Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Факультет дистанционного обучения (ФДО)

Ю. В. Морозова

РАЗРАБОТКА ИНТЕРНЕТ-ПРИЛОЖЕНИЙ

Методические указания
по выполнению лабораторной работы
и организации самостоятельной работы для студентов,
обучающихся с применением ДОТ

Томск 2020

Корректор А. Н. Миронова

Морозова Ю. В.

Разработка интернет-приложений : методические указания по выполнению лабораторной работы и организации самостоятельной работы для студентов, обучающихся с применением ДОТ / Ю. В. Морозова. – Томск : ФДО, ТУСУР, 2020. – 60 с.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящее время Интернет является неотъемлемой частью жизни людей, без которой существование уже не представляется возможным. Для взаимодействия через Интернет нужны специальные приложения, то есть интернет-приложения. Интернет-приложения можно встретить на любом современном устройстве: телефоне, планшете, компьютере. Чаще всего это браузеры, чаты, облачные хранилища, игры – таких приложений безграничное множество. Но у всех интернет-приложений есть и другая сторона – серверная, которая и выполняет функцию хранения и обработки информации [1].

При разработке интернет-приложений можно использовать одну из наиболее популярных технологий, к которым относятся Java Servlets, Java Server Page (JSP), PHP, ASP.NET, Node.js. Различные платформы для создания приложений используют два основных подхода к разработке серверной части:

1. Формирование кода в виде текста определенного формата.
2. Встраивание кода в определенные шаблоны.

Первый подход предоставляет наибольшие возможности по повышению производительности. Он предусматривает передачу всех данных о запросе непосредственно серверу, который может как сформировать ответ со страницей для пользователя, так и открыть на передачу поток двоичных данных, например для передачи изображения. Однако при таком подходе все данные для передачи формируются программным путем, что замедляет разработку простых страниц. Примерами данного подхода являются технологии CGI, Java Servlets.

Второй подход использует оформленные особым образом шаблоны страниц, что позволяет вставлять в них участки кода. Этот подход особенно эффективен при создании web-приложений, основная информация в которых статична, а динамическая информация может быть сгенерирована простыми

программными конструкциями. При разработке больших систем этот вариант усложняет взаимодействие между компонентами и затрудняет реализацию сложной архитектуры. Также он менее эффективен по производительности и ограничивает возможности по реализации сложных страниц. Примерами данного подхода являются наиболее популярные на данный момент технологии PHP, ASP, JSP.

Технология **Java Servlets** (сервлеты) была разработана компанией Sun Microsystems, чтобы использовать преимущества платформы Java для решения проблем производительности технологии CGI и API. Технология решает эту проблему, выполняя все запросы как потоки в одном процессе. Сервлеты не зависят от платформы, поскольку выполняются внутри Java Virtual Machine (JVM) и могут легко разделять ресурсы.

Модель безопасности Java обеспечивает управление уровнем доступа, а обработка исключений делает сервлеты более надежным средством. Технология обладает широкими функциональными возможностями, большое количество библиотек предоставляет самые разнообразные средства, необходимые в разработке.

Технология сервлетов является распространенной и может быть использована со всеми популярными web-серверами, выполняющими функции контейнера сервлетов (Apache Tomcat, Java Web Server от Sun).

Программный интерфейс позволяет сервлетам обрабатывать запросы на низком уровне (заголовки запросов, их тип, и т. д.). Это обеспечивает большую гибкость при разработке нестандартных обработчиков, например при работе с двоичным или мультимедийным содержимым.

В связи с тем что сервлеты обрабатываются в одном процессе с помощью создания внутри него потоков, программный код сервлетов должен быть потокобезопасным. Это накладывает определенную ответственность на программиста. При этом сервлеты приобретают такое неоценимое преимущество, как масштабируемость.

Таким образом, сервлеты обеспечивают компонентный, платформонезависимый метод построения интернет-приложений без ограничений производительности. Они имеют широкий диапазон доступных прикладных API, позволяют использовать все преимущества Java, легко расширяются и масштабируются, поддерживаются всеми популярными web-серверами. Все это обеспечивает использование данной технологии для разработки крупных web-систем.

Цель дисциплины «Разработка интернет-приложений»:

- ознакомление с основными технологиями, необходимыми для создания интернет-приложений;
- получение практических навыков на основе выполнения лабораторной работы.

1 МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ «ОСНОВЫ ТЕХНОЛОГИИ СЕРВЛЕТОВ»

Целью данной лабораторной работы является ознакомление со средой разработки Eclipse и написание простейшего серверного приложения на языке программирования Java.

1.1 Теоретическая часть

Servlet API

Сервлеты – это компоненты приложений Java 2 Platform Enterprise Edition (J2EE), выполняющиеся на стороне сервера, способные обрабатывать клиентские запросы и динамически генерировать ответы на них [2]. Сервлет может применяться, например, для создания серверного приложения, получающего от клиента запрос, анализирующего его и делающего выборку данных из базы данных, а также пересылающего клиенту страницу HTML, сгенерированную с помощью JSP на основе полученных данных.

Клиент – это программа, которая запрашивает сервер выполнить то или иное действие (задачу) и вернуть полученные данные клиенту. Клиент подключается к порту, выделенному на сервере.

Сервер – это специальная программа, обычно запущенная на отдельном компьютере, которая ожидает от клиентских программ запросы и предоставляет им свои ресурсы в виде данных или сервисных функций. На сервере для работы нужно выделить порт [1].

Все сервлеты реализуют общий интерфейс Servlet. Для обработки HTTP-запросов можно воспользоваться в качестве базового класса абстрактным классом `HttpServlet`. Базовая часть классов JSDK помещена в пакет *javax.Servlet*. Однако класс `HttpServlet` и все, что с ним связано, располагаются на один уровень ниже в пакете *javax.Servlet.http*.

Пакет *javax.Servlet* содержит ряд интерфейсов и классов, устанавливающих обрамление, в котором работают сервлеты (табл. 1.1).

Таблица 1.1 – Пакет *javax.Servlet*

Интерфейс	**Описание**
`Servlet`	Объявляет методы цикла жизни для сервлета
`ServletConfig`	Позволяет сервлетам получать параметры инициализации
`ServletContext`	Активизирует возможности сервлетов для регистрации событий и доступа к информации об их среде
`ServletRequest`	Используется для чтения данных из запроса клиента
`ServletResponse`	Используется для записи данных в ответ клиенту
`SingleThreadModel`	Указывает, что сервлет защищен от многопоточности

`ServletRequest` предоставляет клиентскую информацию о параметрах HTTP-запроса сервлету, т. е. обеспечивает данные, включая название параметра и значения, атрибуты и входной поток. Дополнительная информация о запросе доступна сервлету через методы, основные из которых приведены в таблице 1.2.

Таблица 1.2 – Методы интерфейса `HttpServletRequest`

`getAttribute ()`	Возвращает значение указанного атрибута этого запроса
`getContentLength ()`	Размер запроса, если известен

Окончание таблицы 1.2

`getContentType ()`	Возвращает тип MIME тела запроса
`getInputStream ()`	Возвращает `InputStream` для чтения двоичных данных из тела запроса
`GetParameterNames ()`	Возвращает массив строк с именами всех параметров
`getParameterValues ()`	Возвращает массив значений для указанного параметра
`getProtocol ()`	Возвращает протокол и версию для запроса как строку вида `<protocol>/<major version>.<minor version>`
`getReader ()`	Возвращает `BufferedReader` для получения текста из тела запроса
`getRealPath ()`	Возвращает реальный путь для указанного виртуального пути
`getRemoteAddr ()`	IP-адрес клиента, пославшего данный запрос
`getRemoteHost ()`	Имя хоста клиентской машины, пославшего данный запрос
`getScheme ()`	Возвращает схему, используемую в URL этого запроса (например, https, http, ftp и т. д.)
`getServerName ()`	Имя хоста сервера, принявшего данный запрос
`getServerPort ()`	Возвращает номер порта, используемого для приема этого запроса

При каждом обращении к сервлету методы `doGet()` и `doPost()` класса `HttpServlet` принимают объект, который реализует интерфейс `HttpServletResponse`. Web-сервер, исполняющий сервлет, создает объект `HttpServletResponse` и передает его методу `service()` сервлета (который в свою очередь передает его методу `doGet()` или `doPost()`). Интерфейс `HttpServletResponse` определяет ответ клиенту.

Имеется множество методов интерфейса `HttpServletResponse`, дающих возможность сервлету сформировать ответ клиенту. Некоторые из этих методов принадлежат интерфейсу `ServletResponse`, который расширяется интерфейсом `HttpServletResponse`. Ряд ключевых методов представлены в таблице 1.3.

Таблица 1.3 – Методы интерфейса `HttpServletResponse`

Метод	**Описание**
`void addCookie (Cookie cookie)`	Метод используется для добавления Cookie в заголовок ответа клиенту. Установка максимального возраста Cookie, а также разрешение хранить Cookie определяют, будут ли Cookies сохранены на клиенте и время их хранения
`ServletOutputStream getOutputStream()`	Получение бинарного потока вывода для отправления бинарных данных клиенту
`PrintWriter getWriter`	Получение символьного потока вывода для отправления текстовых данных клиенту

Окончание таблицы 1.3

Метод	Описание
`void setContentType(String type)`	Определение MIME-типа ответа браузеру. MIME-тип помогает браузеру определить, как отображать данные. Например, MIME-тип `"text/html"` указывает, что ответ является HTML-документом, поэтому браузер отображает HTML-страницу

Жизненный цикл сервлета

Жизненный цикл сервлета определяют следующие основные методы: `init()`, `service()` и `destroy()`. Они реализуются каждым сервлетом и в нужный момент вызываются сервером.

```
public interface Servlet {
  public void init(ServletConfig config)
    throws ServletException;
  public ServletConfig getServletConfig();
  public void service(ServletRequest req,
    ServletResponse res)
    throws ServletException, IOException;
  public String getServletInfo();
  public void destroy();
}
```

5 шагов жизненного цикла сервлета:

1. Скачать класс `Servlet` в память.

2. Создать объект `Servlet`.

3. Вызвать метод `init()` в `Servlet`.

4. Вызвать метод `service()` в `Servlet`.

5. Вызвать метод `destroy()` в `Servlet`.

Жизненный цикл сервлета начинается с его загрузки в память контейнером сервлетов при старте либо в ответ на первый запрос. Далее происходят инициализация, обслуживание запросов и завершение существования.

Первым вызывается метод `init()`. Он дает сервлету возможность инициализировать данные и подготовиться для обработки запросов. Чаще всего в этом методе программисты помещают код, кеширующий данные фазы инициализации.

После этого сервлет можно считать запущенным, он находится в ожидании запросов от клиентов. Появившийся запрос обслуживается методом `service()` сервлета, а все параметры запроса упаковываются в объект `ServletRequest`, который передается в качестве первого параметра методу `service()`. Второй параметр метода – объект `ServletResponse`. В этот объект упаковываются выходные данные в процессе формирования ответа клиенту. Каждый новый запрос приводит к новому вызову метода `service()`. В соответствии со спецификацией JSDK метод `service()` должен уметь обрабатывать сразу несколько запросов, т. е. быть синхронизирован для выполнения в многопоточных средах. Если же нужно избежать множественных запросов, сервлет должен реализовать интерфейс `SingleThreadModel`, который не содержит ни одного метода и только указывает серверу на однопоточную природу сервлета. При обращении к такому сервлету каждый новый запрос будет ожидать в очереди, пока не завершится обработка предыдущего запроса.

После завершения выполнения сервлета контейнер сервлетов вызывает метод `destroy()`, в теле которого следует помещать код освобождения занятых сервлетом ресурсов.

Интерфейсом Servlet предусмотрена реализация еще двух методов: `getServletConfig()` и `getServletInfo()`. Первый возвращает объект типа `ServletConfig`, содержащий параметры конфигурации сервлета, а второй – строку, описывающую назначение сервлета.

При разработке сервлетов в качестве базового класса в большинстве случаев используют не интерфейс `Servlet`, а класс `HttpServlet`, отвечающий за обработку запросов HTTP.

Класс `HttpServlet` имеет реализованный метод `service()`, служащий диспетчером для других методов, каждый из которых обрабатывает методы доступа к ресурсам. В спецификации HTTP определены следующие методы: `GET`, `HEAD`, `POST`, `PUT`, `DELETE`, `OPTIONS` и `TRACE`. Наиболее часто употребляются методы `GET` и `POST`, с помощью которых на сервер передаются запросы, а также параметры для их выполнения.

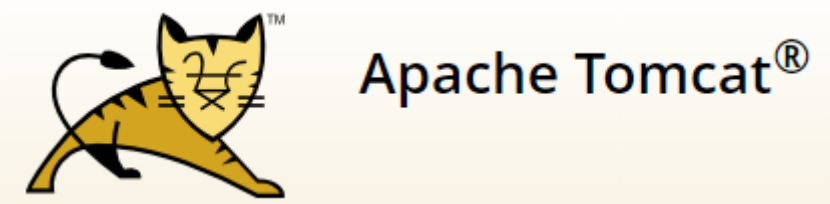
При использовании метода `GET` (по умолчанию) параметры передаются как часть URL, значения могут выбираться из полей формы или передаваться непосредственно через URL. При этом запросы кешируются и имеют ограничения на размер. При использовании метода `POST` (`method=POST`) параметры (поля формы) передаются в содержимом HTTP-запроса и упакованы согласно полю заголовка `Content-Type`. По умолчанию в формате: `<имя>=<значение>&<имя>=<значение>&...`

В задачу метода `service()` класса `HttpServlet` входит анализ полученного через запрос метода доступа к ресурсам и вызов метода, имя которого сходно с названием метода доступа к ресурсам, но перед именем добавляется префикс `do`: `doGet()` или `doPost()`. Кроме этих методов могут использоваться методы: `doHead()`, `doPut()`, `doDelete()`, `doOptions()` и `doTrace()`. Разработчик должен переопределить нужный метод, разместив в нем функциональную логику.

Web-server

Итак, чтобы мы смогли запустить web-приложение, нам нужен web-сервер, который поддерживает Servlet API (т. е. в котором есть расширение, реализующее для web-сервера поддержку Servlet API).

Web-сервер – это сервер, который принимает HTTP-запросы от клиентов и выдает им HTTP-ответы (как правило, вместе с HTML-страницей, изображением, файлом или другими данными). Запрашиваемые ресурсы обознаются URL-адресами. Большинство web-серверов – сложные механизмы, которые состоят из различных компонентов, и каждый из них выполняет определенные функции. Одним из самых популярных web-серверов с поддержкой Servlet API является **Apache Tomcat** (https://tomcat.apache.org/).



Apache Tomcat – это контейнер, который позволяет вам использовать интернет-приложения, такие как Java-сервлеты и JSP (серверные страницы Java). Он написан на языке Java, реализует спецификацию сервлетов и спецификацию JavaServer Pages (JSP) и JavaServer Faces (JSF). Tomcat используется в качестве самостоятельного web-сервера, в качестве сервера контента в сочетании с web-сервером Apache HTTP Server, а также в качестве контейнера сервлетов в серверах приложений JBoss и GlassFish.

Начиная с Tomcat 4.x выпускается с Catalina (контейнер сервлетов), Coyote (HTTP-коннектор) и Jasper (JSP-движок).

Catalina – контейнер сервлетов Tomcat'а, который реализует спецификацию сервлетов Servlet API. Servlet API является основой для всех остальных технологий Java, касающихся Web, и дает возможность динамически генерировать любой web-контент, используя любые библиотеки, доступные для Java. Архитектором Catalina являлся Craig McClanahan.

Coyote – компонент стека HTTP Tomcat’a, который поддерживает протокол HTTP 1.1 для web-серверов или контейнера приложений. Coyote прослушивает входящие соединения на определенном TCP-порте сервера, пересылает запросы в механизм Tomcat для обработки и отправляет ответ назад запрашивающему клиенту.

Jasper – механизм JSP Tomcat’a. Tomcat 5.x использует Jasper 2, который является реализацией спецификации JavaServer Pages 2.0 Sun Microsystems. Jasper анализирует JSP-файлы, чтобы компилировать их в Java-код как сервлеты (которые могут быть обработаны с помощью Catalina). Во время выполнения Jasper может автоматически обнаруживать изменения JSP-файла и перекомпилировать его.

Web-приложение (Web Application)

Согласно главе 10 **Web Applications** спецификации Servlet API [3], *web-приложение* – это набор сервлетов, HTML-страниц, классов и других ресурсов, которые составляют законченное приложение на web-сервере.

JavaServer Pages

Java Server Pages (JSP) является стандартным расширением Java, которое определено на основании сервлетного расширения. Целью JSP является упрощение создания и управления динамическими web-страницами. Tomcat автоматически поддерживает JSP.

JSP позволяет вам комбинировать HTML-код web-страницы с кусочками Java-кода в одном и том же документе. Код Java окружается специальными ярлыками, которые говорят JSP-контейнеру, что он должен использовать этот код для генерации сервлета или его части. Преимущество JSP в том, что вы можете иметь единый документ, который представляет и страницу, и Java-код, который включается в нее.

В первый раз JSP загружается JSP-контейнером (который обычно связан с web-сервером или является его частью), код сервлета, помеченный JSP-ярлыками, автоматически генерируется, компилируется и загружается и контейнер сервлетов. Статическая часть HTML-страницы воспроизводится путем посылки статического объекта `String` в метод `write()`. Динамическая часть включается прямо в сервлет.

Исходя из этого, пока исходный текст JSP-страницы не изменяется, она ведет себя так, как будто это статическая HTML-страница с ассоциированным сервлетом (однако весь HTML-код на самом деле генерируется сервлетом). Если вы изменяете исходный код для JSP, он автоматически перекомпилируется и перегружается при следующем запросе этой страницы. Конечно, из-за всей этой динамики вы увидите замедленный ответ на первый запрос этой JSP-страницы. Но поскольку JSP использует немного больше, чем просто изменения, обычно вы не встретите этой задержки.

Структура JSP-страницы состоит из перемешивания сервлета и HTML-страницы. JSP-ярлыки начинаются и заканчиваются угловыми скобками, так же как и ярлыки HTML, но эти ярлыки также включают символ `%`, так что все JSP-ярлыки обозначаются:

```
<% JSP code here %>
```

За первым знаком процента могут следовать другие символы, которые означают часть JSP-кода в ярлыке.

Ниже приведен очень простой пример JSP, который использует стандартный вызов Java-библиотеки для получения текущего времени в миллисекундах. Так как используется *JSP-выражение* (`<%=`), результат вычислений конвертируется в `String` и помещается на генерируемую web-страницу:

```
<html><body>
<H1>The time in seconds is:
<%= System.currentTimeMillis()%></H1>
</body></html>
```

Директивы являются сообщениями для JSP-контейнера и указываются символом @:

```
<%@ directive {attr="value"}* %>
```

Директивы ничего не посылают в поток out, но они важны в настройке атрибутов ваших JSP-страниц и взаимодействий с JSP-контейнером. К таким атрибутам относятся: language, extends, import, session, buffer, autoFlush, isThreadSafe, info и errorPage.

Например, строка <%@ page language="java" %> сообщает, что на JSP-странице используется язык скриптов Java.

После того как были использованы директивы для настойки среды скриптов, вы можете использовать элементы языка скриптов. JSP 1.1 имеет три элемента языка скриптов: *declarations, scriptlets* и *expressions*. Declaration декларирует элементы, scriptlet – это фрагмент инструкций, а expression – это законченное выражение языка. В JSP каждый элемент сценария начинается с <%. Синтаксис для каждого следующий:

```
<%! declaration %>
<%  scriptlet   %>
<%= expression  %>
```

Пробелы после <%!, <%, <%= и перед %> не обязательны.

Все эти ярлыки базируются на XML. Вы даже можете сказать, что JSP-страница может быть преобразована в XML-документ.

Declaration применяется для объявления переменных и методов в языке скриптов (пока только в Java), используемых JSP-страницей. Декларация должна быть завершенной инструкцией Java и не может совершать вывод в поток out.

Скриплеты могут содержать любой фрагмент кода, содержащий правильные Java-инструкции. Скриплеты выполняются во время обработки запроса. Когда все фрагменты скриплетов в данном JSP скомбинированы так,

как они введены в JSP-страницу, они должны произвести действительное выражение, которое определено в языке программирования Java.

JSP-выражения (expression) могут быть найдены среди HTML-кода в средней части страницы. Выражения должны быть законченными Java-инструкциями, которые вычисляют, приводят к `String` и посылают в `out`. Если результат выражения не может быть приведен к `String`, выбрасывается `ClassCastException`.

1.2 Порядок выполнения лабораторной работы

1.2.1 Установка ПО

Минимальный комплект для разработки программ на Java:

- JDK (Java Development Kit) – комплект разработки программного обеспечения (компилятор, стандартные библиотеки и т. п.);
- JRE (Java Runtime Environment) – программа для запуска и исполнения программ (среда выполнения Java);
- среда программирования Eclipse.

Установка Java SDK (JDK)

Самые новые версии системного программного обеспечения JRE и JDK можно загрузить с сайта компании Oracle (https://www.oracle.com/).

Загрузите и установите Java SE Development Kit 8u261 (рис. 1.1) по ссылке https://www.oracle.com/technetwork/java/javase/downloads/index.html.

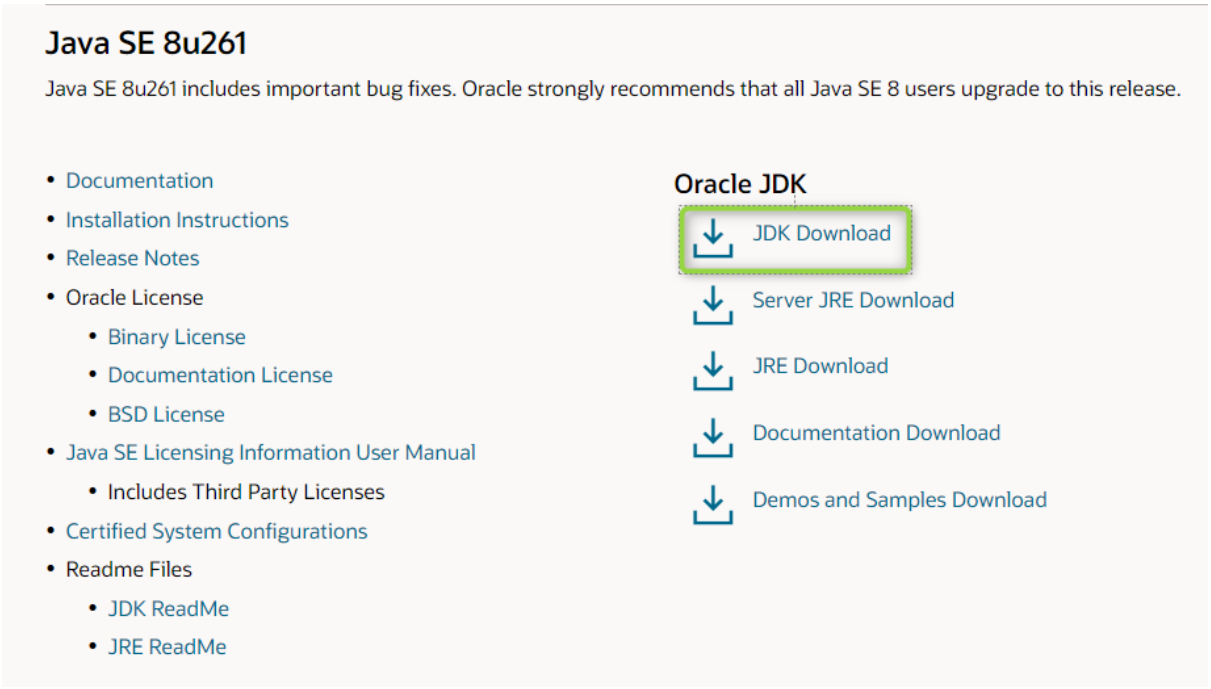


Рис. 1.1 – Загрузка Java SE Development Kit 8u261

В курсе используется версия 8. Эта версия JDK поставляется в комплекте с Java Runtime Environment (JRE), поэтому вам не нужно отдельно загружать и устанавливать JRE. Убедитесь, что правильно выбрали между 32-битной и 64-битной версиями для своей ОС Windows. После установки рекомендуется перезагрузить компьютер.

IDE Eclipse

Eclipse (https://www.eclipse.org) представляет собой бесплатный и гибкий редактор с открытым исходным кодом. Он может оказаться полезен как для новичков, так и для профессионалов. Первоначально создаваемый как среда для Java-разработки сегодня Eclipse имеет широкий диапазон возможностей благодаря большому количеству плагинов и расширений. Поддерживаемые языки: C, C++, Java, Perl, PHP, Python, Ruby и другие. Множество пакетных решений обеспечивает многоязычную поддержку. Существует интеграция с Maven, JUnit и TestNG, а также возможности анализа покрытия кода. Какая

IDE окажется лучшей именно для вас, зависит от используемой операционной системы, языка программирования и тех задач, которые вы хотите выполнять. В курсе показан пример в среде Eclipse (Java EE IDE for Web Developers, Version: 2018-09 (4.9.0)) (рис. 1.2).

Скачать ПО можно по ссылке:
https://www.eclipse.org/downloads/packages/release/2018-09/r/eclipse-ide-java-ee-developers

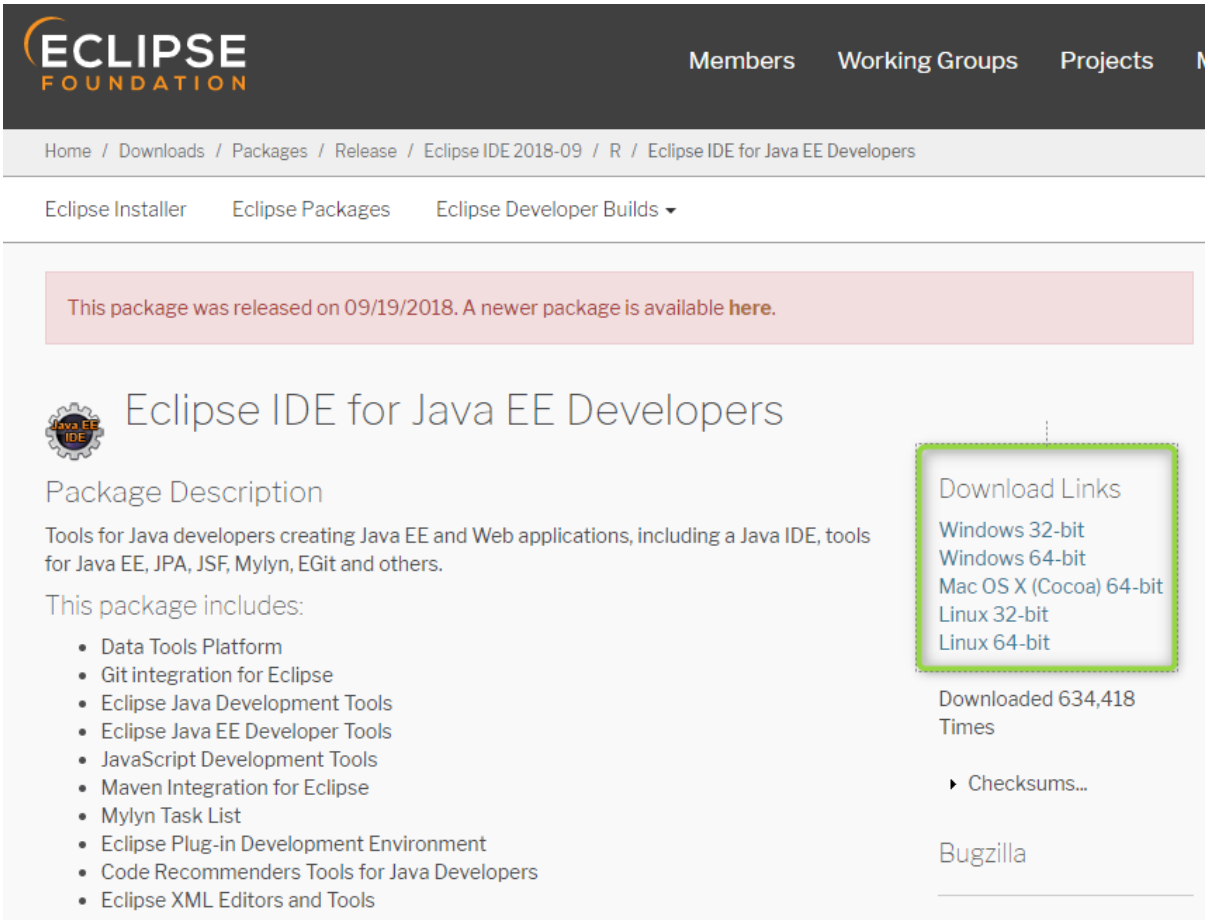


Рис. 1.2 – Загрузка Eclipse

После скачивания архив нужно распаковать в папку C:\Program Files\eclipse.

Запуск Eclipse

Чтобы запустить Eclipse IDE, нужно открыть файл *eclipse.exe*, находящийся в папке C:\Program Files\eclipse\. Начнется запуск IDE (рис. 1.3).

Рис. 1.3 – Запуск Eclipse

При запуске откроется окно, предлагающее выбрать рабочую область (*Workspace*) (рис. 1.4), где будут храниться программные файлы проекта.

В простейшем случае рабочее пространство – это каталог для проектов пользователя, в котором располагаются файлы проекта. Все, что находится внутри этого каталога, считается частью рабочего пространства. Указываем удобную для нас директорию и нажимаем *OK*.

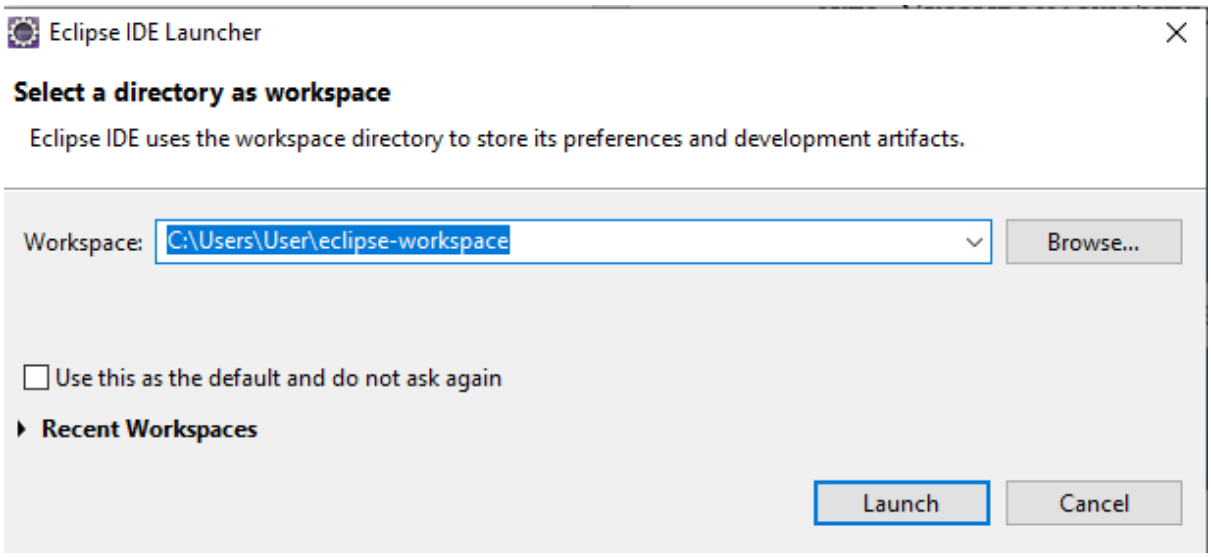


Рис. 1.4 – *Workspace*

Инструментальные средства Eclipse становятся доступны сразу после запуска приложения. Это по существу сама платформа с набором различных функциональных возможностей главного меню, где прежде всего выделяется набор операций по управлению проектом. Фактическая обработка, как правило, осуществляется дополнениями (плагинами), например, редактирование и просмотр файлов проектов осуществляется JDT и т. д.

К *инструментам* (*Workbench*) относится набор соответствующих редакторов и представлений, размещенных в рабочей области Eclipse.

Для конкретной задачи определенный набор редакторов и представлений называют перспективой или компоновкой.

Компоновка (*Perspective*) – это набор представлений и редакторов, расположенных в том порядке, который вам требуется. В каждой компоновке присутствует свой набор инструментов, некоторые компоновки могут иметь общие наборы инструментов. В определенный момент времени активной может быть только одна компоновка. Переключение между различными компоновками осуществляется нажатием клавиш <*Ctrl + F8*>.

Используя компоновки, вы можете настроить свое рабочее пространство под определенный тип выполняемой задачи. В Eclipse имеется также возможность создавать свои компоновки. Открыть компоновку можно командой *Window / Open Perspective* (рис. 1.5).

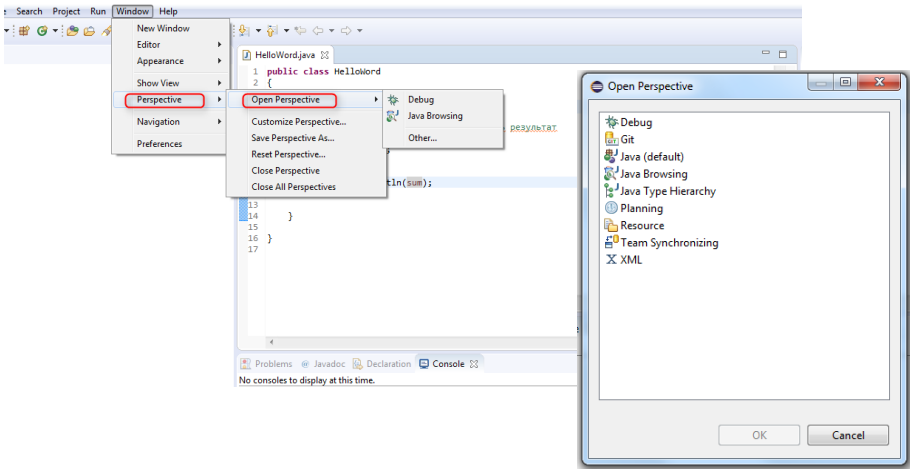


Рис. 1.5 – *Perspective*

Редакторы (*Editors*) представляют собой программные средства, позволяющие осуществлять операции с файлами (создавать, открывать, редактировать, сохранять и др.).

Представления (*Views*) по существу являются дополнениями к редакторам, где выводится информация сопроводительного или дополнительного характера, как правило, о файле, находящемся в редакторе. Открыть представления можно командой *Window / Show View* (рис. 1.6).

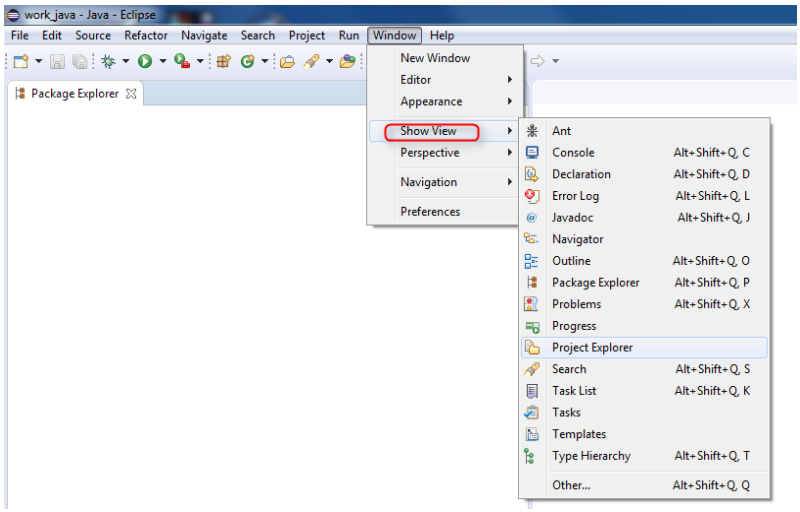


Рис. 1.6 – *Views*

Проект (*Project*) представляет собой набор файлов приложения и сопутствующих дополнений.

Дополнением (*Plug-in*) называют приложение, которое дополнительно может быть установлено в Eclipse.

Примером дополнения может выступать JDT.

Проект *Java development tools (JDT)* с помощью JDT-плагина обеспечивает среду разработки Java-приложений, включая создание Eclipse-плагинов. JDT-плагин добавляет перспективу Java в панель инструментов и Java-группу шаблонов в команду *New* меню *File*, а также предоставляет набор окон, редакторов и других инструментов для работы с Java-кодом.

Eclipse-плагины добавляют к Eclipse-платформе новые типы редакторов, представлений и перспектив. К существующим редакторам, представлениям и перспективам могут добавляться новые действия в меню и панелях инструментов.

После того как вы нажмете кнопку *OK* на окне приветствия, появится страница приветствия (рис. 1.7), на которой имеется 5 графических кнопок:

- *Overview* – обзор, содержащий ссылки на обучающие интернет-ресурсы Eclipse;
- *Tutorials* – уроки, содержат несколько примеров создания простейших приложений Java;
- *What's new* – содержит обзор основных нововведений;
- *Samples* – примеры, содержит несколько примеров разработки, которые должны быть предварительно установлены для того, чтобы их можно было просмотреть;
- *Workbench* – «рабочий стол», рабочая область программиста.

Рис. 1.7 – Окно приветствия Eclipse

Перспектива Java содержит окно редактора и представления *Package Explorer*, *Outline*, *Problems*, *Javadoc*, *Declaration* (табл. 1.4). Перспектива Debug содержит окно редактора и представления *Debug*, *Breakpoints*, *Console*, *Tasks* (табл. 1.4).

Таблица 1.4 – Характеристика представлений

Представления	**Описание**
Package Explorer	Отображает Java-проект с его структурой, определяемой сборкой проекта, в виде узлов папок и библиотек, пакетов, файлов с их внутренней структурой
Outline	Отображает компилируемую структуру редактируемого в данный момент Java-файла
Problems	Отображает ошибки и предупреждения сборщика проекта
Javadoc	Отображает документацию выбранного в данный момент Java-элемента
Declaration	Отображает исходный код выбранного в данный момент Java-элемента
Console	Отображает системный вывод выполнения Java-кода
Debug	Обеспечивает управление процессом отладки и запуска Java-кода
Tasks	Отображает список маркеров задач проекта
Breakpoints	Отображает список контрольных точек отладки Java-кода

Настройка web-сервера Apache Tomcat

Чтобы начать работать с Servlet, вам необходимо скачать Tomcat Web Server и объявить его в IDE.

Скачать можно по ссылке: https://tomcat.apache.org/download-90.cgi

В примере используется версия 9. Скачаем из раздела Binary Distributions тип поставки Core в формате zip и распакуем скачанный архив apache-tomcat-9.0.37.zip (рис. 1.8). Не забудьте, куда вы его распаковали.

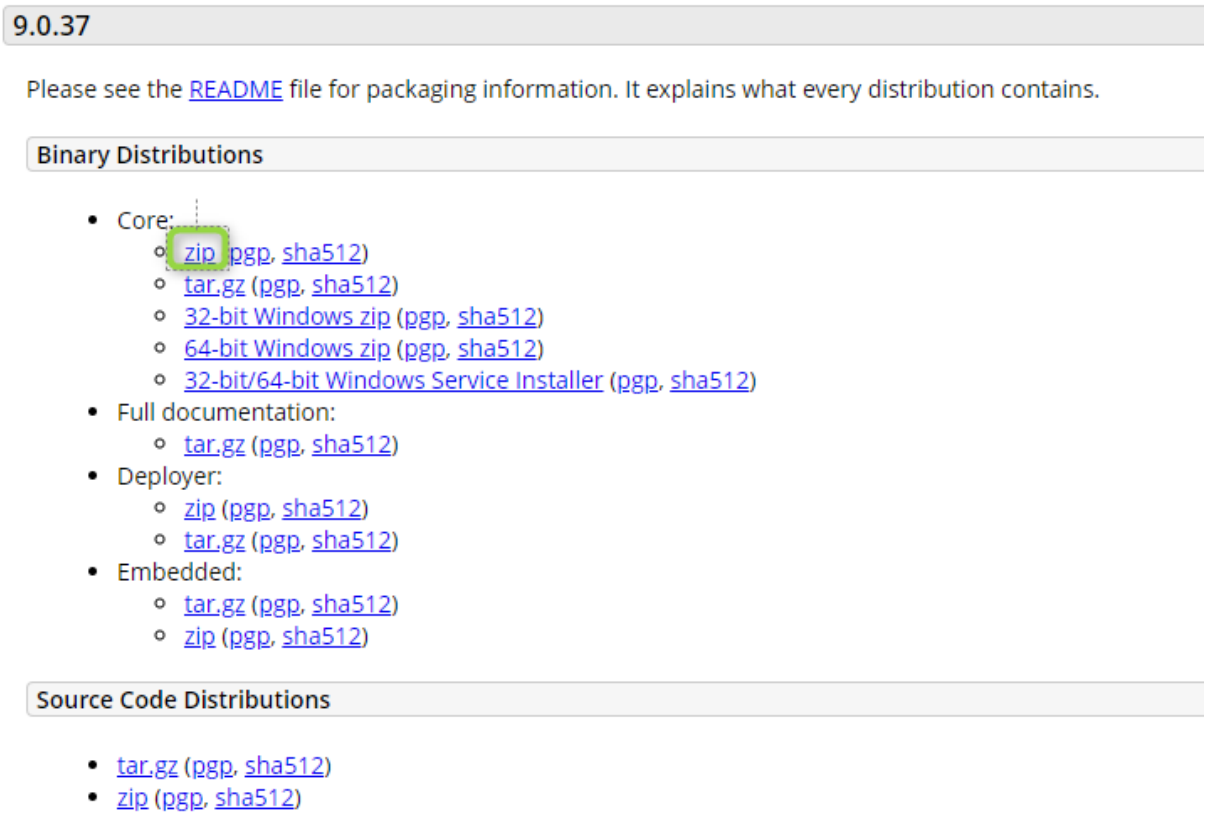


Рис. 1.8 – Загрузка Tomcat Web Server

Для настройки сервера Tomcat используются следующие конфигурационные XML-файлы, размещенные в каталоге /usr/share/tomcat/conf/ (рис. 1.9).

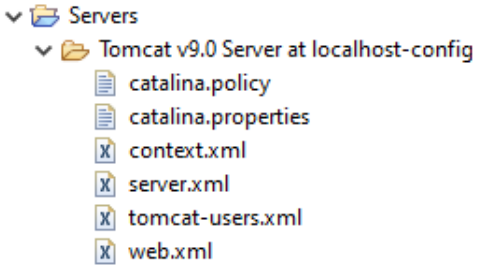


Рис. 1.9 – Конфигурационные XML-файлы Tomcat

server.xml: общие настройки сервера (порты, виртуальные хосты и пр.).

web.xml: параметры, общие для всех web-приложений на текущем сервере. Настройки отдельных web-приложений задаются в их собственных файлах /WEB-INF/web.xml (здесь можно провести аналогию с использованием файла .htaccess в Apache).

context.xml: общие настройки управления контентом.

tomcat-users.xml: список пользователей и групп (ролей).

В Eclipse выберите *Window* → *Preferences* и *Server* → *Runtime Environments* (рис. 1.10, 1.11).

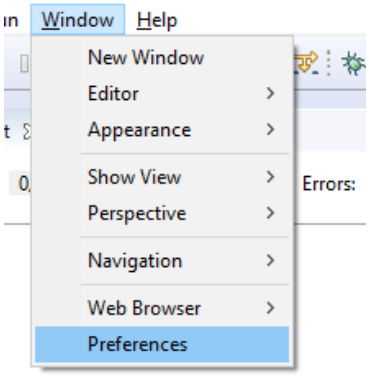


Рис. 1.10 – Настройка сервера Tomcat

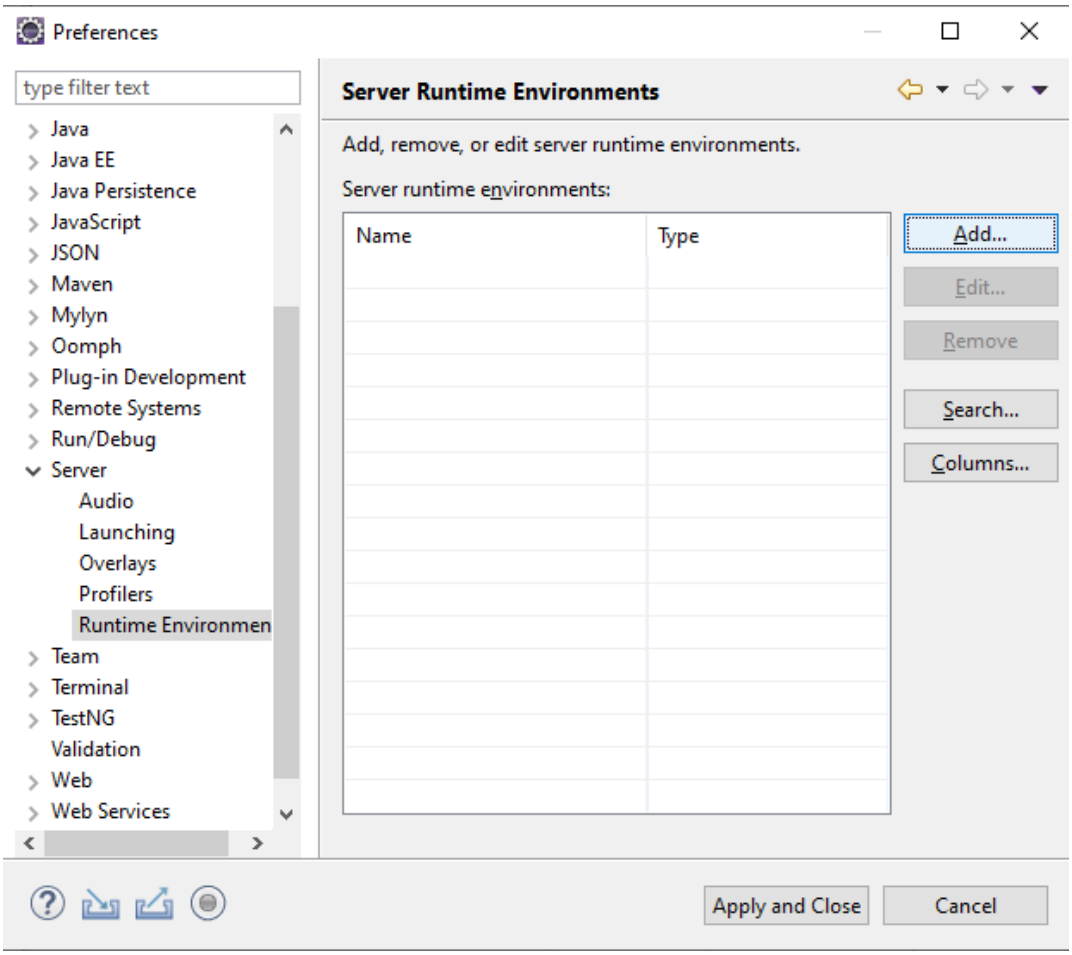


Рис. 1.11 – Объявление Tomcat в Eclipse

Далее необходимо выбрать тип WebServer Tomcat (рис. 1.12).

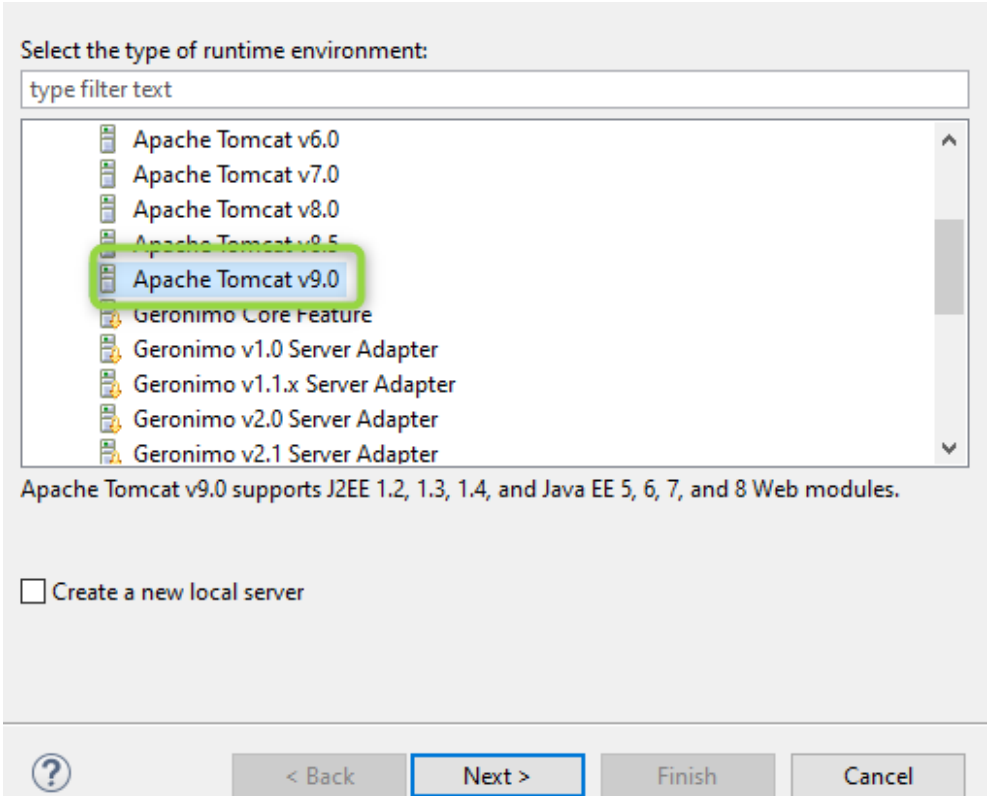


Рис. 1.12 – Выбор типа WebServer Tomcat

Укажите место Tomcat, куда вы извлекли архив на жестком диске и нажмите кнопку *Apply and Close*, чтобы закончить (рис. 1.13, 1.14).

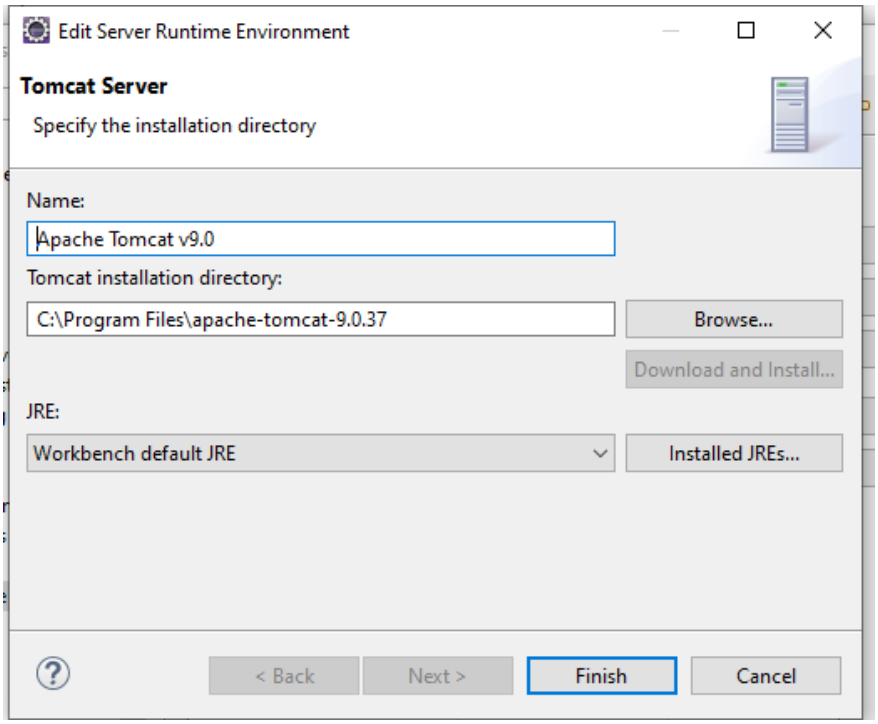


Рис. 1.13 – Настройка пути к серверу Tomcat

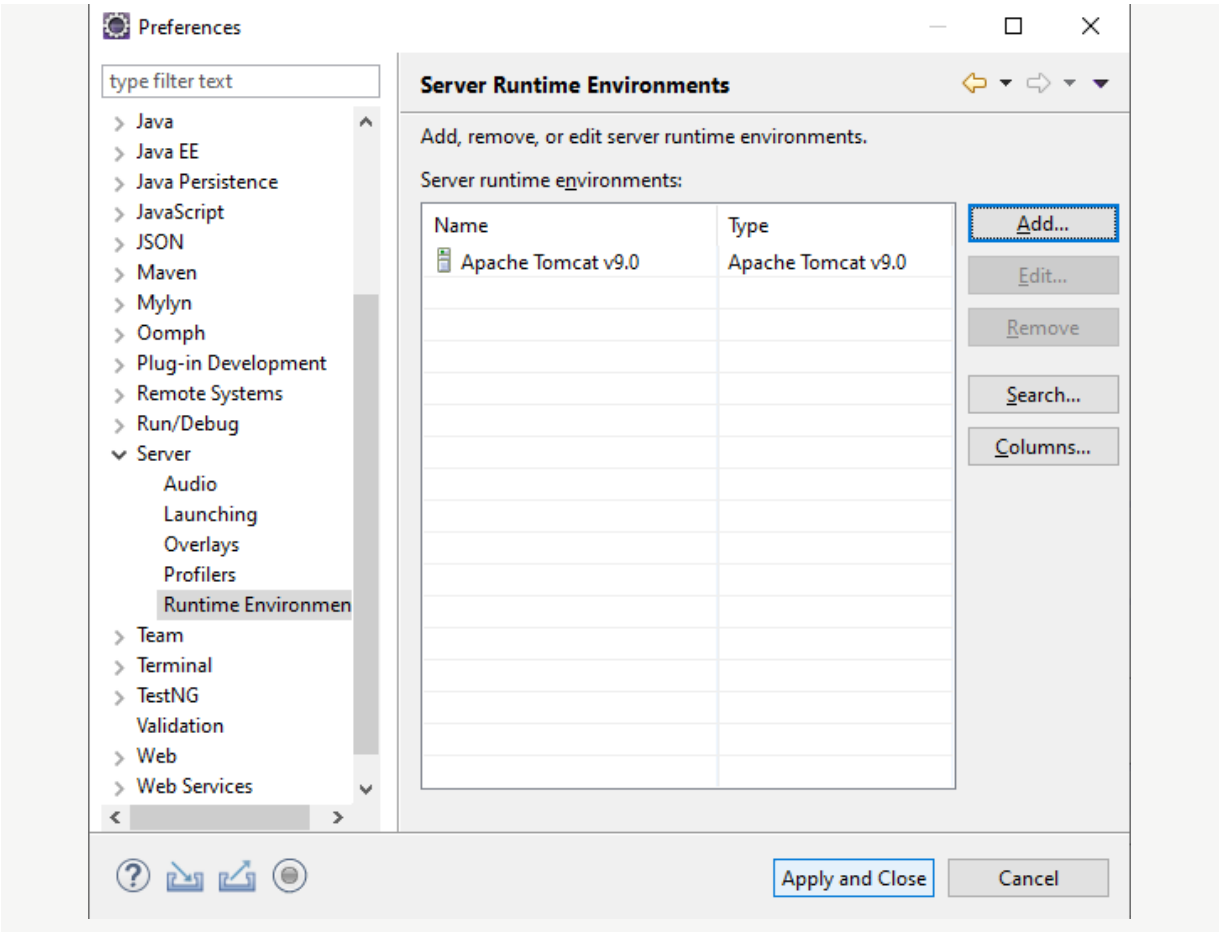


Рис. 1.14 – Настройка среды выполнения сервера

1.2.2 Развертывание сервлета в IDE Eclipse

Создадим web-проект в Eclipse: *File* → *New* → *Dynamic Web Project* (рис. 1.15).

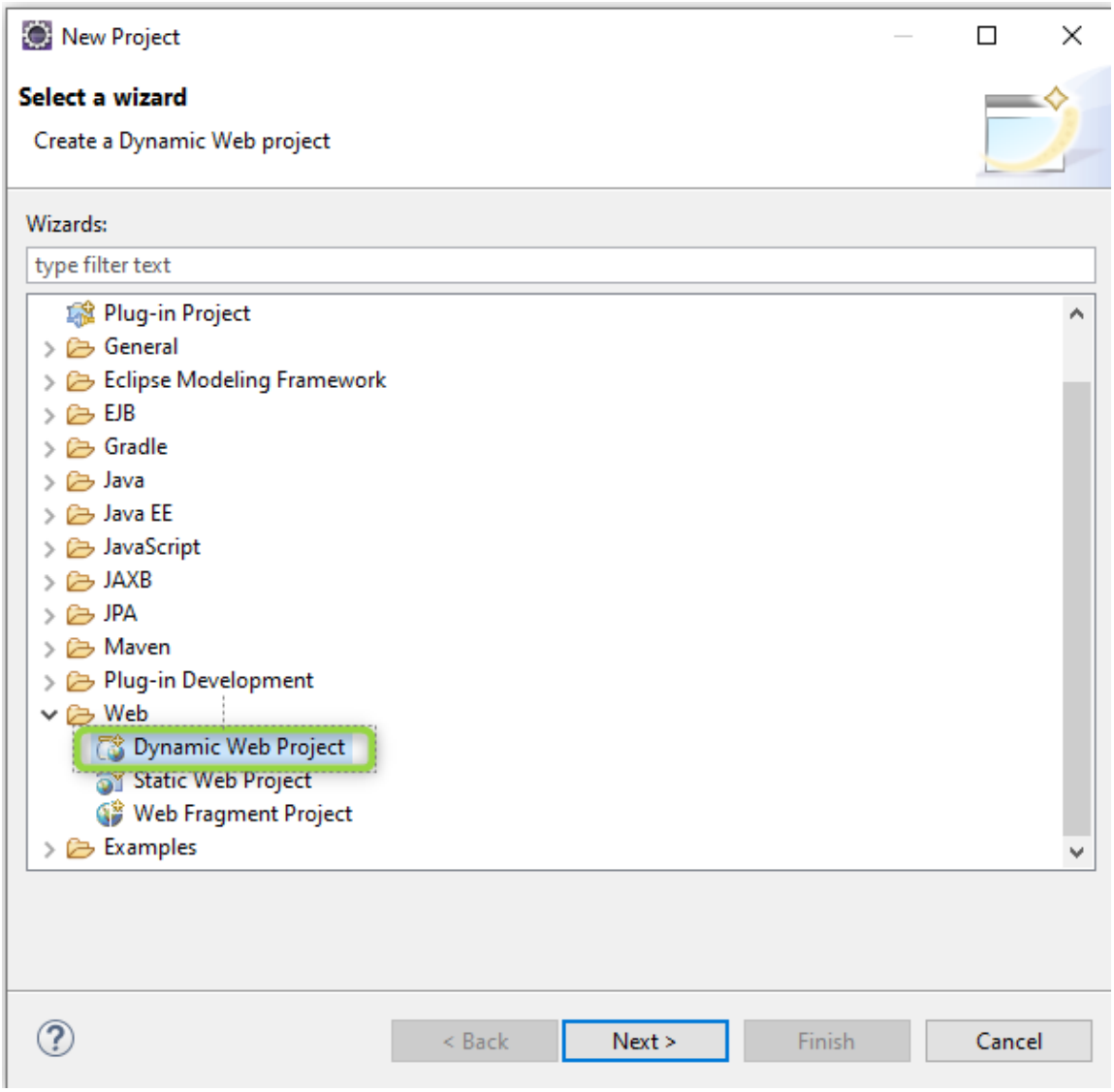


Рис. 1.15 – Выбор мастера настройки нового web-проекта

В появившемся окне вводим название проекта **MyWeb** (в поле *Project name*). Заметьте, что *Target runtime* является сервер, который мы создали ранее (рис. 1.16). Также вы можете поменять версию спецификации сервлетов в *Dynamic web module version*.

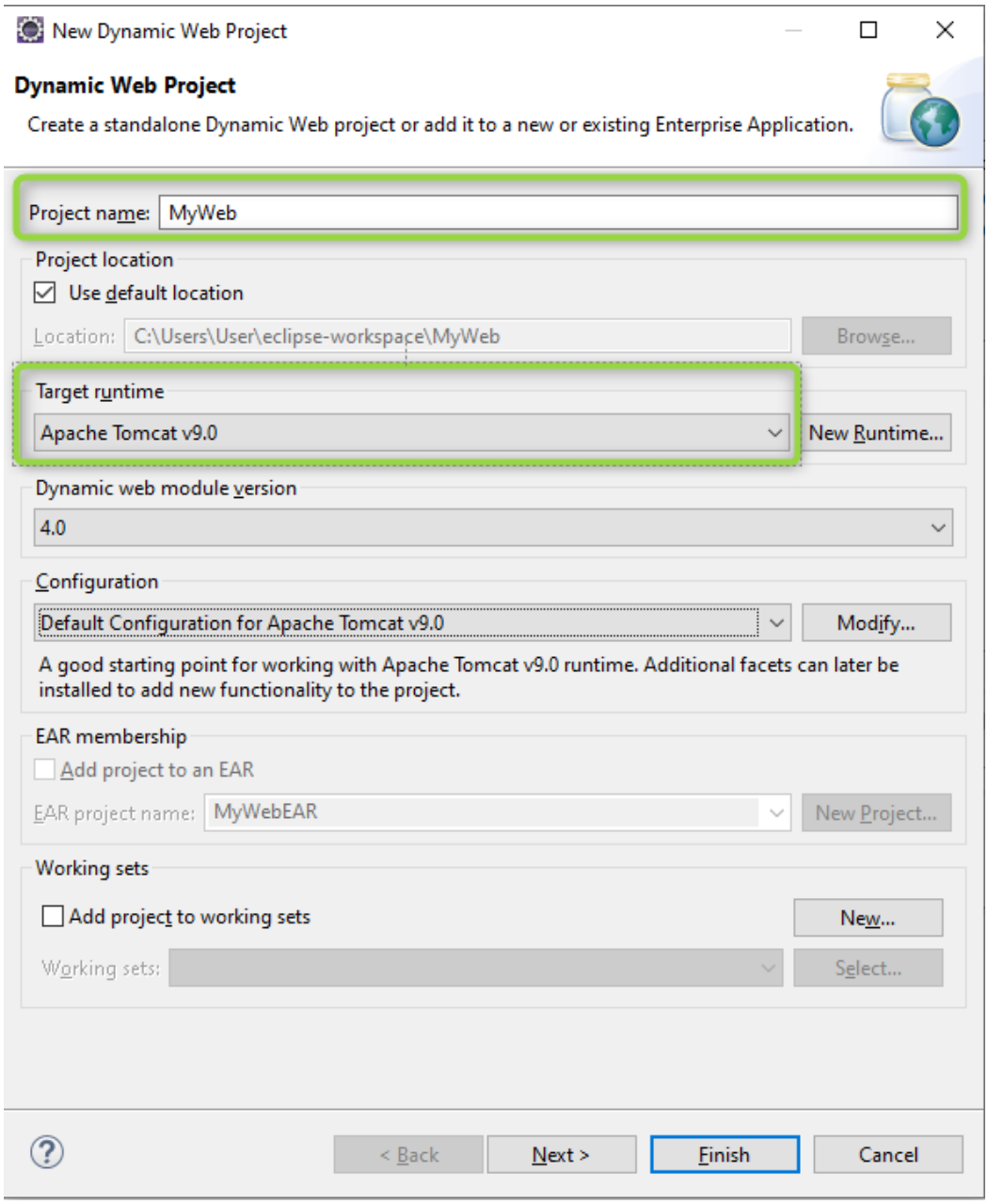


Рис. 1.16 – Настройка нового web-проекта

Нажимаем *Next*, в следующем окне тоже *Next*, в окне *Web Module* ставим метку возле надписи *Generate web.xml deployment descriptor* (это нам понадобится для создания дескриптора, где распишем для Tomcat, какие URL-запросы передавать на какие сервлеты) и жмем *Finish* (рис. 1.17).

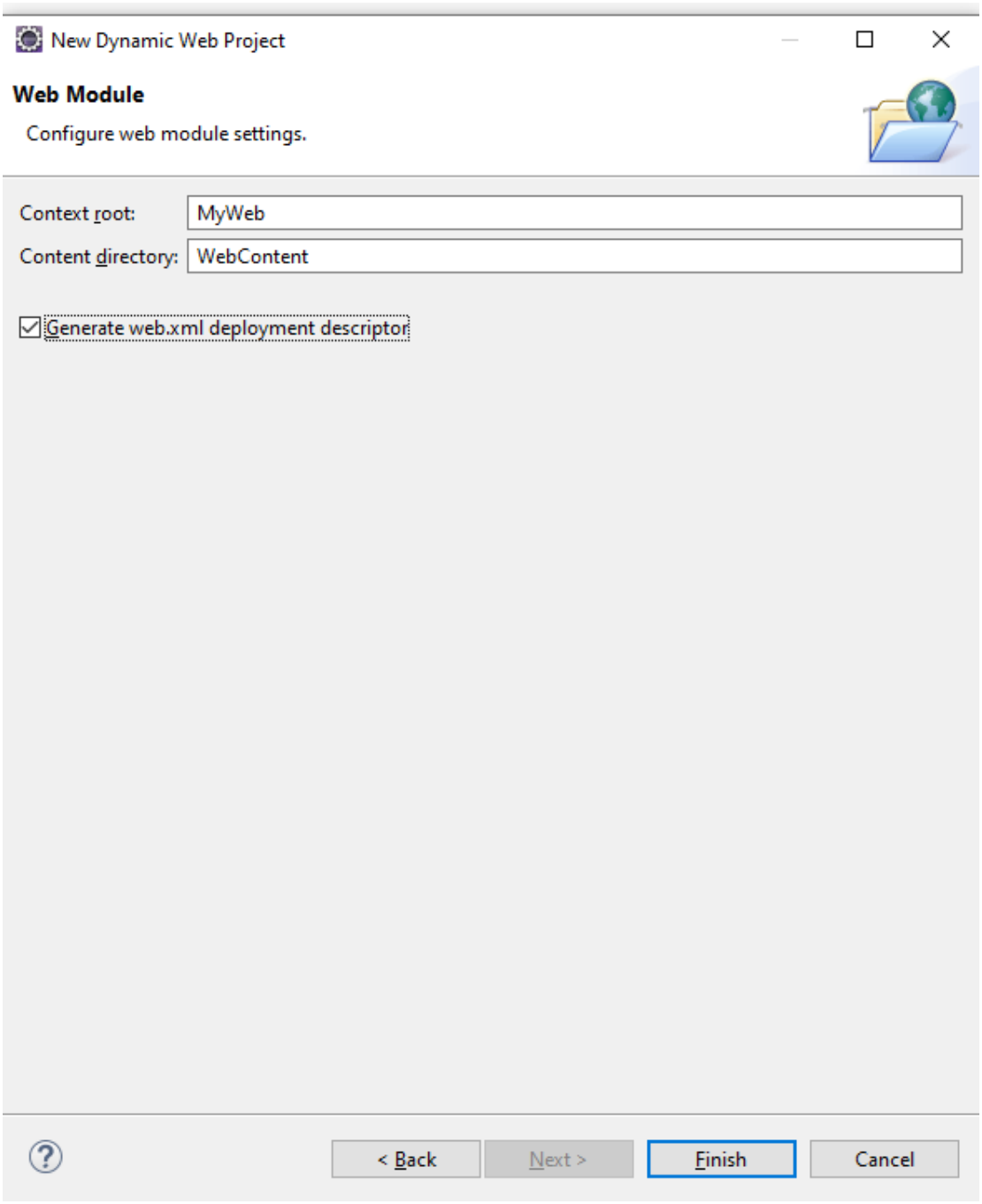


Рис. 1.17 – Настройка конфигурации web-модуля

Проект создан, можно добавлять web-страницы и сервлеты.

Сам проект имеет структуру, представленную на рисунке 1.18.

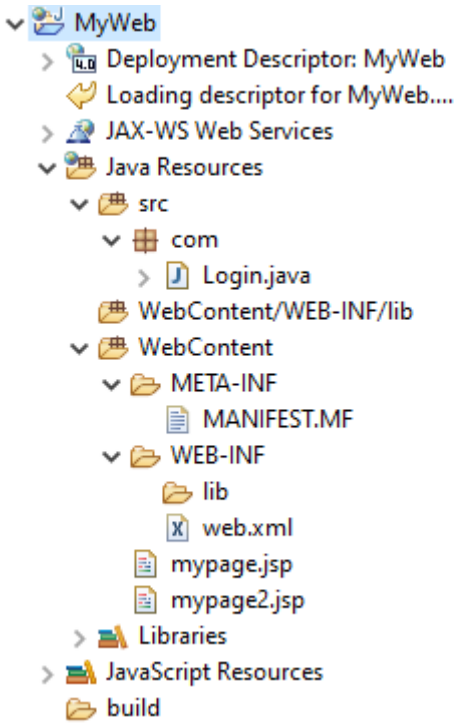


Рис. 1.18 – Структура web-проекта

JavaResources

JavaResources содержит исходный код Java-проекта для классов, компонентов и сервлетов. Когда эти ресурсы добавляются в web-проект, они автоматически компилируются, а созданные файлы добавляются в каталог WEB-INF/classes. Содержимое исходного каталога не упаковывается в файлы WAR, если только опция не указана при создании файла WAR.

WebContent

Обязательное расположение всех web-ресурсов, включая HTML, JSP, графические файлы и т. д. Если файлы не помещены в этот каталог (или в структуру подкаталога этого каталога), файлы не будут доступны при выполнении приложения на сервере. Папка web-содержимого представляет содержимое файла WAR, который будет развернут на сервере. Любые файлы, не находящиеся в папке web-материалов, считаются ресурсами времени разработки (например, файлы .java, .sql и .mif) и не развертываются, когда проект подвергается модульному тестированию или публикации.

Примечание. Хотя для папки по умолчанию задано имя *WebContent*, вы можете изменить имя в *Project Explorer*, щелкнув правой кнопкой мыши по папке и выбрав *Refactor → Rename* или на web-странице диалога свойств проекта. В динамическом web-проекте изменение имени папки приведет к обновлению выходного каталога сборки Java.

WEB-INF

Этот каталог содержит вспомогательные web-ресурсы для web-приложения, включая файл web.xml, а также каталоги классов и lib.

META-INF

Этот каталог содержит файл MANIFEST.MF, который используется для сопоставления путей классов для зависимых файлов JAR, которые существуют в других проектах в том же проекте Enterprise Application. Запись в этом файле обновит путь к классу проекта во время выполнения и параметры сборки Java, чтобы включить в него указанные JAR-файлы.

Lib

Поддерживающие файлы JAR, на которые ссылается ваше web-приложение. Любые классы в файлах .jar, размещенных в этом каталоге, будут доступны для вашего web-приложения.

Добавим в проект пакет **com** (рис. 1.19, 1.20).

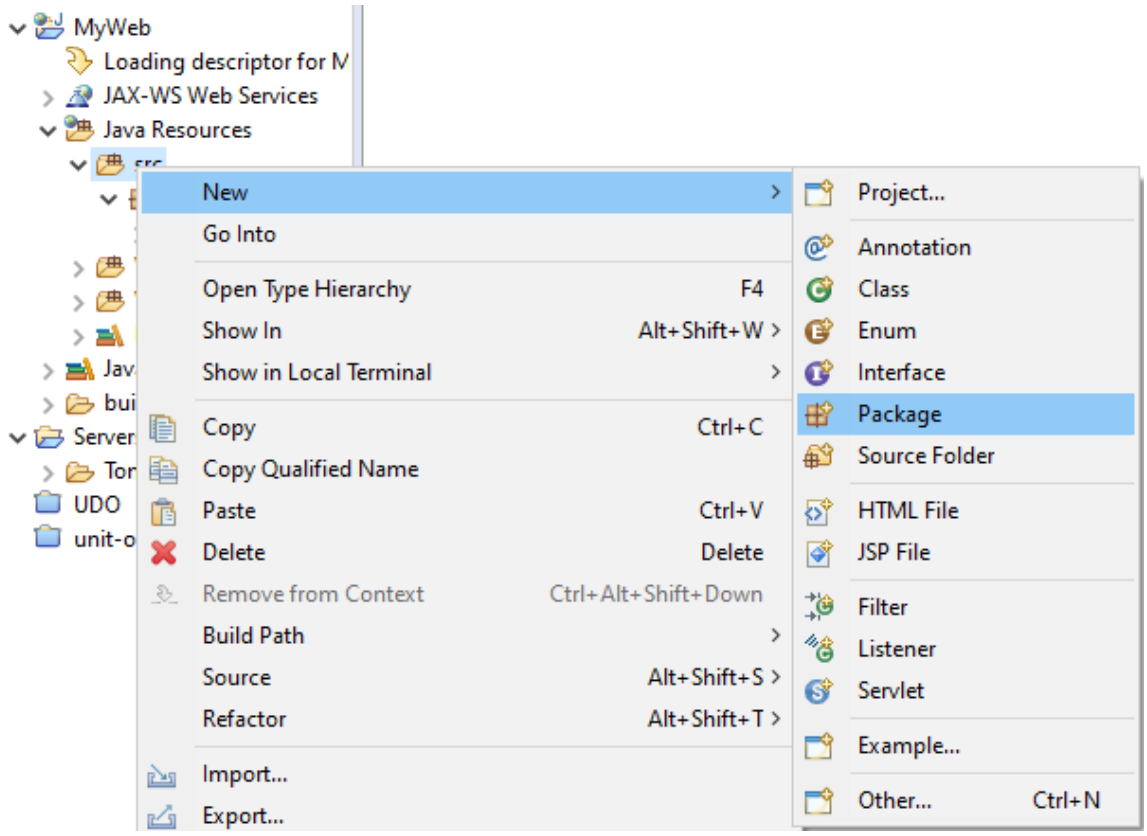


Рис. 1.19 – Выбор мастера Package

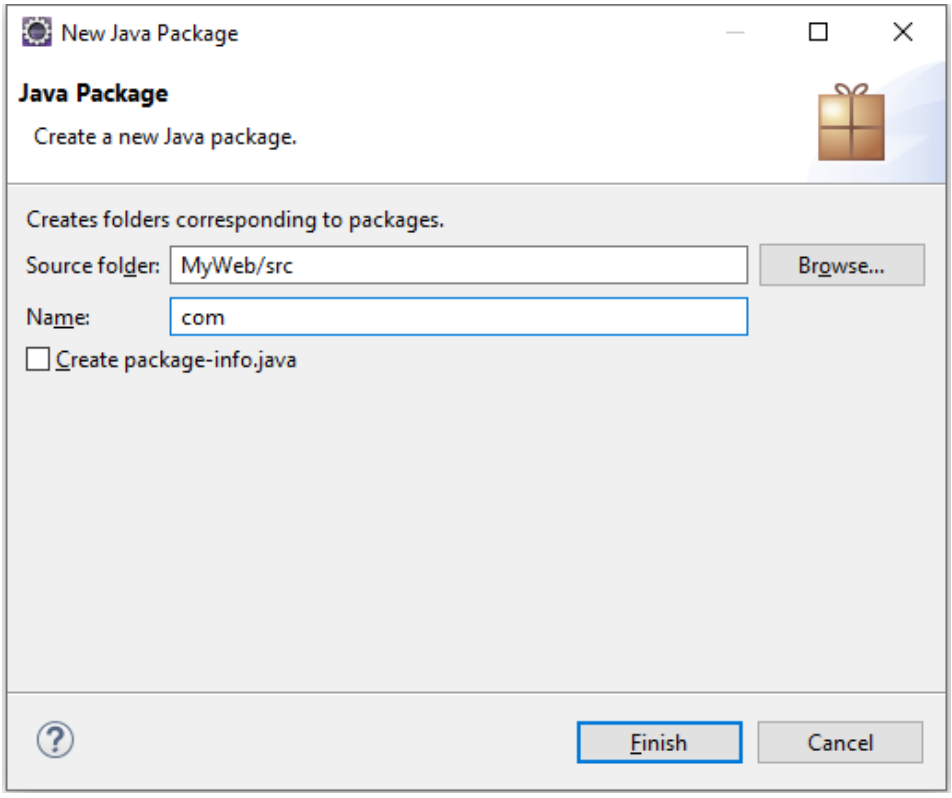


Рис. 1.20 – Настройка нового пакета

Добавим сервлет, который будет обрабатывать ввод данных с web-формы: *Java Resources* → *New* → *Servlet* (рис. 1.21).

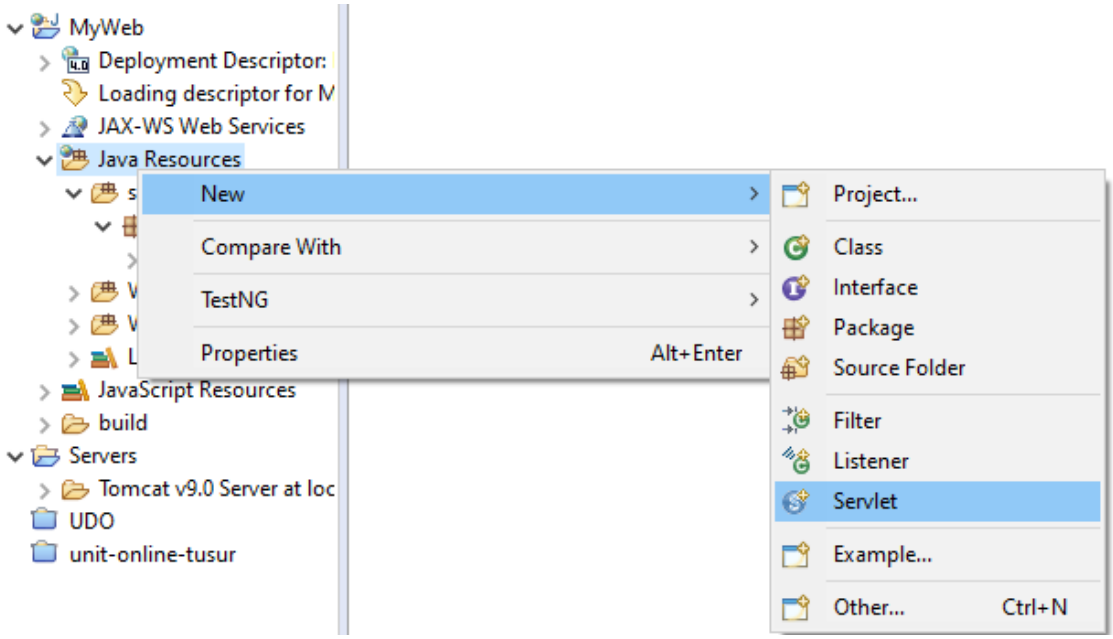


Рис. 1.21 – Выбор мастера Servlet

В появившемся окне в поле *Java package* вставляем название пакета *com*, в поле *Class name* пишем *Login*. Жмем *Next*. В следующем окне жмем *Next*, в последнем окне жмем *Finish* (рис. 1.22).

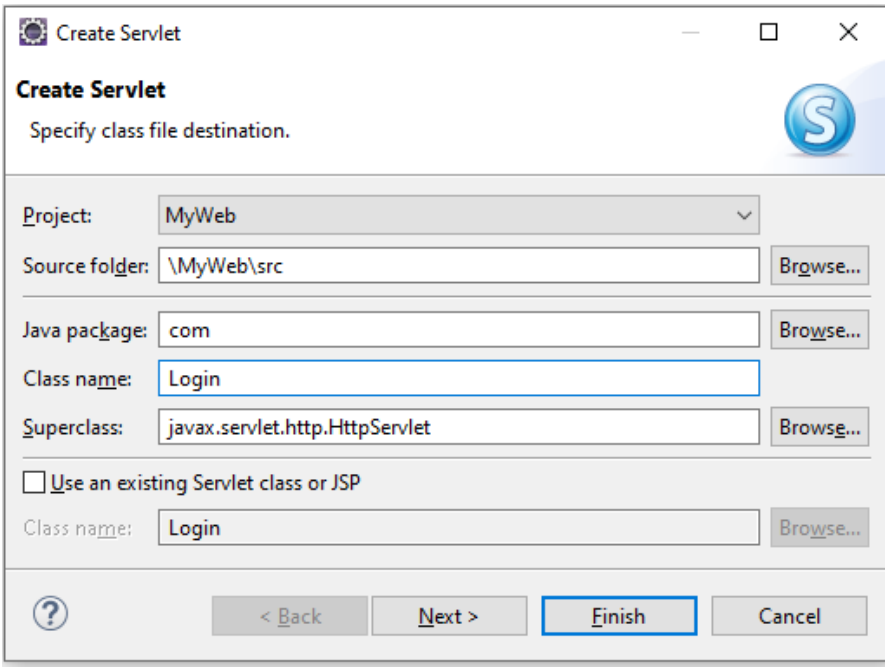


Рис. 1.22 – Настройка нового сервлета

Установим Tomcat в конфигурации среды выполнения (*Runtime Environment*) (рис. 1.23).

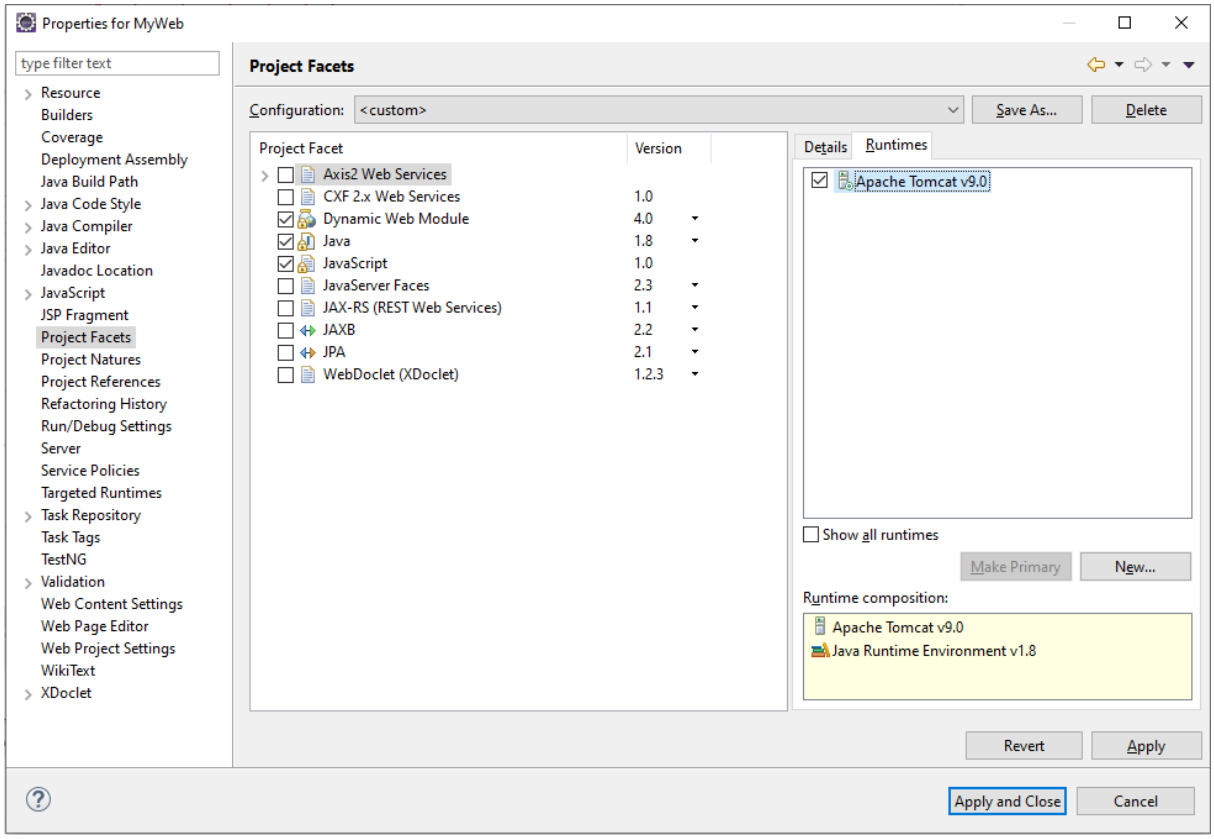


Рис. 1.23 – Настройка конфигурации проекта

Созданный сервлет появится в окне редактора (рис. 1.24).

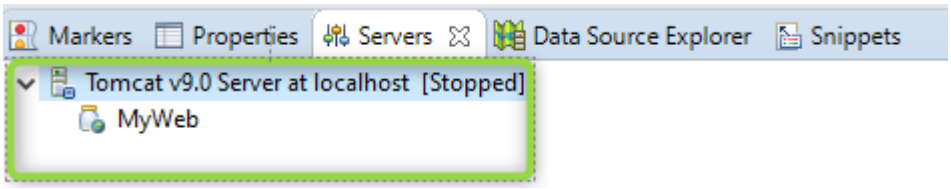


Рис. 1.24 – Отображение сервлета в окне редактора

По давней традиции, восходящей к языку Си, учебники по языкам программирования начинаются с программы *Hello World*. Не будем нарушать эту традицию.

Добавим в метод `doGet()` вывод сообщения *Hello World* (рис. 1.25):

```
response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("Hello World");
```

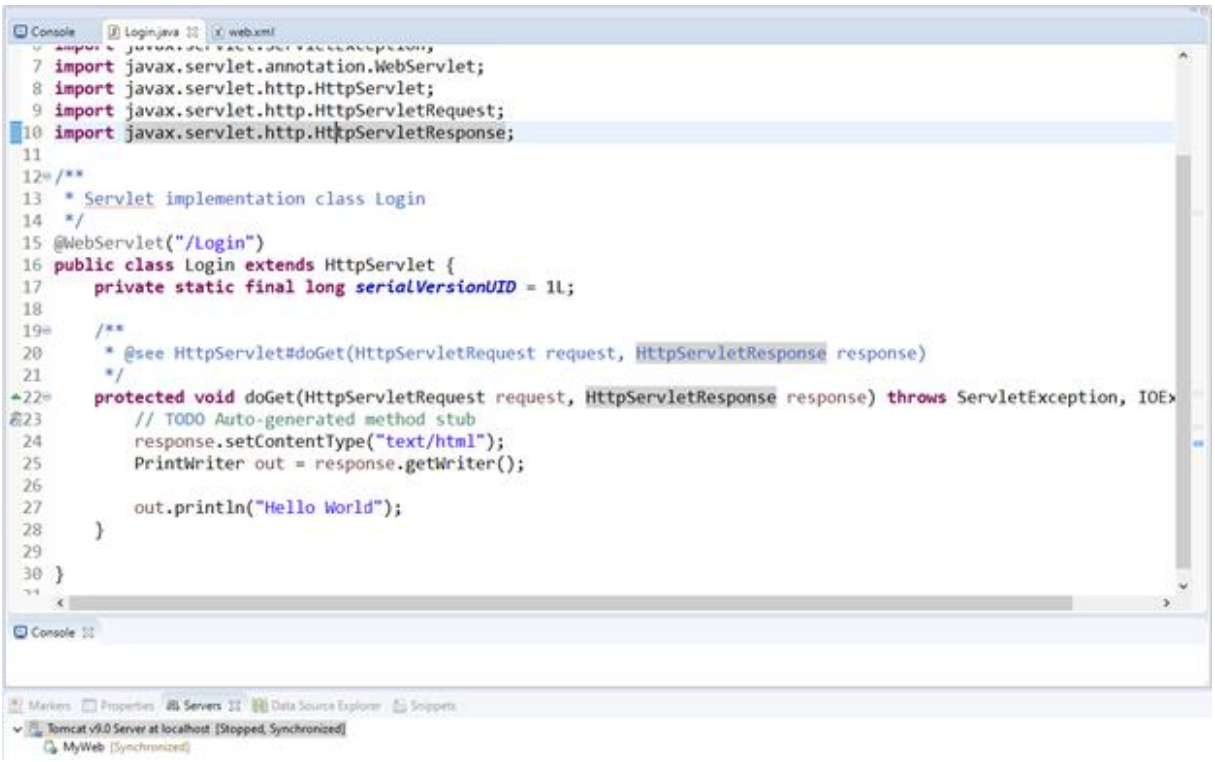


Рис. 1.25 – Программный код сервлета на Java

Это и есть код нашего сервлета. Мы создаем наследника класса `HttpServlet`. В нем реализуем один метод `doGet()`, в котором вначале сообщаем, что сервлет будет выдавать HTML-документ. У метода есть входящие параметры, типы которых – `HttpServletRequest` (запрос со стороны клиента) и `HttpServletResponse` (ответ со стороны сервера).

В строке `response.setContentType("text/html")` мы сообщаем о том, что отображать мы будем страничку как HTML.

Потом вытаскиваем из `response` ссылку на экземпляр PrintWriter. И то, что мы будем писать в нем, будет отдано сервером на запрос от клиента.

Аннотация `@WebServlet("/Login")` указывает на то, что данный сервлет будет доступен по адресу <URL>/Login.

Для того чтобы запустить сервлет, нам нужно отредактировать дескриптор развертывания.

1.2.3 Написание дескриптора развертывания

Первым шагом к созданию приложения является определение так называемого дескриптора развертывания. Он указывает, как приложение должно быть развернуто в определенной среде. Когда дело касается web-приложений, дескриптор развертывания представляет собой простой XML-файл, называемый web.xml, и является частью стандартной спецификации Java EE (рис. 1.26).

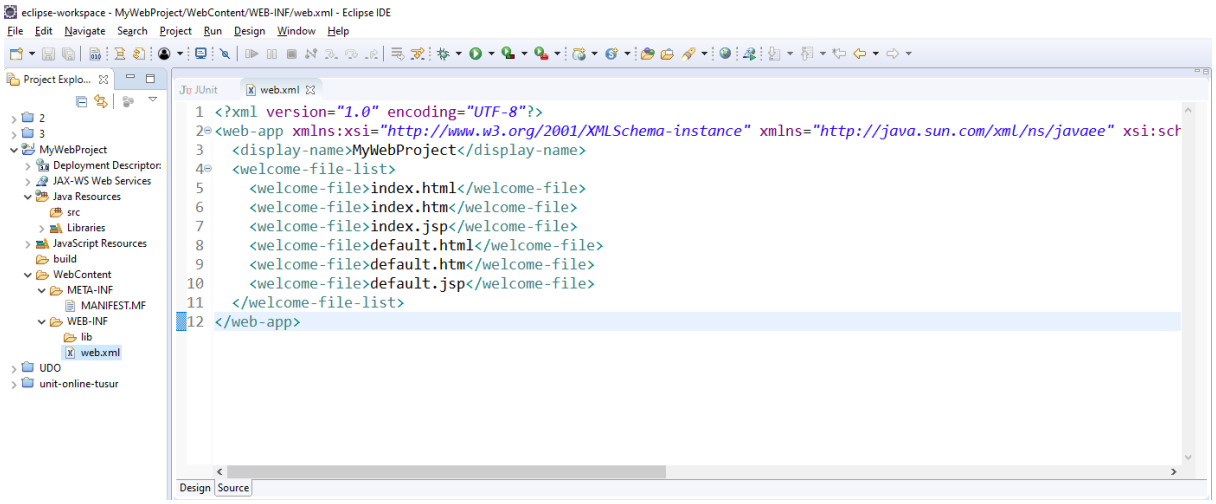


Рис. 1.26 – Программный код web.xml

Web.xml – это файл, где вы

1) регистрируете и создаете отображение URL на ваши сервлеты;

2) регистрируете и задаете любые фильтры и слушатели приложения;

3) задаете начальные параметры контекста в виде пар имя/значение;

4) конфигурируете страницы ошибок;

5) указываете начальные файлы приложения;

6) задаете время простоя сеанса (тайм-аут);

7) задаете настройки безопасности, управляющие тем, кто к каким web-компонентам он может обращаться.

Это только основная часть настроек, которые можно задать в web.xml.

Вставим:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/ja-
vaee http://xmlns.jcp.org/xml/ns/javaee/web-
app_4_0.xsd" id="WebApp_ID" version="4.0">
  <display-name>MyWeb</display-name>
  <Servlet>
    <Servlet-name>Login</Servlet-name>
    <Servlet-class>com.Login</Servlet-class>
  </Servlet>
  <Servlet-mapping>
    <Servlet-name>Login</Servlet-name>
    <url-pattern>/</url-pattern>
</Servlet-mapping>
</web-app>
```

В теге `Servlet-class` мы отмечаем запускаемый класс сервлета, а в теге `url-pattern` указываем `url` – имя запускаемого сервлета.

Далее обязательно перезапустим сервер (рис. 1.27).

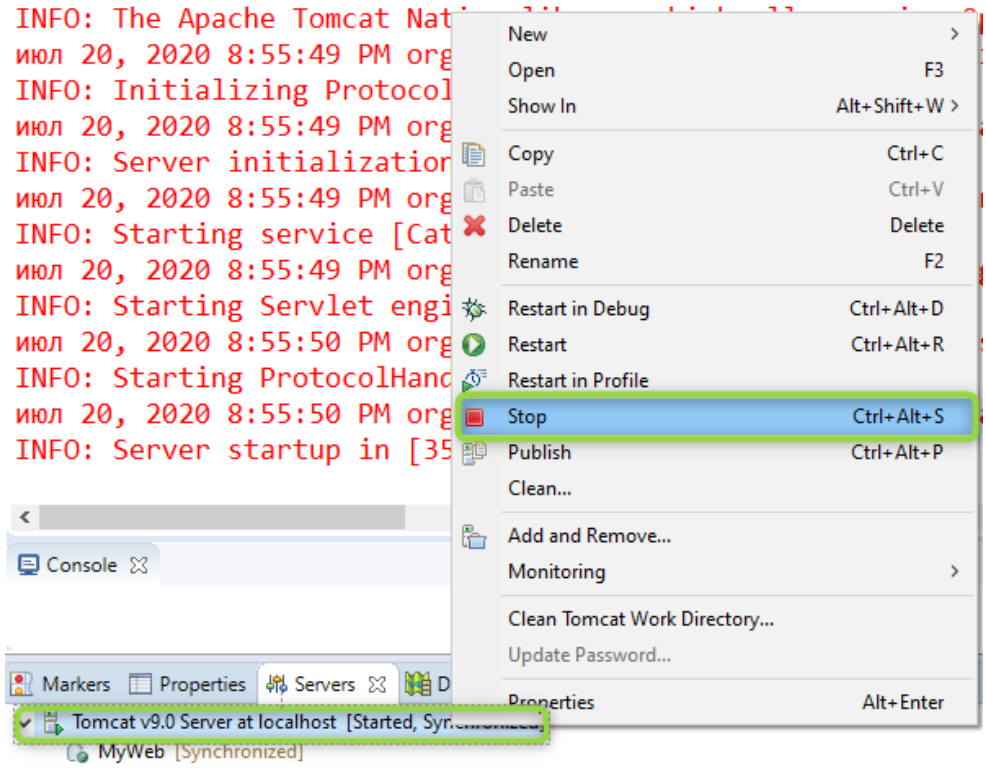


Рис. 1.27 – Остановка сервера

Далее стартуем (рис. 1.28).

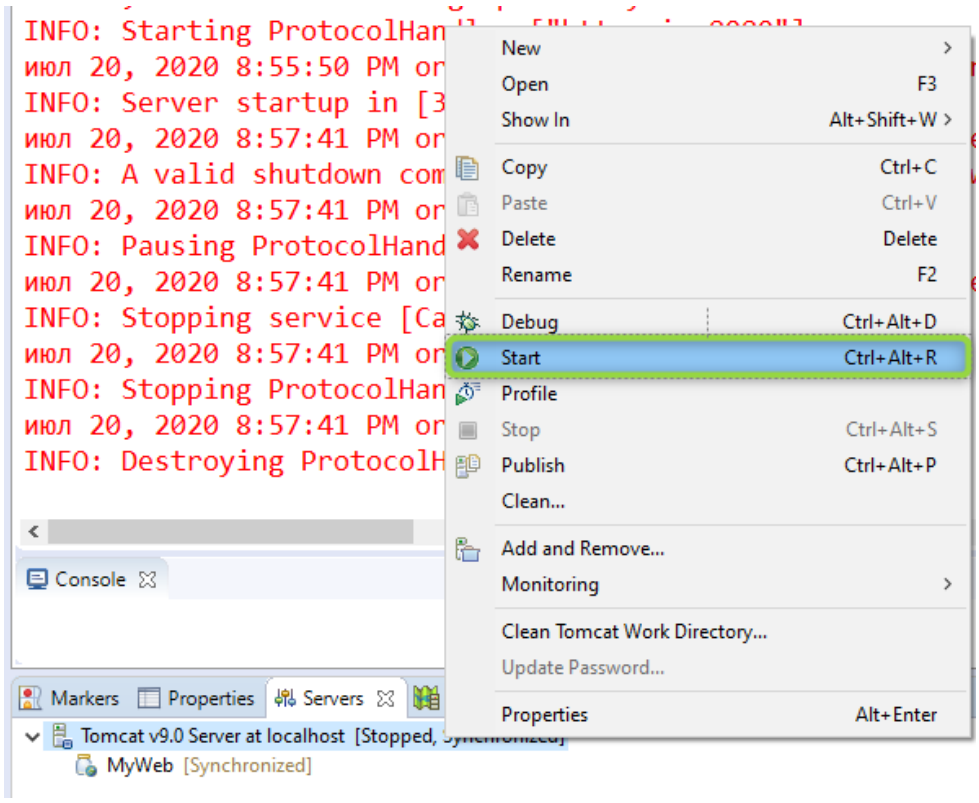


Рис. 1.28 – Запуск сервера

Запускаем сервлет (рис. 1.29).

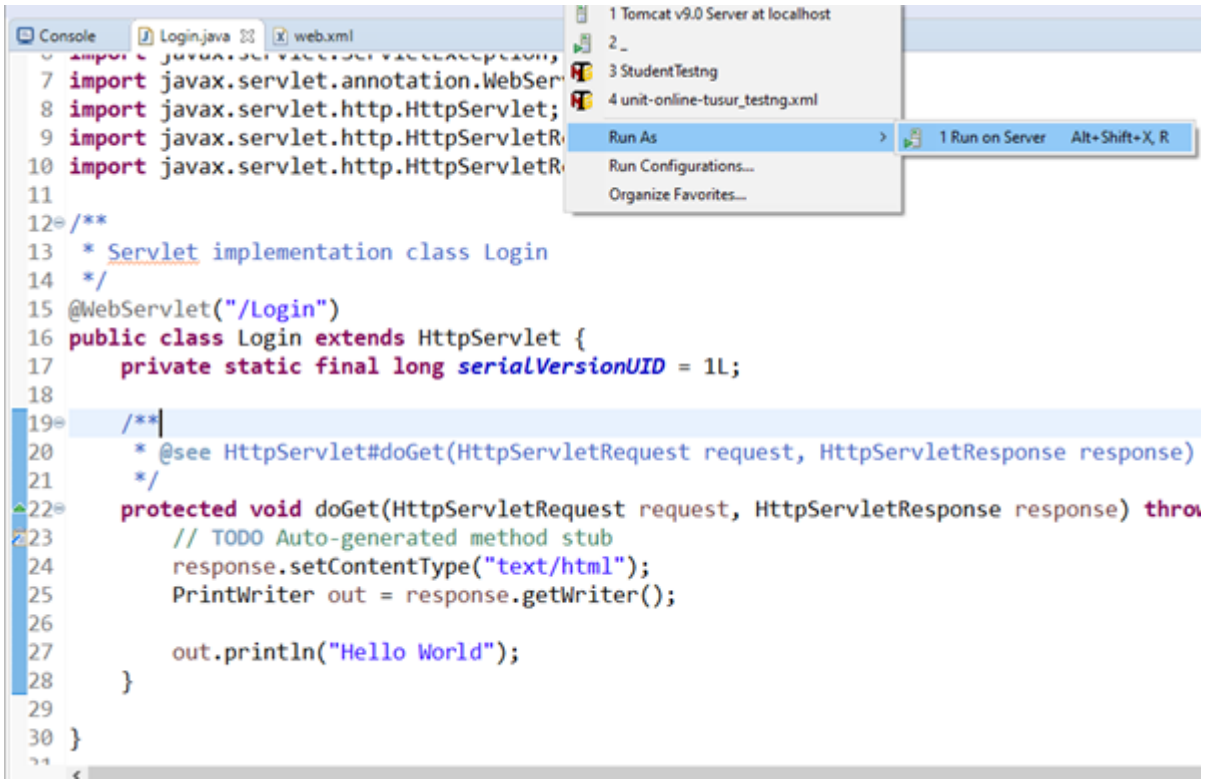


Рис. 1.29 – Запуск сервлета

Если вы увидели в браузере надпись *Hello World* (рис. 1.30), то запуск прошел удачно.

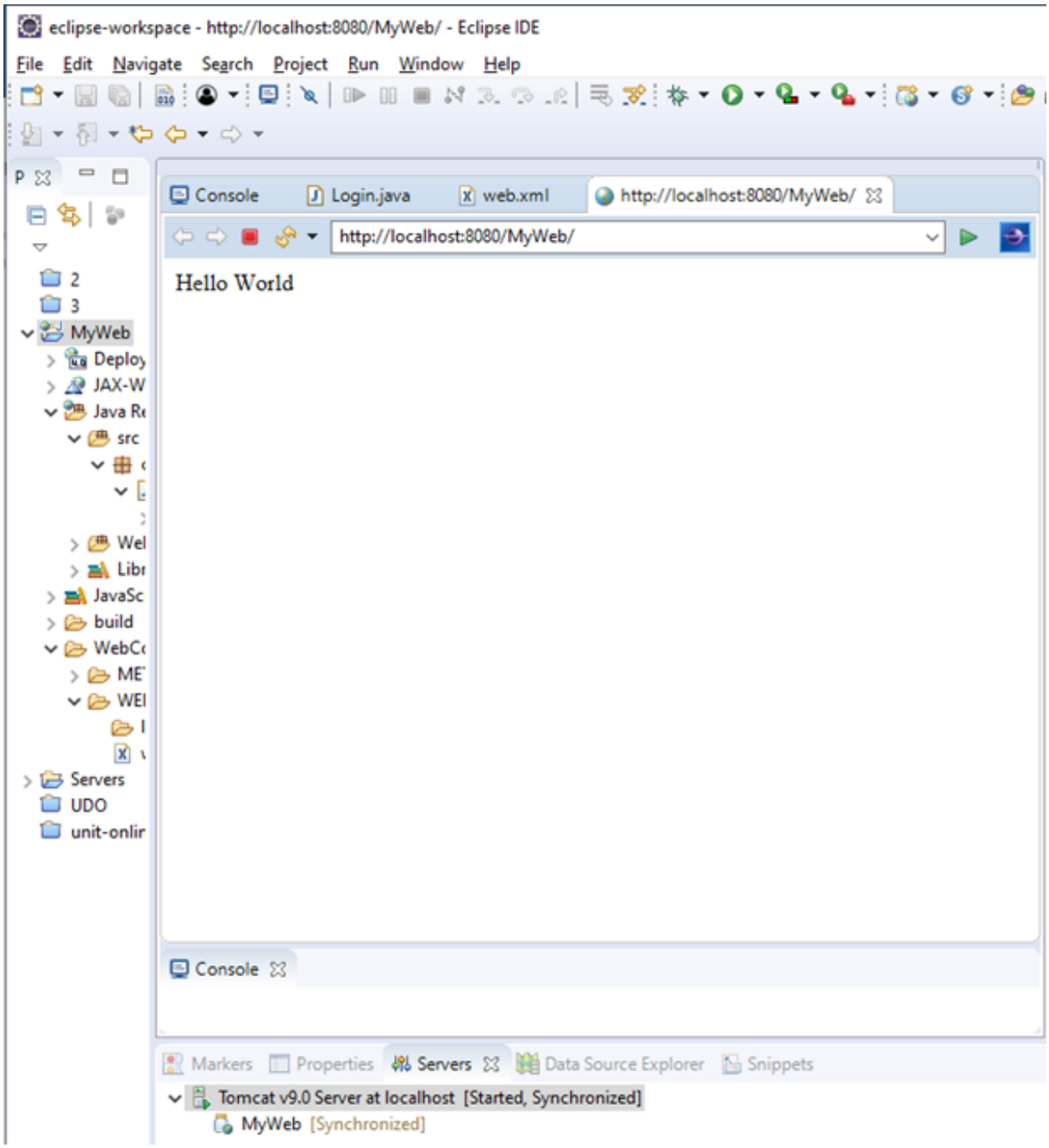


Рис. 1.30 – Результат работы сервлета

В строке http://localhost:8080/MyWeb/ *8080* – это порт по умолчанию, который занимает Tomcat.

Если же вы увидели ошибку 404 или 500, то что-то сделали не так.

1.2.4 Обработка запросов HTTP

Далее попробуем обработать запросы. В классе `HttpServlet` определены методы `doGet()` и `doPost()` для реакции на запросы типа `GET` и `POST` клиента. Эти методы вызываются методом `service()` класса `HttpServlet`, который, в свою очередь, вызывается при поступлении запроса на сервер.

Для начала создадим страницу JSP *mypage.jsp* с формой для ввода имени. Далее сделаем JSP-страницу *mypage2.jsp*, на которую будет осуществляться переход при нажатии кнопки. На этой странице будем отображать приветствие по имени.

Чтобы добавить к проекту web-страницу, идем в папку *WebContent*, вызываем контекстное меню и выбираем пункт *JSP File* (рис. 1.31). В появившемся окне в поле *File name* пишем название файла *mypage.jsp*, в следующем окне выбираем *New* → *JSP* и жмем *Finish*.

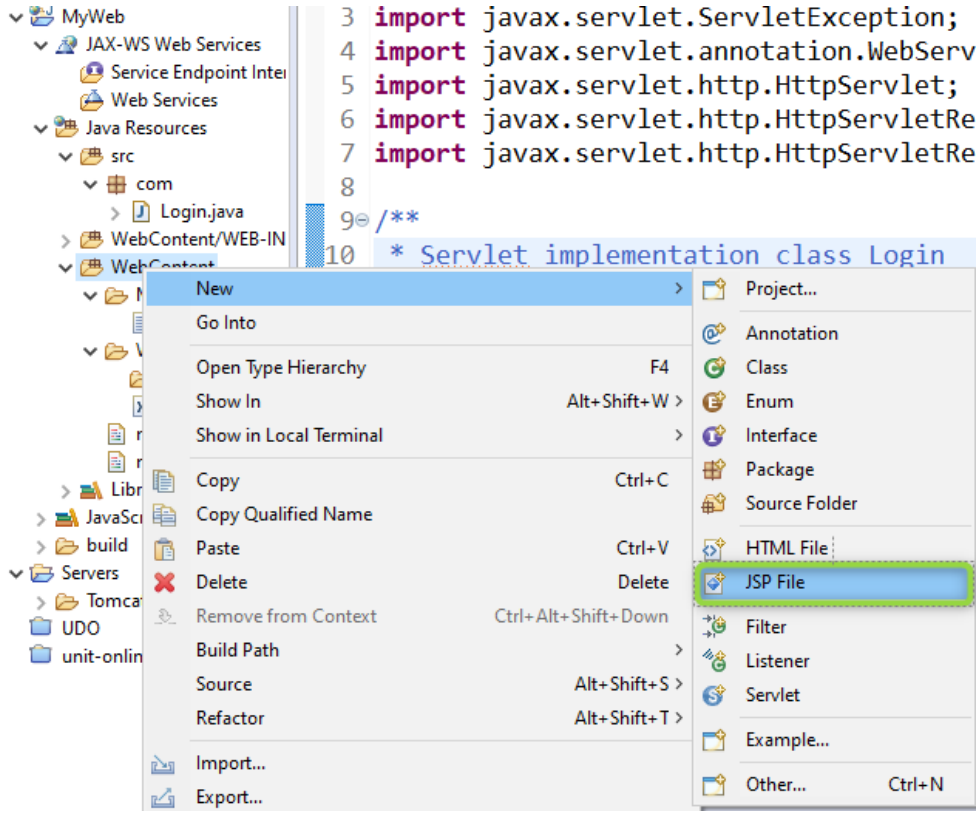


Рис. 1.31 – Создание JSP

Создадим *mypage.jsp* (рис. 1.32).

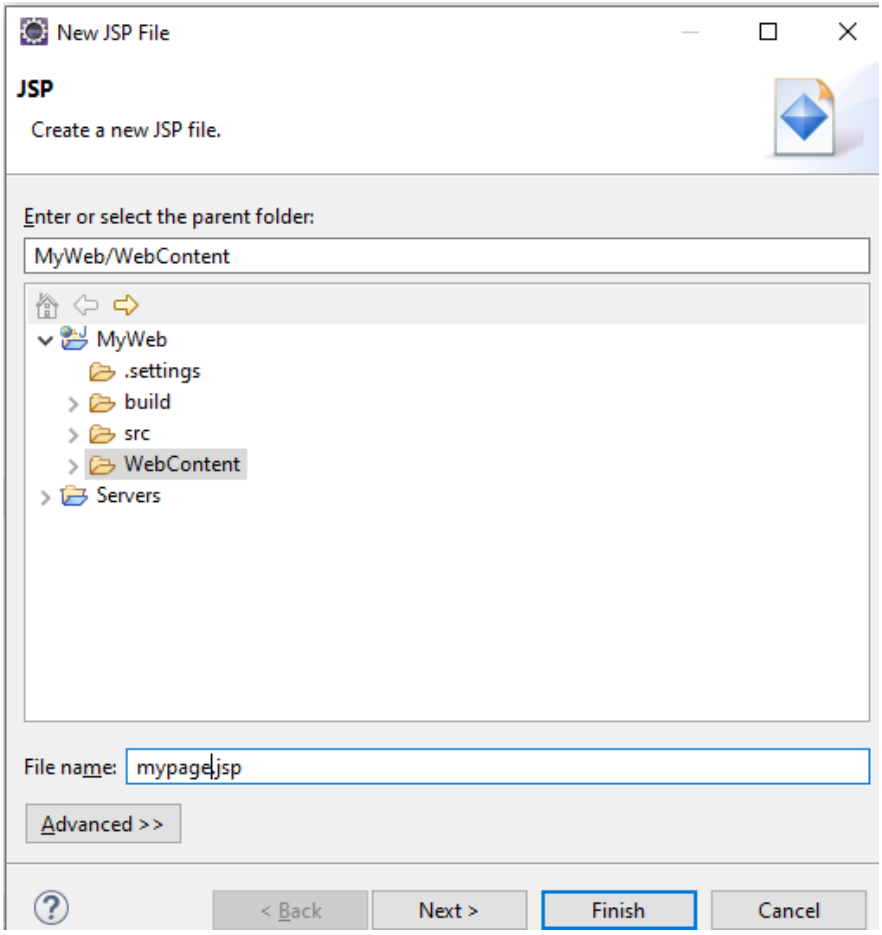


Рис. 1.32 – Создание JSP

Запишем в *mypage.jsp*:

```
<%@ page language="java" contentType="text/html; char-
set=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
```

```
<body>
<form action="Login" method="POST">
Name:<input type="text" name="user"/>
<input type="submit" value="Войти"/>
</form>
</body>
</html>
```

Далее также создадим еще одну страницу *mypage2.jsp* для передачи введенного имени сервлетом. Чтобы принять данные на JSP, нужно обратится к ключу по определенной конструкции `${name}`.

Запишем код в *mypage2.jsp*:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<h1>Welcome, ${name}</h1>
</body>
</html>
```

Добавим эти файлы в приветствие в *web.xml*:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"   xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
```

```
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
id="WebApp_ID" version="4.0">
  <display-name>MyWeb</display-name>
  <Servlet>
    <Servlet-name>Login</Servlet-name>
    <Servlet-class>com.Login</Servlet-class>
  </Servlet>
  <Servlet-mapping>
    <Servlet-name>Login</Servlet-name>
    <url-pattern>/</url-pattern>
</Servlet-mapping>
   <welcome-file-list>
    <welcome-file>mypage.jsp</welcome-file>
    <welcome-file>mypage2.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

Servlet с url-pattern = /

Этот сервлет по умолчанию будет использоваться для обработки запроса (request), который имеет ссылку, не совпадающую ни с каким url-pattern другого сервлета, объявленного в вашем приложении. На рисунке ниже показано, как *WebContainer* решает, какой сервлет использовать для обслуживания запроса от клиента (рис. 1.33) [4].

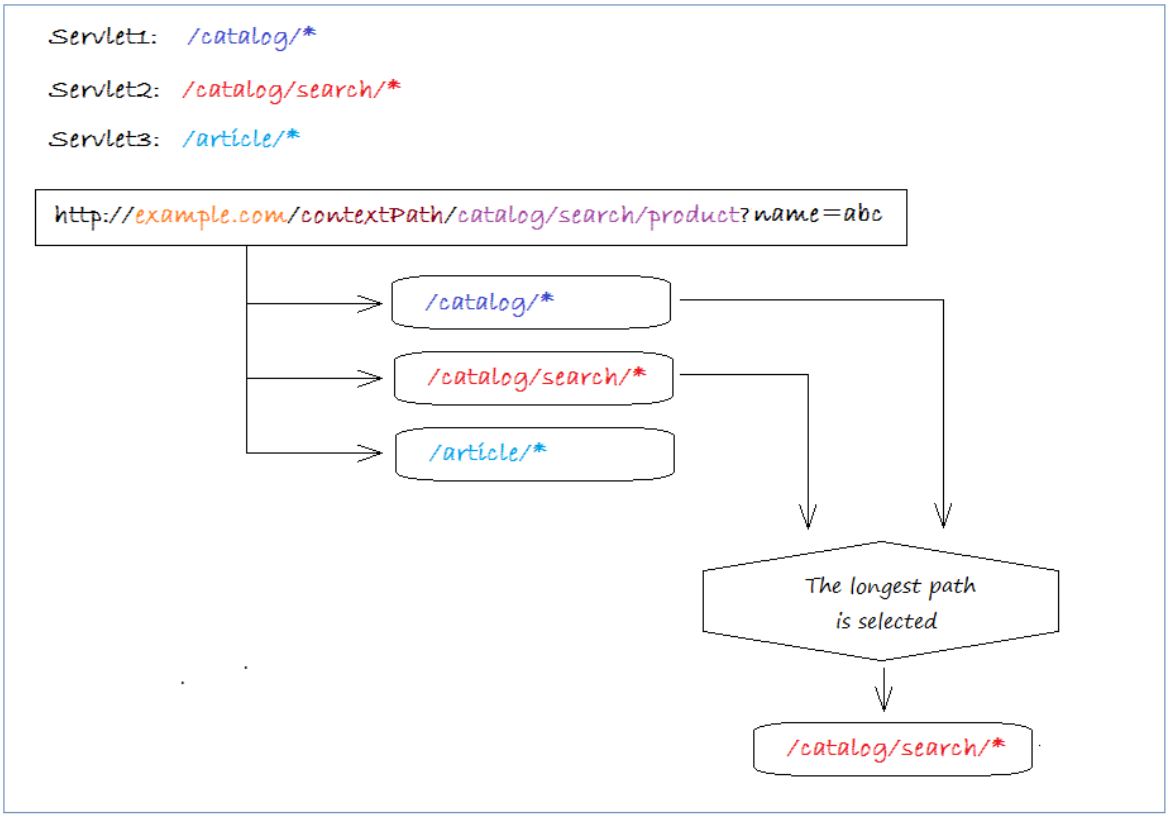


Рис. 1.33 – Конфигурация ссылки для сервлета

Укажем его в аннотации:

```
@WebServlet(urlPatterns = "/Login")
```

Далее перепишем сервлет:

```
package com;
import java.io.IOException;
import javax.Servlet.ServletException;
import javax.Servlet.annotation.WebServlet;
import javax.Servlet.http.HttpServlet;
import javax.Servlet.http.HttpServletRequest;
import javax.Servlet.http.HttpServletResponse;

/**
 * Servlet implementation class Login
 */
```

```
@WebServlet(urlPatterns = "/Login")
public class Login extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Login() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
    request.getServletContext().getRequestDispatcher("/mypage.jsp").forward(request, response);
    }
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    String inputName= request.getParameter("user");
    request.setAttribute("name", inputName);
```

```
        request.getServletContext().getRequestDis-
patcher("/mypage2.jsp").include(request, response);
    }
        }
```

Методы `doGet()` и `doPost()` принимают в качестве параметров объекты `HttpServletRequest` и `HttpServletResponse`, которые дают возможность осуществлять взаимодействие между клиентом и сервером. Методы интерфейса `HttpServletRequest` облегчают доступ к данным запроса. Методы интерфейса `HttpServletResponse` облегчают возврат результатов web-клиенту в виде HTML.

Для того чтобы выполнить перенаправление запроса, вначале с помощью метода `getServletContext()` получаем объект `ServletContext`, который представляет контекст запроса. Затем с помощью его метода `getRequestDispatcher()` получаем объект `RequestDispatcher`. Путь к ресурсу, на который надо выполнить перенаправление, передается в качестве параметра в `getRequestDispatcher`.

Затем у объекта `RequestDispatcher` вызывается метод `forward()`, в который передаются объекты `HttpServletRequest` и `HttpServletResponse`. Метод `forward()` класса `RequestDispatcher` позволяет перенаправить запрос из сервлета на другой сервлет, HTML-страницу или страницу JSP. Также можно использовать метод `include()` в `RequestDispatcher` для включения содержимого в страницу. И если мы обратимся к сервлету, то фактически мы получим содержимое страницы, который будет перенаправлен запрос.

`String inputName= request.getParameter("user");` – получим введенное имя пользователя в текстовое поле по имени **`user`**.

`request.setAttribute("name", inputName);` – далее установим полученное значение атрибуту **`${name}`** в *mypage2.jsp*.

Перезапустим сервлет (рис. 1.34).

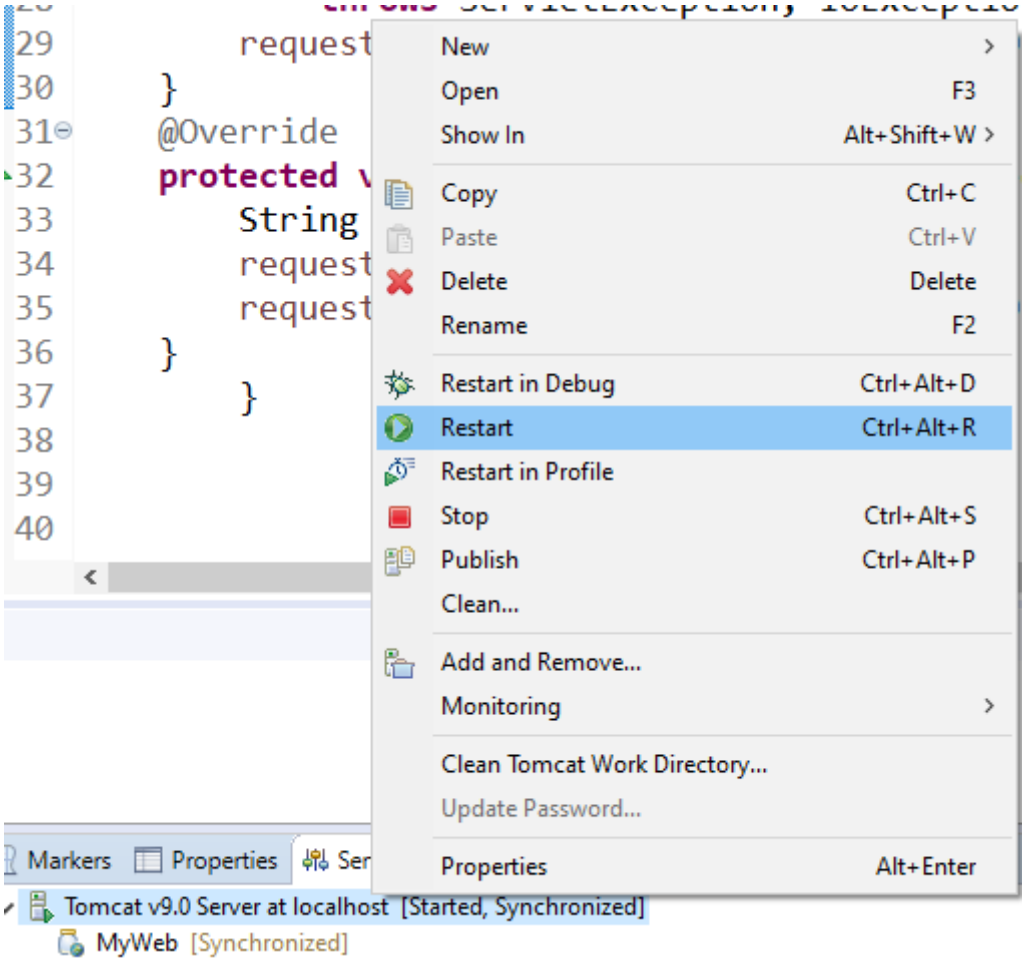


Рис. 1.34 – Перезапуск сервлета

Сначала стартует страница *mypage.jsp* (рис. 1.35), после ввода имени и нажатия кнопки *Войти* откроется страница *mypage2.jsp* (рис. 1.36).

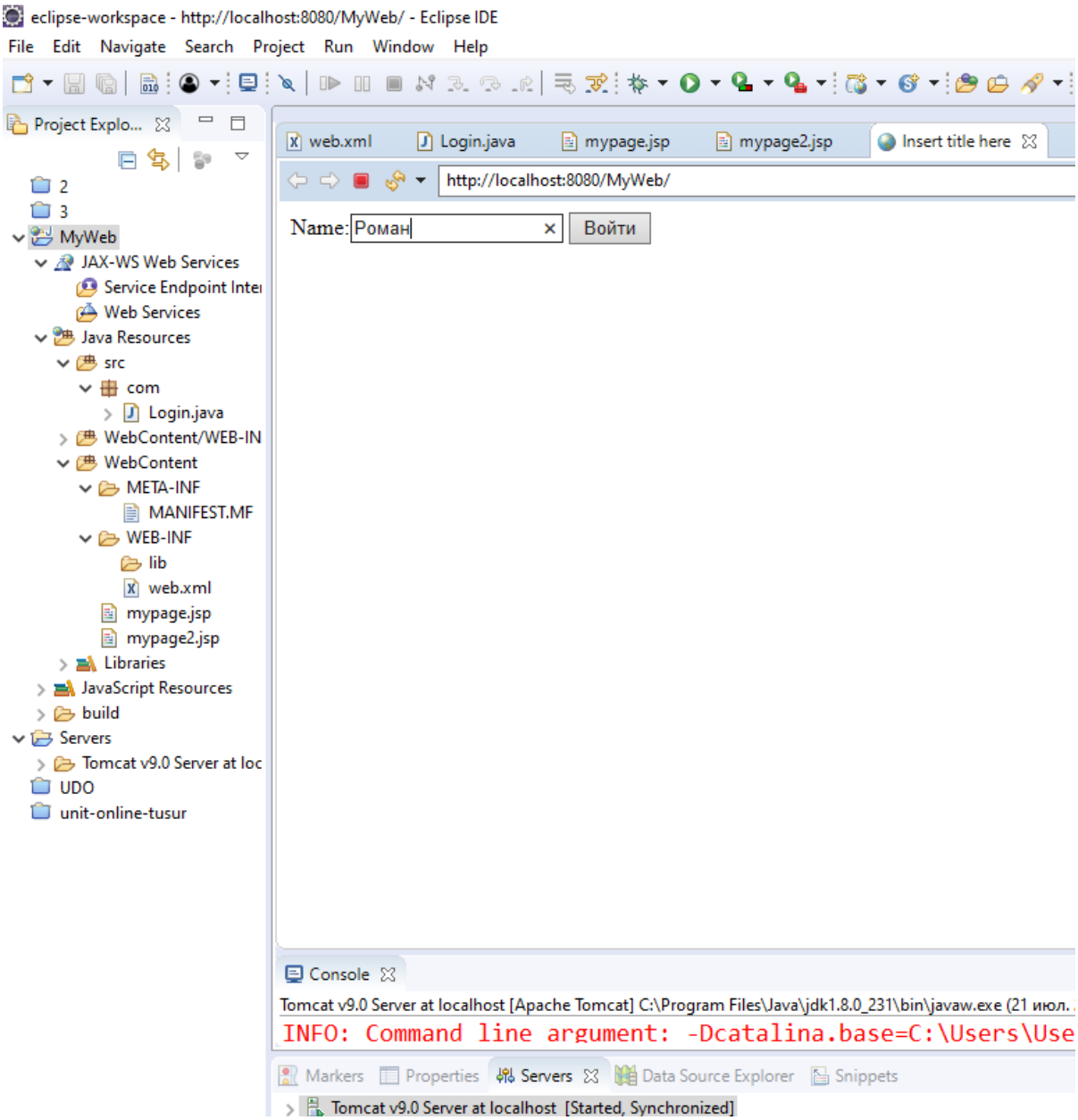


Рис. 1.35 – Открытие страницы *mypage.jsp*

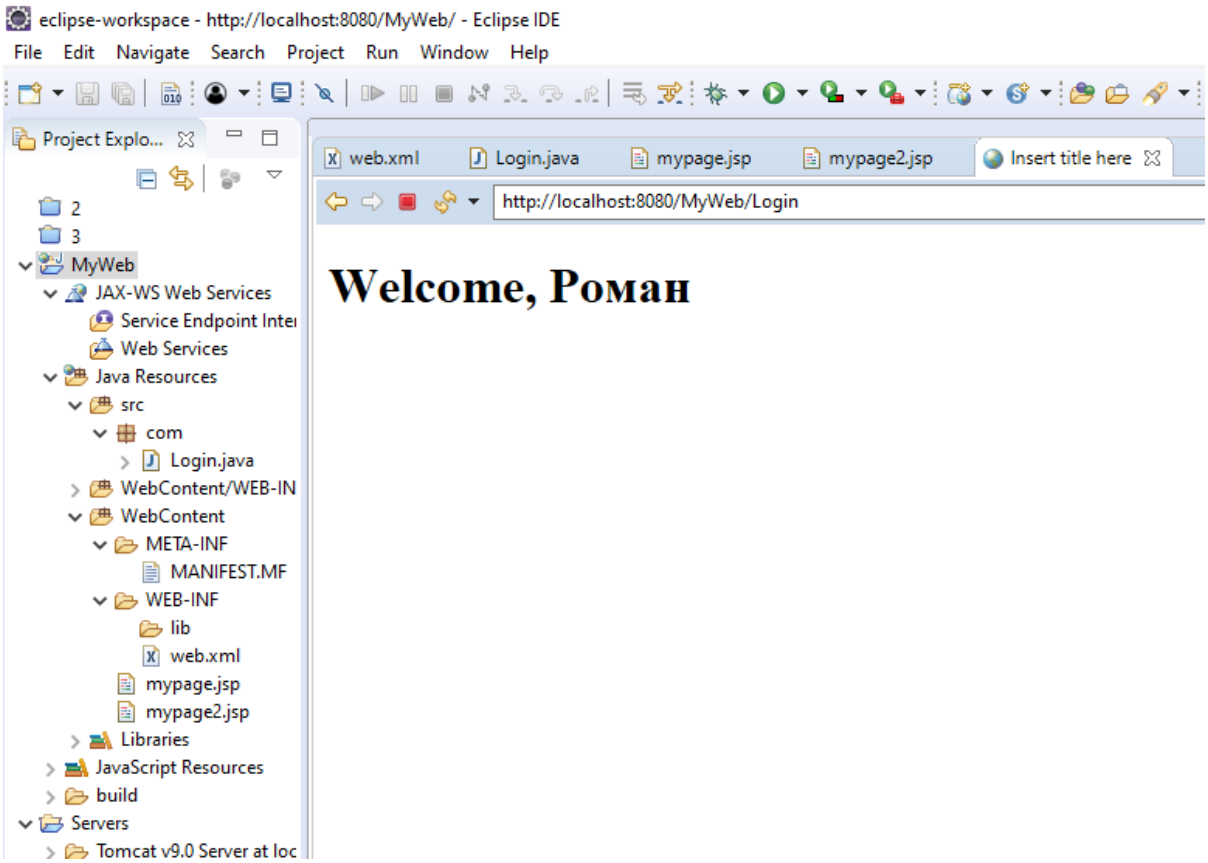


Рис. 1.36 – Открытие страницы *mypage2.jsp*

1.2.5 Общее задание

1. Изучить технологию Java Servlets.
2. Описать жизненный цикл работы сервлетов.
3. Разработать сервлет по варианту.
4. Разработать интерфейс и связать его с сервлетом.
5. Готовое web-приложение разместить на сервере Tomcat.
6. Отправить на проверку проект с отчетом в архиве.

1.2.6 Задание по вариантам

Задание по вариантам предполагает создание страницы с аутентификацией для своего проекта. Необходимо предусмотреть как минимум два вида пользователей с разными итоговыми страницами, на которые осуществляется переход в зависимости от введенного логина и пароля.

Спроектировать обработку ввода неправильного логина или пароля с выводом соответствующего сообщения как о несуществующем логине, так и о вводе несоответствующего пароля для выбранного логина.

Выбор варианта лабораторной работы осуществляется по общим правилам с использованием следующей формулы:

$$V = (N \times K) \text{ div } 100,$$

где V – искомый номер варианта,

N – общее количество вариантов,

K – код варианта,

div – целочисленное деление.

При $V = 0$ выбирается максимальный вариант.

Варианты проектов:

1. Сайт для салона красоты.
2. Сайт для больницы.
3. Сайт для библиотеки.
4. Сайт для кинотеатра.
5. Образовательный сайт по ИТ-программированию.
6. Новостной сайт.
7. Сайт для аптеки.
8. Сайт для проката автомобилей.
9. Сайт для факультета.
10. Сайт для школы.
11. Сайт для хостинга файлов.
12. Сайт для техподдержки пользователей.
13. Сайт для прогноза погоды.
14. Сайт «Доска объявлений».
15. Сайт «Планета Зоо».

1.2.7 Вопросы для самоконтроля

1. Что такое сервлет?
2. Какие еще существуют технологии, похожие на сервлеты?
3. Какова структура каталогов web-приложения?
4. Какой класс является базовым для сервлетов?
5. Каков жизненный цикл у сервлета?
6. Каким образом отправить ответ клиенту?

1.3 Требования к содержанию и оформлению отчета

После выполнения работы студент должен представить отчет о проделанной работе с описанием полученных результатов и выводов. Отчет по лабораторным работам должен содержать следующие структурные части:

- титульный лист (1 стр.);
- содержание (1 стр.);
- цели и задачи лабораторной работы (1 стр.);
- формулировка индивидуального задания;
- перечень библиотек и основных классов, методов, использованных в программе;
- результаты работы программы в виде скриншота;
- анализ полученных результатов;
- выводы о проделанной работе;
- листинг программы с комментариями;
- приложения.

Оформление отчета должно соответствовать требованиям образовательного стандарта вуза ОС ТУСУР 01–2013 «Работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления» (https://regulations.tusur.ru/documents/70).

Образец титульного листа представлен в приложении А. Изложение должно быть последовательным, логичным, конкретным. В текст отчета могут быть включены небольшие фрагменты программного кода, обязательно с комментариями. Рекомендуемый шрифт для выполнения фрагмента кода – Courier New, размер 12 пт. В тексте отчета должны быть даны ссылки на использованные материалы с указанием номера источника по списку литературы. В приложениях размещаются листинг с комментариями, схемы программы, скриншоты интерфейса. Приложения нумеруются русскими буквами в порядке появления ссылок на них в основном тесте документа.

2 МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОРГАНИЗАЦИИ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

2.1 Общие положения

Целью самостоятельной работы являются систематизация, расширение и закрепление теоретических знаний, использование материала, собранного и полученного в ходе самостоятельной подготовки к лабораторным работам.

Самостоятельная работа включает в себя изучение тем теоретической части курса, подготовку к лабораторной и контрольной работам, оформление отчета по лабораторной работе.

2.2 Проработка теоретического материала и подготовка к контрольной работе

Изучение теоретической части дисциплины имеет целью не только получение и закрепление новых знаний, но и развитие у студентов творческих навыков, инициативы и умения организовать свое время.

Проработка теоретического материала включает:

- усвоение теоретического материала курса и изучение рекомендованных источников;
- подготовку к контрольной работе;
- подготовку ответов на вопросы по различным темам дисциплины в той последовательности, в какой они представлены.

Планирование времени, необходимого на изучение дисциплин, студентам лучше всего осуществлять весь семестр, предусматривая при этом регулярное повторение материала.

При изучении дисциплины необходимо по каждой теме прочитать рекомендованную литературу и составить краткий конспект основных положений,

терминов, сведений, требующих запоминания и являющихся основополагающими в этой теме для освоения последующих тем курса. Для расширения знаний по дисциплине следует использовать интернет-ресурсы, рекомендованные преподавателем.

Задачи, стоящие перед студентом при подготовке и написании контрольной работы:

- закрепление полученных ранее теоретических знаний;
- выработка навыков самостоятельной работы;
- выяснение подготовленности к итоговой аттестации.

2.3 Подготовка к лабораторной работе

Проведение лабораторной работы включает в себя следующие этапы:

- определение задач работы;
- определение порядка работы или отдельных ее этапов;
- непосредственное выполнение работы студентами;
- подведение итогов и формулирование основных выводов;
- оформление отчета.

При подготовке к лабораторной работе необходимо заранее изучить методические рекомендации, обратить внимание на цель занятия, основные вопросы для подготовки к занятию, содержание темы занятия.

ЛИТЕРАТУРА

1. Сысолетин, Е. Г. Разработка интернет-приложений [Электронный ресурс] : учеб. пособие для среднего профессионального образования / Е. Г. Сысолетин, С. Д. Ростунцев. – М. : Юрайт, 2020. – 90 с. – (Профессиональное образование) // ЭБС «Юрайт» [сайт]. – Режим доступа: https://urait.ru/bcode/456393 (дата обращения: 22.07.2020).

2. Учебное пособие по J2EE. Обзор [Электронный ресурс] // Сайт CodeNet. – Режим доступа: http://www.codenet.ru/webmast/java/j2ee.php (дата обращения: 22.07.2020).

3. Глава 10. Технология сервлетов Java [Электронный ресурс] // Учебное пособие по Java EE 6. – Режим доступа: https://docs.oracle.com/cd/E19798-01/821-1841/bnafd/index.html (дата обращения: 22.07.2020).

4. Руководство Java Servlet для начинающих [Электронный ресурс] // Сайт o7planning.org. – Режим доступа: https://o7planning.org/ru/10169/java-Servlet-tutorial (дата обращения: 22.07.2020).

5. Тузовский, А. Ф. Проектирование и разработка web-приложений [Электронный ресурс] : учеб. пособие для вузов / А. Ф. Тузовский. – М. : Юрайт, 2020. – 218 с. – (Высшее образование) // ЭБС «Юрайт» [сайт]. – Режим доступа: https://urait.ru/bcode/451207 (дата обращения: 22.07.2020).

6. Шилдт, Г. Java 8 : руководство для начинающих : пер. с англ. / Г. Шилдт. – 6-е изд. – М. : ООО «ИД «Вильямс», 2015. – 720 с.

7. Блинов, И. Н. Java 2 : практ. руководство / И. Н. Блинов, В. С. Романчик. – Минск : УниверсалПресс, 2005. – 400 с.

8. Хорстманн, К. Java. Библиотека профессионала : в 2 т. / Кей Хорстманн. – 10-е изд. – М. : ООО «ИД «Вильямс», 2016.

ПРИЛОЖЕНИЕ А

Пример оформления титульного листа отчета

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное

учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизации обработки информации (АОИ)

НАЗВАНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

Отчет по лабораторной работе по дисциплине

«Разработка интернет-приложений»

Вариант ___

Студент гр. _____

_____

(И. О. Фамилия)

«___»_____20_ г.

Руководитель

доцент каф. АОИ,

канд. техн. наук

_____ Ю. В. Морозова

«___»_____20__ г.

Томск 20_