

# IDE Eclipse

## Установка и запуск

Eclipse — сообщество открытого кода или open source, чьи проекты сфокусированы на создании открытой платформы для разработки, развертывания, управления приложениями с использованием различных фреймворков, инструментов и сред исполнения. Загрузить продукты Eclipse можно на официальном сайте <http://eclipse.org/>

Eclipse Java IDE for Web Developers — среда для разработки веб-приложений на Java. До загрузки IDE можно выбрать сборку для конкретной платформы. Ниже будет рассмотрена IDE из сборки Juno для операционной системы Windows (32-bit).

Для установки Eclipse Java IDE необходимо распаковать загруженный архив в ту директорию, в которой будет храниться IDE. Распакованная IDE занимает менее 300 Мб. Среда Eclipse используется для разработки приложений, поэтому предварительно необходимо установить Java Development Kit (JDK). Оптимальную версию можно загрузить с сайта <http://oracle.com/> и следовать мастеру установки.

Если JDK установлен, то для запуска Eclipse Java IDE следует воспользоваться файлом **eclipse.exe**. При запуске файла на экране появится окно (Рис. 1.), где необходимо выбрать **workspace** — каталог, в котором будут храниться

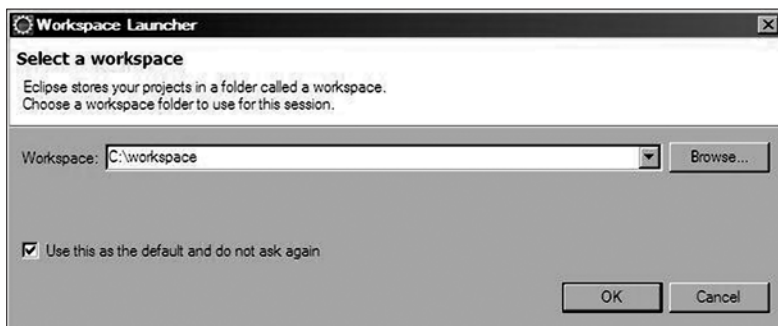


Рис. 1. Выбор директории хранения проектов

проекты. В случае использования одного **workspace**, необходимо установить значение по умолчанию (отметить check box )

## Создание и запуск проекта

При первом запуске проекта на экране появится вкладка *Welcome page* (Рис. 2), содержащая несколько ссылок. При закрытии или сворачивании вкладки *Welcome page*, среда будет выглядеть как на рис. 3.

Создать новый проект в Eclipse IDE можно несколькими способами:

- нажать **Alt+Shift+N**;
- выбрать в меню **File** пункт **New**.

После совершения одного из двух перечисленных действий необходимо выбрать тип проекта. Далее будут подробно рассмотрены два типа проектов: консольный и веб-проект.

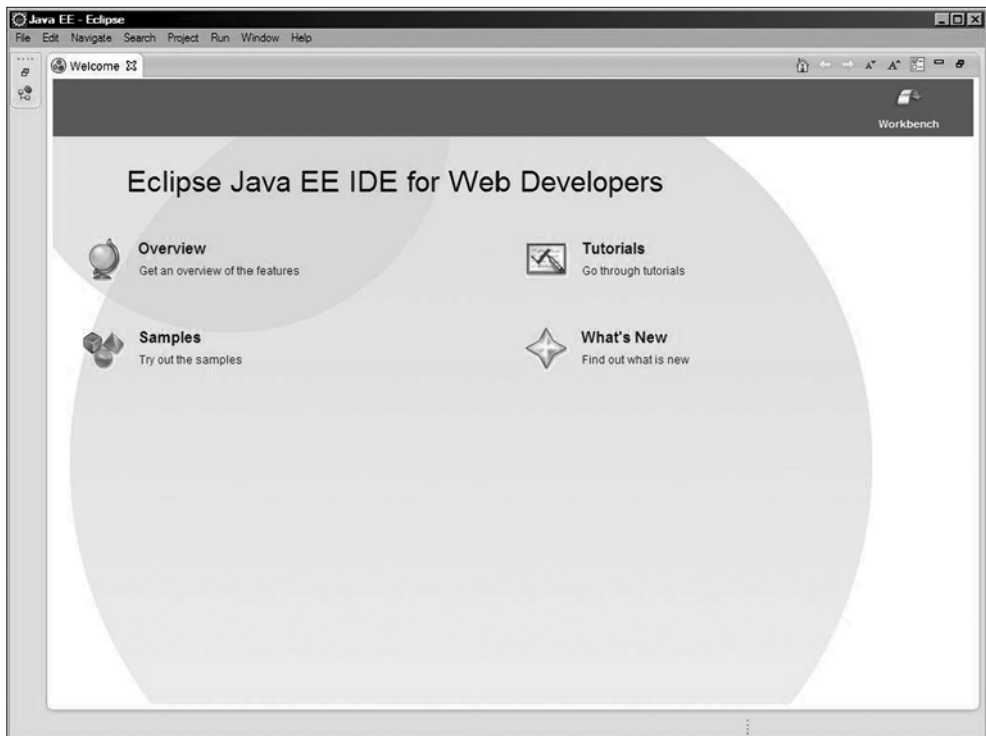


Рис. 2. *Welcome page*

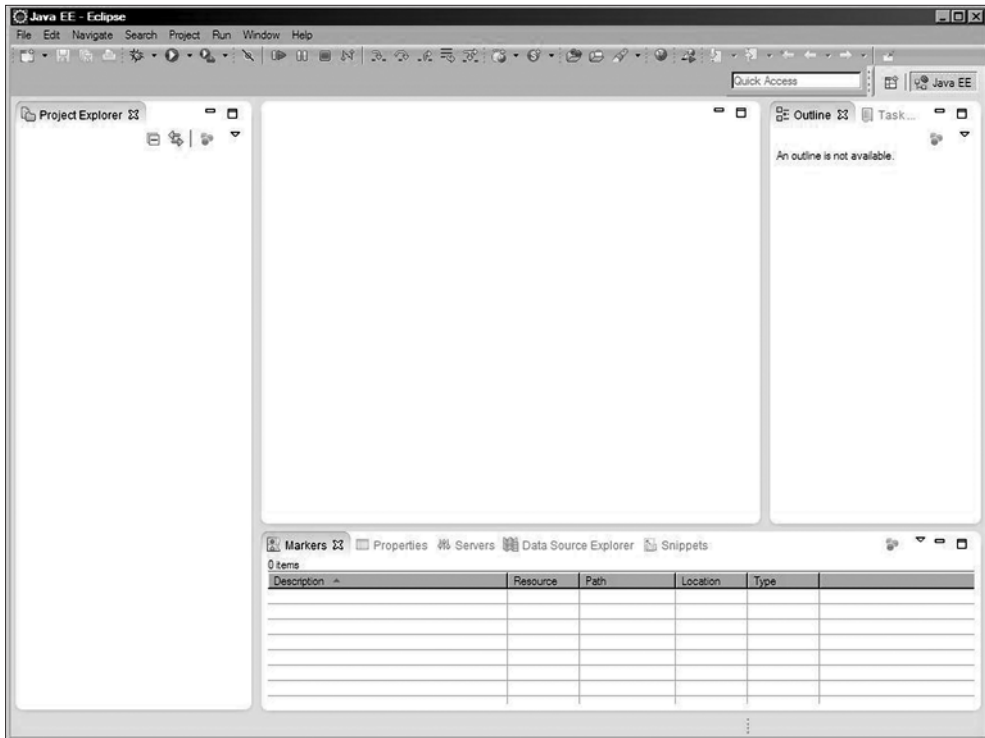


Рис. 3. Eclipse IDE

## Создание, запуск и отладка консольного проекта

Для создания консольного проекта необходимо выбрать меню **File** — **New** — **Project**. В результате выбора откроется мастер создания проекта (Рис. 4.).

В Мастере создания проекта необходимо выбрать **Java Project** и нажать кнопку **Next**. Далее вводится имя проекта, выбирается JRE (Рис. 5.). Можно задать некоторые дополнительные параметры, нажимая кнопку **Next**, можно оставить все остальное по умолчанию и нажать кнопку **Finish** для создания проекта. В *Package explorer* слева появится только что созданный проект (Рис. 6.).

Чтобы создать пакет в новом проекте, необходимо нажать правой кнопкой мыши на название проекта в *Package Explorer* или выбрать в главном меню **File** — **New** — **Package**. Затем ввести имя пакета и нажать кнопку **Finish**. После этого в папке **src** появится пиктограмма пустого пакета с именем. С помощью **File** — **New** можно создать класс (**Class**), интерфейс (**Interface**), перечисление (**Enum**) или другой элемент проекта (Рис. 7.).

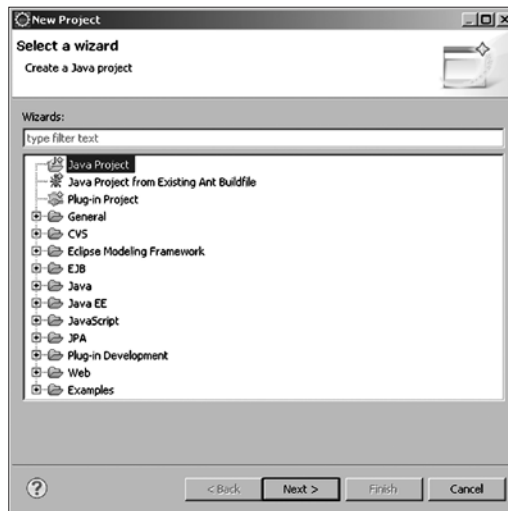


Рис. 4. Мастер создания проекта

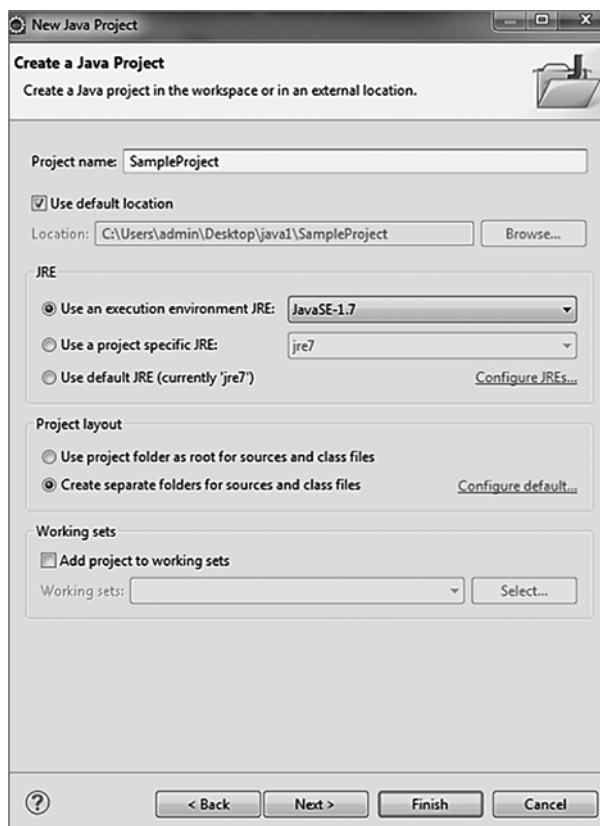


Рис. 5. Создание Project

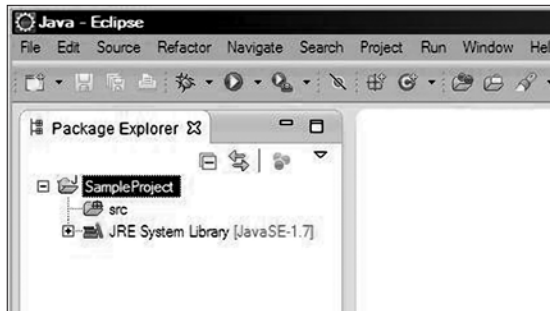


Рис. 6. Package Explorer

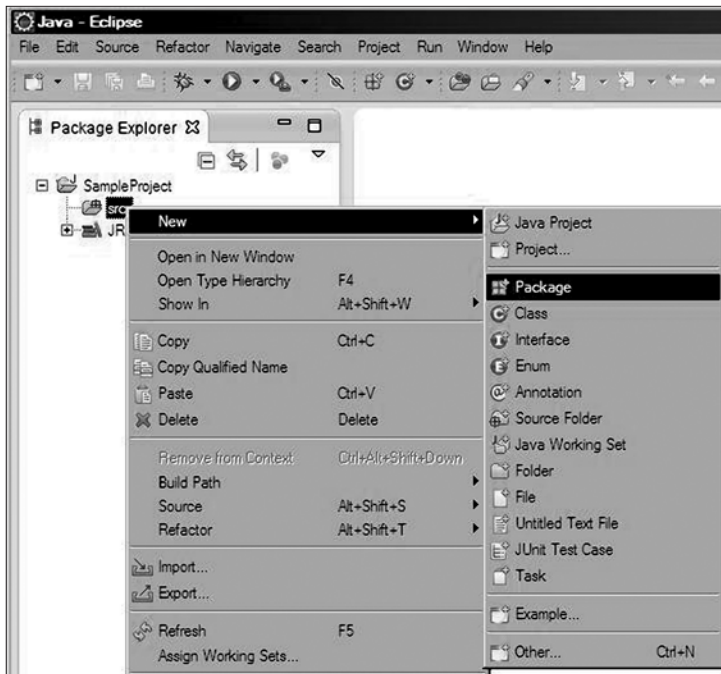


Рис. 7. Создание нового элемента проекта

При создании класса необходимо как минимум ввести его имя. Так же в *Мастере создания класса* можно добавить к классу интерфейсы, указать модификатор доступа, сообщить, что класс будет содержать метод **main()** и прочее (Рис. 8.).

Созданный класс будет выглядеть так (см. Рис. 9.).

Проект можно запустить нажатием сочетания клавиш **Ctrl+F11** или кнопкой **Run** на панели инструментов (зеленый круг с белым треугольником внутри).

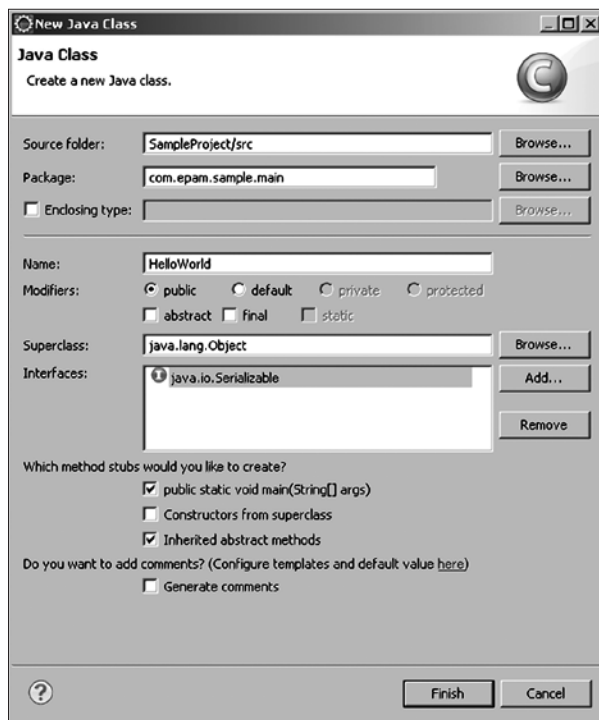


Рис. 8. Создание нового класса

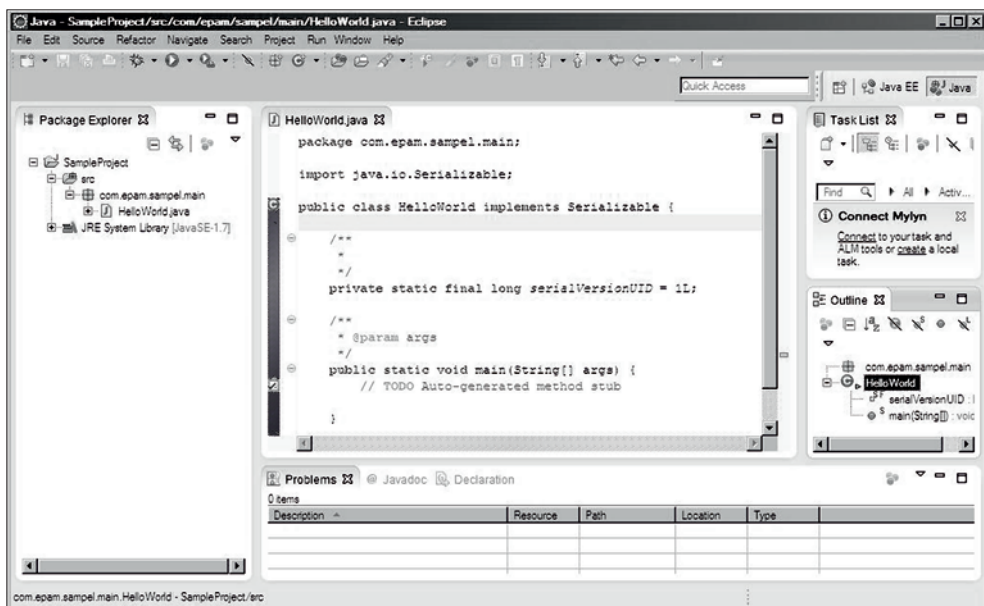


Рис. 9. Сгенерированный класс

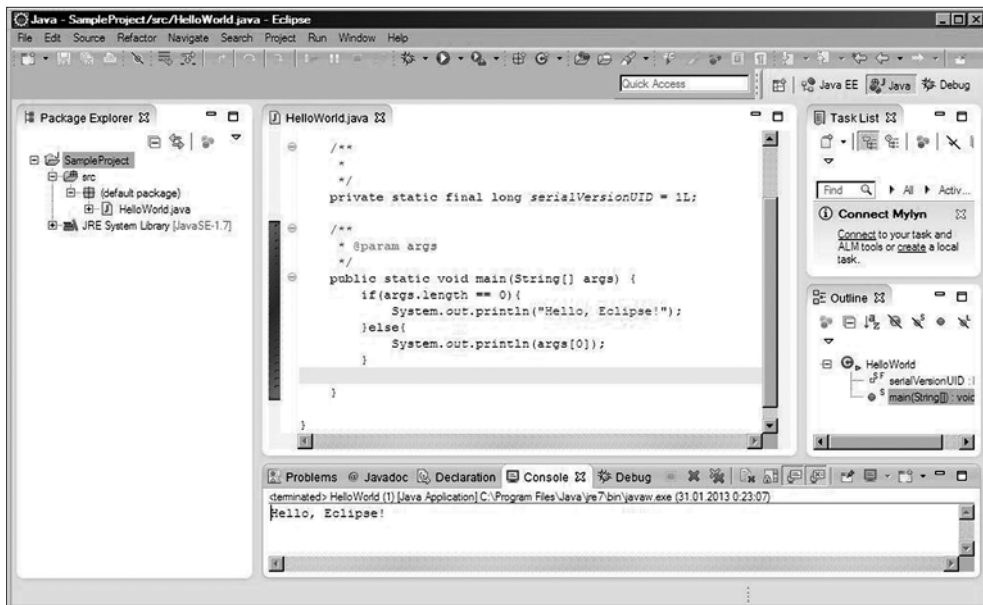


Рис. 10. Результат запуска программы без аргументов

В папке *src* проекта может размещаться только java-код и файлы с расширением *properties*. Все остальные ресурсы проекта всегда должны находиться в проекте, но вне папки *src*.

При разработке может потребоваться временно или постоянно закомментировать участок кода. Выполняется указанное действия после выделения фрагмента кода нажатием сочетания клавиш **Ctrl-/**. Действия по снятию комментария с выделенного фрагмента осуществляется этим же сочетанием клавиш.

Код java имеет для лучшего визуального восприятия лестничную структуру, вид которой можно изменить в настройках редактора. Но в процессе разработки она может быть нарушена программистом, тогда для придания коду форматированного вида следует нажать сочетание клавиш **Ctrl-Shift-F** или выбрать **Source — Format**.

Для передачи аргументов в метод **main()** необходимо выбрать пункт меню **Run — Run Configurations** и далее на вкладке **Arguments** в поле *Program arguments* указать параметры. Параметры разделяются пробелом (Рис. 11.).

Для отладки проект запускается кнопкой **F11** клавиатуры или кнопкой **Debug** панели инструментов (кнопка с изображением жука). Точки останова (breakpoint) ставятся двойным щелчком мыши на сером поле слева от кода класса. Точка останова графически выглядит как маленький голубой шарик.

В режиме отладки среда переключается в следующий вид (Рис. 12.).

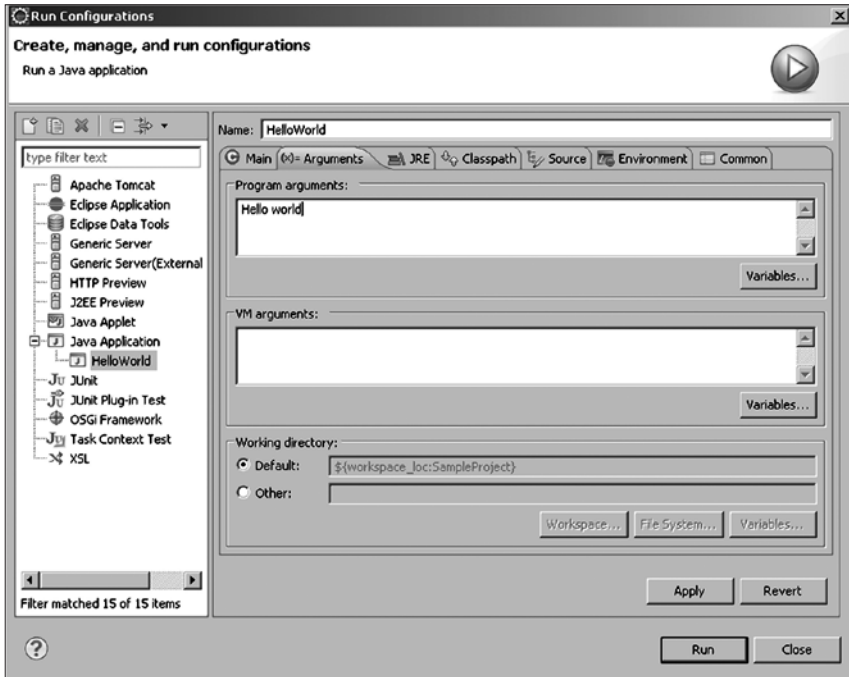


Рис. 11. Передача параметров в main

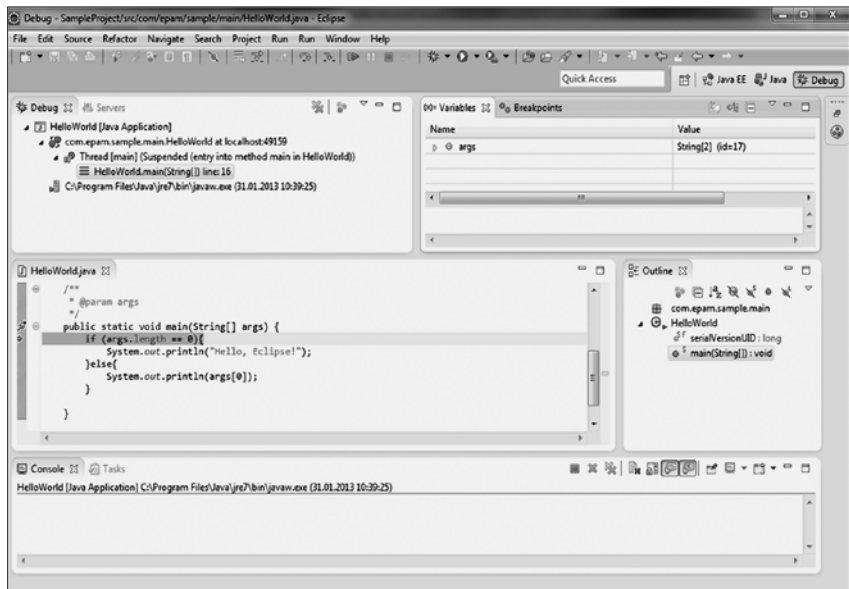


Рис. 12. Debug



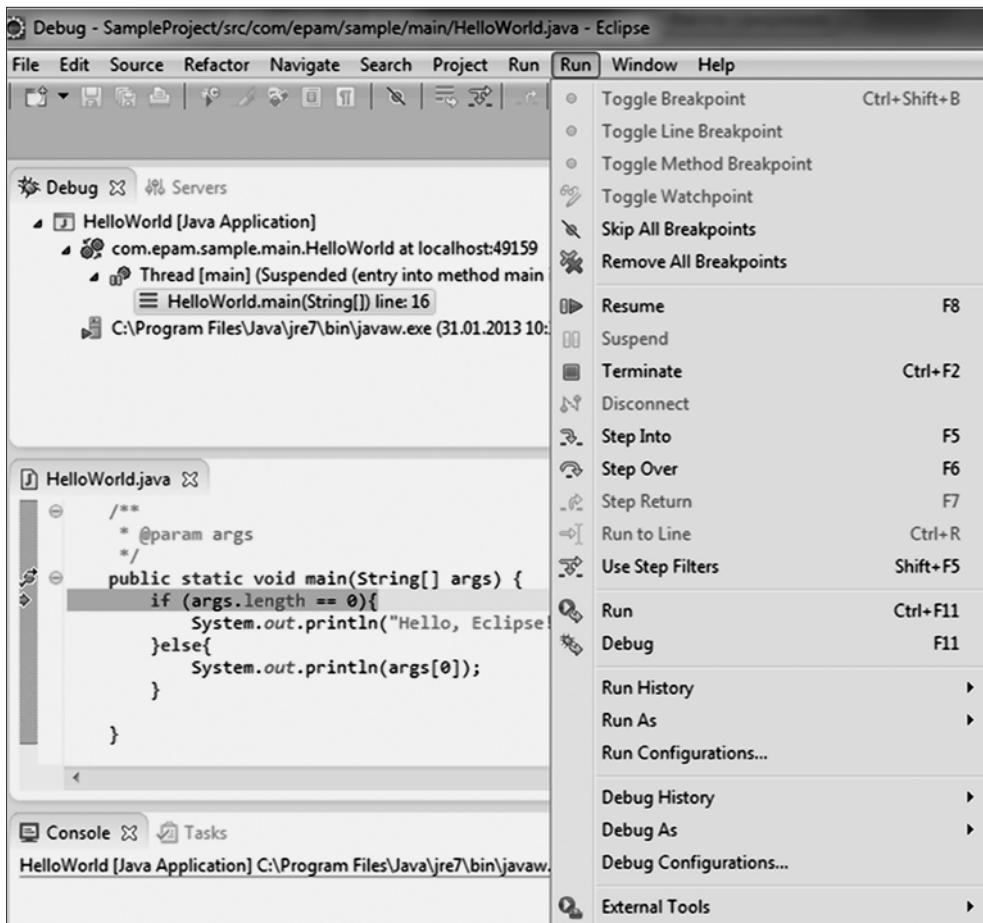


Рис. 13. Управление отладкой

Для управления режимом отладки можно использовать пункты меню **Run**, кнопки на вкладке **Debug** или соответствующие быстрые сочетания клавиш на клавиатуре:

- F8** — продолжить,
- Ctrl+F12** — прервать выполнение программы,
- F5** — выполнить инструкцию,
- F6** — перейти к следующей инструкции,
- Ctrl+R** — перейти к строке.

## Генерация методов

Среда Eclipse предоставляет много дополнительных возможностей по генерации кода, что существенно сокращает рутинные действия при программировании.

Пусть создан класс **Student** с полями **int id** и **String lastName**. Определение инкапсуляции требует, чтобы поля объявлялись как **private**, а доступ к ним осуществлялся через методы **getИмя()** и **setИмя()**. Для получения доступа к меню генерации этих методов следует нажать правую кнопку мыши и в открывшемся меню выбрать **Source** — **Generate Getters and Setters**. В открытом виде будет представлено меню выбора (Рис. 14.). С его помощью можно выбрать необходимые для генерации методы, отметив их в окне *Select*. Выбрать

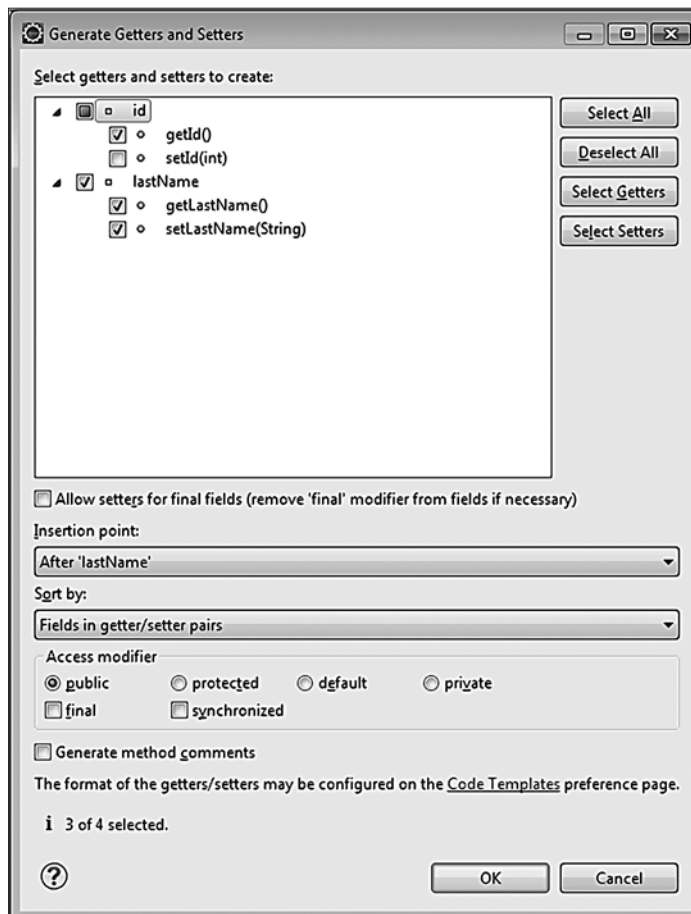


Рис. 14. Генерация *getters & setters*

место размещения в классе в выпадающем списке *Insertion point*. Задать последовательность размещения в коде в выпадающем списке *Sort by*, а также определить модификаторы доступа для создаваемых методов.

В результате выбора осуществленного в рис. 14 будет сгенерирован код в виде (см. Рис. 15).

Обычной практикой для классов, хранящих информацию представляется автоматическая генерация методов `equals()`, `hashCode()` и `toString()`. Для доступа к меню генерации первых двух методов после щелчка правой кнопкой мыши в окне класса в появившемся меню следует выбрать **Source — Generate hashCode() and equals()**. Задать необходимые поля (по умолчанию используются все поля класса), место расположения методов в коде класса и необходимость добавления комментариев.

```

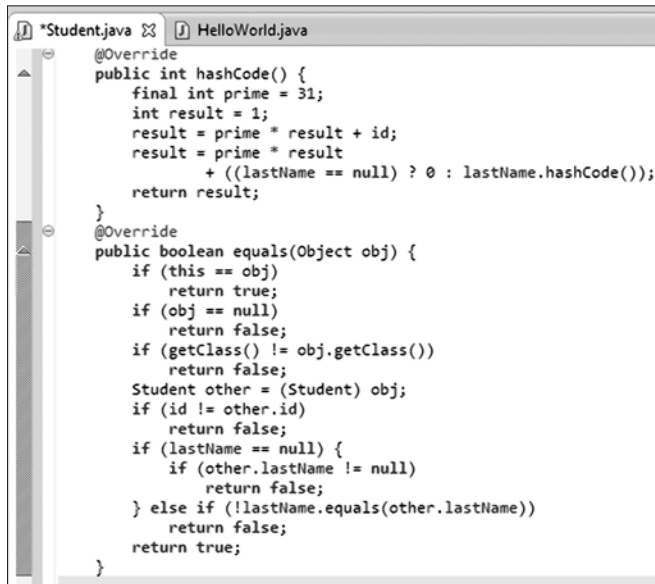
*Student.java HelloWorld.java
package com.epam.sample.main;
public class Student {
    private int id;
    private String lastName;
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
    public int getId() {
        return id;
    }
}
    
```

Рис. 15. Созданный код getters & setters



Рис. 16. Генерация equal & hashCode

В результате по правилам языка Java будет сгенерирован код в виде:



```

*Student.java HelloWorld.java
@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + id;
    result = prime * result
        + ((lastName == null) ? 0 : lastName.hashCode());
    return result;
}
@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Student other = (Student) obj;
    if (id != other.id)
        return false;
    if (lastName == null) {
        if (other.lastName != null)
            return false;
    } else if (!lastName.equals(other.lastName))
        return false;
    return true;
}

```

Рис. 17. Код сгенерированных методов `equal` & `hashCode`

Задать правила генерации метода `toString()` возможно, выбрав в том же меню **Source** — **Generate toString()**.

В отличие от обычно неизменяемого после генерации кода методов `equals()` и `hashCode()`, код метода `toString()` часто подвергается изменениям после генерации в соответствии с потребностями разработчика.

Eclipse предоставляет удобный механизм генерации переопределяемых методов суперкласса. Пусть разработан класс **GroupStudent**, наследующий класс **ArrayList** в виде:

```

package com.epam.sample.main;
import java.util.ArrayList;
public class GroupStudent extends ArrayList<Student> { }

```

При разработке функциональности класса потребовалось переопределить методы добавления студентов в экземпляр разрабатываемого класса. Для решения этой задачи следует выбрать **Override/Implement Methods** в строке меню **Source** и в открывшемся окне развернуть список доступных для переопределения методов класса **ArrayList**. После чего выбрать необходимые методы из представленного списка.

В результате будет сгенерирован синтаксически корректный код с использованием функциональности переопределяемого метода (Рис. 20.). Разработчику остается только внести в код необходимые ему изменения.

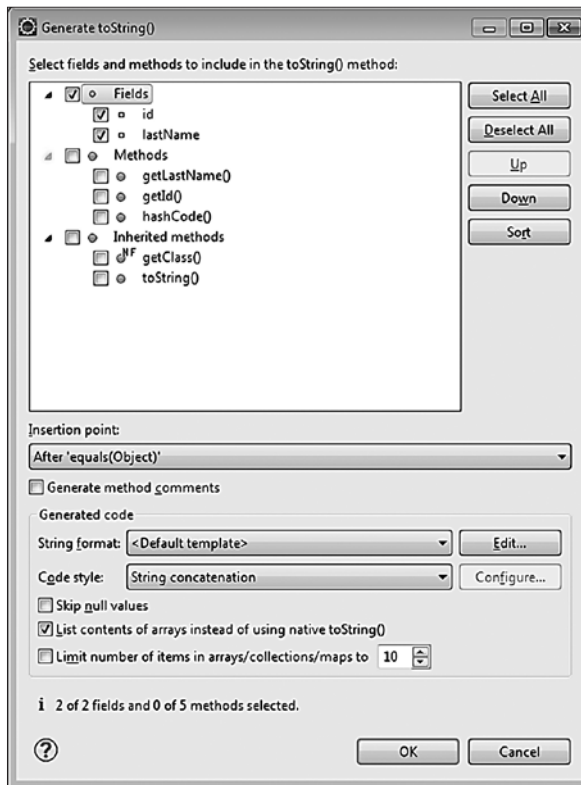


Рис. 18. Генерация toString

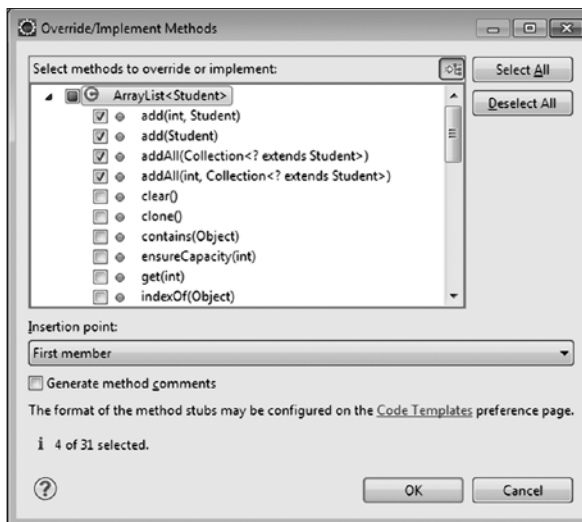


Рис. 19. Генерация переопределяемых методов

```

public class GroupStudent extends ArrayList<Student>{
    @Override
    public void add(int arg0, Student arg1) {
        // TODO Auto-generated method stub
        super.add(arg0, arg1);
    }
    @Override
    public boolean add(Student arg0) {
        // TODO Auto-generated method stub
        return super.add(arg0);
    }
    @Override
    public boolean addAll(Collection<? extends Student> arg0) {
        // TODO Auto-generated method stub
        return super.addAll(arg0);
    }
    @Override
    public boolean addAll(int arg0, Collection<? extends Student> arg1) {
        // TODO Auto-generated method stub
        return super.addAll(arg0, arg1);
    }
}

```

Рис. 20. Созданный код переопределяемых методов

Если же класс не наследует, а агрегирует свойства другого класса, как на пример:

```

package com.epam.sample.main;
import java.util.ArrayList;
public class GroupStudent {
    private ArrayList<Student> students;
}

```

Тогда для реализации необходимых разработчику методов с использованием имен и, возможно, функциональности агрегированного класса следует в меню **Source** выбрать **Generate Delegate Methods**. Последующие действия аналогичны действиям при генерации переопределенного кода.

## Генерация конструкторов

Важную роль в коде практически любого класса выполняют конструкторы. При генерации конструкторов следует принимать во внимание особенности процесса создания экземпляра класса, для которого предназначен тот или иной конструктор. Основное назначение конструктора — инициализация полей. Для генерации конструктора на примере класса **Student** с полями **id** и **lastName** следует выбрать **Source** — **Generate Constructor using fields**.

В предложенном окне следует выбрать поля, которые будут использованы в качестве параметров создаваемого конструктора. Результатом будет один

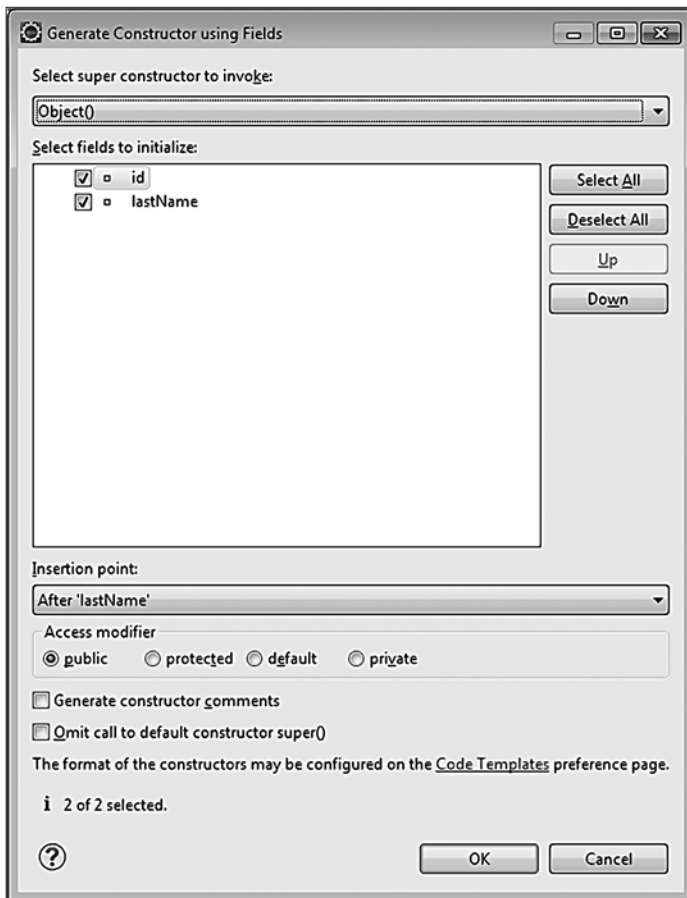


Рис. 21. Генерация конструктора на основе полей класса

конструктор. Если необходим конструктор с другим набором параметров, операцию создания следует повторить.

Если класс наследует другой класс, то при разработке его конструкторов может потребоваться обращение к конструкторам суперкласса. На примере класса **GroupStudent**, выбрав в списке **Source** — **Generate Constructors from Superclass**, будет получено окно, в котором будет представлен список конструкторов суперкласса, доступных для использования в конструкторе подкласса. Выбрав необходимый набор и нажав кнопку **OK**, будет сгенерировано столько конструкторов, сколько было отмечено.

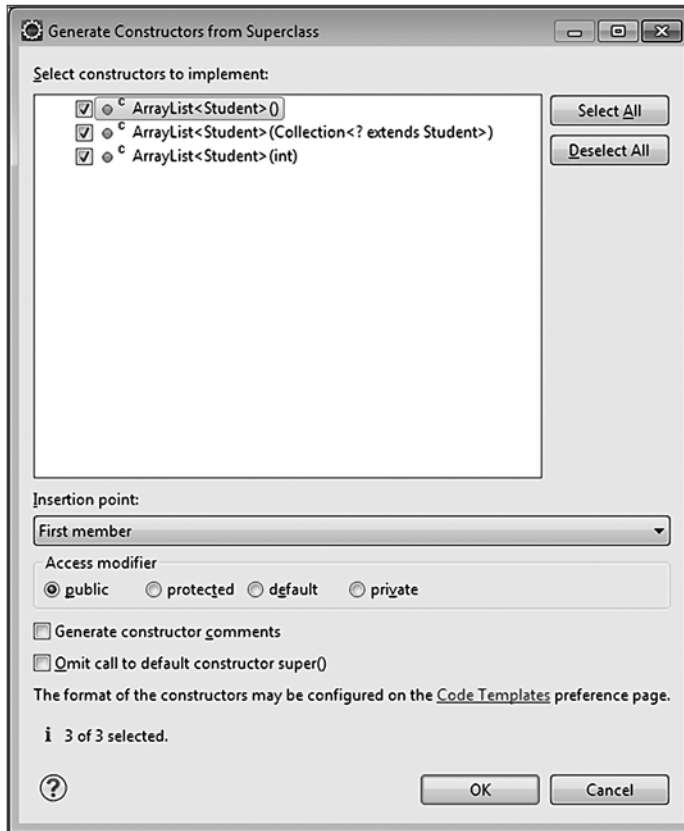


Рис. 22. Генерация конструкторов с использованием суперкласса

## Code assist

Java редактор предоставляет возможность разработчику создавать код быстрее, предоставляя ему возможности *code assist*. Пусть в методе `main()` необходимо создать код, использующий возможности класса `System`. Достаточно набрать в редакторе

`System.`

Сделать небольшую паузу, и автоматически откроется окно *code assist* с предложением выбора статических полей и методов класса `System` (Рис. 23.). Как только выбор будет осуществлен вертикальным перемещением по списку или набором на клавиатуре начальных символов необходимого метода или поля с последующим выбором из списка, результаты выбора появятся в редакторе кода. Вызов *code assist* можно также осуществить нажатием клавиш **Ctrl-Space**.



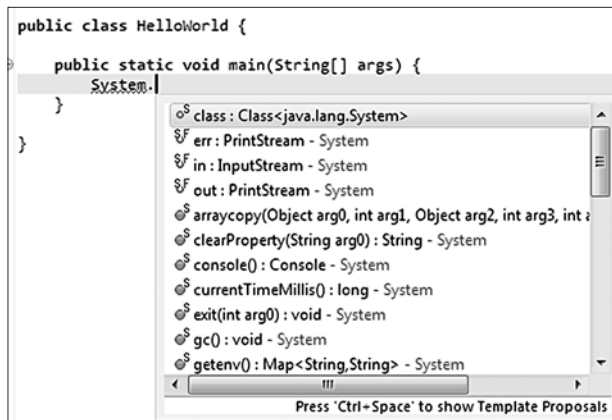


Рис. 23. Code assist

## Шаблоны

Eclipse поддерживает короткий набор части кода с возможностью генерации части конструкции. Для этого применяется набор так называемых templates, созданных в самой среде.

Демонстрация использования шаблонов на примере циклов. В методе набирается слово **for** и нажимаются клавиши **Ctrl-Space**. Появляется меню с предложением выбрать один из четырех циклов языка или классов, начинающихся на символы, набранные в редакторе (Рис. 24.). В правой части окна располагается пример кода, который будет сгенерирован после нажатия кнопки **Enter**. После того, как код будет создан, шаблон предоставляет возможность изменения имени переменной цикла.

Список всех доступных для применения шаблонов можно получить, выбрав **Window — Preferences — Java — Editor — Templates**.

Разрешено создавать собственные шаблоны. Создание шаблона начинается с нажатия кнопки **New** (Рис. 26.).

Шаблоны, в том числе и стандартные, разрешено редактировать. Сохранить и перенести авторский набор шаблонов на другую версию Eclipse можно, применяя кнопки **Export/Import** в окне, представленном на рис. 25.

## Рефакторинг

Созданный код может потребовать исправления. Чаще всего разработчику необходимо переименовать пакет, класс, метод, поле, переменную или константу. Причем требуемое изменения понятие, например метод, может быть

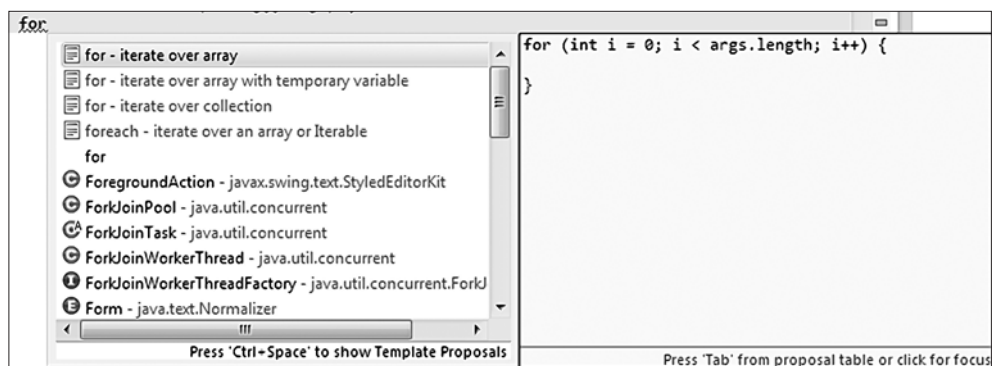


Рис. 24. Вызов шаблона

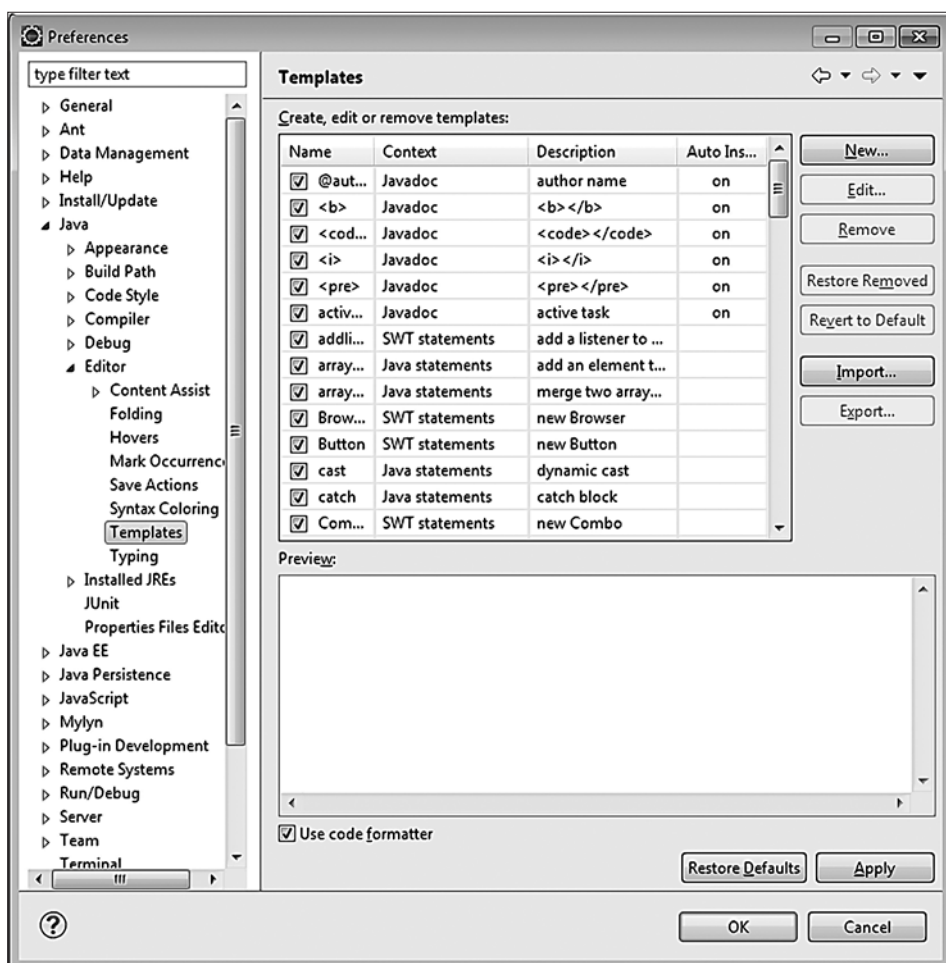


Рис. 25. Список доступных шаблонов

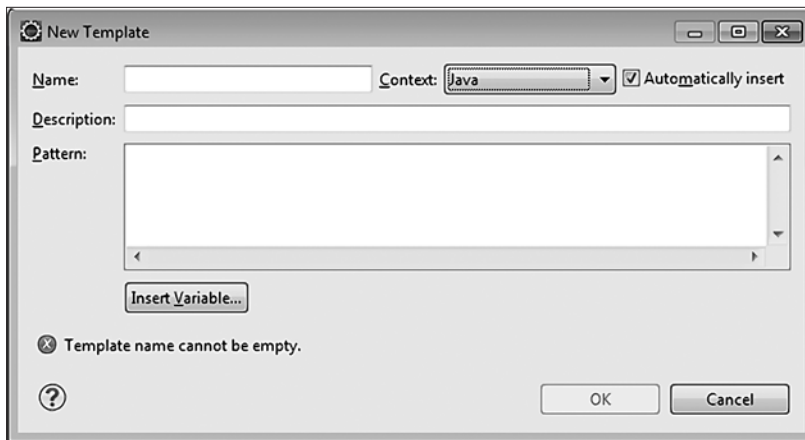


Рис. 26. Меню создания шаблона

многократно использовано в коде. В обычной ситуации разработчику придется перебирать все классы, в которых осуществлялся вызов такого метода и исправлять его имя, что может оказаться весьма трудоемкой задачей.

Eclipse позволяет изменить имя класса, метода, поля или переменной, распространяя изменения на все обращения к изменяемому понятию, в том числе и из других классов и пакетов приложения.

Чтобы осуществить изменение имени, следует двойным щелчком левой клавиши мыши выделить имя, требующее изменения, после чего правой кнопкой вызвать список, выбрать в нем **Source — Refactor — Rename**, ввести новое имя и нажать **Enter** для фиксации изменений в приложении.

При рефакторинге метода может понадобиться не только изменение его имени, но и корректировка всей сигнатуры. Для этого следует выделить метод и выбрать **Source — Refactor — Change Method signature** в результате чего, например, для метода `void setLastName(String lastName)` будет выведена форма редактирования. В этом окне кроме имени можно изменить тип возвращаемого методом значения, спецификатор доступа к методу. Список аргументов метода может быть увеличен добавлением новых аргументов или сокращен при необходимости. Имена аргументов также можно изменять. Предварительный вид сигнатуры будет отображаться в строке *Method signature preview*. Изменения коснутся также всех переопределенных версий метода во всех подклассах.

Рефакторинг предоставляет много других возможностей при работе с классом и его составляющими.

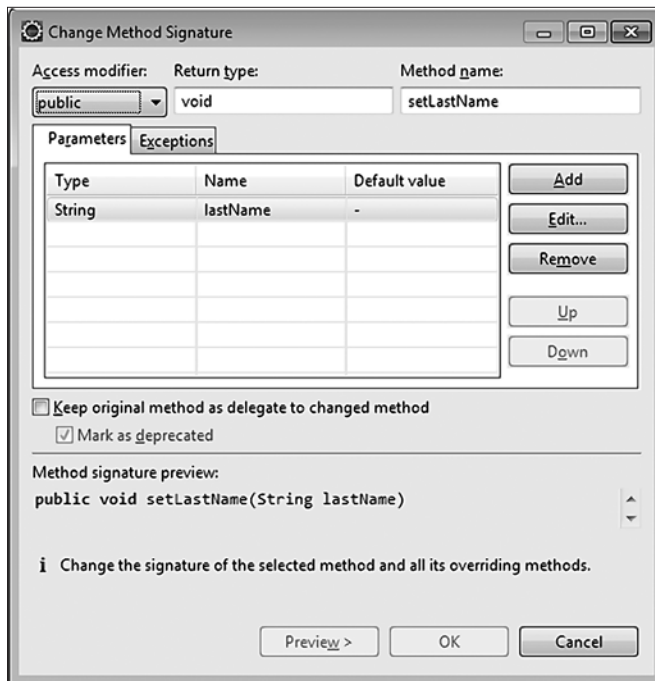


Рис. 27. Рефакторинг сигнатуры метода

## Подключение библиотек

При разработке проектов кроме стандартных библиотек Java необходимы библиотеки сторонних разработчиков, например: Log4J, JUnit, Xerces и прочие. Чтобы воспользоваться функциональностью классов из этих библиотек, их необходимо подключить к проекту. Сначала требуется загрузить jar-архив библиотеки с определенного интернет-ресурса и разместить его на диске, желательно в директории, находящейся внутри основной директории проекта, хотя в общем случае это необязательно. Для включения библиотеки в ресурсы проекта необходимо щелкнуть правой кнопкой мыши по проекту и из выпавшего списка выбрать **Build Path** — **Configure Build Path**.

В появившемся окне следует выбрать закладку **Libraries** и нажать кнопку **Add External JARs** (Рис. 28.). С помощью формы файловой системы найти необходимый jar-архив, например, **log4j-[версия].jar** и нажать кнопку **Open**. После чего указанная библиотека появится в списке подключенных библиотек.

Библиотека Log4J будет видна при открытии папки *Referenced Libraries*.

Теперь классы библиотеки можно использовать в разрабатываемом проекте.

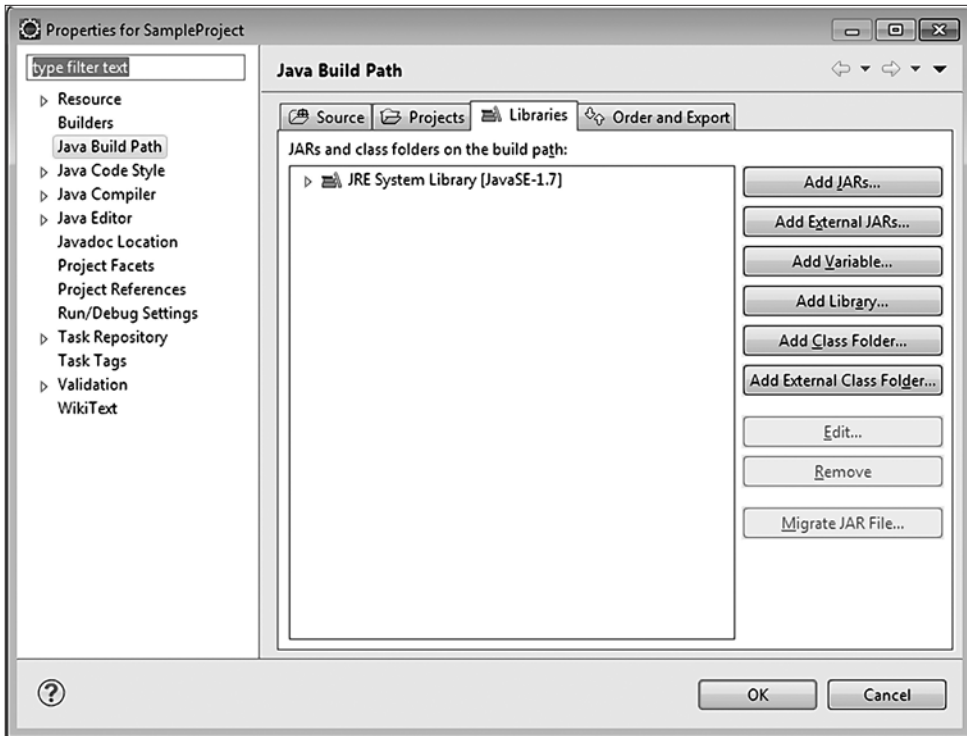


Рис. 28. Подключение внешней библиотеки

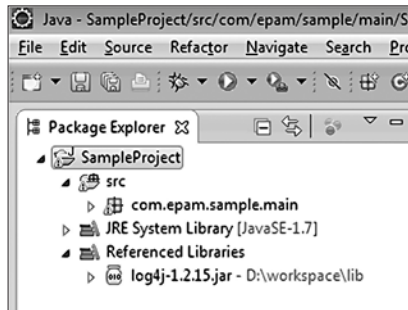


Рис. 29. Проект с подключенной библиотекой

## Подключение plugin-ов

IDE Eclipse представляет собой настраиваемую систему, возможности которой можно наращивать и улучшать. Добавление новых возможностей производится за счет подключения plugin-ов. Plugin представляет собой некую небольшую

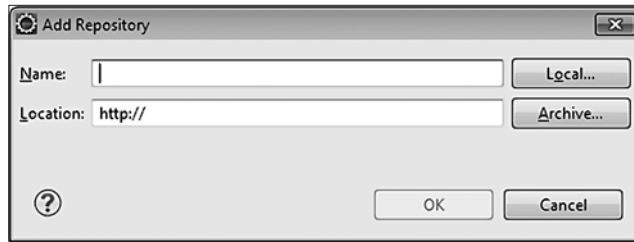


Рис. 30. Проект с подключенной библиотекой

программу, для подключения которой следует выбрать **Help — Install New Software** и в появившейся форме нажать кнопку **Add** (Рис. 30.). В поле *Name* указать имя плагина, в поле *Location* адрес URL его загрузки и запустить процесс. Затем плагин будет закачан на диск разработчика и подключен, после чего сразу становится доступным для использования.

Существуют еще три способа подключения плагинов, если они уже находятся на диске разработчика. Если плагин представлен в виде архива, то следует на форме нажать кнопку **Archive** и выбрать искомый архив с плагином. Если плагин представлен в виде набора папок с файлами, то следует выбрать **Local** и указать на корневую папку плагина. Два последних способа могут не работать, так как плагины, разработанные к разным версиям Eclipse, могут быть несовместимы с представленной версией. Есть еще один кустарный способ подключения плагинов прямым копированием в папку `plugins` самого IDE Eclipse. Несовместимость версий в этом случае может привести к тому, что плагин может быть вообще не представлен в функционале IDE или работать некорректно.

Совместимость версии плагина с версией Eclipse следует проверять на сайте разработчика плагина или на `eclipse.org`.

## Импорт/экспорт проектов

В процессе разработки проект может переноситься с одного компьютера на другой. Для переноса проекта следует правым кликом на проекте вызвать его меню и выбрать **Export — Java — JAR file — Next** (Рис. 31.). В открывшейся форме обязательно пометить check box с надписью *Export Java source files and resources*. Иначе архив, полученный в результате экспорта, не будет содержать файлы с исходным кодом проекта. В поле *JAR file* следует указать месторасположение и имя jar-архива и нажать кнопку **Finish**. После чего можно забрать созданный архив и переносить на другой компьютер.

Экспорт можно выполнить, и в директорию на диске для этого необходимо выбрать **Export — General — File System — Next**. В поле *To directory* открывшейся

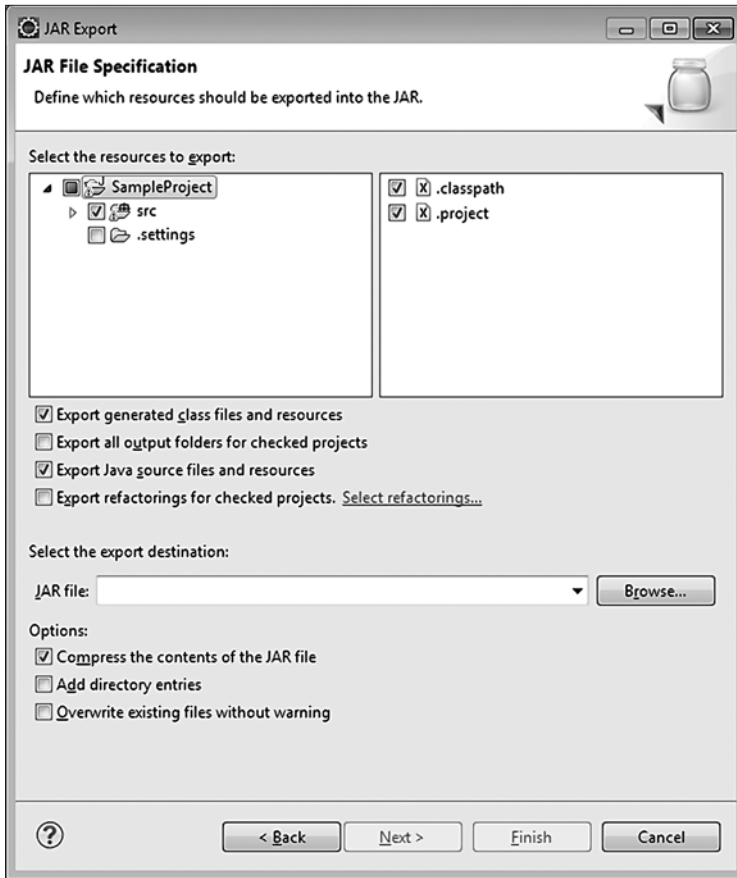


Рис. 31. Экспорт проекта

формы ввести путь, по которому после выполнения экспорта можно будет за-  
брать директорию с файлами проекта.

Для импорта проекта в Eclipse следует сначала создать новый java проект,  
например, с именем *SampleProjectCopy*. После чего выбрать **File** — **Import** —  
**General** — **Archive file** (Рис. 32.). Далее выбрать *SampleProjectCopy/src* в поле  
*Into folder*. В поле *From archive file* выбрать jar-архив импортируемого проекта  
и нажать кнопку **Finish**.

Все исходные коды импортируемого проекта окажутся в папке *src* проекта  
*SampleProjectCopy*. Существуют и другие способы импорта.

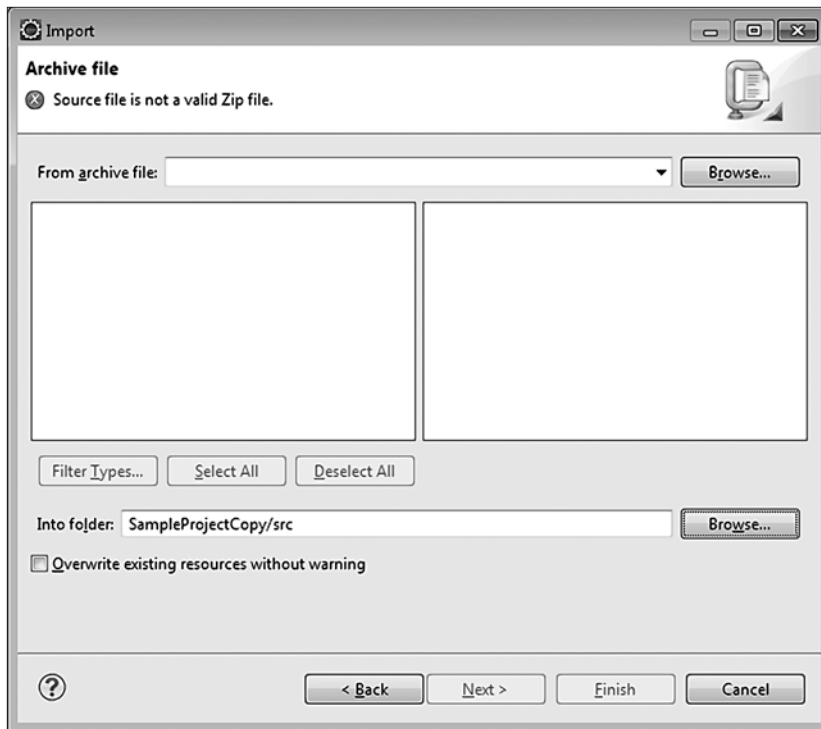


Рис. 32. Импорт проекта

## Создание, запуск и отладка веб-проекта

Перед созданием динамического веб-проекта необходимо настроить сервер java-приложений. Для этого в меню **Window** — **Preferences** выбирается в дереве слева **Server** — **Runtime Environments**. Чтобы добавить application server, например, Apache Tomcat необходимо нажать кнопку **Add...**

После нажатия кнопки **Finish** в списке серверов появится только что добавленный сервер. Теперь необходимо выбрать меню **File** — **New** — **Dynamic web project**, задать имя проекта, выбрать сервер. Можно задать некоторые дополнительные параметры, нажимая кнопку **Next**, можно оставить все остальное по умолчанию и нажать кнопку **Finish** для создания проекта (Рис. 35.). Созданный проект появится в **Project Explorer** (Рис. 36.).

Далее следует нажать **Next**, и на следующей форме также нажать **Next**. На полученной форме конфигурирования веб-модуля выбрать *Generate web.xml deployment descriptor*. Наличие файла **web.xml** необходимо для задания важных параметров инициализации веб-приложения.

Для завершения создания веб-проекта следует нажать **Finish** и в закладке *Project Explorer* появится проект.



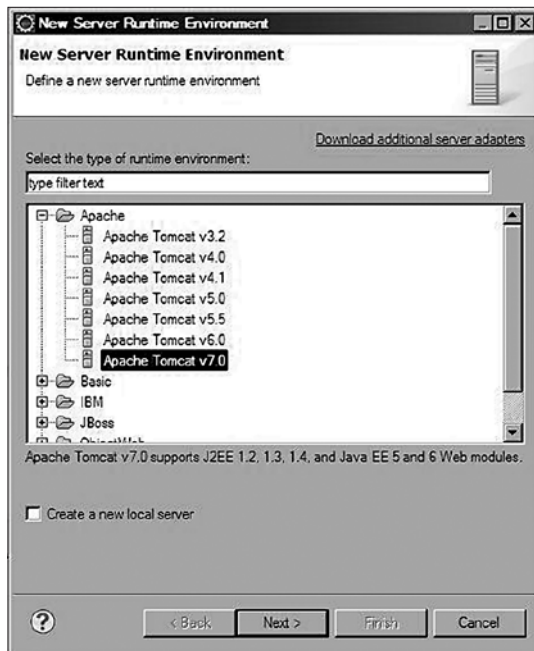


Рис. 33. Выбор сервера для добавления

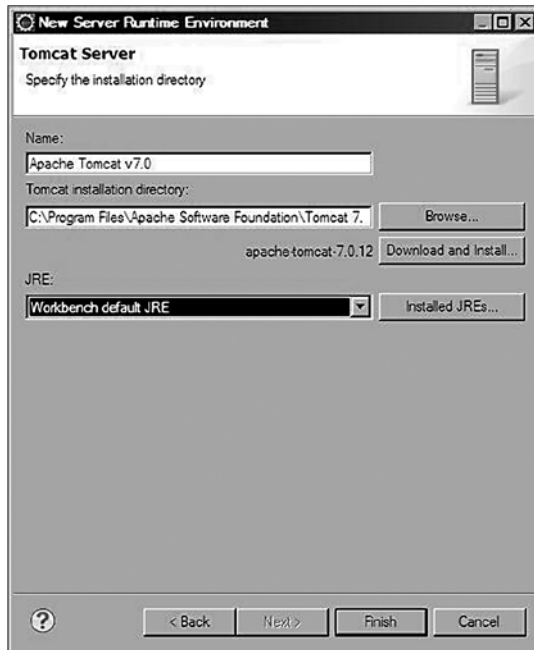


Рис. 34. Определение директории сервера

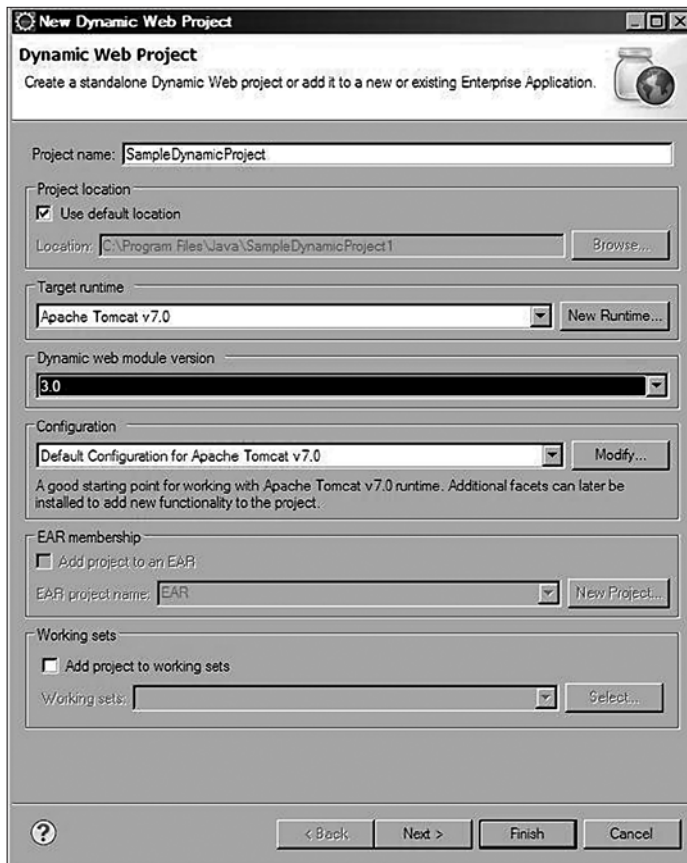


Рис. 35. Создание нового веб-проекта

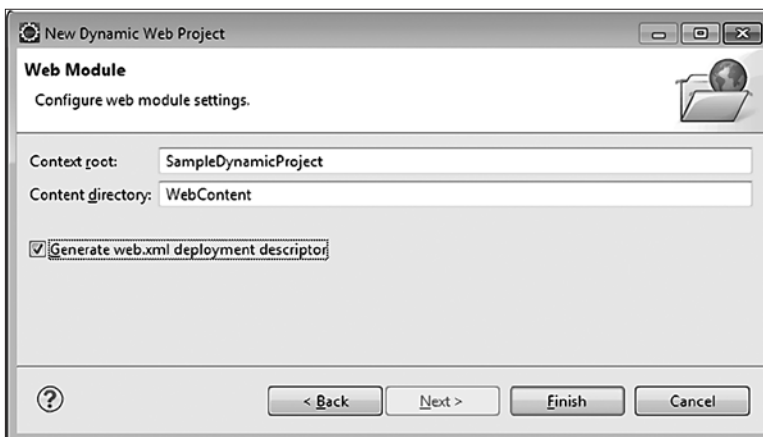


Рис. 36. Создание нового веб-проекта

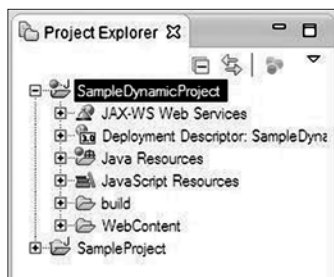


Рис. 37. Вид веб-проекта  
в Project Explorer

Для запуска веб-проекта необходимо нажать правой кнопкой мыши на название проекта в **Project Explorer** и выбрать пункт **Run As — Run On Server**. Для отладки: **Debug As — Debug On Server**. Управление процессом отладки такое же, как в консольном проекте.