ТИМУР МАШНИН

bhv

СОВРЕМЕННЫЕ Java ТЕХНОЛОГИИ НА ПРАКТИКЕ

- ОБЗОР JAVA-ТЕХНОЛОГИЙ И ОБЛАСТЕЙ ИХ ПРАКТИЧЕСКОГО ПРИМЕНЕНИЯ
- БИБЛИОТЕКИ, СПЕЦИФИКАЦИИ, СЕРВИСЫ
- АРХИТЕКТУРА ПЛАТФОРМ JAVA SE, JAVA ME И JAVA EE
- РАЗРАБОТКА АПЛЕТОВ, НАСТОЛЬНЫХ, МОДУЛЬНЫХ И РАСПРЕДЕЛЕННЫХ JAVA-ПРИЛОЖЕНИЙ

PRO

ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

Тимур Машнин

СОВРЕМЕННЫЕ Java технологии НА ПРАКТИКЕ

Санкт-Петербург
«БХВ-Петербург»
2010

УДК 681.3.06
ББК 32.973.26-018.2
М38

Машнин Т. С.

М38 Современные Java-технологии на практике. — СПб.: БХВ-Петербург, 2010. — 560 с.: ил. — (Профессиональное программирование)

ISBN 978-5-9775-0561-1

Рассмотрено создание широкого круга Java-приложений с помощью современных Java-технологий и среды разработки NetBeans. Подробно рассмотрена архитектура платформ Java SE, Java ME и Java EE. Показано создание апплетов с использованием графических библиотек AWT и Swing, настольных приложений на основе платформы Swing Application Framework, а также расширяемых Java-приложений с использованием библиотек ServiceLoader API, Lookup и др. для платформы Java SE. Рассмотрено создание мобильных приложений на основе конфигурации CLDC и профиля MIDP для платформы Java ME. Показано применение технологий Java Servlet, JavaServer Pages, JavaServer Faces, Web-сервисов, Enterprise JavaBeans и др. при программировании для платформы Java EE. Материал книги сопровождается большим количеством примеров с подробным анализом исходных кодов.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 31.05.10.
Формат $70\times100^{1}/_{16}$. Печать офсетная. Усл. печ. л. 45,15.
Тираж 1500 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0561-1

Оглавление

ЧАСТЬ I

Платформа Java SE

Глава 1	Создание апплетов с использованием графической библиотеки AWT (Abstract Window Toolkit)
Глава 2	Создание апплетов с использованием графической библиотеки Swing
Глава 3	Создание настольных приложений на базе платформы Swing Application Framework
Глава 4	Создание расширяемых Java-приложений

Общая структура платформы Java SE была рассмотрена во *введении*. Еще раз можно повторить, что платформа Java SE является основой для всех остальных платформ технологии Java и предназначена для создания апплетов и настольных приложений. Платформа Java SE содержит набор технологий, которые описываются соответствующими спецификациями. Перечень спецификаций платформы Java SE и их общее применение можно посмотреть в *приложении 5*.

Создаваемые на основе платформы Java SE апплеты представляют собой программы, написанные на языке Java и работающие в среде браузера, который загружает их и запускает виртуальную машину JVM для их выполнения.

В отличие от настольных приложений, апплеты являются управляемыми программными компонентами. Это означает, что апплет не может быть запущен, как настольное приложение, одной только виртуальной машиной JVM. Для его работы необходим браузер, который распознает теги `<APPLET>` или `<OBJECT>` и `<EMBED>`, включающие апплет в HTML-страницу. Главный класс апплета должен быть подклассом класса `java.applet.Applet`, при этом класс `Applet` служит интерфейсом между апплетом и браузером. Жизненным циклом апплета управляет компонент Java Plug-in среды выполнения JRE.

Апплеты могут быть двух типов: имеющие электронную подпись и сертификат и не имеющие сертификата. Не имеющие сертификата апплеты, или апплеты, сертификат которых не принял конечный пользователь, работают в "песочнице" — данное понятие обозначает специальный механизм системы безопасности, ограничивающий доступ работающего приложения к компьютеру пользователя. Апплеты, сертификат которых принят пользователем, работают вне "песочницы", в частности, они способны осуществлять доступ к файловой системе пользователя.

Область применения апплетов достаточно широка. Апплеты используются в качестве элементов, украшающих Web-страничку, создавая продвинутую анимацию, служат интерактивной рекламой, представляют интерактивные игры. Однако назначение апплетов гораздо шире, чем использование в Web-дизайне. При создании приложений уровня предприятия с помощью апплетов создается графический интерфейс пользователя *"тонкого"* клиента.

Настольные приложения платформы Java SE — это независимые Java-приложения, которые выполняются виртуальной машиной JVM, при этом точкой входа в приложение является главный класс приложения, содержащий статический метод `main`. С помощью технологии Java можно создать практически любое настольное приложение с областью применения, ограниченной только фантазией разработчика. Это различные проигрыватели, навигаторы, читалки, переводчики, почтовые клиенты и многое другое. В приложениях уровня предприятия настольное приложение представляет *"толстого"* клиента.

ГЛАВА 1

Создание апплетов с использованием графической библиотеки AWT (Abstract Window Toolkit)

Обзор графической библиотеки AWT

Любое приложение, требующее взаимодействия с пользователем, должно иметь интерфейс пользователя. От интерфейса пользователя зависит привлекательность и удобство работы с программой. Интерфейс пользователя может быть как низкоуровневым, в виде командной строки, так и иметь разнообразные графические компоненты — кнопки, переключатели, меню, поля ввода и другие элементы. Технология Java предлагает набор библиотек классов и интерфейсов, на основе которых можно построить эффективный графический интерфейс пользователя, имеющий необходимый внешний вид и поведение для создания комфортной среды конечному потребителю.

Самой первой графической Java-библиотекой была создана библиотека AWT. Она была включена в первую версию JDK 1.0. Затем библиотека AWT была дополнена библиотекой Java 2D API, расширяющей возможности работы с двухмерной графикой и изображениями. Так как технология Java является платформно-независимой, то соответственно и графическая Java-библиотека должна быть платформно-независимой. Сам по себе язык Java не обладает возможностями низкоуровневого взаимодействия с конкретной операционной системой, обеспечивающими передачу информации от мыши или клавиатуры приложению и вывод пикселов на экран. Поэтому библиотека AWT была создана так, что каждый AWT-компонент имеет своего двойника `peer` — интерфейс, обеспечивающий взаимодействие с конкретной операционной системой. Таким образом, переносимость графической библиотеки AWT обусловлена наличием реализации пакета `java.awt.peer` для конкретной

операционной системы. Вследствие этого, AWT-компоненты называют *тяжеловесными*.

Все AWT-компоненты, кроме элементов меню, представлены подклассами класса `java.awt.Component`. Для элементов меню суперклассом является класс `java.awt.MenuComponent`. Архитектура AWT устроена таким образом, что компоненты размещаются в контейнерах (суперкласс `java.awt.Container`) с помощью менеджеров компоновки — классов, реализующих интерфейс `java.awt.LayoutManager`.

Для настольных приложений корневое окно графического интерфейса пользователя представляет контейнер `java.awt.Window`, который в свою очередь должен содержать окно `java.awt.Frame` с заголовком и границами или диалоговое окно `java.awt.Dialog`, также имеющее заголовок и границы. AWT-компоненты добавляются в панель `java.awt.Panel` — контейнер, который может содержать как компоненты, так и другие панели.

Для апплетов класс `java.applet.Applet`, расширяющий класс `java.awt.Panel`, является корневым контейнером для всех графических компонентов.

Помимо графических компонентов, библиотека AWT содержит классы и интерфейсы, позволяющие обрабатывать различные типы событий, генерируемые AWT-компонентами. Суперклассом, представляющим все AWT-события, является класс `java.awt.AWTEvent`. Для обработки событий компонента необходимо создать класс-слушатель, реализующий интерфейс `java.awt.event.ActionListener`, и присоединить его к данному компоненту.

Кроме пакетов `java.awt` и `java.awt.event` библиотека AWT включает в себя пакеты:

- `java.awt.color` используется для создания цвета;
- `java.awt.datatransfer` применяется для передачи данных внутри приложения и между приложениями;
- `java.awt.dnd` реализует технологию drag-and-drop;
- `java.awt.font` обеспечивает поддержку шрифтов;
- `java.awt.geom` реализует двухмерную геометрию;
- `java.awt.im` обеспечивает поддержку нестандартных методов ввода текста;
- `java.awt.image` используется для создания и редактирования графических изображений;
- `java.awt.print` обеспечивает поддержку печати.

Так как AWT-компоненты основываются на peer-объектах, то использование библиотеки AWT является потоково-безопасной (thread safe), поэтому не нужно беспокоиться о том, в каком потоке обновляется состояние графиче-

ского интерфейса. Однако беспорядочное использование потоков может замедлять работу AWT-интерфейса.

Обобщая вышесказанное, можно сказать, что графическая библиотека AWT представляет собой промежуточный уровень между операционной системой и Java-кодом приложения, скрывая все низкоуровневые операции, связанные с построением графического интерфейса пользователя. Такое прямое взаимодействие с конкретной операционной системой является и основным недостатком AWT, т. к. графический интерфейс, созданный на основе AWT, в операционной системе Windows выглядит как Windows-подобный, а в операционной системе Mac OS X — как Mac-подобный.

Применение AWT и сравнение с другими графическими Java-библиотеками

Графическая библиотека AWT — это стандарт платформы Java SE, однако набор графических компонентов, предлагаемый AWT, ограничен. В частности, отсутствуют такие компоненты, как таблицы, строка состояния, переключатель (radio button) и др. Кроме того, внешний вид и поведение графического AWT-интерфейса пользователя зависят от операционной системы, в которой он отображается.

Все это послужило причиной создания графической библиотеки Swing. Библиотека Swing расширяет AWT и создана полностью на языке Java, поэтому Swing-компоненты называются *легковесными*, за исключением четырех компонентов верхнего уровня: `JWindow`, `JFrame`, `JDialog` и `JApplet`, которые являются прямыми наследниками тяжеловесных AWT-компонентов. Библиотека Swing предлагает гораздо более широкий набор компонентов и менеджеров компоновки по сравнению с библиотекой AWT. Кроме того, т. к. Swing создана полностью на Java, то решается проблема одинакового отображения и поведения интерфейса пользователя в различных операционных системах.

Помимо всего прочего, библиотека Swing расширяет модель обработки событий AWT-библиотекой соответствующих классов и интерфейсов. Однако при этом использование библиотеки Swing не является потоково-безопасной, поэтому Java-код, изменяющий состояние интерфейса и обрабатывающий события, должен выполняться в специальном потоке *Event Dispatch Thread (EDT)*.

Так же как и AWT, библиотека Swing — стандарт платформы Java SE.

Альтернативой AWT и Swing служит библиотека SWT (Standard Widget Toolkit) и набор расширенных возможностей JFace проекта Eclipse, не являющий-

ся стандартом Java SE. Это означает, что их использование требует включения в CLASSPATH Java-приложения.

Примечание

CLASSPATH — переменная среды выполнения, содержащая информацию о расположении Java-файлов. Данная информация используется инструментами javac и java при компиляции и выполнении Java-кода соответственно.

Концепция SWT близка концепции AWT. Компоненты SWT также взаимодействуют с операционной системой с помощью интерфейсов peer. Отличие заключается в том, как данное взаимодействие происходит. В SWT интерфейсы peer выполняют функцию оболочек для графических библиотек конкретных операционных систем. Поэтому приложение, созданное с использованием библиотеки SWT, становится полностью совместимым с конкретной операционной системой, тем самым нарушается принцип "Write Once, Run Anywhere". Однако использование графических возможностей самой операционной системы, вместо создания собственной графики, делает применение библиотеки SWT более эффективным. По ассортименту графических компонентов, менеджеров компоновки и модели обработки событий SWT сравнима со Swing. Так же как и Swing, библиотека SWT не является потоковобезопасной.

В платформу Java SE также включена графическая библиотека Java 3D, которая, правда, не поставляется вместе с комплектом разработки JDK, а требует отдельной загрузки и инсталляции. Библиотека Java 3D позволяет создавать 3D-апплеты и Java-приложения, использующие трехмерную графику. С Java 3D можно эффективно конструировать виртуальные миры, создавая отдельные графические элементы и затем соединяя их в древовидные структуры — *scene graph*. Библиотека Java 3D — это результат синтеза лучших идей, взятых из таких технологий, как Direct3D, OpenGL, QuickDraw3D и XGL.

Казалось бы, при таком выборе графических Java-библиотек библиотека AWT должна потерять свою актуальность. Однако если нет необходимости в широком ассортименте графических компонентов, если требуется работа в основном с двухмерной графикой и изображениями, использование библиотеки AWT удобно. Кроме того, библиотека AWT является частью платформы Java ME и используется для создания приложений, работающих в устройствах с ограниченными возможностями — КПК и телевизионных приставках.

Использование AWT на примере создания апплета-игры "Звездные войны"

В качестве примера рассмотрим создание апплета, представляющего игру "Звездные войны".

Примечание

Файлы с кодами апплета-игры "Звездные войны" расположены на прилагаемом к книге компакт-диске в папке Часть_I\Глава_1\StarWar.

Описание игры

Космический корабль летит в звездном пространстве и поражает огнем лазера появляющихся врагов. За каждого пораженного врага начисляются очки. У корабля есть определенное количество жизней, которое уменьшается при столкновении с врагом. Игра прекращается при обнулении количества жизней космического корабля.

В самом начале создадим в простейшем графическом редакторе Paint, поставляемом с операционной системой Windows, графические изображения, необходимые для работы апплета:

- космический корабль — ship.gif;
- пламя двигателя корабля — fire.gif;
- огонь лазера пушки корабля — laser.gif;
- враги — enemy.gif;
- взрыв при столкновении или при поражении врага — explosion.gif.

Далее в среде разработки NetBeans создаем новый проект:

1. В главном меню среды разработки (верхняя строка) выберем **File | New Project**. Появится диалоговое окно **New Project** (рис. 1.1), в котором выберем **Java Class Library**.
2. В строке **Project Name** введем название проекта StarWar (рис. 1.2).
3. Нажмем кнопку **Finish**.
4. В окне **Projects** среды NetBeans щелкнем правой кнопкой мыши на названии проекта StarWar. В появившемся контекстном меню выбираем **New | Other**.
5. В появившемся окне **New File** выберем **Java | Applet** (рис. 1.3). В этом же окне есть пункт шаблон **JApplet**, который отличается от шаблона **Applet** тем, что использует класс `JApplet`, являющийся расширением класса `Applet` и использующий графическую библиотеку Swing, в то время как класс `Applet` создан на основе графической библиотеки AWT.
6. Далее нажмем кнопку **Next** и введем имя класса и пакета (рис. 1.4).
7. Нажмем кнопку **Finish** и попадем в окно редактора исходного кода среды NetBeans (рис. 1.5).

При создании проекта среда NetBeans сгенерировала код, являющийся основой нашего апплета. Для общего понимания проанализируем этот код, представленный в окне редактора (листинг 1.1).

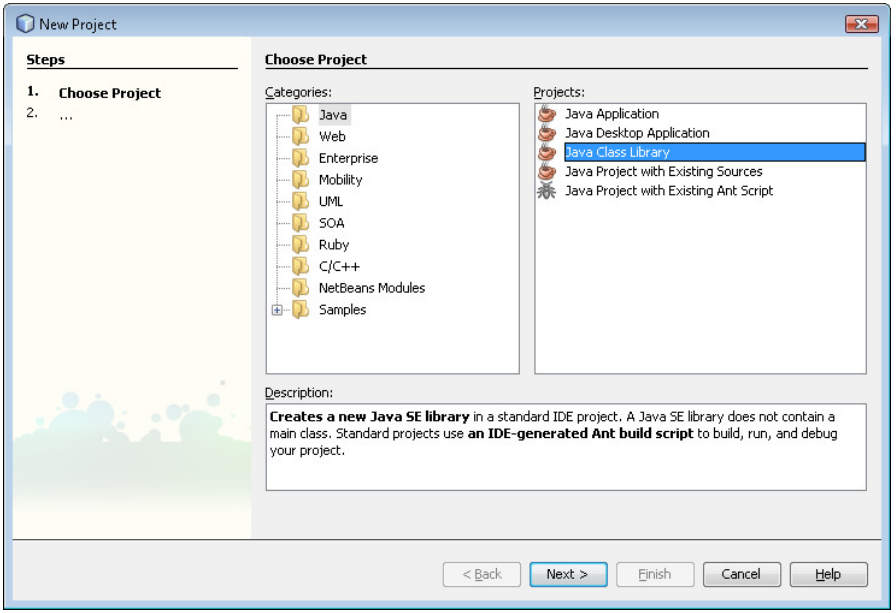

Рис. 1.1. Окно выбора проекта

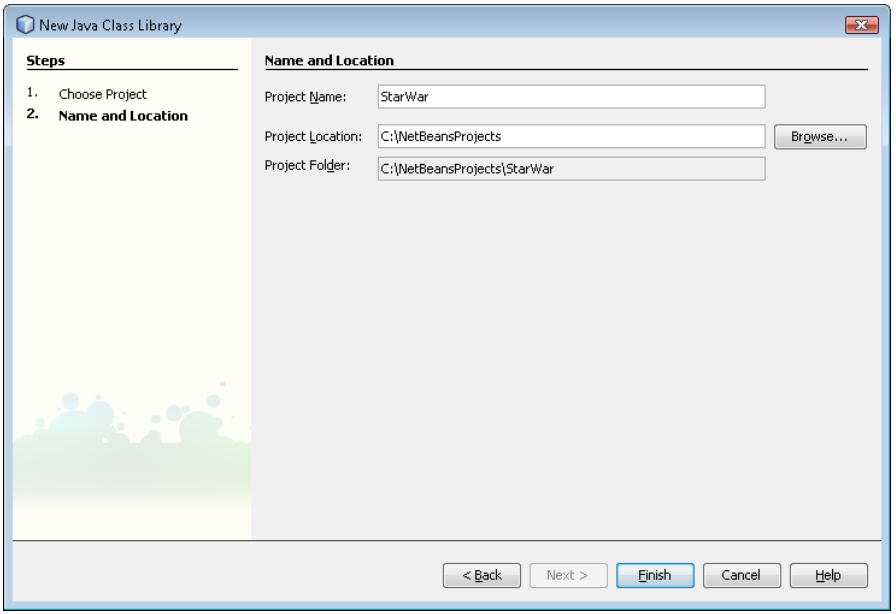

Рис. 1.2. Окно ввода параметров проекта

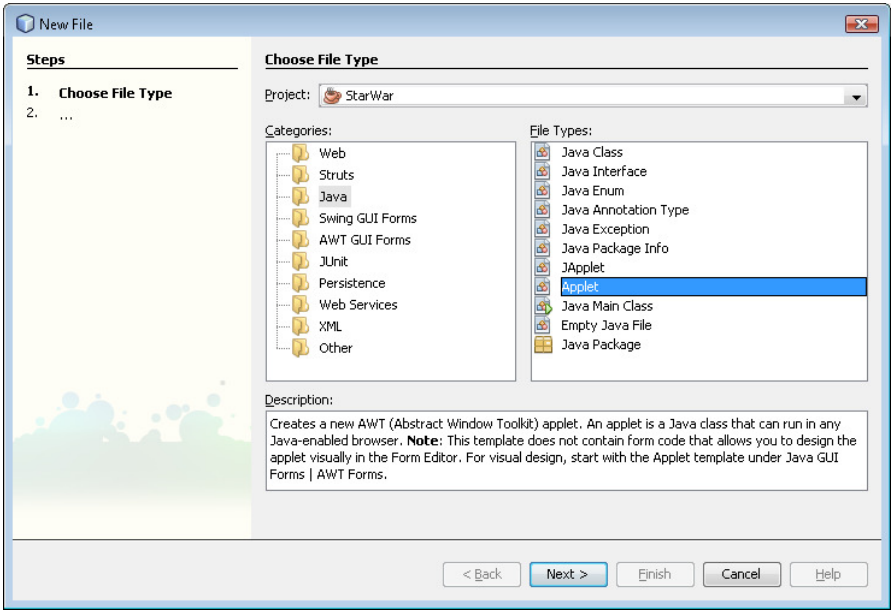

Рис. 1.3. Окно выбора компонентов проекта

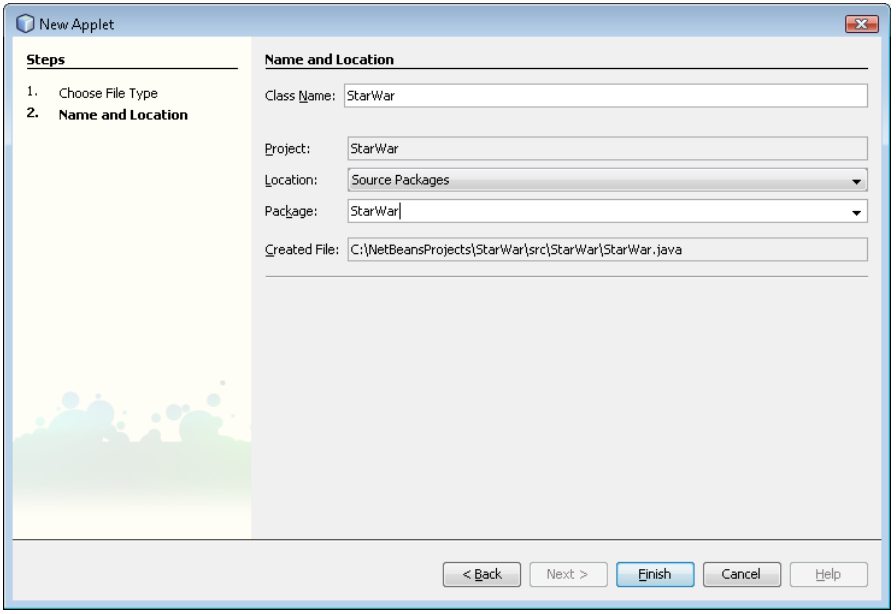

Рис. 1.4. Окно ввода параметров класса и пакета

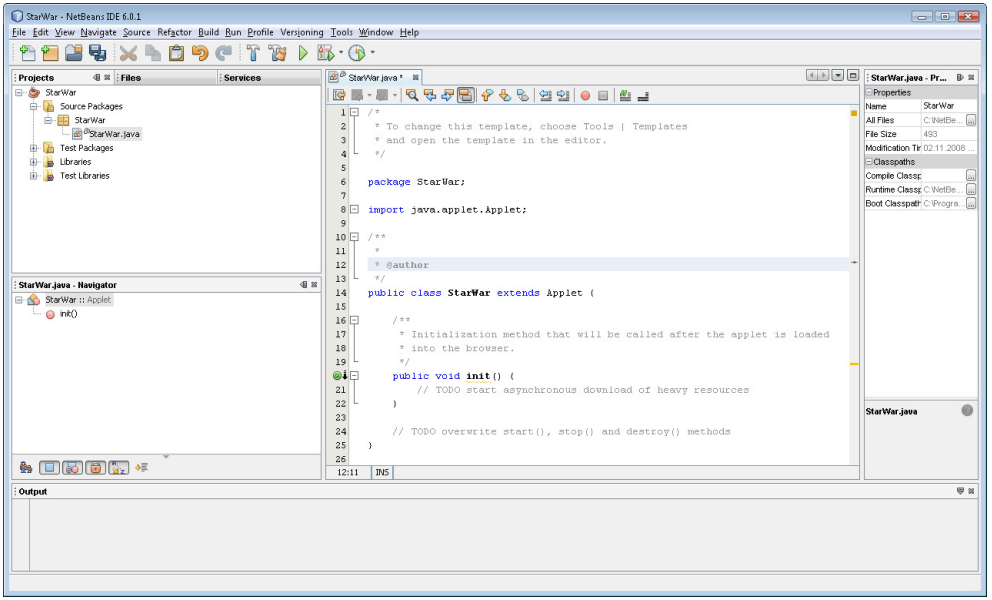

Рис. 1.5. Окно редактора исходного кода

Листинг 1.1. Объявление пакета и класса апплета

```
package StarWar;
import java.applet.Applet;
public class StarWar extends Applet {
    public void init() {
                       }
}
```

В сгенерированном коде строка `import java.applet.Applet;` обеспечивает употребление в коде короткого имени класса `Applet`, принадлежащего пакету `java.applet`. Строка `public class StarWar extends Applet{}` объявляет наш новый класс `StarWar`, расширяющий класс `Applet`. Как уже было сказано, главный класс любого апплета должен быть подклассом класса `Applet`, служащего интерфейсом между апплетом и браузером. Строка `public void init(){}` предназначена для переопределения метода `init()` класса `Applet`.

Роль метода `init()` заключается в том, что он всегда вызывается автоматически браузером для выполнения начальной инициализации апплета.

Теперь необходимо дополнить сгенерированный средой NetBeans код для того, чтобы получить апплет, представляющий создаваемую нами игру.

В качестве первого шага выведем изображение космического корабля в окно апплета (листинг 1.2):

1. Воспользуемся классом `Image` пакета `java.awt`. Абстрактный класс `Image` является суперклассом для всех классов, представляющих графические изображения. Импортируем пакет `java.awt`, тем самым обеспечив употребление коротких имен классов этого пакета.
2. Затем внутри нашего класса `StarWar` объявим объекты класса `Image`, которые будут представлять космический корабль и пламя его двигателя.
3. Вызовем метод `getImage()` класса `Applet`, возвращающий объект `Image`, представляющий изображение, расположенное по адресу, который может быть получен с помощью метода `getCodeBase()` класса `Applet`. Метод `getCodeBase()` возвращает URL-адрес каталога, в котором находится код апплета. Вызовем метод `getImage()` в теле метода `init()`, инициализирующем апплет, при этом воспользуемся ключевым словом `this`, для того чтобы отнести вызываемые методы к описываемому нами классу `StarWar`. В имени графических файлов папка StarWar указывается, потому что был создан одноименный пакет класса, если пакет не создается, папку в имени указывать не надо.
4. Для прорисовки изображений в окне апплета необходимо переопределить метод `paint()`, который наследуется классом `Applet` от класса `Container` пакета `java.awt`. Класс `Container` является суперклассом для всех классов, представляющих компоненты-контейнеры, содержащие другие графические компоненты. В качестве аргумента метода `paint()` выступает объект класса `java.awt.Graphics`. Класс `Graphics` — абстрактный класс для работы с графическим контекстом, представляющим информацию об области изображения, цвете и шрифте. В методе `paint()` зададим черный цвет фона окна апплета при помощи метода `setBackground()`, унаследованного классом `Applet` от класса `java.awt.Component`. В качестве аргумента метода назначим объект класса `java.awt.Color` с определенным полем (`Color.black`). Затем вызовем метод класса `Graphics` — `drawImage()`, аргументами которого выступают: объект `Image`, координаты области, в которой будет нарисовано изображение, и ссылка на объект класса, реализующего интерфейс `ImageObserver`, в нашем случае это сам класс `Applet`, поэтому мы укажем ключевое слово `this`. Интерфейс `ImageObserver` используется для получения информации о процессе конструирования изображения.
5. Чтобы определить при вызове метода `drawImage()` координаты области рисования, привязанные к размерам окна апплета, создадим объект класса `Dimension`, объединяющий такие свойства компонента, как ширина и высота. Затем в методе `init()` присвоим созданному объекту класса `Dimension`

размеры окна апплета при помощи метода `getSize()` класса `Applet`, унаследованного им от класса `Component`. И наконец, определим координаты области рисования с помощью методов класса `Dimension` — `height()` и `width()`, возвращающих, в нашем случае, высоту и ширину окна апплета.

6. В вышеуказанном коде мы переопределяем два метода: `init()` и `paint()`. Проинформируем об этом компилятор с помощью необязательной аннотации `@Override`, которую используют, чтобы избежать ошибок. Если аннотация указана и метод не переопределяет ни одного метода суперкласса, то компилятор выдаст ошибку. Например, ошибка возникнет, если мы вместо `paint()` укажем `Paint()` и, таким образом, создадим новый метод, вместо того чтобы переопределить метод `paint()`.

Листинг 1.2. Вывод изображения космического корабля в окно апплета

```
import java.applet.Applet;
import java.awt.*;
public class StarWar extends Applet {
  Image ship;
  Image fire;
  Dimension d;
  @Override
  public void init() {
    // TODO start asynchronous download of heavy resources
    ship=this.getImage(this.getCodeBase(),"StarWar/ship.gif");
    fire=this.getImage(this.getCodeBase(),"StarWar/fire.gif");
    d=this.getSize();
  }
  // TODO overwrite start(), stop() and destroy() methods
  @Override
  public void paint(Graphics g){
    this.setBackground(Color.Black);
    g.drawImage(ship,d.height/2,d.width-80,this);
    g.drawImage(fire,d.height/2,d.width-40,this);
  }
}
```

Для того чтобы посмотреть результаты нашего труда в окне браузера, соберем наш проект. В окне **Projects** среды NetBeans щелкнем правой кнопкой мыши на названии проекта и выберем пункт **Build**. При сборке среда NetBeans использует инструмент ANT проекта Apache (**http://ant.apache.org/**), который обрабатывает файл build.xml NetBeans-проекта, вызывая компилятор `javac` комплекта разработки JDK. В результате сборки в