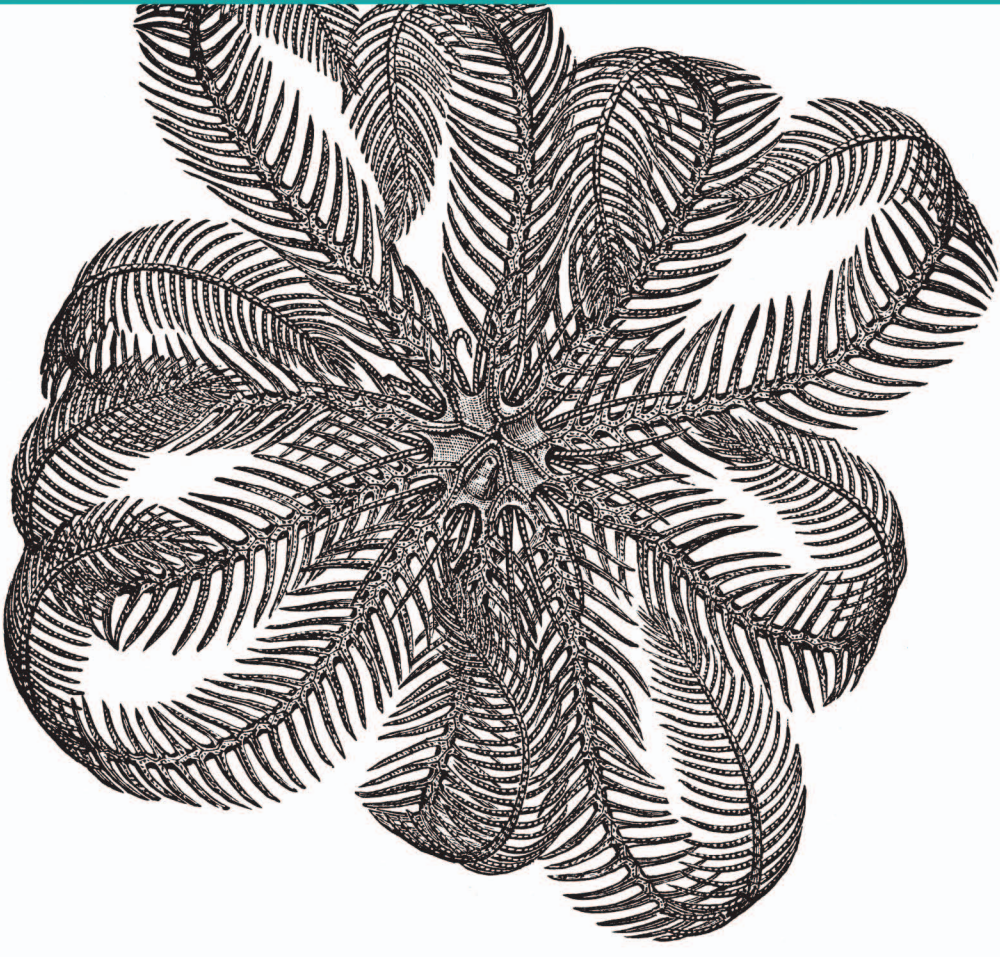
Приемы объектно-ориентированного программирования

ActionScript 3.0
Шаблоны проектирования



O'REILLY®

Adobe Developer Library

Уильям Сандерс
Чандима Кумаранатунг

ActionScript 3.0
Design Patterns

William B. Sanders,
Chandima Cumaranatunge

O'REILLY®

ActionScript 3.0
Шаблоны проектирования

Уильям Сандерс, Чандима Кумаранатунг

Символ®

Санкт-Петербург — Москва
2011

Уильям Сандерс, Чандима Кумаранатунг

ActionScript 3.0. Шаблоны проектирования

Перевод М. Нетова

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Выпускающий редактор	*П. Щеголев*
Редактор	*О. Меркулова*
Научный редактор	*С. Блощинский*
Корректор	*Д. Щепелева*
Верстка	*К. Чубаров*

Сандерс У., Кумаранатунг Ч.

ActionScript 3.0. Шаблоны проектирования. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 592 с., ил.

ISBN 978-5-93286-189-9

Теперь, когда язык ActionScript стал полноценным языком объектно-ориентированного программирования (ООП), часто используемые шаблоны проектирования являются идеальным средством решения многих повторяющихся задач во Flash- и Flex-приложениях. Использование шаблонов не только упрощает планирование и разработку сложных приложений, но и предоставляет решения для многих стандартных проблем, помогает в поддержке и развитии готовых приложений.

В данном издании представлены ключевые особенности ActionScript 3.0, основные ООП-концепции, такие как классы, абстрактность, наследование и полиморфизм, а также преимущества использования шаблонов проектирования. Затем детально рассматриваются конкретные шаблоны: Фабричный метод, Одиночка, Декоратор, Адаптер, Композиция, Команда, Наблюдатель, Стратегия, Состояние, Модель-Представление-Контроллер и Симметричный заместитель. Авторы приводят множество примеров различной степени сложности: веб-приложения для электронной коммерции, динамичные игры, запись и воспроизведение видео и многие другие.

Эта книга необходима любому разработчику Flash или Flex, желающему использовать продвинутые технологии ActionScript 3.0 в создании элегантных программных решений.

ISBN 978-5-93286-189-9
ISBN 978-0-596-52846-1 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, www.symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Подписано в печать 28.12.2010. Формат 70×100 1/16. Печать офсетная.
Объем 37 печ. л. Тираж 1500 экз. Заказ №
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Билл посвящает эту книгу подрастающему поколению,
Рику и Оливии.

Чандима хотел бы посвятить данную книгу
своим родителям, Гайя и Лакшми.

Оглавление

Предисловие

После того как ActionScript эволюционировал от нескольких операторов для Flash до полноценного языка программирования веб-приложений (в своих последних версиях), у нас появилась возможность использовать в своих разработках на нем самые современные технологии, созданные для объектно-ориентированного программирования (ООП). Появление ActionScript 3.0 ознаменовало начало новой эры в программировании Flash и Flex, поскольку теперь он соответствует стандарту ECMAScript для языков разработки в Интернет. Многие возможности таких языков, как C++ и Java, доступны теперь и в ActionScript 3.0.

Вместе с новыми возможностями в языке ActionScript 3.0 к нам пришли современные способы разработки, изменился даже сам процесс *мышления* на данном языке. Хотя мы предполагаем, что большинство читателей этой книги в некоторой степени знакомы с ООП, но, подобно переходу от последовательного или процедурного программирования к ООП, переход к программированию с использованием шаблонов проектирования является скачком вверх для ООП-программистов. Мы думаем, что поскольку в ActionScript 3.0 появились возможности разрабатывать более сложные программные структуры, программистам Flash и Flex потребуются соответствующие техники программирования для успешной работы с ними.

Владея процессом программирования с использованием шаблонов проектирования[1], вы можете писать более эффективный ООП-код и многократно использовать его в других ваших программах. Разработчики, умеющие работать в больших командах и разбирающиеся в современных структурах, для оперирования с которыми были созданы шаблоны проектирования и ООП, занимают более высокооплачиваемые позиции. Использование шаблонов проектирования не только повышает ваше мастерство разработки сложных приложений, но и облегчает само программирование. Большинство трудностей при создании крупных сложных приложений возникает из-за их плохого планирования и неудобных конструкций. Шаблоны проектирования не только предоставляют решения для многих стандартных проблем, но и помогают в процессах обслуживания и изменения готовых программ. Словарь шаблонов проектирования также очень важен, поскольку его освоение позволит вам

[1] В разговорной речи обычно говорится просто «шаблон» или «паттерн». – *Прим. перев.*

стать членом современного сообщества разработчиков, свободно разговаривающего на этом языке.

Для кого эта книга

Мы планировали написать книгу для пользователей ActionScript 3.0 с уровнем подготовки от среднего до продвинутого. В отличие от разработчиков на некоторых других языках программирования, таких как Java, которые обычно являются выпускниками компьютерных факультетов, большинство пользователей ActionScript 3.0 изучили его в процессе работы во Flash. В результате уровень их теоретических знаний варьируется значительно сильнее, как и их общий практический опыт программирования. Мы, конечно, понимаем, что некоторые программисты на ActionScript 3.0 имеют профессиональное высшее образование, и для них большая часть вводного материала из первой главы будет излишней. Но мы также уверены, что часть читателей с уровнем подготовки пониже будет изучать объектно-ориентированное программирование одновременно с освоением стандартных шаблонов проектирования, и у них, возможно, меньше практики программирования.

При таком разбросе исходных позиций обязательно будет что-то слишком сложное для одних и чересчур простое для других. Однако главная задача этой книги – объяснить, как следует использовать различные шаблоны проектирования. Наша целевая аудитория – разработчики ActionScript со средним уровнем подготовки. Сначала мы приводим весь материал, необходимый им, чтобы стать продвинутыми. А уже для продвинутого пользователя мы предоставляем объяснения и примеры применения шаблонов проектирования в ActionScript 3.0.

Как организована данная книга

Структура данного издания во многом повторяет структуру книги «Design Patterns: Elements of Reusable Object-Oriented Software», написанной Эриком Гаммой (Erich Gamma), Ричардом Хелмом (Richard Helm), Ральфом Джонсоном (Ralph Johnson) и Джоном Влиссидесом (John Vlissides) (издательство Addison-Wesley, 1995)[1].

Первая часть данной книги содержит только главу 1, которая является вводной для темы шаблонов проектирования. Мы добавили ее для читателей с минимальным уровнем знания объектно-ориентированного программирования. Более опытным пользователям, вероятно, можно

[1] Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес «Приемы объектно-ориентированного проектирования. Паттерны проектирования». – Пер. с англ. – СПб.: Питер, 2007.

пропустить этот краткий обзор ООП и сразу перейти к материалу, посвященному шаблонам проектирования.

Часть I «Постоянные изменения».

Глава 1 «Объектно-ориентированное программирование, шаблоны проектирования и ActionScript 3.0».

Части II, III и IV – главные в этой книге. В них рассматриваются базовые шаблоны проектирования, сгруппированные по категориям: шаблоны *порождающие, структурные* и *поведенческие*. Примеры таких шаблонов представлены в каждой части, но мы не включали в них абсолютно все шаблоны из книги Э. Гаммы и его коллег.

Часть II «Порождающие шаблоны».

Глава 2 «Шаблон Фабричный метод».

Глава 3 «Шаблон Одиночка».

Часть III «Структурные шаблоны».

Глава 4 «Шаблон Декоратор».

Глава 5 «Шаблон Адаптер».

Глава 6 «Шаблон Компоновщик».

Часть IV «Поведенческие шаблоны».

Глава 7 «Шаблон Команда».

Глава 8 «Шаблон Наблюдатель».

Глава 9 «Шаблонный метод».

Глава 10 «Шаблон Состояние».

Глава 11 «Шаблон Стратегия».

Часть V состоит из двух глав, посвященных использованию множественных шаблонов при разработке программ. Конструкции типа Модель-Представление-Контроллер и Симметричный заместитель включают в себя более одного шаблона проектирования. Эти главы организованы подобно прочим в плане объяснения работы сложных шаблонов. Однако многовариантные конструкции основываются больше на диаграммах объектов, чем на диаграммах классов.

Часть V «Множественные шаблоны».

Глава 12 «Шаблон Модель-Представление-Контроллер».

Глава 13 «Шаблон Симметричный заместитель».

Каждая глава, посвященная шаблонам проектирования, организована так, чтобы обеспечить максимально ясное понимание назначения и способа использования данного шаблона. Следующие разделы, хотя и не обязательно в таком порядке, присутствуют в каждой главе о шаблонах проектирования:

- Что это за шаблон?
- Основные характеристики и возможности данного шаблона.
- Формальная модель шаблона, включая его диаграмму классов.
- Ключевые концепции ООП, использованные в данном шаблоне.
- Небольшой абстрактный пример применения шаблона.
- Прикладные примеры его использования.

Мы построили нашу книгу таким образом для того, чтобы представить нашим читателям всестороннюю картину для каждого шаблона проектирования. Обсуждая предназначение шаблона и его свойства, мы уделяем основное внимание его функциям и структуре. Формальная модель и диаграмма классов дают нам возможность шире взглянуть на данный шаблон, вы познакомитесь с его структурой и связанными с ним классами и интерфейсами. Мы включили также рассмотрение некоторых ключевых концепций ООП для различных шаблонов. Сделали мы это по двум причинам. Во-первых, пользователи среднего уровня смогут лучше усвоить эти концепции на практике, а значит, и глубже понять ООП в целом. Во-вторых, мы рассчитываем, что продвинутые пользователи смогут использовать эти концепции как условные обозначения для быстрого определения того, как данный шаблон проектирования структурирован.

Что необходимо для работы с книгой

Для изучения приводимых примеров программ вам потребуется либо Flash CS3, либо Flex 2. Все приложения были разработаны с помощью Flash IDE, поэтому пользователям Flex 2 потребуется вносить в них некоторые изменения, особенно в те места, где для создания возможностей использовались инструменты и компоненты рисования Flash.

Несколько примеров используют Flash Media Server 2 (FMS2). Такие примеры могут быть созданы и опробованы с помощью версии FMS2 для разработчиков, доступной для бесплатной загрузки по адресу *http://www.adobe.com/downloads/*. Вам потребуется либо ОС Windows, либо ОС Linux для запуска FMS2. Если у вас Макинтош типа «Mactel», вы можете запустить на нем ОС Windows, но если у вас более старая версия Макинтоша, работающая на процессоре Motorolla, вам потребуется локальный или интернет-доступ к Windows или Linux-серверу с FMS2. Либо вы можете просто пропустить примеры с FMS2.

Повторение – мать учения

Читая эту книгу, вы заметите, что материал в ней иногда повторяется. Мы знаем, что у разных людей стиль обучения различен. Одним достаточно общей концепции, вторые учатся на примерах, третьи – на мета-

форах или комбинациях вышеперечисленного. Плюс к этому, возможно, существуют и другие методики, о которых мы не слышали. В книге мы объясняем одни и те же вещи несколькими различными способами в надежде на то, что если вы не поймете материал в первом варианте, то уж точно поймете во втором.

Кроме того, мы полагаем, что обсуждение одних и тех же идей и концепций по-разному и в различных контекстах формирует лучшее *сущностное* понимание этих идей и концепций. Изучив немалый материал по теме шаблонов проектирования, включая опубликованные книги, статьи и информацию в Интернете, мы обнаружили, что некоторые детали описаны там недостаточно аккуратно. Мы очень старались избежать в нашей книге ошибок; рассмотрение понятий в различных контекстах дало нам возможность более четко формулировать то, что мы имеем в виду. Решающее слово неизменно останется за книгой «Design Patterns: Elements of Reusable Object-Oriented Software», поэтому если у вас останутся какие-либо вопросы по нашим разъяснениям, вы всегда сможете уточнить их в указанном оригинале.

В течение нескольких последних лет появился ряд статей, книг, диссертаций и исследований, предлагающих дальнейшее развитие изначальных шаблонов проектирования. Некоторые из этих документов были весьма полезны и даже поддержаны членами «банды четырех» (GoF) – Эриком Гаммой (Erich Gamma), Ричардом Хелмом (Richard Helm), Ральфом Джонсоном (Ralph Johnson) и Джоном Влиссидесом (John Vlissides). Другие оказались менее удачными, особенно в смысле легкости изучения по ним шаблонов проектирования, и имели тенденцию еще больше усложнять и без того непростую тему. Исходя из этого мы не стали сильно отклоняться от пути, указанного нам в исходном тексте «банды четырех».

Руководство пользователя

Эта книга по своей сути является введением в относительно сложную тему написания повторно используемого ООП-кода на ActionScript 3.0. Как и словосочетание «гигантская букашка», фраза «элементарное введение в сложную тему» является оксюмороном. Опытные разработчики, вероятно, захотят, чтобы в ней было меньше элементарного, а менее продвинутые программисты могут потребовать, чтобы в ней было больше подготовительного материала.

Поскольку мы не можем определить уровень подготовки каждого нашего читателя, мы убедительно просим вас прочитать оглавление, пролистать заинтересовавшие вас главы и найти то, что нужно именно вам. Определите свой текущий уровень и используйте книгу в соответствии с ним. Кто-то прочтет ее от корки до корки, а для кого-то данная книга послужит только справочным материалом для поиска того, как

ActionScript 3.0 работает с различными вариантами шаблонов проектирования. В конце концов, это ваша книга, и вы должны использовать ее с максимальной для себя пользой.

Использование Flex 2

Мы разрабатывали все наши примеры с использованием Flash CS3. Так что если вы захотите использовать их во Flex 2, вам потребуется их несколько изменить. В некоторых случаях примеры были созданы с применением пользовательских классов на основе MovieClip, разработанных в IDE Flash CS3 и сохраненных в Библиотеке. Такие примеры невозможно напрямую повторить во Flex 2, поэтому вам придется строить обходные пути.

Использование Flash Media Server 2

В книге есть несколько примеров, использующих Flash Media Server 2 (FMS2). Для их работы требуется только версия FMS2 для разработчиков, которую можно бесплатно загрузить с сайта Adobe по адресу: *http://www.adobe.com/devnet/flashmediaserver/*. Либо вы можете пропустить эти примеры или использовать другое программное обеспечение.

Что еще вам потребуется

В идеальном мире все читатели нашей книги прекрасно знали бы тему объектно-ориентированного программирования и язык ActionScript 3.0 заранее. Однако ActionScript 3.0 появился (не в бета-версии) во Flex 2 всего за шесть месяцев до выхода оригинала этой книги, а во Flash – примерно во время ее публикации. Так что есть вероятность, что вы совсем не знакомы с ActionScript 3.0, а данная книга вовсе не является введением в этот язык. Тогда как минимум вам следует держать под рукой «ActionScript 3.0 Reference Guide», а также прочую документацию по нему, идущую вместе с Flash CS3.

Мы настоятельно рекомендуем вам приобрести книгу «Design Patterns: Elements of Reusable Object-Oriented Software». По крайней мере, возьмите ее на время в библиотеке. Еще одна бесценная, на наш взгляд, книга – это чрезвычайно легкая и понятно написанная «Head First Design Patterns» Эрика и Элизабет Фриман (Eric Freeman, Elisabeth Freeman) (издательство O'Reilly, 2004). Хотя там все примеры написаны на Java, из нее можно узнать очень многое о шаблонах проектирования и ООП. (Перевод Java-примеров на язык ActionScript 3.0 позволит вам отлично изучить его, но перед этим занятием обязательно просмотрите примеры шаблонов проектирования в нашей книге.)

Если у вас еще нет хорошей книги по ActionScript 3.0, советуем приобрести «ActionScript 3.0 Cookbook» Джои Лотта (Joey Lott), Деррона Шалла (Darron Schall) и Кейт Питерс (Keith Peters) (издатель-

ство O'Reilly, 2006)[1] и «Essential ActionScript 3.0» Колина Мука (Colin Moock) (издательство O'Reilly, 2007)[2]. Для очень краткого введения в ActionScript 3.0 прочитайте сокращенное издание «ActionScript 3.0 Programming: Overview, Getting Started, and Examples of New Concepts» Билла Сандерса (Bill Sanders) (издательство O'Reilly, 2007). Еще вам потребуется книга по ООП на ActionScript 3.0. Мы включили введение в ООП в первую главу нашей книги, его будет пока достаточно; книги К. Мука и Дж. Лотта также содержат много материала по этой теме. Однако книга, посвященная основным концепциям ООП, все-таки должна присутствовать в вашей библиотеке, если вы собираетесь писать программы на уровне шаблонов проектирования.

Соглашения, принятые в этой книге

В этой книге используются следующие типографические обозначения:

Рубленый шрифт

Применяется для выделения интерфейса – меню, кнопки, быстрые клавиши (такие как Alt и Ctrl).

Жирный шрифт

Указывает на текст, который должен быть введен пользователем.

Курсив

Указывает на новые термины, веб-ссылки (URL), адреса электронной почты, имена файлов и их расширения, пути поиска файлов, имена каталогов и утилиты ОС UNIX.

Моноширинный шрифт

Указывает на команды, опции, переключатели, переменные, атрибуты, ключи, функции, типы, классы, пространства имен, методы, модули, свойства, параметры, значения, объекты, события, обработчики событий, XML-метки, HTML-метки, а также на результаты работы команд.

Жирный моноширинный шрифт

Обозначает текст, который должен быть введен пользователем дословно.

Моноширинный курсив

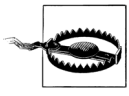
Обозначает текст, который должен быть заменен значениями, определяемыми пользователем.

---

[1] Дж. Лотт, Д. Шалл, К. Питерс «ActionScript 3.0. Сборник рецептов». – Пер. с англ. – СПб.: Символ-Плюс, 2007.

[2] К. Мук «ActionScript 3.0 для Flash. Подробное руководство». – Пер. с англ. – СПб.: Питер, 2009.

Таким способом выделяются советы, предложения и примечания общего характера.



Таким способом выделяются предупреждения и предостережения.

Использование примеров программ

Эта книга призвана помочь вам в работе. В общем вы можете использовать приведенный в ней код в собственных программах или в создаваемой вами документации. Спрашивать у нас разрешения необязательно, если только вы не собираетесь воспроизводить значительную часть кода. Например, не требуется разрешение, чтобы включить в свою программу несколько фрагментов кода из книги. Однако для продажи или распространения примеров на компакт-диске нужно получить разрешение. Можно без ограничений цитировать книгу и примеры в ответах на вопросы. Но чтобы включить значительные объемы кода в документацию по собственному продукту, нужно получить разрешение.

Мы будем благодарны вам за ссылку на данную книгу при цитировании, хотя и не требуем этого. В ссылке обычно приводятся название книги, имена авторов, издательство и ISBN, например: «ActionScript 3.0 Design Patterns by Bill Sanders and Chandima Cumaranatunge. Copyright 2007 O'Reilly Media, Inc., ISBN 978-0-596-52846-1».

Если вы полагаете, что планируемое вами использование кода выходит за рамки законного или разрешений, оговоренных выше, пожалуйста, обращайтесь к нам по адресу *permissions@oreilly.com*.

Как с нами связаться

Пожалуйста, присылайте все комментарии и вопросы, касающиеся данной книги, в издательство:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (в США или Канаде)
707-829-0515 (международный или местный)
707-829-0104 (факс)

На веб-странице данной книги содержатся найденные в ней ошибки, все ее примеры и дополнительная информация. Адрес этой страницы:

http://www.oreilly.com/catalog/9780596528461

Все комментарии и вопросы по данной книге присылайте по адресу:

bookquestions@oreilly.com

Чтобы узнать больше информации о наших книгах, конференциях, центре ресурсов и сети издательства O'Reilly, посетите наш веб-сайт:

http://www.oreilly.com

Safari® Enabled

Если на обложке технической книги есть пиктограмма Safari® Enabled, это означает, что книга доступна в Сети через O'Reilly Network Safari Bookshelf.

У Safari есть преимущество перед обычными электронными книгами. Это виртуальная библиотека, которая позволяет легко находить тысячи лучших технических изданий, копировать примеры программ, загружать отдельные главы и быстро получать точную и актуальную информацию. Библиотека бесплатна и расположена по адресу *http://safari.oreilly.com*.

Благодарности

С тех пор как благодаря аЙо Бинити (aYo Binitie) мы познакомились с шаблонами проектирования, они стали предметом нашего пристального интереса. Как и многие разработчики на ActionScript, мы благодарны Колину Муку (Colin Moock) за начальное освещение темы шаблонов проектирования на ActionScript в книге «Essential ActionScript 2.0»[1]. Мы благодарны также Эрику Фриману (Eric Freeman) и Элизабет Фриман (Elisabeth Freeman) за их замечательную книгу «Head First Design Patterns»; хотя все примеры в ней на Java, она помогла нам оценить применение шаблонов проектирования в ActionScript 3.0. И как основателям темы шаблонов проектирования мы должны выразить нашу признательность «банде четырех», написавшей «Design Patterns: Elements of Reusable Object-Oriented Software», – Эрику Гамме (Erich Gamma), Ричарду Хелму (Richard Helm), Ральфу Джонсону (Ralph Johnson) и Джону Влиссидесу (John Vlissides). Мы провели немало времени, сосредоточенно изучая эту книгу.

Несколько людей из компании Adobe были так добры, что уделили нам свое время, помогли с ActionScript 3.0 и дали нам несколько идей насчет шаблонов проектирования. Вот они: Крис Нууйа (Chris Nuuja), Эрика Нортон (Erica Norton), Джеффри Уильямс (Geoffrey Williams), Грант Скиннер (Grant Skinner), Джеффри Мотт (Jeffrey Mott), Майк Дауни (Mike Downey), Нивеш Райбхандари (Nivesh Rajbhandari), Питер

[1] К. Мук «ActionScript 2.0. Основы». – Пер. с англ. – СПб.: Символ-Плюс, 2006.

Дехаан (Peter DeHaan), Роберт Пеннер (Robert Penner), Гари Гроссман (Gary Grossman), Али Миллс (Ali Mills), Френсис Ченг (Francis Cheng), Дэйвид Менделс (David Mendels), Гордон Смит (Gordon Smith), Роджер Гонсалез (Roger Gonzalez), Шо Кувамото (Sho Kuwamoto), Фрэнсис Чен (Francis Chen), Эмми Хуанг (Emmy Huang), Вернер Шарп (Werner Sharp), Джоан Тан (Joan Tan), Фил Коста (Phil Costa), Молли Гардинер (Mally Gardiner), Аса Вилок (Asa Whillock), Крис Хок (Chris Hock), Тарек Альджабер (Tareq Aljaber), Сан Хонг (San Khong) и Питер фон дем Хаген (Peter von dem Hagen).

Некоторые разработчики из Flash-сообщества помогли нам глубже понять тему использования шаблонов проектирования в ActionScript 3.0. Это Питер Холл (Peter Hall), Арал Балкан (Aral Balkan), Роберт Пеннер (Robert Penner), Бо Амбер (Beau Ambur), Стефан Рихтер (Stefan Richter), Джои Лотт (Joey Lott), Гай Ватсон (Guy Watson), Кейт Питерс (Keith Peters), Уил Лоу (Will Law) и Брайан Лессер (Brian Lesser). Джонатан Кей (Jonathan Kaye), реализовавший абстрактный автомат на языке ActionScript, посодействовал нам с шаблоном «Состояние», который послужил моделью для реализации в данной книге.

Мы очень благодарны также Марго Малей Хатчисон (Margot Maley Hutchison) из компании Waterside Productions за помощь при заключении контракта с издательством O'Reilly. Как и всегда, Марго удалось сильно упростить этот процесс.

Мы благодарны профессору Джону Грею (John Gray), главе программы «Мультимедийный веб-дизайн и разработка» (Multimedia Web Design and Development, MVDD) в университете Хартфорда. Его энтузиазм и поддержка по всем вопросам, касающимся Интернета и веб-разработок, исследований и обучения, обеспечили нам богатую идеями среду и множество удивительных открытий на нашем пути в неизменно расширяющуюся вселенную исследуемых нами технологий.

Технические консультанты

Наши технические консультанты сделали для нас гораздо больше, чем им полагалось. Некоторые из них были экспертами по шаблонам проектирования на C# и Java, но были не знакомы с ActionScript. К счастью, ActionScript 3.0 выглядит и действует подобно прочим языкам ООП, поэтому они смогли нам помочь. Главной в этой группе была Адриана Декер (Adrienne Decker), преподаватель департамента компьютерных и инженерных наук государственного университета Нью-Йорка (SUNY), г. Буффало. После посещения ее выступления на конференции по объектно-ориентированному программированию, системам, языкам и приложениям (OOPSLA) в 2006 г. в Портленде, штат Орегон, стало ясно, что Адриана не только очень много знает о шаблонах проектирования, но и умеет просто и доходчиво рассказывать об их внутренней работе. Она нам очень помогла. В этом проекте участвовали еще

двое ученых: доктор Джеймс Хейлотис (James Heilotis) и доктор Аксель Шрейнер (Axel Schreiner), оба из Рочестерского института технологий, департамент компьютерных наук. Профессор Шрейнер представил на конференции OOPSLA 2006 доклад об их шаблоне проектирования Симметричный заместитель. Как показано в их статье «A Pattern for Distributing Turn-Based Games», новый шаблон оказался отличным инструментом для создания интернет-игр, использующих Flash и Flash Media Server 2. Мы полагаем, что, учитывая новизну и необычность шаблона Симметричный заместитель, было совершенно обоснованно попросить их о технической рецензии на наше объяснение и реализацию этого шаблона. Они нам очень помогли, проследив за верным изложением нами их идей, и мы им обоим очень благодарны.

Нам повезло работать с Тоддом Андерсеном (Todd Anderson). Тодд отлично разбирается в ActionScript 3.0 и шаблонах проектирования. Он чрезвычайно помог нам, и мы особенно благодарны ему за его острый взгляд и точные комментарии. Даррен Ричардсон (Darren Richardson) был нашим техническим консультантом с самого начала нашей работы. Он также привнес в наши работы рассмотрение некоторых аспектов интернационализации, не дав нам стать слишком этноцентричными.

У нас была также уникальная возможность поработать с сообществом Flash-программистов Нью-Йорка (FCNY), возглавляемым Джином-Чарльзом Карелли (Jean-Charles Carelli). Проявив большой энтузиазм, члены этой группы неоднократно просматривали и редактировали наши рукописи в качестве дополнительных технических консультантов. Их комментарии были очень полезны для нас и отражали интересный срез их опыта как разработчиков-практиков. Среди тех, кто особенно способствовал этому процессу, были Тайлер Ларсон (Tyler Larson), Джим Кременс (Jim Kremens), Доминик Танкреди (Dominic Tancredi), Шари Холтер (Shari Halter), Джеймс О'Рейли (James O'Reilly), Эндрю Хант (Andrew Hunt), Брайан Вейсентал (Brian Weisenthal), Оскар Треллес (Oscar Trelles), Сет Хилинджер (Seth Hillinger), Лиза Ларсон (Lisa Larson) и Эдвард Скрипа (Edward Skrypa).

Редакторы

Нашему ведущему редактору, Робин Томас (Robyn Thomas), приходилось не только выделять смысл из наших высокотехнологичных текстов, но и направлять нас в правильном использовании английского языка, гарантирующем, что мы говорим именно то, что подразумеваем. Она управляла также множеством технических редакторов, работающих с главами нашей книги, собирала все эти кусочки вместе, расставляя их по своим местам, и вообще делала процесс написания книги настолько приятным, насколько это только возможно. Стив Вейс (Steve Weiss), наш издатель, всячески нас поддерживал и следил за общей организацией всего процесса. Стив всегда открыт новым перспективам

и всему креативному; скорее всего потому, что он и сам очень творческая личность.

Авторы

Билл Сандерс (Bill Sanders), PhD, и Чандима Кумаранатунг (Chandima Cumaranatunge), PhD, являются профессорами программы мультимедийного веб-дизайна и проектирования (MWDD) в университете Хартфорда. Билл ведет курсы по Flash, ActionScript, Flash Media Server 2, PHP, C#, SQL, CSS, XHTML и другим языкам для разработки веб-приложений. Он опубликовал 44 книги по компьютерной тематике, создавал программное обеспечение на языках от Basic до Flash Media Server ActionScript и работал консультантом в различных программистских компаниях. Чандима читает введение в MWDD, в основном посвященное Flash и ActionScript, курс по играм на Flash и ActionScript и курс по образовательным технологиям в медицинском колледже. Недавно он получил грант на преподавание экспериментального курса по робототехнике.

Билл Сандерс

Билл хотел бы поблагодарить своего соавтора, Чандима, за приятное и плодотворное сотрудничество в написании данной книги и за то, что с ним всегда можно было профессионально обсудить тему шаблонов проектирования. Билл хотел бы также поблагодарить свою жену Делию за то, что, пока она заканчивала свою докторскую диссертацию, ей приходилось терпеть наше сосуществование в режиме двух вечно занятых текстовых процессоров. Наш маниакально навязчивый большой швейцарский зенненхунд Вилде возвращал нам обоим чувство реальности; он всегда знал, что действительно важно – сходить погулять.

Чандима Кумаранатунг

Чандима считает, что ему очень повезло: он нашел в Билле не только партнера по написанию книги, но и профессионального наставника и настоящего друга. Он очень благодарен также своей жене Решмааль за ее всемерную поддержку и терпение по отношению к его долгой писательской работе вдали от семьи. И наконец, своей дочери Саюри, двухлетней «маленькой лилии в океане» (как переводится ее имя с японского и сингальского языков), за то, что она сохраняла рассудок своего папы, напоминая ему каждый день о самых важных вещах в жизни.

I

Постоянные изменения

Мы должны стать теми переменами,
которые мы хотим видеть в мире.

Махатма Ганди

Те, кто хочет пребывать в вечном блаженстве
мудрости, должны часто меняться.

Конфуций

Без изменений что-то спит внутри нас
и редко просыпается. Спящий должен проснуться.

Фрэнк Герберт

Жизнь принадлежит живущим.
И тот, кто живет, должен быть готов к переменам.

Иоганн Вольфганг фон Гете

Если бы потребовалось описать шаблоны проектирования одним предложением, мы бы сказали, что это *инструменты, помогающие справиться с постоянными изменениями* при проектировании и разработке программного обеспечения. Если вы посмотрите на разные шаблоны проектирования, представленные в книге, вы увидите, что все они оптимизированы для того, чтобы позволить программисту изменять и многократно использовать большую часть созданного им кода. Ключевыми понятиями являются изменение и гибкость. Заданная тема будет еще много раз рассматриваться в нашей книге. А часть I предоставляет общую информацию, необходимую для понимания и использования книги.

Для работы с шаблонами проектирования вам необходимо знать основные принципы объектно-ориентированного программирования (ООП). Если вы с ними не знакомы, хорошенько изучите главу 1. Ее заключительная часть переходит к обсуждению некоторых базовых для шаблонов проектирования вещей, их понимание поможет вам лучше разобраться в следующих главах, посвященных конкретным шаблонам.

Даже если вы уже знаете такие основополагающие понятия ООП, как абстрактность, инкапсуляция, наследование и полиморфизм, это еще не означает, что вы хорошо применяете ООП на практике. Чтобы считаться профессиональным проектировщиком и разработчиком программных приложений, вам необходимо уметь проектировать их так, чтобы их можно было легко обслуживать и развивать согласно новым требованиям. Другими словами, вам нужно создавать приложения, отражающие реальный мир. Инструменты, применяемые вами сегодня для разработки веб-приложений, должны позволять вам легко проводить изменения и обновления ваших приложений, а также повторно использовать имеющиеся наработки. В противном случае ваши приложения будут неспособны к адаптации под требования текущего дня.

Шаблоны проектирования дают образцы объектно-ориентированного проектирования приложений, которые позволяют справиться с изменениями в программах посредством различных средств ООП. И если вы думаете об использовании ООП как о методе создания проектов, способном помочь вам разобраться с переделками в приложениях, повторно использовать имеющиеся наработки и добавить гибкости вашим программам, то вы уже начинаете мыслить в духе шаблонов проектирования.

Глава 1 «Объектно-ориентированное программирование, шаблоны проектирования и язык ActionScript 3.0».

1

ООП, шаблоны проектирования и язык ActionScript 3.0

Приучите себя исследовать замысел действий людей и размышляйте об их возможных последствиях как можно чаще; а чтобы такой анализ стал еще более значительным, применяйте его в первую очередь по отношению к самим себе.

Марк Аврелий

История жизни личности – это прежде всего подстраивание под шаблоны и стандарты, традиционно принятые в обществе.

Рут Бенедикт

На нижнем когнитивном уровне находятся процессы восприятия, или, вообще говоря, интуитивные процессы, воспринимающие объекты в оригинале.

Эдмунд Гуссерль

Удовольствие от хорошей работы

Основной идеей применения шаблонов проектирования является выделение некоторого набора таких шаблонов и решение с его помощью всех похожих задач. Кроме того (можно сказать, неразрывно с этим), шаблоны являются некоторым стандартом хороших, общепринятых практик объектно-ориентированного программирования (ООП). Поэтому мы не можем (да и не хотим) отделять шаблоны проектирования от ООП. Отвечая на вопрос, зачем нужно использовать стандартные шаблоны проектирования, мы в действительности размышляем над вопросом, зачем нужно использовать ООП. Традиционный ответ на оба этих вопроса состоит в том, что благодаря шаблонам и ООП разработчики внутри одной

команды получают возможность разговаривать между собой на одном языке. Более того, используя объектно-ориентированные идеологии и практики программирования, проще справляться со сложностями, связанными с программистскими задачами, требующими разделения труда.

Кроме как для решения вопросов координации работ в больших проектах, программисты используют ООП и шаблоны проектирования для последующего внесения изменений в свои программные системы. Одной из их важнейших, ключевых характеристик шаблонов является возможность с их помощью гораздо легче изменять реализованные программы. Чем длиннее ваша программа и чем больше времени вы потратили на ее разработку, тем сложнее будет вносить в нее изменения и тем более непредсказуемыми будут последствия таких действий. Изменение кода в программе может иметь для нее разрушительные последствия, подобные распусканию свитера в результате вытягивания из него одной нити. Шаблоны проектирования и следование правилам ООП значительно облегчают внесение изменений в сложные программы, а также уменьшают объем этих изменений и снижают вероятность возникновения глобальных проблем.

Упрощение решения задач координации работ в команде разработчиков и изменения и поддержки существующего приложения уже является достаточным основанием для изучения шаблонов проектирования. Однако представим, что ваша программа относительно короткая, вы работаете самостоятельно и вас особо не волнуют дальнейшие изменения в вашей программе. В таком случае бывает быстрее просто переписать программу заново. Зачем вам тогда вообще нужно знать про шаблоны проектирования?

Помимо того обстоятельства, что ActionScript 3.0 базируется на ECMAScript и, вероятно, не будет так сильно меняться с каждым новым релизом Flash или Flex, как в прошлом, у вас может быть личная причина для изучения шаблонов проектирования. Александр Нахимовский (Alexander Nakhimovsky) и Том Майерс (Tom Myers) писали в своей книге, посвященной ООП и JavaScript (издательство Wrox, 1998), о ценности удовольствия от качественно проделанной работы. Как и при любом другом занятии, будь то катание на скейтборде или строительство дома, его хорошее выполнение приносит человеку радость. Под словами «сделать хорошо» мы вовсе не подразумеваем некий навязчивый перфекционизм, главным образом потому что он часто ведет к полному параличу во всех делах. Мы скорее говорим о такой работе, в ходе и по итогам которой вы испытываете удовлетворение искусного мастера от творческого процесса и от верных результатов.

Последовательное и процедурное программирование

Если вы никогда не слышали о *последовательном программировании*, то, скорее всего, именно этим видом программирования вы и занимае-

тесь. Большинство непрофессиональных программистов просто пишут в своей программе одно выражение за другим, и при правильной последовательности таких выражений она работает вполне сносно. Однако по мере того как программы все более и более усложнялись, у разработчиков образовывались невообразимые кучи кода, называемые часто *спагетти-кодом* (из-за такой же запутанности). Во избежание подобного эффекта программисты стали организовывать свои программы в виде множеств отдельных процедур и правил их использования. Это и стало началом *процедурного программирования*. Использование подпрограмм вместо неизбежных выражений GOTO, переносящих нас в различные места программы, позволило разбить поток исполнения команд на отдельные модули с соответствующими вызовами GOSUB/RETURN с сохранением опрятного вида всего кода..

Команда RETURN имела раньше немного другое значение, чем сейчас. Ранее RETURN означала возврат на то место в последовательности кода программы, откуда сработала команда GOSUB. В языке ActionScript команда return означает, что данная операция (метод или процедура) высылает назад некоторую свою информацию.

Из процедурного программирования к нам пришло понятие *области видимости*, в которой переменные функций или подпрограмм могут быть использованы программистом повторно, причем ни одна из процедур не повлияет на другую.

Подавляющее большинство языков программирования в наши дни считается процедурными в том смысле, что они концептуально и синтаксически поддерживают данный подход. Различные версии языка Basic являются процедурными, как и языки типа ColdFusion, PHP и Perl. Однако процедурные языки типа C++, ECMAScript (ActionScript 3.0) и Ada считаются скорее объектно-ориентированными языками программирования. Языки типа Java относят к чисто объектно-ориентированным. Не углубляясь сильно в детали, скажем, что язык Java – чисто ООП-язык, поскольку единственным видом процедур в Java является метод класса. Его конструкция вынуждает процедуры работать как методы класса и не позволяет другим процедурам действовать вне структуры классов.

Возможно, вы удивитесь, узнав, насколько острыми бывают дискуссии на тему, является некоторый язык программирования объектно-ориентированным или нет. В отношении этого вопроса существует два варианта критериев. Один из них весьма либерален и позволяет любому языку, в котором есть некие возможности генерировать ООП-код, считаться ООП-языком. (ActionScript 3.0 относится к такого рода языкам.) Согласно другому, гораздо более жесткому варианту *только те* языки, в которых все процедуры являются исключительно методами классов, могут быть допущены в эксклюзивный клуб ООП-языков. Оба этих варианта имеют свои весомые аргументы. Мы обойдем эту дис-

куссию и не будем принимать в ней ничью точку зрения, однако мы отметим, что сторонники обеих вполне согласны с тем, что с помощью процедурных языков программирования можно создавать хороший объектно-ориентированный код.

Короче говоря, вышесказанное не означает, что на других языках программирования невозможно писать настоящие ООП-программы. Разработчики с успехом их реализовывали еще задолго до появления Java. Некоторые языки программирования, особенно использующие структуры классов и перегружаемые методы типа ActionScript 3.0, являются более близкими к ООП-концепциям, чем другие. Язык ActionScript неуклонно приближался к ним по мере своего развития от набора нескольких синтаксических выражений до стандарта ECMAScript.

Переход к ООП

Переход от последовательного или процедурного к объектно-ориентированному программированию – это нечто большее, чем просто выбор языка, предоставляющего вам такую возможность (такого, например, как Java). Определенные изменения в языке программирования могут сделать его более пригодным для применения ООП, даже если он не является чисто ООП-языком согласно некоторым критериям. В следующих разделах мы кратко опишем некоторые новые особенности во Flash CS3, изменяющие способ использования ActionScript.

Код на кнопках и мувиклипах

Как и ветераны ActionScript, начинавшие еще с написания небольших скриптов, использующих команду `on` и ассоциированных с объектами кнопок и клипами (`MovieClip`, `Button`), вы, вероятно, знаете, что последняя версия Flash вообще не содержит механизмов для написания подобного встроенного кода.

Встроенные автоматы: хотя большинство программистов приветствовало кончину внутренних сценариев для кнопок и клипов, один более проницательный из них заметил, что ранее во Flash были встроенные автоматы состояний. Джонатан Кей (Jonathan Kaye), соавтор книги «Flash MX for Interactive Simulation: How to Construct and Use Device Simulations» (издательство Delmar Learning, 2002), обратил внимание на то, что такие скрипты кнопок и клипов часто служили для создания внутренних автоматов в программах. Каждая кнопка или клип могли служить изолированным, чувствительным к контексту триггером для изменения их состояния. (О том, как шаблоны проектирования реализуют объекты с изменяющимся состоянием, читайте в главе 10.)

Вообще утрата внутренних сценариев для кнопок и клипов воспринимается как благоприятная возможность для улучшения программ, осо-

бенно для применения ООП. Отслеживание в программе всех изолированных скриптов для кнопок и клипов могло причинять головную боль даже в случае относительно небольшого приложения. А при построении крупных программ, в которых исключительно полезны ООП и шаблоны проектирования, множество разнообразного кода на кнопках и клипах превращало эту проблему из головной боли в сущий кошмар. Поэтому, изучая шаблоны проектирования, радуйтесь, что вам не нужно думать о маленьких сценариях, изолированных в клипах и кнопках.

Код на временной диаграмме

Другой тип сценариев, который вы будете видеть здесь гораздо реже, чем во Flash, – сценарии, привязанные к временной диаграмме (Timeline). Размещение сценариев на временной диаграмме, возможно, стоило убрать в первую очередь, однако оно оказалось весьма удобным. В ActionScript 2.0 вы могли поместить сценарий в класс и вызвать его из сценария на временной диаграмме; поэтому временные сценарии в действительности использовались только для вызова сценариев классов, размещенных в файлах ActionScript (*.as*). Для таких случаев в свойствах *.fla*-файлов Flash CS3 имеется поле, где вы можете добавить имя вызываемого класса. (Смотрите следующий раздел.) Таким образом, если вы хотите лишь вызвать программу и скомпилировать ее в *swf*-файл, то вам вовсе не требуется использовать в своем коде временную диаграмму.

Flash CS3 не отказался от кода на временной диаграмме. Вы все еще можете им пользоваться, но в данной книге мы будем применять его ограниченно, только для клипов, вызываемых из класса, находящегося вне классов клипов или кнопок. (Смотрите раздел «Классы кнопок и клипов».)

Класс документа

Вам не придется (если вообще придется) писать много кода для временной диаграммы при использовании версии ActionScript 3.0. Теперь не обязательно применять объекты из временной диаграммы; вы можете компилировать ваши *.as*-файлы, просто вводя имя класса, с помощью которого вы хотите запустить ваше приложение. Рисунок 1.1 показывает, как использовать поле класс документа (Document class) из меню Свойства документа (Properties), чтобы ввести имя класса для запуска.

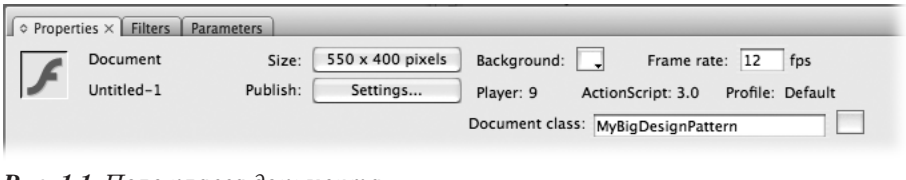


Рис. 1.1. *Поле класса документа*

Вы по-прежнему можете использовать временную диаграмму, однако если у вас нет на это каких-то особых причин, в этом нет необходимости. Большинство примеров в данной книге используют объекты класса Sprite вместо класса MovieClip. У объекта класса Sprite нет своей временной диаграммы в отличие от объекта класса MovieClip, поэтому использование Sprite спасает от связанных с ней дополнительных забот.

Классы кнопок и клипов

Во Flash CS3 объекты MovieClip и Button, которые вы создаете с помощью окна диалога Символ (Symbol) и сохраняете на панели Library (Библиотека), могут быть использованы и в ActionScript 3.0. В отличие от ActionScript 2.0, где символы клипов и кнопок могли быть только ассоциированы с другим классом, во Flash CS3 *они сами могут быть классами*. Имя объекта, введенное в окне Name (Имя) при создании этого объекта, становится его именем класса. (В последних версиях ссылки на клипы делаются через *имя экземпляра (instance name)*; то есть вы все еще можете их создавать, но уже в другом контексте.)

Преимуществом такого нового подхода является то, что теперь любые объекты-символы могут быть представлены, как и любой другой класс, в коде, если эти символы есть в Библиотеке. Не обязательно сразу размещать их на сцене. Они могут быть динамически созданы и выведены на дисплей, как любой другой динамический объект. Более того, объекты, содержащие клипы или кнопки, могут рассматриваться как свойства наравне со всеми прочими классами.

Хотя данная книга вовсе не является введением во Flash CS3, рассмотрение одного примера нового способа создания класса с клипами и кнопками может быть полезно как для новичков, так и для опытных пользователей Flash. Итак, вы можете опробовать новую возможность, выполнив следующие шаги:

1. Откройте новый документ Flash и сохраните его как *rocket.fla*.
2. Выберите из меню Insert → New Symbol (Вставить → Новый символ) и откройте диалоговое окно создания нового символа. В окне Name (Имя) введите Rocket и нажмите на OK, чтобы перейти в режим редактирования символа.
3. В режиме редактирования символа нарисуйте на сцене ракету с размерами W=89 (ширина), H=14 (высота), как показано на рис. 1.2. По

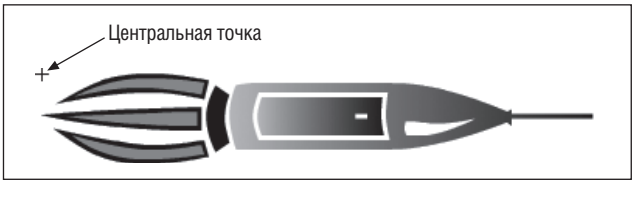


Рис. 1.2. *Рисунок ракеты*

завершении рисования поместите ее изображение в позицию X=0, Y=0. Щелкните на иконке Сцены 1, чтобы выйти из режима редактирования символа.

4. Выберите из меню Insert → New Symbol (Вставить → Новый символ), чтобы открыть диалоговое окно конвертации в символ (Convert to Symbol). Введите FireRocket в поле Name (Имя), выберете клип (Movie clip) в качестве типа и поставьте галочку напротив Export for ActionScript (Экспорт для ActionScript). После этого ваше диалоговое окно станет подобным изображению на рис. 1.3. Заметьте, базовым классом является flash.display.MovieClip. Базовый класс представляет собой ссылку на пакет, необходимый ActionScript для отображения объекта MovieClip. Нажмите OK, чтобы перейти в режим редактирования символа.

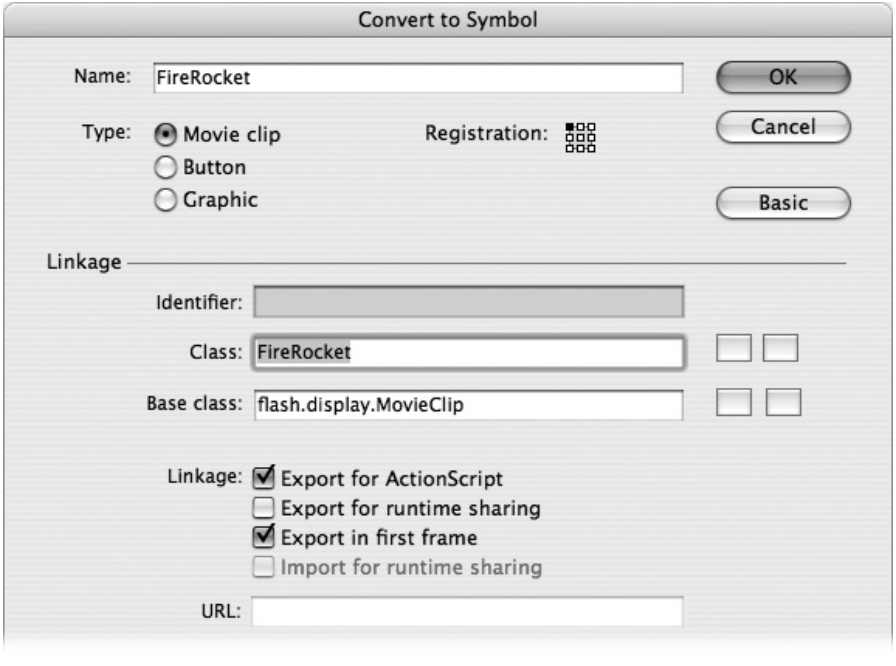


Рис. 1.3. *Установки для класса MovieClip*

5. Перетащите копию клипа ракеты из Библиотеки на центр сцены. Сдвиньте центральную точку клипа к хвосту ракеты и установите ее на позиции X=0, Y=0.
6. Кликните на 40-й кадр временной диаграммы и нажмите F5, чтобы создать на ней 40 кадров. Опять кликните на 40-м кадре и нажмите F6, чтобы создать ключевой кадр. Кликните на ключевом кадре 40-го кадра и перетащите ракету на позицию X=400, Y=0.
7. Кликните на первом ключевом кадре и в выпадающем окне инспектора свойств выберите Motion (Движение). Вы должны увидеть синюю

стрелку на временной диаграмме. Подвигайте указатель кадра слева направо и убедитесь, что движение ракеты реализовано должным образом. Рисунок 1.4 показывает то, что вы должны увидеть.

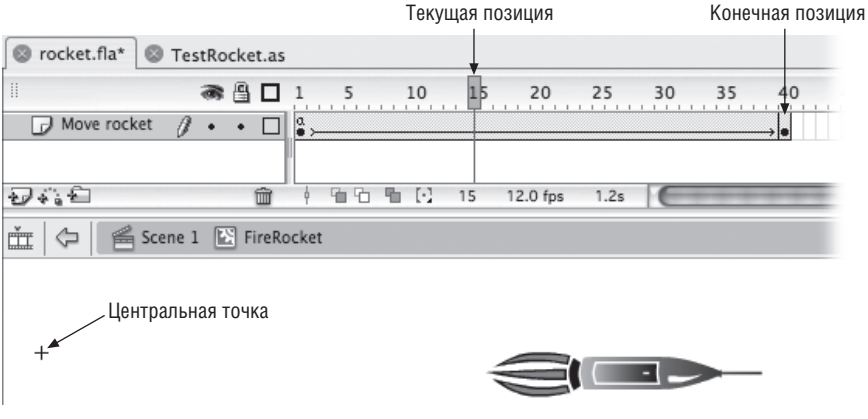


Рис. 1.4. *Ракета в движении*

8. Откройте панель Actions (Действия). Кликните на пустой области сцены, чтобы у вас точно не было выбранного объекта, и затем кликните на первом кадре. В панели действий введите выражение `stop()`. Сохраните файл *Rocket.fla*.
9. Откройте новый файл ActionScript и сохраните его под именем *TestRocket.as* в том же каталоге, где содержится файл *Rocket.fla*. Введите скрипт из примера 1.1 в файл *TestRocket.as* и сохраните файл еще раз.

Пример 1.1. TestRocket.as

```
package
{
    import flash.display.Sprite;
    public class TestRocket extends Sprite
    {
        private var fireRocket:FireRocket;
        public function TestRocket( )
        {
            fireRocket=new FireRocket( );
            fireRocket.x=50;
            fireRocket.y=100;
            addChild(fireRocket);
            fireRocket.gotoAndPlay(2);
        }
    }
}
```

10. И, наконец, откройте файл *Rocket.fla*, в поле класса документа на панели свойств наберите `TestRocket` и сохраните файл. Затем протестируйте ваш минифильм, нажав `Ctrl+Enter` (`Command+Return` на Макинтошах). Вы должны увидеть ракету, летящую слева направо по экрану и потом возвращающуюся на исходную позицию.

Более или менее традиционное использование Flash для создания клипов все еще остается важной областью применения ActionScript, но кое-что изменилось. Вы больше не можете присоединять к клипу класс, как это делалось в предыдущих версиях. Однако при создании приложений, использующих шаблоны проектирования, вы по-прежнему можете интегрировать в них различные объекты, созданные в среде проектирования (IDE) Flash. Так что, хотя ActionScript и сделал прыжок к ECMAScript, он не оторвался от своих корней в анимационной графике.

Основы ООП

Если вы хорошо знакомы с ООП и успешно применяете его на практике, вам, вероятно, можно пропустить этот раздел или только бегло просмотреть его на случай, не добавили ли мы что-то новое об ООП или нет ли в нем нового материала, касающегося языка ActionScript. Далее в этой главе мы подробно рассмотрим хорошо зарекомендовавшие себя концепции ООП, на которых базируются все шаблоны проектирования. Эти концепции рассчитаны на хорошее понимание основ ООП. К сожалению, наше краткое их обсуждение не сможет обеспечить вам достаточной глубины их понимания. Если это ваше первое знакомство с ООП, то вам, несомненно, потребуется пополнить ваши знания по данной теме с помощью хороших учебников, посвященных ООП.

На протяжении всей книги вы будете сталкиваться с примерами того, как шаблоны проектирования используют различные базовые принципы ООП. В каждой ее главе содержится раздел, посвященный некоторым ключевым концепциям ООП в шаблонах; в данной вводной главе вы встретитесь лишь с первым из множества случаев их описания. Это сделано нами специально. Мы полагаем, что рассмотрение концепций ООП с различных точек зрения поможет вам лучше разобраться во многих их нюансах. Для нас самих было удивительно, что разные шаблоны проектирования настолько по-разному освещают одни и те же концепции ООП и помогают глубже понять их смысл.

Для начала мы вспомним четыре базовые концепции ООП:

- Абстрактность
- Инкапсуляция
- Наследование
- Полиморфизм

Каждая из них нуждается в осмыслении; если вы новичок в ООП, не мучайтесь, стараясь понять их с первого раза. Мы будем еще много раз возвращаться к этим концепциям в главах, посвященных конкретным шаблонам проектирования.

Абстрактность

В общем абстракция – это модель или образец. Вы не знаете все ее детали, а только некоторые основные параметры, которые могут быть уточнены и дополнены. При этом абстракция достаточно точна, чтобы отличаться от других. Рассмотрим для примера две свободные вакансии в вашей компании: веб-дизайнера и программиста. В объявлении о них вы скорее укажете основные характеристики кандидатов, требуемые для выполнения соответствующей работы, чем какие-то индивидуальные их особенности. Тогда у вас получатся две абстракции, представляющие две различные позиции:

Открыты две вакансии:

- Программист
  - Имеющий опыт работы в команде
  - Имеющий опыт программирования межплатформенного программного обеспечения (ПО) и баз данных
  - Знакомый с языком ECMAScript
  - Имеющий навыки объектно-ориентированного программирования и знание шаблонов проектирования
- Веб-дизайнер
  - Имеющий опыт создания графических сайтов
  - Знакомый с анимацией
  - Имеющий опыт работы с векторной графикой
  - Практикующий клиенто-ориентированный подход

Здесь легко заметить разницу между двумя позициями и основными предъявляемыми к ним требованиями (свойствами); но их детали остаются открытыми для заполнения. Веб-дизайнер вряд ли будет претендовать на работу программиста, равно как и программист – на позицию веб-дизайнера. Однако кандидаты на эти должности будут обладать множеством различных дополнительных свойств и навыков, которые и станут конкретными деталями для позиций. Например, один из программистов может иметь опыт работы с PHP и/или с базой данных MySQL, в то время как другой программист может знать ASP.NET, C# и MS SQL. Таким образом, абстракция дается в описании должности, а деталями служат навыки и опыт работы каждого из кандидатов.

В своей книге «Object-Oriented Design with Applications» (издательство Benjamin/Cummings) Гради Буч (Grady Booch), один из пионеров в об-

ласти шаблонов проектирования, дал следующее определение абстракции, одновременно краткое и понятное:

> Абстракция выделяет существенные характеристики некоторого объекта, отличающие его от всех других видов объектов, и, таким образом, четко определяет его концептуальные границы с точки зрения наблюдателя.

Под определение Буча очень хорошо подходят наши описания двух свободных позиций. Они содержат в себе важнейшие характеристики кандидатов на эти вакансии и позволяют четко отличить одну должность от другой.

Абстракции в ActionScript 3.0

Обращаясь теперь к абстракциям в программировании на ActionScript, в качестве примера мы рассмотрим простой видеопроигрыватель. Он будет состоять из некоторых необходимых нам элементов, поэтому мы начнем с их перечисления как абстракций:

- Сетевое соединение
- Видеоэкран
- Поток видео
- *flv*-файл для воспроизведения

Если нам удастся правильно собрать вместе все эти части, мы сможем проигрывать видео. Однако мы хотим начать нашу деятельность не с абстракций, а с чего-то конкретного, работающего уже сейчас. Введите код из примера 1.2 и сохраните его в файле с именем из заголовка примера:

Всюду в книге, за исключением нескольких случаев, имена из названий примеров используются как имена файлов для сохранения их кода.

Пример 1.2. PlayVideo.as

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    import flash.media.Video;
    import flash.display.Sprite;

    public class PlayVideo extends Sprite
    {
        public function PlayVideo()
        {
            var nc:NetConnection=new NetConnection( );
            nc.connect(null);
            var ns:NetStream = new NetStream(nc);
```

```
            var vid:Video=new Video( );
            vid.attachNetStream(ns);
            ns.play("adp.flv");
            addChild(vid);
            vid.x=100;
            vid.y=50;
        }
    }
}
```

Вам потребуется *flv*-файл с именем *adp.flv*; любой *flv*-файл с таким именем будет работать. Откройте новый Flash файл, введите `PlayVideo` в поле класс документа и протестируйте его.

Чтобы сделать наш код абстрактным, нужно убрать из него все конкретные значения, за исключением `null` из метода `NetConnection.connect()`. (Его можно было бы передавать как строку, но для простоты мы его оставляем.) В примере 1.3 приводится то же приложение, только абстрагированное до «описания» того, что ему требуется для работы.

Пример 1.3. PlayVideoAbstract.as

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    import flash.media.Video;
    import flash.display.Sprite;

    public class PlayVideoAbstract extends Sprite
    {
    public function PlayVideoAbstract(nc:NetConnection,
        ns:NetStream,vid:Video,flick:String,xpos:uint,ypos:uint)
        {
            nc=new NetConnection();
            nc.connect(null);
            ns= new NetStream(nc);
            vid=new Video();
            vid.attachNetStream(ns);
            ns.play(flick);
            vid.x=xpos;
            vid.y=ypos;
            addChild(vid);
        }
    }
}
```

Здесь все конкретные значения (за исключением `null`) заменены их абстракциями для *описания* работы данного объекта. Но, как и требования в описании вакансии, эти абстракции в классе `PlayVideoAbstract` необходимо заполнить чем-то конкретным. Все частности помещены в один длинный набор параметров:

```
    PlayVideoAbstract(nc:NetConnection,ns:NetStream,
        vid:Video,flick:String,xpos:uint,ypos:uint)
```

Эти абстрактные параметры в конструкторе класса позволяют нам добавлять в него любой конкретный, нужный нам элемент, включая имя файла видео для проигрывания. Пример 1.4 показывает, как конкретные элементы используются в абстрактном классе.

Пример 1.4. PlayAbstract.as.

```
package
{
    import flash.display.Sprite
    import flash.net.NetConnection;
    import flash.net.NetStream;
    import flash.media.Video;

    public class PlayAbstract extends Sprite
    {
        private var conn:NetConnection;
        private var stream:NetStream;
        private var vid:Video;
        private var flick:String="adp.flv";

        public function PlayAbstract( )
        {
            var playIt:PlayVideoAbstract=new PlayVideoAbstract(conn,stream,vid,
flick,100,50);
            addChild(playIt);
        }
    }
}
```

Все, что делает данный класс, – это создает один объект класса `PlayVideoAbstract` и размещает его на сцене. Закрытые переменные служат для передачи большинства конкретных значений необходимых параметров. Константы указывают горизонтальную (`x`) и вертикальную (`y`) позиции для проигрывания видео. Чтобы протестировать данный пример, просто измените во Flash для *fla*-файла имя в поле класса документа на `PlayAbstract`.

Почему абстракции так важны

Можно выделить две основные причины, по которым абстракции так важны и для ООП, и для шаблонов проектирования. Во-первых, абстракции позволяют нам сосредоточиться на тех независящих от своих деталей частях, которые требуются для решения наших задач, а не заниматься мелочами. Означает ли это, что детали игнорируются? Вовсе нет. Детали добавляются, когда они действительно нужны. Например, в примере в предыдущем разделе конкретный видеофайл нам не важен. Важно лишь, что какое-то имя видеофайла (конкретная деталь) будет

нам передано, когда мы будем готовы проиграть это видео. Так же как нам не нужно строить целый кинотеатр ради просмотра одного фильма, нам не нужно создавать класс ради просмотра одного видеофайла.

Во-вторых, абстракции дают нам большую гибкость в наших решениях. Если вам показалось, что реализация нашего приложения из примера 1.2 предыдущего раздела была проще и потребовала меньше кода и классов, вы абсолютно правы. Представьте теперь, что вам требуется разместить на сцене сразу четыре видео. Тогда вместо того, чтобы переписывать свой класс еще три раза, вам нужно будет лишь создать четыре объекта этого абстрактного класса. Другими словами, вторая реализация, использующая абстракции, гораздо более гибкая к изменениям. Мы можем не только создавать несколько видеообъектов, но и легко изменять в них имена видеофайлов для воспроизведения.

Инкапсуляция

Инкапсуляция – то, что делает код объекта целостным объектом. Если у вас имеются в наличии хвост, четыре ноги, холодный нос и звук лая, это еще не означает, что у вас есть собака. У вас есть только набор исходных частей, из которых она состоит. Если вы соберете все их вместе и получите собаку, то хотя вы и будете помнить, что все ее части являются частями, вы станете воспринимать собаку как уникальное целое, не думая больше о частях. То есть объектом будет уже собака, а не ее части, сложенные вместе. Инкапсуляция производит схожий эффект на наборы операций и свойств.

Понятие инкапсуляции часто упоминается вместе с такими понятиями, как *компонент* и *модуль*. В контексте ООП инкапсуляцию очень часто называют *черным ящиком*, в том смысле, что вы можете наблюдать внешние проявления объекта, но не можете видеть, как все устроено и работает у него внутри. На самом деле многие вещи, с которыми мы встречаемся в нашей жизни, являются для нас черными ящиками, например, собака. Мы можем смотреть, как собака делает множество разнообразных вещей, мы можем контактировать с ней, но мы толком не знаем (да нам это обычно и не важно), как при этом работает ее организм, поскольку она не прозрачная. Собака – черный ящик.

Хорошей стороной концепции черного ящика является то, что нам не приходится заботиться о его внутренней работе или устройстве. Нам необходимо лишь уметь правильно с ним обращаться, будучи подкрепленными знанием, что пока он действует так, как нам нужно, внутри у него все в порядке.

Сокрытие внутренних данных «во избежание»

Чтобы понять, почему нам важно инкапсулировать свои данные, мы рассмотрим две программы. Одна будет без инкапсуляции, что приве-

дет к нежелательным последствиям, а вторая – с инкапсуляцией, что поможет нам избежать странных результатов ее работы.

Если вы создаете объект Собака, то вы, наверное, захотите включить в него операции, дающие собаке возможность как-то общаться с внешним миром. Для иллюстрации мы создадим метод под названием `dogTalk` («собачий язык»), который позволит собаке издавать некоторые звуки. Собака будет у нас общаться следующими способами:

- Гавкать
- Скулить
- Выть
- Рычать

Мы начнем с *плохого* применения ООП, чтобы показать вам, как в словаре собаки могут образовываться нежелательные выражения. Код примера 1.5 является неинкапсулированным и в будущем способен поставить вашу собаку в неудобное положение.

Пример 1.5. NoEncap.as

```
package
{
    // Это плохое ООП – без инкапсуляции
    import flash.text.TextField;
    import flash.display.Sprite;

    public class NoEncap extends Sprite
    {
        public var dogTalk:String="Woof, woof!";
        public var textFld:TextField=new TextField( );

        public function NoEncap( )
        {
            addChild(textFld);
            textFld.x=100;
            textFld.y=100;
        }
        function showDogTalk( )
        {
            textFld.text=dogTalk;
        }
    }
}
```

В классе, имеющем свойства черного ящика, у вас *не должно быть* возможности менять его внутреннюю работу. Но этот класс, как вы увидите, открыт для таких изменений. С инкапсулированным объектом вы можете взаимодействовать только через его интерфейс, и вы не долж-

ны позволять приложению вносить внутрь него какие-либо изменения. Код в примере 1.6 вламывается внутрь нашего объекта и меняет его нежелательным образом.

Пример 1.6. TestNoEncap.as

```
package
{
    import flash.display.Sprite;

    public class TestNoEncap extends Sprite
    {
        public var noEncap:NoEncap;
        public function TestNoEncap()
        {
            noEncap=new NoEncap();
            noEncap.dogTalk="Meow";
            noEncap.showDogTalk();
            addChild(noEncap);
        }
    }
}
```

Откройте новый Flash-файл и в поле класс документа введите `TestNoEncap`. Когда вы запустите этот файл, вы увидите на экране «Meow» («Мяу»). Такой ответ от вашего объекта-собаки никуда не годится. Собаки не мяукают, а кошки не лают. Тем не менее подобное вполне может произойти, если вы не инкапсулировали свой класс. А если вы умножите эту проблему на количество всех неинкапсулированных классов в вашем приложении, то вы можете представить себе, какой ужасный беспорядок в результате у вас получится. Так что давайте-ка исправлять ситуацию.

Приватные переменные

Простейшим способом обеспечить инкапсуляцию ваших данных является использование приватных (или «закрытых») переменных. Ключевое слово `private` языка ActionScript 3.0, будучи примененным к переменным, константам или методам (функциям), гарантирует, что только класс, в котором они были определены или объявлены, будет иметь к ним доступ. Оно не только закрывает от них наши приложения, пытающиеся присвоить им какие-либо значения, но и исключает подклассы. (В этом состоит отличие от ActionScript 2.0; поэтому будьте внимательны при преобразовании приложения с ActionScript 2.0 на ActionScript 3.0.)

Давайте посмотрим, как слово `private` изменит работу нашего приложения. В примере 1.7 к переменным класса `NoEncap` были добавлены атрибуты `private`.

Пример 1.7. Encap.as

```
package
{
    // Это хорошее ООП – инкапсуляция присутствует
    import flash.text.TextField;
    import flash.display.Sprite;

    public class Encap extends Sprite
    {
        private var dogTalk:String="Woof, woof!";
        private var textFld:TextField=new TextField();

        public function Encap()
        {
            addChild(textFld);
            textFld.x=100;
            textFld.y=100;
        }
        function showDogTalk()
        {
            textFld.text=dogTalk;
        }
    }
}
```

Небольшие изменения нужно внести и в тестовый файл. Его реализация суперкласса должна быть изменена. В примере 1.8 показан новый тестовый класс, `TestEncap`, для класса `Encap`.

Пример 1.8. TestEncap.as

```
package
{
    import flash.display.Sprite;

    public class TestEncap extends Sprite
    {
        public var encap:Encap
        public function TestEncap()
        {
            encap=new Encap();
            encap.dogTalk="Meow";
            encap.showDogTalk();
            addChild(encap);
        }
    }
}
```

Попробуйте его запустить, изменив имя класса документа на `TestEncap`. Вы получите сообщение об ошибке от компилятора: