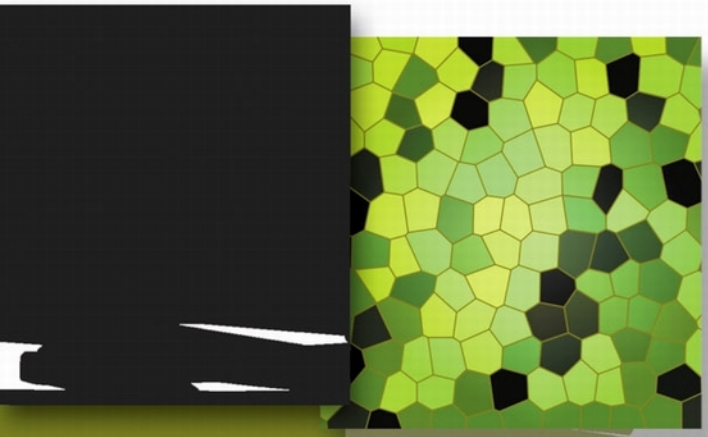
ТИМУР МАШНИН

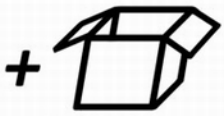
bhv®

JavaFX 2.0
Разработка RIA-приложений



- Новые GUI-компоненты с поддержкой CSS
- Визуальные эффекты, трансформации и анимации
- Воспроизведение аудио и видео
- Язык FXML для создания GUI-интерфейса

PRO
ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ



Материалы на www.bhv.ru

Тимур Машнин

JavaFX 2.0
Разработка RIA-приложений

Санкт-Петербург
«БХВ-Петербург»
2012

УДК 681.3.068
ББК 32.973.26-018.1
М38

Машнин Т. С.

М38 JavaFX 2.0: разработка RIA-приложений. — СПб.: БХВ-Петербург, 2012. — 320 с.: ил. — (Профессиональное программирование)

ISBN 978-5-9775-0820-9

Книга посвящена разработке RIA-приложений (Rich Internet Applications) с использованием технологии JavaFX 2.0. Рассмотрены архитектура платформы JavaFX 2.0, ее основные компоненты графического интерфейса пользователя, применение CSS-стилей, создание визуальных эффектов, трансформация и анимация изображений, совместное использование JavaScript и JavaFX, Swing и JavaFX, выполнение фоновых задач, использование компонентов JavaFX Beans и связывание данных, язык FXML и др. Приведен справочник программного интерфейса JavaFX 2.0 API. Материал книги сопровождается большим количеством примеров с подробным анализом исходных кодов. На сайте издательства находятся проекты примеров из книги, а также дополнительные материалы.

Для программистов

УДК 681.3.068
ББК 32.973.26-018.1

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	*Марины Дамбиевой*
Зав. производством	*Николай Тверских*

Подписано в печать 31.01.12.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 25,8.
Тираж 1200 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

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

ISBN 978-5-9775-0820-9

Оглавление

Введение **7**

Глава 1. Архитектура платформы JavaFX 2.0 ..... **11**
Программный интерфейс JavaFX API 15
Модель программирования приложений платформы JavaFX 2.0 16
Развертывание JavaFX-приложений 18

Глава 2. Компоненты графического интерфейса пользователя **23**
Кнопка *Button* 24
Флажок *CheckBox* 29
Гиперссылка *Hyperlink* 34
Кнопка *MenuButton* 38
Кнопка *SplitMenuButton* 42
Кнопка *ToggleButton* 46
Переключатель *RadioButton* 51
Метка *Label* 54
Список *ListView* 57
Таблица *TableView* 63
Список *ChoiceBox* 67
Панель *MenuBar* и меню *Menu* 71
Дерево *TreeView* 76
Меню *ContextMenu* 80
Окно подсказки *Tooltip* 82
Всплывающее окно *Popup* 84
Окно выбора файлов *FileChooser* 87
Многострочное поле *TextArea* 89
Поле ввода *TextField* 92
Поле ввода пароля *PasswordField* 95
Панель *ScrollPane* 98
Панель с вкладками *TabPane* 101
Панель *TitledPane* 104
Панель *Accordion* 106
Индикаторы *ProgressBar* и *ProgressIndicator* 109
Разделитель *Separator* 112

Ползунок *Slider* ... 115
Панель компоновки *AnchorPane* ... 118
Панель *BorderPane* ... 122
Панель *FlowPane* ... 124
Панель *GridPane* ... 127
Панели *VBox* и *HBox* ... 129
Панель *StackPane* ... 131
Панель *TilePane* ... 134
Панель *SplitPane* ... 137
Панель *ToolBar* ... 140
Узел изображения *ImageView* ... 143
Сцена *Scene* ... 144
Группа *Group* ... 147
Окно *Stage* ... 148
2D-графика ... 149
Дуга *Arc* ... 149
Линия *Line* ... 150
Круг *Circle* ... 151
Кубическая кривая Безье *CubicCurve* ... 151
Квадратичная кривая Безье *QuadCurve* ... 152
Эллипс *Ellipse* ... 153
Прямоугольник *Rectangle* ... 153
Ломаная линия *Polyline* ... 154
Многоугольник *Polygon* ... 154
Фигура *Path* ... 155
Фигура *SVGPath* ... 157
Узел *Text* ... 157
Диаграммы ... 158
Круговая диаграмма *PieChart* ... 162
Диаграмма *AreaChart* ... 164
Диаграмма *BarChart* ... 167
Диаграмма *BubbleChart* ... 169
Диаграмма *LineChart* ... 172
Диаграмма *ScatterChart* ... 174
Отображение Web-контента ... 176
Редактор *HTMLEditor* ... 181
Воспроизведение аудио и видео ... 184
Проигрыватель *AudioClip* ... 189

Глава 3. JavaFX CSS ... 193

Глава 4. Визуальные эффекты ... 221
Эффект смешивания *Blend* ... 221
Эффект свечения *Bloom* ... 222
Эффект свечения *Glow* ... 223
Эффект тени *DropShadow* ... 224
Эффект тени *Shadow* ... 226
Эффект тени *InnerShadow* ... 227
Эффект размытия *BoxBlur* ... 229

Эффект размытия *MotionBlur* ... 230
Эффект размытия *GaussianBlur* ... 231
Эффект *ColorAdjust* ... 232
Эффект *DisplacementMap* ... 233
Эффект *Lighting* ... 235
Эффект перспективы *PerspectiveTransform* ... 240
Эффект отражения *Reflection* ... 242
Эффект *SepiaTone* ... 244

Глава 5. Трансформация и анимация ... 247

Глава 6. События ... 261

Глава 7. Совместное использование JavaScript и JavaFX ... 267
Вызов JavaFX-апплета из JavaScript-кода ... 267
Вызов JavaScript-кода из JavaFX-апплета ... 271
Использование JavaScript в *WebView* ... 273

Глава 8. Выполнение фоновых задач ... 277

Глава 9. Совместное использование Swing и JavaFX ... 281

Глава 10. Компоненты JavaFX Beans и связывание данных ... 291

Глава 11. Заставка запуска JavaFX-приложения ... 297

Глава 12. Язык FXML ... 305

Приложение. Описание электронного архива ... 311

Введение

Развитие Всемирной паутины привело в 2004 году к рождению архитектуры Web 2.0 — набору рекомендаций и решений для создания и поддержки Web-ресурсов, при реализации которых распределенные системы Web 2.0 становятся тем более насыщенными контентом, чем больше пользователей ими пользуются, причем качество такой распределенной системы зависит от уровня взаимодействия и активности пользователей, в отличие от интернет-систем архитектуры Web 1.0, где за качество поставляемого контента целиком и полностью отвечает владелец Web-ресурса.

Web-приложения архитектуры Web 2.0 ориентированы на совместное использование информации, а также взаимодействие и сотрудничество участников Всемирной паутины. Примером систем Web 2.0 могут служить социальные сети, блоги, файловые и видеохостинги и т. д.

При миграции клиент-серверных систем от Web 1.0 к Web 2.0 клиентское приложение становится не просто "тонким" клиентом, отображающим формируемое сервером статическое содержимое в виде HTML-страниц, а насыщенным интернет-приложением RIA (Rich Internet Application) — Web-приложением, предоставляющим большую интерактивность для клиента.

Большая, по сравнению с традиционным "тонким" клиентом, интерактивность RIA-приложения обеспечивается за счет богатого графического GUI-интерфейса пользователя, содержащего, помимо разнообразных компонентов контроля, анимацию, векторную графику и аудио/видеоролики, при этом приемлемая скорость работы такого приложения достигается с помощью переноса части выполняемого кода с сервера на сторону клиента. Таким образом, RIA-приложение является промежуточным между "тонким" клиентом и "толстым" клиентом — настольным приложением.

Традиционный "тонкий" клиент использует для взаимодействия с сервером HTML-разметку и простой JavaScript-код и не требует загрузки и инсталляции дополнительного программного обеспечения. В отличие от "тонкого" клиента RIA-приложение создается на базе определенной платформы, предоставляющей язык программирования и набор библиотек программного интерфейса, и поэтому его работа на стороне клиента требует наличия среды выполнения соответствующей

платформы, которую необходимо предварительно загрузить и инсталлировать. Такой средой выполнения, как правило, служит плагин Web-браузера.

Примечание

RIA-приложения можно также создавать на основе языка разметки HTML5 и JavaScript-библиотек, таких как jQuery, Yahoo!, MochaUI и др.

Преимуществом RIA-приложения перед настольным приложением является отсутствие необходимости его специальной установки на клиентском компьютере — RIA-приложение предоставляется определенной областью на страничке Web-браузера, при этом RIA-приложение имеет автоматическое обновление версий и кроссплатформенность, т. к. его оберткой служит Web-браузер. Кроме того, работа RIA-приложения автоматически защищена "песочницей" Web-браузера.

Наиболее популярными технологиями для создания RIA-приложений на сегодняшний день являются платформы Adobe Flash, JavaFX и Microsoft Silverlight.

Все вышеупомянутые RIA-платформы позволяют разрабатывать не только Web-приложения, работающие в Web-браузере, но и настольные приложения, которые способны работать в режиме оффлайн. Для платформы Adobe Flash — это AIR-приложения или SWF-файлы, запускаемые автономно плеером Flash Player, для платформы Microsoft Silverlight — это Web-приложения, помещенные в специальный каталог со ссылками с рабочего стола или из меню **Пуск**.

Что касается технологии JavaFX, то один и тот же Java-код, созданный на базе платформы JavaFX, может запускаться как настольное приложение, которое разворачивается на клиентском компьютере автономно, или разворачиваться как Java Web Start-приложение, или отображаться в Web-браузере как JavaFX-апплет, встроенный в HTML-страничку.

Платформа JavaFX может использоваться совместно с технологией Swing, а также с другими языками — JRuby, Groovy и JavaScript для создания больших и комплексных приложений с насыщенным графическим интерфейсом пользователя.

Технология JavaFX обеспечивает создание мощного графического интерфейса пользователя (Graphical User Interface, GUI) для крупномасштабных приложений, ориентированных на обработку данных, насыщенных медиаприложений, поставляющих разнообразный медиаконтент пользователю, Mashup-приложений, объединяющих различные Web-ресурсы для пользователя, компонентов высококачественной графики и анимации для Web-сайтов, различного рода пользовательских программ с графикой, анимацией и интерактивными элементами.

Технологии создания RIA-приложений платформы Java берут свое начало от Java-апплетов, GUI-интерфейсы которых использовали графические системы AWT и Swing для организации взаимодействия с пользователем и отображения ему данных, текста, графики и анимации.

Графическая библиотека AWT была самой первой графической Java-системой набора JDK 1.0, дополненной затем библиотекой Java 2D двумерной графики и изображений. Библиотека AWT предоставляет разработчику возможность использования таких основных компонентов GUI-интерфейса, как кнопки, переключатели,

списки, метки, окна выбора файла, меню, компоненты визуализации и редактирования текста, функции drag and drop, возможность обработки событий UI-компонентов, компоновки компонентов в рабочей области, работы с цветом, шрифтом, графикой, рисования и печати. Библиотека AWT является тяжеловесной, т. к. она содержит собственную (родную, native) библиотеку java.awt.peer, через которую взаимодействует с операционной системой компьютера, поэтому отображение AWT GUI-интерфейса зависит от операционной системы, в которой приложение развернуто.

Ограниченность набора GUI-компонентов библиотеки AWT и ее тяжеловесность послужили причиной создания графической системы Swing, которая основывается на библиотеке AWT и поэтому является уже легковесной. Кроме того, библиотека Swing дополняет библиотеку AWT такими компонентами GUI-интерфейса, как панель выбора цвета, индикатор состояния, переключатель, слайдер и спиннер, панель с вкладками, таблицы и деревья, расширенными возможностями компоновки GUI-компонентов, таймером, возможностью изменения внешнего вида LookAndFeel GUI-интерфейса, отображения HTML-контента. Библиотека Swing реализует архитектуру MVC (Model-View-Controller) и потоковую модель Event Dispatch Thread (EDT).

Несмотря на богатые возможности графических систем AWT и Swing, они не удовлетворяют современным требованиям работы с медиаконтентом, что и послужило причиной создания платформы JavaFX, которая предоставляет современные GUI-компоненты, богатый набор библиотек графического и медиапрограммного API-интерфейса, а также высокопроизводительную среду выполнения приложений.

Первоначально, в 2007—2010 годах, версии 1.1, 1.2 и 1.3 платформы JavaFX содержали:

- декларативный язык программирования JavaFX Script создания UI-интерфейса;
- набор JavaFX SDK, обеспечивающий компилятор и среду выполнения;
- плагины для сред выполнения NetBeans IDE и Eclipse;
- плагины для Adobe Photoshop и Adobe Illustrator, позволяющие экспортировать графику в код JavaFX Script, инструменты конвертации графического формата SVG в код JavaFX Script.

Платформа JavaFX версии 2.0 выпуска 2011 года кардинально отличается от платформы JavaFX версии 1.*x*.

Платформа JavaFX 2.0 больше не поддерживает язык JavaFX Script, а вместо этого предлагает новый программный интерфейс JavaFX API для создания JavaFX-приложений полностью на языке Java. Для альтернативного декларативного описания графического интерфейса пользователя платформа JavaFX 2.0 предлагает новый язык FXML. Кроме того, платформа JavaFX 2.0 обеспечивает новые графический и медийный движки, улучшающие воспроизведение графического и мультимедийного контента, встраивание HTML-контента в приложение, новый плагин для Web-браузеров, широкий выбор UI-компонентов с поддержкой CSS3. При этом платформа JavaFX версии 2.0 содержит:

- набор JavaFX SDK, предоставляющий инструмент JavaFX Packager tool компиляции, упаковки и развертывания JavaFX-приложений, Ant-библиотеку для сборки JavaFX-приложений, библиотеки JavaFX API и документацию;
- среду выполнения JavaFX Runtime для работы настольных JavaFX-приложений и JavaFX-апплетов;
- поддержку платформы JavaFX 2.0 для среды выполнения NetBeans IDE 7;
- примеры JavaFX-приложений.

В дальнейшем платформа JavaFX будет интегрирована в платформу JDK 8 и не потребует отдельной инсталляции.

Сайт новой платформы JavaFX 2.0 находится по адресу: **http://javafx.com/**.

В данной книге рассматривается новая технология JavaFX 2.0 создания RIA-приложений платформы Java с насыщенным графическим GUI-интерфейсом и мощными возможностями по воспроизведению графического и мультимедийного контента, обработки данных и совмещению с другими Java-технологиями.

ГЛАВА 1

Архитектура платформы JavaFX 2.0

Платформа JavaFX версии 2.0 состоит из программного интерфейса JavaFX API, альтернативного декларативного языка FXML описания GUI-интерфейса, среды выполнения JavaFX Runtime, набора разработчика JavaFX SDK и поддержки платформы JavaFX 2.0 для среды выполнения NetBeans IDE 7.

Работу настольного JavaFX-приложения или JavaFX-апплета на стороне клиента обеспечивает реализация платформы JavaFX — среда выполнения JavaFX Runtime. Среда выполнения JavaFX Runtime обеспечивает работу JavaFX-кода в виртуальной машине JVM, поэтому для запуска JavaFX-приложения требуется установка набора Java Development Kit (JDK) (**http://www.oracle.com/technetwork/java/javase/downloads/index.html**) и среды выполнения JavaFX Runtime.

Среда выполнения JavaFX Runtime устанавливается с помощью инсталлятора javafx-2_0-windows-i586.exe, доступного для скачивания по адресу **http://www.oracle.com/technetwork/java/javafx/downloads/index.html**, который создает в файловой системе пользователя каталог C:\Program Files\Oracle\JavaFX Runtime 2.0, содержащий папки bin и lib.

Папка bin среды выполнения JavaFX Runtime содержит DLL-библиотеки, обеспечивающие работу JavaFX-кода, включая DLL-библиотеки JavaFX-плагина Web-браузера, а также инструменты Java Web Start (JavaWS), Java Quick Starter (JQS), Java Control Panel, SSVAGENT загрузки и запуска JavaFX-приложений, контроля и обновления среды выполнения JavaFX Runtime.

Папка lib среды выполнения JavaFX Runtime содержит вспомогательные файлы развертывания Java Quick Starter (JQS), файл политики безопасности Java Web Start (JavaWS), а также JAR-библиотеки JQS-развертывания, JavaWS-загрузки, программного интерфейса JavaFX API и JavaFX-плагина Web-браузера.

В отличие от технологий Adobe Flash и Microsoft Silverlight, JavaFX-плагин Web-браузера не является надстройкой Web-браузера, а его работа определяется JavaScript-кодом, который встраивает JavaFX-код в качестве JavaFX-апплета в Web-страничку, подключая установленную на локальном компьютере среду выполнения JavaFX Runtime, содержащую JavaFX-плагин Web-браузера.

Разрабатывать JavaFX-приложения можно с помощью набора разработчика JavaFX SDK или среды разработки NetBeans IDE с поддержкой платформы JavaFX 2.0.

Инсталлятор набора разработчика JavaFX SDK доступен для скачивания по адресу **http://www.oracle.com/technetwork/java/javafx/downloads/index.html**. Набор JavaFX SDK содержит библиотеки платформы JavaFX, документацию, а также инструмент JavaFX Packager tool компиляции, упаковки и развертывания JavaFX-приложений и Ant-библиотеку для использования в Ant-сценарии сборки JavaFX-приложений.

Среда разработки NetBeans IDE с поддержкой платформы JavaFX 2.0 доступна для скачивания по адресу **http://www.oracle.com/technetwork/java/javafx/downloads/index.html**. Для подключения Java-платформы с поддержкой JavaFX 2.0 необходимо открыть среду NetBeans и в меню **Сервис** выбрать пункт **Платформы Java**. В появившемся диалоговом окне нажать кнопку **Добавить платформу** и выбрать папку установленного набора JDK. После нажатия кнопок **Далее** и **Готово** нужно открыть вкладку **JavaFX** и, установив флажок **Enable JavaFX**, в полях **JavaFX SDK** и **JavaFX Runtime** кнопкой **Browse** выбрать соответствующие папки.

Теперь при выборе в меню среды NetBeans **Создать проект | JavaFX** появятся шаблоны проектов **JavaFX Application**, **JavaFX Preloader** и **JavaFX FXML Application**.

Для работы как с набором JavaFX SDK, так и со средой NetBeans IDE требуются инсталлированные набор JDK и среда JavaFX Runtime.

Примеры JavaFX-приложений содержат скомпилированный и исходный Java-код и доступны для скачивания по адресу **http://www.oracle.com/technetwork/java/javafx/downloads/index.html**.

Так как платформа JavaFX обеспечивает графические и анимационные возможности для приложений, технология JavaFX оперирует терминологией и понятиями компьютерной графики.

- *Компьютерная графика* — создание и отображение данных изображения компьютером с помощью программного обеспечения и компьютерного оборудования. Существуют различные виды компьютерной графики — 2D-графика (создание цифровых изображений из двумерных геометрических моделей), растровая графика (представление цифрового изображения в виде сетки пикселов), векторная графика (представление цифрового изображения в виде математических формул, описывающих изображение как набор геометрических примитивов), 3D-графика (создание цифровых изображений из трехмерного представления геометрических данных).
- *Рендеринг* (rendering) — это процесс генерации растрового изображения из *модели* (model) или *сцены* (scene) с сопутствующими эффектами. Одним из быстрых методов рендеринга является растеризация (rasterisation) — геометрическое проецирование моделей на плоскость изображения с генерацией растрового изображения. Рендеринг, основанный на растеризации, выполняется по графическому конвейеру (graphics pipeline). Графический конвейер представляет стадии процесса рендеринга, основанного на растеризации, и может быть разделен на три этапа — тесселяция, геометрическая обработка и растеризация. Тесселя-

ция (tessellation) — процесс разбиения поверхности модели на *полигоны*, производится программным обеспечением без аппаратного ускорения. Геометрическая обработка — трансформация, отсечение, освещение, проецирование и предварительная обработка, производится, как правило, программным обеспечением до фазы предварительной обработки, которая частично или полностью осуществляется на аппаратном уровне. Растеризация — конвертация двумерного представления сцены в растровое изображение, выполняется на аппаратном уровне видеокартой. Наиболее распространенные технологии графического конвейера — OpenGL и Direct3D.

- *Модель* — описание или набор данных, представляющий форму объекта. В JavaFX-технологии модель представлена экземпляром класса GUI-компонента.
- *Сцена* — это скомпонованный в рабочей области набор моделей и объектов, вызывающих различные эффекты, например, источник света и камера, которые создают эффекты освещенности и перспективы. Модели внутри сцены характеризуются размером и взаимным расположением.
- *Полигон* (polygon) — замкнутая фигура, созданная путем соединения отрезков, где каждый конец отрезка соединяется только с одним концом двух других отрезков (треугольник, прямоугольник, окружность и т. д.). Отрезки называются краями или сторонами (edges или sides), а точки соединения отрезков — вершинами (vertices).
- *Растровое изображение модели*, интегрированное в сцену, называется *спрайтом* (sprite). Такой рендеринг называется предварительным (pre-rendering), когда используются предварительно сгенерированные изображения моделей перед отображением всей сцены в реальном времени.
- *Граф сцены* (scene graph) — структура данных, коллекция узлов (node) дерева, которая упорядочивает логическую структуру сцены.
- *Компьютерная анимация* — быстрый показ последовательности изображений, созданных с помощью компьютерной графики, для создания иллюзии движения. В технологии JavaFX используются два вида анимации — *анимация по ключевым кадрам*, когда разработчик расставляет ключевые кадры на временной шкале, а промежуточные кадры автоматически генерируются программным обеспечением, и *программируемая анимация*, когда движение отдельных объектов программируется.
- *Ключевой кадр* (key frame) указывает значение свойства в определенное время в период выполнения анимации. Анимация создается с помощью изменения таких свойств объектов, как размер, положение, цвет и т. д. С помощью ключевых кадров задаются значения свойств на временной шкале, а программное обеспечение само генерирует плавные переходы между ключевыми кадрами.

Компоненты графического интерфейса пользователя (Graphical User Interface, GUI) JavaFX-приложения образуют сцену, логическая структура которой описывается графом сцены. Отображением GUI-интерфейса JavaFX-приложения является графическое представление графа сцены.

Для отображения GUI-интерфейса, разработанного на основе платформы JavaFX, и соответственно создания графического представления графа сцены, среда выполнения JavaFX Runtime предоставляет графическую систему, содержащую следующие модули.

- Prism производит растеризацию и рендеринг JavaFX-сцен с использованием аппаратного ускорения и на основе технологии DirectX 9 — для Windows XP и Windows Vista, DirectX 11 — для Windows 7 и OpenGL — для Mac, Linux, встраиваемых систем. При отсутствии поддержки аппаратного ускорения рендеринг осуществляется на основе технологии Java2D.
- Glass Windowing Toolkit — платформозависимая реализация, связывающая платформу JavaFX с операционной системой компьютера. Помимо обеспечения системных сервисов управления окном, видом и временем, система Glass отвечает за очередь событий. В отличие от графической системы Abstract Window Toolkit (AWT), которая создает собственную очередь событий и два потока — один для работы Peer-компонентов, а другой для работы Java-компонентов, система Glass использует очередь событий операционной системы и работает в том же потоке, что и JavaFX-приложение. При этом основной поток JavaFX-приложения отличается от AWT и Swing потока Event Dispatch Thread (EDT).
- Quantum Toolkit связывает системы Prism и Glass вместе и делает их доступными для других модулей среды выполнения JavaFX Runtime.
- Media Engine обеспечивает воспроизведение аудиофайлов MP3, AIFF и WAV и видеофайлов FLV.
- Web Engine основывается на проекте WebKit (**http://www.webkit.org/**) и обеспечивает поддержку HTML5, CSS, JavaScript, DOM и SVG, отображение локального и удаленного HTML-контента, обновление и редактирование HTML-контента с поддержкой истории и навигации, обработки событий, выполнения JavaScript-кода и применения эффектов.

Для обеспечения работы JavaFX-приложения среда выполнения JavaFX Runtime создает следующий набор параллельных потоков:

- основной поток JavaFX-приложения JavaFX Application Thread отвечает за обновление сцены, обработку анимации и событий;
- поток рендеринга системы Prism, который может содержать дополнительные потоки растеризации, отвечает за отрисовку сцены;
- фоновый медиапоток отвечает за декодирование, буферизацию и воспроизведение аудио и видео.

Синхронизация графа сцены с его графическим представлением осуществляется с помощью событий `Pulse`, которые генерируются средой выполнения JavaFX Runtime с максимальной частотой 1/60 секунды и посылаются в очередь событий при анимации и всякий раз, когда изменяется граф сцены, вызывая перерисовку сцены с применением компоновок и стилей.

Программный интерфейс JavaFX API

Программный интерфейс JavaFX API дает возможность разрабатывать приложения Rich Client Application (RIA) с насыщенным графическим интерфейсом пользователя, код которых сочетает широкие возможности платформы Java с богатой графической и медиафункциональностью платформы JavaFX. Программный интерфейс JavaFX API версии 2.0 содержит перечисленные далее пакеты.

- javafx.animation дает возможность создавать программируемую анимацию GUI-компонентов со встроенной временной шкалой и анимацию по ключевым кадрам.
- javafx.application обеспечивает жизненный цикл JavaFX-приложения.
- javafx.beans содержит интерфейс `Observable`, который является базовым интерфейсом для интерфейсов связывания данных и свойств компонентов JavaFX Beans и позволяет присоединить слушателя `InvalidationListener` для обработки события недействительности значения.
- javafx.beans.binding позволяет создавать связывание данных — синхронизацию объектов-источников данных, при которой изменения одного объекта автоматически отражаются в других объектах.
- javafx.beans.property обеспечивает новую архитектуру компонентов JavaFX Beans для представления свойств объектов, улучшающую и расширяющую модель JavaBeans-компонентов.
- javafx.beans.value содержит интерфейсы `ObservableValue` и `WritableValue` и их расширения, а также интерфейс `ChangeListener`. Интерфейс `WritableValue` реализуется свойствами компонентов JavaFX Beans, интерфейс `ObservableValue` — свойствами компонентов JavaFX Beans и классами связывания данных. Объект `ObservableValue` позволяет присоединить слушателя `ChangeListener` для обработки события изменения значения и слушателя `InvalidationListener` для обработки события недействительности значения. Интерфейс `WritableValue` обеспечивает считывание и запись значения.
- javafx.collections представляет расширение интерфейсов `java.util.List` и `java.util.Map` платформы Java Collections Framework.
- javafx.concurrent позволяет выполнять код в фоновом потоке, отличном от основного потока JavaFX-приложения.
- javafx.embed.swing обеспечивает встраивание JavaFX-контента в Swing-приложения.
- javafx.event представляет модель событий платформы JavaFX 2.0.
- javafx.fxml содержит класс `FXMLLoader`, обеспечивающий загрузку XML-файла, содержащего декларативное FXML-описание графического интерфейса пользователя.
- javafx.geometry содержит классы двумерной графики.

- javafx.scene содержит базовые классы программного интерфейса графа сцены JavaFX Scene Graph API.
- javafx.scene.chart обеспечивает создание диаграмм для представления данных.
- javafx.scene.control предоставляет GUI-компоненты контроля.
- javafx.scene.control.cell содержит классы, представляющие ячейки.
- javafx.scene.effect обеспечивает создание эффектов для GUI-компонентов.
- javafx.scene.image содержит классы для загрузки и показа изображений.
- javafx.scene.input обеспечивает обработку событий, инициированных мышью или клавиатурой.
- javafx.scene.layout содержит контейнеры, компонующие свои дочерние узлы графа сцены различным образом.
- javafx.scene.media обеспечивает интеграцию аудио- и видеоконтента в JavaFX-приложения.
- javafx.scene.paint позволяет заполнять области цветом или градиентом цветов и создавать цветной фон сцены.
- javafx.scene.shape содержит классы 2D-графики.
- javafx.scene.text обеспечивает встраивание текста в граф сцены.
- javafx.scene.transform обеспечивает пространственные трансформации графических объектов: вращение, масштабирование, перемещение и сдвиг.
- javafx.scene.web позволяет добавлять HTML-контент в JavaFX-приложения.
- javafx.stage содержит контейнеры верхнего уровня окна JavaFX-приложения.
- javafx.util содержит вспомогательные классы и интерфейсы JavaFX API.

> ***Примечание***
>
> Публичные свойства, поля, конструкторы и методы классов пакетов программного интерфейса JavaFX 2.0 API рассматриваются в приложениях, которые можно найти в электронном архиве *(см. приложение в конце книги)*.

Классы пакетов программного интерфейса JavaFX 2.0 API предоставляют для использования в разработке свои публичные свойства, поля, конструкторы и методы. При этом свойства записи/чтения JavaFX-классов доступны с помощью традиционных методов `getXXX()` и `setXXX()`, а также методов JavaFX Beans свойств:

```
public XXXProperty XXXProperty()
```

Свойства только для чтения JavaFX-классов доступны с помощью методов `getXXX()` и методов JavaFX Beans свойств.

Модель программирования приложений платформы JavaFX 2.0

Один и тот же код JavaFX-приложения может запускаться в качестве настольного приложения, которое разворачивается на клиентском компьютере автономно, мо-

жет разворачиваться как приложение Java Web Start или отображаться в Web-браузере как JavaFX-апплет, встроенный в HTML-страницу.

Точкой входа в JavaFX-приложение служит Java-класс, расширяющий абстрактный класс `javafx.application.Application` и содержащий метод `main()`:

```
public class JavaFXApp extends Application {
  public static void main(String[] args) {
    launch(args);
  }
  public void init(){
    //Инициализация приложения
    . . .
  }
  @Override
  public void start(Stage primaryStage) {
    //Установка параметров сцены
    . . .
    primaryStage.setScene(scene);
      primaryStage.setVisible(true);
  }

  public void stop(){
    //Освобождение ресурсов приложения
    . . .
  }
}
```

В методе `main()` главного класса JavaFX-приложения вызывает метод `launch()` класса `Application`, отвечающий за загрузку JavaFX-приложения. Кроме того, главный класс JavaFX-приложения должен переопределить абстрактный метод `start()` класса `Application`, обеспечивающий создание и отображение сцены JavaFX-приложения.

Методы `init()` и `stop()` класса `Application` могут использоваться для инициализации данных и освобождения ресурсов JavaFX-приложения.

Так как метод `init()` вызывается перед созданием главного потока приложения JavaFX Application Thread, то инициализация JavaFX-приложения в методе `init()` с участием узлов графа сцены должна осуществляться с применением статического метода `javafx.application.Platform.runLater()`.

Для выполнения JavaScript-кода на Web-странице, содержащей JavaFX-приложение, главный класс JavaFX-приложения может использовать метод `getHostServices()` класса `Application` и объект `netscape.javascript.JSObject`.

Обработка входных аргументов или параметров в главном классе JavaFX-приложения может быть осуществлена с помощью вызова метода `getParameters()` класса `Application`.

Улучшить отображение и обработку процесса запуска и загрузки JavaFX-приложения можно несколькими способами. Первый способ — это использование обработчика `onGetSplash` JavaScript-библиотеки Deployment Toolkit API для создания заставки запуска JavaFX-апплета, встроенного в Web-страничку. Другой способ — это применение CSS-стилей к Preloader-предзагрузчику по умолчанию. И наконец, можно создать свой класс предзагрузчика, расширяющий абстрактный класс `javafx.application.Preloader`, и сослаться на него в JNLP-дескрипторе развертывания JavaFX-приложения. При этом для связи главного класса JavaFX-приложения с предзагрузчиком можно использовать метод `notifyPreloader()` класса `Application`.

Метод `start()` класса `Application` содержит в качестве параметра объект `javafx.stage.Stage`, представляющий графический контейнер главного окна JavaFX-приложения. Данный объект `Stage` создается средой выполнения при запуске JavaFX-приложения и передается в метод `start()` главного класса JavaFX-приложения, что позволяет использовать методы объекта `Stage` для установки и отображения сцены JavaFX-приложения. Вместо объекта `Stage`, аргумента метода `start()`, разработчик может создать свой экземпляр класса `Stage` для отображения сцены JavaFX-приложения.

Перед установкой и отображением сцены в графическом контейнере `Stage` главного окна JavaFX-приложения необходимо создать граф сцены, состоящий из корневого узла и его дочерних элементов, и на его основе создать объект `javafx.scene.Scene` сцены.

Как правило, в качестве корневого узла используется объект `javafx.scene.Group`, который создается с помощью конструктора и выступает в качестве аргумента конструктора при создании объекта `javafx.scene.Scene`.

Дочерние узлы графа сцены, представляющие графику, элементы контроля GUI-интерфейса, медиаконтент, добавляются в корневой узел с помощью метода `getChildren().add()` или метода `getChildren().addAll()`. При этом дочерние узлы могут иметь визуальные эффекты, режимы наложения, CSS-стили, прозрачность, трансформации, обработчики событий, участвовать в анимации по ключевым кадрам, программируемой анимации и др.

Развертывание JavaFX-приложений

Как уже было сказано, один и тот же код JavaFX-приложения может разворачиваться как настольное приложение, как приложение Java Web Start или отображаться в Web-браузере как JavaFX-апплет, встроенный в HTML-страницу. Такая универсальность использования Java-кода обеспечивается моделью развертывания приложений платформы JavaFX или, точнее, сборкой JavaFX-приложения.

Скомпилировать и собрать Java-код в JavaFX-приложение можно несколькими способами.

Первый способ — это использовать среду разработки NetBeans IDE с поддержкой платформы JavaFX 2.0. При этом проект приложения должен иметь NetBeans-

шаблон **JavaFX Application**. С помощью выбора меню **Очистить и построить** среды NetBeans в каталоге проекта создается папка dist, содержащая:

- ❒ исполняемый JAR-файл JavaFX-приложения;
- ❒ JNLP-файл для JWS-развертывания приложения;
- ❒ HTML-страничку, включающую JavaFX-апплет и гиперссылку на JNLP-файл;
- ❒ папку web-files с JavaScript-файлом dtjava.js инструмента Deployment Toolkit и изображениями загрузки и ошибки.

Скомпилировать и собрать Java-код в JavaFX-приложение можно и другим способом, с помощью команды `-makeall` инструмента командной строки JavaFX Packager tool набора JavaFX SDK:

```
javafxpackager -makeall -appclass [полное имя главного класса] -name
"[заголовок HTML-страницы]" -width [ширина апплета] -height [высота апплета]
```

> ***Примечание***
>
> Команда `-makeall` компилирует исходный код и комбинирует команды `-createjar` и `-deploy`, обеспечивающие упаковку в JAR-файлы с конвертацией CSS-файлов в бинарный формат и генерацию HTML-страницы и JNLP-файла. С помощью команды `-signjar` можно также подписать приложение для выхода за пределы "песочницы". В среде NetBeans для создания цифровой подписи приложения необходимо в свойствах проекта выбрать опцию **Request unrestricted access**.

Сборка JavaFX-приложения может быть также осуществлена с помощью инструмента `Ant` и Ant-библиотеки JavaFX SDK:

```
<taskdef resource="com/sun/javafx/tools/ant/antlib.xml"
         uri="javafx:com.sun.javafx.tools.ant"
         classpath="${javafx.sdk.path}/tools/ant-javafx.jar"/>
//Создание JAR-файла приложения
<fx:jar destfile="dist-web/[имя приложения].jar">
  <fx:application mainClass="[имя главного класса приложения]"/>
  <fileset dir="build/classes/">
    <include name="**"/>
  </fileset>
</fx:jar>
//Создание цифровой подписи JAR-файла приложения с использованием сертификата
<fx:signjar destdir="dist"
            keyStore="sampleKeystore.jks" storePass="****"
            alias="javafx" keyPass="****">
  <fileset dir='dist/*.jar'/>
</fx:signjar>
//Генерация пакета, содержащего JAR-файлы, JNLP-файл и HTML-страницу
<fx:deploy width="800" height="600"
           outdir="dist-web" outfile="[имя приложения]">
  <fx:info title=""/>
  <fx:application name=""
            mainClass="[имя главного класса приложения]"/>
```

```
  <fx:resources>
    <fx:fileset dir="dist-web" includes="[имя приложения].jar"/>
  </fx:resources>
</fx:deploy>
```

Примечание

Плагин среды NetBeans для сборки JavaFX-приложения "за кадром" использует Ant-библиотеку JavaFX SDK.

Таким образом, сборка JavaFX-приложения с помощью NetBeans-плагина или инструмента JavaFX Packager tool содержит все необходимые файлы для запуска JavaFX-кода как настольного приложения, как JWS-приложения и как JavaFX-апплета.

Исполняемый JAR-файл JavaFX-приложения не содержит библиотек платформы JavaFX, вместо этого при компиляции и сборке генерируется пакет com\javafx\main с классом `Main`, который и служит на самом деле точкой входа в JavaFX-приложение и обеспечивает связь JavaFX-приложения с установленной на локальном компьютере средой выполнения JavaFX Runtime.

Пакет com\javafx\main также содержит сгенерированный Java-апплет NoJavaFXFallback, отображаемый в случае, если локальный компьютер не имеет установленной требуемой версии JRE или JavaFX Runtime.

Сгенерированный JNLP-файл JavaFX-приложения содержит следующий код:

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0" xmlns:jfx="http://javafx.com"
      href="JavaFXApplication.jnlp">
  <information>
    <title>JavaFXApplication</title>
    <vendor> </vendor>
    <description>Sample JavaFX 2.0 application.</description>
    <offline-allowed/>
  </information>
<resources os="Windows" arch="x86">
  <jfx:javafx-runtime version="2.0+" href="http://download.oracle.com/otn-
pub/java/javafx/javafx-windows-i586__Vlatest.exe "/>
</resources>
<resources os="Windows" arch="x64">
  <jfx:javafx-runtime version="2.0+" href="http://download.oracle.com/otn-
pub/java/javafx/javafx-windows-x64__Vlatest.exe "/>
</resources>
<resources>
  <j2se version="1.6+" href="http://java.sun.com/products/autodl/j2se"/>
  <jar href="JavaFXApplication.jar" size="13142" download="eager" />
</resources>
<applet-desc width="800" height="600"
             main-class="com.javafx.main.NoJavaFXFallback"
             name="JavaFXApplication" />
```

```
  <jfx:javafx-desc width="800" height="600"
            main-class="javafxapplication.JavaFXApplication"
            name="JavaFXApplication" />
  <update check="background"/>
</jnlp>
```

В данном JNLP-файле элементы `<jfx:javafx-runtime>` должны содержать верные адреса для загрузки требуемой версии среды выполнения JavaFX Runtime.

Элемент `<jar>` JNLP-файла указывает адрес исполняемого JAR-файла JavaFX-приложения, элемент `<applet-desc>` — имя Java-апплета, отображаемого в случае отсутствия необходимой версии JRE или JavaFX Runtime, элемент `<jfx:javafx-desc>` указывает имя главного класса JavaFX-приложения.

HTML-страничка, содержащая JavaFX-апплет и гиперссылку на JNLP-файл, состоит из следующего кода:

```
<html><head>
  <SCRIPT src="./web-files/dtjava.js"></SCRIPT>
<script>
  function launchApplication(jnlpfile) {
    dtjava.launch(  {
      url : 'JavaFXApplication.jnlp',
      jnlp_content : '. . .'
    },
    { javafx : '2.0+'  },
    {}
    );
    return false;
  }
</script>

<script>
  function javafxEmbed() {
    dtjava.embed(
    {  id : 'fxApp',
       url : 'JavaFXApplication.jnlp',
       placeholder : 'javafx-app-placeholder',
       width : 800,
       height : 600,
       jnlp_content : '. . .'  },
    {  javafx : '2.0+'  },
    {}
    );
  }
  <!-- Embed FX application into web page once page is loaded -->
  dtjava.addOnloadCallback(javafxEmbed);
</script>
```

```
</head><body>
  <h2>Test page for <b>JavaFXApplication</b></h2>
  <b>Webstart:</b> <a href='JavaFXApplication.jnlp' onclick="return
launchApplication('JavaFXApplication.jnlp');">click to launch this app as
webstart</a><br><hr><br>
  <!-- Applet will be inserted here -->
  <div id='javafx-app-placeholder'></div>
</body></html>
```

Данная HTML-страничка использует JavaScript-файл dtjava.js папки web-files для загрузки и встраивания JavaFX-приложения в Web-страницу. При этом размеры JavaFX-апплета можно изменять с помощью значений параметров `width` и `height` функции `dtjava.embed()`. Кроме методов `dtjava.launch(app, platform, callbacks)` и `dtjava.embed(app, platform, callbacks)`, обеспечивающих загрузку не встроенного JavaFX-приложения и встраивание JavaFX-приложения в Web-страницу в качестве JavaFX-апплета соответственно, программный интерфейс Deployment Toolkit API предлагает следующие методы:

- `dtjava.install(platform, callbacks)` — инсталлирует платформу;
- `dtjava.validate(platform)` — проверяет наличие требуемой платформы;
- `dtjava.hideSplash(id)` — скрывает панель загрузки для приложения с указанным идентификатором.

Для обработки событий Deployment Toolkit API содержит следующие обработчики обратного вызова:

- `onDeployError: function(app, mismatchEvent)` — вызывается при отсутствии требуемой платформы;
- `onInstallFinished: function(placeholder, component, status, relaunchNeeded)` — вызывается по окончании инсталляции компонента;
- `onInstallNeeded: function(app, platform, cb, isAutoinstall, needRelaunch, launchFunc)` — вызывается, если приложение требует инсталляции дополнительных компонентов;
- `onInstallStarted: function(placeholder, component, isAuto, restartNeeded)` — вызывается перед инсталляцией компонента;
- `onGetNoPluginMessage function(app)` — вызывается для создания отображаемого контента в отсутствие инсталлированного Java-плагина Web-браузера;
- `onGetSplash: function(app)` — вызывается для создания панели загрузки JavaFX-апплета;
- `onJavascriptReady: function(id)` — вызывается, когда JavaFX-апплет готов принимать JavaScript-вызовы;
- `onRuntimeError: function(id)` — вызывается при ошибке загрузки приложения.

ГЛАВА 2

Компоненты графического интерфейса пользователя

Компоненты графического интерфейса пользователя платформы JavaFX представлены такими пакетами JavaFX API, как javafx.scene.control, javafx.scene.chart, javafx.scene.image, javafx.scene.layout, javafx.scene.media, javafx.scene.shape, javafx.scene.text, javafx.scene.web и javafx.stage.

Все компоненты GUI-интерфейса являются объектами `Node` узлов графа сцены и характеризуются идентификатором, CSS-стилем, границами, визуальными эффектами, прозрачностью, трансформациями, обработчиками событий, состоянием, режимом наложения и участием в анимации.

Пакет javafx.scene.control предоставляет такие GUI-компоненты, как панель `Accordion`, кнопку `Button`, флажок `CheckBox`, список `ChoiceBox`, контекстное меню `ContextMenu`, гиперссылку `Hyperlink`, метку `Label`, список `ListView`, меню `Menu`, панель `MenuBar`, кнопку `MenuButton`, поле ввода пароля `PasswordField`, индикатор `ProgressBar`, индикатор `ProgressIndicator`, переключатель `RadioButton`, панель `ScrollPane`, прокрутку `ScrollBar`, разделитель `Separator`, бегунок `Slider`, кнопку `SplitMenuButton`, панель `SplitPane`, таблицу `TableView`, панель с вкладками `TabPane`, многострочное поле `TextArea`, поле ввода `TextField`, панель `TitledPane`, кнопку `ToggleButton`, группу `ToggleGroup`, панель `ToolBar`, окно подсказки `Tooltip`, дерево `TreeView`.

Пакет javafx.scene.chart обеспечивает создание диаграмм `PieChart`, `AreaChart`, `BarChart`, `BubbleChart`, `LineChart` и `ScatterChart`.

Пакет javafx.scene.image содержит GUI-компонент изображения `ImageView`.

Пакет javafx.scene.layout предоставляет панели компоновки `AnchorPane`, `BorderPane`, `FlowPane`, `GridPane`, `HBox`, `StackPane`, `TilePane`, `VBox`.

Пакет javafx.scene.media содержит GUI-компоненты медиаконтента `MediaView` и аудиоконтента `AudioClip`.

Пакет javafx.scene.shape обеспечивает рисование геометрических форм с помощью таких GUI-компонентов, как `Arc`, `Circle`, `CubicCurve`, `Ellipse`, `Line`, `Path`, `Polygon`, `Polyline`, `QuadCurve`, `Rectangle`, `SVGPath` и `Path`.

Пакет javafx.scene.text содержит GUI-компонент текста `Text`.

Пакет javafx.scene.web обеспечивает отображение HTML-контента с помощью GUI-компонента `WebView` и редактирование HTML-контента посредством GUI-компонента `HTMLEditor`.

Пакет javafx.scene содержит группу `Group` и сцену `Scene`.

Пакет javafx.stage предоставляет GUI-компоненты окон `Stage`, `Popup` и `FileChooser`.

Кнопка *Button*

Компонент `Button` представлен классом `javafx.scene.control.Button`, экземпляр которого может быть создан с помощью класса-фабрики `ButtonBuilder` или посредством конструктора:

```
Button btn = new Button();
```

Класс `Button` имеет:

- унаследованные от класса `javafx.scene.Node` свойства: `blendMode`, `boundsInLocal`, `boundsInParent`, `cacheHint`, `cache`, `clip`, `cursor`, `depthTest`, `disabled`, `disable`, `effect`, `eventDispatcher`, `focused`, `focusTraversable`, `hover`, `id`, `inputMethodRequests`, `layoutBounds`, `layoutX`, `layoutY`, `managed`, `mouseTransparent`, `onDragDetected`, `onDragDone`, `onDragDropped`, `onDragEntered`, `onDragExited`, `onDragOver`, `onInputMethodTextChanged`, `onKeyPressed`, `onKeyReleased`, `onKeyTyped`, `onMouseClicked`, `onMouseDragged`, `onMouseEntered`, `onMouseExited`, `onMouseMoved`, `onMousePressed`, `onMouseReleased`, `opacity`, `parent`, `pickOnBounds`, `pressed`, `rotate`, `rotationAxis`, `scaleX`, `scaleY`, `scaleZ`, `scene`, `style`, `translateX`, `translateY`, `translateZ`, `visible`;
- унаследованное от класса `javafx.scene.Parent` свойство `needsLayout`;
- унаследованные от класса `javafx.scene.control.Control` свойства: `contextMenu`, `height`, `maxHeight`, `maxWidth`, `minHeight`, `minWidth`, `prefHeight`, `prefWidth`, `skin`, `tooltip`, `width`;
- унаследованные от класса `javafx.scene.control.Labeled` свойства: `alignment`, `contentDisplay`, `font`, `graphic`, `graphicTextGap`, `labelPadding`, `mnemonicParsing`, `textAlignment`, `textFill`, `textOverrun`, `text`, `underline`, `wrapText`;
- унаследованные от класса `javafx.scene.control.ButtonBase` свойства `armed` и `onAction`;
- собственные свойства `cancelButton` и `defaultButton`.

Для создания кнопки `Button` графического интерфейса пользователя JavaFX-приложения откроем среду NetBeans с поддержкой платформы JavaFX 2.0 *(см. главу 1)* и в меню **Файл** выберем **Создать проект | JavaFX | JavaFX Application**, нажмем кнопку **Далее** и введем имя проекта JavaFXApplicationButton, отметим флажок **Create Main Class**, введем имя класса `javafxapplication.JavaFXApplication` и нажмем кнопку **Готово**.

В окне редактора среды NetBeans дополним код сгенерированного класса `JavaFXApplication`, который расширяет класс `javafx.application.Application` и имеет два метода — `main()` и `start()`.

Примечание

Проект JavaFXApplicationButton с примером создания кнопки `Button` находится в папке Примеры\Глава2 в электронном архиве *(см. приложение в конце книги)*.

В методе `main()` класса `JavaFXApplication` вызывается функция загрузки JavaFX-приложения:

```
Application.launch(JavaFXApplication.class, args);
```

В методе `start()` класса `JavaFXApplication` производится обработка объекта `Stage`, передаваемого методу `start()` в качестве аргумента, с созданием и отображением графа сцены:

1. Устанавливается заголовок основного окна JavaFX-приложения:

```
primaryStage.setTitle("Тестирование GUI-компонентов");
```

2. Создается корневой узел графа сцены и на его основе экземпляр сцены:

```
Group root = new Group();
Scene scene = new Scene(root, 300, 300, Color.LIGHTGREEN);
```

3. Создается экземпляр кнопки `Button`, который помещается в левый верхний угол основного окна JavaFX-приложения и для которого устанавливается текст кнопки, а также обработчик событий кнопки:

```
Button btn;
  btn = new Button();
  btn.setLayoutX(20);
  btn.setLayoutY(20);
  btn.setText("Тестировать свойства");
  btn.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent event) {. . .} });
```

4. В обработчике событий последовательно выводятся в консоль значения свойств кнопки:

```
System.out.println("Свойства, унаследованные от класса Node:"+"\n"+
"Свойство blendMode: "+btn.blendModeProperty().getValue()+"\n"+
"Свойство boundsInLocal: "+btn.boundsInLocalProperty().getValue()+"\n"+
. . .);
```

5. Создается вторая кнопка, которая размещается ниже первой кнопки. Для второй кнопки устанавливается текст, стиль, предпочтительные размеры и обработчик событий:

```
Button btnON;
btnON=ButtonBuilder.create().build();
  btnON.setLayoutX(20);
  btnON.setLayoutY(150);
  btnON.setText("Установить свойства");
  btnON.setStyle("-fx-font:  bold italic 12pt Arial;-fx-text-fill: #660000;
-fx-background-color: #ff99ff; -fx-border-width: 3px; -fx-border-radius: 30;
-fx-background-radius: 30;-fx-border-color: #660066;" );
```

```
  btnON.setPrefSize(200,30);
  btnON.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent event) {. . . }});
```

6. В обработчике событий второй кнопки устанавливаются такие свойства первой кнопки, как режим наложения, маска, курсор мыши, визуальный эффект, управление компоновкой, прозрачность, вращение, перемещение, масштабирование, предпочтительные размеры, всплывающая подсказка, значок, стиль, выравнивание, подчеркивание текста, перенос строк, привязка к клавиши активации, задний план:

```
btn.setBlendMode(BlendMode.DARKEN);
javafx.scene.shape.Circle clip=new javafx.scene.shape.Circle(75,53,80);
// btn.setClip(clip);
  btn.setCursor(Cursor.CLOSED_HAND);
DropShadow effect=new DropShadow();
  effect.setOffsetX(10);
  effect.setOffsetY(10);
  btn.setEffect(effect);
  //btn.setManaged(false);
  //btn.setMouseTransparent(true);
  btn.setOpacity(0.5);
  btn.setRotate(10);
  btn.setLayoutX(80);
  btn.setScaleX(1.8);
  btn.setLayoutY(170);
  btn.setTranslateZ(-50);
  btn.setPrefSize(150,100);
  btn.setTooltip(new Tooltip
    ("Это кнопка тестирования свойств класса Button"));
Image im=new Image(this.getClass().getResource("image.png").toString());
ImageView imv=new ImageView(im);
  imv.setFitHeight(50);
  imv.setFitWidth(50);
btn.setGraphic(imv);
btn.setStyle("-fx-font:  bold italic 10pt Helvetica;");
//btn.setFont(Font.font("Helvetica", FontWeight.BOLD,
//            FontPosture.ITALIC, 10));
btn.setAlignment(Pos.CENTER);
btn.setContentDisplay(ContentDisplay.RIGHT);
btn.setUnderline(true);
btn.setWrapText(true);
//btn.setCancelButton(true);
//btn.toBack();
```

7. И в заключение обе кнопки добавляются в корневой узел, для объекта `Stage` устанавливается созданная сцена, и объект `Stage` становится видимым:

```
root.getChildren().add(btnON);
root.getChildren().add(btn);
primaryStage.setScene(scene);
primaryStage.show();
```

Запустить созданное JavaFX-приложение можно, щелкнув в окне **Проекты** среды NetBeans правой кнопкой мыши на узле проекта JavaFXApplication и выбрав пункт **Очистить и построить**, а затем — **Выполнить**.

В результате можно увидеть две кнопки (рис. 2.1), нажатие одной из которых позволит вывести в консоль значения свойств этой кнопки, а нажатие другой кнопки приведет к изменению первой кнопки (рис. 2.2).

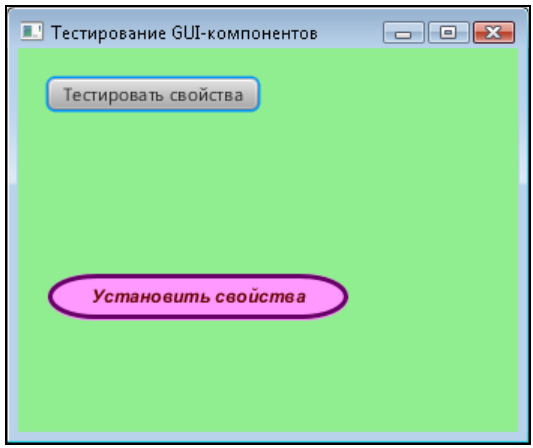


Рис. 2.1. JavaFX-приложение с GUI-интерфейсом, содержащим две кнопки

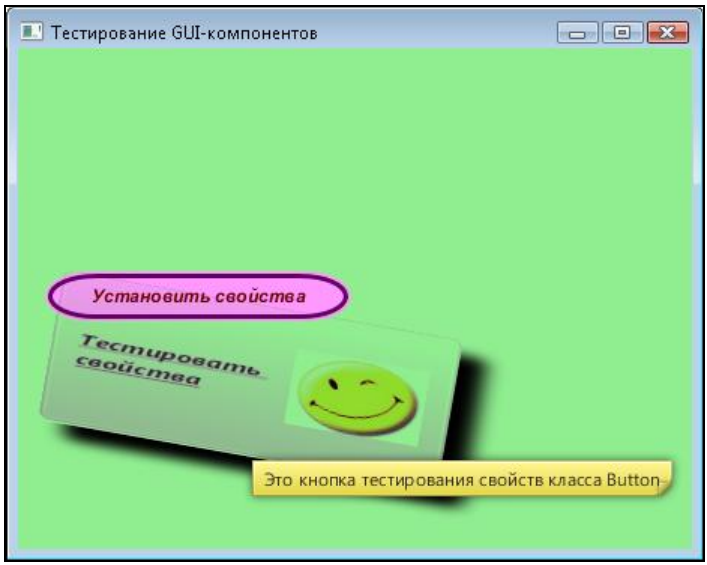


Рис. 2.2. Изменение свойств первой кнопки с помощью нажатия второй кнопки

Нажав кнопку **Тестировать свойства** до ее изменения и после изменения кнопки, в окне **Вывод** среды NetBeans можно увидеть первоначальные и измененные значения свойств кнопки.

Из сравнения локальных границ кнопки, границ относительно родительского узла и компоновочных границ до и после изменения кнопки видно, что компоновочные границы `layoutBounds` соответствуют геометрическим границам компонента без учета эффектов, масок, трансформаций, но с учетом установленных размеров. Компоновочные границы `layoutBounds` компонента определяются в локальной системе координат компонента.

Локальные границы `boundsInLocal` также определяются в локальной системе координат компонента, однако включают в себя контур, эффекты и маски, но не учитывают трансформации.

Границы `boundsInParent` относительно родительского узла определяются в системе координат родительского узла и учитывают эффекты, маски и трансформации.

Границы до изменения кнопки:

```
Свойство boundsInLocal: BoundingBox [minX:-1.399999976158142,
minY:-1.399999976158142, minZ:0.0, width:171.79998779296875,
height:21.598827362060547, depth:0.0, maxX:170.3999878168106,
maxY:20.198827385902405, maxZ:0.0]
Свойство boundsInParent: BoundingBox [minX:18.600000381469727,
minY:18.600000381469727, minZ:0.0, width:171.79998779296875,
height:21.59882926940918, depth:0.0, maxX:190.39998817443848,
maxY:40.198829650878906, maxZ:0.0]
Свойство layoutBounds: BoundingBox [minX:0.0, minY:0.0, minZ:0.0, width:169.0,
height:18.798828125, depth:0.0, maxX:169.0, maxY:18.798828125, maxZ:0.0]
```

Границы после изменения кнопки:

```
Свойство boundsInLocal: BoundingBox [minX:-1.399999976158142,
minY:-1.399999976158142, minZ:0.0, width:171.79998779296875,
height:121.80000305175781, depth:0.0, maxX:170.3999878168106,
maxY:120.40000307559967, maxZ:0.0]
Свойство boundsInParent: BoundingBox [minX:9.117060661315918,
minY:145.5007781982422, minZ:-50.0, width:323.7459716796875,
height:173.64854431152344, depth:0.0, maxX:332.8630323410034,
maxY:319.1493225097656, maxZ:-50.0]
Свойство layoutBounds: BoundingBox [minX:0.0, minY:0.0, minZ:0.0, width:150.0,
height:100.0, depth:0.0, maxX:150.0, maxY:100.0, maxZ:0.0]
```

При получении фокуса кнопкой свойство `focused` принимает значение `true`.

При нажатии кнопки с помощью клавиши <Enter> свойство `hover` остается равным `false`, а при нажатии кнопки мыши свойство `hover` принимает значение `true`.

Значение `true` свойства `focusTraversable` означает, что фокус можно перемещать посредством клавиш-стрелок и клавиши <Tab>.

При установке свойства `managed` со значением `false` не учитываются максимальные, минимальные и предпочтительные размеры кнопки (рис. 2.3).

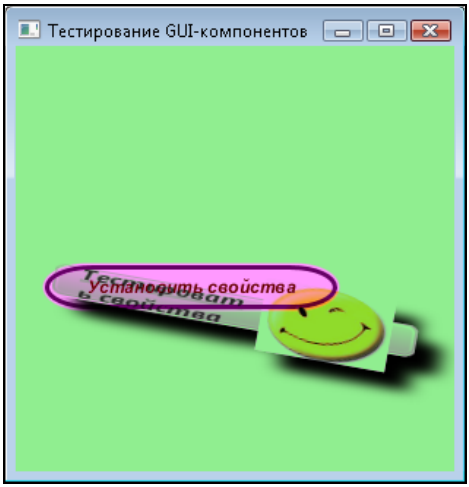


Рис. 2.3. Изменение кнопки со значением `false` свойства `managed`

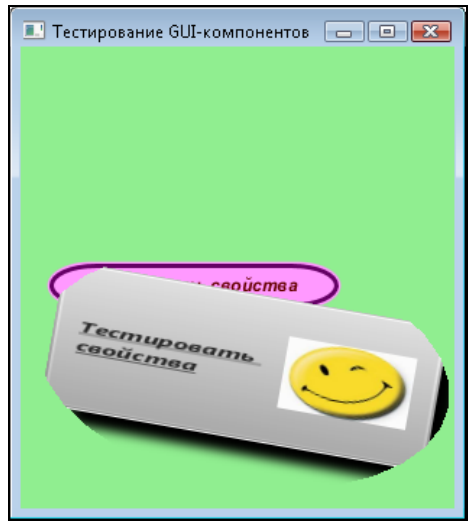


Рис. 2.4. Изменение кнопки с установкой маски

Определение узла-маски для кнопки нивелирует установку его режима наложения (рис. 2.4).

Установка свойства `cursor` позволяет изменять изображение курсора мыши при его наведении на кнопку.

При установке свойства `mouseTransparent` со значением `true` кнопка активируется только с помощью клавиатуры.

Присоединение эффекта к компоненту может нивелировать установку прозрачности компонента. В данном случае это происходит при установке тени для кнопки.

При нажатии кнопки с помощью мыши значение свойства `pressed` остается `false`, а значение свойства `armed` становится равным `true`, т. к. кнопка активируется не нажатием кнопки мыши, а нажатием и освобождением кнопки мыши.

Установка параметров шрифта текста на кнопке более надежна посредством определения CSS-стиля кнопки, чем с помощью установки свойства `font`, т. к. CSS-стиль кнопки может быть определен с учетом псевдоклассов.

Если установить свойство `cancelButton` со значением `true`, тогда кнопка будет активироваться клавишей <Esc>, а не клавишей <Enter>.

Порядок наложения одного узла на другой определяется очередностью добавления узлов графа сцены в его корневой узел. Однако данный порядок можно менять с помощью применения функций `toBack()` и `toFront()`.

Флажок *CheckBox*

Компонент `CheckBox` представлен классом `javafx.scene.control.CheckBox`, экземпляр которого может быть создан с помощью класса-фабрики `CheckBoxBuilder` или посредством одного из конструкторов: