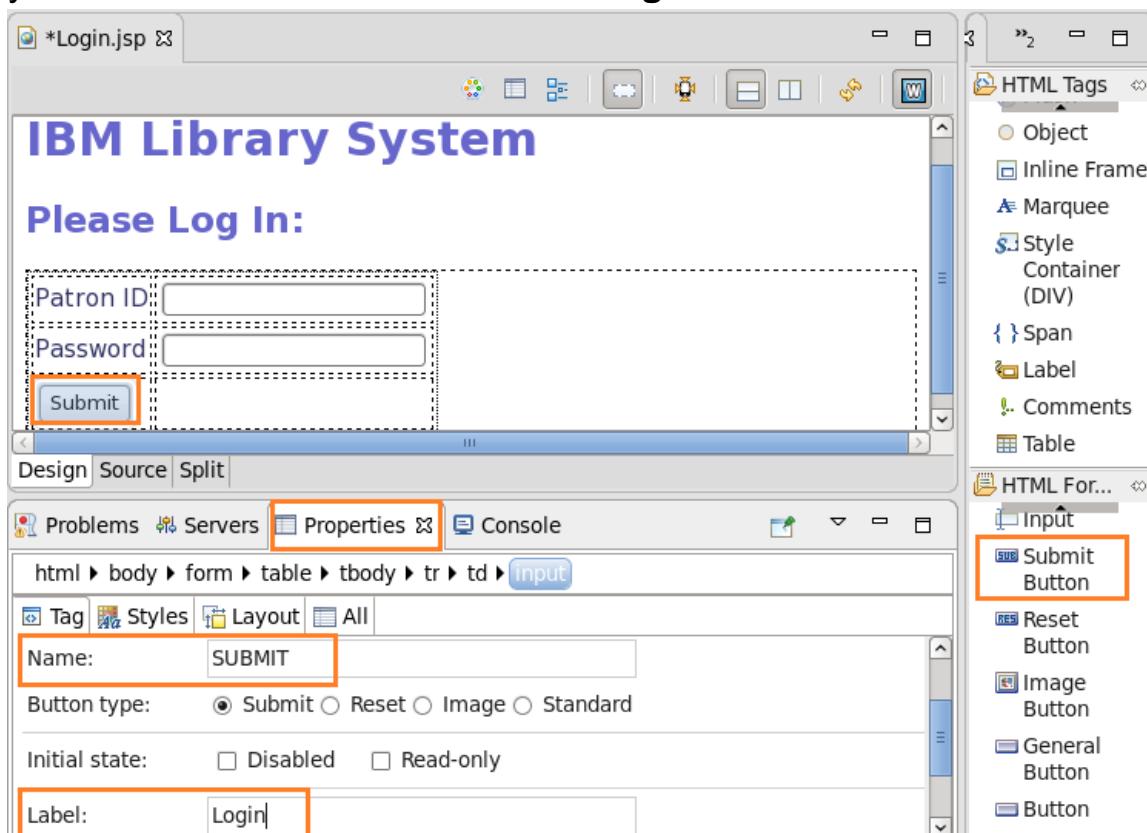


- k. Вставьте элемент **Text Field** во вторую строку, вторую колонку, вытащив его из палитры. Выберите его и перейдите на вкладку

Properties. Так укажите имя **PASSWORD** и тип **password**.



- I. Вставьте элемент **Submit Button** в третью строку, первый столбец. Выделите его и перейдите на вкладку **Properties**. Там укажите имя – **SUBMIT** и Label – **Login**.



м. **Сохраните** сделанные изменения.

н. Перейдите на вкладку **Source**. Там вы должны увидеть разметку, соответствующую той, что показана ниже:

```

6 <link rel="stylesheet" href="../../theme/Master.css" type="text/css">
7 <title>Login</title>
8 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
9 </head>
10<body>
11     <h1>IBM Library System</h1>
12     <h2>Please Log in:</h2>
13     <form action="/Library/ProcessLogin" method="post">
14         <table>
15             <tbody>
16                 <tr>
17                     <td>Patron ID</td>
18                     <td><input type="text" name="PATRON_ID"></td>
19                 </tr>
20                 <tr>
21                     <td>Password</td>
22                     <td><input type="password" name="PASSWORD"></td>
23                 </tr>
24                 <tr>
25                     <td><input type="submit" name="SUBMIT" label="Login"></td>
26                     <td></td>
27                 </tr>
28             </tbody>
29         </table>
30     </form>
31
32 </body>
33 </html>

```

о. Название атрибута **label** для элемента **input** подсвечено как некорректное. Замените слово **label** на **value**.

р. **Сохраните** сделанные изменения. Убедитесь, что предупреждение по поводу этого атрибута пропало.

4. Добавьте на страницу элемент, позволяющий отобразить сообщение от сервера.

а. Перейдите на вкладку **Source**.

б. Добавьте в конец кода JSP страницы (до закрывающего тега **</BODY>**) следующие строки:

```

<%String message =
(String)request.getAttribute("MESSAGE");%>
<h4><%= (message == null) ? "" : message %></h4>

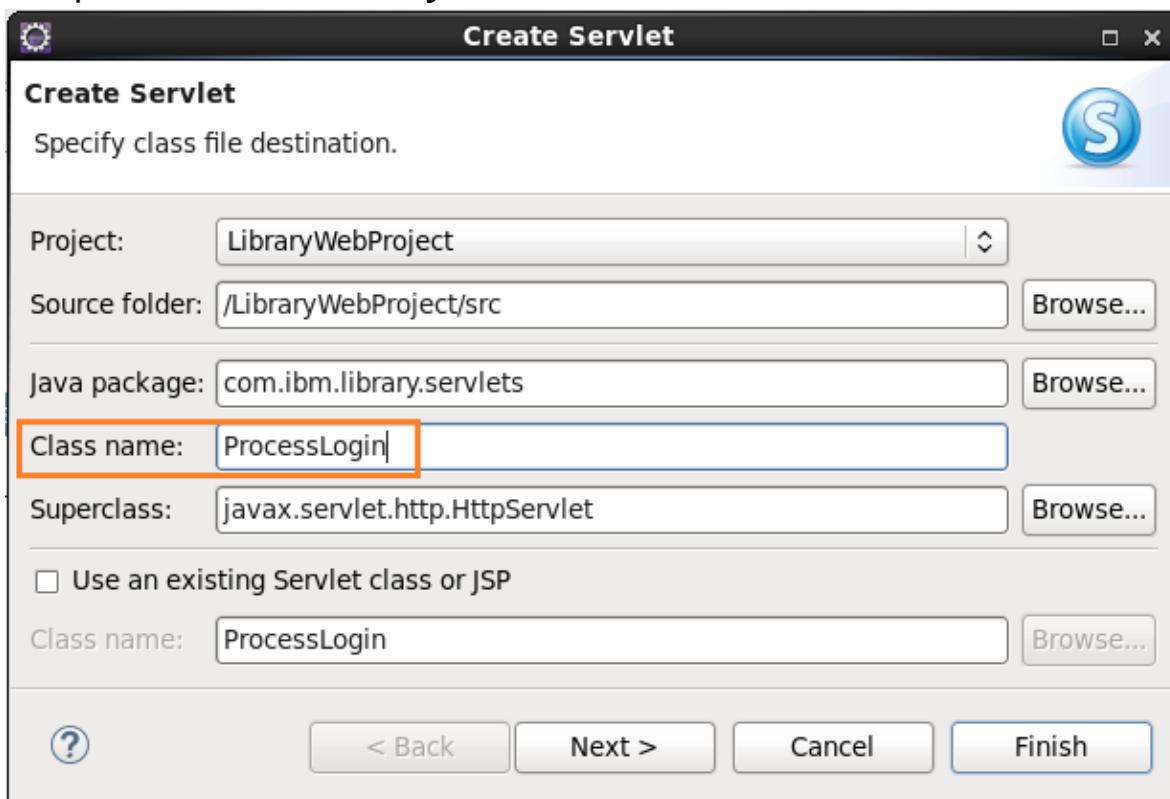
```

с. **Сохраните** сделанные изменения.

Раздел 3. Создание сервлета ProcessLogin

В этой части лабораторной работы создается сервлет ProcessLogin для обработки данных в процессе аутентификации. После извлечения данных учетной записи и из проверки, сервлет создает сессию. Объект типа Patron сохраняется в сессии и может быть извлечен из нее в любом интересующим нас сервлете до тех пор, пока не завершится сеанс работы пользователя. Это в частности позволяет нам не обращаться к базе данных каждый раз, при необходимости получить какие-то данные об этом объекте. После прохождения аутентификации пользователь должен попасть на страницу со списком книг (соответствующий сервлет и JSP будут разработаны позже).

1. В проекте **LibraryWebProject** создайте новый сервлет **ProcessLogin** в пакете **com.ibm.library.servlets**. Задайте для него URL **/ProcessLogin**. В методах **doGet()** и **doPost()** используйте вызов метода **processRequest()** для обработки запросов.
 - a. В представлении **Enterprise Explorer** нажмите правой кнопкой мыши по **LibraryWebProject -> Java Resources -> src -> com.ibm.library.servlets** и выберите **New -> Servlet**. Откроется мастер создания нового сервлета.
 - b. Введите **ProcessLogin** в поле **Class name**. Убедитесь, что в поле **Folder** выбрано **\LibraryWebProject\src**, а в поле **Java package** выбрано **com.ibm.library.servlets**. Нажмите **Next**.



- c. На следующей странице убедитесь, что выбран URL **/ProcessLogin** и нажмите **Next**.
 - d. На следующей странице убедитесь, что выбраны опции **Public**, **doGet** и **doPost** и нажмите **Finish**.
2. Реализуйте логику работы сервлета таким образом, чтобы извлекать параметры формы с данными аутентификации и вызывать метод **verifyLogon()** объекта **Patron** для их проверки. В случае успешной аутентификации, необходимо создать сессию (**HttpSession**), сохранить в нее объект типа **Patron** и передать управление сервлету **ProcessListItems**.
- a. Добавьте метод **processRequest** со следующей декларацией:

```
private void processRequest  
(HttpServletRequest req., HttpServletResponse resp)  
throws ServletException, IOException { }
```

- b. Реализуйте методы **doGet** и **doPost**, заменив их код на вызов метода **processRequest()**:

```
processRequest(request, response);
```

- c. В метод **processRequest()** добавьте строчки для получения контекста и считывания параметров формы:

```
ServletContext context = getServletContext();  
String password = req.getParameter("PASSWORD");  
String idString = req.getParameter("PATRON_ID");
```

- d. Так как метод **verifyLogon()** требует передачи идентификатора типа **int**, требуется выполнить преобразование. Добавьте в метод соответствующий код:

```
int id = 0;  
try {  
    id = Integer.valueOf(idString).intValue();  
} catch (NumberFormatException e) {}
```

- e. После того, как все необходимые данные получены можно вызывать метод для проведения проверки, записывать результат

в сессию и передавать управление другому сервлету:

```
try{
    Patron.verifyLogon(id, password);
    HttpSession session = req.getSession();

    Patron patron = Patron.findById(id);
    session.setAttribute("PATRON", patron);

    context.getRequestDispatcher("/ProcessListItems")
        .forward(req, resp);
}
```

- f. Метод **verifyLogon()** может выбрасывать 3 разных исключения: **PatronNotFoundException**, **SystemUnavailableException**, **InvalidPassword**. Для каждой из них нужно предусмотреть обработчик, который возвращал бы управление странице **Login.jsp**, при этом передавая туда специфическое сообщение об ошибке:

```
catch (PatronNotFoundException e) {
    req.setAttribute("MESSAGE",
        "Login failed: Patron " + idString + "
not registered");
    context.getRequestDispatcher("/WEB-
INF/jsp/Login.jsp")
        .forward(req, resp);
} catch (SystemUnavailableException e) {
    req.setAttribute("MESSAGE",
        "Login failed: There is a "
        + "system error. Please try to login
later.");
    context.getRequestDispatcher("/WEB-
INF/jsp/Login.jsp")
        .forward(req, resp);
} catch (InvalidPassword e) {
    req.setAttribute("MESSAGE", "Login failed:
Incorrect password");
    context.getRequestDispatcher("/WEB-
INF/jsp/Login.jsp")
```

```
        .forward(req, resp);  
    }  
}
```

g. После написание этого кода среда разработки сигнализирует о нескольких ошибках, связанных с невыполнеными импортами необходимых классов. Решите эти проблемы обычным способом: нажатие правой кнопкой мыши в редакторе, **Source -> Organize Imports.**

h. Код метода, получившийся в итоге должен быть похож на представленный ниже фрагмент.



```
46  
47  private void processRequest(HttpServletRequest req, HttpServletResponse resp)  
48      throws ServletException, IOException {  
49      ServletContext context = getServletContext();  
50      String password = req.getParameter("PASSWORD");  
51      String idString = req.getParameter("PATRON_ID");  
52      int id = 0;  
53      try {  
54          id = Integer.valueOf(idString).intValue();  
55      } catch (NumberFormatException e) {  
56      }  
57      try {  
58          Patron.verifyLogon(id, password);  
59          HttpSession session = req.getSession();  
60          Patron patron = Patron.findById(id);  
61          session.setAttribute("PATRON", patron);  
62          context.getRequestDispatcher("/ProcessListItems").forward(req, resp);  
63      } catch (PatronNotFoundException e) {  
64          req.setAttribute("MESSAGE", "Login failed: Patron " + idString + " not registered");  
65          context.getRequestDispatcher("/WEB-INF/jsp/Login.jsp").forward(req, resp);  
66      } catch (SystemUnavailableException e) {  
67          req.setAttribute("MESSAGE",  
68              "Login failed: There is a " + "system error. Please try to login later.");  
69          context.getRequestDispatcher("/WEB-INF/jsp/Login.jsp").forward(req, resp);  
70      } catch (InvalidPasswordException e) {  
71          req.setAttribute("MESSAGE", "Login failed: Incorrect password");  
72          context.getRequestDispatcher("/WEB-INF/jsp/Login.jsp").forward(req, resp);  
73      }  
74  }
```

i. Убедитесь, что в представлении **Problems** больше не осталось сообщений об ошибках и **сохраните** все сделанные изменения.

Раздел 4. Создание сервлета *ProcessListItems*

В этой части лабораторной работы создается сервлет **ProcessListItems** для отображения списка заказанных книг. В данном упражнении реализуется логика по проверке того факта, что пользователь прошел аутентификацию. В последующих упражнениях добавляется логика для извлечения списка книг из базы данных.

Логика сервлета начинается с проверки наличия валидной **HttpSession** для данного запроса. Сессия создается при успешной аутентификации. Также проверяется наличие в сессии объекта типа **Patron**. Если все в порядке, то управление передается странице **ListItems.jsp**, которая отображает список книг (в данном случае описанный с помощью

констант). Если сессия отсутствует или в ней нет объекта **Patron**, то управление передается сервлету **Login** для прохождения соответствующей процедуры.

1. В проекте **LibraryWebProject** создайте новый сервлет **ProcessListItems** в пакете **com.ibm.library.servlets**. Задайте для него URL **/ProcessListItems**. В методах **doGet()** и **doPost()** используйте вызов метода **processRequest()** для обработки запросов.
 - a. В представлении **Enterprise Explorer** нажмите правой кнопкой мыши по **LibraryWebProject -> Java Resources -> src -> com.ibm.library.servlets** и выберите **New -> Servlet**. Откроется мастер создания нового сервлета.
 - b. Введите **ProcessListItems** в поле **Class name**. Убедитесь, что в поле **Folder** выбрано **\LibraryWebProject\src**, а в поле **Java package** выбрано **com.ibm.library.servlets**. Нажмите **Next**.
 - c. На следующей странице убедитесь, что выбран URL **/ProcessListItems** и нажмите **Next**.
 - d. На следующей странице убедитесь, что выбраны опции **Public**, **doGet** и **doPost** и нажмите **Finish**.
2. Реализуйте логику работы сервлета, описанную выше.
 - a. Добавьте метод **processRequest** со следующей декларацией:

```
private void processRequest  
(HttpServletRequest req., HttpServletResponse resp)  
throws ServletException, IOException {}
```

- b. Реализуйте методы **doGet** и **doPost**, заменив их код на вызов метода **processRequest()**:

```
processRequest(request, response);
```

- c. В метод **processRequest()** добавьте следующие строчки:

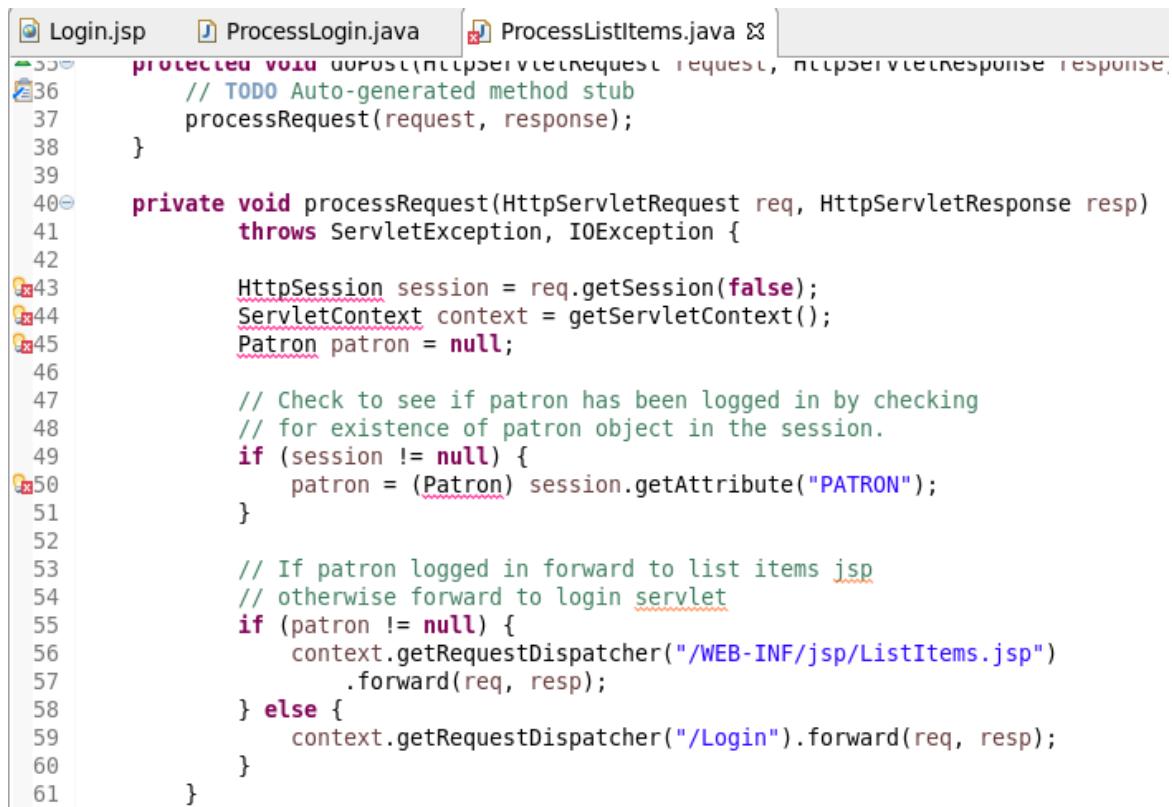
```
HttpSession session = req.getSession(false);  
ServletContext context = getServletContext();  
Patron patron = null;
```

```
if (session != null) {
```

```
    patron = (Patron) session.getAttribute("PATRON");  
}  
  
if (patron != null) {  
    context.getRequestDispatcher("/WEB-  
    INF/jsp/ListItems.jsp")  
        .forward(req, resp);  
} else {  
    context.getRequestDispatcher("/Login").forward(re  
q, resp);  
}
```

Метод **getSession(false)** возвращает **null**, если сессия не была создана до этого и не создает ее самостоятельно.

d. Итоговый код метода должен быть похож на показанный ниже:



```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
    processRequest(request, response);  
}  
  
private void processRequest(HttpServletRequest req, HttpServletResponse resp)  
throws ServletException, IOException {  
  
    HttpSession session = req.getSession(false);  
    ServletContext context = getServletContext();  
    Patron patron = null;  
  
    // Check to see if patron has been logged in by checking  
    // for existence of patron object in the session.  
    if (session != null) {  
        patron = (Patron) session.getAttribute("PATRON");  
    }  
  
    // If patron logged in forward to list items jsp  
    // otherwise forward to login servlet  
    if (patron != null) {  
        context.getRequestDispatcher("/WEB-INF/jsp>ListItems.jsp")  
            .forward(req, resp);  
    } else {  
        context.getRequestDispatcher("/Login").forward(req, resp);  
    }  
}
```

e. После написание этого кода среда разработки сигнализирует о нескольких ошибках, связанных с невыполненнымими импортами необходимых классов. Решите эти проблемы обычным способом: нажатие правой кнопкой мыши в редакторе, **Source -> Organize Imports**.

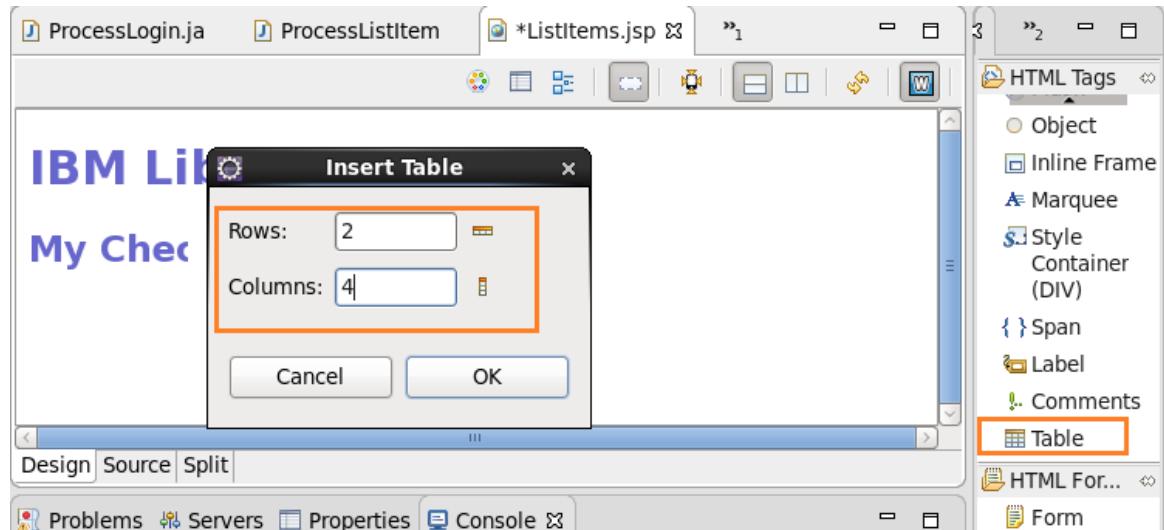
f. Сохраните сделанные изменения и убедитесь в отсутствии ошибок.

Раздел 5. Создание страницы *ListItems JSP*

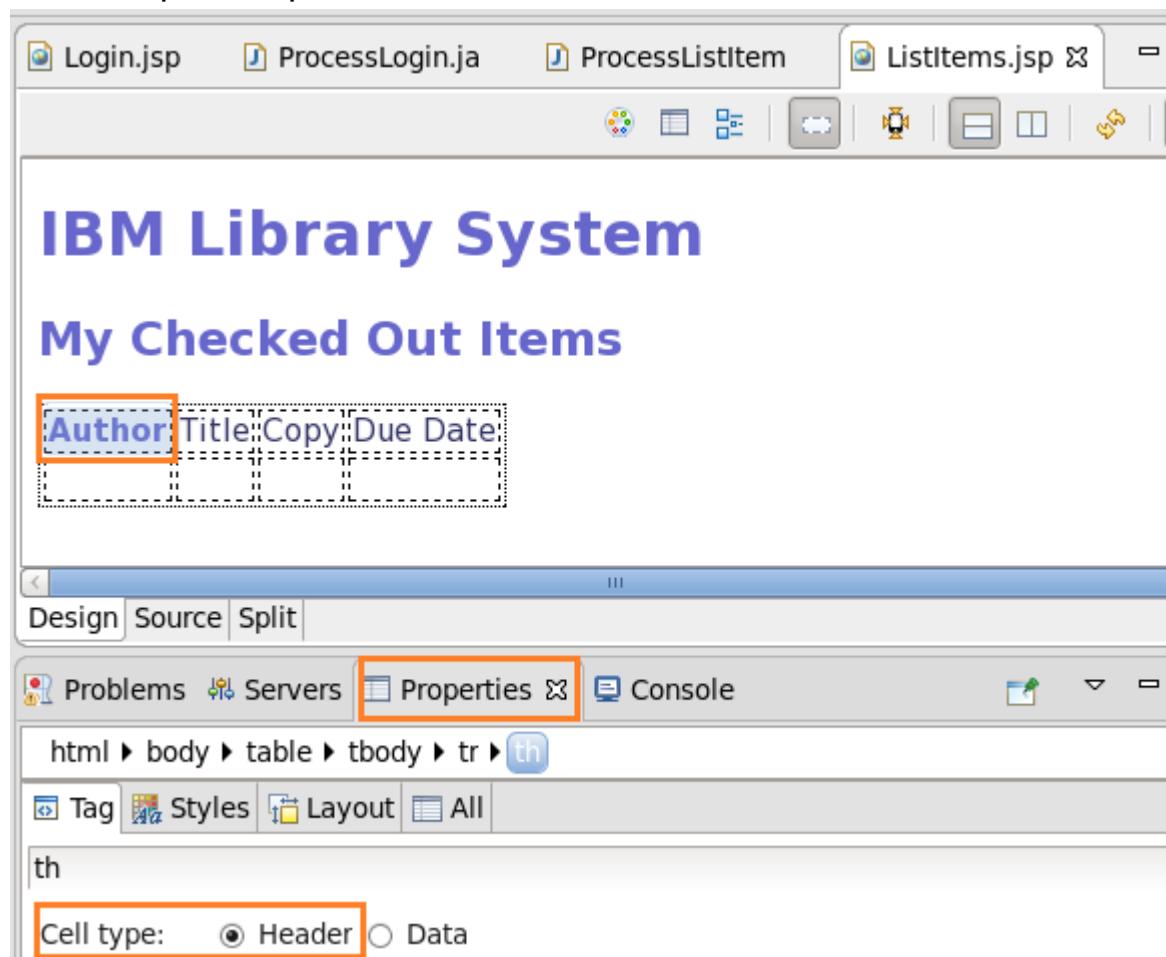
В этой части лабораторной работы создается JSP страница **ListItems.jsp**. На этой странице отображается таблица со списком заказанных книг.

1. Создайте страницу **Login.jsp** в проекте **LibraryWebProject**.
 - a. В представлении **Enterprise Explorer** нажмите правой кнопкой мыши по каталогу **LibraryWebProject -> WebContent -> WEB-INF -> jsp**. Выберите **New -> Web Page**.
 - b. Укажите имя **ListItems.jsp**. Нажмите кнопку **Options**.
 - c. На странице настроек страницы укажите **Document Type – HTML 4.01 Strict**.
 - d. На вкладке **Style Sheets** убедитесь, что выбран файл **/theme/Master.css**.
 - e. Нажмите **Close**. Нажмите **Finish**, завершая работу мастера по созданию новой страницы.
 - f. Создается JSP страница и открывается в редакторе. Перейдите на вкладку **Design**.
2. Сделайте на странице необходимые заголовки.
 - a. Выполните перенос элемента **Heading 1** из палитры (**Palette**) в редактор. Дважды кликните по нему левой кнопкой мыши так, чтобы открылся интерфейс (форма) для ввода текста заголовка. Введите текст **IBM Library System**.
 - b. По тому же сценарию разместите под первым заголовком второй типа **Heading 2** с текстом **My Checked Out Items**.
3. Создайте таблицу с четырьмя колонками: **Author, Title, Copy, Due Date**.
 - a. Из палитры в редактор вынесите элемент **Table**. Будет задан вопрос относительно количества столбцов и строк в этой таблице.

Укажите 2 строки и 4 столбца.



- b. В первой строке введите названия колонок: **Author**, **Title**, **Copy**, **Due Date**.
- c. Выберите **первую ячейку первой строки** и в представлении **Properties** выберите опция **Header**. Повторите процедуру для всех ячеек первой строчки.



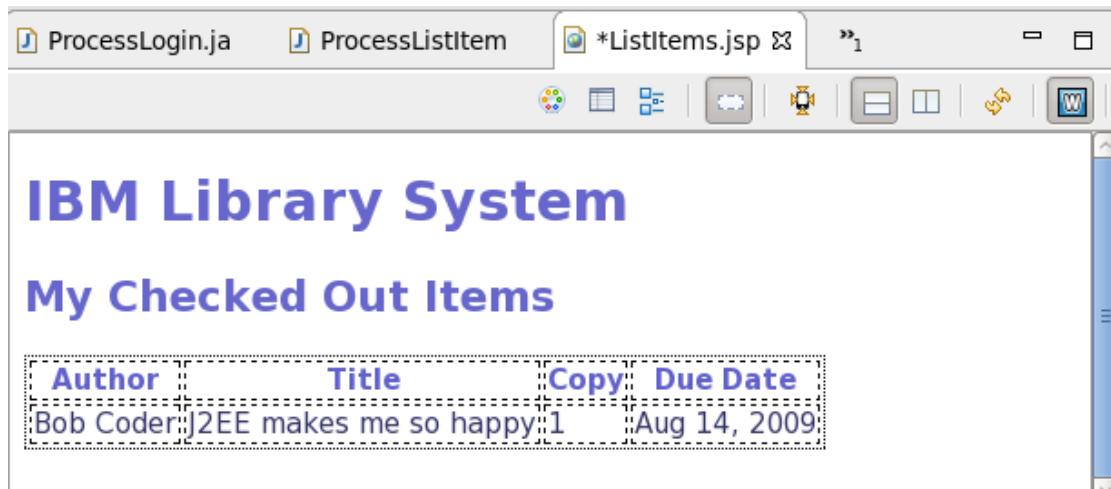
При появлении сообщения об ошибке отображения страницы в

редакторе можно закрыть страницу и открыть ее заново, ошибка пропадет, и ячейка таблицы будет показана корректно.

d. В ячейках второй строки введите тестовые данные:

Author: Bob Coder
Title: J2EE makes me so happy
Copy: 1
Due Date: Aug 14, 2009

e. По итогам страница должна выглядеть примерно так:



f. Сохраните сделанные изменения.

Раздел 6. Тестирование сервлетов и JSP

В этой части лабораторной работы вы проверяете логику работы системы: все разработанные сервлеты и JSP для системы аутентификации. Для доступа к системе используется URL <http://localhost:9080/Library/Login>. В качестве данных учетной записи можно использовать те учетные записи, которые были введены в одной из предыдущих лабораторных работ. Или же использовать одну из учетных записей, которые попали в базу данных сразу после ее создания:

ID	Password
1	brown
2	vogel
3	davis
4	straach

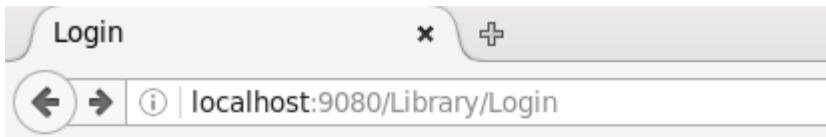
5	vanloon
6	weightman

1. Проверьте стандартный сценарий работы системы

a. В браузере перейдите по ссылке

http://localhost:9080/Library/Login.

b. Введите **id** и **пароль**. Нажмите кнопку **Login**.



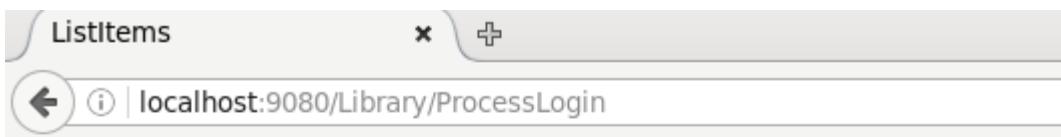
IBM Library System

Please Log in:

Patron ID

Password

c. Если данные были введены верно, то вы попадете на страницу со списком книг.



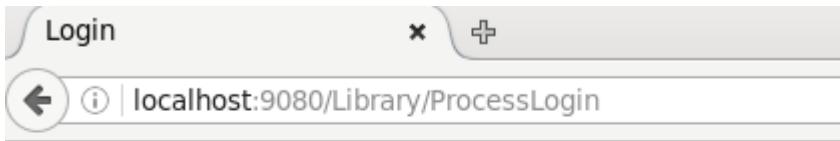
IBM Library System

My Checked Out Items

Author	Title	Copy	Due Date
Bob Coder	J2EE makes me so happy 1		Aug 14, 2009

d. Вернитесь на страницу **Login** и попробуйте ввести некорректные данные. Неправильный пароль, отсутствующую в системе учетную

запись.



IBM Library System

Please Log in:

Patron ID

Password

Login failed: Incorrect password

- е. В том случае, если некорректно отображаются заголовки на страницы (не работают CSS стили), попробуйте обновить ссылки на CSS в JSP страницах на следующие:
`<%=request.getContextPath()%>/theme/Master.css`

Раздел 7. Сохранение Patron ID в Cookie

В текущей реализации приложения предполагается, что пользователь вводит свой идентификатор каждый раз, когда ему понадобится воспользоваться системой. Что может быть чрезвычайно неудобно, если идентификаторы будут длинными и сложными. Можешь сократить количество вводимых данных пользователем с помощью следующей техники: пользователь вводит данные один раз, после чего для него генерируется Cookie, который сохраняется в браузере и подставляется в HTTP запрос в виде заголовка при последующих обращениях на сервер. По этому Cookie, содержащему идентификатор клиента можно будет распознать его и не требовать повторного ввода ID. Причем возможно это будет происходить даже после того, как пользователь закрыл браузер и завершил свою сессию.

Для реализации этой техники понадобится расширить функции серверов **Login**, **ProcessLogin** и страницы **Login.jsp**. Сервлеты смогут получить доступ к Cookie, извлекая его из объекта **HttpServletRequest**. Далее считанное значение будет передаваться как атрибут в JSP, где

соответствующее значение будет подставляться автоматически в поле идентификатора.

Cookie будет отправляться браузеру клиента путем создания соответствующего объекта со значением идентификатора пользователя и его добавления к объекту **HttpServletResponse**. Срок жизни для него можно определить с помощью метода **setMaxAge()**.

1. Добавьте новый метод в сервлет Login – **idCookie()**. В нем необходимо извлекать из входящего запроса Cookie с именем IDCOKIE. Это метод будет возвращать значение Cookie или пустую строку, если его нет в запросе.
 - a. В представлении **Enterprise Explorer** найдите и откройте файл **LibraryWebProject -> Java Resources -> src -> com.ibm.library.servlets -> Login.java**.
 - b. Добавьте в класс следующий метод:

```
private String idCookie(HttpServletRequest req) {  
    // Look for any cookies with id of IDCOKIE  
    String savedId = "";  
    Cookie[] cookies = req.getCookies();  
    if (cookies != null) {  
        for (int i = 0; i < cookies.length; i++) {  
            Cookie theCookie = cookies[i];  
            if  
(theCookie.getName() .equals("IDCOOKIE")) {  
                savedId = theCookie.getValue();  
                break;  
            }  
        }  
    }  
    return savedId;  
}
```

- c. После появления сообщений об ошибках разрешите их, импортировав класс **Cookie** из пакета **javax.servlet.http**.
d. Сохраните сделанные изменения и убедитесь в отсутствии ошибок в представлении **Problems**.
2. Обеспечьте передачу значение Cookie из сервлета в **Login.jsp**.

a. В методе **processRequest()** после первой строчки

`ServletContext = getServletContext();` добавьте
следующий код:

```
request.setAttribute("idcookie", idCookie(request));
```

b. Сохраните сделанные изменения.

3. Измените **Login.jsp** таким образом, чтобы в поле **Patron ID** по умолчанию отображалось бы значение переданного на страницу атрибута **idcookie**.

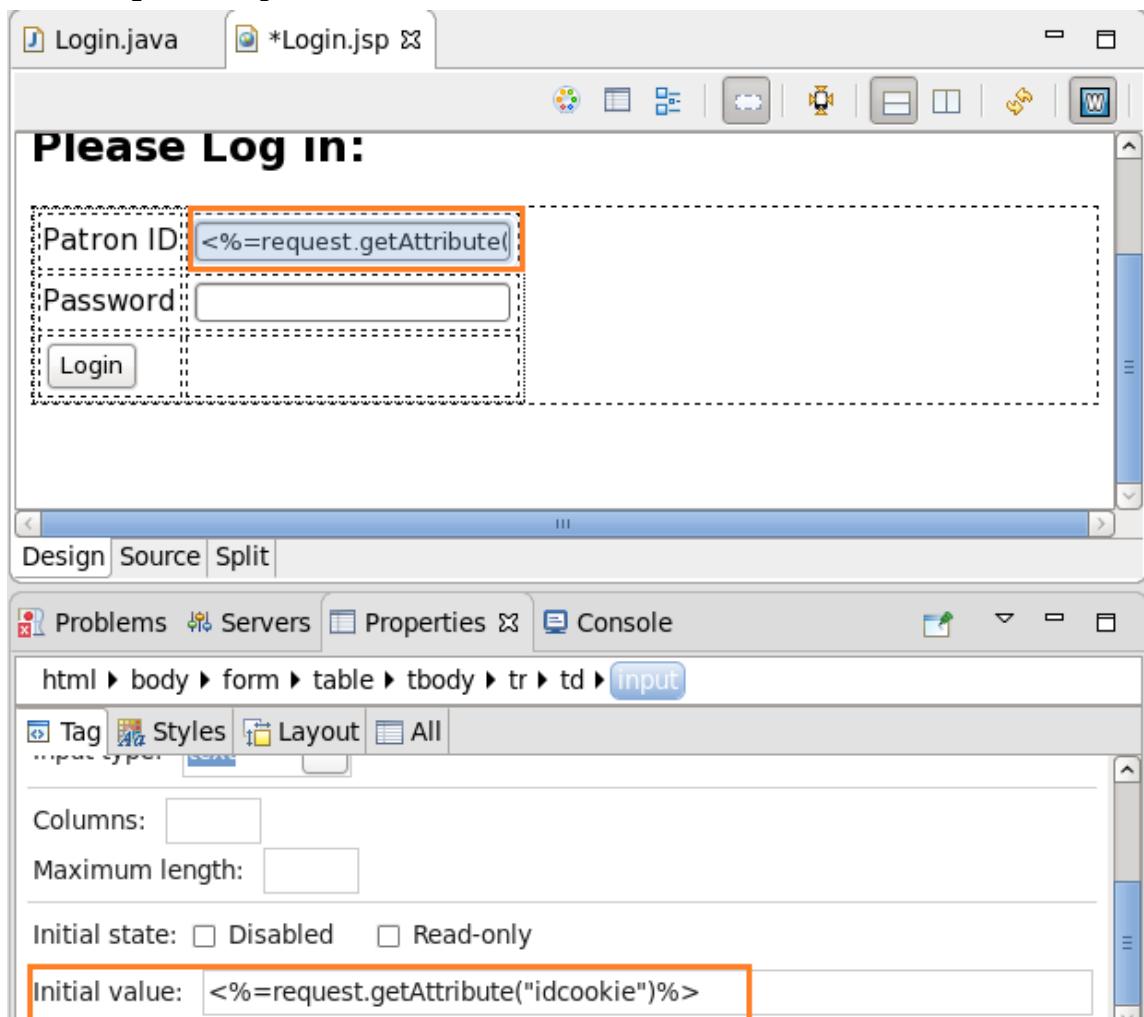
а. В представлении **Enterprise Explorer** найдите и откройте файл **LibraryWebProject -> WebContent -> WEB-INF -> jsp -> Login.jsp**.

Перейдите на вкладку **Design**.

б. Выберите поле для ввода **Patron ID**. Откройте представление **Properties**.

с. В поле **Initial value** введите

```
<%=request.getAttribute("idcookie")%>
```



d. Сохраните сделанные изменения.

4. Измените метод **processRequest()** сервлета **ProcessLogin** таким образом, чтобы сохранять cookie с идентификатором клиента в объекте **HttpServletResponse**. Эту логику можно включить в код метода после добавления в сессию объекта типа **Patron**. Cookie должен называться **IDCOOKIE** и иметь срок жизни в **1 неделю**. Также нужно удалять этот атрибут, в том случае, если срабатывает один из **catch** блоков.

- a. В представлении **Enterprise Explorer** найдите и откройте файл **LibraryWebProject -> Java Resources -> src -> com.ibm.library.servlets -> ProcessLogin.java**.
- b. В метод **processRequest()** добавьте следующий код для создания cookie. Разместите его после строчки

```
session.setAttribute("PATRON", patron);
```

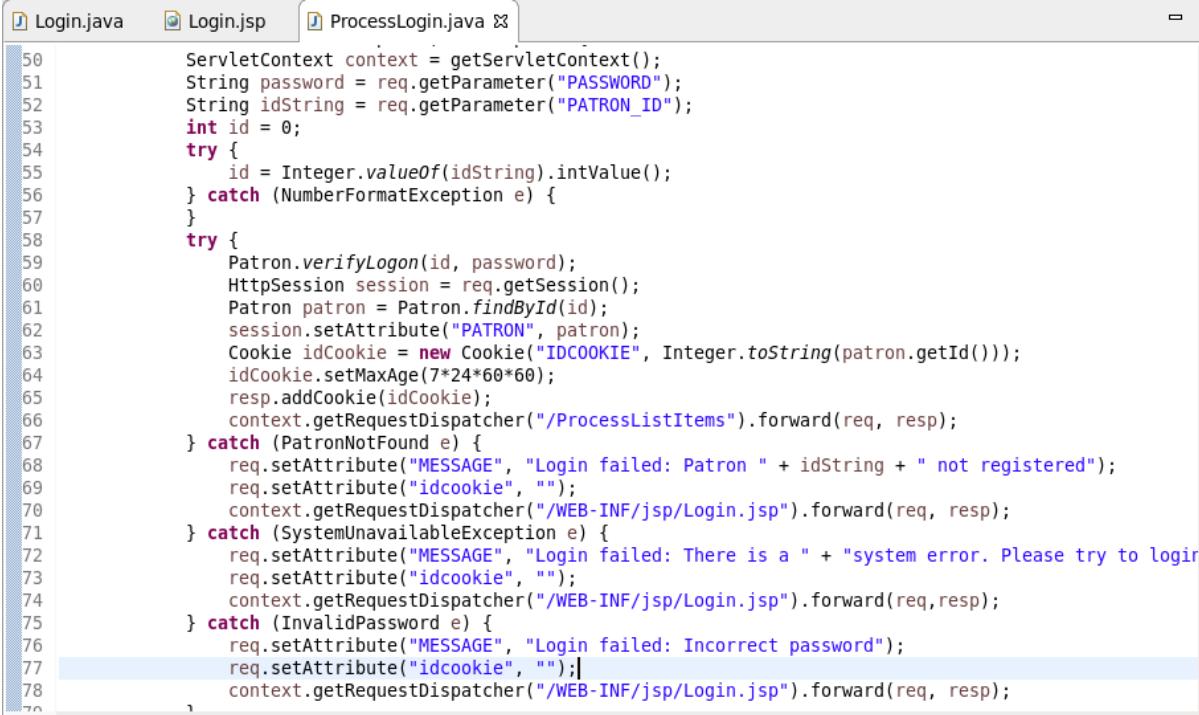
```
Cookie idCookie = new Cookie("IDCOOKIE",
Integer.toString(patron.getId()));
idCookie.setMaxAge(7*24*60*60);
resp.addCookie(idCookie);
```

- c. После появления сообщений об ошибках разрешите их, импортировав класс **Cookie** из пакета **javax.servlet.http**.
- d. В каждом из **3 catch** блоков добавьте следующий код для удаления значения cookie перед вызовом метода **getRequestDispatcher()**:

```
req.setAttribute("idcookie", "");
```

- e. Сохраните сделанные изменения и убедитесь в отсутствии ошибок в представлении **Problems**.

f. Убедитесь, что ваш код соответствует показанному ниже:



```
50     ServletContext context = getServletContext();
51     String password = req.getParameter("PASSWORD");
52     String idString = req.getParameter("PATRON_ID");
53     int id = 0;
54     try {
55         id = Integer.valueOf(idString).intValue();
56     } catch (NumberFormatException e) {
57     }
58     try {
59         Patron.verifyLogon(id, password);
60         HttpSession session = req.getSession();
61         Patron patron = Patron.findById(id);
62         session.setAttribute("PATRON", patron);
63         Cookie idCookie = new Cookie("IDCOOKIE", Integer.toString(patron.getId()));
64         idCookie.setMaxAge(7*24*60*60);
65         resp.addCookie(idCookie);
66         context.getRequestDispatcher("/ProcessListItems").forward(req, resp);
67     } catch (PatronNotFoundException e) {
68         req.setAttribute("MESSAGE", "Login failed: Patron " + idString + " not registered");
69         req.setAttribute("idcookie", "");
70         context.getRequestDispatcher("/WEB-INF/jsp/Login.jsp").forward(req, resp);
71     } catch (SystemUnavailableException e) {
72         req.setAttribute("MESSAGE", "Login failed: There is a " + "system error. Please try to login");
73         req.setAttribute("idcookie", "");
74         context.getRequestDispatcher("/WEB-INF/jsp/Login.jsp").forward(req, resp);
75     } catch (InvalidPassword e) {
76         req.setAttribute("MESSAGE", "Login failed: Incorrect password");
77         req.setAttribute("idcookie", "");
78         context.getRequestDispatcher("/WEB-INF/jsp/Login.jsp").forward(req, resp);
79     }
80 }
```

g. Протестируйте ваши изменения. Убедитесь в том, что поле **Patron ID** заполняется после успешного прохождения процедуры аутентификации. Проведите тест с закрытием браузера.

Конец упражнения

Упражнение 7. Создание JavaBean

О чём это упражнение:

В этой лабораторной работе вы создадите JavaBean для автоматизированной системы библиотеки, который будет хранить информацию, полученную из прикладной базы данных (список заказанных книг). Эти данные будут отображаться на соответствующей JSP. JavaBean работает как некий объект, который используется для передачи данных от одного компонента приложения к другому.

Будет создан JavaBean для передачи сообщений о текущем статусе или возникшей ошибке между различными компонентами системы библиотеки.

Для соответствующего объекта будут определены методы для установки значений некоторых параметров и методы для их считывания.

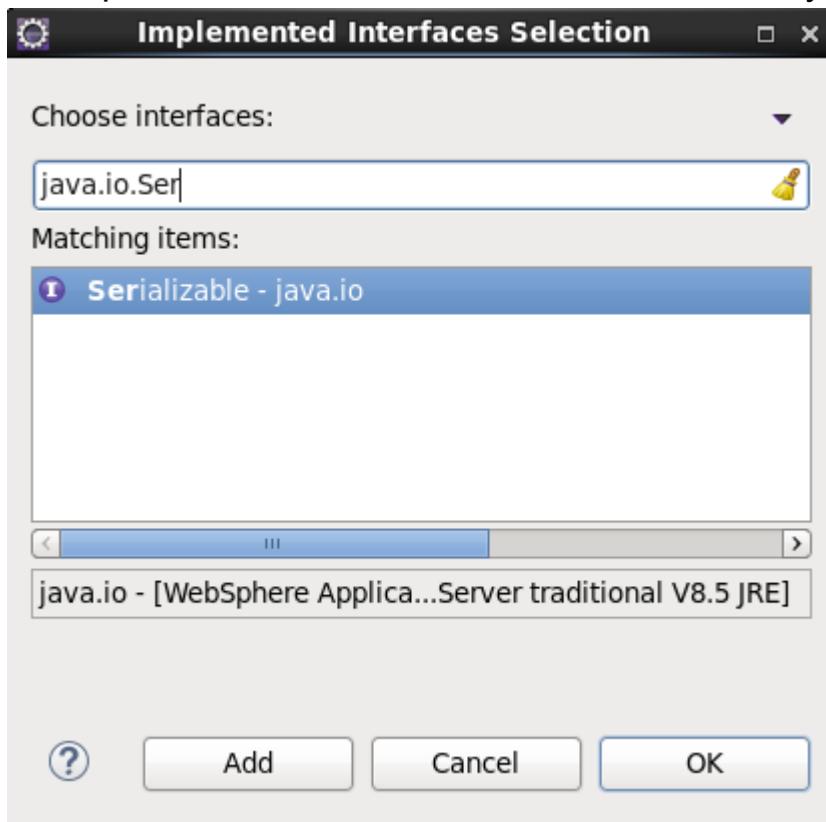
Что вы должны будете сделать:

- Создать JavaBean для хранения списка объектов
- Создать JavaBean для хранения текстового сообщения

Раздел 1. Создание *LoanedCopyListBean* JavaBean

В данном разделе создается JavaBean **LoanedCopyListBean**, который хранит информацию о списке книг, заказанных клиентом. Все элементы списка представляют собой объекты класса **LoanedCopy** из пакета **com.ibm.library.model**.

1. Создайте Java класс **LoanedCopyListBean** в проекте **LibraryWebProject**. Добавить этот класс стоит в пакет **com.ibm.library.servlets**. Так как речь идет о JavaBean, этот класс должен реализовать интерфейс **java.io.Serializable**.
 - a. В представлении **Enterprise Explorer** нажмите правой кнопкой мыши по **LibraryWebProject -> Java Resources -> src -> com.ibm.library.servlets**. Выберите **New -> Class**. Открывается мастер создания нового класса.
 - b. Укажите имя **LoanedCopyListBean**.
 - c. Нажмите кнопку **Add** рядом со списком интерфейсов.
 - d. Открывается список доступных интерфейсов. Он пустой до тех пор, пока вы не начнете вводить значение фильтра. Начните вводить **java.io.Ser**. В списке появится интерфейс Serializable. Выберите его и нажмите **Add**. Нажмите кнопку **OK**.



- e. Выберите опцию **Constructors from superclass** для того, чтобы получить в классе конструктор без параметров.

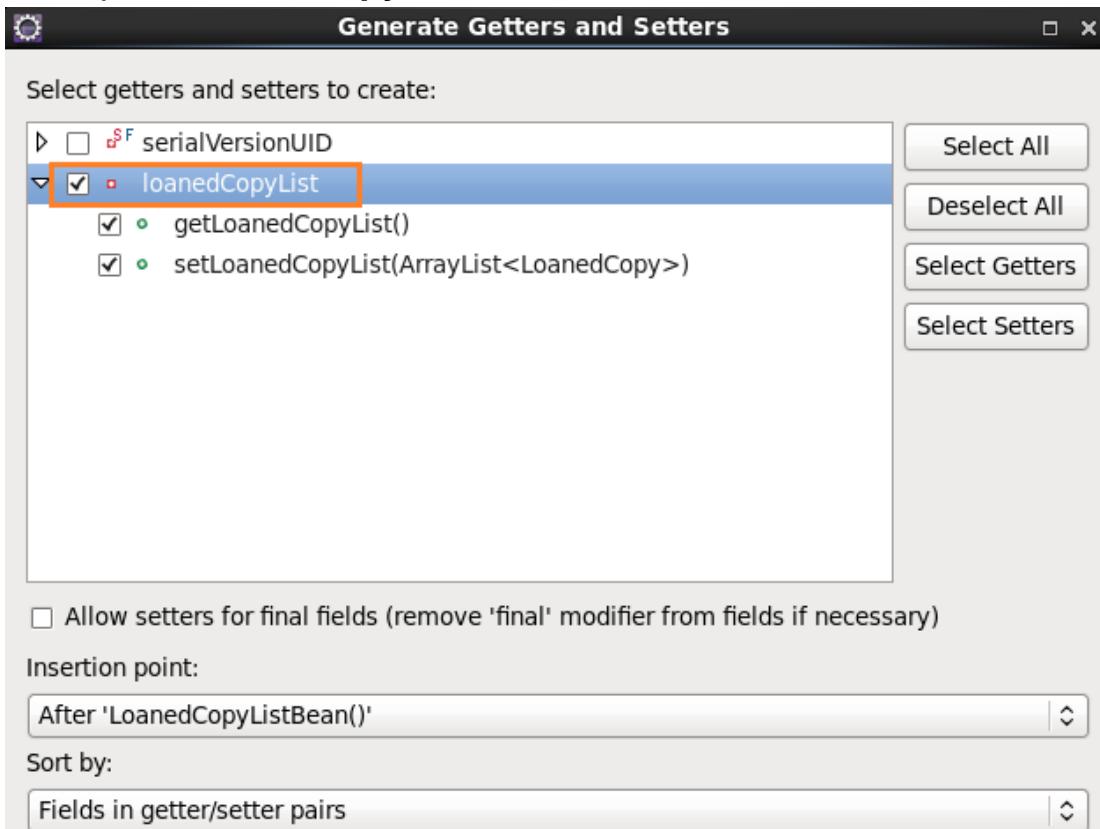


- f. Нажмите **Finish**. Код класса автоматически открывается в редакторе.
- g. Нажмите левой кнопкой мыши по лампочке рядом со строкой объявления класса и выберите **Add generated serial version ID**. При появлении всплывающего окна, нажмите **OK**.
2. Создайте приватный атрибут класса типа **ArrayList** объектов **LoanedCopy** с назначением **loanedCopyList**.
- a. Добавьте в класс следующий код сразу после его объявления:

```
private ArrayList<LoanedCopy> loanedCopyList;
```

- b. Разрешите возникшую проблему: нажмите правой кнопкой мыши в редакторе и выберите **Source -> Organize Imports**: в результате в данном классе будут импортированы класс **ArrayList** из пакета **java.util** и класс **LoanedCopy** из пакета **com.ibm.library.model**.

- c. Установите курсор мыши на строчке после объявления и реализации пустого конструктора. Нажмите правой кнопкой в редакторе, выберите **Source -> Generate Getters and Setters**. Открывается диалог по созданию соответствующих методов.
- d. Выберите **loanedCopyList** и нажмите **OK**.



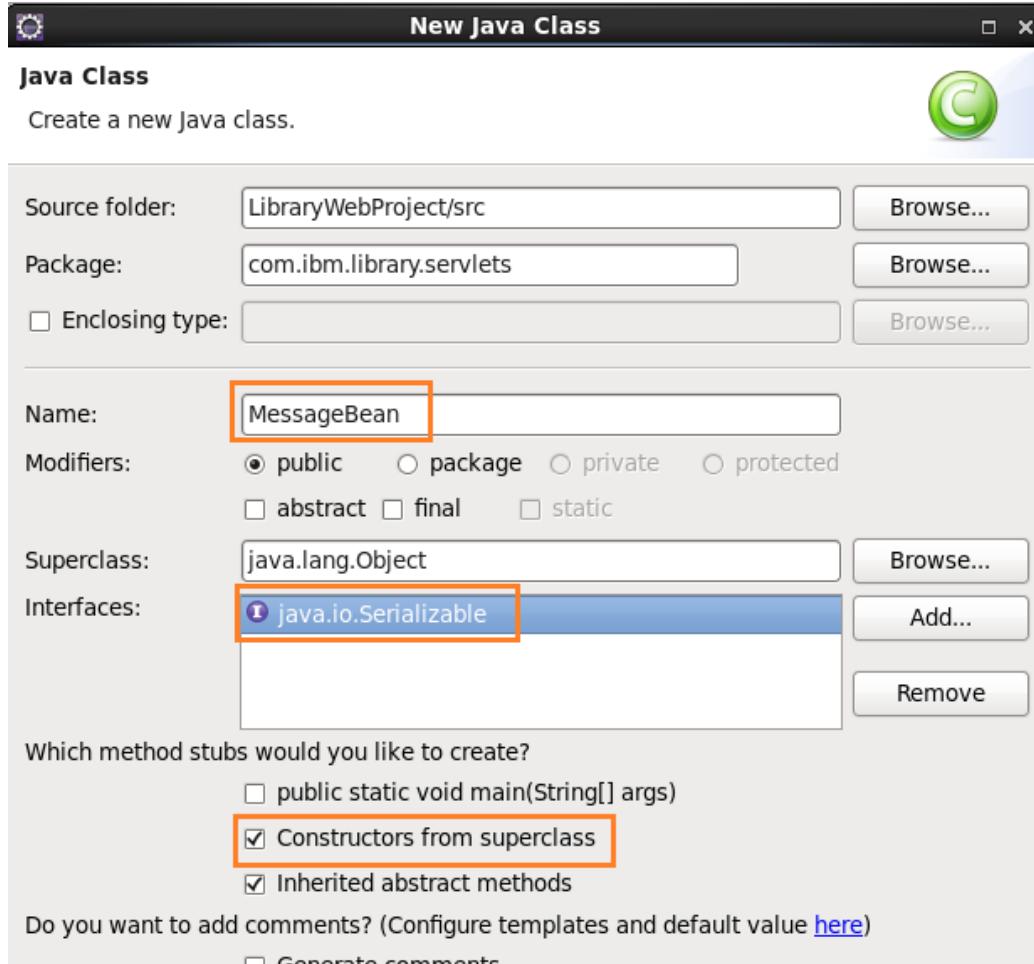
- e. Сохраните сделанные изменения.

Раздел 2. Создание **LoanedCopyListBean** JavaBean

В данном разделе создается JavaBean **MessageBean**, который хранит информацию о сообщении, которое может использоваться различными компонентами системы.

1. Создайте Java класс **MessageBean** в проекте **LibraryWebProject**. Добавить этот класс стоит в пакет **com.ibm.library.servlets**. Так как речь идет о JavaBean, этот класс должен реализовать интерфейс **java.io.Serializable**.
 - a. В представлении **Enterprise Explorer** нажмите правой кнопкой мыши по **LibraryWebProject -> Java Resources -> src -> com.ibm.library.servlets**. Выберите **New -> Class**. Открывается мастер создания нового класса.
 - b. Укажите имя **MessageBean**.
 - c. Нажмите кнопку **Add** рядом со списком интерфейсов.

- d. Открывается список доступных интерфейсов. Он пустой до тех пор, пока вы не начнете вводить значение фильтра. Начните вводить **java.io.Ser**. В списке появится интерфейс Serializable. Выберите его и нажмите **Add**. Нажмите кнопку **OK**.
- e. Выберите опцию **Constructors from superclass** для того, чтобы получить в классе конструктор без параметров.

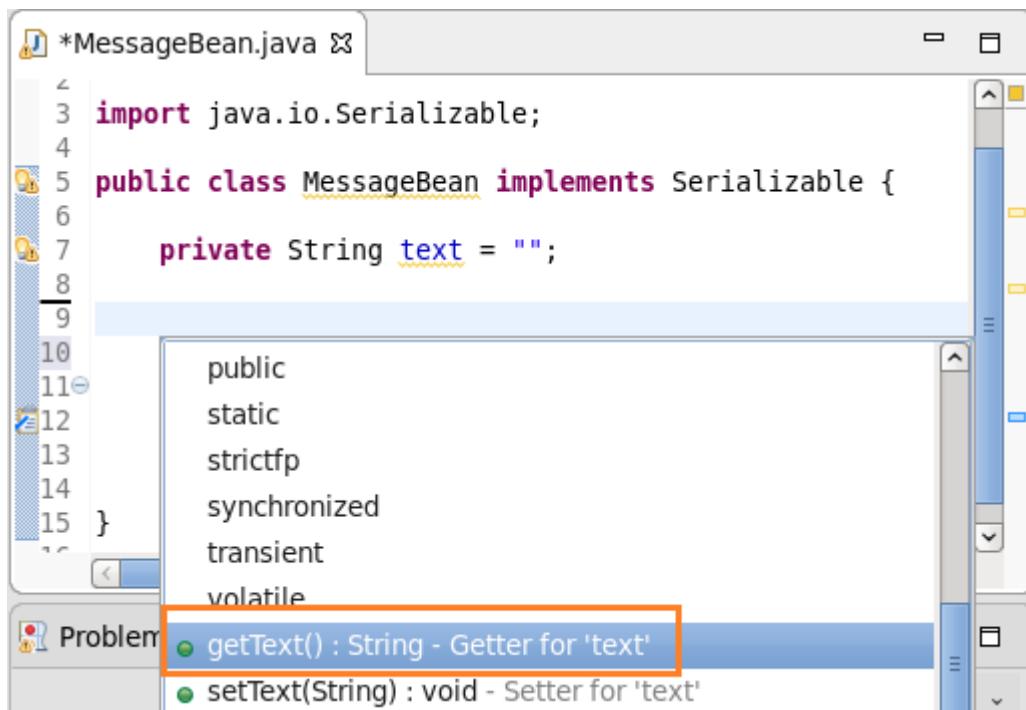


- f. Нажмите **Finish**. Код класса автоматически открывается в редакторе.
2. Добавьте в класс приватное поле **message** типа **String**. Сделайте методы для чтения (getter) и установки (setter) этого атрибута.
- a. Добавьте следующий код под объявлением класса:

```
private String text = "";
```

- b. Для создания метода getter воспользуйтесь меню **Content Assist**. Перейдите на пустую строку под кодом объявления атрибута **text**. Нажмите **Ctrl + Space**. Выберите **getText() String – Getter for**

`text`.



The screenshot shows an IDE interface with a code editor window titled "*MessageBean.java". The code defines a class MessageBean that implements Serializable. It contains a private String variable named text initialized to an empty string. A code completion dropdown menu is open at the end of the line, listing various Java modifiers: public, static, strictfp, synchronized, transient, and volatile. Below the modifiers, two method suggestions are shown: "getText() : String - Getter for 'text'" and "setText(String) : void - Setter for 'text'". The "setText(String) : void - Setter for 'text'" suggestion is highlighted with a red rectangle.

```
1 import java.io.Serializable;
2
3 public class MessageBean implements Serializable {
4
5     private String text = "";
6
7     public
8     static
9     strictfp
10    synchronized
11    transient
12    volatile
13
14    public String getText() {
15        return text;
16    }
17
18    public void setText(String text) {
19        this.text = text;
20    }
21
22 }
```

- c. Перейдите на следующую строчку. Нажмите **Ctrl + Space**.
Выберите **setText() void – Setter for `text`**.



The screenshot shows the completed code for MessageBean.java. The class now includes a public void setText(String text) method that sets the value of the private text variable. The code editor highlights the closing brace of the setText method.

```
1 package com.ibm.library.servlets;
2
3 import java.io.Serializable;
4
5 public class MessageBean implements Serializable {
6
7     private String text = "";
8
9     public String getText() {
10         return text;
11     }
12
13     public void setText(String text) {
14         this.text = text;
15     }
16
17     public MessageBean() {
18         // TODO Auto-generated constructor stub
19     }
20
21 }
22 }
```

- d. Сохраните сделанные изменения.

Конец упражнения

Упражнение 8. Совместное использование сервлетов, страниц JSP и JavaBeans

О чём это упражнение:

В этой лабораторной работе вы настроите совместную работу серверов, JSP и JavaBeans в автоматизированной системе библиотеки для того, что реализовать архитектуру приложения в соответствии с принципами паттерна Model-View-Controller. В процессе этой интеграции будет сделано следующее:

- Список взятых книг будет подгружаться из базы данных. При этом будет использоваться JavaBean, созданный в предыдущем упражнении.
- JSP со списком книг будет отображать данные на основе заполненного JavaBean с помощью тегов `<useBean>` и `<getProperty>`.
- Для отображения информации об ошибках будет использоваться специальная JSP страница.
- На страницах для регистрации будут использованы теги `<useBean>` и `<getProperty>`.

Что вы должны будете сделать:

- Связать все компоненты системы в соответствии с шаблоном Model-View-Controller (MVC)
- Заполнять JavaBeans на основе данных из модельного слоя системы (классы из пакета model, использующиеся для работы с данными)
- Использовать теги `<useBean>` и `<getProperty>` для извлечения данных при необходимости отобразить их в соответствующих (view) компонентах
- Настроить систему обработки ошибок и их визуализации для пользователей

Раздел 1. Обновление сервера ProcessListItems

В этом разделе будет изменен метод **processRequest()** в сервере **ProcessListItems**. Теперь в этом методе список заказанных книг будет загружаться из базы данных для того клиента, который вошел в систему. Для получения этого списка будет использоваться метод **retrieveLoanedCopies()** класса **Patron**. Этот список будет сохраняться в JavaBean **LoanedCopyItemListBean**, который был создан в предыдущем упражнении. Этот JavaBean будет доступен для view компонентов после сохранения его в сессию.

1. Измените метод **processRequest()** в сервере **ProcessListItems** для реализации логики, описанной выше. Если в списке есть хотя бы одна книга, то нужно заполнять **LoanedCopyListBean**, сохранять этот объект в сессии под именем **listitems** и передавать управление странице **ListItems.jsp**. Если в списке нет книг, нужно передавать управление странице **NoListItems.jsp**. В случае возникновения исключения **SystemUnavailableException** необходимо возвращать клиенту **500** ошибку с помощью метода **sendError()**.
 - a. Откройте файл **LibraryWebProject -> Java Resources -> src -> com.ibm.library.servlets -> ProcessListItems.java**.
 - b. Найдите метод **processRequest()**.
 - c. Замените текущую реализацию метода на ту, которая показана ниже:

```
private void processRequest(
    HttpServletRequest req,
    HttpServletResponse resp)
throws ServletException, IOException {

    HttpSession session = req.getSession(false);
    ServletContext context = getServletContext();
    Patron patron = null;

    if (session != null) {
        patron = (Patron) session.getAttribute("PATRON");
        if (patron == null) {

            context.getRequestDispatcher("/Login").forward(re
q, resp);
        }
    }
}
```

```
        }

    }

try {
    ArrayList<LoanedCopy> loanList =
    (ArrayList<LoanedCopy>)
    patron.retrieveLoanedCopies();
    if (loanList.size() > 0) {
        LoanedCopyListBean listBean = new
        LoanedCopyListBean();
        listBean.setLoanedCopyList(loanList);
        session.setAttribute("listitems", listBean);
        context.getRequestDispatcher(
            "/WEB-
INF/jsp/ListItem.jsp").forward(req, resp);
    } else {
        context.getRequestDispatcher(
            "/WEB-
INF/jsp/NoListItems.jsp").forward(req, resp);
    }
} catch (SystemUnavailableException e) {
    resp.sendError(HttpServletRequest.SC_INTERNAL_SE
RVER_ERROR,
    "System unavailable exception in ProcessListItems
function");
}
}
```

- d. Устраните возникшие ошибки, импортировав необходимые классы.
- e. Используйте меню Quick Fix для того, чтобы устранить проблему с вызовом метода **retrieveLoanedCopies()**. Выберите вариант **Add @SuppressWarnings ‘unchecked’ to processRequest() method**. Это добавит в код соответствующую аннотацию, которая заставит

среду разработки не показывать такого рода предупреждение.

```

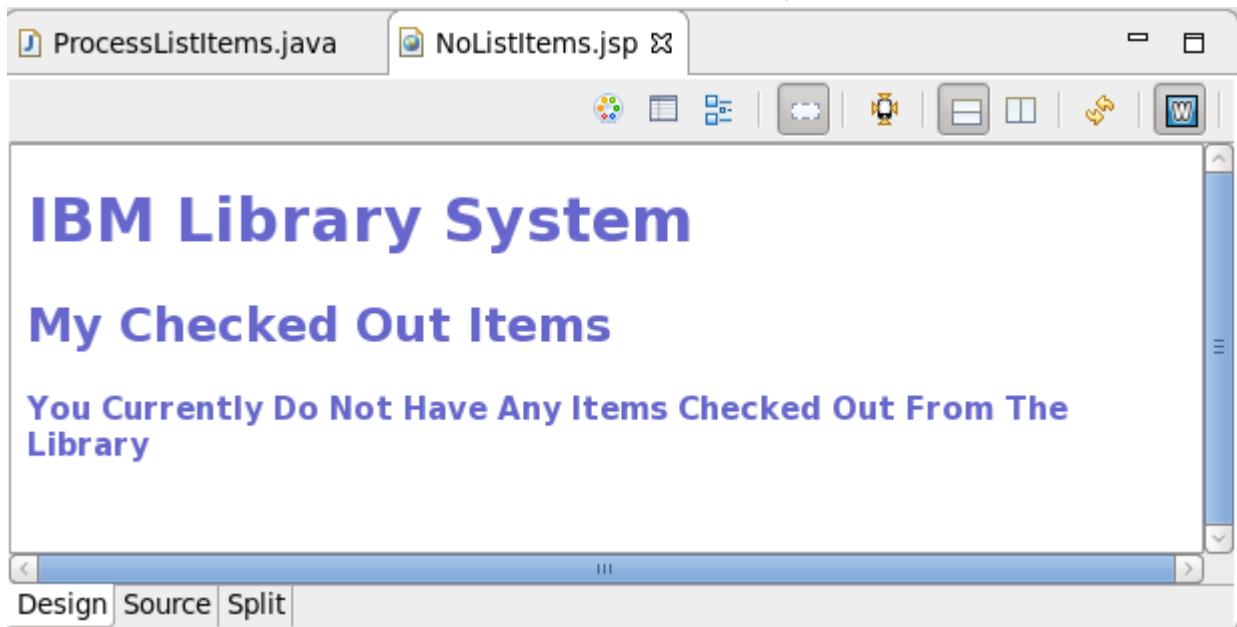
56 Patron patron = null;
57
58     // Check to see if patron has been logged in by checking
59     // for existence of patron object in the session.
60     if (session != null) {
61         patron = (Patron) session.getAttribute("PATRON");
62         if (patron == null) {
63             context.getRequestDispatcher("/Login").forward(req, resp);
64         }
65     }
66
67     try {
68
69         // Get list of items the patron has checked out,
70         // put the list in the loaned copy list bean.
71         // Forward to item list jsp (unless there were no items)
72         ArrayList<LoanedCopy> loanList = (ArrayList<LoanedCopy>) patron.retrieveLoanedCopies();
73 ...
74     @SuppressWarnings("unchecked")
75     private void processRequest(
76         HttpServletRequest req,
77         ...
78     )
79
80
81

```

Extract to local variable (replace all occurrences)
 Extract to local variable
 Extract to method (Ctrl+2 M)
 Add @SuppressWarnings 'unchecked' to 'loanList'
@ Add @SuppressWarnings 'unchecked' to 'processRequest()'
 Add parentheses around cast
 Configure problem severity

- f. Метод **sendError()**, который используется в случае возникновения исключения **SystemUnavailableException**, принимает два аргумента: HTTP код возврата, который вернется клиенту (в данном случае используется **500**, скрывающийся за константой **HttpServletResponse.SC_INTERNAL_SERVER_ERROR**), и сообщение, которое отобразится у него на странице.
- g. **Сохраните** сделанные изменения.
2. Создайте новую JSP страницу **NoListItems.jsp** в каталоге **WEB-INF/jsp**. В ней не будет динамического контента, только статическая информация о том, что у данного клиента нет взятых книг.
 - a. В представлении **Enterprise Explorer** нажмите правой кнопкой мыши по каталогу **LibraryWebProject -> WebContent -> WEB-INF -> jsp**. Выберите **New -> Web Page**. Открывается мастер создания новой страницы.
 - b. Укажите **NoListItems.jsp** в поле **File Name**.
 - c. Нажмите кнопку **Options**.
 - d. Выберите **HTML 4.01 Strict** в поле **Document Type**.
 - e. Нажмите **Close**.
 - f. Нажмите **Finish**, завершая работу мастера по созданию новой страницы. Страница сразу открывается в редакторе.
Переключитесь на вкладку **Design**.
3. Измените страницу, добавив на нее необходимое статическое сообщение.

- a. Перенесите из палитры в редактор последовательно 3 элемента, расположив их вертикально: **Heading 1, Heading 2, Heading 4**.
- b. Заполните заголовок первого уровня текстом **IBM Library System**. Для ввода текста этого заголовка нужно дважды кликнуть по соответствующему элементу и дождаться открытия формы ввода.
- c. Заполните заголовок второго уровня текстом **My Checked Out Items**.
- d. Заполните заголовок четвертого уровня текстом **You currently do not have any items checked out from the library**.
- e. По итогам страница должна выглядеть следующим образом:



- f. Переключитесь на вкладку **Source**.
- g. Измените ссылку на CSS подобно тому, как это было сделано в лабораторной работе 6. Измените часть пути `..J..` на `<%=request.getContextPath()%>`
- h. Сохраните сделанные изменения.

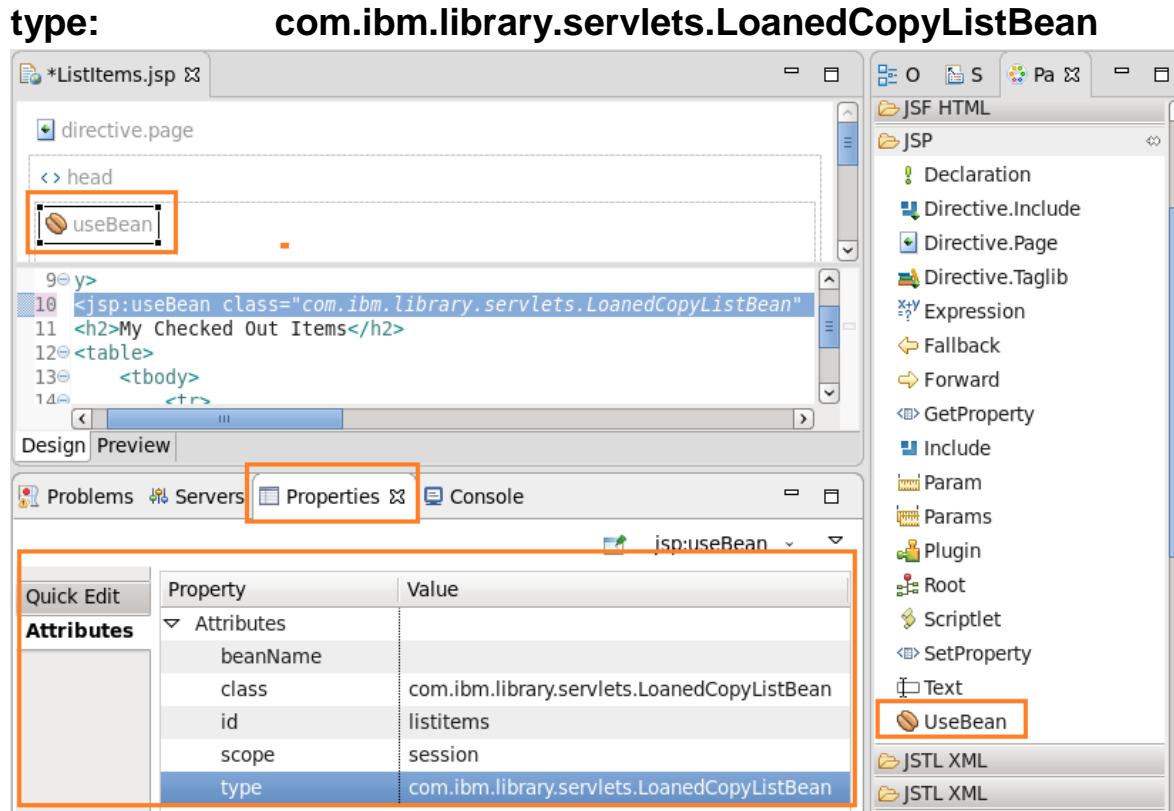
Раздел 2. Обновление JSP ListItems

В предыдущих лабораторных работах для работы **ListItems.jsp** были указаны константы, которые использовались при каждом попадании на эту страницу. Теперь же здесь будут использоваться реальные данные, передаваемые на страницу с помощью JavaBean. Их использование будет осуществляться с помощью специальных тегов `<useBean>` и `<getProperty>`. Такой подход является канонических для шаблона MVC.

JavaBean сохраняется в сессии как атрибут с именем **listitems**. В JSP требуется считать его из сессии и использовать при построении HTML таблицы.

1. Добавьте на страницу ссылку на JavaBean.
 - a. Откройте файл **LibraryWebProject -> WebContent -> WEB-INF -> jsp -> ListItems.jsp**.
 - b. **Удалите** ссылку на css файл. Это нужно для того, чтобы открыть этот файл в специальном редакторе, который позволяет в визуальном режиме добавлять элементы в JSP.
 - c. **Сохраните** сделанные изменения.
 - d. Выберите файл **LibraryWebProject -> WebContent -> WEB-INF -> jsp -> ListItems.jsp**, нажмите по нему правой кнопкой мыши, выберите **Open With -> Web Page Editor**.
 - e. Переключитесь на вкладку **Design**.
 - f. Найдите в палитре (**Palette**), в секции **JSP**, элемент **UseBean** и перенесите его в редактор в область в начале страницы.
 - g. Выберите добавленный элемент на странице, перейдите в представление **Properties**, выберите вкладку **Attributes**.
 - h. Укажите следующие параметры:

class:	com.ibm.library.servlets.LoanedCopyListBean
id:	listitems
scope:	session



2. Реализуйте логику для динамической генерации таблицы взятых книг.

- Переключитесь на вкладку с обычным редактором для работы с JSP (тот, где присутствует вкладка **Source**). Если эта вкладка уже закрыта, в представлении **Enterprise Explorer** выберите файл **LibraryWebProject -> WebContent -> WEB-INF -> jsp -> ListItems.jsp**, нажмите по нему правой кнопкой мыши, выберите **Open With -> Rich Page Editor**.
- Переключитесь на вкладку **Source**.
- Удалите вторую строку в таблице, где указана статическая информация по книжке, которая была описана в одной из предыдущих лабораторных работ – все элементы, включая открывающий и закрывающий теги **<tr>**. Первую строку с заголовками удалять не нужно.
- Вместо удаленной строчки добавьте следующий код:

```

<%
for (LoanedCopy i : listitems.getLoanedCopyList()) {
    pageContext.setAttribute("item", i);
%>

```

```
<jsp:useBean id="item"
  class="com.ibm.library.model.LoanedCopy"
  scope="page"></jsp:useBean>
<tr>
  <td align="center"><jsp:getProperty
  name="item" property="author" /></td>
  <td align="center"><jsp:getProperty
  name="item" property="title" /></td>
  <td align="center"><jsp:getProperty
  name="item" property="copyNumber" /></td>
  <td align="center"><jsp:getProperty
  name="item" property="due" /></td>
</tr>
<% } %>
```

В данном фрагменте запускается цикл по всем элементам из JavaBean со списком книг. Каждый элемент из этого списка записывается в переменную **item**. Далее для каждого такого элемента в таблицу добавляется строчка со всеми атрибутами. Обратите внимание на теги **<jsp:useBean>**, с помощью которого объявляется JavaBean, и тег **<jsp:getProperty>** с помощью которогочитываются параметры этого JavaBean.

- e. Добавьте в директиву **<%@page %>** атрибут **import="com.ibm.library.model.LoanedCopy "**.
- f. Верните ссылку на css файл. Внутри тега **<head>** добавьте следующую строку:

```
<link rel="stylesheet"
  href="<%=request.getContextPath() %>/theme/Master.css
  " type="text/css">
```

- g. Сохраните сделанные изменения.

Раздел 3. Создание Error.jsp

В первом разделе этой лабораторной работы в сервлете был добавлен вызов метода и при возникновении исключительной ситуации. Это метод возвращал клиенту код **500** и специальное сообщение об ошибке. В данном разделе создается специальная JSP страница, которая будет вызываться каждый раз при возникновении ошибки для донесения

соответствующего сообщения клиенту системы. Для того, чтобы использовать эту JSP страницу при возникновении ошибки потребуется вносить некоторые изменения в дескриптор развертывания приложения.

1. Создайте новую страницу Error.jsp.

a. В представлении **Enterprise Explorer** нажмите правой кнопкой мыши по каталогу **LibraryWebProject -> WebContent -> WEB-INF -> jsp**. Выберите **New -> Web Page**. Открывается мастер создания новой страницы.

b. Укажите **Error.jsp** в поле **File Name**.

c. Нажмите **Finish**, завершая работу мастера по созданию новой страницы. Страница сразу открывается в редакторе.

Переключитесь на вкладку **Design**.

2. Добавьте на страницу статический контент.

a. Перенесите из палитры в редактор последовательно 2 элемента, расположив их вертикально: **Heading 1, Heading 2**.

b. Заполните заголовок первого уровня текстом **IBM Library System**. Для ввода текста этого заголовка нужно дважды кликнуть по соответствующему элементу и дождаться открытия формы ввода.

c. Заполните заголовок второго уровня текстом **An error has been detected**.

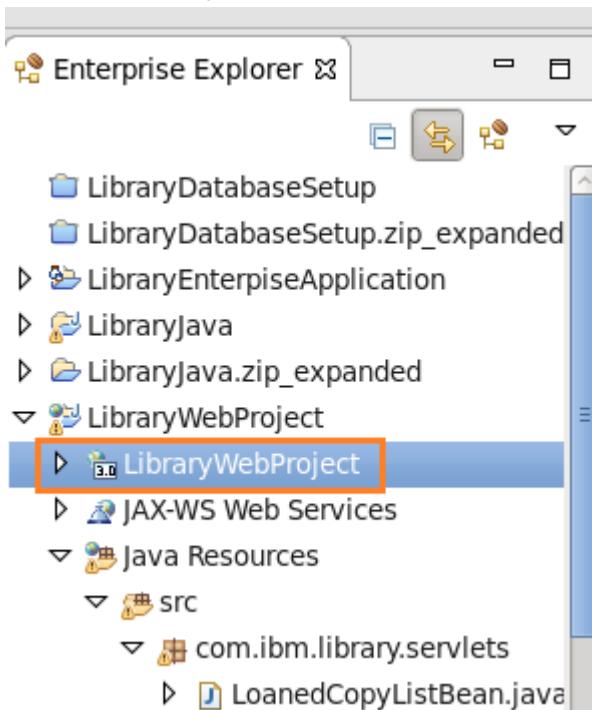
3. Добавьте на страницу код для извлечения кода и сообщения об ошибке, а также их отображения на странице.

a. Переключитесь на вкладку **Source**.

b. Добавьте в нижнюю часть страницы (до закрывающего тега **</BODY>**) следующий код:

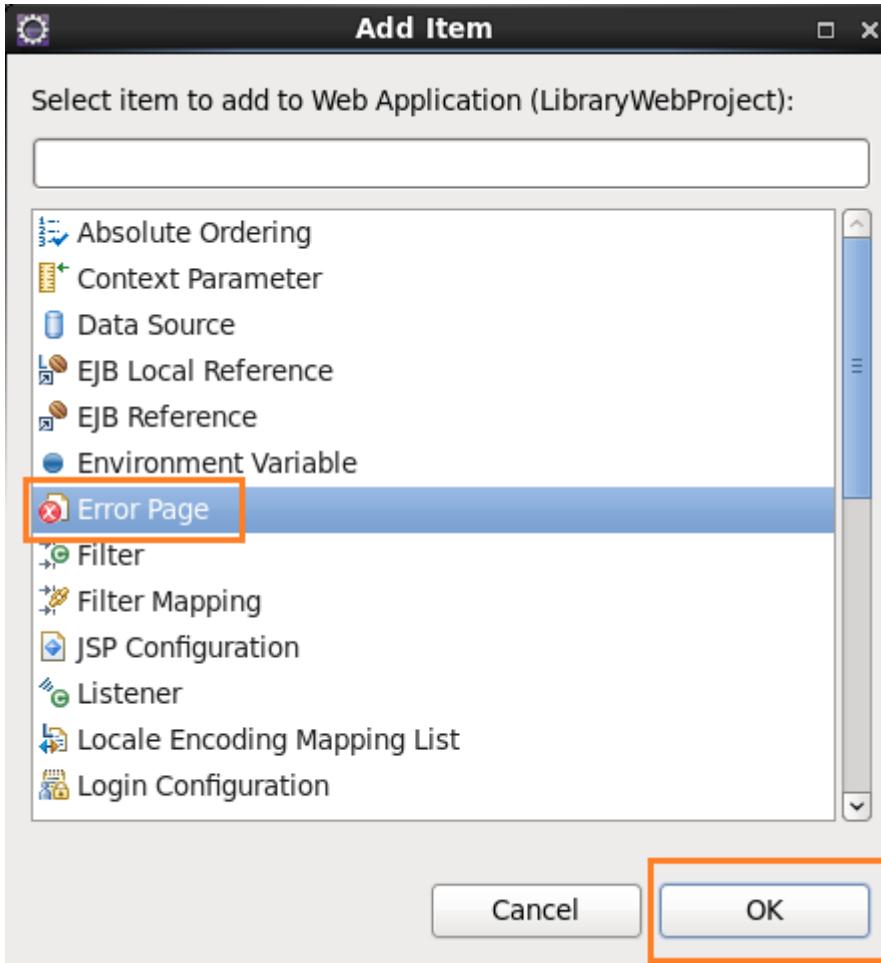
```
<%  
Integer code =  
(Integer) request.getAttribute("javax.servlet.error.s  
tatus_code");  
String message =  
(String) request.getAttribute("javax.servlet.error.me  
ssage");  
%>  
<p>Error code: <%=code%></p>  
<p>Error message: <%=message%></p>
```

- с. В ссылке на css файл измените фрагмент пути `..../..` на
`<%=request.getContextPath()%>`
4. Измените дескриптор развертывания таким образом, чтобы использовать созданную страницу в качестве обработчика для 500 ошибок.
- а. В представлении **Enterprise Explorer** дважды кликните по дескриптору развертывания веб приложения:

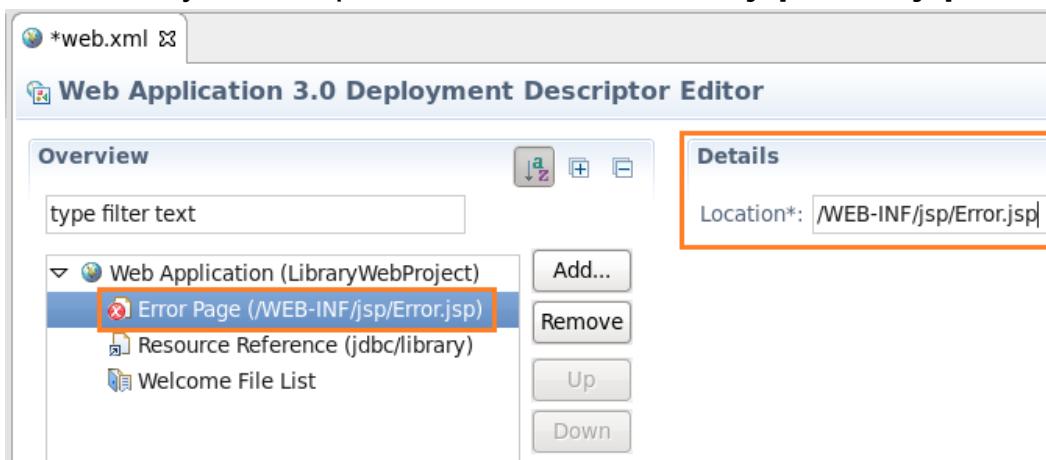


- б. Открывается редактор дескриптора развертывания. Нажмите на кнопку **Add...**

с. Выберите Error Page. Нажмите OK.



д. Укажите путь до страницы JSP: /WEB-INF/jsp/Error.jsp.



е. Сохраните сделанные изменения.

Раздел 4. Тестирование системы

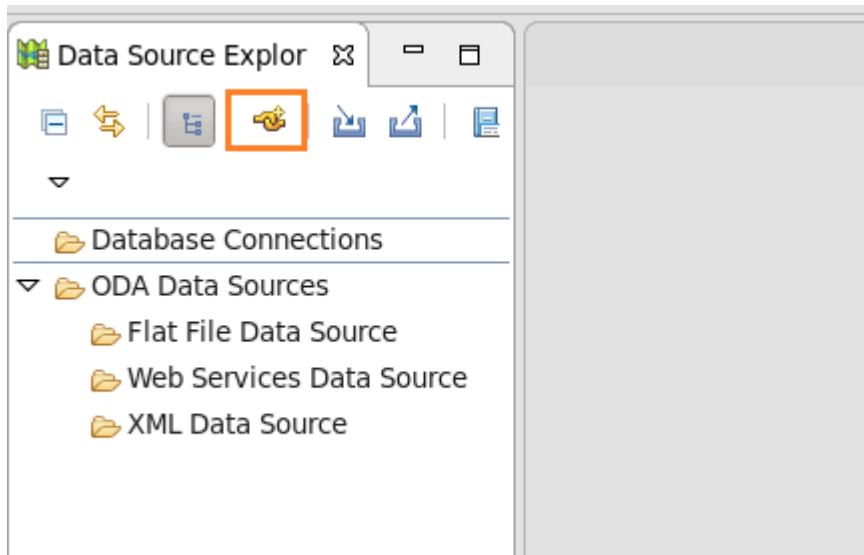
В этом разделе тестируются все новые функции, которые были разработаны в этой лабораторной работе до этого. При отображении списка заказанных книг можно увидеть следующие данные (дата возврата книги указана от даты создания базы/генерации тестовых

данных):

ID	Password	Book Title	Copy	Due
1	brown	J2EE 1.4 makes me so happy	2	+4
2	vogel	J2EE 1.4 makes me so happy	3	+2
		Java for coffee drinkers	2	0
		Enterprise JavaBeans	2	-1
3	davis	None	NA	NA
4	straach	Enterprise JavaBeans	3	+1
5	vanloon	Webs Fear: A guide for arachnophobics	2	+3
6	weightman	None	NA	NA

1. Проведите регрессионное тестирования для того, чтобы убедиться, что вход в систему работает.
2. Убедитесь, что для пользователей отображается корректный список выданных книг.
3. Проверьте ситуацию, когда в систему входит пользователь, у которого нет взятых книг. Должна открываться другая страница с соответствующим сообщением.
4. Проверьте работу **Error.jsp**. Для моделирования ошибки установите блокировку на базу данных в среде Eclipse, перспективе **Database Development**.
 - а. Остановите сервер приложений через представление **Servers**.
 - б. Откройте перспективу **Database Development**. Главное меню: **Window -> Perspective -> Open Perspective -> Other -> Database Development**.

c. В представлении **Data Source Explorer** нажмите по пиктограмме **New Connection Profile**.

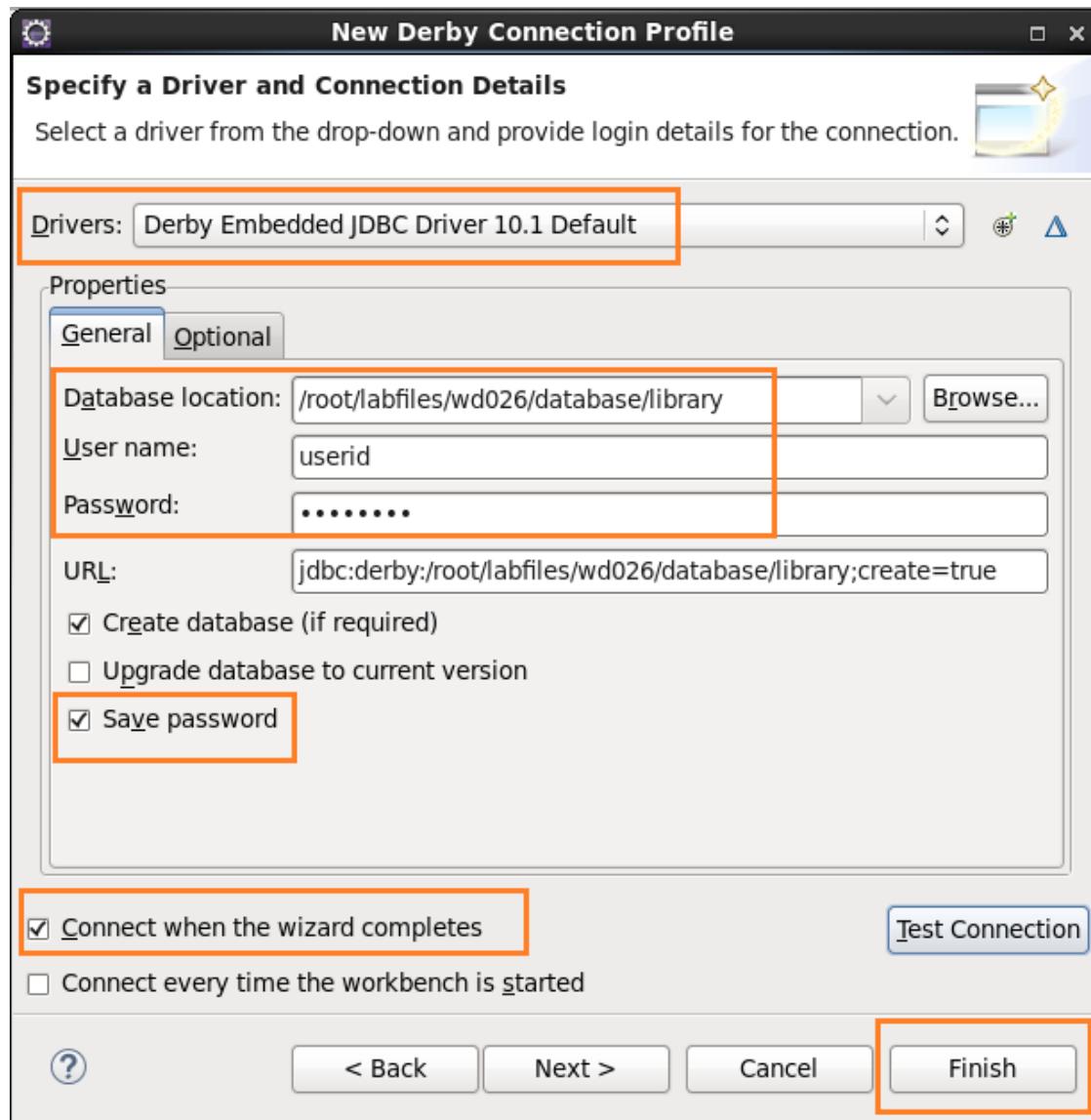


d. Выберите тип СУБД **Derby**. Нажмите **Next**.

e. Укажите следующие параметры соединения:

Drivers: Derby Embedded JDBC Driver 10.1 Default
Database location: /root/labfiles/wd026/database/library
User name: userid
Password: password

- f. Установите опции **Save password** и **Connect when the wizard complete**.

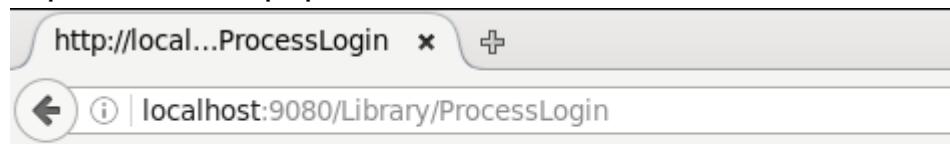


g. Нажмите **Finish**.

h. Переключитесь в перспективу **Web**.

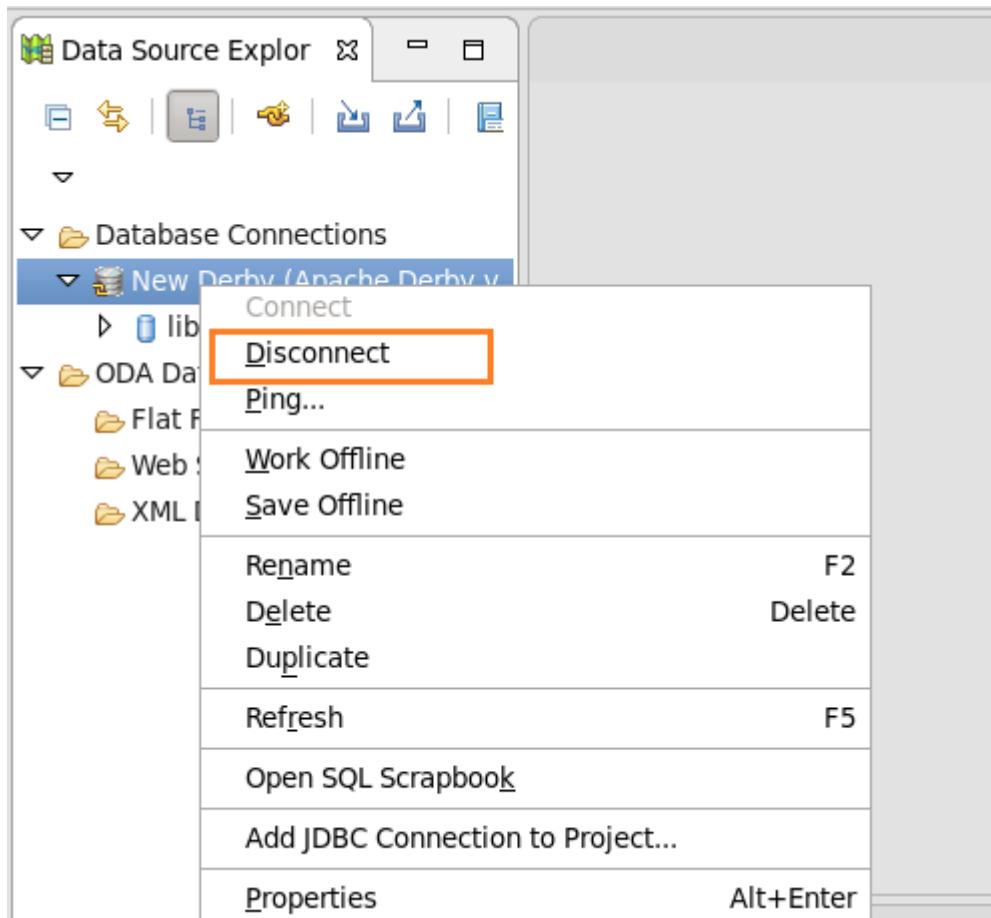
i. Запустите сервер приложений с помощью представления **Servers**.

j. Попробуйте войти в систему. Это должно привести к ошибке из-за невозможности добраться до базы данных, и должна открыться страница с информацией об ошибке.



k. Переключитесь в перспективу **Database Development**, выберите установленное соединение, нажмите по нему правой кнопкой

мыши и выберите **Disconnect**.



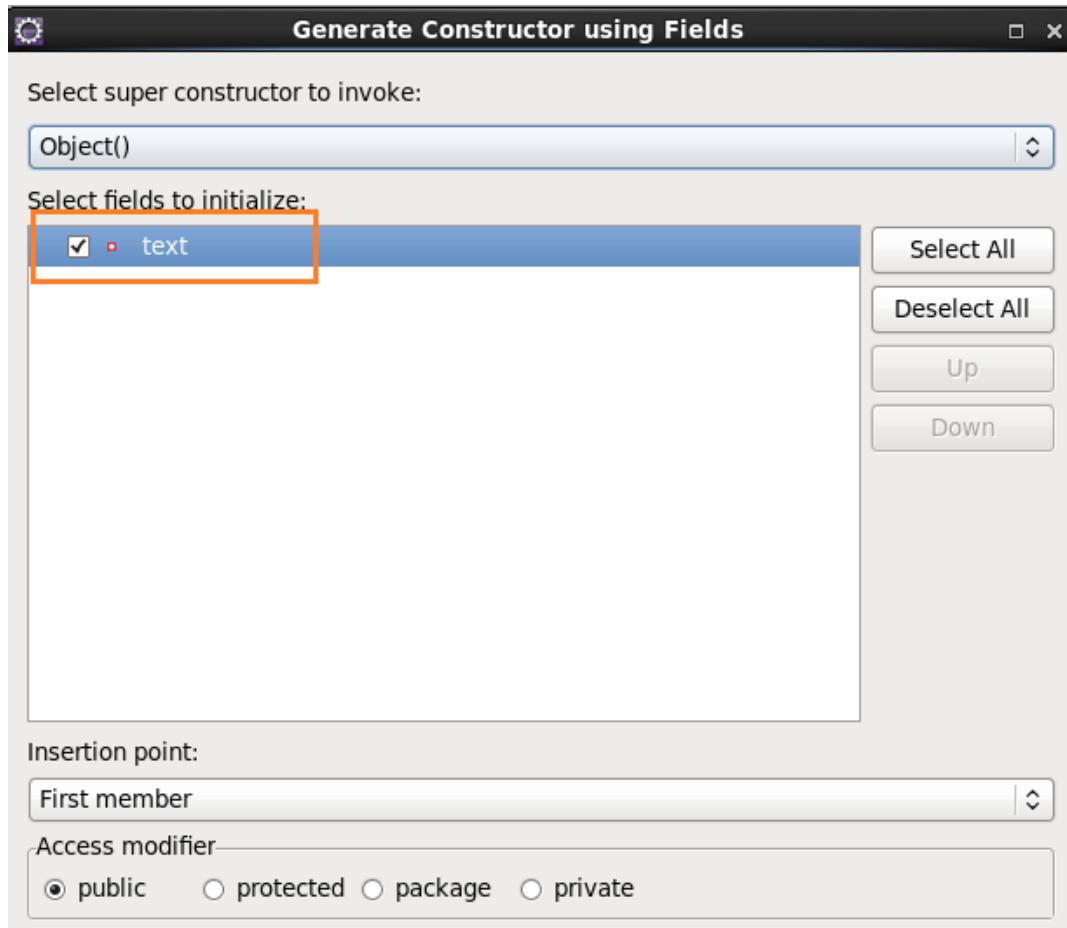
- I. Проверьте работу системы, убедитесь, что ошибок нет.

Раздел 5. Обновление JSP страниц для использования тегов <useBean>

В этом разделе организуется передача данных (сообщения о статусе/ошибке) от сервлета к JSP страницам **Register** и **RegistrationSuccess** с помощью JavaBean – **MessageBean**.

1. Измените код класса **MessageBean** для того, чтобы добавить в него новый конструктор, принимающий аргумент в виде текстового сообщения – строки.
 - a. При необходимости переключитесь в перспективу **Web**.
 - b. В представлении **Enterprise Explorer** выберите файл **LibraryWebProject -> Java Resources -> src -> com.ibm.library.servlets -> MessageBean.java**, нажмите по нему правой кнопкой мыши и выберите **Source -> Generate Constructors using Fields**.

- с. Открывается диалог по созданию конструктора. Выберите поле **text** и нажмите **OK**.



- d. Открывается редактор класса **MessageBean**, в котором появился новый конструктор с таким кодом:

```
public MessageBean(String text) {  
    super();  
    this.text = text;  
}
```

- e. Сохраните сделанные изменения и закройте файл.

2. Измените сервлет **RegisterPatron**, чтобы использовать **MessageBean** для хранения сообщения об ошибке вместо обычной строки.

- а. Откройте файл **RegisterPatron.java**.

- б. Найдите строчку в блоке `if (!isValid(patron)):`

```
req.setAttribute("message", "Registration failed: ...  
again");
```

с. Замените ее на другую строку:

```
req.setAttribute("message", new  
MessageBean("Registration failed: ... again"));
```

- d. Аналогичные инструкции (для установки атрибута) имеют место в каждом из трех **catch** блоков, следующих далее. Проделайте аналогичные манипуляции с ними.
- e. **Сохраните** сделанные изменения.
3. Измените страницу **Register.jsp**. Используйте тег **<useBean>** для обращения к объекту **Patron**, передаваемому для этой страницы. Используйте тег **<getProperty>** для чтения полей **firstName**, **lastName**, **email**. Используйте аналогичную комбинацию тегов для того, чтобы считывать и отображать информацию об ошибке.
- a. Откройте страницу **Register.jsp**. Переключитесь на вкладку **Source**.
- b. Удалите код, отвечающий за чтение атрибута **patron** из запроса:

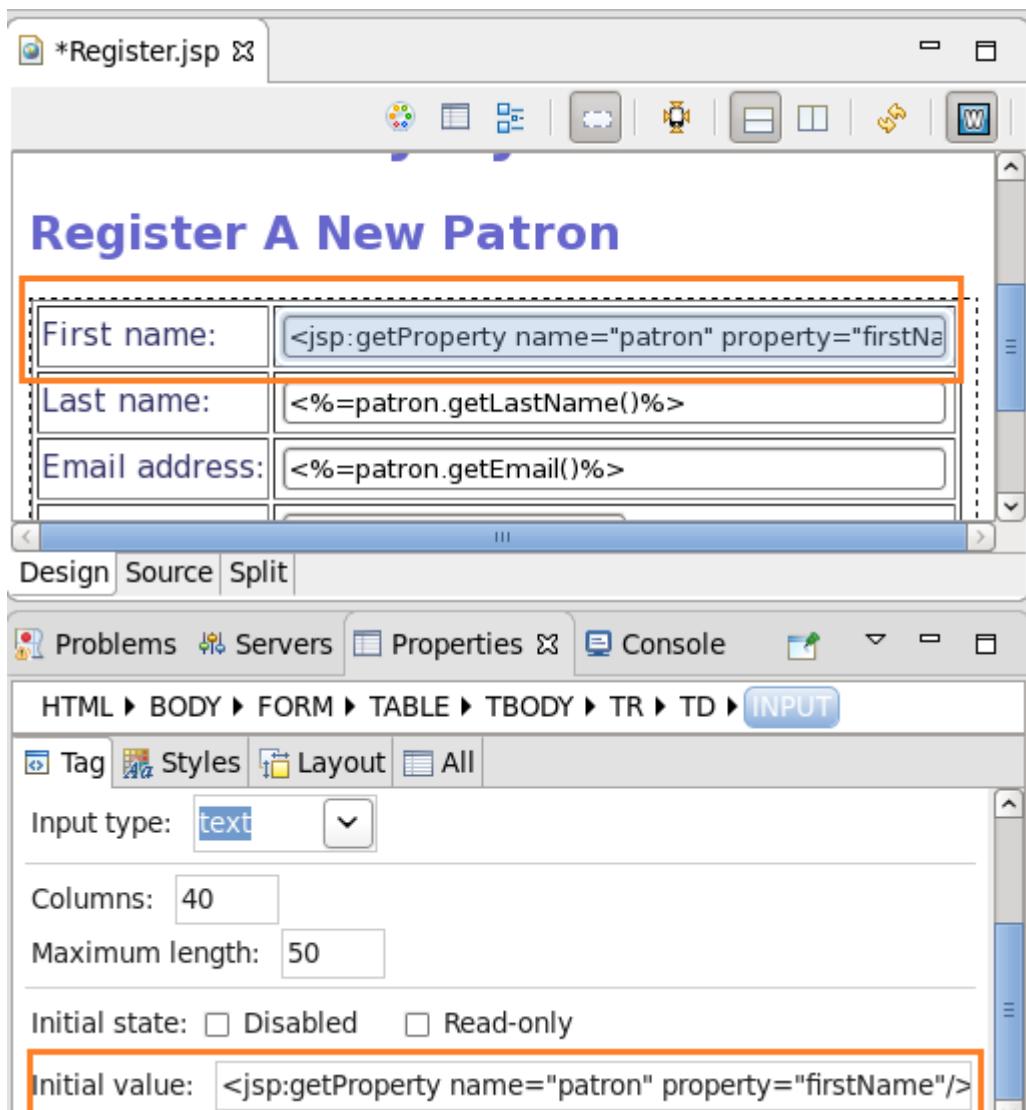
```
<%Patron patron = (Patron)  
request.getAttribute("patron");  
if (patron ==null){  
    patron = new Patron();  
} %>
```

c. Замените этот код объявлением JavaBean:

```
<jsp:useBean id="patron"  
class="com.ibm.library.model.Patron"  
type="com.ibm.library.model.Patron"  
scope="request"></jsp:useBean>
```

- d. Перейдите на вкладку **Design**.
- e. Выберите форму ввода имени (**first name**) и перейдите на вкладку **Properties**.
- f. Замените текущее поле **Initial value** на следующее:

```
<jsp:getProperty name="patron" property="firstName"  
/>
```



- g. Аналогичным образом замените изначальные значения для полей **last name** и **email**.
- h. Замените нижнюю часть страницы, где считывается значение атрибута **message**, устанавливается значение по умолчанию, и оно выводится на страницу, на следующий код:

```
<jsp:useBean id="message"
    class="com.ibm.library.servlets.MessageBean"
    type="com.ibm.library.servlets.MessageBean"
    scope="request"></jsp:useBean>
<jsp:getProperty name="message" property="text" />
```

- i. По итогам всех этих манипуляций код страницы должен быть похож на следующий:



```
26     <TD>First name:</TD>
27     <TD><INPUT name="FIRST_NAME" maxlength="50" size="40" type="text" value='<jsp:getProperty name="'
28 |     </TR>
29     <TR>
30         <TD>Last name:</TD>
31         <TD><INPUT name="LAST_NAME" maxlength="50" size="40" type="text" value='<jsp:getProperty name="'
32     </TR>
33     <TR>
34         <TD>Email address:</TD>
35         <TD><INPUT name="EMAIL" maxlength="255" size="40" type="text" value='<jsp:getProperty name="pat
36     </TR>
37     <TR>
38         <TD>Password</TD>
39         <TD><INPUT name="PASSWORD" maxlength="20" size="20" type="password"></TD>
40     </TR>
41     <TR>
42         <TD><INPUT name="SUBMIT" type="submit" value="Register"></TD>
43         <TD></TD>
44     </TR>
45   </TBODY>
46 </TABLE>
47 </FORM>
48
49 <jsp:useBean id="message" class="com.ibm.library.servlets.MessageBean"
50   type="com.ibm.library.servlets.MessageBean" scope="request"></jsp:useBean><jsp:getProperty
51   name="message" property="text" />
52
53 </BODY>
```

- j. Сохраните сделанные изменения.

4. Измените страницу **RegistrationSuccess.jsp**. Используйте тег **<useBean>** для обращения к объекту **Patron**, передаваемому для этой страницы. Используйте тег **<getProperty>** для чтения полей **email** и **id**.

- a. Откройте страницу **RegistrationSuccess.jsp**. Переключитесь на вкладку **Source**.
- b. Удалите весь код, отвечающий за чтение атрибута **patron** и его заполнение значениями по умолчанию. Замените его следующим:

```
<jsp:useBean id="patron"
  class="com.ibm.library.model.Patron"
  type="com.ibm.library.model.Patron"
  scope="request"></jsp:useBean>
```

- c. Замените абзац для вывода сообщения об успешной регистрации на следующий код:

```
<p>Patron <jsp:getProperty name="patron"
  property="email" /> was added with id
  <jsp:getProperty name="patron" property="id" /></p>
```

d. После выполнения всех описанных манипуляций код страницы должен быть похож на следующий:

```
*RegistrationSuccess.jsp
1 <!DOCTYPE HTML><%@page language="java"
2     contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" import="com.ibm.library.model.Patron"%>
3<html>
4<head>
5 <link rel="stylesheet" href="theme/Master.css" type="text/css">
6 <title>RegistrationSuccess</title>
7 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
8 </head>
9<body>
10 <h1>IBM Library System</h1>
11
12<jsp:useBean id="patron"
13    class="com.ibm.library.model.Patron"
14    type="com.ibm.library.model.Patron" scope="request"></jsp:useBean>
15
16<p>Patron <jsp:getProperty name="patron" property="email" /> was added with id <jsp:getProperty
17    name="patron" property="id" /></p>
18
19</body>
20</html>
```

e. Сохраните сделанные изменения.

5. Повторите все сценарии тестирования для проверки работы измененных страниц и сервлетов. Убедитесь, что система работает в соответствии с вашими ожиданиями.

Конец упражнения

Упражнение 9. Использование JSP Expression Language

О чём это упражнение:

В этой лабораторной работе вы должны переделать JSP страницы вашего проекта, что воспользоваться преимуществами JSP Expression Language (JSP EL). Такой подход в ряде случаев позволяет сократить время на создание логики JSP страниц.

Что вы должны будете сделать:

- Заменить обычный Java код в JSP на директивы JSP EL
- Сократить количество Java кода в JSP страницах
- Отказаться от использования тегов <useBean> и <getProperty> в пользу использования соответствующих инструкций JSP EL

Раздел 1. Изменение JSP страниц для системы регистрации

В этом разделе лабораторной работы вы вносите корректизы, описанные выше, в страницы **Register.jsp** и **RegistrationSuccess.jsp**.

1. Замените все скриптовые фрагменты и обращение к JavaBean в коде страницы **Register.jsp** на соответствующие выражения JSP EL.
 - a. Откройте страницу **Register.jsp**. Перейдите на вкладку **Source**.
 - b. Найдите первый тег **<useBean>** и удалите его вместе со всем содержимым.
 - c. Найдите тег **<getProperty>** для параметра **firstName**. Удалите его и вместо него используйте следующую конструкцию:

```
 ${patron.firstName}
```

- d. После этого соответствующая строчка должна выглядеть так:

```
<TD><INPUT name="FIRST_NAME" maxlength="50"
size="40" type="text"
value="${patron.firstName}"></TD>
```

- e. Для следующих двух полей ввода аналогично замените значения для полей ввода, получаемые с помощью тега **<getProperty>** на соответствующие JSP EL конструкции: **\${patron.lastName}** и **\${patron.email}**. В языке JSP EL есть специальный компонент, который разрешает имена переменных, отыскивая их в разных областях видимости: page, request, session, application. В данном случае без дополнительного указания этот компонент понимает, что речь идет об области видимости request.
 - f. Найдите второй тег **<useBean>** (для **message**) и удалите его.
 - g. Найдите тег **<getProperty>** для объекта **message** и удалите его.

Замените его на следующее выражение:

```
<P>${message.text}<P>
```

- h. Переключитесь на вкладку **Design**. Страница должна выглядеть следующим образом.

The screenshot shows the 'Register A New Patron' page of the 'IBM Library System'. At the top, there are two tabs: 'Register.jsp' (selected) and 'RegistrationSuccess.jsp'. The main content area has a title 'IBM Library System' and a subtitle 'Register A New Patron'. Below this is a form with five input fields:

First name:	<code> \${patron.firstName}</code>
Last name:	<code> \${patron.lastName}</code>
Email address:	<code> \${patron.email}</code>
Password	<input type="password"/>
Register	

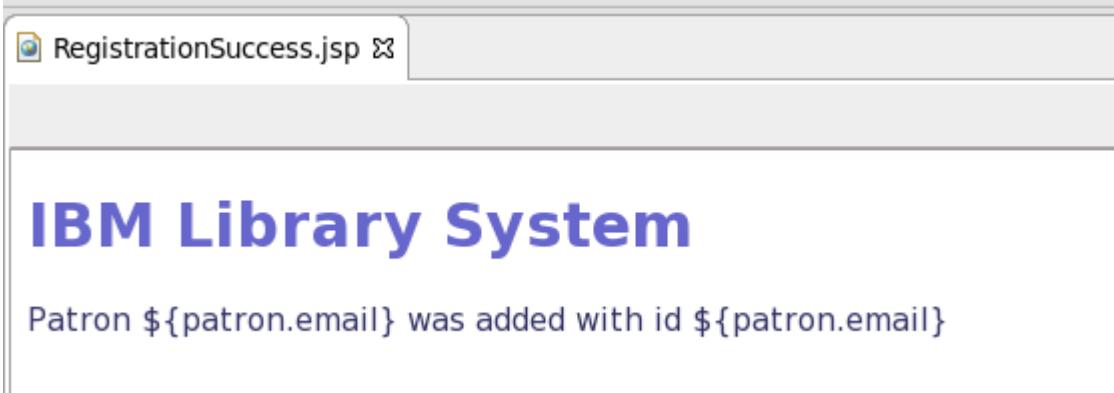
Below the form, there is a message: `${message.text}`.

- i. Сохраните сделанные изменения и закройте файл **Register.jsp**.
2. Замените все скриптовые фрагменты и обращение к JavaBean в коде страницы **RegistrationSuccess.jsp** на соответствующие выражения JSP EL.

- a. Откройте страницу **RegistrationSuccess.jsp**. Перейдите на вкладку **Source**.
b. Найдите первый тег **<useBean>** и удалите его вместе со всем содержимым.
c. Найдите тег **<getProperty>** для параметра **email**. Удалите его и вместо него используйте следующую конструкцию:

```
 ${patron.email}
```

- d. На следующей строчке замените тег **<getProperty>** на выражение `${patron.id}`
e. Перейдите на вкладку **Design**. Страница должна выглядеть следующим образом:



- f. Сохраните сделанные изменения и закройте файл **RegistrationSuccess.jsp**.
3. Проверьте работу системы регистрации.
 - a. Проверьте кейс с незаполненными полями при регистрации.
 - b. Проверьте кейс с регистрацией без ввода пароля.
 - c. Попробуйте зарегистрировать пользователя, который уже есть в системе.
 - d. Зарегистрируйте нового пользователя с корректным вводом всех данных.

Раздел 2. Изменение JSP страниц системы входа и просмотра списка книг

В этом разделе лабораторной работы вы вносите корректизы, описанные выше, в страницы **Login.jsp** и **ListItems.jsp**

1. Замените скриптовые фрагменты в коде страницы **Login.jsp** на соответствующие выражения JSP EL.
 - a. Откройте страницу **Login.jsp**. Перейдите на вкладку **Source**.
 - b. Найдите Java код для описания значения в первом поле ввода (**INPUT**) и удалите его. Замените его на следующее выражение:

`${idcookie}`

- c. После этого соответствующая строчка должна выглядеть так:

```
<td><input type="text" name="PATRON_ID" size="40"  
maxlength="50" value="${idcookie}"></td>
```

- d. В конце файла найдите выражение для считывания и вывода сообщения от ошибке:

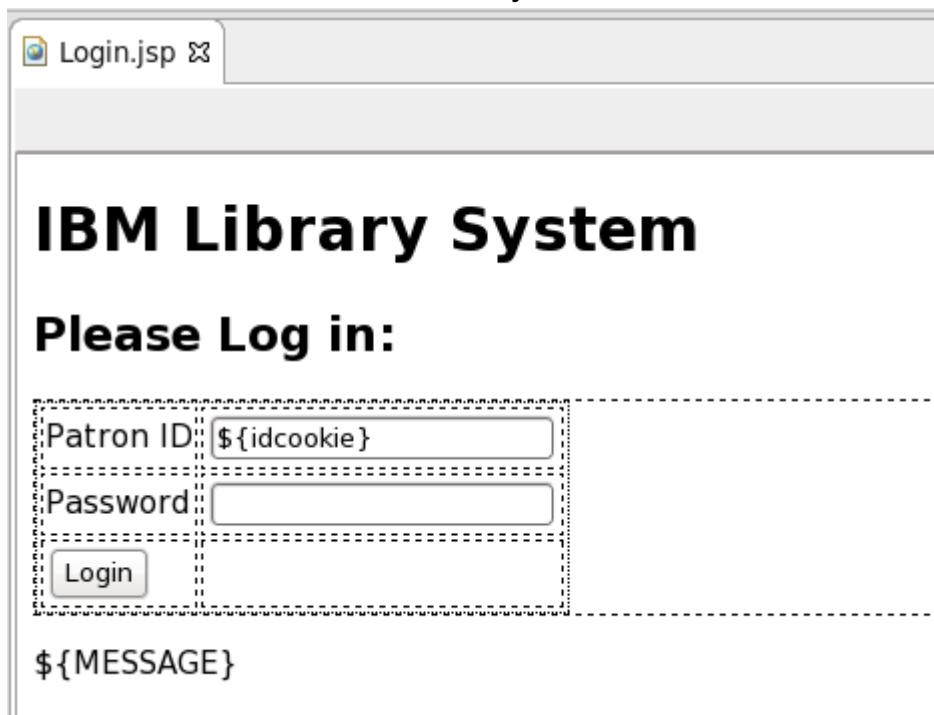
```
<%String message = (String)  
request.getAttribute("MESSAGE") ;%>  
<p><%= (message == null) ? "" : message%></p>
```

е. Замените его следующим выражением:

```
<p>${MESSAGE}</p>
```

f.

g. Переключитесь на вкладку **Design**. Внешний вид страница должен соответствовать показанному ниже.



h. Сохраните сделанные изменения и закройте страницу.

2. Замените скриптовые фрагменты и обращения к JavaBean в коде страницы **Login.jsp** на соответствующие выражения JSP EL

а. Откройте страницу **ListItems.jsp**. Переключитесь на вкладку **Source**.

б. Найдите **второй** тег **<useBean>**, который относится к объекту **item**. Удалите его.

с. Найдите первый тег **<getProperty>**, который относится к параметру **author**. Замените его на следующее выражение:

```
${item.author}
```

- d. Аналогичным образом замените остальные теги `<getProperty>` для параметров `title`, `copyNumber`, `due` на соответствующие выражения JSP EL: `${item.title}`, `${item.copyNumber}`, `${item.due}`.
- e. Переключитесь на вкладку **Design**. Внешний вид страницы должен быть похож на показанный ниже:

The screenshot shows a web application window titled "ListItems.jsp". The main content area has a header "IBM Library System" and a sub-header "My Checked Out Items". Below this is a table with four columns: "Author", "Title", "Copy", and "Due Date". The table contains one row with the following values: "\${item.author}", "\${item.title}", "\${item.copyNumber}", and "\${item.due}".

Author	Title	Copy	Due Date
\${item.author}	\${item.title}	\${item.copyNumber}	\${item.due}

- f. Сохраните сделанные изменения.
- g. Убедитесь, что в представлении **Problems** отсутствуют сообщения об ошибках.
3. Проверьте работу системы.
- Проверьте кейс с незаполненными полями при входе в систему.
 - Проверьте кейс с неправильным паролем.
 - Проверьте кейс с пользователем, который уже взял какие-то книги.
 - Проверьте кейс с пользователем, у которого пока нет взятых книг.
 - Убедитесь, что система работает также как раньше, до внесенных изменений.

Конец упражнения

Упражнение 10. Новые JSP теги

О чём это упражнение:

В этой лабораторной работе вы расширите функционал слоя представления (view) приложения. Новые функции позволят выбирать книгу из списка выданных и продлять ее (сдвигать вперед срок ее возврата). В данной лабораторной работе используются специальные JSP теги для форматирования данных в таблице, а именно для указания на те книги, которые были продлены.

Что вы должны будете сделать:

- Использовать тег из JSTL
- Создать свой собственный JSP тег
- Создать файл описание (дескриптор) для своего собственного тега.
- Использовать созданный тег в JSP странице
- Использовать View Bean на JSP странице

Раздел 1. Использование тега JSTL для цикла

В этом разделе лабораторной работы вы удаляете оставшийся Java код из страницы **ListItems.jsp**, заменяя его на стандартный тег из библиотеки JSTL **<forEach>**. Этот тег позволяет пройтись по некому перечню элементов в цикле. Этот перечень (коллекция, карта, массивы объектов или примитивных типов) указывается в атрибуте **items**, а текущее значение в рамках итерации попадает в переменную **var**.

1. Замените скриптовый фрагмент страницы **ListItems.jsp** на JSTL тег **<forEach>**.

- Откройте страницу **ListItems.jsp**. Перейдите на вкладку **Source**.
- Добавьте директиву для подключения необходимой библиотеки тегов. Пусть эта строчка будет второй в файле (перед тегом **<html>**):

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

- Замените старую конструкцию для прохода в цикле на новую, с использованием JSTL, как показано на картинке:



```

9 <link rel="stylesheet" href="<%=request.getContextPath()%>/theme/Master.css" type="text/css">
10 <title>ListItems</title>
11 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
12 </head>
13<body>
14   <jsp:useBean class="com.ibm.library.servlets.LoanedCopyListBean" id="listitems" scope="session">
15   <h2>My Checked Out Items</h2>
16<table>
17  <tbody>
18    <tr>
19      <th>Author</th>
20      <th>Title</th>
21      <th>Copy</th>
22      <th>Due Date</th>
23    </tr>
24<c:forEach items="${listitems.loanedCopyList}" var="item">
25  <tr>
26    <td align="center">${item.author}</td>
27    <td align="center">${item.title}</td>
28    <td align="center">${item.copyNumber}</td>
29    <td align="center">${item.due}</td>
30  </tr>
31</c:forEach>
32</tbody>
33</table>
34
35</body>
36</html>

```

- Сохраните сделанные изменения.

2. Протестируйте работу страницы **ListItems.jsp**. Вы должны убедиться в том, что изменений в работе системы не произошло.

Раздел 2. Создание своего тега JSP

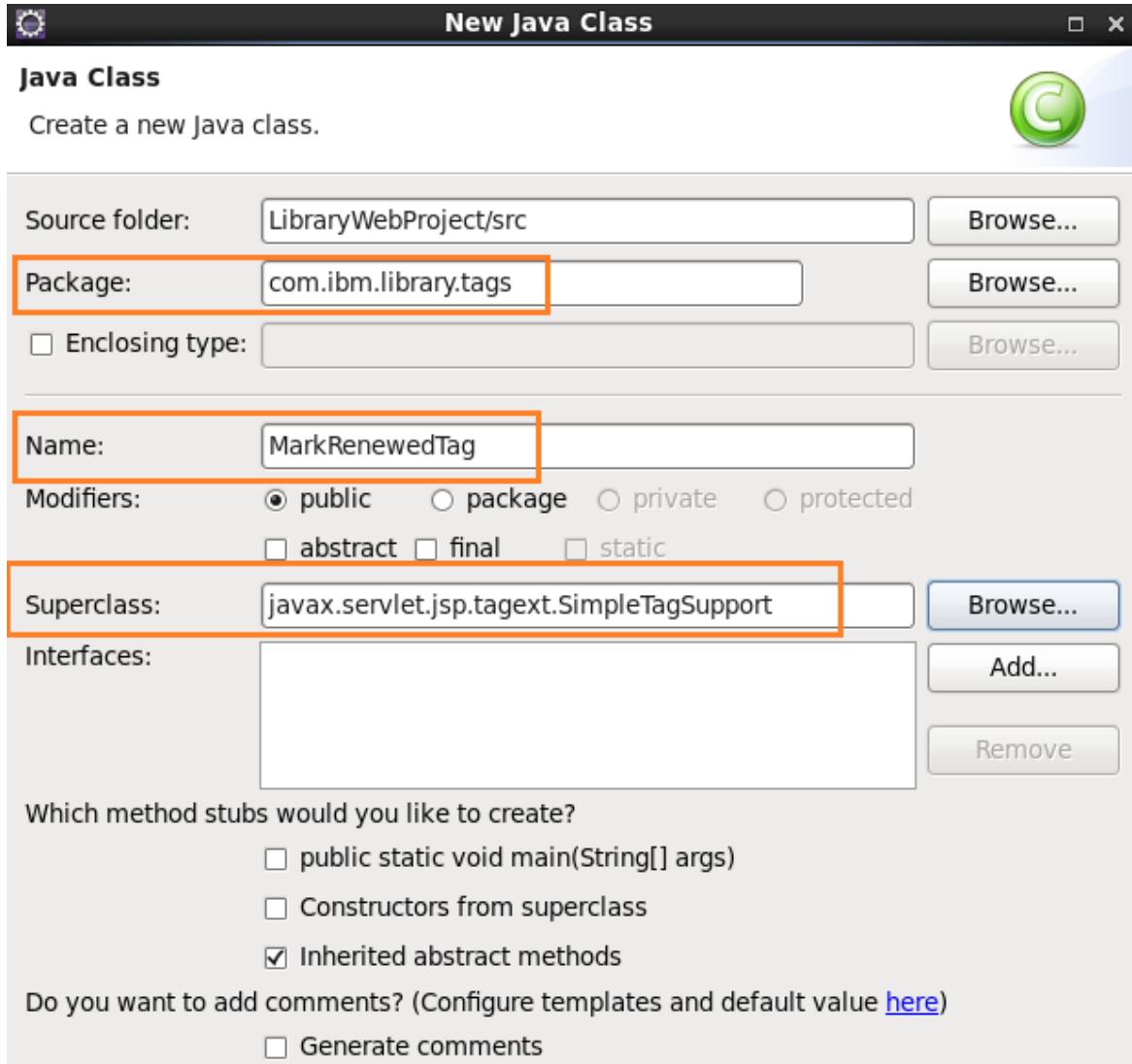
В этом разделе лабораторной работы создается тег **hilite**. Его идея в том, что при успешной продлении книжки, она должна подсвечиваться. У тега будет два атрибута: **name** – обязательный, и **color** – опциональный. **Name** ссылается на название связанного объекта **LoanedCopy**, а **color** определяет цвет жирного шрифта для обозначения обновленной позиции. Логика работы тега следующая: у связанного объекта есть поле **renewAccomplished** типа **boolean**, если значение этого параметра **true**, то текст внутри этого тега меняется: становится полужирным и меняется его цвет на тот, который указан в атрибуте **color**. Пример использования:

```
<ilib:hilite name="item" color="navy">  
    ${item.author}  
</ilib:hilite>
```

1. Создайте новый класс **MarkRenewedTag** в пакете **com.ibm.library.tags**, который расширяет класс **SimpleTagSupport**.
 - a. При необходимости переключитесь в перспективу **Web**.
 - b. В представлении **Enterprise Explorer** нажмите правой кнопкой мыши по **LibraryWebProject -> Java Resources -> src** и выберите **New -> Class**.
- c. Введите следующие данные в мастере создания нового класса:

Package:	com.ibm.library.tags
Name:	MarkRenewedTag
Superclass:	нажав кнопку Browse выберите

SimpleTagSupport



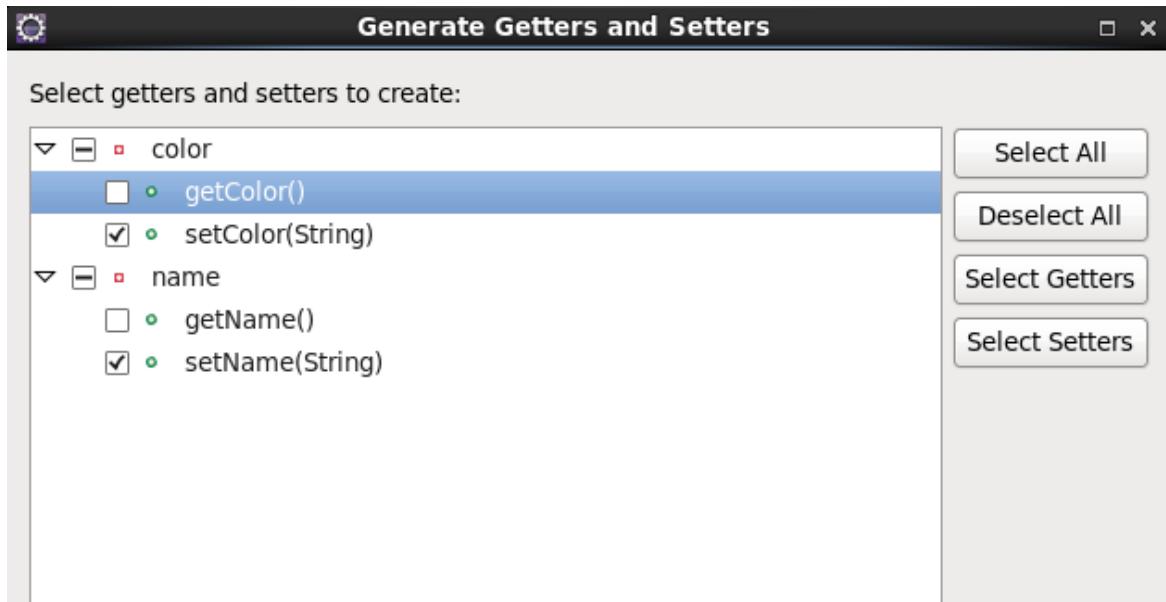
d. Нажмите **Finish**.

2. Создайте приватные поля класса для атрибутов (**name** и **color**) тега и сгенерируйте методы для их изменения.
a. В классе добавьте следующие строчки:

```
private String color = "green";  
private String name;
```

- b. Перейдите на пустую строку в редакторе, нажмите правую кнопку мыши и выберите **Source -> Generate Getters and Setters**. Выберите методы **setColor(String)** и **setName(String)** для полей

color и **name**. Нажмите **OK**.



с. После этого код класса должен быть похож на этот:

```
1 package com.ibm.library.tags;
2
3 import javax.servlet.jsp.tagext.SimpleTagSupport;
4
5 public class MarkRenewTag extends SimpleTagSupport {
6
7     private String color = "green";
8     private String name;
9
10    public void setColor(String color) {
11        this.color = color;
12    }
13    public void setName(String name) {
14        this.name = name;
15    }
16
17
18
19 }
```

3. В созданном классе создайте метод **doTag()** без параметров, который выбрасывает исключения **JspException**, **IOException**. Этот метод вызывается веб-контейнером, при обработке этого тега. Метод наследуется от родительского класса.

а. Расположите курсор после реализации последнего метода класса.
б. Нажмите правой кнопкой мыши и выберите **Source -> Override/Implement Methods**. Выберите метод **doTag()** и нажмите кнопку **OK**.

4. Реализуйте первую часть метода **doTag()**, которая обрабатывает значение атрибута **name**. Логика должна быть следующей: используя имя, указанное в качестве значения этого атрибута, нужно

найти соответствующий объект в **JspContext** с использованием метода **findAttribute()**. Нужно убедиться, что это не **null**, в противном случае выбросить **JspException**. Нужно убедиться, что это тип этого объекта **LoanedCopy**, в противном случае выбросить **JspException**. Далее нужно определить продлялся ли этот объект с помощью метода **isRenewAccomplished()**.

а. Замените текущую реализацию метода **doTag()** следующим кодом:

```
Object obj = getJspContext().findAttribute(name);
if (obj == null) {
    throw new JspException("Hilite tag: Name " + name
+ " not found");
}
if (!(obj instanceof LoanedCopy) ) {
    throw new JspException("Hilite tag: Name is not
of type "
+ "com.ibm.library.model.LoanedCopy");
}
boolean renewed = ((LoanedCopy)
obj).isRenewAccomplished();
```

б. Для разрешения возникшей ошибки импортируйте класс **com.ibm.library.model.LoanedCopy**.

5. Вторая часть метода должна генерировать необходимый фрагмент HTML разметки по следующим правилам: сначала нужно получить ссылку на содержимое тега (объект **JspFragment**) с помощью метода **getJspBody()**. Запись туда можно вести с помощью объекта **StringWriter**. Если продление было произведено, то содержимое тега нужно предварить элементами ****. Завершаться же содержимое тега должно соответствующими закрывающими тегами ****. Использование тега **** противоречит современным хорошим практикам работы с HTML, и в данном случае он использован только лишь для обеспечения лаконичности и простоты примера.

а. Добавьте следующий код в конец метода **doTag()**:

```
// Process tag body
StringWriter bodyText = new StringWriter();
```

```
JspFragment body = getJspBody();
if (body == null) return;

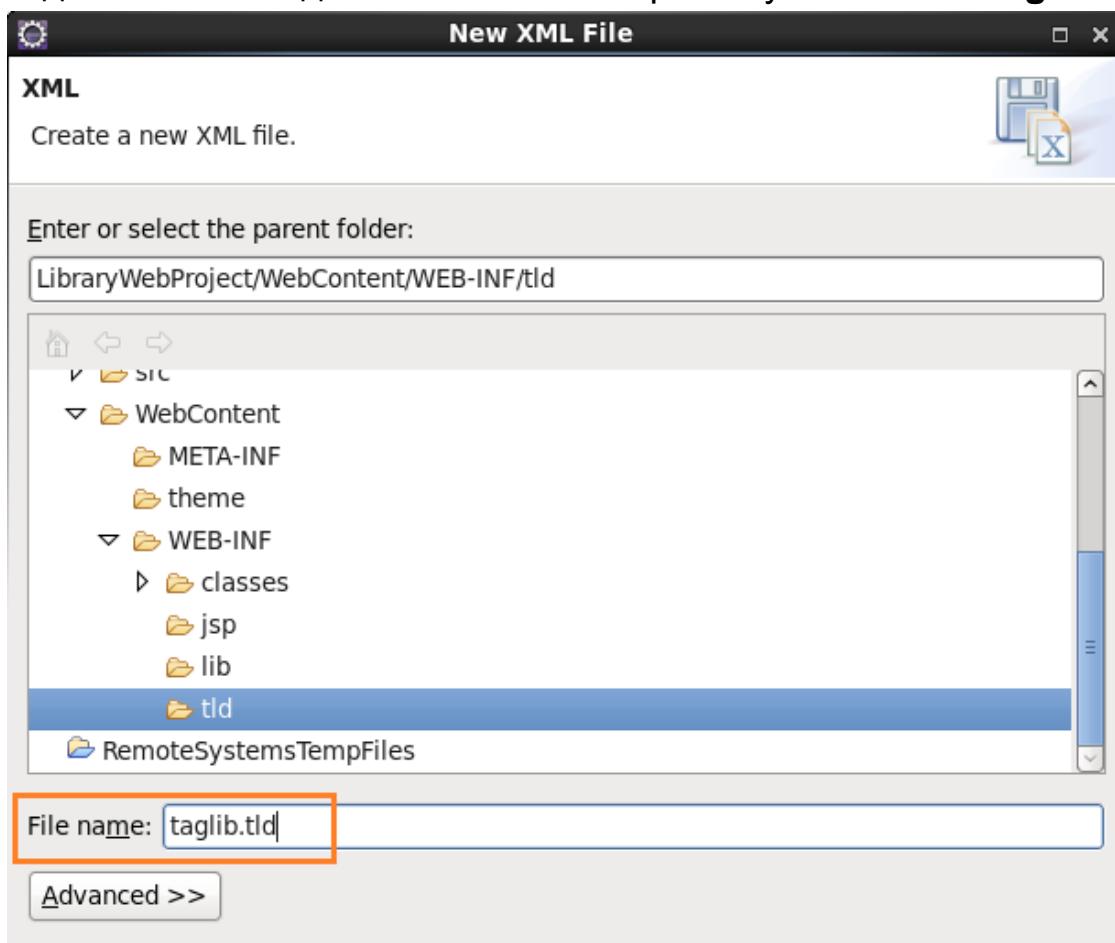
body.invoke(bodyText);
StringBuffer buffer = new StringBuffer();
// Place font and bold tags at start of body
// if renewed flag set
if (renewed) {
    buffer.append("<b><font color=\"");
    buffer.append(color);
    buffer.append("\">");
}
// Copy in the body for either case
buffer.append(bodyText.toString());
// Place font end tag at end of body if renewed
if (renewed) {
    buffer.append("</font></b>");
}
try {
    JspWriter out = getJspContext().getOut();
    out.write(buffer.toString());
} catch (IOException e) {
    throw new JspException("Hilite tag: " +
e.getMessage());
}
```

- b. **Импортируйте** необходимые классы для разрешения возникших ошибок.
- c. **Сохраните** сделанные изменения.

Раздел 3. Создание дескриптора для тега собственной разработки

В этом разделе лабораторной работы создается tag library descriptor (TLD) для описания только что созданного JSP тега. JSP контейнер использует этот дескриптор, чтобы связать тег с нужным Java классом. Также он помогает определить корректность использования тега и его атрибутов. Для того, чтобы использовать этот TLD в JSP странице нужно использовать директиву, которая ссылается на него.

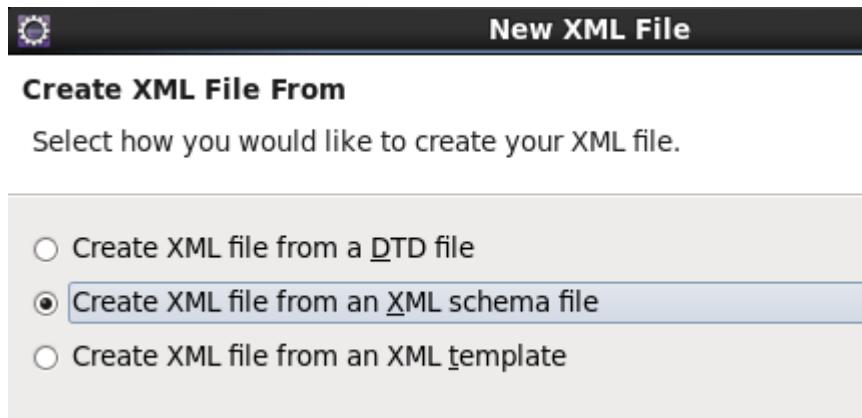
1. Создайте новый каталог **tld** в **LibraryWebProject -> WebContent -> WEB-INF**.
 - a. Нажмите правой кнопкой мыши по **LibraryWebProject -> WebContent -> WEB-INF** и выберите **New -> Folder**.
 - b. Введите **tld** в поле **Folder Name** и нажмите **Finish**.
2. Создайте файл **taglib.tld** для описания тега в новом каталоге.
 - a. Нажмите правой кнопкой мыши по каталогу **tld** и выберите **New -> Other**.
 - b. Выберите пункт **XML -> XML File**.
 - c. В диалоге по созданию нового XML файла укажите имя **taglib.tld**.



- d. Нажмите **Next**.

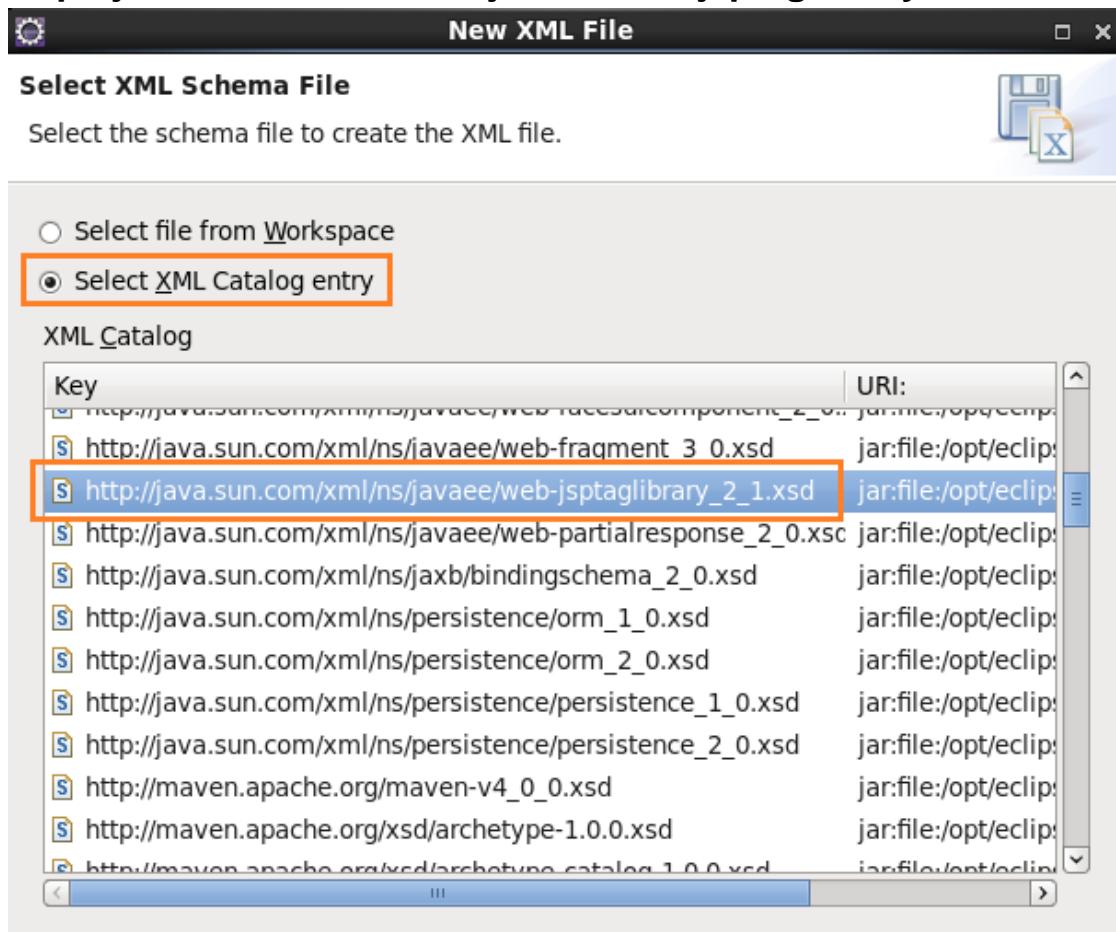
e. Выберите опцию **Create XML file from an XML schema file**.

Нажмите **Next**.



f. На следующем шаге выберите опцию **Select XML Catalog entry** и пункт:

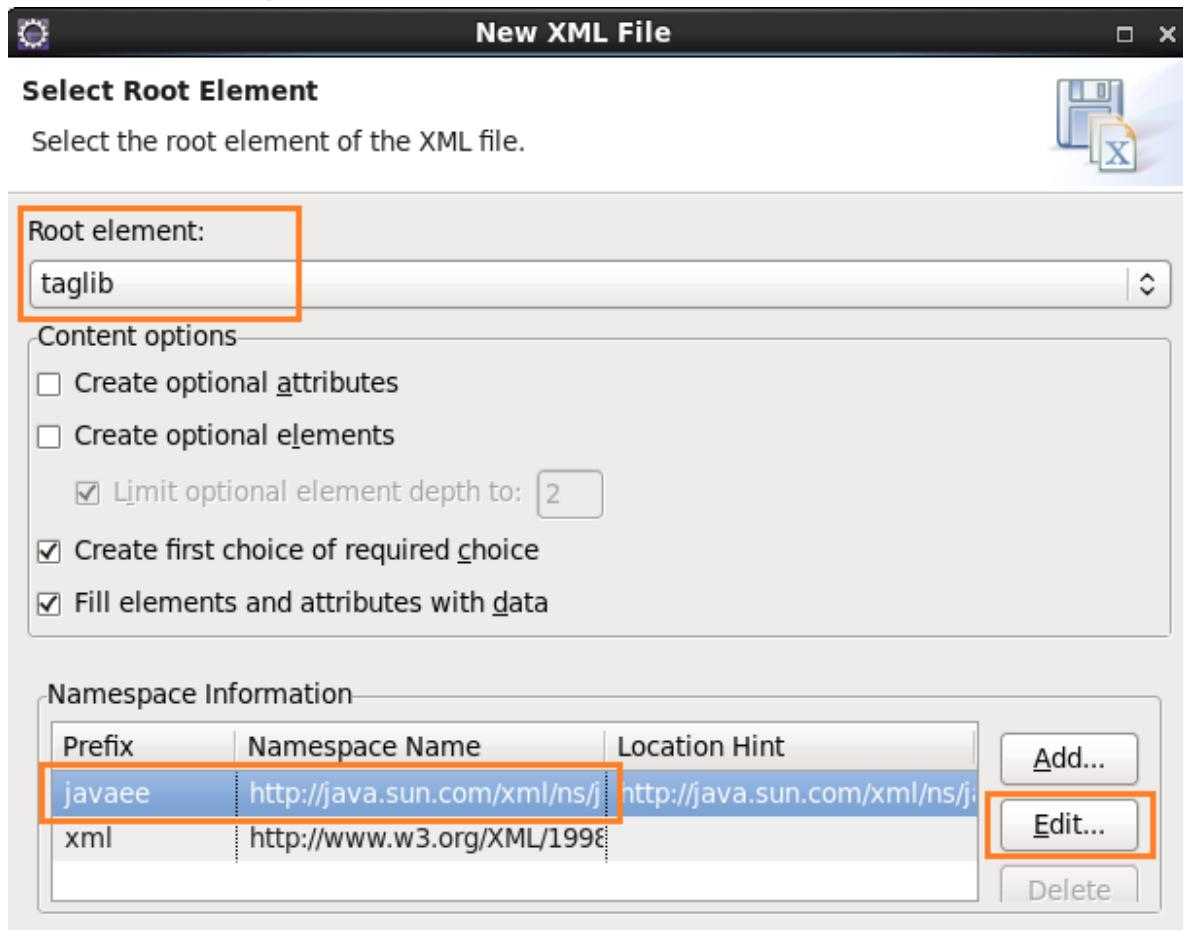
http://java.sun.com/xml/ns/javaee/web-jsptaglibrary_2_1.xsd



g. Нажмите **Next**.

h. В следующем интерфейсе оставьте корневой элемент по умолчанию – **taglib**.

- i. Выберите строчку **javaee** в таблице **Namespace Information** и нажмите кнопку **Edit**.



- j. Очистите поле **Prefix** и нажмите кнопку **OK**.



- k. Нажмите кнопку **Finish**.

- l. Файл **taglib.tld** открывается в редакторе. Перейдите на вкладку **Source**.

3. Добавьте в файл **taglib.tld** необходимое содержимое для описания нового тега.

а. Установите значение **tlib-version – 1.0**.

б. Установите значение **short-name – ilib**.

с. Добавьте следующий код после **short-name** и до закрывающего тега **</taglib>**.

```
<tag>
    <name>hilite</name>
    <tag-
class>com.ibm.library.tags.MarkRenewedTag</tag-
class>
    <body-content>scriptless</body-content>
    <attribute>
        <name>name</name>
        <required>true</required>
    </attribute>
    <attribute>
        <name>color</name>
        <required>false</required>
    </attribute>
</tag>
```

д. Сохраните сделанные изменения.

Раздел 4. Изменение *ListItems.jsp* для использования нового тега

В этом разделе лабораторной работы изменяется страница **ListItems.jsp**. Во-первых, можно будет выбрать один из пунктов списка, во-вторых, добавляется кнопка **Renew**. Также для отображения элементов списка нужно использовать новый тег **hilite**.

1. Откройте для редактирования файл **ListItems.jsp** и перейдите на вкладку **Source**.
2. Добавьте на страницу форму, которая будет связана с URL **/Library/RenewItems**. Добавьте кнопку для отправки формы на сервер.
 - а. После закрывающего тела **</h2>** и перед таблицей добавьте следующий тег:

```
<form method="post" action="/Library/RenewItems">
```

- b. После закрывающего тега таблицы `</table>` добавьте закрывающий тег для формы: `</form>`.
- c. После цикла для вывода элементов списка в таблицу, но до закрытия этой таблицы добавьте следующую строчку с описанием кнопки:

```
<tr>
    <td><input type="submit" name="renew"
value="Renew"></td>
</tr>
```

- 3. Добавьте в левую часть таблицы колонку с названием **Renew**.

Внутри будет **check box**. Для отображения элементов списка будет использован новый тег. Значение атрибута **value** должно быть привязано к выбранному элементу из списка – **itemId** (типа **LoanedCopy**).

- a. Под первой директивой **taglib** добавьте на страницу еще одну, для подключения новой библиотеки тегов:

```
<%@taglib uri="/WEB-INF/tld/taglib.tld"
prefix="ilib" %>
```

- b. Добавьте в таблицу еще одну колонку. **Перед** всеми элементами `<th>` добавьте еще один:

```
<th>Renew</th>
```

- c. Замените текущую реализацию цикла `<c:forEach>` на следующую:

```
<c:forEach items="\${listitems.loanedCopyList}"
var="item">
    <tr>
        <td><input type="checkbox" name="RENEW_ITEM"
value="\${item.itemId}">
        </td>
        <td align="center"><ilib:hilite name="item"
color="red">
            \${item.author}
        </ilib:hilite></td>
```

```
<td align="center"><ilib:hilite name="item"
color="red">
    ${item.title}
</ilib:hilite></td>
<td align="center"><ilib:hilite name="item"
color="red">
    ${item.copyNumber}
</ilib:hilite></td>
<td align="center"><ilib:hilite name="item"
color="red">
    ${item.due}
</ilib:hilite></td>
</tr>
</c:forEach>
```

В данном цикле по сравнению с предыдущей реализацией добавляется новый столбец, который можно будет использовать для продления. А все остальные столбцы оборачиваются в новый тег **hilite** для корректного отображения статуса продления.

- d. Сохраните сделанные изменения.

Раздел 5. Создание сервлета RenewItems

В этом разделе лабораторной работы создается сервлет для обработки события продления. URL для вызова этого сервлета был уже добавлен на страницу **ListItems.jsp** в предыдущей части лабораторной работы.

1. Импортируйте файл **RenewItems.java** в пакет **com.ibm.library.servlets** в проекте **LibraryWebProject**. Найти этот файл можно в каталоге **/root/labfiles/wd026/setup/**
 - a. В представлении **Enterprise Explorer** нажмите правой кнопкой мыши по пакету **LibraryWebProject -> Java Resources -> src -> com.ibm.library.servlets**. Выберите **Import**.
 - b. Выберите опцию **File System** и нажмите **Next**.
 - c. Нажмите кнопку **Browse** и найдите каталог **/root/labfiles/wd026/setup/**, нажмите **OK**.
 - d. Выберите файл **RenewItems.java** и нажмите **Finish**.
2. Добавьте аннотацию, указывающую на URL для данного сервлета.
 - a. Откройте файл **RenewItems.java** и добавьте следующую строку перед объявлением класса:

```
@WebServlet("/RenewItems")
```

- b. Разрешите возникшую проблему, **импортировав** класс **WebServlet**.

```
12 import javax.servlet.http.HttpSession;
13
14 import com.ibm.library.datastore.exceptions.SystemUnavailableException;
15 import com.ibm.library.model.LoanedCopy;
16 import com.ibm.library.model.Patron;
17
18 import javax.servlet.annotation.WebServlet;
19
20 /**
21 * Servlet implementation class for Servlet: RenewItem
22 *
23 */
24
25 @WebServlet("/RenewItems")
26 public class RenewItems extends HttpServlet {
27
28     /**
29     *
30     */
31 }
```

- c. Посмотрите реализованную логику импортированного сервлета. Принципиальная структура этого сервлета аналогична структуре тех сервлетов, которые были разработаны до этого. Смысловая часть сервлета заключается в вызове приватного метода **markRenewed()**, который находит выбранные элементы из списка и запускает для них метод **setRenewRequested()**. Опосредованно вызывается метод **renew** для изменения атрибута **Patron**.

Раздел 6. Тестирование новых функций

В этом разделе лабораторной работы проверяется работы новых сервлетов и JSP, которые были созданы в предыдущих частях. На данный момент, можно выполнить вход в систему, посмотреть список взятых позиций, запросить продление. После успешного продления отображается тот же самый список, но с индикацией успешно продленного элемента.

Список учетных записей с паролями, а также со списками взятых позиций представлен ниже:

ID	Password	Book Title	Copy	Due
1	brown	J2EE 1.4 makes me so happy	2	+4
2	vogel	J2EE 1.4 makes me so happy	3	+2
		Java for coffee drinkers	2	0
		Enterprise JavaBeans	2	-1
3	davis	None	NA	NA
4	straach	Enterprise JavaBeans	3	+1
5	vanloon	Webs Fear: A guide for arachnophobics	2	+3
6	weightman	None	NA	NA

При проведении тестирование обратите внимание на то, что возможность продления определяется в методе `renew()`. На данный момент продление возможно, если книжка до этого была продлена не более чем `MAX_TIMES_RENEWED` раз (это константа равна 3) и если книга не просрочена на момент продления. Этот факт нужно учитывать при тестировании. При необходимости данные в базе можно перегенерировать, вызвав класс `setUpLibraryDatabase`, по аналогии со вторым упражнением. Перед выполнением этой операции нужно остановить сервер.

1. Проведите регрессионное тестирование с проверкой всех старых функций.
2. Проверьте функции для продления книги и отображения продленных книг.

The screenshot shows a web browser window for the 'IBM Library System'. The title bar says 'ListItems' and the address bar shows 'http://localhost:9080/Library...'. Below the address bar is a navigation bar with icons for back, forward, search, and refresh. The main content area has a blue header 'IBM Library System' and a sub-header 'My Checked Out Items'. A table displays a single row of data:

Renew	Author	Title	Copy	Due Date
<input type="checkbox"/>	Bob Coder	J2EE 1.4 makes me so happy	2	Mon Nov 14 16:39:03 MSK 2016

Below the table is a button labeled 'Renew'.

Раздел 7. Отображение статуса продления с помощью View Bean

В этом разделе лабораторной работы добавляется очередной столбец в таблицу на странице **ListItems.jsp**. Для отображения этого столбца будет использован **view bean** (JavaBean). У этого view bean есть параметр **String**, который содержит HTML разметку для отображения на странице. Для доступа к этому JavaBean используются стандартные теги **<useBean>** и **<getProperty>**.

Новый столбец будет содержать статус продления (атрибут **renewMessage**). Если оно прошло успешно, статус будет пустым. В этом случае такая позиция подсвечивается и отображается полужирным шрифтом. В противном случае в эту строчку будет записываться сообщение, которое нужно отобразить в этом столбце.

1. Импортируйте файл **RenewalStatusBean.java**.

- a. В представлении **Enterprise Explorer** нажмите правой кнопкой мыши по пакету **LibraryWebProject -> Java Resources -> src -> com.ibm.library.servlets**. Выберите **Import**.
- b. Выберите **File System**. Нажмите **Next**.
- c. Нажмите кнопку **Browse** и найдите каталог **/root/labfiles/wd026/setup/**, нажмите **OK**.
- d. Выберите файл **RenewalStatusBean.java** и нажмите **Finish**.

2. Посмотрите на код, импортированного класса.

Метод **getStatusMarkup()** генерирует HTML для отображения статуса и возвращает его в виде объекта типа **String**.

Этот bean использует два параметра **color** и **item**. **Color** может быть установлен один раз, например, при создании этого JavaBean с помощью тега **<useBean>**. Атрибут **item** же должен быть установлен для каждого из элементов списка.

3. Обновите страницу **ListItems.jsp** для добавления туда нового столбца и использования импортированного JavaBean.

- a. Откройте файл **ListItems.jsp**. Перейдите на вкладку **Source**.
- b. Добавьте заголовок **Renewal Status** для нового последнего столбца таблицы сразу после уже существующего столбца **Due Date**:

```
<th>Due Date</th>
<th>Renewal Status</th>
```

- c. Используйте тег **<useBean>** для создания нового экземпляра JavaBean **RenewalStatusBean** с идентификатором **status**. Установите для него параметр **color**. Для этого добавьте следующий код перед открытием цикла **<c:forEach>**:

```
<jsp:useBean id="status" scope="page"
    class="com.ibm.library.servlets.RenewalStatusBean">
    <jsp:setProperty name="status" property="color"
        value="red"/>
</jsp:useBean>
```

- d. Внутри цикла **<c:forEach>** (сразу после его открытия) установите значение атрибута **item** для созданного JavaBean в текущий элемент списка:

```
<jsp:setProperty name="status" property="item"
    value="${item}" />
```

- e. Добавьте в таблицу столбец для отображения статуса продления – он должен стать последним в строке:

```
<td>${status.statusMarkup}</td>
```

- f. По итогам должен получиться следующий код:

```
<th>Title</th>
<th>Copy</th>
<th>Due Date</th>
<th>Renewal Status</th>
</tr>

<jsp:useBean id="status" scope="page" class="com.ibm.library.servlets.RenewalStatusBean">
    <jsp:setProperty name="status" property="color" value="red" />
</jsp:useBean>
<c:forEach items="${listitems.loanedCopyList}" var="item">
    <jsp:setProperty name="status" property="item" value="${item}" />
    <tr>
        <td><input type="checkbox" name="RENEW_ITEM"
            value="${item.itemId}"></td>
        <td align="center"><ilib:hilite name="item" color="red">${item.author}</ilib:hilite></td>
        <td align="center"><ilib:hilite name="item" color="red">${item.title}</ilib:hilite></td>
        <td align="center"><ilib:hilite name="item" color="red">${item.copyNumber}</ilib:hilite>
        <td align="center"><ilib:hilite name="item" color="red">${item.due}</ilib:hilite></td>
        <td>${status.statusMarkup}</td>
    </tr>
</c:forEach>
```

- g. Сохраните сделанные изменения.

- h. Проведите тестирование аналогичное тому, что было проведено в предыдущем разделе.

Раздел 8. Отображение количества продлений для книги с помощью Tool tip

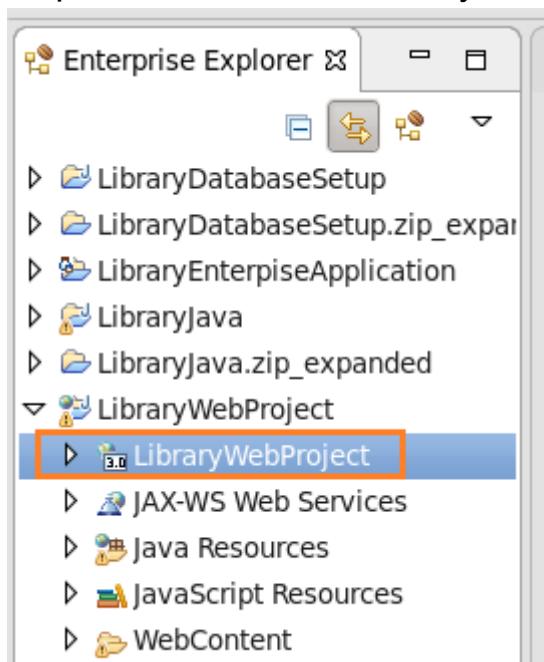
В этом разделе лабораторной работы создается новый тег файл, который использует CSS стили для отображения информации относительно количества продлений позиции (объекта типа **LoanedCopy**). Пользователь должен будет навести курсор на вопросительный знак рядом в поле **Due Date**, после чего появится всплывающее окно с требуемой информацией.

Тег файлы позволяют создавать универсальные компоненты для многократного использования. По сути дела они являются JSP страницами, генерирующими какую-то часть содержимого страницы.

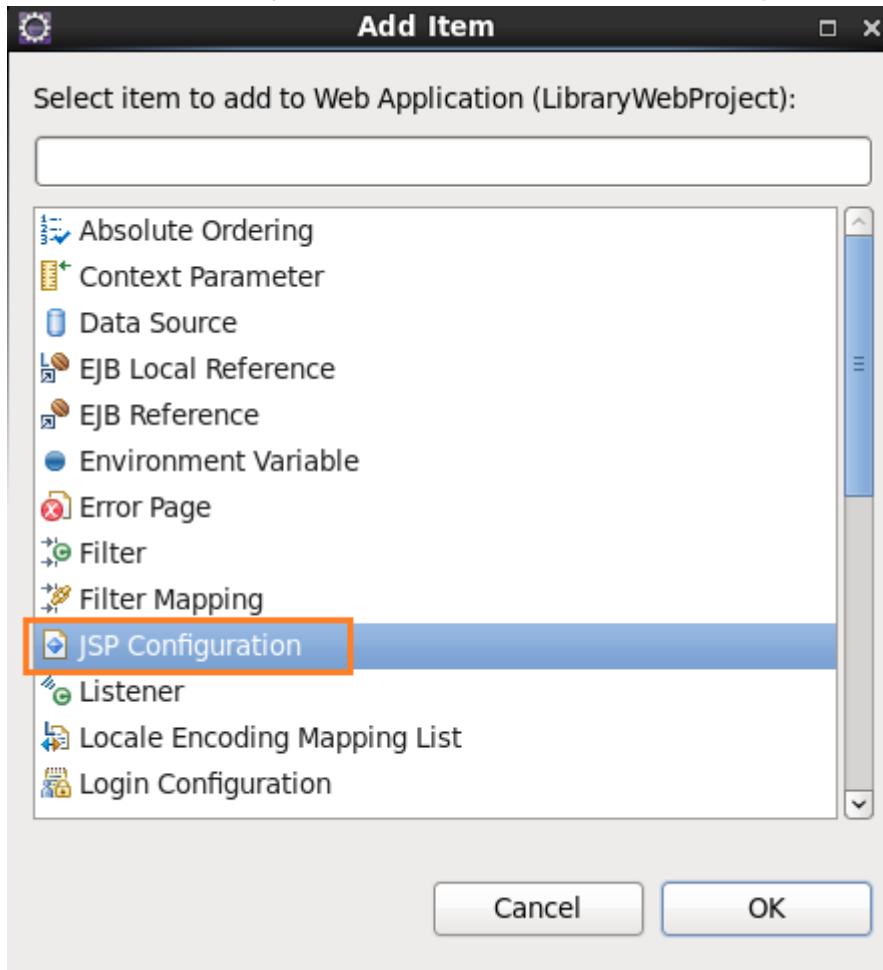
1. Добавьте необходимые стили в CSS файл, используемый JSP файлами проекта.
 - a. Откройте файл **LibraryWebProject -> WebContent -> theme -> Master.css**.
 - b. Вставьте в конец этого файла содержимое файла
/root/labfiles/wd026/snippets/exercise10-4.txt
 - c. Сохраните сделанные изменения.
2. Создайте новый каталог **tags** в директории **WEB-INF** и импортируйте туда файлы **details.tag** и **tags.tld** из каталога **/root/labfiles/wd026/setup**.
Файл **tags.tld** определяет новую библиотеку тегов **info**, новый тег **details** и расположение файла **details.tag**. Во втором же файле описана реализация тега **details**. Этот тег требует ссылки на объект **loanedCopy** и отображает в описанном выше месте количество продлений соответствующей позиции.
 - a. В представлении **Enterprise Explorer** нажмите правой кнопкой мыши по каталогу **LibraryWebProject -> WebContent -> WEB-INF**. Выберите **New -> Folder**.
 - b. Укажите **tags** как имя каталога и нажмите **Finish**.
 - c. Нажмите правой кнопкой мыши по каталогу **tags** и выберите **Import**.
 - d. Выберите **File System** и нажмите **Next**.
 - e. Нажмите кнопку **Browse** и найдите каталог **/root/labfiles/wd026/setup/**, нажмите **OK**.
 - f. Выберите файлы **details.tag** и **tags.lib**, нажмите **Finish**.
3. Посмотрите содержимое файла **details.tag**. Там описано, что тег должен быть пустым, описан атрибут **loanedCopy** типа **LoanedCopy**

и другие детали реализации, которые могут быть сопоставлены со стилями из **Master.css**.

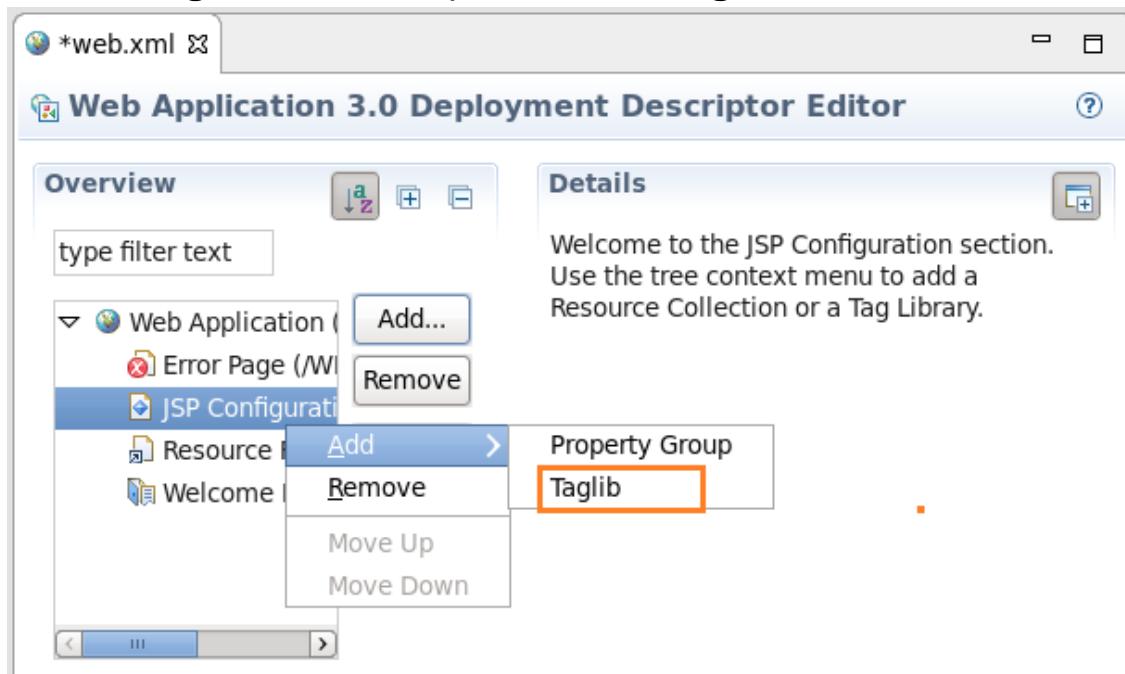
4. Дайте ссылку на новый дескриптор тега в дескрипторе развертывания веб-приложения **LibraryWebProject**.
 - a. Откройте дескриптор развертывания проекта **LibraryWebProject**. Переключитесь на вкладку **Design**.



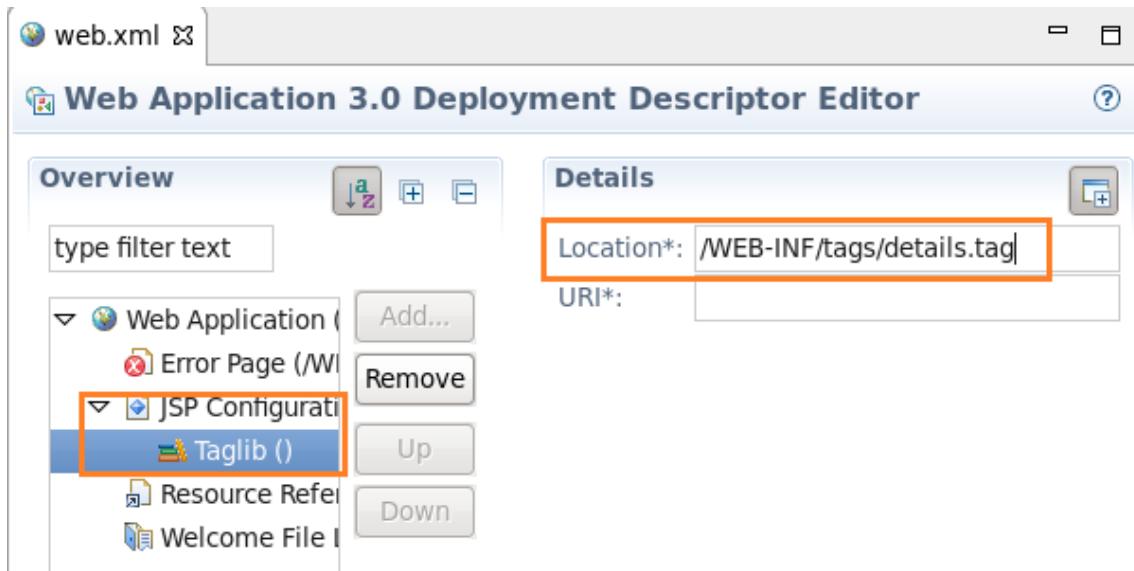
б. Нажмите кнопку **Add**. Выберите **JSP Configuration**. Нажмите **OK**.



с. Нажмите правой кнопкой мыши по только что добавленному узлу **JSP Configuration**. Выберите **Add -> Taglib**.



- d. В разделе **Details** укажите **/WEB-INF/tags/details.tag** в поле **Location**.



e. Сохраните сделанные изменения.

5. Добавьте на страницу **ListItems.jsp** тег **details** для отображения количества продлений.
- Откройте файл **ListItems.jsp**.
 - Перейдите на вкладку **Source**.
 - Найдите в верхней части страницы две директивы **<taglib>**. Ниже них добавьте еще одну аналогичную директиву для подключения новой библиотеки:

```
<%@taglib tagdir="/WEB-INF/tags" prefix="info" %>
```

d. Добавьте в ячейку, где указывается срок сдачи, новый тег:

```
<td align="center">
    <ilib:hilite name="item" color="red">
        <info:details loanedCopy="${item}" />
        ${item.due}
    </ilib:hilite>
</td>
```

e. Сохраните сделанные изменения.

6. Проверьте новые функции. Если стиль страницы не обновляется, почистите кэш браузера. Убедитесь в корректном отображении

количества продлений:

IBM Library System

My Checked Out Items

Renew	Author	Title	Copy	Due Date	Renewal Status
<input type="checkbox"/>	Peter Parker	Webs Fear: A guide for arachnophobics	2	Mon Nov 14 00:00:00 MSK 2016	<div style="border: 1px solid blue; padding: 5px; text-align: center;">Times Renewed = 2</div>

Конец упражнения

Упражнение 11. Сервлетные фильтры

О чём это упражнение:

В этой лабораторной работе вы создадите два сервлетных фильтра. Один нужен для подсчета времени на обработку запросов к разработанному приложению. Второй нужен для добавления дополнительного контента к тем страницам, которые возвращаются пользователю.

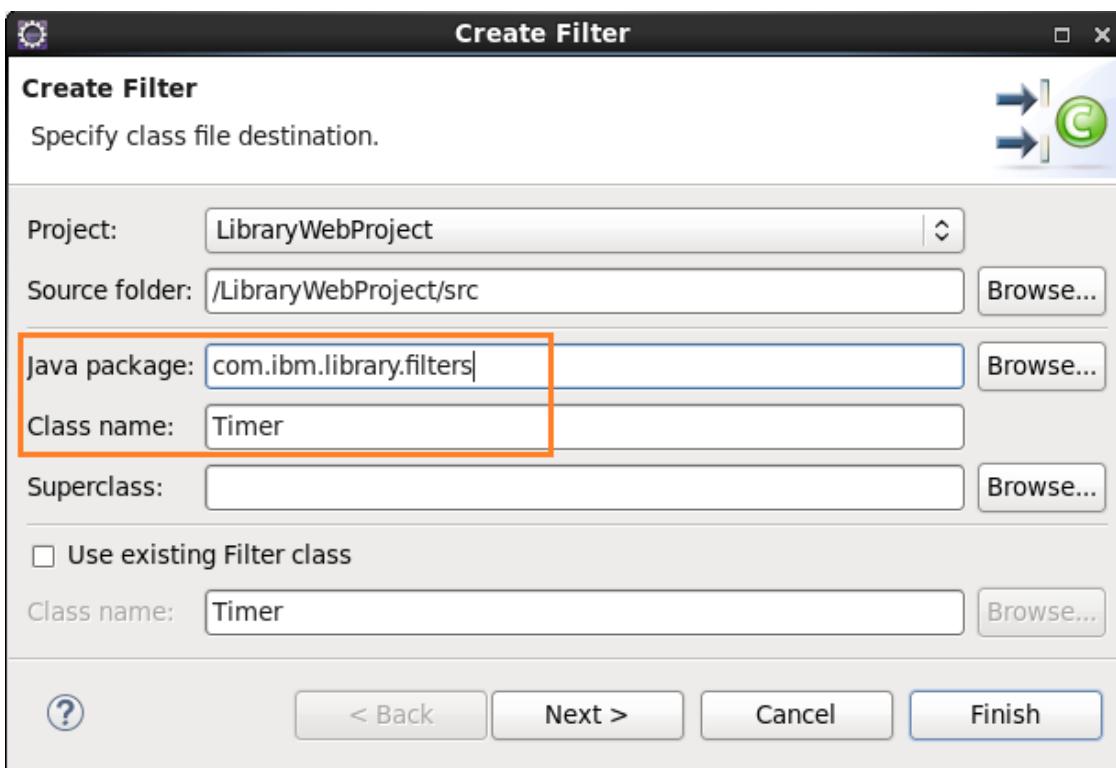
Что вы должны будете сделать:

- Создать заготовки сервлетных фильтров с помощью соответствующего мастера среды Eclipse
- Разработать логику сервлетного фильтра для сервлета
- Сделать фильтр, изменяющий возвращаемый пользователю контент
- Настроить использование фильтров в дескрипторе развертывания

Раздел 1. Создание и настройка фильтра таймера

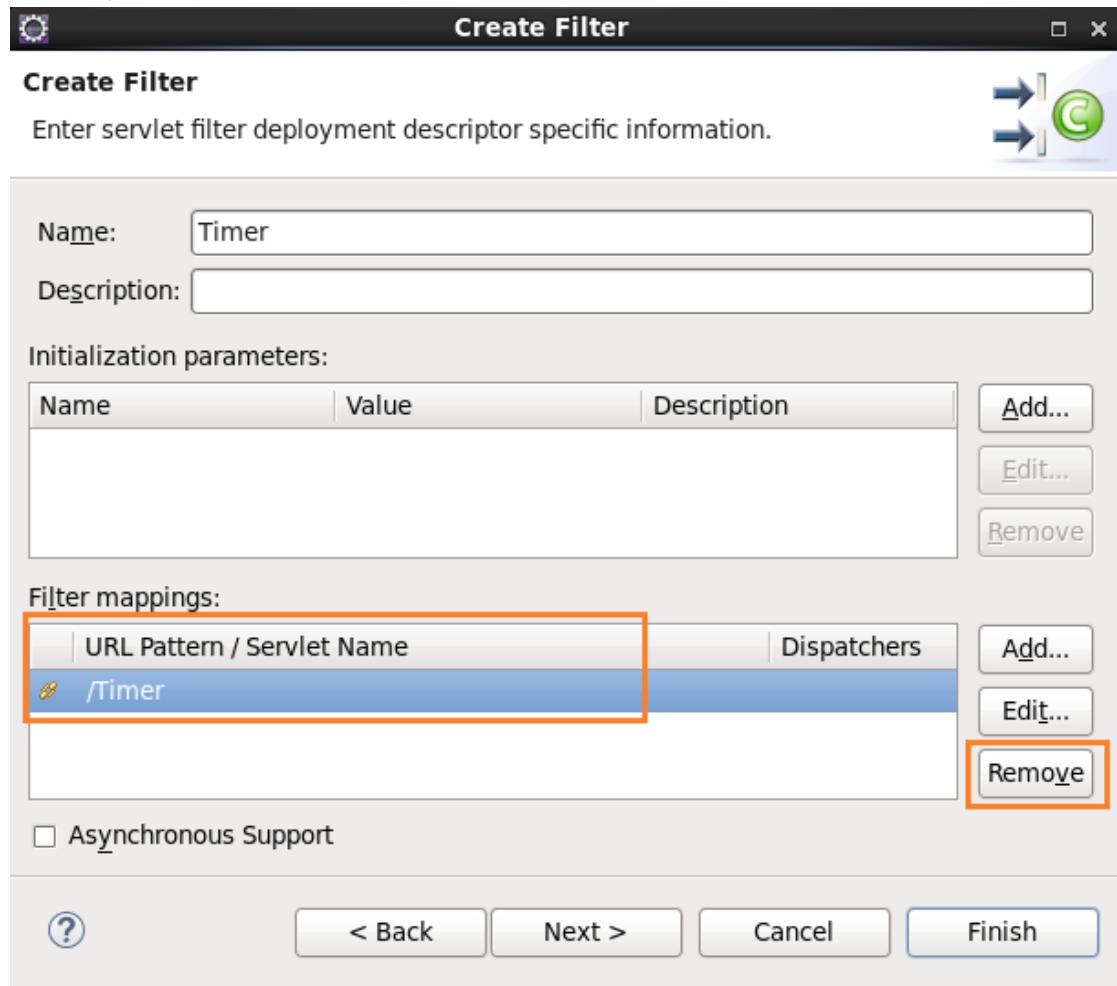
В этом разделе лабораторной работы создается фильтр, который будет использоваться для подсчета времени обработки запросов.

1. Создайте новый класс фильтра **Timer.java** в новом пакете **com.ibm.library.filters**. Настройте его таким образом, чтобы он попал в цепочку фильтров, используемых сервлетом **ProcessLogin**.
 - a. В представлении **Enterprise Explorer** нажмите правой кнопкой мыши по проекту **LibraryWebProject**. Выберите **New -> Filter**.
 - b. Укажите **com.ibm.library.filters** в поле **Java package** и **Timer** в поле **Class name**:



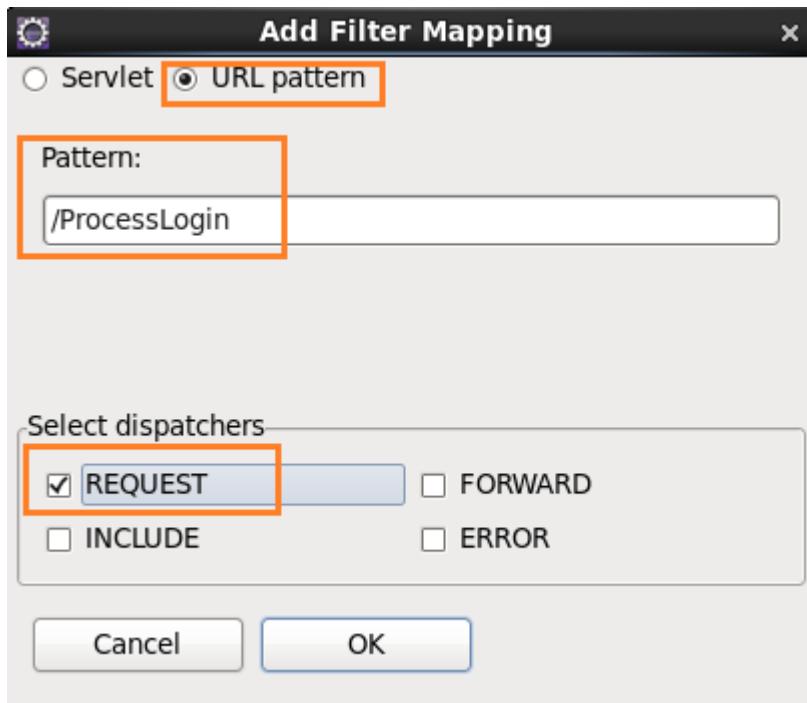
- c. Нажмите **Next**.

d. В интерфейсе **Filter Mappings** выберите пункт **/Timer** и нажмите кнопку **Remove**.



e. Нажмите кнопку **Add...**. Открывается диалог по созданию нового связывания для этого фильтра.

- f. Выберите опцию **URL pattern**, pattern – /ProcessLogin сервлет и опцию **REQUEST**. Нажмите **OK**.



- g. В мастере по созданию нового фильтра нажмите **Finish**. Если кнопка **Finish** не активна нажмите **Back**, а затем **Next**. Создается новый фильтр **Timer** в пакете **com.ibm.library.filters**. Этот класс открывается в редакторе.

Раздел 2. Добавление логики фильтра

В этом разделе лабораторной работы вы реализуете логику созданного фильтра.

1. В методе **init()** класса **Timer.java**, опишите сохранение объекта типа **FilterConfiguration** в переменной **config**, для того чтобы позже к ней можно было бы получить доступ.
 - a. Найдите метод **init()**.
 - b. Добавьте в конец метода следующую строку:

```
config = fConfig;
```

- c. Воспользуйтесь меню **Quick Fix** и выберите опцию **Create field 'config'**
2. В методе **doFilter()** добавьте логику для подсчета и фиксации времени, которое тратится на обработку запроса. Для этого нужно сохранять текущее время в миллисекундах до обработки запроса,

после – и считать разницу. Результат нужно записать в консоль с помощью метода **log()** объекта **ServletContext**.

а. Вы можете использовать следующий код для метода **doFilter()**:

```
long before = System.currentTimeMillis();  
  
chain.doFilter(request, response);  
  
long duration = System.currentTimeMillis() - before;  
String msg = config.getFilterName() + ": servlet  
duration was " + duration + " milliseconds";  
config.getServletContext().log(msg);
```

б. Сохраните сделанные изменения.

Раздел 3. Тестирование фильтра

В этом разделе лабораторной работы протестируйте работу созданного фильтра. Запишите время, которое выводится в консоли. Попробуйте провести этот тест несколько раз. Есть ли заметная разница между первым тестом и последующими? С чем это может быть связано?

Раздел 4. Создание Helper класса

В этом разделе лабораторной работы создается вспомогательный класс **StringResponse**, который будет использоваться во втором фильтре вместо стандартного **HttpServletResponse**. В этом классе можно переопределить некоторые стандартные методы. В данном случае в классе будет создан дополнительный атрибут типа **StringWriter**, который будет использоваться в фильтре для представления **response**, который возвращается сервлетом в виде объекта типа **String**.

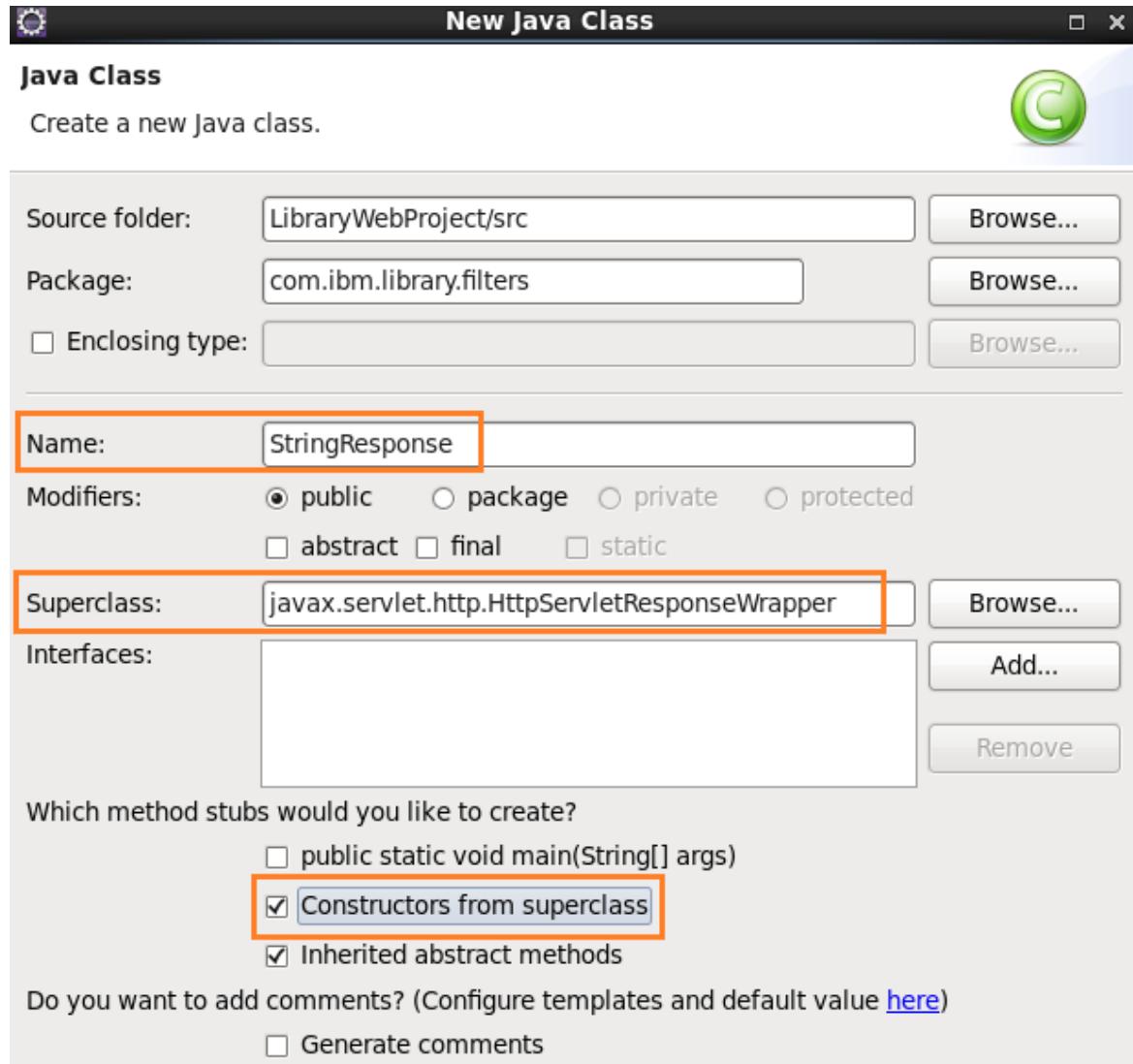
1. Создайте класс **StringResponse**, расширяющий **HttpServletResponseWrapper**.

а. Нажмите правой кнопкой мыши по пакету **LibraryWebProject -> Java Resources -> src -> com.ibm.library.filters** и выберите **New -> Class**.

б. Введите **StringResponse** как имя класса.

с. Рядом с полем **Superclass** нажмите кнопку **Browse** и выберите класс **HttpServletResponseWrapper**, начав вводить его название. Нажмите **OK**.

d. Выберите опцию **Constructors from superclass** и нажмите **Finish**.



2. Реализуйте новый класс.

a. Сделайте код класс следующим:

```
private StringWriter responseWriterBuffer;

public StringResponse(HttpServletResponse response)
{
    super(response);
    responseWriterBuffer = new StringWriter();
}

@Override
public String toString() {
    return responseWriterBuffer.toString();
}
```

```
@Override  
public PrintWriter getWriter() throws IOException {  
    return new PrintWriter(responseWriterBuffer);  
}
```

- b. Импортируйте необходимые классы.
- c. Сохраните сделанные изменения.

Раздел 5. Создание и настройка фильтра Appender

В этом разделе лабораторной работы создается еще один фильтр, который будет изменять содержимое страниц, возвращаемых сервером.

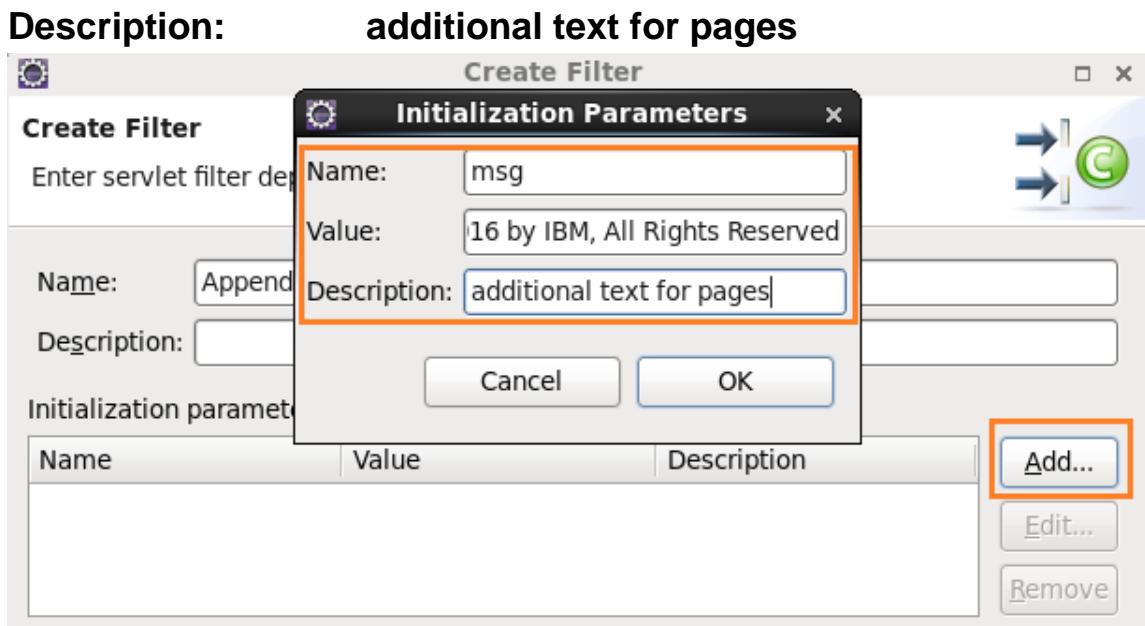
1. Создайте фильтр **Appender.java** в пакете **com.ibm.library.filters**.

Сделайте для него один единственный параметр инициализации – строку msg со следующим содержимым: “**Copyright 2016 IBM, All Rights Reserved**”. Сделайте так, чтобы эта строка добавлялась бы к любой странице, которая возвращается клиенту библиотечной системы.

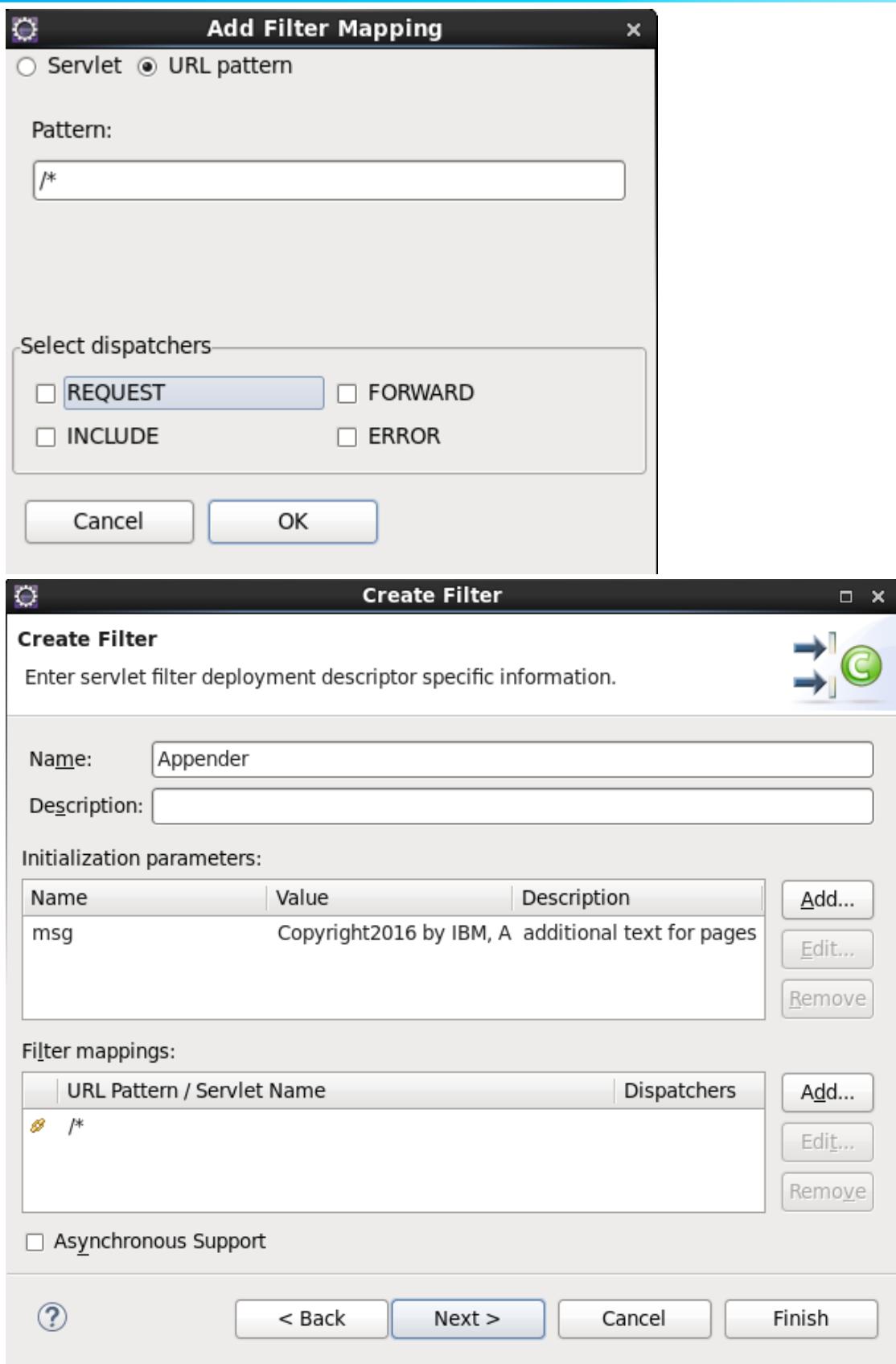
- a. Нажмите правой кнопкой мыши по пакету **LibraryWebProject -> Java Resources -> src -> com.ibm.library.filters**. Выберите **New -> Filter**.
- b. Укажите **Appender** в поле **Class name**. Нажмите **Next**.
- c. Нажмите кнопку **Add** в разделе **Initialization Parameters**.
- d. В открывшемся диалоге укажите следующие параметры:

Name: msg

Value: Copyright 2016 IBM, All Rights Reserved



- e. Нажмите **OK**.
- f. В разделе **Filter Mappings** выберите пункт **/Appender** и нажмите **Remove**.
- g. Нажмите кнопку **Add** в разделе **Filter Mappings**. Выберите опцию **URL pattern**, введите **/*** и нажмите **OK**.



h. Нажмите **Finish**. Класс **Appender** открывается для редактирования.

Раздел 6. Создание логики фильтра Appender

В этом разделе лабораторной работы добавляете код для созданного фильтра.

1. В методе **init()** получите доступ к параметру инициализации сервлета с помощью метода **getInitParameter()** и сохраните его в переменную **msg** типа **String**.

a. Найдите метод **init()** класса **Appender**.

b. Добавьте в код метода следующую строку:

```
msg = fConfig.getInitParameter("msg");
```

c. Используйте меню **Quick Fix** и выберите опцию **Create field 'msg'**.

d. Текущую логику метода **doFilter()** замените на следующую:

```
StringResponse strResp = new  
StringResponse((HttpServletResponse) response);  
  
chain.doFilter(request, strResp);  
  
PrintWriter out = response.getWriter();  
String responseString = strResp.toString();  
  
int endBodyIndex =  
responseString.indexOf("</body>");  
if (endBodyIndex > -1) {  
    StringBuffer finalResponse = new  
StringBuffer(responseString  
        .substring(0, endBodyIndex - 1));  
    finalResponse.append("<H4>");  
    finalResponse.append(msg);  
    finalResponse.append("</H4></BODY></HTML>");  
    String finalResponseString =  
finalResponse.toString();  
    out.write(finalResponseString);  
} else {  
    out.write(responseString);  
}
```

```
out.close();
```

- e. Импортируйте необходимые классы.
- f. Сохраните сделанные изменения.

Раздел 7. Тестирование фильтра

В этом разделе лабораторной работы протестируйте работу созданного фильтра. Убедитесь, что все страницы, которые есть в приложении, снабжены необходимой строкой внизу.

IBM Library System

Please Log In:

Patron ID

Password

Copyright 2016 By IBM, All Rights Reserved

Конец упражнения

Упражнение 12. Создание JSF модулей

О чём это упражнение:

В этой лабораторной работе вы реализуете заглушку JSF модуля, который можно использовать для входа в систему библиотеки. Вы создадите проект с необходимым функционалом, а также добавите две простые страницы с правилами навигации между ними.

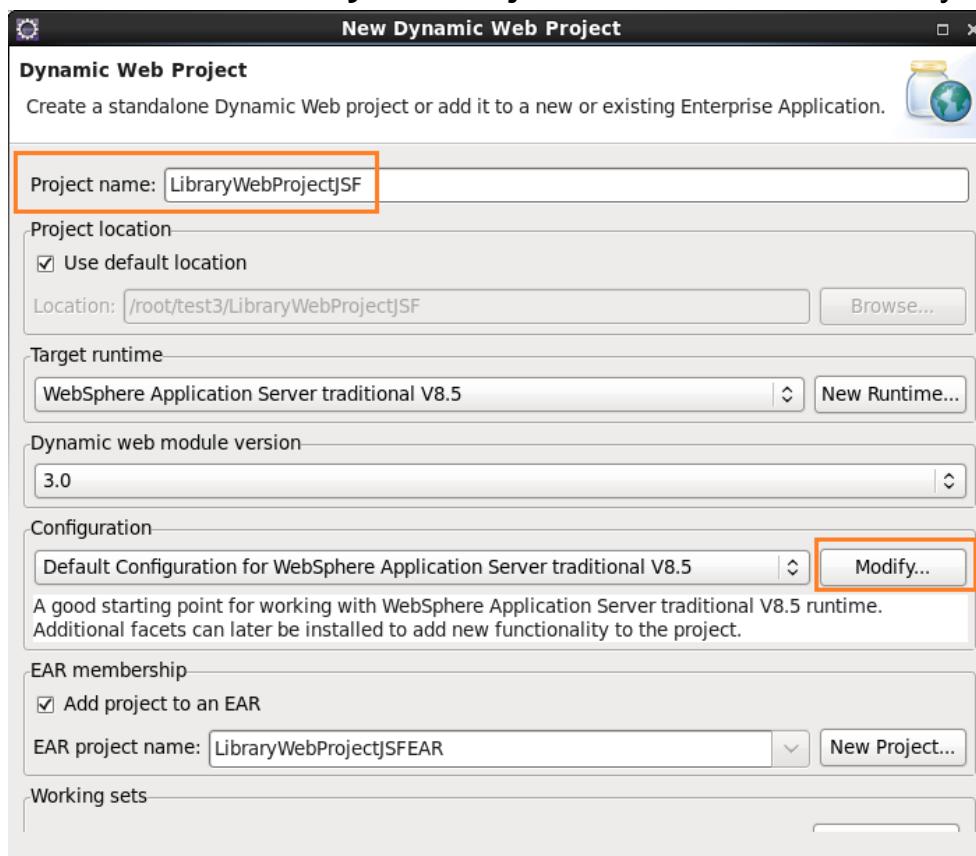
Что вы должны будете сделать:

- Создать JSP страницы с использованием JSF компонентов
- Привязать JavaBean к UI компонентам

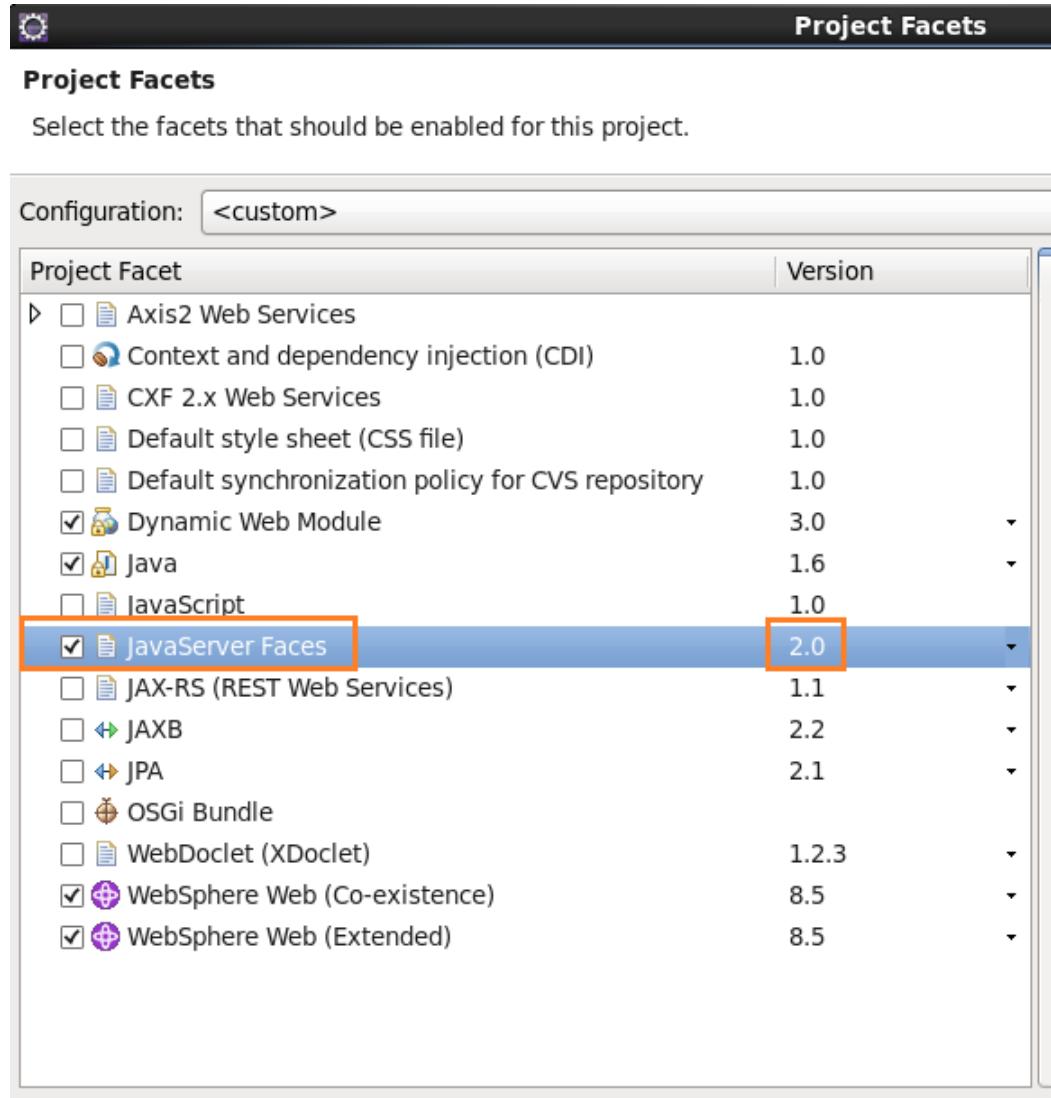
Раздел 1. Создание проектов

В этом разделе лабораторной работы создается динамический веб проект, готовый для работы с JSF, и EAR проект, с помощью которого этот модуль будет разворачиваться на сервере.

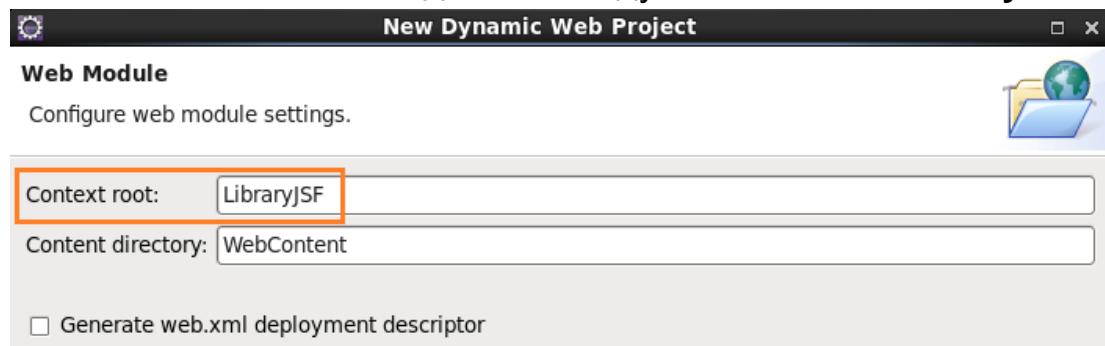
1. Создайте проект типа **Dynamic Web Project** с именем **LibraryWebProjectJSF** с **Context root – /LibraryJSF** и поддержкой технологии JSF.
 - a. В меню выберите **File -> New -> Dynamic Web Project**.
 - b. Укажите имя **LibraryWebProjectJSF** и нажмите кнопку **Modify**.



c. Выберите функцию **JavaServer Faces** и установите версию **2.0**.

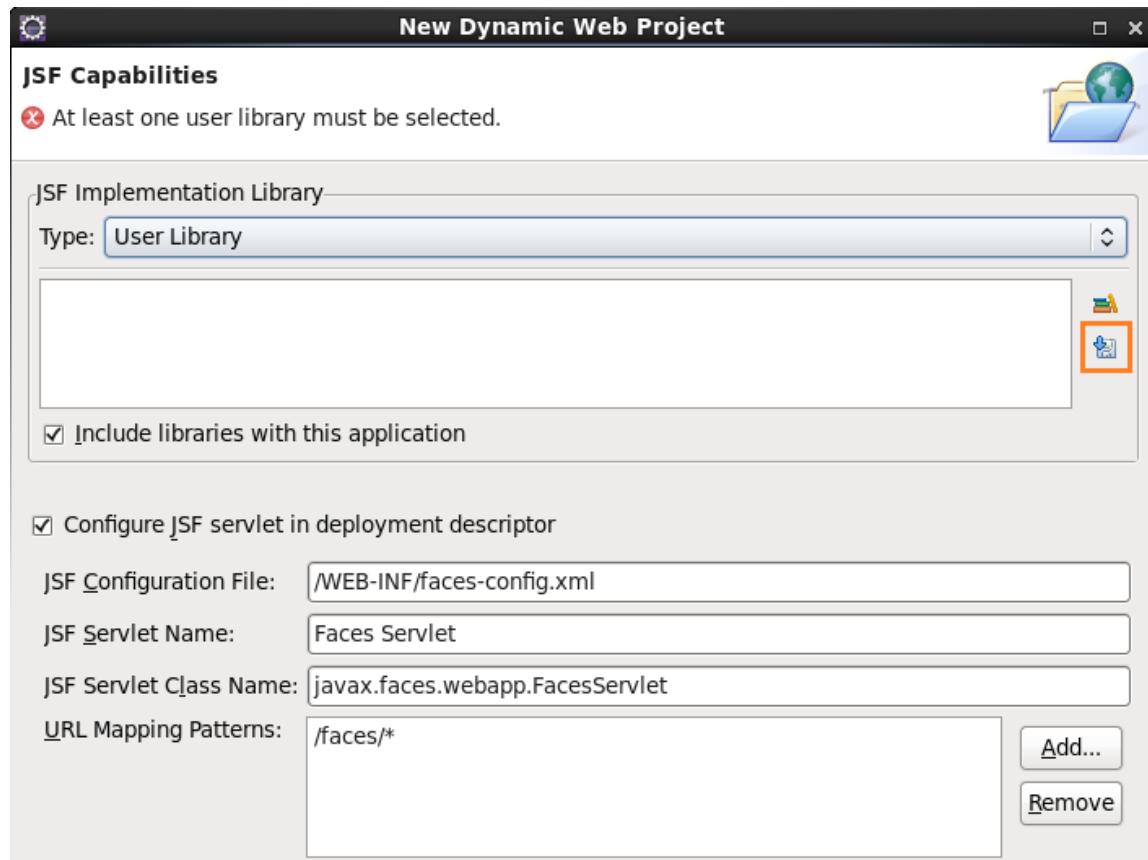


- d. Убедитесь, что в поле **EAR project name**, указано **LibraryWebProjectJSFEAR**. Нажмите **OK**.
e. Нажмите **Next** 2 раза.
f. Измените **Context root** для веб модулей. Укажите **LibraryJSF**.

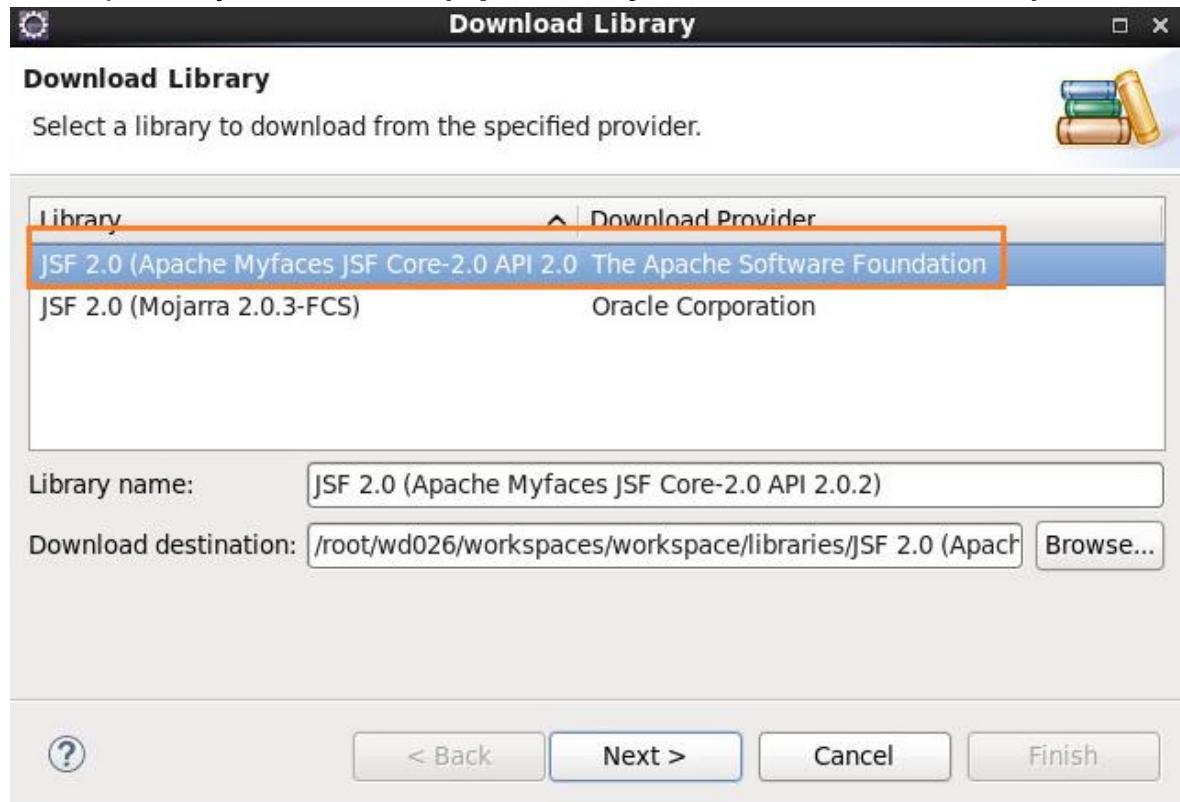


- g. Нажмите **Next**.

h. В следующем интерфейсе нажмите по кнопке для загрузки библиотек JSF.



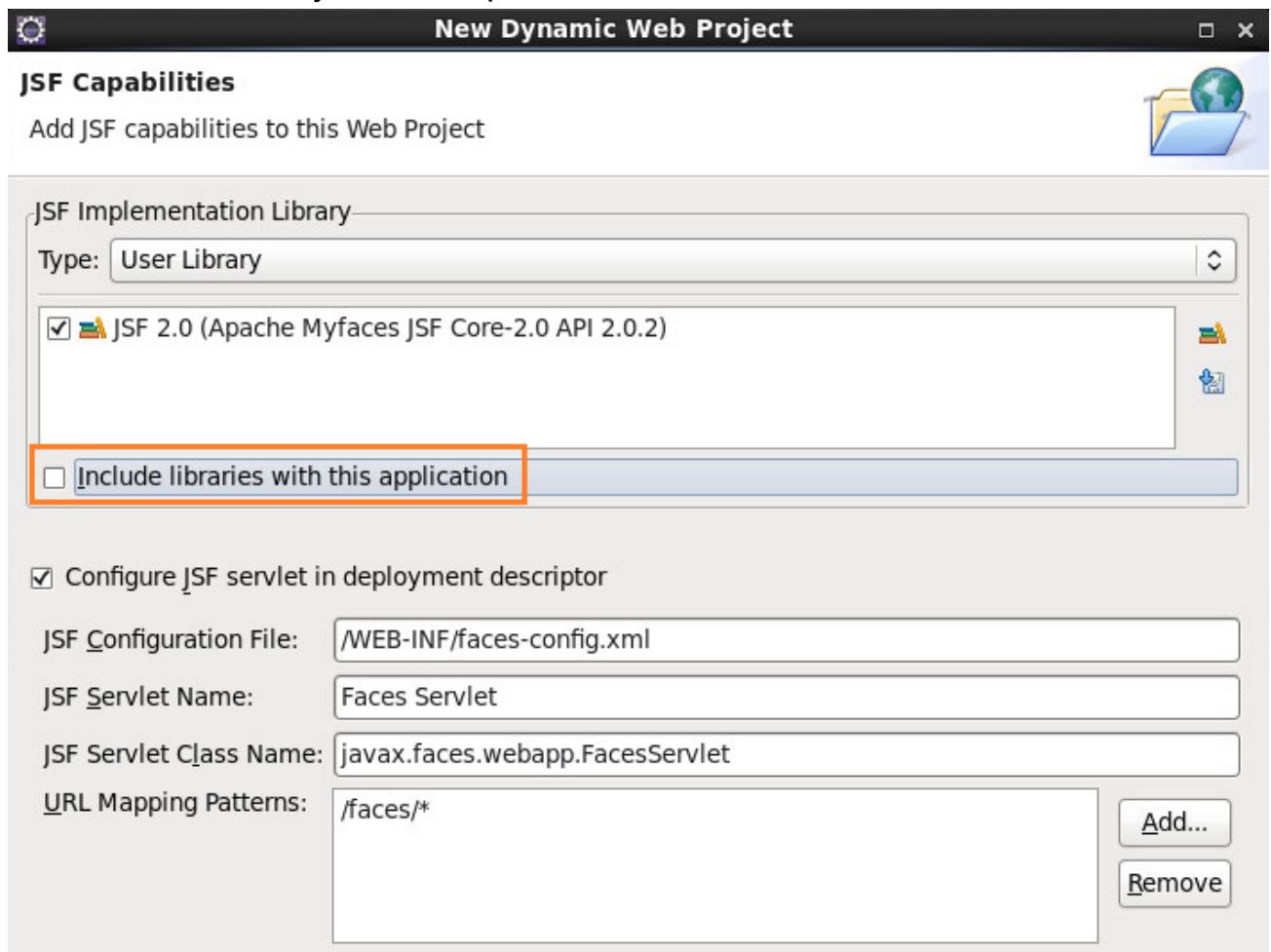
i. Выберите пункт **JSF 2.0 (Apache Myfaces JSF Core-2.0 ...)**



j. Нажмите **Next**.

k. Установите опцию **I accept the terms of this license** и нажмите **Finish**.

l. После завершения загрузки и установки необходимой библиотеки, **отключите** опцию **Include libraries with this application**. Нажмите кнопку **Finish** в мастере создания проекта. Согласитесь с предложением переключиться в перспективу **Web**. После создания веб модуля EAR проект создается автоматически.

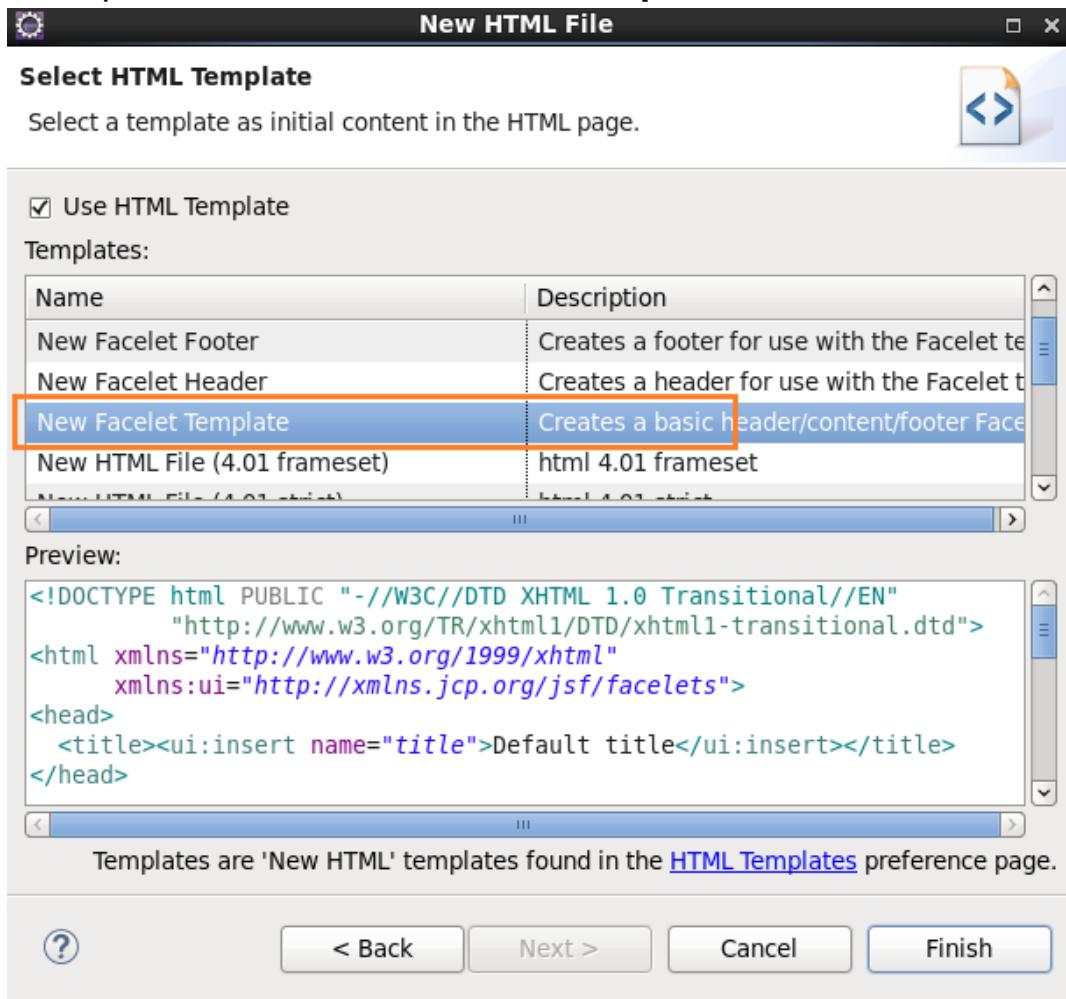


Раздел 2. Создание шаблонов и страниц для JSF проекта

В этом разделе лабораторной работы создается шаблон для новых веб-страниц, которые будут появляться в проекте. Далее с помощью этого шаблона создаются две страницы, которые затем будут использованы в приложении.

1. Создайте новый каталог для шаблонов страниц.
 - a. Нажмите правой кнопкой мыши по каталогу **LibraryWebProjectJSF -> WebContent -> WEB-INF** и выберите **New -> Folder**.
 - b. Укажите имя **templates** и нажмите **Finish**.

2. Создайте в только что добавленном каталоге базовый шаблон для страницы и шаблоны для ее верхней и нижней частей.
- Нажмите правой кнопкой мыши по каталогу **templates** и выберите **New -> HTML File**.
 - Укажите имя **BasicTemplate.xhtml** и нажмите **Next**.
 - Выберите шаблон **New Facelet Template**. Нажмите **Finish**.



- Нажмите правой кнопкой мыши по каталогу **templates** и выберите **New -> HTML File**.
 - Укажите имя **header.xhtml** и нажмите **Next**.
 - Выберите шаблон **New Facelet Header**. Нажмите **Finish**.
 - Нажмите правой кнопкой мыши по каталогу **templates** и выберите **New -> HTML File**.
 - Укажите имя **footer.xhtml** и нажмите **Next**.
 - Выберите шаблон **New Facelet Footer**. Нажмите **Finish**.
3. Измените созданные файлы шаблонов для обеспечения стиля приложения **IBM Library System**.
- Откройте файл **BasicTemplate.xhtml** и перейдите на вкладку **Source**.

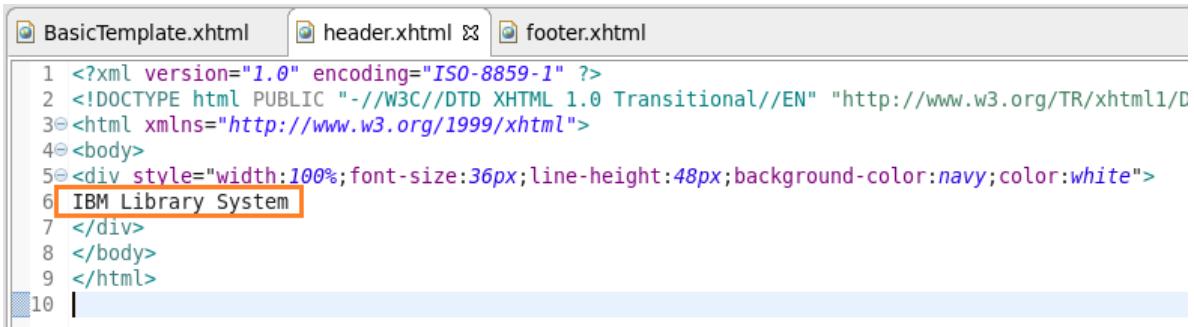
b. Замените текущий код разметки страницы следующим:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0  
Transitional//EN"  
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-  
transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml"  
    xmlns:ui="http://java.sun.com/jsf/facelets">  
<head>  
    <title><ui:insert name="title">IBM Library  
System</ui:insert></title>  
    <link href="/theme/Master.css" rel="stylesheet"  
type="text/css" />  
</head>  
  
<body>  
  
<div id="header">  
    <ui:insert name="header">  
        <ui:include src="/WEB-  
INF/templates/header.xhtml"/>  
    </ui:insert>  
</div>  
  
<div id="content">  
    <ui:insert name="content">  
    </ui:insert>  
</div>  
  
<div id="footer">  
    <ui:insert name="footer">  
        <ui:include src="/WEB-  
INF/templates/footer.xhtml"/>  
    </ui:insert>  
</div>  
</body>  
</html>
```

Отличие от изначальной версии состоит в том, что вместо

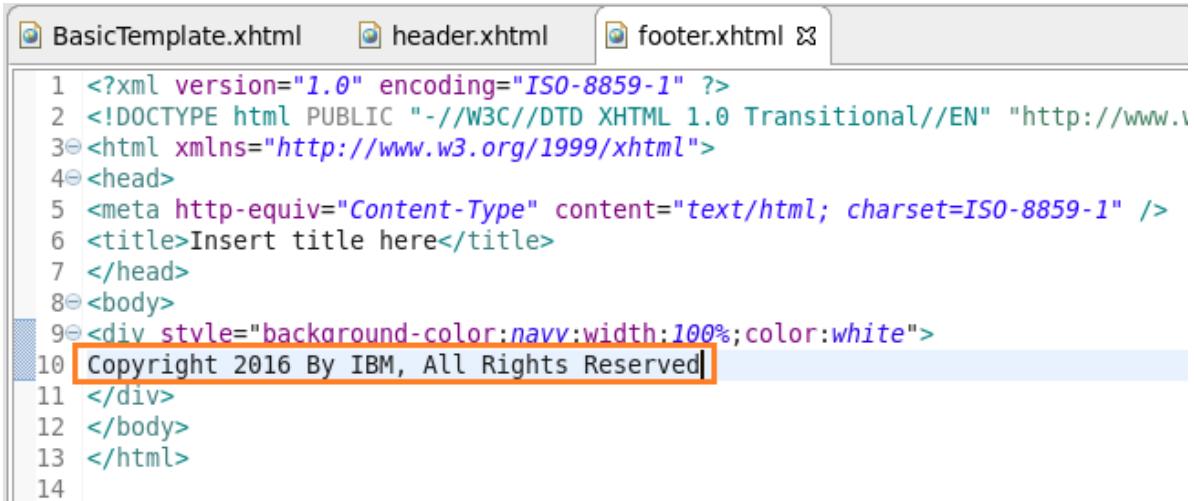
предопределенных констант на месте **header** и **footer** теперь даются ссылки на другие части шаблона, которые были созданы ранее. Также меняется адрес в одном из пространств имен.

- c. Откройте файл **header.xhtml** и перейдите на вкладку **Source**.
- d. Замените текущий заголовок на **IBM Library System**



```
1 <?xml version="1.0" encoding="ISO-8859-1" ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/D
3<html xmlns="http://www.w3.org/1999/xhtml">
4<body>
5<div style="width:100%;font-size:36px;line-height:48px;background-color:navy;color:white">
6 IBM Library System
7 </div>
8 </body>
9 </html>
10 |
```

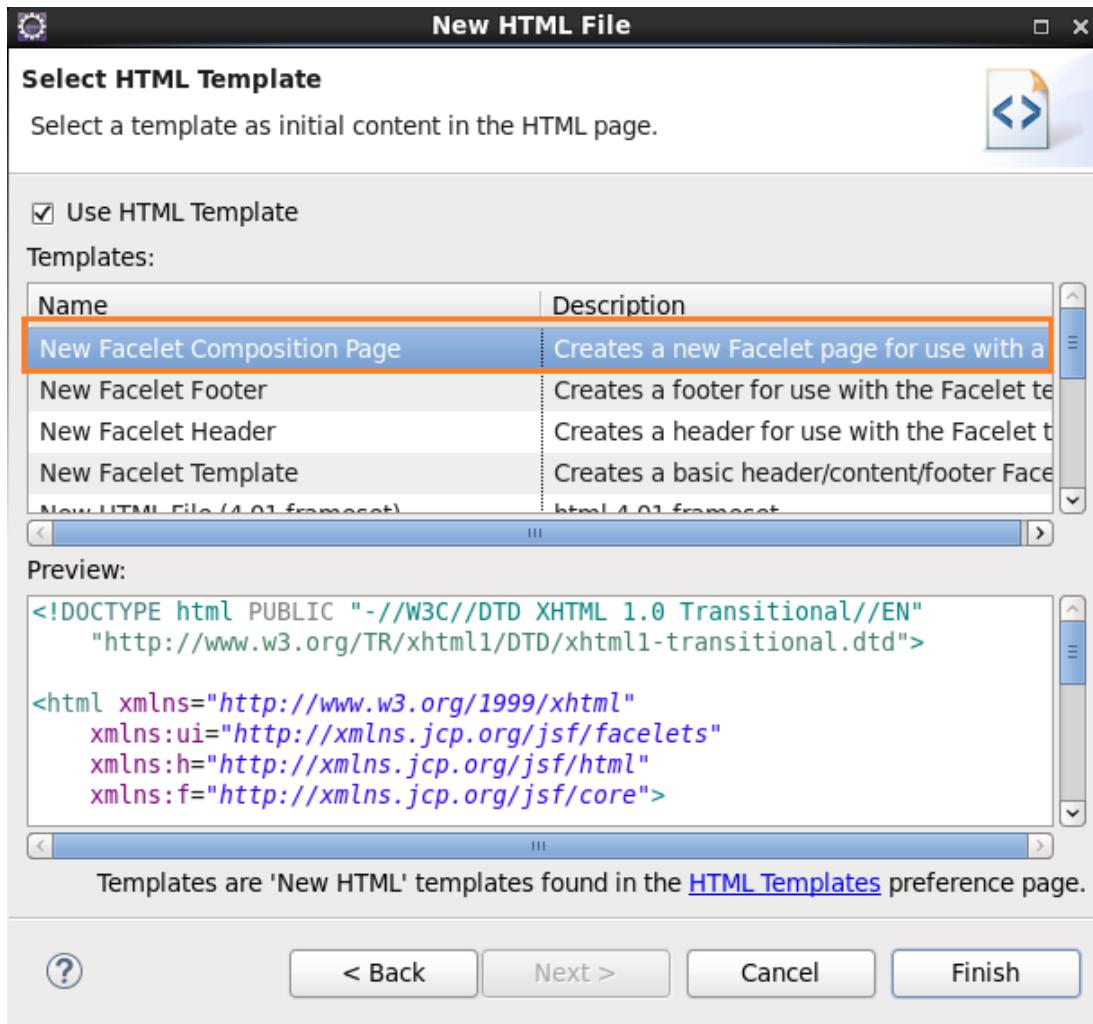
- e. Откройте файл **footer.xhtml** и перейдите на вкладку **Source**.
- f. Добавьте в секцию **<div>** строчку **Copyright 2016 By IBM, All Rights Reserved**



```
1 <?xml version="1.0" encoding="ISO-8859-1" ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/D
3<html xmlns="http://www.w3.org/1999/xhtml">
4<head>
5 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
6 <title>Insert title here</title>
7 </head>
8<body>
9<div style="background-color:navy;width:100%;color:white">
10 Copyright 2016 By IBM, All Rights Reserved
11 </div>
12 </body>
13 </html>
14 |
```

- g. Сохраните сделанные изменения во всех файлах.
4. Создайте страницу **login.xhtml** для отображения формы входа в систему.
 - a. Нажмите правой кнопкой мыши по каталогу **LibraryWebProjectJSF** -> **WebContent** и выберите **New -> HTML File**.
 - b. Укажите имя **login.xhtml** и нажмите **Next**.

с. Выберите шаблон **New Facelet Composition Page** и нажмите **Finish**.



д. Перейдите на вкладку **Source**.

е. Обратите внимание на строчку `<ui:composition template="">` – здесь предполагается указать выбранный шаблон для страницы. Дополните эту строчку:

```
<ui:composition template="/WEB-
INF/templates/BasicTemplate.xhtml">
```

ф. Все вложенные теги для `<ui:composition>` замените на следующие строки:

```
<ui:define name="content">
    <h:form>
        <h:panelGrid columns="2">
            <h:outputText
                value="Name"></h:outputText>
```

```
<h:inputText  
value="#{patronBean.name}"></h:inputText>  
    <h:outputText  
value="Password"></h:outputText>  
        <h:inputSecret  
value="#{patronBean.password}"></h:inputSecret>  
    </h:panelGrid>  
    <h:commandButton value="Login"  
action="login"></h:commandButton>  
</h:form>  
</ui:define>
```

На данную страницу добавляются стандартные JSF элементы для вывода текста, поля ввода, кнопка, которая будет использована для перехода к следующей части интерфейса. В полях для ввода есть ссылки на поля для JavaBean, который будет определен позже.

- g. Замените в ссылках, указывающих на namespace этого документа, адреса `xmlns:jcp.org` на адреса **java.sun.com**.

```
><html xmlns="http://www.w3.org/1999/xhtml"  
      xmlns:ui="http://java.sun.com/jsf/facelets"  
      xmlns:h="http://java.sun.com/jsf/html"  
      xmlns:f="http://java.sun.com/jsf/core">
```

- h. Сохраните сделанные изменения.

5. Создайте страницу **welcome.xhtml** для отображения результата аутентификации (по факту аутентификации не будет, будет реализован лишь переход от одной страницы на другую, без анализа введенных данных).

- a. Нажмите правой кнопкой мыши по каталогу **LibraryWebProjectJSF** -> **WebContent** -> **WEB-INF** и выберите **New -> HTML File**.
b. Укажите имя **welcome.xhtml** и нажмите **Next**.
c. Перейдите на вкладку **Source**.
d. Обратите внимание на строчку `<ui:composition`
`template="">` – здесь предполагается указать выбранный шаблон для страницы. Дополните эту строчку:

```
<ui:composition template="/WEB-  
INF/templates/BasicTemplate.xhtml">
```

- e. Все вложенные теги для **<ui:composition>** замените на следующие строки:

```
<ui:define name="content">
    <h:outputLabel value="Welcome
#{patronBean.name}"></h:outputLabel>
</ui:define>
```

- f. Замените в ссылках, указывающих на namespace этого документа, адреса xmlns.jcp.org на адреса **java.sun.com**.

```
><html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
```

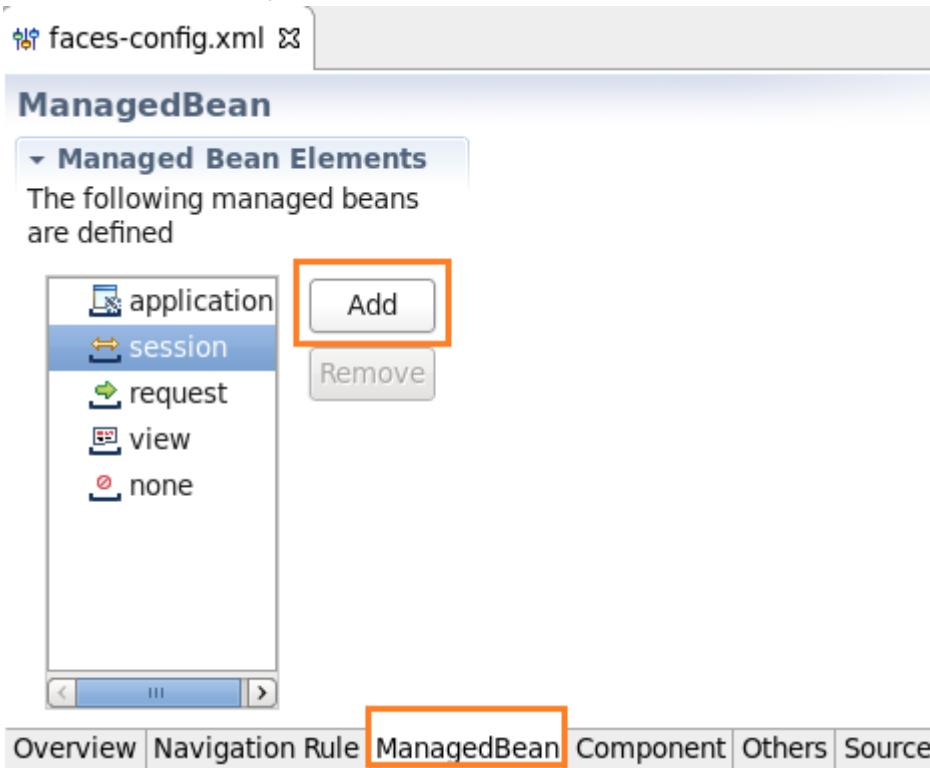
- g. Сохраните сделанные изменения.

Раздел 3. Добавление JavaBean и правила навигации

В этом разделе лабораторной работы создается JavaBean, который используется на страницах, а также добавляется правило навигации, позволяющее переходить от страницы **login** к странице **welcome**.

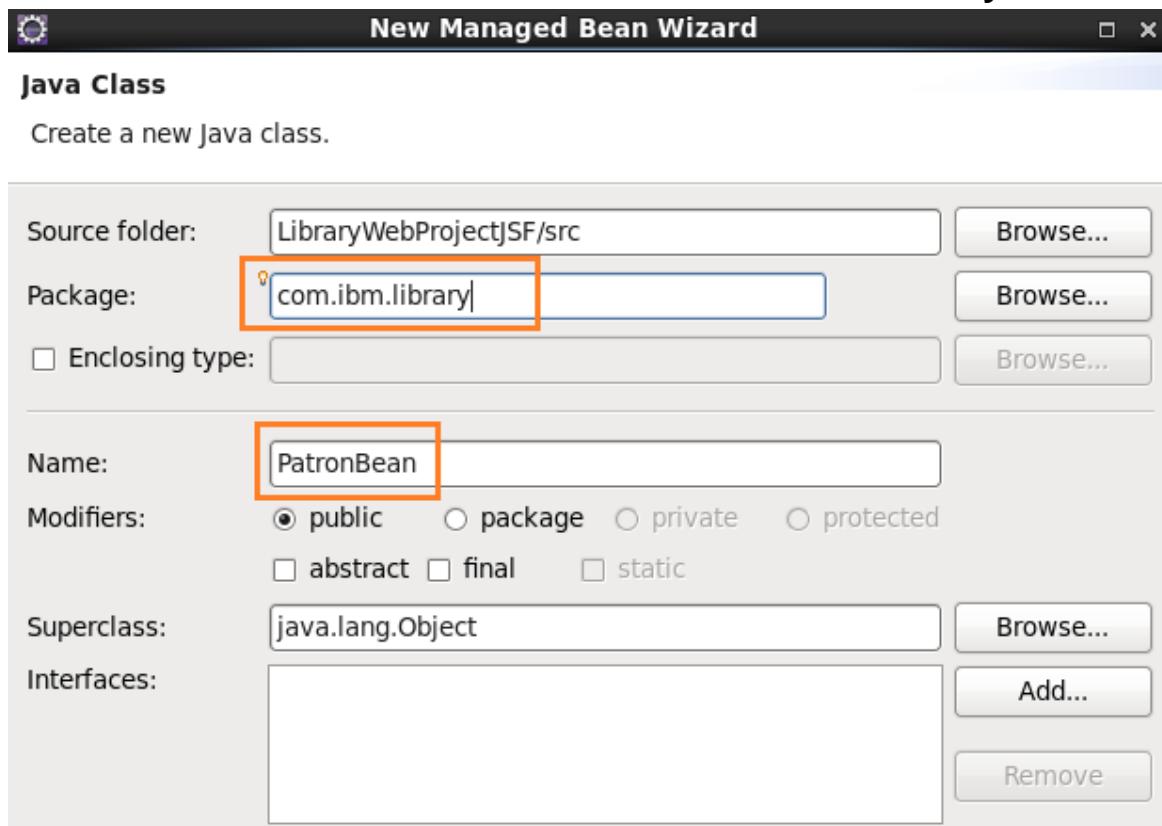
1. Насторойте **ManagedBean** для работы с учетной записью пользователя.
 - a. Откройте файл **LibraryWebProjectJSF -> WebContent -> WEB-INF -> faces-config.xml**. Он открывается в специальном редакторе для работы с конфигурационными файлами JSF.
 - b. Перейдите на вкладку **ManagedBean**.

с. Нажмите кнопку **Add**.

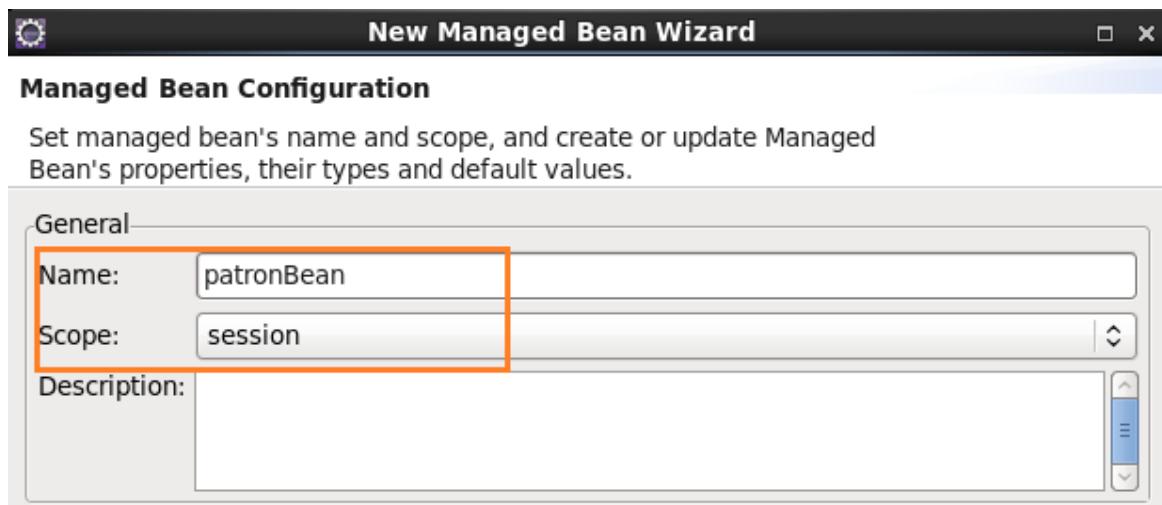


d. Выберите опцию **Create a new Java class**. Нажмите **Next**.

e. Укажите имя класса **PatronBean** и пакет **com.ibm.library**.

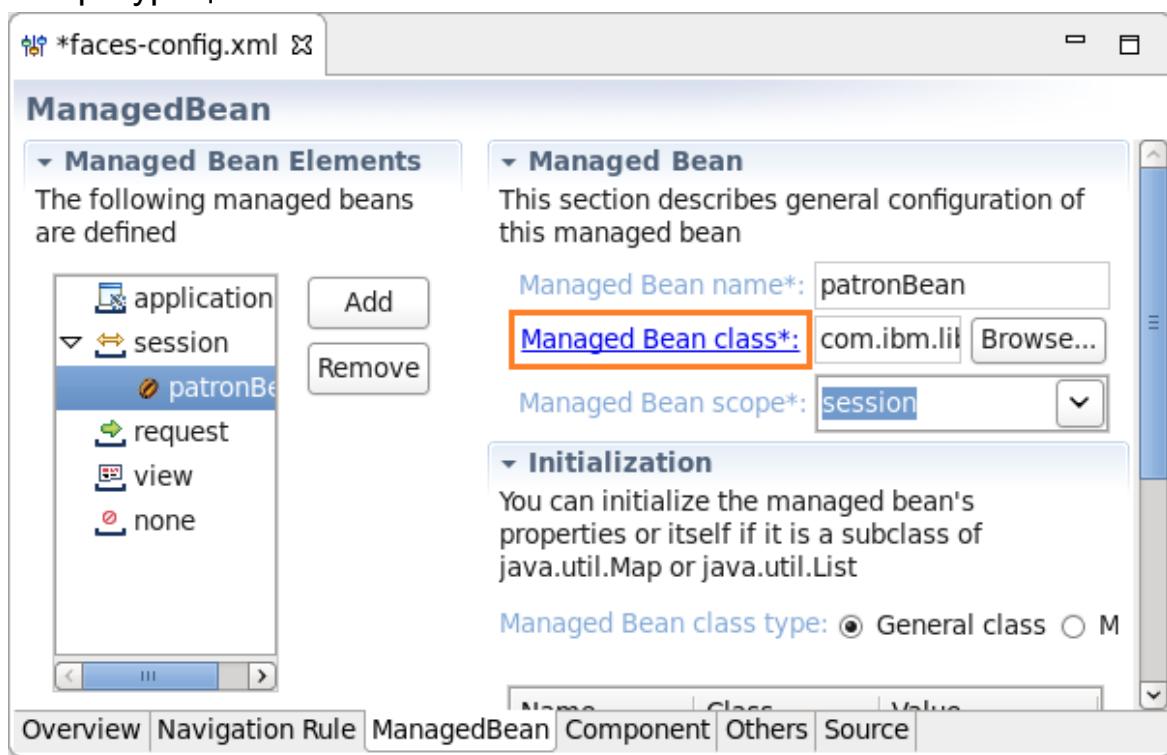


- f. Нажмите **Next**. Убедитесь, что выбрано имя **patronBean** и scope – **session**.



- g. Нажмите **Finish**.

- h. Откройте только что созданный класс. Для этого проще всего кликнуть по ссылке **Managed Bean class** в редакторе JSF конфигурации.

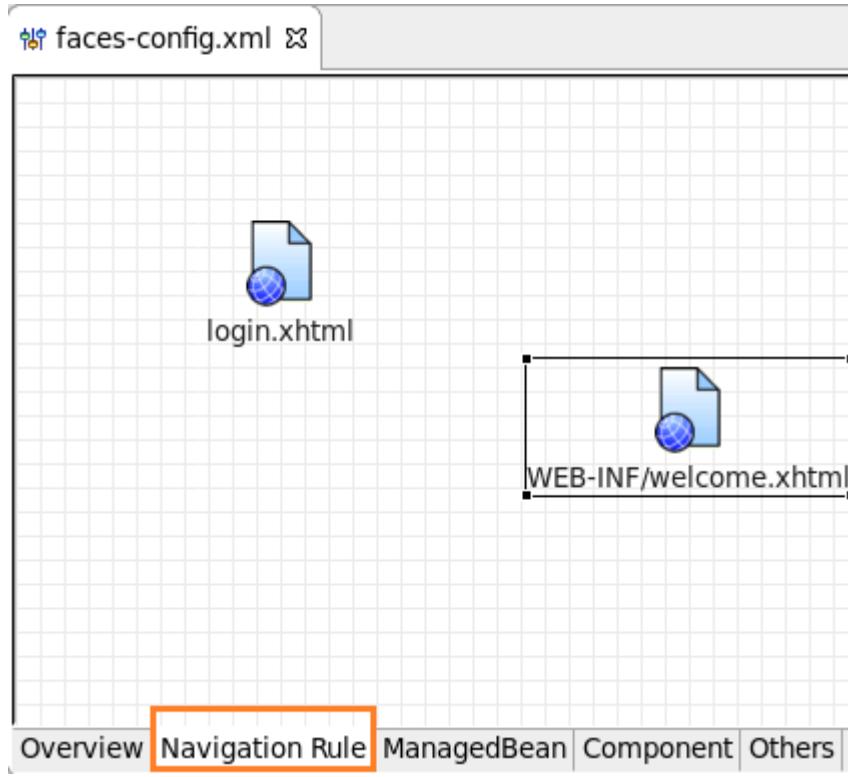


- i. Добавьте в код класса атрибуты **name** и **password** и реализуйте методы getter и setter для каждого из них.

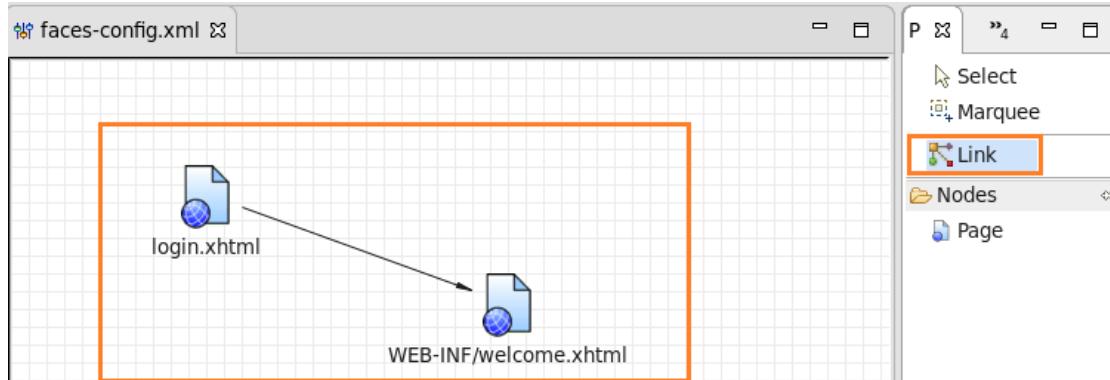
```
1 package com.ibm.library;
2
3 public class PatronBean {
4     private String name;
5     private String password;
6
7     public String getName (){
8         return name;
9     }
10
11    public void setName (final String name){
12        this.name = name;
13    }
14
15    public String getPassword (){
16        return password;
17    }
18
19    public void setPassword (final String password){
20        this.password = password;
21    }
22 }
23
```

- j. Сохраните сделанные изменения в классе **PatronBean**.
k. Сохраните изменения в файле конфигурации JSF.
2. Добавьте правило навигации, по которому будет осуществляться безусловный переход со страницы **login** на страницу **welcome** при нажатии на кнопку **Login**.
- Откройте файл **LibraryWebProjectJSF -> WebContent -> WEB-INF -> faces-config.xml**. Он открывается в специальном редакторе для работы с конфигурационными файлами JSF.
 - Перейдите на вкладку **Navigation Rule**.
 - Методом **Drag and Drop** перенесите в редактор страницы **login.xhtml** и **welcome.xhtml** из представления **Enterprise**

Explorer.

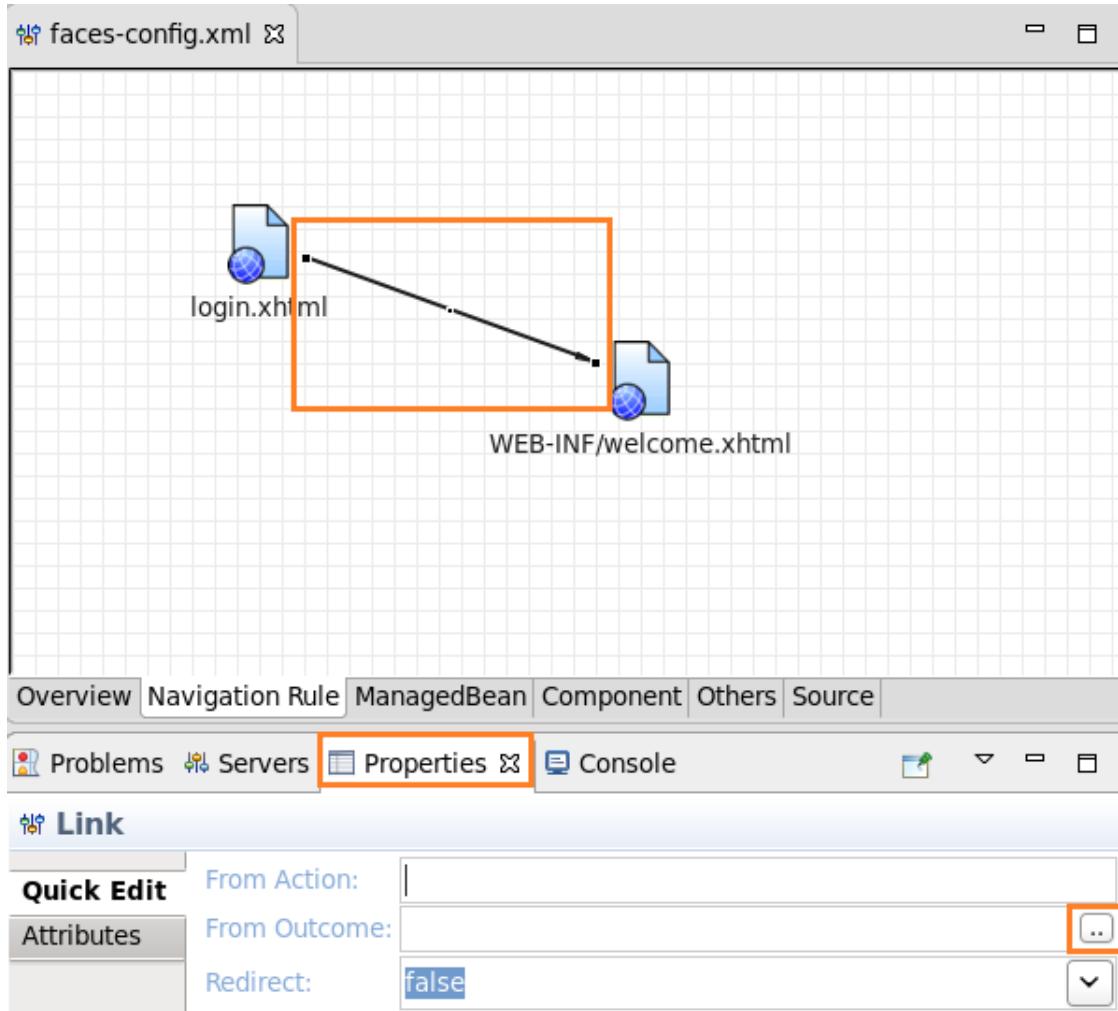


- d. Откройте представление **Palette**.
- e. Выберите элемент **Link** и нарисуйте с помощью него связь из страницы **login.xhtml** к странице **welcome.xhtml**.

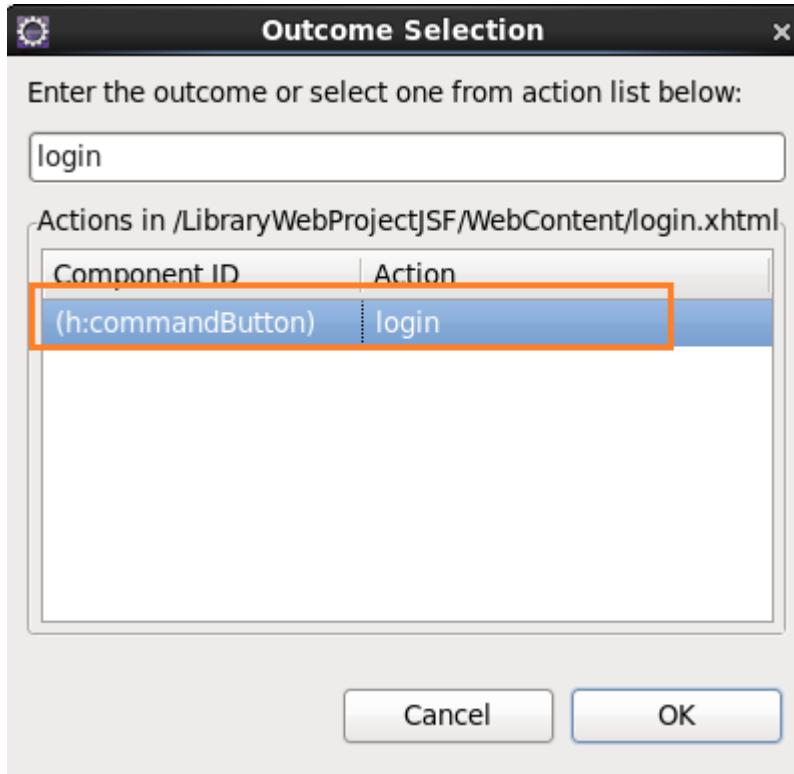


- f. В представлении **Palette** выберите элемент **Select**.
- g. Выберите связь, нарисованную между двумя страницами, и откройте представление **Properties**.

h. Нажмите по кнопке рядом с пунктом **From Outcome**.



i. Выберите пункт (**h:commandButton**) login. Нажмите OK.

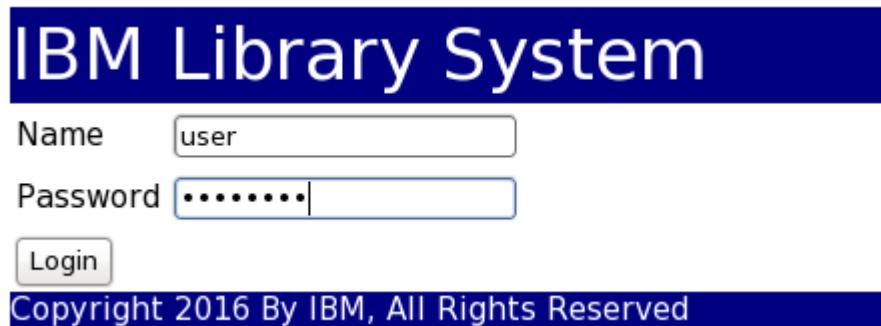


- j. Сохраните сделанные изменения в файле **faces-config.xml** и закройте его.

Раздел 4. Тестирование разработанного функционала

В этом разделе лабораторной работы вы проверяете работу созданного модуля.

1. Установите приложение **LibraryWebProjectJSF** на сервер приложений.
 - a. В представлении **Servers** нажмите правой кнопкой мыши по **WebSphere Application Server...**
 - b. Выберите **Add and Remove...**
 - c. Нажмите **Add All** и **Finish**.
2. Проверьте работоспособность страниц и переход между ними.
 - a. В браузере перейдите по ссылке
http://localhost:9080/LibraryJSF/faces/login.xhtml
 - b. Убедитесь, что вы можете открыть первую страницу и перейти на вторую, введя произвольные данные.



IBM Library System

Name

Password

Copyright 2016 By IBM, All Rights Reserved



IBM Library System

Welcome user

Copyright 2016 By IBM, All Rights Reserved

Конец упражнения