AWT
Swing
JavaFX
SWT/JFace

G U I

J A V A

Графические интерфейсы пользователя Java

Тимур Машнин

Тимур Машнин

Графические интерфейсы пользователя Java

http://www.litres.ru/pages/biblio_book/?art=43650864
ISBN 9785005027429

Аннотация

Эта книга для тех, кто уже знаком с языком программирования Java и хотел бы научиться разрабатывать настольные приложения Java с графическим интерфейсом пользователя.С этой книгой Вы познакомитесь с такими Java библиотеками графического интерфейса пользователя, как AWT, Swing, SWT/JFace и JavaFX.

Содержание

Графические интерфейсы пользователя Java

Тимур Машнин

ISBN 978-5-0050-2742-9
Создано в интеллектуальной издательской системе Ridero

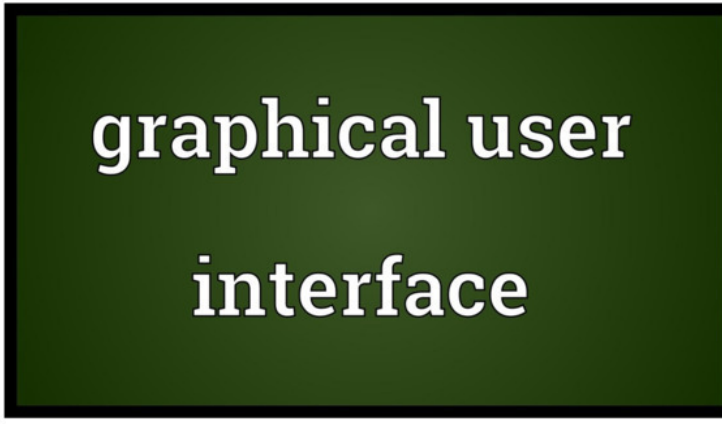
Введение

Графические интерфейсы пользователя Java

Лекция 1
Введение

Любое приложение, требующее взаимодействия с пользователем, должно иметь интерфейс пользователя или GUI Graphical User Interface.

От интерфейса пользователя зависит привлекательность и удобство работы с программой.

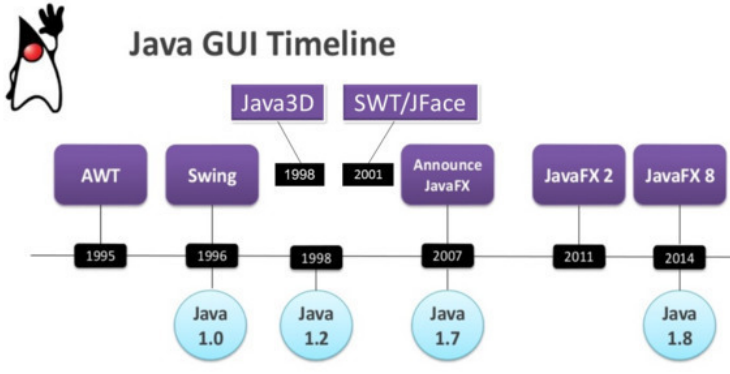


Интерфейс пользователя может быть, как низкоуровневым, в виде командной строки, так и иметь разнообразные графические компоненты – кнопки, переключатели, меню, поля ввода и другие элементы.

Технология Java предлагает набор библиотек классов и интерфейсов, на основе которых можно построить эффективный графический интерфейс пользователя, имеющий необходимый внешний вид и поведение для создания комфортной среды конечному потребителю.

Самой первой графической библиотекой платформы Java была библиотека AWT набора JDK 1.0.

Далее библиотека AWT была дополнена библиотекой Java 2D двухмерной графики и изображений.



Библиотека AWT обеспечивает для разработчика приложений возможность использования таких основных компонентов GUI-интерфейса как кнопка, флажок, список выбора, прокручивающийся список, метка, диалоговое окно, окно выбора файла, меню, панель с прокруткой, текстовая область и текстовое поле, использования функции Drag-and-Drop, возможность обработки событий UI-компонентов, компоновки компонентов в рабочей области, работы с цветом, шрифтом, графикой, рисования и печати.

Библиотека AWT считается тяжеловесной, так как она содержит нативную библиотеку java.awt.peer, через которую взаимодействует с операционной системой компьютера таким образом, что AWT-компоненты имеют своих двойников, реализованных для конкретной операционной системы,

с которыми они связаны интерфейсами пакета java.awt.peer.

Поэтому отображение AWT GUI-интерфейса зависит от операционной системы, в которой приложение развернуто.

Для расширения набора GUI-компонентов библиотеки AWT, устранения ее тяжеловесности и возможности выбора внешнего вида и поведения (Look and Feel) GUI-компонентов была создана графическая библиотека Swing.

Библиотека Swing основывается на библиотеке AWT и напрямую не связана, как библиотка AWT, с операционной системой, в которой она работает.

Поэтому библиотека Swing является уже легковесной.

Кроме того, библиотека Swing дополняет библиотеку AWT такими компонентами GUI-интерфейса как панель выбора цвета, индикатор состояния, переключатель radio button, слайдер и спиннер, панель с закладками, таблицы и деревья, расширенными возможностями компоновки GUI-компонентов, таймером, возможностью отображения HTML-контента.

С помощью библиотеки Swing стало возможным создать набор отображений Look and Feel, которые разработчик может выбирать, не оглядываясь на операционную систему, и изменять внешний вид GUI-интерфейса.

Также библиотека Swing реализует архитектуру MVC (Model-View-Controller) и потоковую модель Event Dispatch Thread (EDT).

Библиотека SWT (Standard Widget Toolkit) является альтернативой библиотекам AWT/Swing, однако она не включена в официальный релиз JRE/JDK.

Библиотека SWT была создана в процессе работы над проектом Eclipse и является попыткой взять лучшее из архитектур библиотек AWT и Swing и предоставить возможность создания быстрых GUI-интерфейсов с отображением Look and Feel, как можно более полно соответствующим операционной системе, в которой они работают.

Архитектура системы SWT построена таким образом, что SWT-компоненты представляют собой лишь Java-оболочки GUI-компонентов конкретной операционной системы.

Для операционной системы, в которой отсутствует реализация какого-либо компонента, система SWT обеспечивает Java-эмуляцию.

Так в системе SWT достигается скорость работы и полное соответствие внешнему виду и поведению операционной системе.

Для создания GUI-интерфейса система SWT предоставляет такие GUI-компоненты как кнопка, включая флажок и переключатель, список, метка, меню, текстовая область, диалоговое окно, индикатор прогресса, панель с прокруткой, слайдер и спиннер, таблица и дерево, панель с вкладками, панель выбора даты, панель инструментов, встроенный Web-браузер, гиперссылка, а также обеспечивает компоновку SWT-компонентов, встраивание AWT-компонен-

тов, отображение OpenGL-контента, печать, поддержку операций Drag and Drop, 2D-графики, технологии Win32 OLE.

Библиотека JFace дополняет библиотеку SWT и создана на основе библиотеки SWT с реализацией архитектуры MVC (Model-View-Controller), предоставляя такие MVC GUI-компоненты как таблица, дерево, список, текстовая область и диалоговое окно, обеспечивая определение пользовательских команд независимо от их представления в GUI-интерфейсе, управление шрифтами и изображениями, помощь пользователю в выборе соответствующего содержания для полей в GUI-компонентах, выполнение длительных задач.

Библиотека Java 3D была создана как высокоуровневая библиотека для создания 3D графики, включая анимацию 3D-объектов.

Библиотека Java 3D не включена в официальный релиз JRE/JDK и требует отдельной установки.

Библиотека Java 3D оперирует объектами, размещаемыми в графе сцены, который представляет собой дерево 3D-объектов и предназначен для визуализации.

Java3D-приложение может работать как настольное приложение, как апплет или как настольное приложение и апплет.

По сути, библиотека Java 3D является оберткой графических библиотек OpenGL и DirectX.

Каждая из вышеперечисленных графических библиотек

имеет свои преимущества и недостатки, в числе которых надо отметить тот факт, что данные библиотеки не удовлетворяют современным требованиям работы с медиаконтентом.

Кроме того, данные библиотеки не обеспечивают возможность декларативного создания GUI-интерфейса на основе языка XML.

Библиотека JavaFX была создана как универсальная платформа, предоставляющая современные GUI-компоненты с возможностью их декларативного описания, богатый набор библиотек для работы с медиаконтентом и 2D/3D графикой, а также высокопроизводительную среду выполнения приложений.

Первоначально, с 2007 по 2010 год, 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 отличалась от платформы JavaFX версии 1.x.

Платформа JavaFX 2.x больше не поддерживала язык

JavaFX Script, а вместо этого предлагала новый программный интерфейс JavaFX API для создания JavaFX-приложений полностью на языке Java.

Для альтернативного декларативного описания графического интерфейса пользователя платформа JavaFX 2.x предлагала новый язык FXML.

Кроме того, платформа JavaFX 2.x обеспечивала новые графический и медиа движки, улучшающие воспроизведение графического и мультимедийного контента, обеспечивала встраивание HTML-контента в приложения, новый плагин для Web-браузеров, широкий выбор GUI-компонентов с поддержкой CSS3, 2D графику, создание отчетов с диаграммами, интеграцию с библиотекой Swing.

При этом платформа JavaFX версии 2.x содержала:

– 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.x для среды выполнения NetBeans IDE 7.

– Примеры JavaFX-приложений.

– Приложение JavaFX Scene Builder 1.x для визуальной компоновки GUI-компонентов в GUI-интерфейс с использо-

ванием языка FXML.

В дальнейшем платформа JavaFX была полностью интегрирована в платформу Java 8, и перестала требовать отдельной инсталляции.

Кроме того, платформа JavaFX 8 была дополнена программным интерфейсом CSS API создание стилей, интеграцией с библиотекой SWT, программным интерфейсом Printing API, 3D графикой, программным интерфейсом SubScene API.

Также платформа JavaFX 8 предлагает улучшенный инструмент JavaFX Scene Builder 2.x для визуальной компоновки GUI-компонентов в GUI-интерфейс на основе языка FXML, дополненный поддержкой новых возможностей платформы JavaFX 8, программным интерфейсом JavaFX Scene Builder Kit для встраивания инструмента в Java приложения, возможностью добавлять пользовательские GUI-компоненты и др.

В настоящее время технология JavaFX обеспечивает создание мощного графического интерфейса пользователя (Graphical User Interface (GUI)), 2D и 3D графику для крупномасштабных приложений, ориентированных на обработку данных, насыщенных медиа-приложений, поставляющих разнообразный медиа-контент пользователю, Mashup-приложений, объединяющих различные Web-ресурсы для пользователя, обеспечивает создание компонентов высококачественной графики и анимации для Web-сайтов, различно-

го рода пользовательских программ, насыщенных графикой, анимацией и интерактивными элементами.

Библиотека AWT

Графические интерфейсы пользователя Java

Лекция 2
Библиотека AWT

Итак, самой первой графической Java-библиотекой была создана библиотека AWT (Abstract Window Toolkit).

Она была включена в первую версию JDK 1.0.

Затем библиотека AWT была дополнена библиотекой Java 2D API, расширяющей возможности работы с двухмерной графикой и изображениями.

Так как технология Java является платформо-независимой, то соответственно и графическая Java-библиотека должна быть платформо-независимой.

Сам по себе язык Java не обладает возможностями низко-

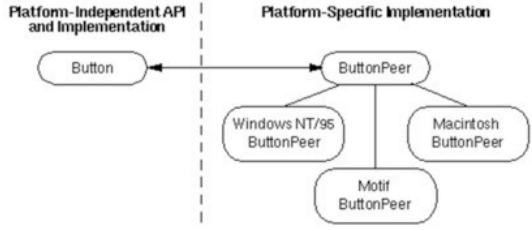
уровневого взаимодействия с конкретной операционной системой, обеспечивающими передачу информации от мышки или клавиатуры приложению и вывод пикселей на экран.

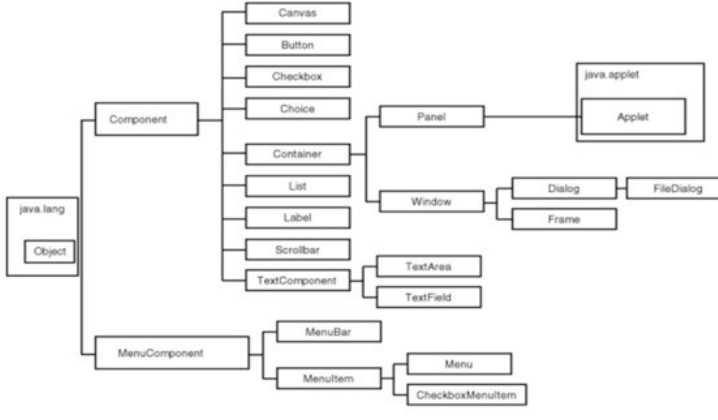
Поэтому библиотека AWT была создана так, что каждый AWT-компонент имеет своего двойника-пира peer – интерфейс, обеспечивающий взаимодействие с конкретной операционной системой.

Таким образом, переносимость графической библиотеки AWT обусловлена наличием реализации пакета java.awt.peer для конкретной операционной системы.

Вследствие этого, AWT-компоненты называют тяжеловесными.

Все AWT-компоненты, кроме элементов меню, представлены подклассами класса java.awt.Component.





Для элементов меню суперклассом является класс java.awt.MenuComponent.

Архитектура AWT устроена таким образом, что компоненты размещаются в контейнерах (суперкласс java.awt.Container) с помощью менеджеров компоновки – классов, реализующих интерфейс java.awt.LayoutManager.

Для настольных приложений корневое окно графического интерфейса пользователя представляет контейнер java.awt. Window, который в свою очередь должен содержать окно java.awt.Frame с заголовком и границами или диалоговое окно java.awt. Dialog, также имеющее заголовок и границы.

AWT-компоненты добавляются в панель java.awt.Panel – контейнер, который может содержать как компоненты, так

и другие панели.

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

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

В отличие от настольных приложений, апплеты являются управляемыми программными компонентами.

Это означает, что апплет не может быть запущен, как настольное приложение, одной только виртуальной машиной JVM.

Для его работы необходим браузер, который распознает тэги <APPLET> или <OBJECT> и <EMBED>, включающие апплет в HTML-страницу.

Главный класс апплета должен быть подклассом класса java. applet. Applet, при этом класс Applet служит интерфейсом между апплетом и браузером.

Жизненным циклом апплета управляет компонент Java Plug-in среды выполнения JRE.

Настольные приложения платформы Java SE – это независимые Java-приложения, которые выполняются виртуальной машиной JVM, при этом точкой входа в приложение является главный класс приложения, содержащий статический метод main.

Начиная с версии Java SE 7 с апреля 2013 года все Java-апплеты и приложения Web Start должны подписываться доверенным сертификатом.

Это фактически уничтожило свободную разработку и распространение апплетов, так как приобретение доверенного сертификата является платной и не дешевой услугой.

Более того, Java 9 вообще запрещает использование апплетов, которые теперь уходят в историю.

Java Web Start

Графические интерфейсы
пользователя Java

Лекция 3
Java Web Start

Java Web Start (JWS) – это технология, основанная на протоколе Java Network Launching Protocol (JNLP), позволяет загружать и запускать приложения с сайта, с помощью браузера.

Преимущество данной технологии в том, что пользователю не нужно самому специально инсталлировать приложение на своем компьютере и беспокоиться о его обновлениях.

Вся площадка для загрузки и старта приложения находится на сервере сайта.

Настольное Java приложение можно распространять дву-

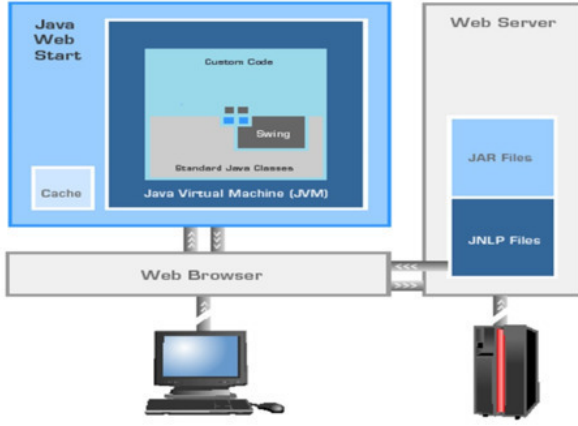
мя способами.

Можно выкладывать для скачивания версии приложения, которые будут скачиваться пользователем, и который сам должен следить за обновлениями приложения.

Или можно использовать технологию Java Web Start.

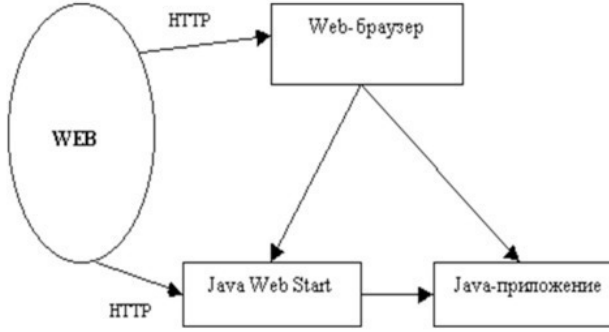
Java Web Start – это технология развертывания приложений, которая позволяет пользователям запускать полнофункциональные приложения одним щелчком мыши из любого веб-браузера.

Web-страничка браузера должна содержать ссылку, указывающую на файл с расширением. JNLP, который включает в себя инструкции для загрузки, кэширования и запуска приложения.



Пользователи могут загружать и запускать приложения, не выполняя сложные процедуры установки.

С помощью Java Web Start пользователи запускают приложения, перейдя по ссылке на веб-странице.



Если приложение отсутствует на компьютере, Java Web Start автоматически загружает все необходимые файлы.

Затем файлы кэшируются на компьютере, чтобы приложение всегда было готово к перезагрузке в любое время, когда пользователь захочет – либо с помощью значка на рабочем столе, либо с помощью ссылки браузера.

Независимо от того, какой метод используется для запуска приложения, всегда отображается самая последняя версия приложения.

Технология, лежащая в основе Java Web Start, – это протокол Java Network Launching Protocol & API (JNLP).

Java Web Start – это эталонная реализация для спецификации JNLP.

Технология JNLP определяет, помимо прочего, файл JNLP, который является стандартным файловым форматом, описывающим запуск приложения.

С технологической точки зрения Java Web Start имеет ряд преимуществ.

Технология Java Web Start построена исключительно для запуска приложений, написанных на платформе Java, Standard Edition.

Таким образом, одно приложение может быть доступно на веб-сервере, а затем развернуто на самых разных платформах, включая Windows, Linux и macOS.

Java Web Start поддерживает несколько версий платформы Java Standard Edition.

Таким образом, приложение может запрашивать определенную версию требуемой платформы, например, Java SE 9.

Несколько приложений могут запускаться одновременно с разными версиями платформ, не вызывая конфликтов.

Java Web Start позволяет запускать приложения независимо от веб-браузера.

Это можно использовать для автономной работы приложения, когда запуск приложения через браузер неудобен или невозможен.

Приложение также можно запускать с помощью ярлыков на рабочем столе, что делает запуск развернутого в Интернете приложения похожим на запуск нативного приложения.

Java Web Start использует функции безопасности платформы Java.

Приложения запускаются в защитной среде или песочнице с ограниченным доступом к локальным дисковым и сетевым ресурсам.

Пользователи также должны давать согласие на запуск приложения при первом запуске.

Приложения, запущенные с помощью Java Web Start, локально кэшируются.

Таким образом, уже загруженное приложение запускается аналогично традиционно установленному приложению.

Java Web Start входит в состав платформы Java Platform, Standard Edition (JDK) и Java Runtime Environment (JRE) и поддерживает функции безопасности платформы Java.

Это означает, что приложение, запускаемое с помощью Java Web Start, должно быть подписано доверенным сертификатом.

Что значительно сужает применение Java Web Start, так как получение доверенного сертификата является платным.

Таким образом, с помощью Java Web Start приложение можно запустить тремя способами:

Из веб-браузера, нажав ссылку.

Из значка на рабочем столе или в меню «Пуск».

Из Java Cache Viewer.

Независимо от того, какой путь используется, Java Web Start будет подключаться к веб-серверу каждый раз при запуске приложения, чтобы проверить, доступна ли обновленная версия приложения.

При запуске из веб-браузера используется HTML-ссылка, которая вместо того, чтобы указывать на другую веб-страницу, ссылается на специальный файл конфигурации, называемый JNLP-файлом.

Веб-браузер проверяет расширение файла или тип MIME файла и видит, что он принадлежит Java Web Start.

Поэтому браузер запускает Java Web Start с загруженным файлом JNLP в качестве аргумента.

Далее уже Java Web Start работает с загрузкой, кэшированием и запуском приложения, как это описано в файле JNLP.

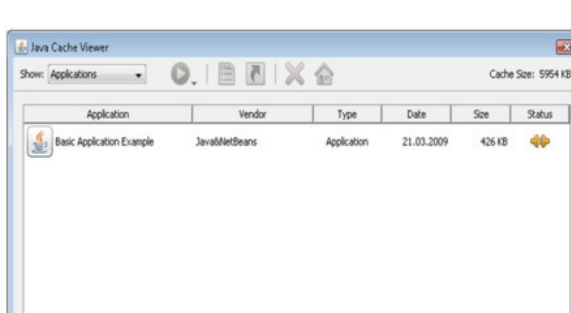
Также технология Java Web Start может автоматически создавать ярлыки для вашего приложения на рабочем столе и в меню «Пуск».

Вы можете использовать панель управления Java для управления настройками ярлыков, которая находиться в панели управления компьютером.

Ярлыки также можно добавить с помощью Java Web Start Cache Viewer.

При первой загрузке приложения с помощью технологии JWS все необходимые файлы сохраняются в компьютере пользователя в специальной папке cache, поэтому повтор-

но запустить приложение можно без использования браузера и соединения с интернетом, с помощью Java Cache Viewer.



Интерфейс Java Cache Viewer позволяет запустить приложение, просмотреть JNLP-файл приложения, инсталлировать ярлык приложения на рабочем столе компьютера, удалить приложение с компьютера и перейти на домашнюю страничку приложения.

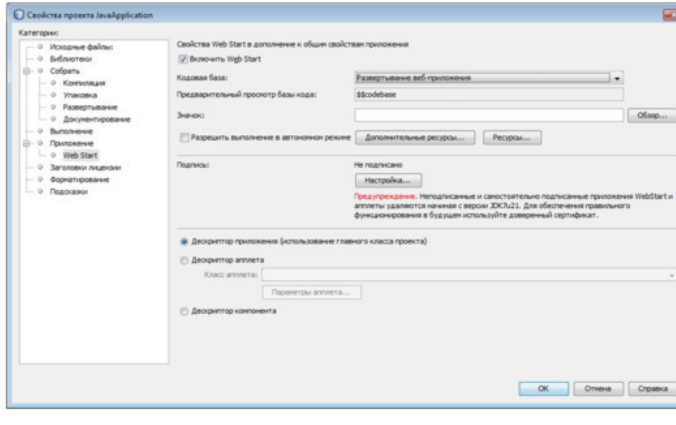
Интерфейс Java Cache Viewer отражает в качестве названия приложения заголовок главного окна приложения, а не имя JAR-файла.

Открыть интерфейс Java Cache Viewer можно с помощью контрольной панели Java, которая находиться в панели управления компьютером.

JNLP-файл представляет собой XML-файл, содержащий инструкции для JWS-инструмента javaws как загружать и запускать приложение.

Именно расширение файла. jnlp, при инсталляции JRE, связывается с Java-инструментом javaws, поэтому при открытии JNLP-файла автоматически запускается загрузчик javaws.

В среде разработки NetBeans включить использование технологии Java Web Start можно в свойствах проекта, при этом в процессе сборки проекта будут созданы все необходимые артефакты.



Зачем же Java уничтожила апплеты?

Современные браузеры работают над ограничением

или уменьшением поддержки плагинов, таких как Flash, Silverlight и Java, и поэтому разработчикам приложений, которые полагались на Java плагин для браузера, необходимо рассмотреть альтернативные варианты.

Они должны рассмотреть возможность перехода с Java-апплетов на технологию Java Web Start без плагинов или просто настольные приложения.

Поддержка Java в браузерах возможна только до тех пор, пока поставщики браузеров будут поддерживать плагин.

К концу 2015 года многие поставщики браузеров либо удалили, либо объявили временные рамки для удаления поддержки Java плагина.

Поэтому, Oracle решила отказаться от Java плагина браузера в JDK 9.

Однако приложения Java Web Start не полагаются на плагин браузера и не будут затронуты этими изменениями.

Архитектура AWT

Графические интерфейсы пользователя Java

Лекция 4
Архитектура AWT

Вернемся к библиотеке AWT.

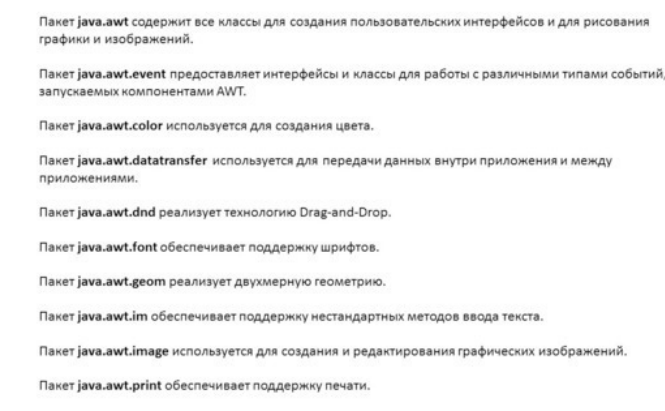
Помимо графических компонентов, библиотека AWT содержит классы и интерфейсы, позволяющие обрабатывать различные типы событий, генерируемые AWT-компонентами.

Суперклассом, представляющим все AWT-события, является класс java.awt.AWTEvent.

Для обработки событий компонента необходимо создать класс-слушатель, реализующий интерфейс java.awt.event.ActionListener, и присоединить его к данному

компоненту.

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



Пакет java.awt.color используется для создания цвета.

Пакет java.awt. datatransfer используется для передачи данных внутри приложения и между приложениями.

Пакет java.awt. dnd реализует технологию Drag-and-Drop.

Пакет java.awt.font обеспечивает поддержку шрифтов.

Пакет java.awt.geom реализует двухмерную геометрию.

Пакет java.awt.im обеспечивает поддержку нестандартных методов ввода текста.

Пакет java.awt.image используется для создания и редактирования графических изображений.

Пакет java.awt.print обеспечивает поддержку печати.

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

Однако беспорядочное использование потоков может замедлять работу AWT-интерфейса.

Суммируя сказанное, можно сказать, что графическая библиотека AWT представляет собой промежуточный уровень между операционной системой и Java-кодом приложения, скрывая все низкоуровневые операции, связанные с построением графического интерфейса пользователя.

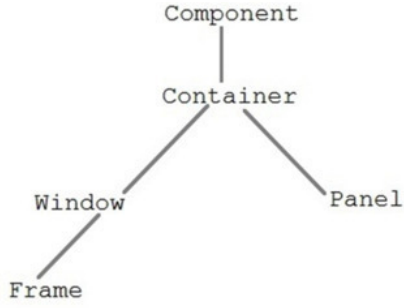
Такое прямое взаимодействие с конкретной операционной системой является и основным недостатком AWT, так как графический интерфейс, созданный на основе AWT, в операционной системе Windows выглядит как Windows-подобный, а в операционной системе Macintosh выглядит как Mac-подобный.

Казалось бы, при наличии таких графических Java-библиотек, как Swing, SWT, Java3D, JavaFX, библиотека AWT должна потерять свою актуальность.

Однако если нет необходимости в широком ассортименте графических компонентов, если требуется работа в основном с двухмерной графикой и изображениями, использование библиотеки AWT удобно.

Кроме того, библиотека AWT является частью платформы Java ME и используется для создания приложений, работающих в устройствах с ограниченными возможностями.

Вернемся к иерархии классов AWT.



Класс Component находится наверху иерархии AWT.

Component – это абстрактный класс, который инкапсулирует все атрибуты визуального компонента.

Объект Component отвечает за запоминание текущих цветов переднего плана и фона, выбранного шрифта текста, а также размеров и местоположения.

Container – это компонент AWT, который содержит другие компоненты, такие как кнопка, текстовое поле, таблицы и т. д.

Контейнер является подклассом класса компонентов.

Класс контейнера отслеживает и компонует добавляемые компоненты.

Класс Panel – это конкретный подкласс класса Container.

Панель не содержит строку заголовка, строку меню или границу.

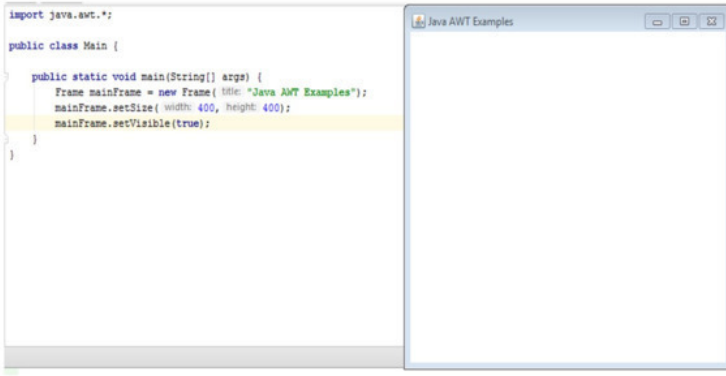
Это контейнер, который используется для содержания компонентов.

Класс Window создает окно верхнего уровня. Окно не имеет границ и меню.

Frame является подклассом класса Window и имеет заголовок и границы, а также изменяемый пользователем размер.

Таким образом, для создания AWT приложения, в первую очередь, нужно создать объект Frame как окно верхнего уровня.

Здесь мы создаем объект Frame, устанавливаем его размеры и делаем его видимым.



В результате получаем окно с заголовком и кнопками минимизации и закрытия окна.

Однако закрыть такое окно мы не сможем.

Для этого мы должны добавить слушателя событий окна.

```
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class Main {

  public static void main(String[] args) {
    Frame mainFrame = new Frame("Java AWT Examples");
    mainFrame.setSize(400,400);

    mainFrame.addWindowListener(new WindowAdapter(){
      public void windowClosing(WindowEvent windowEvent){
        System.exit(0);
      }
    });

    mainFrame.setVisible(true);
  }
}
```

Абстрактный класс WindowAdapter реализует интерфейсы слушателя событий окна и предназначен для получения событий окна.

Метод windowClosing вызывается при нажатии кнопки закрытия окна.

И мы определяем этот метод, вызывая в нем метод exit системы.

Таким образом, теперь мы можем закрыть это окно.

Так как Frame – это контейнер, мы можем добавлять в него меню и другие элементы графического интерфейса пользователя, кнопки, метки, поля и т. д.

Однако, чтобы упростить компоновку Frame окна, вы можете разбить окно на регионы и собирать каждый из них отдельно.

```
public class Main {

  public static void main(String[] args) {
    Frame mainFrame = new Frame("Java AWT Examples");
    mainFrame.setSize(400,500);
    mainFrame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent windowEvent){
        System.exit(0);
      }
    });
    FlowLayout layout = new FlowLayout();
    layout.setHgap(10);
    layout.setVgap(10);
    mainFrame.setLayout(layout);

    Panel panel1=new Panel();
    panel1.setLayout(layout);
    panel1.setPreferredSize(new Dimension(200,200));
    panel1.setBackground(Color.gray);
    Button b1=new Button("Button 1");
    b1.setBackground(Color.yellow);
    panel1.add(b1);

    Panel panel2=new Panel();
    panel2.setLayout(layout);
    panel2.setPreferredSize(new Dimension(200,200));
    panel2.setBackground(Color.pink);
    Button b2=new Button("Button 2");
    b2.setBackground(Color.green);
    panel2.add(b2);

    mainFrame.add(panel1);
    mainFrame.add(panel2);
    mainFrame.setVisible(true);
  }
}
```

Каждый регион называется панелью.

Окно Frame и каждая панель могут иметь свою собственную компоновку LayoutManager.

Панели не имеют видимых ограничивающих линий.

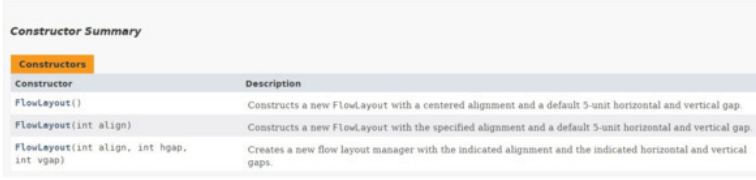
Вы можете разграничить их разными цветами фона.

Метод setLayout класса Container устанавливает компоновку LayoutManager, которая отвечает за расположение элементов контейнера.

Существует пять стандартных AWT компоновок – классов, реализующих интерфейс LayoutManager, это FlowLayout, BorderLayout, CardLayout, GridLayout, и GridBagLayout.

Компоновка FlowLayout является компоновкой по умол-

чанию для панели, поэтому мы могли бы ее не устанавливать для панели методом setLayout.



Constructor Summary

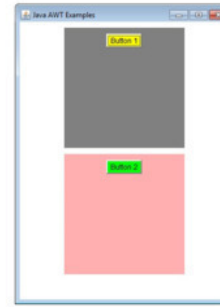
Constructors

Constructor	Description
`FlowLayout()`	Constructs a new FlowLayout with a centered alignment and a default 5-unit horizontal and vertical gap.
`FlowLayout(int align)`	Constructs a new FlowLayout with the specified alignment and a default 5-unit horizontal and vertical gap.
`FlowLayout(int align, int hgap, int vgap)`	Creates a new flow layout manager with the indicated alignment and the indicated horizontal and vertical gaps.

Эта компоновка изменяет размеры компонентов контейнера до их предпочтительного размера, поэтому, чтобы задать размер компонента, нужно использовать метод setPreferredSize.

Метод setSize работать не будет.

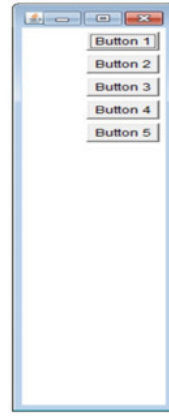
Эта компоновка помещает компоненты в строки слева направо, сверху вниз, обеспечивая по умолчанию 5 пикселей между элементами в строке и между строками.



Компоненты в строках по умолчанию находятся в центре.

Выравнивание компонента в строке можно изменить с помощью константы FlowLayout. LEFT или FlowLayout. RIGHT в конструкторе класса FlowLayout.

```
FlowLayout layout = new FlowLayout(FlowLayout.RIGHT);

    mainFrame.setLayout(layout);
   for(int i=1; i<6; i++) {
      mainFrame.add(new Button("Button " + i));
   }

    mainFrame.setVisible(true);
```



Компоновка BorderLayout является компоновкой по умолчанию для окон Frame и Dialog, поэтому мы можем ее не устанавливать для окна методом setLayout.

Class BorderLayout

java.lang.Object
    java.awt.BorderLayout

All Implemented Interfaces:

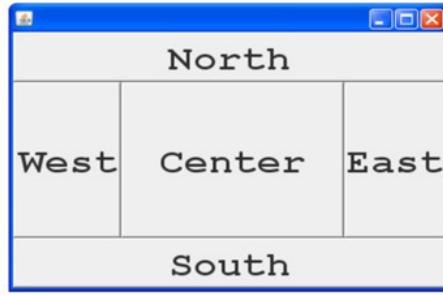
LayoutManager, LayoutManager2, Serializable

```
public class BorderLayout
extends Object
implements LayoutManager2, Serializable
```

A border layout lays out a container, arranging and resizing its components to fit in five regions: north, south, east, west, and center. Each region may contain no more than one component, and is identified by a corresponding constant: NORTH, SOUTH, EAST, WEST, and CENTER. When adding a component to a container with a border layout, use one of these five constants, for example:

```
    Panel p = new Panel();
    p.setLayout(new BorderLayout());
    p.add(new Button("Okay"), BorderLayout.SOUTH);
```

Эта компоновка разделяет контейнер на пять регионов.



Каждый регион идентифицируется соответствующей константой BorderLayout – NORTH, SOUTH, EAST, WEST, и CENTER

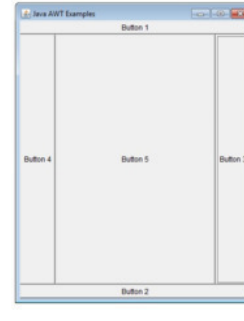
NORTH и SOUTH изменяют размер компонента до его предпочтительной высоты.

EAST и WEST изменяют размер компонента до его предпочтительной ширины.

Центру предоставляется оставшееся пространство.

При добавлении компонента в контейнер, указывается регион, куда добавлять компонент.

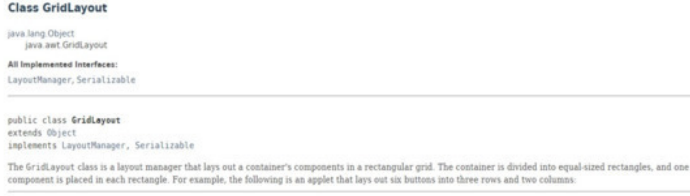
```
mainFrame.add(new Button("Button 1"), BorderLayout.NORTH);
mainFrame.add(new Button("Button 2"), BorderLayout.SOUTH);
mainFrame.add(new Button("Button 3"), BorderLayout.EAST);
mainFrame.add(new Button("Button 4"), BorderLayout.WEST);
mainFrame.add(new Button("Button 5"), BorderLayout.CENTER);
mainFrame.setVisible(true);
```



По умолчанию, компонент будет добавляться в центр.

Компоновка GridLayout разделяет окно на прямоугольники равного размера на основе количества указанных строк

и столбцов.



Class GridLayout

java.lang.Object
  java.awt.GridLayout

All Implemented Interfaces:

LayoutManager, Serializable

public class **GridLayout**
extends Object
implements LayoutManager, Serializable

The GridLayout class is a layout manager that lays out a container's components in a rectangular grid. The container is divided into equal-sized rectangles, and one component is placed in each rectangle. For example, the following is an applet that lays out six buttons into three rows and two columns:

Элементы размещаются в ячейки слева направо, сверху вниз.

Эта компоновка игнорирует предпочтительный размер компонента, и каждый компонент изменяется в соответствии с его ячейкой.

Слишком мало компонентов приводит к пустым ячейкам, и слишком много компонентов приводит к дополнительным столбцам.

При создании компоновки указывается количество строк и столбцов.

```
mainFrame.setLayout(new GridLayout(2,3)); // 2 rows, 3 cols
mainFrame.add(new Button("Button One"));
mainFrame.add(new Button("Button Two"));
mainFrame.add(new Button("Button Three"));
mainFrame.add(new Button("Button Four"));
mainFrame.add(new Button("Button Five"));
mainFrame.add(new Button("Button Six"));
mainFrame.setVisible(true);
```

По умолчанию создается одна строка с одним столбцом.

Компоновка устанавливается для контейнера методом setLayout и затем компоненты добавляются в контейнер методом add.

Компоновка CardLayout работает как стек, помещая компоненты друг поверх друга, и связывает имя с каждым компонентом в окне.

Class CardLayout

java.lang.Object
    java.awt.CardLayout

All Implemented Interfaces:

LayoutManager, LayoutManager2, Serializable

```
public class CardLayout
extends Object
implements LayoutManager2, Serializable
```

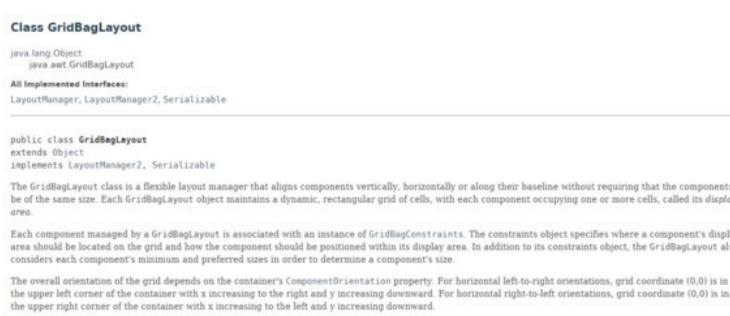
A CardLayout object is a layout manager for a container. It treats each component in the container as a card. Only one card is visible at a time, and the container acts as a stack of cards. The first component added to a CardLayout object is the visible component when the container is first displayed.

The ordering of cards is determined by the container's own internal ordering of its component objects. CardLayout defines a set of methods that allow an application to flip through these cards sequentially, or to show a specified card. The addLayoutComponent(java.awt.Component, java.lang.Object) method can be used to associate a string identifier with a given card for fast random access.

Эта компоновка хорошо подходит для размещения единственного компонента в окне.

```
CardLayout layout = new CardLayout();
mainFrame.setLayout(layout);

Panel panel = new Panel();
panel.setLayout(new GridLayout(2,3)); // 2 rows, 3 cols
panel.add(new Button("Button One"));
panel.add(new Button("Button Two"));
panel.add(new Button("Button Three"));
panel.add(new Button("Button Four"));
panel.add(new Button("Button Five"));
panel.add(new Button("Button Six"));

mainFrame.add("Main Panel",panel);
mainFrame.setVisible(true);
```

Компоновка GridBagLayout разделяет окно на ячейки сетки, не требуя, чтобы компоненты были одного размера.



Class GridBagLayout

java.lang.Object
  java.awt.GridBagLayout

All Implemented Interfaces:
LayoutManager, LayoutManager2, Serializable

public class **GridBagLayout**
extends Object
implements LayoutManager2, Serializable

The GridBagLayout class is a flexible layout manager that aligns components vertically, horizontally or along their baseline without requiring that the components be of the same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells, with each component occupying one or more cells, called its *display area*.

Each component managed by a GridBagLayout is associated with an instance of GridBagConstraints. The constraints object specifies where a component's display area should be located on the grid and how the component should be positioned within its display area. In addition to its constraints object, the GridBagLayout also considers each component's minimum and preferred sizes in order to determine a component's size.

The overall orientation of the grid depends on the container's ComponentOrientation property. For horizontal left-to-right orientations, grid coordinate (0,0) is in the upper left corner of the container with x increasing to the right and y increasing downward. For horizontal right-to-left orientations, grid coordinate (0,0) is in the upper right corner of the container with x increasing to the left and y increasing downward.

При этом каждый компонент, управляемый компоновкой, ассоциируется с экземпляром GridBagConstraints.

GridBagConstraints указывает как компонент располагается в области отображения, в какой ячейке начинается и заканчивается компонент, как компонент растягивается, когда имеется дополнительная ячейка, а также определяет выравнивание в ячейке.

Для использования этой компоновки сначала создается экземпляр GridBagLayout, который устанавливается для контейнера методом setLayout.

```
Panel panel = new Panel();
GridBagLayout layout = new GridBagLayout();
panel.setLayout(layout);
GridBagConstraints constraints = new GridBagConstraints();
GridBagConstraints gbc = new GridBagConstraints();

gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.gridx = 0;
gbc.gridy = 0;
panel.add(new Button("Button 1"),gbc);

gbc.gridx = 1;
gbc.gridy = 0;
panel.add(new Button("Button 2"),gbc);

gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.ipady = 20;
gbc.gridx = 0;
gbc.gridy = 1;
panel.add(new Button("Button 3"),gbc);

gbc.gridx = 1;
gbc.gridy = 1;
panel.add(new Button("Button 4"),gbc);

gbc.gridx = 0;
gbc.gridy = 2;
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.gridwidth = 2;
panel.add(new Button("Button 5"),gbc);

mainFrame.add(panel);

mainFrame.pack();
mainFrame.setVisible(true);
```

Затем создается экземпляр GridBagConstraints, параметры которого изменяются для каждого компонента.

gridx и gridy указывают номер ячейки для компонента.

gridwidth и gridheight указывают количество столбцов и строк, которые компонент занимает.

fill указывает что делать с компонентом, который меньше, чем размер ячейки.

ipady и ipadx указывают отступ.

Обратите внимание, что в конце мы используем метод pack для окна, который подгоняет размер окна под его содержимое.

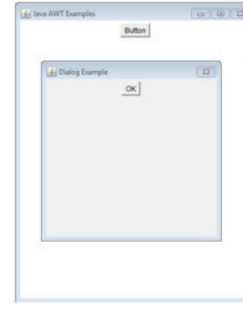
Добавлять компоненты в окно можно и без менеджера компоновки, установив его в нуль.

```
mainFrame.setLayout(null);

Button b1 = new Button("Button 1");
Button b2 = new Button("Button 2");
b1.setBounds(10, 100, 150, 50);
b2.setBounds(250, 100, 75, 50);
mainFrame.add(b1);
mainFrame.add(b2);

mainFrame.setVisible(true);
```

Для этого нужно точно указывать координаты компонента и его размеры, например, с помощью метода setBounds.

До сих пор мы рассматривали окно Frame.

```
final Dialog d = new Dialog(mainFrame , "Dialog Example", true);
d.setLayout( new FlowLayout() );
Button bD = new Button ("OK");
bD.addActionListener ( new ActionListener()
{
   public void actionPerformed( ActionEvent e )
   {
      d.setVisible(false);
   }
});
d.add(bD);
d.setSize(300,300);
Button b = new Button("Button");
b.addActionListener(new ActionListener()
{
   public void actionPerformed( ActionEvent e )
   {
      d.setVisible(true);
   }
});
mainFrame.setLayout( new FlowLayout() );
mainFrame.add(b);
mainFrame.setVisible(true);
```



Однако помимо окна Frame, библиотека AWT позволяет создавать диалоговые окна с помощью класса Dialog.

Для создания диалогового окна, которое открывается из окна Frame, нужно создать экземпляр класса Dialog, указав в его конструкторе родительское окно, из которого будет открываться данное диалоговое окно.

Также можно указать заголовок окна и будет ли окно модальным, то есть будет ли оно блокировать все входные данные для окон верхнего уровня.

Далее можно наполнить диалоговое окно содержимым, напомним, что по умолчанию компоновка диалогового окна – это BorderLayout.

Для закрытия окна, к его компоненту нужно присоединить слушателя, в обработчике которого нужно вызвать ме-

тод setVisible (false) окна, сделав его невидимым.

Далее в основном окне, к его компоненту нужно присоединить слушателя, в обработчике которого нужно вызвать метод setVisible (true) диалогового окна, сделав его видимым.

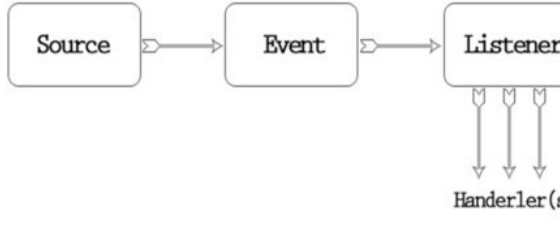
И наконец, методом add нужно добавить диалоговое окно в основное окно, как обычный компонент.

Модель событий AWT

Графические интерфейсы пользователя Java

Лекция 5
Модель событий AWT

Мы уже несколько раз сталкивались со слушателями событий и обработчиками событий.



Event-Driving Programming Model

В отличие от процедурных программ, для программ с графическим интерфейсом пользователя требуется управляемая событиями модель, в которой низлежащее окружение сообщает вашей программе, когда что-то происходит.

Например, когда пользователь нажимает на мышь, низлежащая среда выполнения генерирует событие, которое оно отправляет в программу.

Затем программа должна выяснить, что означает щелчок мыши и действовать соответствующим образом.

Процедурное программирование, которое рассматривается как традиционное программирование, определяет процесс программирования как разработку процедур, которые прямо управляют потоком данных и элементами контроля.

Программирование, управляемое событиями, определяет

процесс программирования как разработку процедур, которые реагируют на поток данных и элементы контроля в соответствии с действиями пользователя, программы или операционной системы.

Эти модели программирования отличаются потоком выполнения и структурой.

В AWT существуют две разные модели событий или способы обработки событий.

В Java 1.0 и ранее события отправлялись непосредственно соответствующим компонентам.

Сами события были инкапсулированы в один класс Event.

Для обработки таких событий нужно было переопределить метод action или handleEvent компонента, который вызывался при получении события компонентом.

Таким образом, в этой ранней модели обработчики событий, такие как action и handleEvent, были реализованы классом Component.

И версии этих методов по умолчанию ничего не делали и возвращали false.

Эта модель событий была заменена моделью Java 1.1.

Java 1.1 реализует модель делегирования, в которой события передаются только объектам, зарегистрированным для получения события.

Эта модель позволяет любому объекту получать события, генерируемые компонентом.

И это означает, что вы можете отделить сам пользова-

тельский интерфейс от кода обработки событий, в отличие от ранней модели, где вы должны были для обработки события переопределять метод самого компонента пользовательского интерфейса.

В модели событий Java 1.1 вся функциональность обработки событий содержится в пакете java.awt. event.

Внутри этого пакета подклассы абстрактного класса AWTEvent представляют собой различные виды событий.

Класс AWTEvent и его подклассы заменяют Event предыдущей модели событий.

Пакет также включает в себя ряд интерфейсов EventListener, которые реализуются классами, которые хотят получать различные виды событий.

Эти интерфейсы определяют методы, вызываемые при возникновении событий соответствующего типа.

Пакет также содержит ряд классов адаптеров.

Они реализуют интерфейсы EventListener и являются абстрактными классами, которые предоставляют нулевые реализации методов соответствующего прослушивателя.

Эти классы удобны для создания объектов-слушателей.

Вместо того, чтобы объявлять, что ваш класс реализует определенный интерфейс EventListener, вы можете объявить, что ваш класс расширяет соответствующий адаптер.

Эта модель требует, чтобы объекты регистрировались для получения событий.

Затем, когда происходит событие, об этом уведомляются

только те объекты, которые были зарегистрированы.

Эта модель называется «делегирование».

Она реализует шаблон проектирования Observer.

Таким образом, эта модель обеспечивает четкое разделение между компонентами GUI и обработкой событий.

Важно, чтобы любой объект, а не только компонент, мог получать события.

Таким образом, вы можете отделить свой код обработки событий от вашего графического интерфейса.

Один набор классов может реализовать пользовательский интерфейс, а другой набор классов может реагировать на события, генерируемые интерфейсом.

Это означает, что, если вы разработали хороший интерфейс, вы можете повторно использовать его в разных приложениях, изменив обработку событий.

Модель делегирования важна для JavaBeans, которые позволяют взаимодействовать между Java и другими платформами.

Чтобы обеспечить такое взаимодействие, необходимо было отделить источник события от получателя.

В модели делегирования события могут передаваться нескольким получателям; любое количество классов может быть зарегистрировано для получения события.

В старой модели обработчик событий мог заявить, что он полностью не обработал событие, передавая событие контейнеру, или обработчик событий мог сгенерировать новое со-

бытие и доставить его другому компоненту.

В любом случае разработчику приходилось планировать, как передавать события другим получателям.

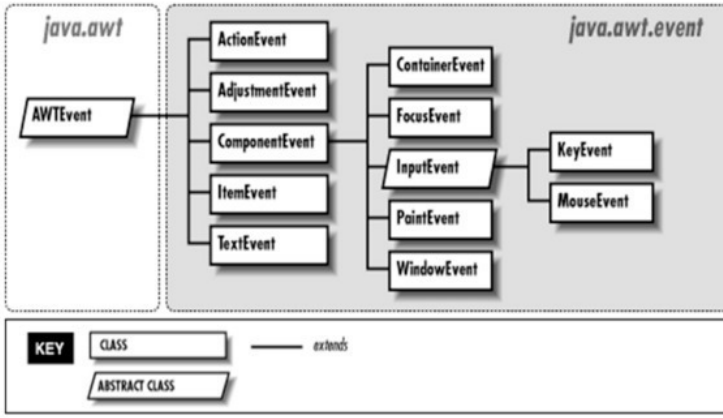
В Java 1.1 это не требуется.

Событие будет передано каждому объекту, зарегистрированному в качестве слушателя для этого события, независимо от того, что другие объекты делают с событием.

Наконец, эта модель событий представляет идею очереди событий.

Вместо того, чтобы переопределять handleEvent для просмотра всех событий, вы можете заглянуть в очередь событий системы, используя класс EventQueue.

В Java 1.1 каждый компонент является источником события, который может генерировать определенные типы событий, которые являются подклассами класса AWTEvent.



Объекты, которые интересуются событием, называются слушателями.

Каждый тип события соответствует интерфейсу прослушивателя, который определяет методы, вызываемые при возникновении события.

Чтобы получить событие, объект должен реализовать соответствующий интерфейс слушателя и должен быть зарегистрирован в источнике события, путем вызова метода «add listener» компонента, который генерирует событие.

И мы это уже видели.

```
Button b = new Button("Button");
 b.addActionListener(new ActionListener()
 {
    public void actionPerformed( ActionEvent e )
    {
                ...
    }
 });
```

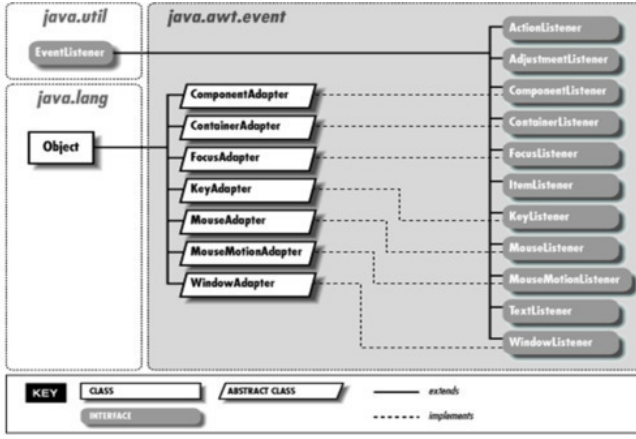
Здесь мы создаем кнопку и присоединяем к ней слушателя, как экземпляр анонимного класса, который реализует интерфейс слушателя.

В этом классе мы переопределяем обработчик событий, метод интерфейса слушателя.

Как только объект зарегистрирован, метод actionPerformed будет вызываться всякий раз, когда пользователь делает что-либо в компоненте, который генерирует событие действия.

В некотором роде метод actionPerformed очень похож на метод action старой модели событий, за исключением того, что он не привязан к иерархии Component, а он является частью интерфейса, который может быть реализован любым объектом, который заинтересован в получении событий.

Вместо реализации интерфейсов, можно расширять классы адаптеров, которые реализуют интерфейсы слушателей, переопределяя абстрактные методы адаптеров.



Некоторые интерфейсы слушателей предназначены для работы с несколькими типами событий.

Например, интерфейс MouseListener объявляет пять методов обработки различных типов событий мыши: мышь вниз, мышь вверх, щелчок, вход мыши в компонент и выход мыши.

Строго говоря, это означает, что объект, интересующийся событиями мыши, должен реализовывать MouseListener и поэтому должен переопределять все пять методов всех возможных действий мыши.

Это звучит как создание излишнего кода; большую часть времени вас интересует только одно или два из этих событий.

К счастью, вам этого делать не нужно.

Вы можете использовать класс адаптера, который обеспечивает нулевую реализацию всех методов интерфейса.

И если вы хотите написать класс обработки событий, который имеет дело только с щелчками мыши, вы можете объявить, что ваш класс расширяет MouseAdapter.

И ваша единственная задача программирования состоит в том, чтобы переопределить единственный метод, который вам нужен – это mouseClicked.

Таким образом, резюмируя.

Компоненты генерируют AWTEvents, когда что-то происходит.

Различные подклассы AWTEvent представляют различные типы событий.

Например, события мыши представлены классом MouseEvent.

И каждый компонент может генерировать определенные подклассы класса AWTEvent.

Обработчики событий регистрируются для приема событий с помощью метода «add listener» в компоненте, который генерирует событие.

Существуют различные методы «add listener» для каждого вида событий AWTEvent, которые может генерировать ком-

понент.

Например, чтобы заявить о своем интересе к событию мыши, вы вызываете метод addMouseListener компонента.

Каждый тип события имеет соответствующий интерфейс прослушивателя, который определяет методы, вызываемые при возникновении этого события.

Чтобы иметь возможность принимать события, обработчик событий должен реализовать соответствующий интерфейс прослушивателя.

Например, MouseListener определяет методы, вызываемые при возникновении событий мыши.

Большинство типов событий также имеют класс адаптера.

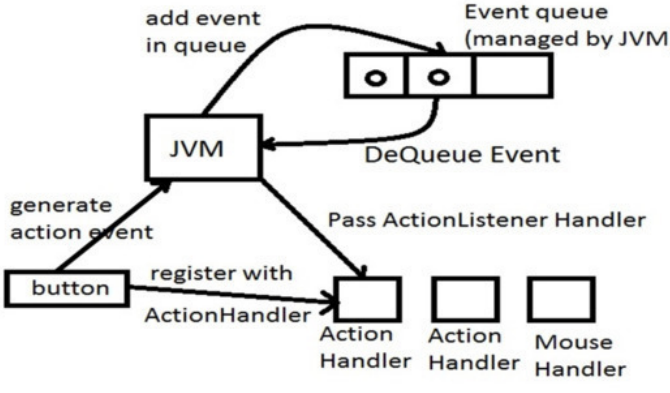
Например, события MouseEvent имеют класс MouseAdapter.

Класс адаптера реализует соответствующий интерфейс прослушивателя, но обеспечивает реализацию заглушки каждого метода, т. е. метод просто возвращает без каких-либо действий.

Классы адаптеров используются, когда нужны только некоторые из методов в интерфейсе слушателя.

Например, вместо реализации всех пяти методов интерфейса MouseListener класс может расширить класс MouseAdapter и переопределить один или два метода, которые нужны.

Класс EventQueue позволяет напрямую управлять событиями Java 1.1.



Обычно вам не нужно самостоятельно управлять событиями; система сама заботится о доставке событий.

Однако, если вам нужно, вы можете получить очередь событий системы, вызвав Toolkit.getSystemEventQueue, затем вы можете заглянуть в очередь событий, вызвав peekEvent или опубликовать новые события, вызвав postEvent.

EventQueue – платформо-независимый класс, представляющий собой очередь событий, получаемых как из классов-помощников peer, которые организуют взаимодействие классов AWT с операционной системой, так и из классов приложения.

Данный класс обеспечивает диспетчеризацию событий, т.е. извлечение событий из очереди и отправки их вызо-

вом внутреннего метода dispatchEvent (AWTEvent event), который в качестве параметра принимает объект класса AWTEvent, представляющий собой AWT события.

Таким образом, класс EventQueue обеспечивает последовательную обработку событий в порядке очереди.

Метод dispatchEvent класса EventQueue определяет, к какому графическому компоненту относится данное событие и производит вызов метода dispatchEvent соответствующего компонента.

Метод dispatchEvent наследуется каждым компонентом от базового класса java.awt.Component.

Далее событие передается из метода dispatchEvent методу processEvent (AWTEvent e), который в свою очередь передает событие методу process <event type> Event, определенному для каждого класса событий.

После этого метод process <event type> Event передает событие объекту интерфейса <event type> Listener, зарегистрированному соответствующим слушателем add <event type> Listener, где событие и обрабатывается методом, определенном в интерфейсе.

Объект класса EventQueue автоматически создается виртуальной машиной Java, когда приложением создается объект – наследник класса java.awt.Component.

При этом автоматически создается также объект класса java.awt. EventDispatchThread, который представляет собой поток, работающий параллельно основному потоку програм-

мы.

Данный поток собственно и осуществляет диспетчеризацию событий, хранящихся в очереди.

Методом pumpEvents класса EventDispatchThread события извлекаются из очереди и передаются методу dispatchEvent класса EventQueue.

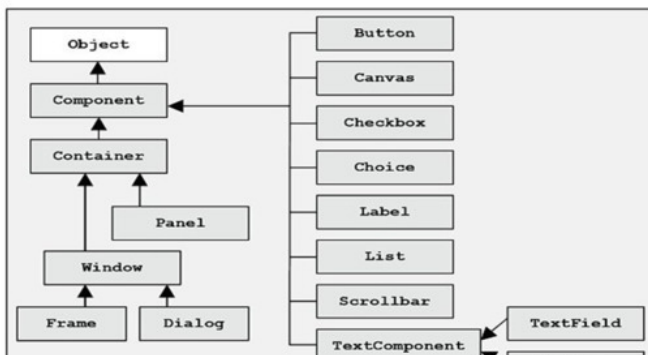
Таким вот образом события передаются от источника слушателю.

Компоненты управления AWT

Графические интерфейсы пользователя Java

Лекция 6
Компоненты управления AWT

Каждый пользовательский интерфейс состоит из трех основных аспектов.



Это элементы пользовательского интерфейса.

Они являются основными визуальными элементами, которые пользователь в конечном итоге видит и взаимодействует с ними.

Макеты или компоновки.

Они определяют, как элементы пользовательского интерфейса должны быть организованы на экране и обеспечивать окончательный внешний вид графического интерфейса пользователя.

И поведение.

Это события, которые происходят, когда пользователь взаимодействует с элементами пользовательского интерфейса.

Элементы управления, с помощью которых создает-

ся AWT графический интерфейс, наследуют от класса Component.

Класс Button представляет кнопку, элемент управления, который имеет метку и генерирует событие при нажатии.

```
Button submitButton = new Button("Submit");

submitButton.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {

      statusLabel.setText("Submit Button clicked.");

    }
});
```

Когда кнопка нажата и отпущена, AWT отправляет экземпляр ActionEvent события к кнопке, вызывая метод processEvent кнопки.

Метод processEvent кнопки получает все события для кнопки, и он передает событие, вызывая собственный метод processActionEvent.

Этот метод передает событие любому слушателю, который зарегистрировал свой интерес к событиям, сгенерированным этой кнопкой.

Если приложение хочет выполнить какое-либо действие на основе нажатия и отпускания кнопки, оно должно реализовать интерфейс ActionListener и зарегистрировать нового слушателя для приема событий этой кнопки, с помощью метода addActionListener кнопки.

Элемент Checkbox используется для включения опции (true) или ее выключения (false).

```
final Label statusLabel = new Label();

Checkbox chkApple = new Checkbox("Apple");
chkApple.setState(true);

    chkApple.addItemListener(new ItemListener() {
      public void itemStateChanged(ItemEvent e) {
        statusLabel.setText("Apple Checkbox: "
        + (e.getStateChange()==1?"checked":"unchecked"));
      }
    });
```

Для каждого флажка есть метка, обозначающая, что делает флажок.

И состояние флажка можно изменить, щелкнув по нему.

Объект флажка создается с помощью конструктора, которому передается метка флажка.

Состояние флажка, выбран он или нет, устанавливается

с помощью метода setState, или сразу указав в конструкторе состояние флажка.

Здесь используется слушатель ItemListener, а не ActionListener.

Он слушает изменение состояния компонента, а не действия, предоставляя метод itemStateChanged.

Превратить флажок в радио кнопку, можно создав группу флажков.

```
CheckboxGroup fruitGroup = new CheckboxGroup();

Checkbox chkApple = new Checkbox("Apple", fruitGroup, true);
Checkbox chkMango = new Checkbox("Mango", fruitGroup, false);
Checkbox chkPeer = new Checkbox("Peer", fruitGroup, false);

statusLabel.setText("Apple Checkbox: checked");
chkApple.addItemListener(new ItemListener() {
  public void itemStateChanged(ItemEvent e) {
    statusLabel.setText("Apple Checkbox: checked");
  }
});

chkMango.addItemListener(new ItemListener() {
  public void itemStateChanged(ItemEvent e) {
    statusLabel.setText("Mango Checkbox: checked");
  }
});

chkPeer.addItemListener(new ItemListener() {
  public void itemStateChanged(ItemEvent e) {
    statusLabel.setText("Peer Checkbox: checked");
  }
});
```


При создании каждого флажка, эта группа передается в конструктор, тем самым добавляя флажок в группу флажков.

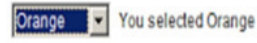
Радио кнопка отличается от флажка тем, что одновременно может быть выбрана только одна радио кнопка, а флажки

могут быть отмечены сразу несколько.

Класс CheckboxGroup имеет метод getSelectedCheckbox, который возвращает выбранную радио кнопку.

Компонент выбора Choice используется для отображения всплывающего меню выбора.

```
final Choice c = new Choice();
final Label l = new Label();

  // Add items
  c.add("Apple");
  c.add("Mango");
  c.add("Guava");
  c.add("Orange");

// Add item listener
  c.addItemListener(new ItemListener(){
    public void itemStateChanged(ItemEvent ie)
    {
    l.setText("You selected "+c.getSelectedItem());
    }
  });
```



Выбранный элемент отображается в верхней части меню.

Класс Choice имеет метод getSelectedItem, который возвращает выбранный элемент в виде строки.

Label – это пассивный элемент управления, так как он не создает никакого события пользователя.

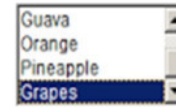
```
Label label = new Label();
label.setText("Welcome to AWT Tutorial.");
label.setAlignment(Label.CENTER);
label.setBackground(Color.GRAY);
label.setForeground(Color.WHITE);
```

Метка просто отображает одну строку текста, доступную только для чтения.

Текст метки может быть изменен программным способом, но никак не может быть изменен конечным пользователем.

Список List представляет собой список текстовых элементов.

```
final List list = new List(4, true);
final Label label = new Label();

  list.add("Apple");
  list.add("Mango");
  list.add("Guava");
  list.add("Orange");
  list.add("Pineapple");
  list.add("Grapes");

list.addItemListener(new ItemListener(){
     public void itemStateChanged(ItemEvent ie)
     {
     label.setText("You selected "+list.getSelectedItem());
     }
  });
```



You selected Grapes

Список может быть настроен так, что пользователь может выбрать один или несколько элементов, с помощью второго аргумента конструктора.

В этом отличие списка от выбора Choice.

Также List – это статический список, а не выпадающий список выбора, как Choice.

Компоненты List, TextArea и ScrollPane поставляются с готовыми полосами прокрутки.

```
class Scroll extends Panel implements AdjustmentListener
    {
      private Scrollbar hsb, vsb;
      private  int hr, vr;
       private Dimension preferredDimension;
       private int width, height;

public Dimension getPreferredSize() {
          return(preferredDimension);
        }

public Dimension getMinimumSize() {
          return(preferredDimension);
        }

public void setCenter(int x, int y) {
          setLocation(x - width/2, y - height/2);
        }
```

```
  public Scroll (int w, int h)
{
  this.setLayout(new FlowLayout(FlowLayout.RIGHT));
  width=w;
  height=h;
  preferredDimension = new Dimension(w, h);
  setSize(preferredDimension);
  hsb=new Scrollbar (Scrollbar.HORIZONTAL);
  hsb.setMaximum (100);
  hsb.setPreferredSize(new Dimension(width,10));
  hsb.setMinimum (10);
  vsb=new Scrollbar (Scrollbar.VERTICAL);
  vsb.setMaximum (100);
  vsb.setPreferredSize(new Dimension(10,height));
  vsb.setMinimum (10);
  add (hsb);
  add (vsb);
  hsb.addAdjustmentListener (this);
  vsb.addAdjustmentListener (this);
  hr=hsb.getValue();
  vr=vsb.getValue();
  setVisible (true);
}
```

Однако, если вы хотите прокрутить любой другой объект, вам придется использовать полосу прокрутки Scrollbar.

Полосы прокрутки в большинстве случаев используются для перемещения видимой области.

Они также могут использоваться для установки значения между двумя числами.

Или они могут использоваться для пролистывания нескольких экранов.

В этом примере мы создаем пользовательский компонент, который расширяет панель и реализует интерфейс AdjustmentListener, для получения событий прокрутки Scrollbar.

При создании пользовательского компонента мы должны расширить класс Component или один из его подклассов.

Также мы должны определить методы расчета размеров компонента и метод paint отрисовки компонента.

В конструкторе этого класса мы создаем горизонтальную и вертикальную полосы прокрутки.

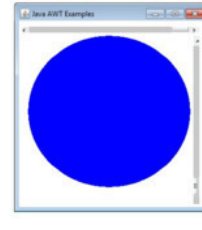
Устанавливаем их размеры на основе размеров пользовательского компонента.

И добавляем эти полосы прокрутки в пользовательский компонент.

Также мы определяем, что данный пользовательский компонент является слушателем событий прокрутки полос.

Полоса прокрутки генерирует событие настройки, когда изменяется значение полосы прокрутки.

```
public void adjustmentValueChanged (AdjustmentEvent ae)
        {
          hr=hsb.getValue();
          vr=vsb.getValue();
          repaint ();
        }


public void paint (Graphics g)
        {
          g.setColor (Color.BLUE);
          g.fillOval (10, 20, width*hr/100, height*vr/100);

        }
      }
```



Для обработки этого события определяется метод

adjustValueChanged интерфейса AdjustmentListener.

В этом методе мы получаем значения полос прокрутки и перерисовываем наш компонент.

В методе paint мы рисуем круг с диаметром, на основе значений полос прокрутки.

Таким образом, перемещая ползунки полос прокрутки, мы изменяем диаметр круга.

Элемент управления TextArea в AWT предоставляет многострочную область редактора.

```
TextArea commentTextArea = new TextArea("This is a AWT tutorial "
    +"to make GUI application in Java.", 5 ,30);
```

Пользователь может вводить здесь столько, сколько он хочет.

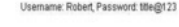
Когда текст в текстовой области становится больше, чем область просмотра, автоматически появляется полоса про-

крутки, которая позволяет прокручивать текст вверх, вниз, вправо и влево.

При создании компонента TextArea указывается количество строк и столбцов видимой области текста.

Компонент TextField позволяет пользователю редактировать одну строку текста.

```
final TextField userText = new TextField(6);
final TextField passwordText = new TextField(6);
passwordText.setEchoChar('*');

passwordText.addActionListener(new ActionListener() {

  public void actionPerformed(ActionEvent e) {
    String data = "Username: " + userText.getText();
    data += ", Password: " + passwordText.getText();
    statusLabel.setText(data);

  }
});
```



Когда пользователь вводит символ в текстовое поле, в поле отправляется событие нажатия клавиши.

Событие нажатия клавиши передается зарегистрированному слушателю KeyListener.

Также можно обрабатывать событие ActionEvent, которое запускается нажатием клавиши enter.

В этом примере, после ввода пароля и нажатия клавиши

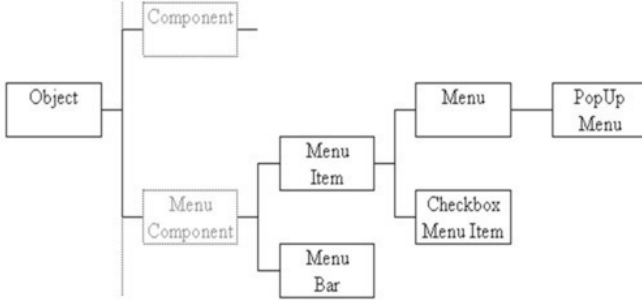
enter, текст одного и другого поля будут выведены в метку.

При создании поля можно указать количество видимых символов.

Метод setEchoCharacter указывает символ, который будет отображаться вместо символов, вводимых пользователем.

Этот метод используется для ввода паролей.

Обычно, окно верхнего уровня имеет связанную с ним панель меню.



Эта панель меню состоит из различных вариантов меню, доступных конечному пользователю.

Для создания меню, пакет java.awt предоставляет четыре основных класса – MenuBar, Menu, MenuItem и CheckboxMenuItem.

Все эти четыре класса не являются компонентами AWT, поскольку они не являются подклассами класса Component.

На самом деле, они являются подклассами класса MenuComponent, который никак не связан в иерархии с классом Component.

Панель меню MenuBar содержит само меню.

```
//create a menu bar
MenuBar menuBar = new MenuBar();
//create menus
Menu fileMenu = new Menu("File");
//create menu items
MenuItem newMenuItem = new MenuItem("New",new MenuShortcut(KeyEvent.VK_N));
newMenuItem.setActionCommand("New");

newMenuItem.addActionListener(new ActionListener {
public void actionPerformed(ActionEvent e) {
  statusLabel.setText(e.getActionCommand()
    + " MenuItem clicked.");
} );
//add menu items to menus
fileMenu.add(newMenuItem);
//add menu to menubar
menuBar.add(fileMenu);
//add menubar to the frame
mainFrame.setMenuBar(menuBar);
mainFrame.setVisible(true);
```

MenuBar добавляется к Frame с помощью метода setMenuBar.

По умолчанию, панель меню добавляется сверху окна Frame.

MenuBar не может быть добавлена к другим сторонам окна.

Меню Menu содержит пункты меню.

Меню добавляется в панель меню с помощью метода add.

В меню можно добавить подменю.

Элемент MenuItem отображает опцию, которую может выбрать пользователь.

Элементы меню добавляются в меню с помощью метода addMenuItem.

Между пунктами меню может быть добавлен разделитель с помощью метода addSeparator.

Элемент CheckboxMenuItem отличается от элемента MenuItem тем, что он отображается вместе с флажком.

В этом примере сначала мы создаем панель меню.

Затем создаем меню.

Далее создаем элементы меню и присоединяем к ним слушателей событий.

Метод setActionCommand устанавливает имя команды для события действия, которое генерируется этим пунктом меню.

По умолчанию для команды действия, имя устанавливается как метка элемента меню.

Затем элемент меню добавляется в меню.

Меню добавляется в панель меню.

А панель меню устанавливается для окна верхнего уровня.

Всплывающее меню PopupMenu представляет собой меню, которое можно динамически показывать в указанной позиции внутри компонента.

```
final PopupMenu editMenu = new PopupMenu("Edit");

MenuItem cutMenuItem = new MenuItem("Cut");
cutMenuItem.setActionCommand("Cut");

MenuItemListener menuItemListener = new MenuItemListener();

cutMenuItem.addActionListener(menuItemListener);

editMenu.add(cutMenuItem);

Panel controlPanel = controlPanel = new Panel();
controlPanel.add(editMenu);

controlPanel.addMouseListener(new MouseAdapter() {
  public void mouseClicked(MouseEvent e) {
      editMenu.show(controlPanel, e.getX(), e.getY());
  }
});
mainFrame.add(controlPanel);
mainFrame.setVisible(true);
```

Для этого класс PopupMenu предоставляет метод show, который в качестве аргумента получает компонент, в котором отображается всплывающее меню, и координаты позиции для отображения меню.

В этом примере мы создаем всплывающее меню.

И в слушателе клика мыши панели, показываем всплывающее меню.

Компонент Canvas

Графические интерфейсы пользователя Java

Лекция 7
Компонент Canvas

Компонент Canvas представляет собой прямоугольную область, где приложение может рисовать что-либо, или может служить основой для пользовательского компонента, который не содержит другой компонент, например, кнопка с изображением.

```
import java.awt.*;
/** A Circle component built using a Canvas. */

public class Circle extends Canvas {
private int width, height;
public Circle(Color foreground, int radius) {
setForeground(foreground);
width = 2*radius;
height = 2*radius;
setSize(width, height);
}
public void paint(Graphics g) {
g.fillOval(0, 0, width, height);
}
public void setCenter(int x, int y) {
setLocation(x - width/2, y - height/2);
}
}
```

Для использования Canvas создается класс, расширяющий Canvas, и переопределяется метод paint, отвечающий за отрисовку холста.

Этот метод автоматически вызывается средой выполнения при создании объекта класса, и получает в качестве аргумента объект Graphics.

Класс Graphics обеспечивает основу для всех графических операций в AWT.

Он играет две разные, но связанные роли.

Во-первых, это графический контекст.

Графический контекст – это информация, которая влияет на операции рисования.

Эта информация включает в себя цвета фона и переднего плана, шрифт, а также расположение и размеры отсекающе-

го прямоугольника – область компонента, в котором можно рисовать графику.

Во-вторых, класс Graphics предоставляет методы для рисования простых геометрических фигур, текста и изображений.

Так как класс Graphics является абстрактным базовым классом, он не может быть создан непосредственно.

Экземпляр Graphics создается при создании компонента и передается в качестве аргумента методам update и paint компонента.

При использовании класса Graphics, фактическая работа выполняется конкретными его реализациями, которые тесно связаны с конкретной платформой.

Виртуальная машина Java предоставляет необходимые конкретные классы для вашей среды.

Вам не нужно беспокоиться о классах, специфичных для платформы; как только у вас есть объект Graphics, вы можете вызывать все методы класса Graphics, и быть уверенными, что классы, специфичные для платформы, будут работать правильно, где бы ни работала ваша программа.

Класс Graphics позволяет нарисовать линию.

```
g.drawLine(0, 0, 250, 250);

g.setColor(Color.RED);
g.fillOval(75, 75, 100, 100);

g.setColor(Color.BLUE);
g.fillRect(200, 200, 100, 100);

Image image = Toolkit.getDefaultToolkit().getImage("images/img.gif");
g.drawImage(image, 0, 0, this);

int fontSize = 20;
g.setFont(new Font("TimesRoman", Font.PLAIN, fontSize));
g.setColor(Color.red);
g.drawString("www.java.com", 10, 20);
```

Закрасить цветом овал и прямоугольник.

Нарисовать строку текста и другое.

Также можно нарисовать изображение, которое можно получить с помощью инструмента Toolkit, который обеспечивает связь с нативной платформой.

Здесь вызов метода drawImage запускает новый поток, который загружает запрошенное изображение.

Последний аргумент метода this – это наблюдатель изображения, который отслеживает процесс загрузки изображения.

Поток, который загружает изображение, уведомляет наблюдателя изображения, когда появляются новые данные.

Класс Component реализует интерфейс ImageObserver, поэтому можно использовать this в качестве наблюдателя

изображения при вызове метода drawImage.

Объект Toolkit представляет собой абстрактный класс, который предоставляет интерфейс для получения специфических для платформы деталей, таких как размер окна, доступные шрифты и печать.

Каждая платформа, поддерживающая Java, должна предоставить конкретный класс, который расширяет класс Toolkit.

Вы можете использовать объект Toolkit, если вам нужно получить изображение в приложении, получить информацию о шрифтах, получить цветовую модель, получить параметры экрана и так далее.

Java 2D

Графические интерфейсы пользователя Java

Лекция 8
Java2D

Как уже было сказано, библиотека AWT была дополнена библиотекой Java 2D API, расширяющей возможности работы с двухмерной графикой и изображениями.

Java 2D API предоставляет единую модель рендеринга для разных типов устройств.

На уровне приложения процесс рендеринга одинаковый, является ли целевое устройство рендеринга экраном или принтером.

Когда компонент должен отображаться, автоматически вызывается его метод paint или update с соответствующим

графическим контекстом Graphics.

Java 2D API предоставляет класс Graphics2D, который расширяет класс Graphics, чтобы обеспечить доступ к расширенной графике и функциям рендеринга Java 2D API.

```
public void paint (Graphics g) {
   Graphics2D g2 = (Graphics2D) g;
   ...
}
```

Чтобы использовать функции Java 2D API в приложении, нужно привести объект Graphics, переданный в метод рендеринга компонента, к объекту Graphics2D.

И объект Graphics2D предоставит следующие функции:

Это отображение набора примитивов, реализующих интерфейс Shape.

При этом примитив может быть заполнен цветом и его контуры могут быть нарисованы с указанием определенной толщины и шаблоном штрихов.

Graphics2D позволяет создать композицию примитивов, переместить, масштабировать, повернуть и обрезать форму.

Также Graphics2D позволяет конвертировать текстовую строку в глифы, которые затем могут быть заполнены цветом.

Примитивы – это точки, линии, прямоугольники, эллипсы, дуги, кривые.

```
Rectangle2D shape = new Rectangle2D.Float();
shape.setFrame(100, 150, 200,100);
Graphics2D g2 = (Graphics2D) g;
g2.draw (shape);

QuadCurve2D shape = new QuadCurve2D.Double();
shape.setCurve(250D,250D,100D,100D,200D,150D);
Graphics2D g2 = (Graphics2D) g;
g2.draw (shape);

int x2Points[] = {0, 100, 0, 100};
int y2Points[] = {0, 50, 50, 0};
GeneralPath polyline =
     new GeneralPath(GeneralPath.WIND_EVEN_ODD,
x2Points.length);

polyline.moveTo (x2Points[0], y2Points[0]);

for (int index = 1; index < x2Points.length; index++) {
      polyline.lineTo(x2Points[index], y2Points[index]);
};
Graphics2D g2 = (Graphics2D) g;
g2.draw(polyline);
```

Также можно нарисовать произвольную форму с помощью пути рисования GeneralPath.

Контур примитива можно определить с помощью объекта Stroke.

```
float dash1[] = {10.0f};
BasicStroke dashed =
new BasicStroke(1.0f, BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER, 10.0f, dash1, 0.0f);
Graphics2D g2 = (Graphics2D) g;
g2.setStroke(dashed);
g2.draw(new RoundRectangle2D.Double(x, y, rectWidth, rectHeight, 10, 10));
```

```
Paint redtowhite = new GradientPaint(0,0,color.RED,100, 0,color.WHITE);
Graphics2D g2 = (Graphics2D) g;
g2.setPaint(redtowhite);
g2.fill (new Ellipse2D.Double(0, 0, 100, 50));
```

А заполнить примитив цветом можно с помощью объекта Paint.

Java 2D API предоставляет различные возможности для отображения текста, включая установку атрибутов шрифта и выполнения компоновки текста.

```
String text = "Hello World";
int x = 10;
int y = 100;

Font font = new Font("Georgia", Font.ITALIC, 50);
Graphics2D g2 = (Graphics2D) g;

g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);

TextLayout textLayout = new TextLayout(text, font, g2.getFontRenderContext());
g2.setPaint(new Color(150, 150, 150));
textLayout.draw(g2, x + 3, y + 3);

g2.setPaint(Color.BLACK);
textLayout.draw(g2, x, y);
```

Если вы просто хотите нарисовать статическую текстовую строку, проще всего это сделать с помощью метода drawString класса Graphics, указав шрифт методом setFont класса Graphics.

Если вы хотите контролировать качество отображения текста и его компоновку, вы можете использовать Java 2D API.

Java 2D API позволяет контролировать качество отображения с помощью подсказок класса RenderingHints, например, указав сглаживание текста.

Класс TextLayout предоставляет различные возможности для стилизации текста.

Перед тем, как текст может быть отображен, он должен быть правильно сформирован и расположен с использовани-

ем соответствующих символов и лигатур.

Этот процесс называется компоновкой текста.

Процесс компоновки текста включает в себя формирование текста с использованием соответствующих глифов и лигатур, упорядочивание текста, измерение и позиционирование текста.

Что касается изображений, Java 2D API позволяет загрузить внешний файл изображения формата GIF, PNG, JPEG во внутреннее представление изображения, используемое Java 2D.

Непосредственно создать 2D Java изображение и отобразить его.

Отрисовать содержимого 2D-изображения Java на поверхности.

Сохранить содержимое 2D Java изображения во внешний файл изображения GIF, PNG или JPEG.

Для работы с изображениями используются два класса – это класс Image – суперкласс, представляющий графические изображения в виде прямоугольных массивов пикселей.

И класс BufferedImage, который расширяет класс Image, чтобы приложение могло работать непосредственно с данными изображения, например, извлечение или настройка цвета пикселя.

И приложения могут напрямую создавать экземпляры этого класса.

Класс BufferedImage управляет изображением в памяти

и предоставляет методы для хранения, интерпретации и получения данных пикселей.

Так как BufferedImage является подклассом Image, его контент может быть визуализирован с помощью методов Graphics и Graphics2D, которые принимают параметр Image.

Для загрузки изображения из внешнего источника используется Image I/O API, которое поддерживает форматы изображения GIF, PNG, JPEG, BMP.

```
BufferedImage img = null;
try {
   URL url = new URL(getCodeBase(), "examples/strawberry.jpg");
   img = ImageIO.read(url);
} catch (IOException e) {
}
```

```
try {
   // retrieve image
   BufferedImage bi = . . .
   File outputfile = new File("saved.png");
   ImageIO.write(bi, "png", outputfile);
} catch (IOException e) {
   ...
}
```

Соответственно Image I/O API используется и для сохранения объекта BufferedImage во внешний формат изображения.

Для получения изображения из внешнего источника также можно использовать класс Toolkit, как мы видели уже

раньше.

Для отрисовки полученного изображения нужен объект Graphics или Graphics2D.

```
BufferedImage img = ImageIO.read(imageSrc);

int w = img.getWidth(null);
int h = img.getHeight(null);

BufferedImage bi = new BufferedImage(w, h, BufferedImage.TYPE_INT_ARGB);

Graphics g = bi.getGraphics();

g.drawImage(img, 0, 0, null);

Graphics2D g2 = bi.createGraphics();

g2.drawImage(img, 0, 0, null);
```

Если мы не используем метод paint или update компонента, получить объект Graphics2D можно вызвав метод createGraphics, но только изображения, которое вы создали сами.

Поэтому, получив изображение из внешнего источника, мы создаем пустое изображение того же размера, получаем из него объект Graphics2D, и используем его для отрисовки нашего изображения.

Используя возможность получения графического контекста из изображения, можно, например, создать свое изоб-

ражение, рисуя в нем с помощью объекта Graphics или Graphics2D.

```
BufferedImage bufferedImage = new
BufferedImage(200,200,BufferedImage.TYPE_INT_RGB);
Graphics g = bufferedImage.getGraphics();

g.drawString("java.com", 20,20);
```

Работа с изображениями

Графические интерфейсы пользователя Java

Лекция 9
Работа с изображениями

AWT предоставляет некоторые возможности для управления изображениями с помощью пакета java.awt.image.

Как мы уже говорили, получить изображение из внешнего источника можно с помощью объекта Toolkit.

```
Image image = Toolkit.getDefaultToolkit().getImage(filename);
Image image = Toolkit.getDefaultToolkit().getImage(url);

BufferedImage bufferedImage = new BufferedImage(image.getWidth(null), image.getHeight(null),
BufferedImage.TYPE_INT_RGB);

Graphics2D g2 = bufferedImage.createGraphics();
g2.drawImage(image, x, y, this);

Image scaled_image = image.getScaledInstance(new_width,new_height, Image.SCALE_DEFAULT);

Graphics g = image.getGraphics();
g.setFont(new Font("Dialog",Font.BOLD,20));
g.setColor(Color.red);
g.drawString("This is a cool picture",100,100);
```

Затем с помощью BufferedImage мы можем получить объект графического контекста и нарисовать изображение.

Далее вы можете масштабировать это изображение.

Если вы хотите что-то нарисовать на своем изображении, вы можете получить графический объект и делать все что захотите.

Java была разработана для загрузки изображения во время работы программы.

Таким образом, вы можете вызвать методы getWidth и getHeight до того, как Java узнает размер изображения.

В этом случае размер изображения будет установлен в -1.

Это основная проблема, когда вам нужно знать размер изображения.

Решение этой проблемы заключается в том, чтобы ваша

программа дождалась загрузки изображения.

Сделать это можно с помощью класса MediaTracker, предназначенного для отслеживания состояния изображений.

```
MediaTracker media_tracker = new MediaTracker(this);

// add your image to the tracker with an arbitrary id
int id = 0;
media_tracker.addImage(my_image, id);

// try to wait for image to be loaded
// catch if loading was interrupted
try
{
   media_tracker.waitForID(id);
}
catch(InterruptedException e)
{
   System.out.println("Image loading interrupted : " + e);
}
```

Мы добавляем изображение для отслеживания и вызываем метод waitForID, который начинает загрузку изображения, отслеживаемого этим медиа-трекером, с указанным идентификатором.

Этот метод ожидает завершения загрузки изображения с указанным идентификатором.

Отслеживать загрузку изображения также можно с помощью интерфейса ImageObserver, который реализуется классом Component.

ImageObserver – это интерфейс, используемый для прие-

ма уведомлений о том, как генерируется изображение.

ImageObserver определяет только один метод: ImageUpdate ().

Использование наблюдателя изображения позволяет выполнять (параллельно с загрузкой изображения) другие действия, такие как показ индикатора хода работы или дополнительного экрана, который информирует о ходе загрузки изображения.

Многие из разработчиков Java находили интерфейс ImageObserver слишком сложным для понимания и управления загрузкой множественных изображений.

Поэтому был создан класс MediaTracker, который позволяет параллельно проверять состояние произвольного числа изображений.

Для многих реальных задач обработки изображений необходимо получить массив с пикселями изображения.

```
int w = ti.projected_image.getWidth(null);
int h = ti.projected_image.getHeight(null);
// the rectangle (x,y) to (x+w,y+h) will be grabbed
int x=0, y=0;
// offset into the array of the top left corner of the grabbed rectagle
int off = 0;
// horizontal size of the array
int scansize = w;
// pixels[] MUST BE ALLOCATED
int[] pixels = new int[h*w]; // allocate a larger array if offset is not zero
// create a pixel grabber object
// pixels are stored in the default RGB color model
PixelGrabber pixel_grabber = new PixelGrabber(my_image, x, y, w, h, pixels, off, scansize);
// grab pixels
try{
    pixel_grabber.grabPixels();
}catch(InterruptedException e)
{
    System.out.println("Couldn't grab pixels " + e);
}
```

Здесь показан код, который позволяет сделать это.

В этом коде мы предполагаем, что изображение полностью загружено.

Сначала мы получаем размер изображения.

Затем создаем массив для пикселей изображения.

Далее создаем объект PixelGrabber, который с помощью метода grabPixels позволяет извлечь пиксели в массив, указанный при создании объекта PixelGrabber.

После получения массива пикселей изображения, его можно обработать и создать новое изображение.

```
int pink = Color.pink.getRGB();
for(int j=0;j<h;j++)
for(int i=0;i<w;i++)
pixels[i+j*scansize+off] = pink;

MemoryImageSource source = new MemoryImageSource(
                           width,
                           height,
                           ColorModel.getRGBdefault(),
                           pixels,
                           offset,
                           scansize);

Image image = Toolkit.getDefaultToolkit().createImage(source);

g.drawImage(image, x, y, this);
```

Для создания изображения из массива пикселей используется вспомогательный класс MemoryImageSource и объект Toolkit.

И наконец полученное изображение можно нарисовать.

На слайде показано, как создаются данные изображения за сценой.


```
NewImageProducer nip = new NewImageProducer();

Image _im = createImage(nip);

NewImageConsumer nic = new NewImageConsumer();

    nip.addConsumer(nic);

public void paint(Graphics g)
  {
    g.drawImage(_im, 50, 50, this);
  }
```

Производитель изображения – это объект, который реализует интерфейс ImageProducer, и создает необработанные данные для объекта изображения Image.

Производитель изображения предоставляет эти данные потребителю изображения – объекту, который реализует интерфейс ImageConsumer.

Если вам не нужно манипулировать или создавать пользовательские изображения, вам обычно не нужно знать о производителе изображения и потребителе изображения.

AWT автоматически использует производителя и потребителя изображения за сценой.

При создании экземпляра класса Image требуется наличие производителя изображений.

Код глубоко внутри AWT создает производителя изобра-

жений.

Как альтернатива программист может предоставить создателя изображения, при создании объекта изображения Image.

При рисовании экземпляра класса Image методом draw, глубоко внутри AWT создается потребитель изображения.

Здесь показано, как при создании пользовательского компонента, можно с помощью метода createImage класса Component создать пользовательское изображение, реализуя собственные ImageProducer и ImageConsumer, и нарисовать это изображение в методе paint компонента.

Java позволяет «фильтровать» изображения с помощью класса ImageFilter или его подклассов.


```
Image sourceImage;

ImageFilter filter = new SomeImageFilter();

ImageProducer producer = new FilteredImageSource(sourceImage.getSource(), filter);

Image resultImage = createImage(producer); // Component

Image resultImage = Toolkit.getDefaultToolkit().createImage(producer);
```

AWT поддерживает обработку изображений, позволяя вставлять фильтры изображений между производителем изображения и потребителем изображения.

Фильтр изображения представляет собой объект ImageFilter, который находится между производителем и потребителем изображения, изменяя данные изображения до того, как потребитель получит его.

ImageFilter реализует интерфейс ImageConsumer, поскольку фильтр изображения перехватывает сообщения, которые производитель отправляет потребителю.

Для использования фильтра, создается объект фильтра.

Далее нужно создать объект класса ImageProducer.

Этот объект создается при помощи конструктора класса FilteredImageSource.

В качестве первого параметра мы передаем конструктору ссылку на источник данных исходного изображения, полученный методом getSource, а через второй – ссылку на свой фильтр.

Затем мы можем создать новое изображение методом createImage.

Чтобы дождаться процесса завершения формирования изображения, используйте класс MediaTracker.

После этого изображение готово и доступно для рисования методом drawImage.

Все фильтры изображений должны быть подклассами класса ImageFilter.

Если ваш фильтр изображения изменяет цвета или прозрачность изображения, вы можете создать подкласс класса RGBImageFilter, который расширяет класс ImageFilter.

Здесь показан простой фильтр изображения, который отфильтровывает отдельные цветовые компоненты (красный, зеленый и синий) изображения.

```
class ColorFilter extends RGBImageFilter {
boolean red, green, blue;

public ColorFilter(boolean r, boolean g, boolean b) {
red = r;
 green = g;
 blue = b;
 canFilterIndexColorModel = true;
 }

public int filterRGB(int x, int y, int rgb) {
// Filter the colors
int r = red ? 0: ((rgb >> 16) & 0xff);
int g = green ? 0: ((rgb >> 8) & 0xff);
int b = blue ? 0: ((rgb >> 0) & 0xff);

// Return the result
return (rgb & 0xff000000) | (r << 16) | (g << 8) | (b << 0);
}
 }
```

Класс ColorFilter расширяет класс RGBImageFilter и содержит три логические переменные, которые определяют, какие цвета должны быть отфильтрованы из изображения.

Эти переменные задаются параметрами, переданными в конструктор.

Переменная canFilterIndexColorModel, унаследованная от RGBImageFilter, установлена в значение true, чтобы ука-

зать, что записи цветовой карты могут быть отфильтрованы, если входящее изображение использует модель индексированного цвета.

В Java цвета пикселей управляются через цветовые модели.

Цветовые модели Java обеспечивают важную абстракцию, которая позволяет Java работать с изображениями разных форматов единым образом.

Цветовая модель представляет собой объект Java, который предоставляет методы для перевода значений пикселей в соответствующие красные, зеленые и синие цветовые компоненты изображения.

Это может показаться тривиальной задачей, зная, что компоненты цвета пикселей аккуратно упакованы в 32-битное значение.

Тем не менее, существуют различные типы цветовых моделей, отражающие различные методы определения цветов пикселей.

Двумя типами цветовых моделей, поддерживаемых Java, являются прямая цветовая модель и модель индексированных цветов.

Прямая цветовая модель работает со значениями пикселей, которые представляют цвет RGB и альфа-информацию отдельно, которые упакованы для каждого пикселя в одно значение.

Модель индексированных цветов поддерживается 8-бит-

ными изображениями, содержащими не более 256 цветов.

Эта модель работает с картой цветов изображения, в которой хранятся и индексируются цвета, используемые в изображении.

Эта позволяет уменьшить размер файла изображения, при этом сохраняя качество изображения.

Вернемся к нашему примеру.

Помимо конструктора, класс ColorFilter реализует только один метод filterRGB, который является абстрактным методом, определенным в классе RGBImageFilter.

Метод filterRGB принимает три параметра: положение x и y пикселя внутри изображения и 32-битное (целочисленное) значение цвета.

Единственный параметр, которым вы занимаетесь, является значение цвета, rgb.

Стандартная цветовая модель RGB помещает красные, зеленые и синие компоненты в нижние 24 бита 32-битного значения цвета.

И каждый из них можно извлечь, сдвигая параметр rgb.

Эти отдельные компоненты хранятся в локальных переменных r, g и b.

Обратите внимание, что здесь каждый компонент цвета смещается только в том случае, если он не фильтруется.

Для фильтрованных цветов, компонент цвета установлен в 0.

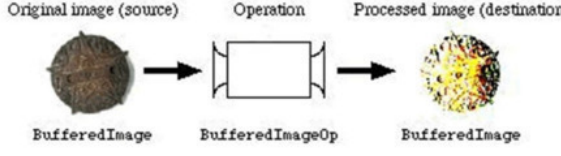
Новые цветовые компоненты затем переносятся обрат-

но в 32-битное значение цвета и возвращаются из метода filterRGB.

Обратите внимание, что альфа-компонент значения цвета не изменяется.

Для этого используется маска 0xff000000, потому что альфа-компонент находится в верхнем байте значения цвета.

Помимо обработки изображений, с помощью вставки фильтров изображений между производителем изображения и потребителем изображения,



Java поддерживает фильтрацию изображений с помощью интерфейса BufferedImageOp.

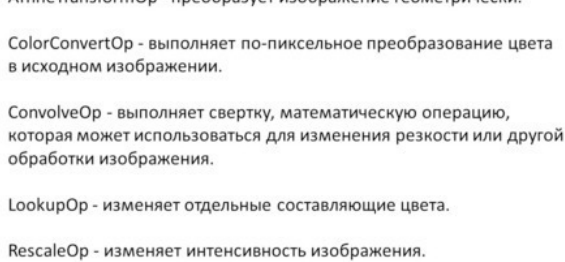
Метод filter интерфейса BufferedImageOp принимает объект BufferedImage как вход (исходное изображение) и выпол-

няет обработку данных изображения, создавая другой объект BufferedImage (конечное изображение).

Напомним, что класс BufferedImage расширяет класс Image, обеспечивая доступ к буферу данных изображения.

Java 2D API предоставляет набор реализаций интерфейса BufferedImageOp.

AffineTransformOp – преобразует изображение геометрически.



ColorConvertOp – выполняет по-пиксельное преобразование цвета в исходном изображении.

ConvolveOp – выполняет свертку, математическую операцию, которая может использоваться для размытия, изменения резкости или другой обработки изображения.

LookupOp – изменяет отдельные составляющие цвета.

RescaleOp – изменяет интенсивность изображения.

Здесь показан пример применения фильтра RescaleOp, изменяющего интенсивность цвета.

```
Image img = Toolkit.getDefaultToolkit().getImage("img.gif");
 try{
     MediaTracker mt = new MediaTracker(this);
     mt.addImage(img, 0);
     mt.waitForID(0);
  }catch(Exception e){}
  bi = new BufferedImage(img.getWidth(this), img.getHeight(this), BufferedImage.TYPE_INT_RGB);
  Graphics2D big = bi.createGraphics();
  big.drawImage(img, 0, 0, this);

 public void paint(Graphics g){
  Graphics2D g2 = (Graphics2D)g;
  int w = getSize().width;
  int bw = bi.getWidth(this);
  int bh = bi.getHeight(this);
  BufferedImage bimg = new BufferedImage(bw, bh, BufferedImage.TYPE_INT_RGB);
  RescaleOp rop = new RescaleOp(0.5f, 70.0f, null);
  rop.filter(bi, bimg);

  g2.drawImage(bi, null, 10, 30);
  g2.drawImage(bimg, null, w/2+3, 30);
 }
```

В этом примере сначала создается исходный объект BufferedImage на основе изображения, затем создается пустой объект BufferedImage.

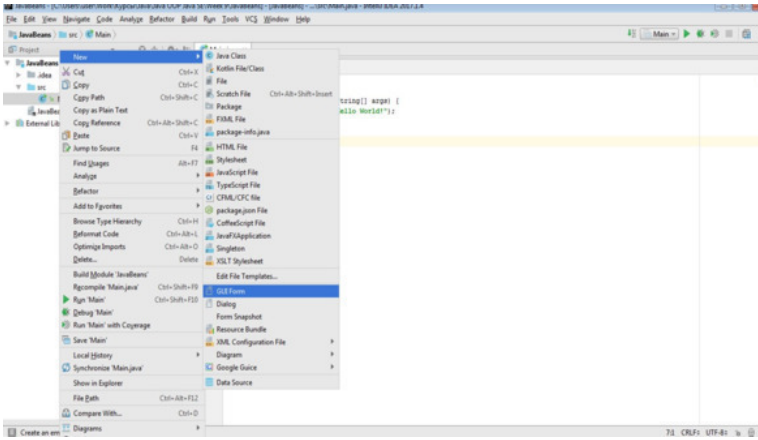
Который заполняется отфильтрованными данными исходного изображения, с помощью метода filter созданного объекта RescaleOp.

JavaBeans и POJO

Откроем среду IntelliJ IDEA с созданным проектом Java приложения.

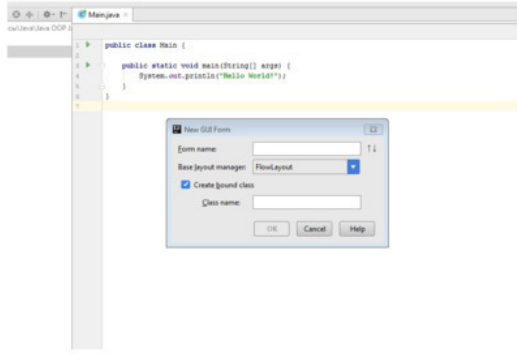
Графические интерфейсы
пользователя Java

Лекция 10
JavaBeans и POJO

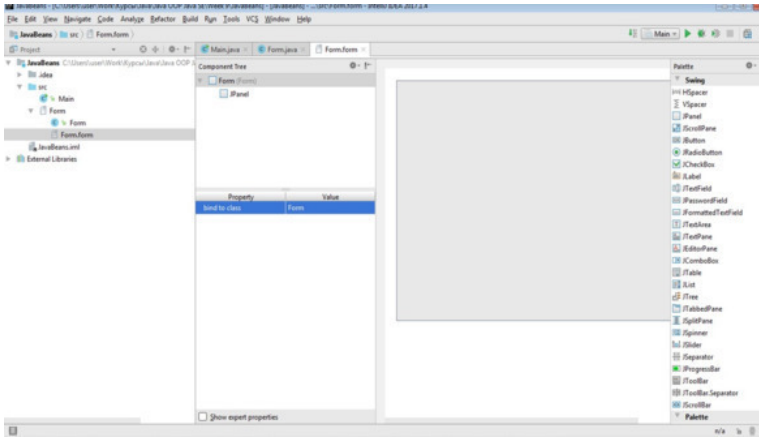


Нажмем правой кнопкой мыши на пакете приложения, и в меню выберем New – GUI Form.

Введем имя формы.



В результате будет создан Java класс и связанное с ним XML описание, которое открывается в редакторе IntelliJ IDEA GUI Designer.

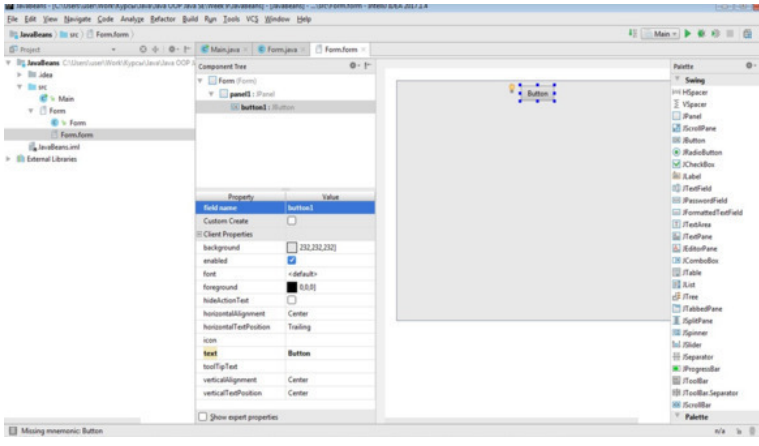


Редактор IntelliJ IDEA GUI Designer позволяет создавать графические пользовательские интерфейсы (GUI) приложения, используя компоненты библиотеки Swing.

Этот инструмент помогает создавать диалоговые окна и группы элементов управления, которые будут использоваться в контейнере верхнего уровня, таком как JFrame.

Когда вы создаете форму с помощью GUI Designer, вы создаете панель, а не фрейм.

Пользуясь палитрой компонентов редактора, вы можете перетаскивать компоненты в форму и редактировать их свойства.



Для того чтобы компонент графического интерфейса пользователя можно было применять в таком визуальном средстве разработки, он должен обладать дополнительными качествами.

У него должен быть ярлык, помещаемый в палитру компонентов.

Среди полей компонента должны быть выделены свойства (properties), которые будут показаны в окне свойств.

У него должны быть определены методы доступа getXxx () /setXxx () /isXxx () к каждому свойству.

Этими методами будет пользоваться IDE среда разработки, чтобы определить свойства компонента.

Компонент, имеющий эти и другие определенные свойства, в технологии Java называется компонентом JavaBean.

В него может входить один или несколько классов.

Как правило, файлы этих классов упаковываются в один jar-файл и отмечаются в его файле MANIFEST. MF как JavaBean: True.

Все компоненты AWT и Swing являются компонентами JavaBeans.

Спецификация Sun Microsystems определяет JavaBeans как повторно используемые программные компоненты, которыми можно управлять, используя графические конструкторы и средства IDE.

Визуальные средства разработки – это не основное применение JavaBeans.

Главное достоинство компонентов, оформленных как JavaBeans, в том, что они без труда встраиваются в любое приложение.

И приложение можно собрать из готовых JavaBeans как из строительных блоков, остается только настроить их свойства.

Это называется компонентное программирование.

Чтобы класс мог работать как bean, он должен соответствовать определённым соглашениям об именах методов, конструкторе и поведении.

Эти соглашения дают возможность создания инструментов, которые могут использовать, замещать и соединять JavaBeans.

Эти соглашения следующие:

Класс должен иметь конструктор без параметров, с модификатором доступа public.

Такой конструктор позволяет инструментам создать объект без дополнительных сложностей с параметрами.

Свойства класса должны быть доступны через get, set и другие методы (так называемые методы доступа), которые должны подчиняться стандартному соглашению об именах.

Это легко позволяет инструментам автоматически определять и обновлять содержание bean'ов.

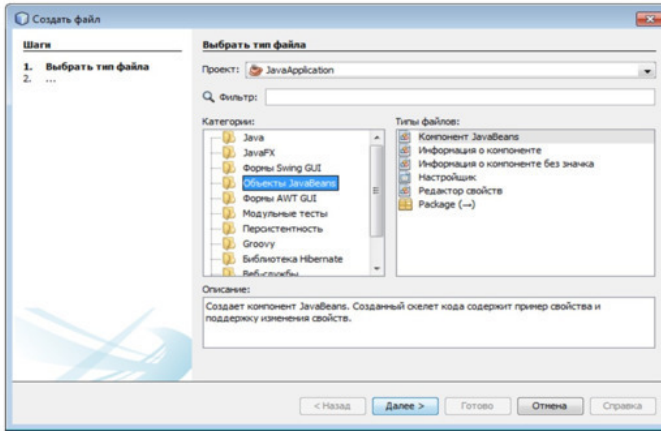
Многие инструменты даже имеют специализированные редакторы для различных типов свойств.

Класс должен быть сериализуем.

Это даёт возможность надёжно сохранять, хранить и восстанавливать состояние bean независимым от платформы и виртуальной машины способом.

Класс должен иметь переопределенные методы equals, hashCode и toString.

В среде разработки NetBeans можно легко создать компонент JavaBeans с помощью меню New проекта приложения.



Для создания компонентов JavaBeans Java SE API включает в себя пакет java.beans.

В частности, помимо обычных свойств, представленных отдельными значениями и массивами значений, компонент JavaBeans может иметь связанные свойства, которые уведомляют слушателей, когда изменяется их значение.

Класс компонента JavaBeans содержит методы addPropertyChangeListener и removePropertyChangeListener для управления слушателями, и когда связанное свойство изменяется, компонент отправляет событие PropertyChangeEvent своим зарегистрированным слушателям.

Также класс компонента JavaBeans может запускать любой тип события, включая пользовательские события.

Как и в случае со свойствами, события идентифицируются по определенной схеме имен методов, add <Event> Listener и remove <Event> Listener.

Другое понятие, с которым вы можете столкнуться – это Plain Old Java Object.

Plain Old Java Object – это объект Java, который не расширяет или не реализует специализированные классы и интерфейсы каких-либо фреймворков.

POJO это просто класс Java, который содержит только поля, без логики их обработки и обеспечивает доступ ко все полям только через методы get/set.

Компоненты JavaBeans дополняют Plain Old Java Object наличием конструктора по умолчанию и наличием слушателей listener изменения свойств.

То есть Plain Old Java Object описывает структуру данных для дальнейшего использования в приложении.

Сериализация

Графические интерфейсы пользователя Java

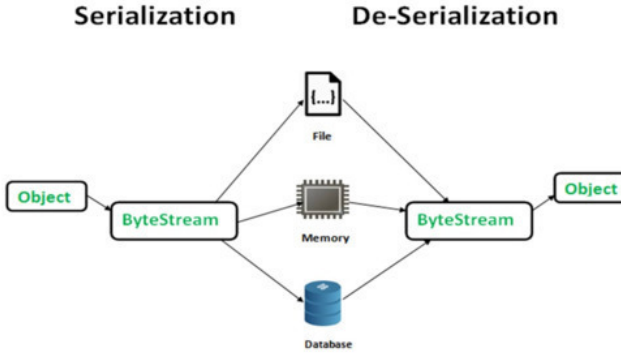
Лекция 11
Сериализация

Все мы знаем о том, что Java позволяет создавать в памяти объекты для многократного использования.

Однако все эти объекты существуют лишь до тех пор, пока выполняется создавшая их виртуальная машина.

Было бы неплохо, если бы создаваемые нами объекты могли бы существовать и вне пределов жизненного цикла виртуальной машины?

Как уже было сказано, JavaBeans обладает свойством сохранения, когда его свойства, поля и информация о состоянии могут сохраняться и извлекаться из хранилища.



Механизм, который делает возможным такое сохранение, называется сериализацией.

Сериализация объектов означает преобразование объекта в поток данных и запись его в хранилище, при этом объект представлен как последовательность байтов, которая включает в себя данные объекта, а также информацию о типе объекта и типах данных, хранящихся в объекте.

Любое приложение, которое использует такой компонент, может затем «восстановить» его путем десериализации.

Затем объект возвращается в исходное состояние.

Например, приложение Java может сериализовать окно Frame на машине Windows, сериализованный файл можно отправить с помощью электронной почты на машину Solaris,

а затем приложение Java сможет восстановить окно Frame в точное состояние, существующее на машине Windows.

Чтобы так сохраняться, компоненты должны поддерживать сериализацию, реализуя либо интерфейс java.io.Serializable, либо интерфейс java.io.Externalizable.

При сериализации объекта в файл, по соглашению создается файл с расширением. ser.

```
public class Employee implements java.io.Serializable {
   public String name;
   public String address;
   public transient int SSN;
   public int number;
   }
```

```
Employee e = new Employee();

try {
      FileOutputStream fileOut =
      new FileOutputStream("/tmp/employee.ser");
      ObjectOutputStream out = new
ObjectOutputStream(fileOut);
      out.writeObject(e);
      out.close();
      fileOut.close();
      System.out.printf("Serialized data is saved in
/tmp/employee.ser");
    } catch (IOException i) {
      i.printStackTrace();
    }
```

```
Employee e = null;
    try {
      FileInputStream fileIn =
new FileInputStream("/tmp/employee.ser");
      ObjectInputStream in = new ObjectInputStream(fileIn);
      e = (Employee) in.readObject();
      in.close();
      fileIn.close();
    } catch (IOException i) {
      i.printStackTrace();
      return;
    } catch (ClassNotFoundException c) {
      System.out.println("Employee class not found");
      c.printStackTrace();
      return;
    }
```

Классы ObjectInputStream и ObjectOutputStream представляют собой потоки, которые содержат методы сериализации и десериализации объекта.

Это два класса, с помощью которых собственно осуществляют сериализацию и десериализацию объекта.

Если какой-либо класс в иерархии наследования клас-

са реализует интерфейс Serializable или Externalizable, тогда этот класс и его подклассы сериализуются.

Примеры сериализуемых классов включают в себя классы Component, String, Date, Vector и Hashtable.

Известные классы, которые не поддерживающие сериализацию, включают в себя классы Image, Thread, Socket и InputStream.

Попытка сериализации объектов этих типов приведет к исключению NotSerializableException.

Java Object Serialization API автоматически сериализует большинство полей объекта Serializable в поток байтов.

Сюда входят примитивные типы, массивы и строки.

API не сериализует или десериализует поля, отмеченные как transient или static.

Transient (нерезидент) – модификатор полей класса.

Отмеченные этим модификатором поля не записываются в поток байт при применении стандартного алгоритма сериализации.

При десериализации объекта такие поля инициализируются значением по умолчанию.

Этот модификатор применяется, например, если поле класса является объектом несериализуемого класса, или некоторые поля могут не сериализоваться из соображений безопасности, например, поле пароля, или значение поля корректно только в рамках текущего контекста.

Вы можете контролировать уровень сериализации объек-

та.

Есть несколько способов управления серилизацией:

Это автоматическая сериализация, реализуемая интерфейсом Serializable.

Программное обеспечение сериализации Java сериализует весь объект, за исключением полей transient или static.

Интерфейс Serializable не объявляет никаких методов; он действует как маркер, сообщая инструментам сериализации объектов, что ваш класс сериализуется.

Маркировка класса интерфейсом Serializable означает, что вы сообщаете виртуальной машине Java (JVM), что ваш класс будет работать с сериализацией по умолчанию.

Классы, которые реализуют Serializable, должны иметь доступ к конструктору без аргументов супертипа.

Этот конструктор будет вызываться, когда объект будет «восстанавливаться» из файла. ser, в котором хранится сериализованный объект.

Вам не нужно реализовывать Serializable в вашем классе, если он уже реализован в суперклассе.

Все поля, кроме полей transient или static, сериализуются.

Используйте модификатор transient, чтобы указать поля, которые вы не хотите сериализовать, и указать классы, которые не являются сериализуемыми.

Если объект однажды уже записанный в поток, спустя какое-то время записывается в него снова, то по умолчанию, ObjectOutputStream сохраняет ссылки на объекты, которые

в него записываются.

Это означает, что, если состояние записываемого объекта, который уже был записан, будет записано снова, новое состояние не сохраняется.

Существует два способа это исправить.

Во-первых, вы можете каждый раз после вызова метода записи закрывать поток, а затем открывать его снова.

Во-вторых, вы можете вызвать метод ObjectOutputStream.reset, который сигнализирует потоку о том, что необходимо освободить кэш от ссылок, которые он хранит, чтобы новые вызовы методов записи действительно записывали данные.

Вы можете дополнительно контролировать, как объекты сериализуются, путем написания собственных реализаций методов writeObject и readObject и включить их в свой сериализуемый класс.

```
private void writeObject(ObjectOutputStream out) throws IOException
{
  out.defaultWriteObject();
...
 }
private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException
{
in.defaultReadObject();
...
}
```

Классы, требующие специальной обработки во время процесса сериализации и десериализации, должны определить эти методы с этими точными сигнатурами.

Метод writeObject класса определяет сериализацию объекта.

А метод readObject восстанавливает поток данных, который вы определили с помощью writeObject.

Если же вы хотите полностью контролировать сериализацию объекта, используйте интерфейс Externalizable, например, при записи и чтении определенного формата файла.

Для использования интерфейса Externalizable вам необходимо реализовать два метода: readExternal и writeExternal.

Классы, которые реализуют Externalizable, должны иметь конструктор без аргументов.

Класс XMLEncoder позволяет сериализовать объект в XML-файл.

```
MyBean mb = new MyBean();
FileOutputStream fos = new FileOutputStream("mybean.xml");
 BufferedOutputStream bos = new BufferedOutputStream(fos);
 XMLEncoder xmlEncoder = new XMLEncoder(bos);
 xmlEncoder.writeObject(mb);
 xmlEncoder.close();
```

```
FileInputStream fis = new FileInputStream("mybean.xml");
 BufferedInputStream bis = new BufferedInputStream(fis);
 XMLDecoder xmlDecoder = new XMLDecoder(bis);
 MyBean mb = (MyBean) xmlDecoder.readObject();
```

Одно из достоинств этой формы сериализации заключается в том, что вы можете легко просмотреть данные объекта в сохраненной форме.

Вы также можете изменить данные в файле в текстовом редакторе.

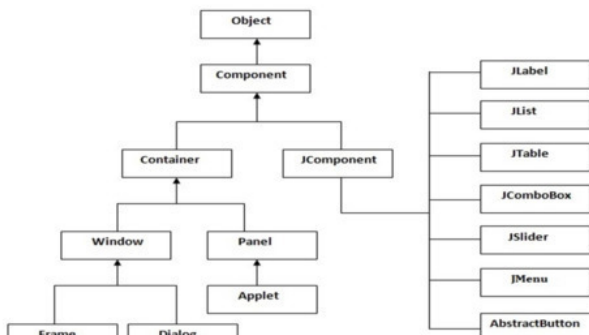
Соответственно с помощью класса XMLEncoder, можно легко восстановить потом объект из XML-файла.

Библиотека Swing

Графическая библиотека Swing была создана на основе библиотеки AWT для решения таких проблем AWT как недостаточный выбор графических компонентов и зависимость внешнего вида и поведения AWT графического интерфейса пользователя от конкретной операционной системы.

Графические интерфейсы пользователя Java

Лекция 12
Библиотека Swing



Эти проблемы были решены созданием классов и интерфейсов библиотеки Swing с использованием одного только языка Java.

Компоненты AWT являются тяжеловесными, в то время как компоненты Swing являются легковесными.

Библиотека AWT использует нативный код (код, специфичный для конкретной операционной системы) для отображения компонентов.

Каждая реализация среды выполнения Jave Runtime Environment должна обеспечивать собственные реализации кнопок, меток, панелей и всех других компонентов AWT.

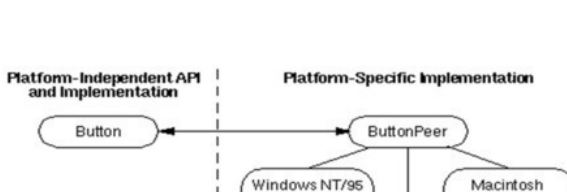
Всякий раз, когда используется компонент AWT, все запросы на рисование пересылаются нативному коду.

Это позволяет использовать кнопку AWT в приложении,

но кнопка отображается как кнопка Windows, или кнопка Mac или другая кнопка конкретной платформы, на которой работает приложение.

Эти части нативного кода, к которым обращается среда выполнения JRE при использовании AWT компонентов, называются пирами, так как они берут на себя ответственность за отображение компонента.

Прежде чем AWT компонент будет сначала нарисован на экране, будет создан его пир.



Код пира включает в себя нативный код для отображения компонента, определения текущего размера экрана и решения других проблем с нативной платформой.

Например, когда Java рисует метку, она не просто рисует

строку в нужном месте на экране.

Она создает пир и помещает его на экран.

В свою очередь компоненты Swing используют пиры для двух задач.

Во-первых, Swing компоненты JFrame, JWindow, JDialog и JApplet расширяют свои AWT-аналоги.

И все они используют пиры для отображения фактической области рисования на экране.

Другие компоненты Swing легковесные; у них нет пиров.

Эти компоненты отрисовывают себя поверх существующих JFrame, JWindow, JDialog или JApplet.

И поэтому на самом деле они также неявно используют пиры.

В AWT библиотеке использование пиров значительно затрудняло создавать подклассы этих компонентов для изменения их поведения.

Потому что их поведение исходит от нативного пира и поэтому не может быть легко переопределено или расширено.

Далее, наличие нативного кода значительно затрудняет перенос Java на новую платформу.

Проще говоря, хотя сам язык Java был кросс-платформенным, библиотека AWT была его ахиллесовой пятой.

Наконец, нативные пиры потребляют много ресурсов.

Можно было бы ожидать, что использование нативного кода будет намного более эффективным, чем создание компонентов на Java.

Тем не менее, для создания большого количества элементов GUI может потребоваться много времени, когда для каждого из них требуется создание его пира.

Решением было создание легковесных компонентов библиотеки Swing, которые полностью написаны на Java, и поэтому не требуют прямого использования нативного кода.

Легковесный компонент – это просто компонент, полностью реализованный на Java.

Для создания легковесного компонента достаточно расширить AWT класс Component или Container напрямую, реализуя внешний вид компонента на Java, а не делегируя внешний вид пиру.

Легковесные компоненты могут быть прозрачными, и они не должны быть прямоугольными, что является ограничением при работе с компонентами, имеющими пиры.

Вы реализуете весь внешний вид, используя методы paint и update, и вы реализуете поведение компонента, улавливая пользовательские события и, возможно, генерируя новые события.

В библиотеке AWT для создания совершенно нового компонента вам нужно было бы расширять класс Canvas.

Легковесному компоненту достаточно расширить класс Component или Container.

Когда легковесный компонент помещается в контейнер, он не получает нативный пир.

Вместо этого Toolkit создает для компонента объект

LightweightPeer, который служит как указатель и идентифицирует компонент как легковесный.

LightweightPeer помечает компонент как зависящий от нативного контейнера, так, чтобы события, связанные с окном, могли быть перенаправлены компоненту.

Поэтому тяжеловесный контейнер, который содержит легковесный компонент, берет на себя обязанности, которые в противном случае обрабатывались бы нативным пиром, а именно берет на себя низкоуровневую доставку событий и запросы на рисование.

Контейнер принимает события перемещения мыши, нажатия клавиш, запросы на рисование и т. д. для легковесного компонента и отправляет их ему.

Метод setBackground легковесного компонента, просто расширяющего класс Component, не работает, так как всю работу по изображению кнопки на экране выполняет не peer-двойник компонента, а тяжеловесный контейнер, в котором расположен компонент.

Контейнер ничего не знает о том, что надо обратиться к методу setBackground, он рисует только то, что записано в методе paint.

Если же мы создаем легковесный контейнер, то такой контейнер не умеет рисовать находящиеся в нем легковесные компоненты, поэтому в конце метода paint легковесного контейнера нужно вызвать метод paint суперкласса.

Тогда рисованием займется тяжеловесный суперк-

ласс-контейнер. Он нарисует и лежащий в нем легковесный контейнер, и размещенные в контейнере легковесные компоненты.

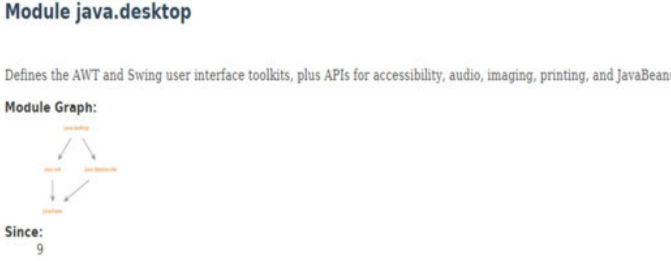
Предпочтительный размер тяжеловесного компонента устанавливается peer-объектом, поэтому для легковесных компонентов его надо задать явно, переопределив метод getPreferredSize, иначе некоторые менеджеры компоновки, например FlowLayout (), установят нулевой размер, и компонент не будет виден на экране.

Легковесные компоненты изначально рисуются прозрачными, и незакрашенная часть прямоугольного объекта не будет видна. Это позволяет создать компонент любой видимой формы.

Таким образом, на основе библиотеки AWT, появилась библиотека Swing – Java-библиотека классов и интерфейсов, представляющих такие стандартные графические компоненты интерфейса пользователя как кнопки, таблицы, поля ввода и др., а также методы для работы с ними.

Библиотека Swing составляет основную часть библиотеки JFC (Java Foundation Classes), которая объединяет классы и интерфейсы, обеспечивающие интерфейс пользователя Swing-компонентами, выбор внешнего вида и поведения интерфейса, поддержку технологий для людей с ограниченными возможностями, работу с высококачественной двухмерной графикой, текстом и изображениями, а также интернационализацию графического интерфейса пользователя.

В настоящее время библиотека Swing входит в набор настольных Java-технологий.



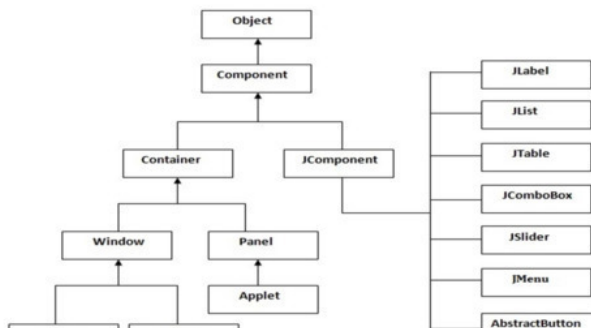
Module java.desktop

Defines the AWT and Swing user interface toolkits, plus APIs for accessibility, audio, imaging, printing, and JavaBeans.

Module Graph:

Since:
9

Все классы Swing-компонентов являются наследниками базового класса JComponent пакета javax. swing.



При конструировании Swing-интерфейса компоненты помещаются в контейнеры, образующие иерархию, которая берет свое начало от контейнеров верхнего уровня, представленных классами JApplet, который уже запрещен, JDialog, JFrame и JWindow.

Основные свойства графической библиотеки Swing можно суммировать как:

Кроссплатформенность.

Механизм сменного внешнего вида и поведения компонентов (pluggable look and feel).

Внешний вид и поведение графического интерфейса пользователя может определяться программным образом или может выбираться при выполнении приложения.

Расширяемость за счет возможности расширения классов

и интерфейсов библиотеки.

Архитектура Swing-компонентов основана на технологии Java Beans.

Возможность настройки внешнего вида Swing-компонентов за счет изменения свойств таких элементов компонентов как рамки, цвет, фон и др.

Легковесность.

Так как AWT-компоненты непосредственно взаимодействуют с операционной системой и представляют собой каждый маленькое индивидуальное окно, то их называют тяжеловесными.

Компоненты Swing представляют собой области в окне, характеризующиеся координатами и размером, при этом они не работают напрямую с операционной системой, их отображение основывается на реализации Java 2D API, поэтому их называют легковесными.

Если легковесные компоненты используются совместно с тяжеловесными, то так как тяжеловесные напрямую связаны с операционной системой, то они будут прорисовываться в первую очередь, поэтому в случае размещения тяжеловесных компонентов в легковесных, они будут перекрывать собой легковесные, что приведет к неверному отображению графического интерфейса.

В библиотеке Swing автоматически заложена двойная буферизация при выводе изображения, т.е. изображение сначала полностью формируется в оперативной памяти, а затем

целиком выводится на экран, таким образом устраняется эффект мерцания изображения и повышается скорость рисования.

Использование библиотеки Swing не является потоково-безопасной.

Доступ к Swing-компонентам должен осуществляться в специальном потоке Event Dispatch Thread (EDT).

Поддержка технологий для людей с ограниченными возможностями.

Снабжение компонентов всплывающими подсказками и механизмом управления клавиатурой.

Возможность настройки текста, отображаемого Swing-компонентами с помощью HTML-тэгов.

С помощью разметки HTML, компоненты могут отображать многострочный, многошрифтовый текст с использованием простого форматирования HTML.

Чтобы отобразить форматированный текст, нужно просто указать строку текста HTML, которая начинается с тега <HTML>.

Возможность отображения иконок Swing-компонентами.

Реализация модели MVC (Model-View-Controller) – модели отделения данных от внешнего вида и поведения интерфейса.

Многие Swing-компоненты имеют связанные с ними интерфейсы (Model), которые отвечают за доступ к данным и генерацию событий, связанных с изменением данных.

Библиотека Swing

- Кроссплатформенность.
- Механизм сменного внешнего вида и поведения компонентов (pluggable look and feel).
- Расширяемость.
- Основана на технологии Java Beans.
- Возможность настройки внешнего вида Swing-компонентов.
- Легковесность.
- Двойная буферизация.
- Доступ к Swing-компонентам должен осуществляться в специальном потоке Event Dispatch Thread (EDT).
- Поддержка технологий для людей с ограниченными возможностями.
- Снабжение компонентов всплывающими подсказками и механизмом управления клавиатурой.
- Возможность настройки текста, отображаемого Swing-компонентами с помощью HTML-тэгов.
- Возможность отображения иконок Swing-компонентами.
- Реализация модели MVC (Model-View-Controller)

Архитектура MVC (модель – вид – контроллер) – это объектно-ориентированная архитектура пользовательских интерфейсов, состоящая из трех частей.

Модель предназначена для хранения, изменения и получения данных графического компонента.

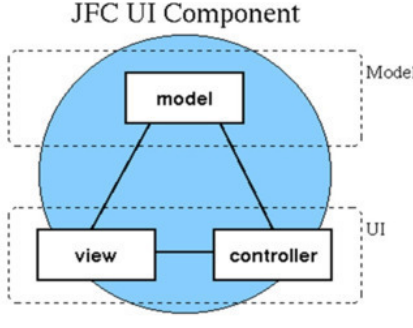
Вид представляет данные на экране.

Контроллер обеспечивает реакцию модели и вида в ответ на действия пользователя.

Архитектура MVC реализована в библиотеке Swing в виде архитектуры с разделенной моделью, состоящей из двух, а не из трех частей.

Вид и контроллер, в архитектуре Swing, объединены вместе в один элемент – представителя пользовательского ин-

терфейса (delegate UI).



Класс графического компонента связывает модель и представителя с помощью менеджера пользовательского интерфейса (UIManager), который определяет внешний вид и поведение интерфейса (Look and Feel).

Именно поэтому библиотека Swing имеет подключаемую архитектуру look-and-feel.

Таким образом, для реализации модели MVC библиотека Swing использует делегирование (delegation) полномочий, назначая в качестве модели данных представителя (delegate) – экземпляр класса с именем вида xxxModel.

Класс библиотеки Swing содержит защищенное или даже закрытое поле model – объект этого класса-модели, и ме-

тод getModel, предоставляющий разработчику доступ к полю model.

Для обеспечения внешнего вида и поведения Swing компонента также используется делегирование полномочий.

Swing класс JComponent содержит защищенное поле ui – экземпляр класса-представителя ComponentUI из пакета javax.swing.plaf, непосредственно отвечающего за вывод изображения на экран в нужном виде.

Класс-представитель содержит методы paint и update, формирующие и обновляющие графику компонента.

Такие представители образуют целую иерархию с общим суперклассом ComponentUI.

Они собраны в пакет javax.swing.plaf и его подпакеты.

В их именах есть буквы UI (User Interface), например, ButtonUI.

Представители класса тоже являются полями класса Swing компонента, и доступ к ним осуществляется методами вида getUI.

При построении графического интерфейса пользователя редко приходится обращаться к моделям и представителям компонента.

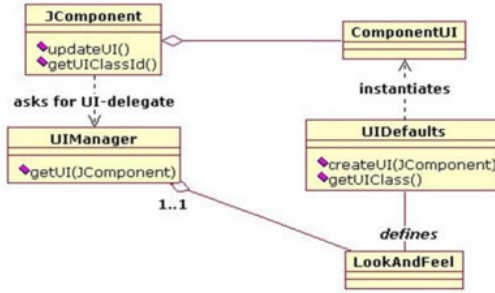
В большинстве случаев достаточно обращаться к методам самого класса компонента.

Если есть желание поменять модель, принятую по умолчанию, можно заменить ее другой моделью, реализовав подходящий интерфейс или расширив существующий класс

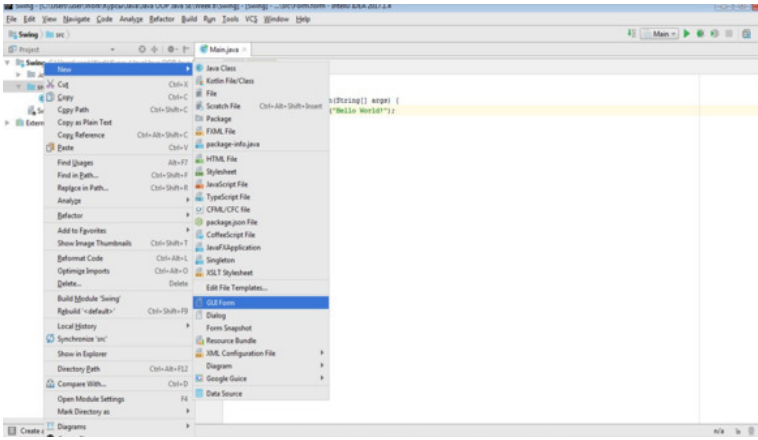
xxxModel.

Новая модель данных устанавливается методом setModel (xxxModel).

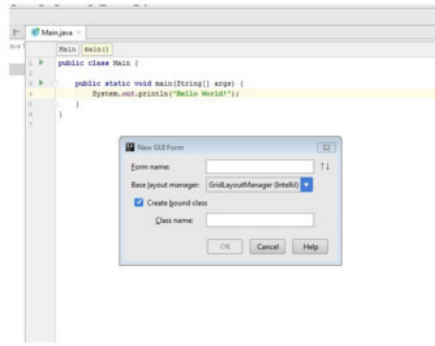
Для создания графического интерфейса пользователя с использованием библиотеки Swing в среде разработки IntelliJ IDEA, нужно нажать правой кнопкой мышки на пакете приложения и выбрать New – GUI Form.



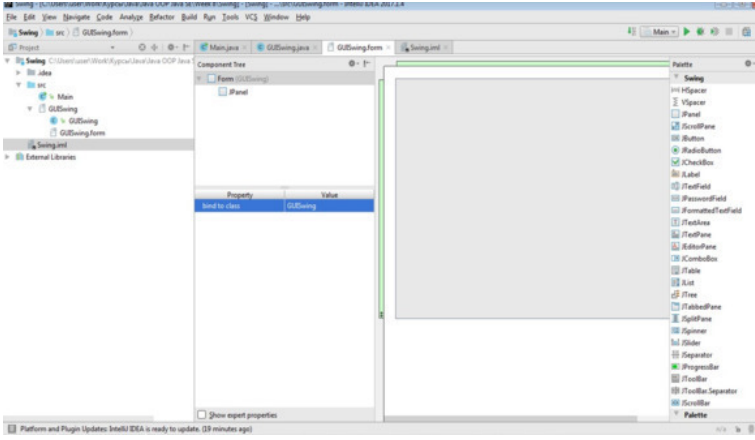
Ввести имя класса и нажать OK.



В результате будет создан Java класс и его XML описание с расширением. form, которое открывается в визуальном редакторе, позволяя визуально создавать и редактировать GUI интерфейс.



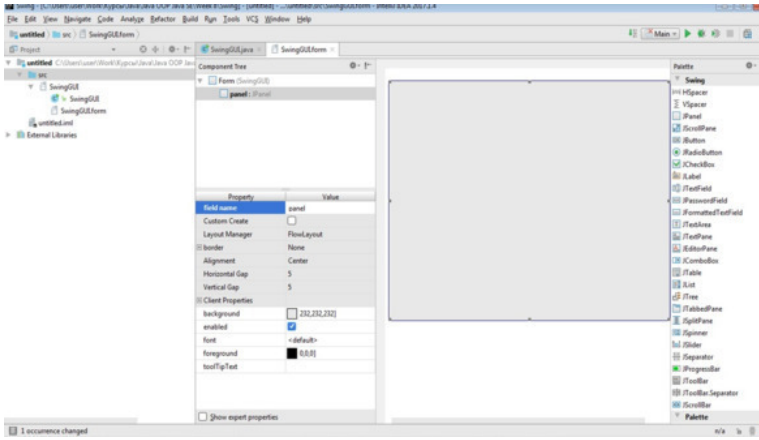
Первоначально форма состоит только из контейнера JPanel.



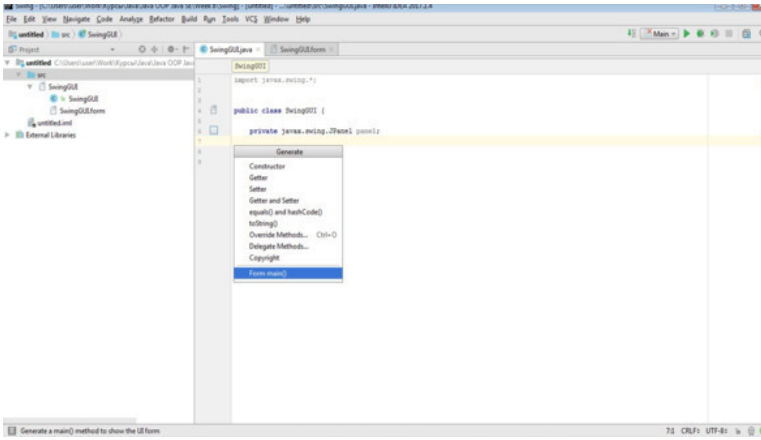
Для дальнейшей работы с GUI интерфейсом, нужно в поле field name ввести имя этого компонента.

При желании можно изменить компоновку компонента.

Далее нужно открыть связанный с формой Java класс и в коде класса нажать правой кнопкой мыши и выбрать Generate – Form main ().



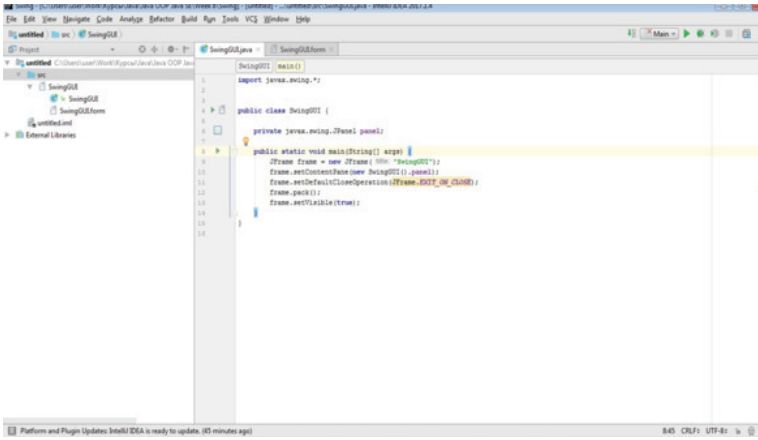
В результате будет сгенерирован код запуска формы.



А именно, будет сгенерирован метод main, в котором будет создано окно верхнего уровня JFrame.

И в это окно будет добавлена панель формы.

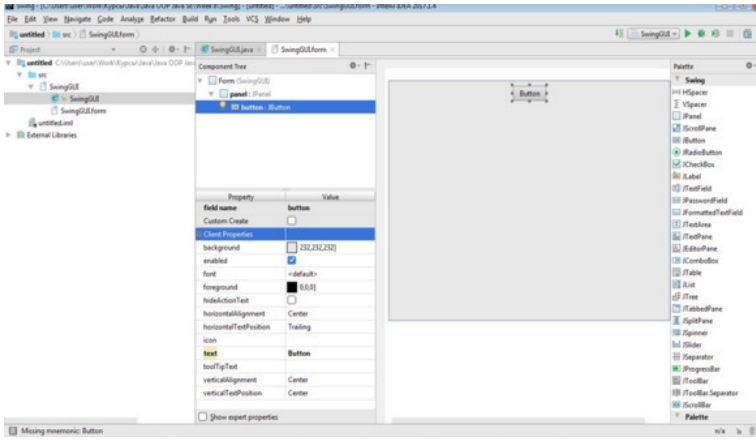
Теперь можно вернуться в визуальный редактор и добавлять компоненты в форму.



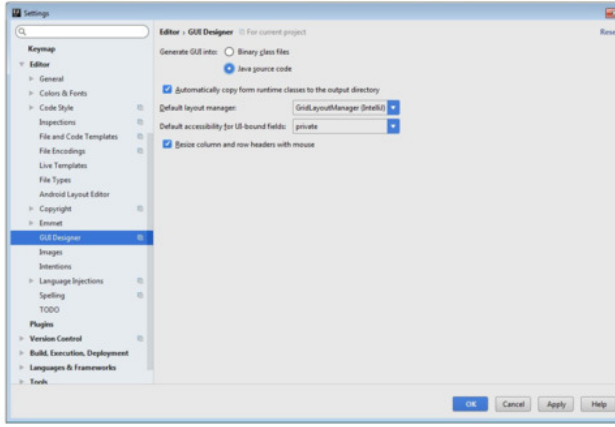
При этом по умолчанию в Java класс будут добавляться только соответствующие поля.

Весь остальной код будет генерироваться сразу в байт-код при компиляции на основании XML файла описания формы.

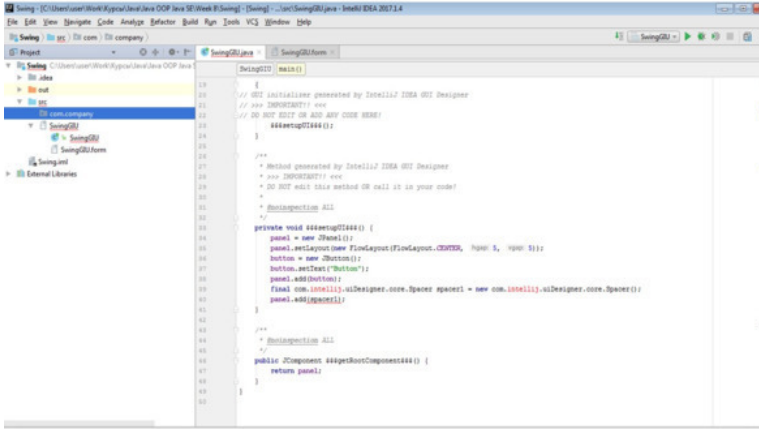
Изменить это можно в настройках File | Settings | Editor | GUI Designer, выбрав опцию Java source code.



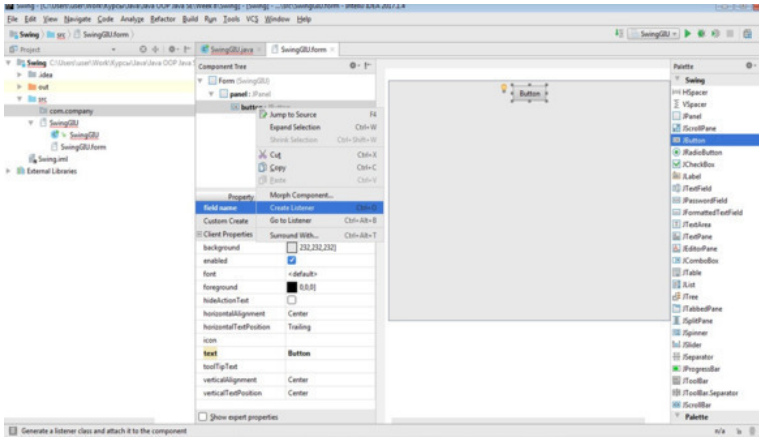
При этом при компиляции будет генерироваться исходный код, который менять нельзя.



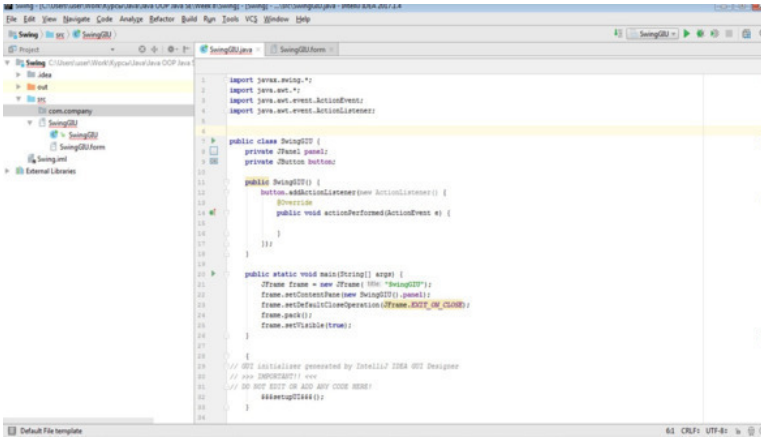
Для добавления слушателя к компоненту, нужно нажать правой кнопкой мыши на компоненте и выбрать Create Listener.



В результате в конструкторе класса будет сгенерирован слушатель действий компонента, в который можно добавлять нужный код.

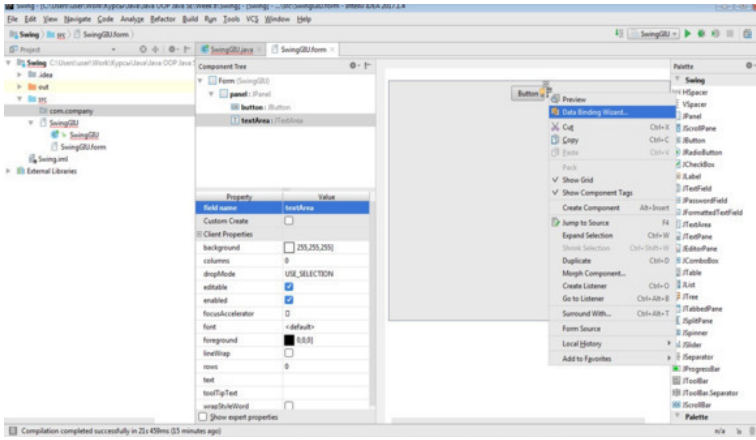


Если у вас есть поля в Swing компоненте, которые вы хотите связать с данными класса JavaBean, нужно нажать правой кнопкой мыши на компоненте и выбрать Data Binding Wizard.



Дальше следовать подсказкам.

В результате будет сгенерирован Plain Old Java Object класс.



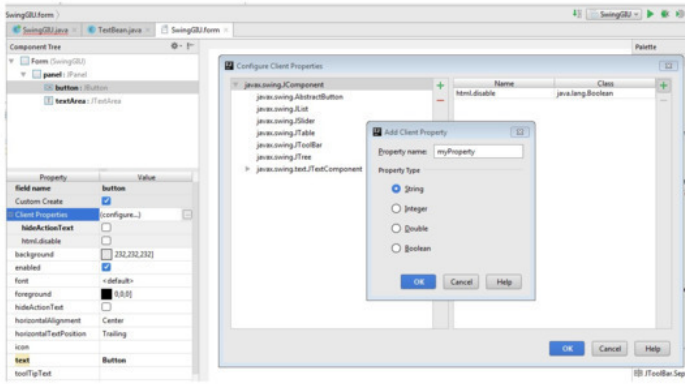
И будет сгенерирован код, связывающий Plain класс с GUI компонентом.

В GUI компонент можно добавить пользовательское свойство, которое отобразится в визуальном редакторе.

```
public class TextBean {
   private String text;

   public TextBean() {
   }

   public String getText() {
      return text;
   }

   public void setText(final String text) {
      this.text = text;
   }
}
```

```
   public void setData(TextBean data) {
      textArea.setText(data.getText());
   }

   public void getData(TextBean data) {
      data.setText(textArea.getText());
   }

   public boolean isModified(TextBean data) {
      if (textArea.getText() != null ?
!textArea.getText().equals(data.getText()) : data.getText() !=
null)
         return true;
      return false;
   }
```

Для этого нужно нажать на поле Client Properties.



И открыть окно свойств.

В левой панели выберите класс, для которого вы хотите изменить свойства.

С выбранным классом в левой панели нажмите кнопку + на правой панели.

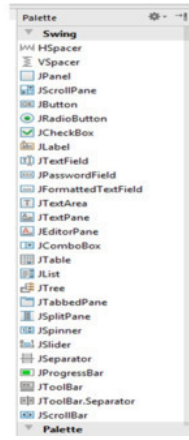
И добавьте свойство.

JButton и JLabel

Графические интерфейсы пользователя Java

Лекция 13
JButton и JLabel

Палитра компонентов позволяет добавить кнопку JButton.



Для кнопки можно изменить цвет фона, шрифт, цвет надписи, выравнивание надписи, саму надпись, текст подсказки при наведении курсора мыши на кнопку, добавить слушателя действий пользователя.

```
JButton cancelButton = new JButton("Cancel", icon);

cancelButton.setHorizontalTextPosition(SwingConstants.LEFT);

cancelButton.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        statusLabel.setText("Cancel Button clicked.");
      }
    });
```

Также можно добавить значок в кнопку.

Компонент JLabel может отображать либо текст, либо изображение, либо и то, и другое.

```
JLabel label  = new JLabel("", JLabel.CENTER);

label.setText("Welcome to Swing Tutorial.");

label.setOpaque(true);
label.setBackground(Color.GRAY);
label.setForeground(Color.WHITE);
```

```
JLabel label = new JLabel();
label.setText("<html><h1>This is </h1><br/><h3>label</h3>");
```

Содержимое метки может выравниваться, с помощью установки вертикального и горизонтального выравнивания в области отображения.

Компонент JLabel используется для отображения одной строки только для чтения.

Текст может быть изменен приложением, но пользователь не может его редактировать напрямую.

С помощью разметки HTML можно создавать многострочную метку, комбинируя шрифты и цвета.

JColorChooser

Графические интерфейсы пользователя Java

Лекция 14
JColorChooser

Компонент JColorChooser используется для создания диалогового окна выбора цвета.

```
JFrame mainFrame = new JFrame("Java Swing Examples");
mainFrame.setSize(400,400);
mainFrame.setLayout(new FlowLayout());
mainFrame.addWindowListener(new WindowAdapter() {
   public void windowClosing(WindowEvent windowEvent){
     System.exit(0);
   } });
JPanel controlPanel = new JPanel();
controlPanel.setLayout(new FlowLayout());
JButton chooseButton = new JButton("Choose Background");
chooseButton.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
     Color backgroundColor = JColorChooser.showDialog(mainFrame, "Choose background color", Color.white);

     if(backgroundColor != null){
       controlPanel.setBackground(backgroundColor);
       mainFrame.getContentPane().setBackground(backgroundColor);
     }
   }
});
controlPanel.add(chooseButton);
mainFrame.add(controlPanel);
mainFrame.setVisible(true);
```

Использовать компонент JColorChooser очень просто.

Вызываем статический метод showDialog, который отображает модальное диалоговое окно выбора цвета.

Это диалоговое окно блокирует приложение до тех пор, пока оно не будет закрыто.

В метод showDialog передается цвет выбора по умолчанию.

Пользователь может в диалоге изменить этот цвет выбора.

После закрытия диалогового окна метод showDialog возвращает цвет выбора, который далее может быть использован в приложении.

JCheckBox, JRadioButton, JToogleButton

Графические интерфейсы пользователя Java

Лекция 15
JCheckBox, JRadioButton, JToggleButton

Компонент JCheckBox используется для создания флажка.

```
JCheckBox chkApple = new JCheckBox("Apple");

chkApple.setMnemonic(KeyEvent.VK_C);

chkApple.addItemListener(new ItemListener() {

    public void itemStateChanged(ItemEvent e) {
      statusLabel.setText("Apple Checkbox: "
        + (e.getStateChange()==1?"checked":"unchecked"));
    }

  });
```

Щелчок мыши на флажке изменяет его состояние с «on» на «off» и наоборот.

Методом setMnemonic можно определить горячие клавиши для этой кнопки.

А флажок является кнопкой, так как наследует от класса AbstractButton.

Компонент JRadioButton используется для создания переключателя.

```
JRadioButton radApple = new JRadioButton("Apple", true);
JRadioButton radMango = new JRadioButton("Mango");
JRadioButton radPeer = new JRadioButton("Peer");

    radApple.addItemListener(new ItemListener() {
      public void itemStateChanged(ItemEvent e) {
        statusLabel.setText("Apple RadioButton: "
            + (e.getStateChange()==1?"checked":"unchecked"));
      }
    });

    //Group the radio buttons.
    ButtonGroup group = new ButtonGroup();
    group.add(radApple);
    group.add(radMango);
    group.add(radPeer);

    controlPanel.add(radApple);
    controlPanel.add(radMango);
    controlPanel.add(radPeer);

    mainFrame.add(controlPanel);
    mainFrame.setVisible(true);
```

Он используется для выбора только одной опции из нескольких вариантов.

Первоначальный выбор радио кнопки указывается в конструкторе класса, с помощью значения аргумента true.

Для того чтобы выбор кнопки был единственным, кнопки нужно объединить в группу ButtonGroup.

Если это не сделать, выбор будет множественным.

Компонент JToggleButton реализует функции переключения, которые наследуются компонентами JCheckBox и JRadioButton.

```
final JToggleButton button = new JToggleButton("ON");

frame.add(button);

button.addItemListener(new ItemListener() {
 public void itemStateChanged(ItemEvent e) {

  if (button.isSelected())
       button.setText("OFF");
     else
       button.setText("ON");
  }
    });
```

Компонент JToggleButton используется для создания кнопок с двумя состояниями – включен и выключен.

Компонент JToggleButton создается с помощью конструктора класса, в котором можно указать надпись кнопки, иконку и состояние.

Проверить нажата ли кнопка, можно с помощью метода isSelected.

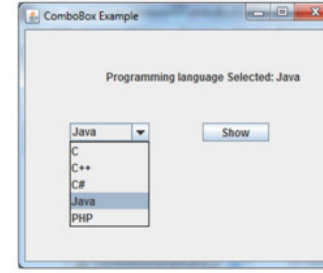
JComboBox

Графические интерфейсы пользователя Java

Лекция 16
JComboBox

Компонент JComboBox объединяет кнопку или редактируемое поле и раскрывающийся список.

```
String languages[]={"C","C++","C#","Java","PHP"};

final JComboBox cb=new JComboBox(languages);

cb.setBounds(50, 100,90,20);

. . .
String data = cb.getItemAt(cb.getSelectedIndex());
```



По умолчанию компонент представляет собой не редактируемое поле, в котором есть кнопка и раскрывающийся список значений.

Создать экземпляр раскрывающегося списка можно конструктором по умолчанию JComboBox, а затем вносить в него элементы методами addItem (Object) и insertItemAt (Object, int).

Однако удобнее предварительно создать массив или вектор, содержащий элементы, и внести его в список сразу же при его создании конструктором JComboBox (Object []) или JComboBox (Vector).

Получить элемент списка, который выбрал пользователь, можно с помощью метода getSelectedIndex или getSelectedItem.

Вместо строк, список можно составить из изображений, или других объектов.

```
Object[] data = {new ImageIcon("apple.gif"),
                 new ImageIcon("grape.gif"),
                 new ImageIcon("pear.gif")};

JComboBox fruits = new JComboBox(data);
```

Список становится редактируемым с помощью вызова метода setEditable (true).

```
String languages[]={"C","C++","C#","Java","PHP"};
final JComboBox cb=new JComboBox(languages);
cb.setEditable(true);
cb.setBounds(50, 100,90,20);
cb.addActionListener(new ActionListener() {
   private int selectedIndex = -1;
   @Override
   public void actionPerformed(ActionEvent e) {
      int index = cb.getSelectedIndex();
      if(index >= 0) {
         selectedIndex = index;
      }
      else if("comboBoxEdited".equals(e.getActionCommand())) {
         Object data = cb.getSelectedItem();
         cb.removeItemAt(selectedIndex);
         cb.insertItemAt(data,selectedIndex);
      }
      String data = "Programming language Selected: "
            + cb.getItemAt(cb.getSelectedIndex());
      label.setText(data);
   }
});
```

Здесь показан пример, в котором пользователь может отредактировать выбранное поле, и оно будет сохранено в списке.

Редактирование выбранного элемента списка не приводит к изменению этого элемента в списке, а влияет только на объект, возвращаемый методом getSelectedItem.

Поэтому здесь отредактированный элемент сохраняется в списке программным способом.

Здесь нужно учитывать, что слушатель ActionListener получает событие ActionEvent, когда был сделан выбор.

И если поле со списком доступно для редактирования, тогда событие ActionEvent также будет сгенерировано, когда закончится редактирование.

Таким образом, обработчик ActionListener вызывается

два раза.

Поэтому сначала мы запоминаем индекс выбранного элемента, а затем при получении команды редактирования, сохраняем новое значение по указанному индексу.

Для изображения элементов списка используется объект, реализующий интерфейс ListCellRenderer.

```
cb.setRenderer(new CheckComboRenderer());

class CheckComboRenderer implements ListCellRenderer {
 JCheckBox checkBox;

 public CheckComboRenderer() {
  checkBox = new JCheckBox();
 }

 public Component getListCellRendererComponent(JList list, Object value,
    int index, boolean isSelected, boolean cellHasFocus) {

  checkBox.setText((String)value);
  checkBox.setSelected(isSelected);
  checkBox.setBackground(isSelected ? Color.red : Color.white);
  checkBox.setForeground(isSelected ? Color.white : Color.black);
  return checkBox;
 }
}
```

Этот объект последовательно выводит элементы, переходя от одного элемента к другому.

С помощью такого объекта устраняется необходимость создания своего графического объекта для каждого элемента списка, что значительно экономит ресурсы.

Метод getListCellRendererComponent интерфейса ListCellRenderer отвечает за формирование компонента

и размещения в нем текущего элемента списка value, имеющего порядковый номер index.

Полученный таким образом компонент затем выводится на экран своим методом paint.

В библиотеке Swing интерфейс ListCellRenderer реализован классами BasicComboBoxRenderer и DefaultListCellRenderer, расширяющими класс JLabel.

Именно потому, что выводом элементов фактически занимается класс JLabel, можно использовать в элементах списка текст или изображение.

Здесь показана пользовательская реализация интерфейса ListCellRenderer, которая выводит флажки в качестве элементов списка.

JScrollPane

Компонент JScrollPane является контейнером, который может содержать только один компонент и используется для создания прокручиваемого представления компонента.

Графические интерфейсы пользователя Java

Лекция 17
JScrollPane

```
JTextArea textArea = new JTextArea(20, 20);

JScrollPane scrollableTextArea = new JScrollPane(textArea);

    scrollableTextArea.setHorizontalScrollBarPolicy(
JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);

scrollableTextArea.setVerticalScrollBarPolicy(
JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
```

Когда размер экрана ограничен, используется панель прокрутки для отображения большого компонента или компонента, размер которого может изменяться динамически.

Полосы прокрутки могут всегда отображаться на экране, отображаться при необходимости или не отображаться вообще. Это определяется методами setVerticalScrollBarPolicy и setHorizontalScrollBarPolicy.

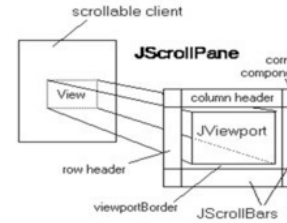
На самом деле кроме своего содержимого и двух полос прокрутки эта панель может содержать еще шесть компонентов: заголовок, столбец слева, и четыре компонента по углам.

Эти компоненты устанавливаются методами setColumnHeaderView, setRowHeaderView и setCorner соответственно.

Компонент помещается на панель прокрутки сразу же при

ее создании конструктором класса JScrollPane или позднее методом setViewportView.

На самом деле компонент отображается в панели прокрутки в окне содержимого JViewport, которое содержит панель JScrollPane.




```
JButton jbutton = new JButton();
jscrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
jscrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

jscrollPane.setViewportBorder(new LineBorder(Color.RED));

jscrollPane.getViewport().add(jbutton, null);
```

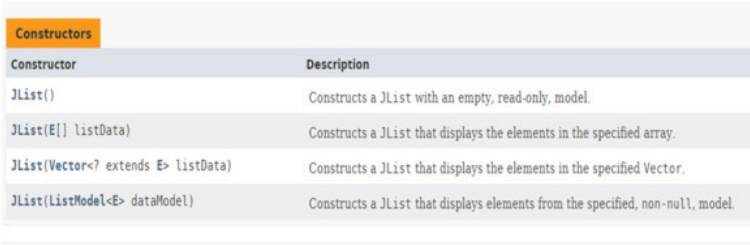
Это окно можно получить методом getViewport класса JScrollPane, а затем можно добавить в него компонент обычным методом add.

JList

Компонент JList отображает список объектов и позволяет пользователю выбрать один или несколько элементов в списке.

Графические интерфейсы
пользователя Java

Лекция 18
JList



Constructors

Constructor	Description
`JList()`	Constructs a JList with an empty, read-only, model.
`JList(E[] listData)`	Constructs a JList that displays the elements in the specified array.
`JList(Vector<? extends E> listData)`	Constructs a JList that displays the elements in the specified Vector.
`JList(ListModel<E> dataModel)`	Constructs a JList that displays elements from the specified, non-null, model.

Список можно создать с помощью конструктора по умолчанию JList, создающего пустой список.

Можно создать список с заданным массивом объектов конструктором JList (Object []), или с заданным вектором при помощи конструктора JList (Vector) или с определенной заранее моделью JList (ListModel).

Это делается так же, как и при создании выпадающего списка JComboBox.

Чтобы ограничить число видимых на экране строк, но при этом отобразить весь список, следует поместить список в панель JScrollPane.

```
final DefaultListModel fruitsName = new DefaultListModel();

fruitsName.addElement("Apple");
fruitsName.addElement("Grapes");
fruitsName.addElement("Mango");
fruitsName.addElement("Peer");

final JList fruitList = new JList(fruitsName);
fruitList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
fruitList.setLayoutOrientation(JList.HORIZONTAL_WRAP);
fruitList.setSelectedIndex(0);
fruitList.setVisibleRowCount(-1);

JScrollPane fruitListScrollPane = new JScrollPane(fruitList);
```

При этом метод setVisibleRowCount задает число видимых на экране строк.

Значение по умолчанию равно 8.

Вызов этого метода с отрицательным значением эквивалентно значению нуль.

Метод setLayoutOrientation определяет компоновку элементов списка.

Константа VERTICAL определяет расположение элементов вертикально в одном столбце.

Константа HORIZONTAL_WRAP определяет расположение элементов горизонтально, при необходимости с переносом их на новую строку.

Константа VERTICAL_WRAP определяет расположение элементов по вертикали, при необходимости с переносом их

в новый столбец.

В сочетании с вызовом метода setLayoutOrientation, вызов метода setVisibleRowCount (-1) делает список отображающим максимальное количество элементов в доступном пространстве на экране.

Методами setFixedCellHeight и setFixedCellWidth можно установить высоту и ширину отображаемых строк списка.

Так же, как и в раскрывающийся список JComboBox, в список JList можно добавлять не только текст, но и изображения.

По умолчанию в списке можно выбрать любое число любых элементов, держа нажатой клавишу <Ctrl>.

После применения метода setSelectionMode с константой SINGLE_SELECTION, в списке можно будет выбрать только один элемент.

Также, как и для раскрывающегося списка JComboBox, для списка JList, можно определить пользовательский объект ListCellRenderer, и отображать в списке любые компоненты, а не только строки и изображения.

Список генерирует события выбора списка при каждом изменении выбора.

```
ListSelectionListener listSelectionListener = new ListSelectionListener() {

 public void valueChanged(ListSelectionEvent listSelectionEvent) {
  System.out.println("First index: " + listSelectionEvent.getFirstIndex());
  System.out.println(", Last index: " + listSelectionEvent.getLastIndex());
  boolean adjust = listSelectionEvent.getValueIsAdjusting();

  System.out.println(", Adjusting?" + adjust);
  if (!adjust) {
   JList list = (JList) listSelectionEvent.getSource();
   int selections[] = list.getSelectedIndices();
   Object selectionValues[] = list.getSelectedValues();
   for (int i = 0, n = selections.length; i < n; i++) {
    if (i == 0) {
     System.out.println(" Selections: ");
    }
    System.out.println(selections[i] + "/" + selectionValues[i] + " ");
   }
  }
 }
};
jlist.addListSelectionListener(listSelectionListener);
```

Вы можете обработать эти события, добавив в список слушатель с помощью метода addListSelectionListener.

Слушатель выбора списка должен реализовать один метод valueChanged.

Метод getValueIsAdjusting возвращает true, если пользователь все еще выбирает в списке.

Так как нам нужен только конечный результат действия пользователя, метод valueChanged делает что-либо, только если getValueIsAdjusting возвращает false.

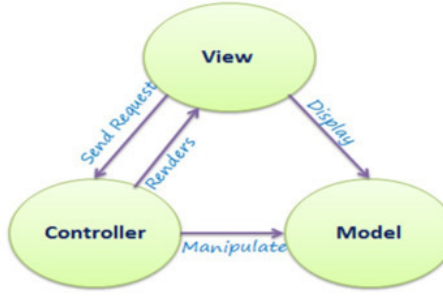
В этом примере описан случай, когда список допускает множественный выбор, поэтому метод valueChanged обрабатывает массив индексов выбранных пользователем элементов списка.

Архитектура Model-View-Controller

Графические интерфейсы пользователя Java

Лекция 19
Архитектура MVC

Теперь, когда мы уже рассмотрели ряд Swing компонентов, давайте вернемся к архитектуре model-view-controller (MVC).



MVC – это общий подход к построению графических приложений.

Суть идеи заключается в том, чтобы низлежащие данные, отображение и логика, которая контролирует данные и отображение, должны быть развязаны.

Эта идея исходит из того, что мы хотели бы иметь более одного способа посмотреть на одни и те же данные.

Например, в большинстве приложений, разработанных для финансовых компаний, существуют разные экраны, которые позволяют просматривать один и тот же торговый процесс по-разному.

Еще один пример из реляционных баз данных.

Там мы можем запускать разные запросы для выбора и упорядочивание данных, но низлежащие данные всегда

одинаковы.

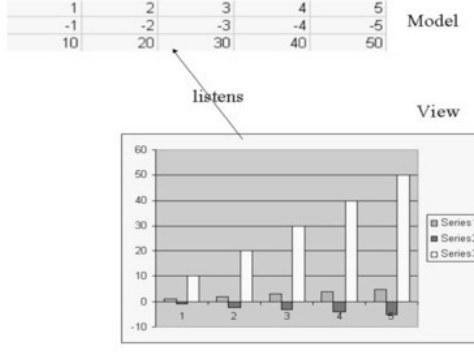
Поэтому архитектура MVC предлагает нам строить графические интерфейсы с помощью моделей, видов и контроллеров.

Модель является источником данных, а вид – это Представление данных.

Модель ничего не знает вообще о Представлениях.

Однако, если Модель является динамичной, тогда она обеспечивает интерфейс прослушивания.

Таким образом, Представление является просто наблюдателем модели.



Когда данные в Модели меняются, она генерирует событие, отражающее изменение.

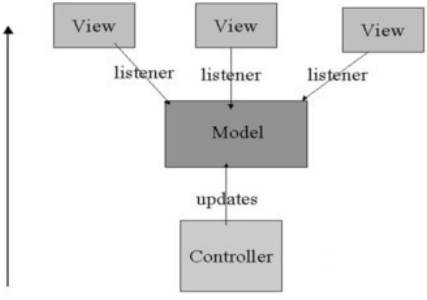
Все Представления, которые слушают эту Модель в качестве наблюдателей, получают обновление и перерисовывают сами себя.

Хорошо, где Контроллер в этой картине?

Контролер отвечает за изменение модели.

С помощью Контроллера, мы можем изменить то, как компонент отвечает на запросы пользователя без изменения его визуального представления.

Обычно Контроллер состоит из графической части и некоторой логики приложения.



Например, вы хотите добавить меню в свой редактор.

Предположим, перед этим, ваш контроллер захватывал события нажатия определенных комбинаций клавиш клави-

атуры и выполнял соответствующие действия на их основе.

Теперь, добавляя меню, вы делаете его частью своего контроллера.

Когда пользователь выбирает элемент меню он действует так, как если бы была бы нажата определенная комбинация клавиш.

Контроллер – это способ, которым пользователь меняет модель.

Контроллер не обновляет Представление, потому что оно автоматически получает обновления, как наблюдатель модели.

Давайте теперь посмотрим на пример MVC.

Рассмотрим модель SimpleStringModel, которая будет иметь один контроллер и несколько видов.

В этой модели у нас есть два метода getString и setString.

```
import java.util.Vector;

public class SimpleStringModel
{
    public String getString() { return sValue; }
    public void setString( String sValue ) {
        // When value changes, notify listeners
        this.sValue = sValue;
        for ( int i = 0; i < vListeners.size(); i++ ) {
            ((IStringModelListener)vListeners.elementAt( i )).
                stringModelChanged( this );
        }
    }

    public void addStringModelListener( IStringModelListener l ) {
        vListeners.addElement( l );
    }

    public void removeStringModelListener( IStringModelListener l ) {
        vListeners.removeElement( l );
    }

    private String sValue;
    private Vector vListeners = new Vector();
}
```

В методе setString мы устанавливаем новое значение поля класса и уведомляем всех слушателей модели, вызывая метод интерфейса, который эти слушатели реализуют.

Соответственно объект модели хранит список своих слушателей.

Представление здесь – это компонент, расширяющий метку.

```
public interface IStringModelListener
{
    public void stringModelChanged( SimpleStringModel model );
}
```

```
import javax.swing.JLabel;

public class SimpleStringView extends JLabel
{
    public SimpleStringView( SimpleStringModel model ) {
     setModel( model );
    }

    public void setModel( SimpleStringModel model ) {
     // If we already had a model, do not listen to it anymore
     if ( this.model != null ) {
         this.model.removeStringModelListener( modelListener );
     }
     this.model = model;
     model.addStringModelListener( modelListener );
     setText( model.getString() );
    }

    // You can defined model listener as an inplace inner class!
    private IStringModelListener modelListener = new IStringModelListener() {
      public void stringModelChanged( SimpleStringModel model ) {
         // Set the value and repaint
         setText( model.getString() );
         repaint();
      }
    };

    private String sValue;
    private SimpleStringModel model;
}
```

Представление имеет метод setModel, в котором Представление становится слушателем Модели.

При изменении модели, автоматически вызывается метод setText метки, который изменяет надпись метки.

Контроллер здесь текстовое поле, в которое пользователь вводит строку текста, и эта строка становится новым значением Модели.

```
import java.awt.event.*;
import javax.swing.JTextField;

public class SimpleStringController extends JTextField
{
   public SimpleStringController( SimpleStringModel model ) {
    this.model = model;
    setText( model.getString() );
    // Listen to the Text Field for action events
    // and set the value on the model
    addActionListener( new ActionListener() {
        public void actionPerformed( ActionEvent ev ) {
            SimpleStringController.this.model.setString( getText() );
        }
    } );
   }

   private SimpleStringModel model;
}
```

Этот базовый пример иллюстрирует, как реализуется архитектура MVC.

Еще одна интересная и очень полезная функция, которую мы получаем, когда используем MVC.

Предположим, что допустимы не все значения, которые пользователь может ввести в текстовое поле.

Путем выброса исключения в методе set модели мы можем запретить изменение представления.

```
public class SimpleStringModel
{
    public String getString() { return sValue; }
    public void setString( String sValue ) throws Exception {
        // When value changes, notify listeners
        if ( sValue.length() < 5 ) {
            throw new Exception( "Length of String must me at least 5");
        }

        this.sValue = sValue;
        for ( int i = 0; i < vListeners.size(); i++ ) {
            ((IStringModelListener)vListeners.elementAt( i )).
                stringModelChanged( this );
        }
    }
```

Swing реализация MVC объединяет Controller и View представление.

На самом деле это не очень сложно сделать.

```
public class SimpleStringControllerView extends JTextField
{
   public SimpleStringControllerView( SimpleStringModel model ) {
    setModel( model );
    // Here is the code for controller
    // Listen to the Text Field for action events
    // and set the value on the model
    addActionListener( new ActionListener() {
        public void actionPerformed( ActionEvent ev ) {
            try {
                SimpleStringControllerView.this.model.setString( getText() );
            } catch ( Exception e ) {}
            // Here is the round trip, no matter what -
            // update from the model
            SimpleStringControllerView.this.setText(
                SimpleStringControllerView.this.model.getString() );
        }
    } );
   }

   public void setModel( SimpleStringModel model ) {
    // If we already had a model, do not listen to it anymore
    if ( this.model != null ) {
        this.model.removeStringModelListener( modelListener );
    }
    this.model = model;
    model.addStringModelListener( modelListener );
    setText( model.getString() );
   }

   // You can defined model listener as an inplace inner class!
   private IStringModelListener modelListener = new IStringModelListener() {
     public void stringModelChanged( SimpleStringModel model ) {
        // Set the value and repaint
        setText( model.getString() );
        repaint();
     }
   };

   private String sValue;
   private SimpleStringModel model;
}
```

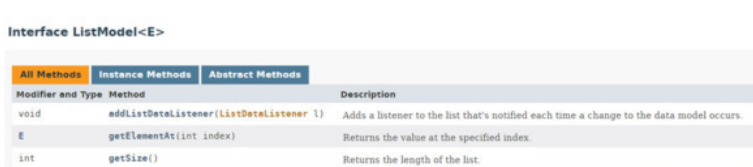
Вы просто размещаете функции View представления и Controller в одном классе.

В предыдущем примере у нас было два графических компонента – один для Представления, а второй для Контоллера.

Здесь у нас один графический компонент, который при взаимодействии с пользователем меняет модель.

При этом этот же компонент становится слушателем модели, перерисовывая себя при изменении модели.

Давайте посмотрим, как реализована архитектура MVC в Swing на примере списка.



Interface ListModel<E>

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method	Description
void	addListDataListener(ListDataListener l)	Adds a listener to the list that's notified each time a change to the data model occurs.
E	getElementAt(int index)	Returns the value at the specified index.
int	getSize()	Returns the length of the list.
void	removeListDataListener (ListDataListener l)	Removes a listener from the list that's notified each time a change to the data model occurs.

Давайте посмотрим на интерфейс ListModel, представляющий модель данных списка.

Во-первых, этот интерфейс легковесный, так как в нем нет ссылки на сам список.

Во-вторых, этот интерфейс присоединяет слушателя модели, как задумано в MVC.

И есть способ получения данных модели, с помощью методов size/get, что гораздо лучше, чем использование метода, например, возвращающего массив данных, так как при использовании методов size/get не занимается память под массив данных, не происходит копирование данных.

Давайте посмотрим на этот пример модели списка, в которой низлежащие данные изменяются динамически.

```
public class DynamicModelDemo
{
    public static void main( String[] args ) {
        JFrame f = new JFrame("List Model Demo");
        JList l  = new JList();
        // Create list Model and add entries to it
        DefaultListModel lm = new DefaultListModel();
        // set the model on the list
        l.setModel( lm );

        // add list to the frame
        f.getContentPane().setLayout( new BorderLayout() );
        f.getContentPane().add( l );

        f.setSize( f.getPreferredSize() );
        f.setVisible( true );

        Thread runner = new Thread( new ListDataGenerator( lm ) );
        runner.start();
    }

    static class ListDataGenerator implements Runnable {
        ListDataGenerator( DefaultListModel model ) {
            this.model = model;
        }
        public void run() {
            while ( true ) {
                if ( random.nextBoolean() ) {
                    // Add
                    model.addElement( new Integer( (int)(random.nextDouble()*100) ) );
                } else {
                    int idx = (int)(random.nextDouble()*(model.getSize()-1));
                    if ( idx > 0 && idx < model.getSize() ) {
                        model.remove( idx );
                    }
                }
                try {
                    Thread.sleep( 2000 );
                } catch ( InterruptedException e ) {}
            }
        }
        DefaultListModel model;
        Random random = new Random( System. currentTimeMillis() );
    }
}
```

Часто бывает, что данные, которые предоставляются пользователю, меняются, и экран необходимо обновлять.

Так как модель данных ListModel, как и все модели MVC, добавляет в качестве слушателя список, список на экране уведомляется каждый раз, когда происходят изменения модели.

Таким образом, списки просто перерисовываются, чтобы отражать изменения.

В этом примере у нас есть список, который отображает список случайно генерируемых целых чисел.

Теперь, еще одной очень полезной концепцией является отфильтрованная модель.

```
public class FilteredListModel implements ListModel
{
    public FilteredListModel( ListModel lm, IBooleanCriteria bc ) {
        setCriteria( bc );
        setModel( lm );
    }

    // Loop through and compute the number of elements
    // which satisfy given criteria
    public int getSize() {
        if ( bc == null ) return lm.getSize();

        int iCount = 0;
        for ( int i = 0; i < lm.getSize(); i++ ) {
            if ( bc.satisfies( lm.getElementAt( i ) ) ) {
                iCount++;
            }
        }
        return iCount;
    }

    public Object getElementAt( int i ) {
       if ( bc == null ) return lm.getElementAt( i );

       // Loop through counting elements which satisfy criteria
       // and return the appropriate one
       int iCount = 0;
       for ( int j = 0; j < lm.getSize(); j++ ) {
            Object oElement = lm.getElementAt( j );
            if ( bc.satisfies( oElement ) ) {
                if ( iCount == i ) return oElement;
                else iCount++;
            }
       }
       return null;
    }

    public void setCriteria( IBooleanCriteria bc ) {
        this.bc = bc;
        fireChanged();
    }
```

Часто бывает так, что мы хотим создать представление, содержащее только подмножество данных.

В этом случае мы используем интерфейс, который принимает каждый элемент данных модели, и проверяет его на соответствие определенному критерию.

Таким образом, происходит фильтрация данных модели.

Еще одна полезная концепция – это слияние моделей.

```
public class MergedListModel implements ListModel
{
    public MergedListModel( ListModel firstModel, ListModel secondModel ) {
        setFirstModel( firstModel );
        setSecondModel( secondModel );
    }

    public int getSize() {
        int iSize1 = ( firstModel == null ) ? 0 : firstModel.getSize();
        int iSize2 = ( secondModel == null )?  0 : secondModel.getSize();
        return iSize1 + iSize2;
    }

    public Object getElementAt( int i ) {
       int iSize1 = ( firstModel == null ) ? 0 : firstModel.getSize();
       int iSize2 = ( secondModel == null ) ? 0 : secondModel.getSize();

        if ( iSize1 > i ) return firstModel.getElementAt( i );
        return secondModel.getElementAt( i - iSize1 );
    }

    public synchronized void setFirstModel( ListModel firstModel ) {
        if ( this.firstModel != null ) {
            // Detach listener
            this.firstModel.removeListDataListener( modelListener );
        }
        this.firstModel = firstModel;
        firstModel.addListDataListener( modelListener );
        fireChanged();
    }

    public synchronized void setSecondModel( ListModel secondModel ) {
        if ( this.secondModel != null ) {
            // Detach listener
            this.secondModel.removeListDataListener( modelListener );
        }
        this.secondModel = secondModel;
        secondModel.addListDataListener( modelListener );
        fireChanged();
    }
```

В этом случае есть модель, которая принимает две модели в качестве аргумента и представляет их как одну.

Опять же это делается с помощью методов size/get без копирования данных исходных моделей.

Таким образом, есть много интересных вещей, которые мы можем делать с моделями.

Помимо отдельного класса модели, некоторые Swing компоненты также используют отдельный класс для рендеринга и просмотра.

Например, JList позволяет пользователю определить ListCellRenderer, который является небольшим классом, который заботится о том, как конкретный элемент списка будет визуализирован.

Этот интерфейс имеет только один метод

getListCellRendererComponent.

```
static class Student {
    public Student( String sFirstName, String sLastName, int ssn ) {
        this.sFirstName = sFirstName;
        this.sLastName = sLastName;
        this.ssn = ssn;
    }
    public String getFirstName()   { return sFirstName; }
    public String getLastName()    { return sLastName; }
    public int    getSSN()         { return ssn; }

    private String sFirstName, sLastName;
    private int ssn;
}
```

```
// We extend default list cell renderer, because
// it takes care of painting focus, etc.
static class StudentListCellRenderer extends DefaultListCellRenderer {
    public Component getListCellRendererComponent(JList list, Object value,
                          int index, boolean isSelected, boolean cellHasFocus)
    {
        // Cast to Student!
        Student student = (Student)value;
        String sValue = student.getFirstName() + " " +
                        student.getLastName() + " : " +
                        student.getSSN();
        return super.getListCellRendererComponent( list, sValue,
                                index, isSelected, cellHasFocus );
    }
}
```

Каждый раз, когда список себя перерисовывает, он запрашивает средство визуализации ячейки ListCellRenderer.

Это полезно по нескольким причинам.

Во-первых, обычно данные, которые вы хотите отобразить в списке, не хранятся в виде строк.

И список не знает, как отобразить произвольный объект как String.

Вы можете думать о CellRenderer как об адаптере, который знает, как адаптировать объект к строке.

Как вы видите в этом примере, фактический список содержит учеников, а не строки, которые отображаются.

```
public static void main( String[] args ) {
    JFrame f = new JFrame("Student List Renderer Demo");
    JList l  = new JList();
    // Create list Model and add entries to it
    DefaultListModel lm = new DefaultListModel();
    lm.addElement( new Student( "Al", "Gore", 333444555 ) );
    lm.addElement( new Student( "George", "Bush", 555444333 ) );
    lm.addElement( new Student( "George", "Gore", 444555333 ) );
    lm.addElement( new Student( "Al", "Bush", 444555333 ) );

    // set the model on the list
    l.setModel( lm );
    l.setCellRenderer( new StudentListCellRenderer() );

    // add list to the frame
    f.getContentPane().setLayout( new BorderLayout() );
    f.getContentPane().add( l );

    f.setSize( f.getPreferredSize() );
    f.setVisible( true );
}
```

Предположим, мы используем этот список, чтобы пользователи могли выбирать в нем студентов.

Это означает, что, когда пользователь выбирает строку на экране, тогда выбранное значение списка фактически должно быть объектом Student, а не строкой.

Для этого все, что нам нужно сделать, это написать адаптер ListModel, который переводит интерфейс ListModel в интерфейс реального источника данных.

Здесь реальный источник данных – это список java.util.List.

```
static class ListToListModelAdapter implements ListModel {
    ListToListModelAdapter( java.util.List list ) {
        this.list = list;
    }
    public int getSize()                    { return list.size(); }
    public Object getElementAt( int idx )   { return list.get( idx ); }

    // No point, because list
    public void addListDataListener( ListDataListener l ) {
        lListeners.add( l );
    }

    public void removeListDataListener( ListDataListener l  ) {
        lListeners.remove( l );
    }

    // This should be called by whoever updates the actual list
    public void update() {
        for ( int i = 0; i < lListeners.size(); i++ ) {
            ((ListDataListener)lListeners.get( i )).contentsChanged( changeEvent );
        }
    }

    private java.util.List list, lListeners = new java.util.ArrayList();
    private ListDataEvent changeEvent = new ListDataEvent( this,
                                    ListDataEvent.CONTENTS_CHANGED, -1, -1 );
}
```

И этот адаптер использует этот список в своих методах size/get.

Здесь видно, что не происходит дублирование данных, и мы работаем с реальными объектами, а не с некоторыми их строковыми представлениями.

Еще одно преимущество, которое мы получаем от использования объекта ListCellRenderer, заключается в том, что мы можем представлять разные ячейки по-разному.

```
static class StudentListCellRenderer implements ListCellRenderer {

    public Component getListCellRendererComponent(JList list, Object value,
                            int index, boolean isSelected, boolean cellHasFocus)
    {
        // Cast to Student!
        Student student = (Student)value;
        String sValue = student.getFirstName() + " " +
                        student.getLastName() + " : " +
                        student.getSSN();
        // Here is what we will do, we will vary colors of cells
        // and switch between buttons and labels
        Component c = null;
        if ( index % 2 == 0 ) {
            c = button;
            button.setText( sValue );
        } else {
            c = label;
            label.setText( sValue );
            label.setOpaque( true );
        }
        c.setBackground( new Color( (int)(random.nextDouble()*255),
                                    (int)(random.nextDouble()*255),
                                    (int)(random.nextDouble()*255 ) ) );
        return c;
    }

    JButton button = new JButton();
    JLabel label = new JLabel();
    private Random random = new Random( System.currentTimeMillis() );
}
```

Здесь мы четные ячейки представляем, как кнопки, а нечетные – как метки.

JTextField и JPasswordField

Графические интерфейсы пользователя Java

Лекция 20
JTextField и JPasswordField

Компонент JTextField позволяет редактировать одну строку текста.

```
final JTextField userText = new JTextField(6);
final JPasswordField passwordText = new JPasswordField(6);

JButton loginButton = new JButton("Login");

loginButton.addActionListener(new ActionListener() {

  public void actionPerformed(ActionEvent e) {
    String data = "Username " + userText.getText();
    data += ", Password: " + new String(passwordText.getPassword());
    statusLabel.setText(data);
  }
});
```

Методом setFont можно изменить шрифт текста.

Методом setText можно установить текст поля, а методом getText получить его.

Выделенный в поле текст можно заменить другим текстом с помощью метода replaceSelection.

Метод setHorizontalAlignment позволяет изменить выравнивание текста по умолчанию по левому краю.

Выключить редактирование текста можно методом setEditable (false).

Метод setCaretColor позволяет изменить цвет курсора.

Позицию курсора можно отследить методом getCaretPosition, или установить методом setCaretPosition.

Переместить курсор можно, выделив таким способом участок текста, методом moveCaretPosition.

Цвет выделенного текста можно задать методом setSelectedTextColor, а цвет фона выделенного текста методом setSelectionColor.

При использовании текстового поля могут возникнуть такие задачи, как создание поля с подсказкой, которая исчезает при вводе текста, или создание текстового поля с автозавершением.

Для решения этих задач придется расширить компонент JTextField, переопределяя его методы focusGained и focusLost, а также реализуя интерфейс DocumentListener с определением метода insertUpdate.

Компонет JPasswordField предназначен для ввода пароля и позволяет редактировать одну строку текста.

Класс JPasswordField расширяет класс JTextField, и отличается от него тем, что в этом поле вместо вводимых символов повторяется один символ, по умолчанию – звездочка.

Звездочку можно заменить другим символом с помощью метода setEchoChar.

Второе отличие заключается в том, что вместо метода getText для получения текста из поля пароля используется метод getPassword, возвращающий массив символов типа char, а не строку.

JFormattedTextField

Графические интерфейсы пользователя Java

Лекция 21
JFormattedTextField

Класс JFormattedTextField расширяет класс JTextField и обеспечивает вывод объекта в текстовую строку.

```
JFormattedTextField ftf = new JFormattedTextField(new Date());
ftf.addActionListener(this);

// . . . . . .

public void actionPerformed(ActionEvent e){
  newDate = (Date)ftf.getValue();
}


JFormattedTextField salaryField  = new JFormattedTextField();
salaryField.setValue(new  Double(12345.98));
```

По умолчанию реализация обеспечивает форматированный вывод объектов типа Number и Date в виде строки.

Метод getValue возвращает объект типа Object, полученный в результате обратного преобразования отредактированной в окне строки в первоначальный объект.

Преобразованием объекта в строку и обратно занимается вложенный в JFormattedTextField абстрактный класс AbstractFormatter.

Библиотека Swing предоставляет реализации этого класса – классы DateFormatter, NumberFormatter и MaskFormatter.

Мы можем создать компонент JFormattedTextField с помощью объекта форматирования.

```
DateFormat dateFormat = new SimpleDateFormat("mm/dd/yyyy");
DateFormatter dateFormatter = new DateFormatter(dateFormat);

JFormattedTextField dobField = new JFormattedTextField(dateFormatter);


NumberFormat numFormat = new DecimalFormat("$#0,000.00");
NumberFormatter numFormatter = new NumberFormatter(numFormat);

JFormattedTextField salaryField = new JFormattedTextField(numFormatter);

MaskFormatter ssnFormatter = null;
JFormattedTextField ssnField = null;
try {
   ssnFormatter = new MaskFormatter("###-##-####");
   ssnField = new JFormattedTextField(ssnFormatter);
}
catch (ParseException e) {
   e.printStackTrace();
}
```

Для этого можно использовать классы DateFormatter, NumberFormatter и MaskFormatter для форматирования даты, числа и строки, соответственно.

В этом примере показано создание объекта форматирования для вывода даты в заданном формате, числа и строки.

JTextArea

Компонент JTextArea представляет многострочную область для отображения обычного текста.

Графические интерфейсы
пользователя Java

Лекция 22
JTextArea

```
JTextArea commentTextArea = new JTextArea("This is a Swing tutorial "
    +"to make GUI application in Java.", 5, 20);

JScrollPane scrollPane = new JScrollPane(commentTextArea);
```


```
commentTextArea.setWrapStyleWord(true);

commentTextArea.setLineWrap(true);
```

Шрифт текста устанавливается методом setFont.

Введенный текст можно получить методом getText.

Установить текст методом setText, или добавить текст методом append, или вставить текст методом insert.

По умолчанию компонент JTextArea не обеспечивает возможность прокрутки большого текста.

Поэтому компонент нужно поместить в контейнер JScrollPane.

При этом следует задать размеры текстовой области – число строк и столбцов, или предпочтительный размер контейнера JScrollPane.

По умолчанию весь текст в области показывается в виде одной строки, выходящей за пределы окна.

Для переноса слова целиком и слов на новую строку нуж-

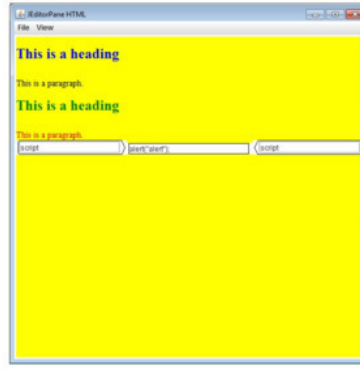
но применить методы setWrapStyleWord (true) и setLineWrap (true).

JEditorPane

Компонент JEditorPane используется для создания окна текстового редактора.

Графические интерфейсы пользователя Java

Лекция 23
JEditorPane



По умолчанию этот компонент может отображать и редактировать простой текст, документ HTML и Rich Text Format (RTF).

В случае HTML, компонент JEditorPane может отображать HTML-документ, соответствующий спецификации Html 3.2 с ограниченной поддержкой css и без поддержки Javascript.

В этом примере мы создаем компонент JEditorPane и помещаем его в окно JFrame.

```
final JFrame myFrame = new JFrame("JEditorPane HTML");
myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
myFrame.setSize(600,600);
final JEditorPane myPane = new JEditorPane();
myPane.setContentType("text/html");
myFrame.setContentPane(myPane);
JMenuBar myBar = new JMenuBar();
JMenu myMenu = new JMenu("File");
JMenuItem myItem = new JMenuItem("Open");
myItem.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
      JFileChooser fileDialog = new JFileChooser();
      int returnVal = fileDialog.showOpenDialog(myFrame);
      if (returnVal == JFileChooser.APPROVE_OPTION) {
         java.io.File file = fileDialog.getSelectedFile();
         try {
            String text = new String(Files.readAllBytes(file.toPath()));
            myPane.setText(text);
         } catch (IOException e1) {
            e1.printStackTrace();
         }
      }        }     });
myMenu.add(myItem);
```

Метод setContentType класса JEditorPane устанавливает тип содержимого для обработки редактором.

При этом вызывается метод getEditorKitForContentType, а затем метод setEditorKit.

Компонент JEditorPane использует реализации EditorKit редакторов для текстового содержимого определенного типа.

Здесь мы устанавливаем отображение и редактирование HTML контента.

Далее мы создаем меню для загрузки контента в компонент JEditorPane.

Контент в виде строки загружается методом setText.

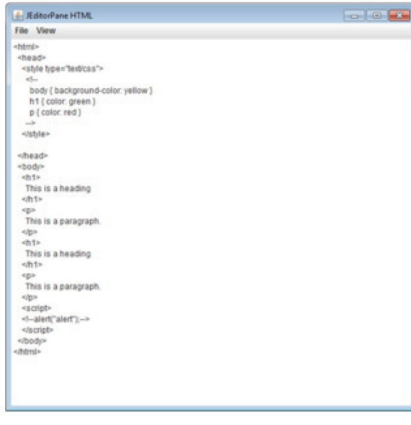
Также можно использовать метод setPage для загрузки контента по URL адресу.

В этом примере мы также создаем меню для переключения типа содержимого с HTML на простой текст и обратно.

```
myMenu = new JMenu("View");
ButtonGroup myGroup = new ButtonGroup();
myItem = new JRadioButtonMenuItem("HTML");
myItem.setSelected(true);
myItem.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
      String   text = myPane.getText();
      myPane.setContentType("text/html");
      myPane.setText(text);
   }      });
myMenu.add(myItem);
myGroup.add(myItem);

myItem = new JRadioButtonMenuItem("Plain");
myItem.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
      String text = myPane.getText();
      myPane.setContentType("text/plain");
      myPane.setText(text);
   }      });
myMenu.add(myItem);
myGroup.add(myItem);
```

Если вы нажмете команду Plain в меню View, содержимое будет отображаться в виде обычного текста, как исходный HTML-код.

При этом надо отметить, что вызов метода setContentType фактически удаляет текущий текстовый контент в области редактора.




```
<!DOCTYPE html>
<html>
<head>
<style>
body {background-color: yellow;}
h1  {color: green;}
p   {color: red;}
</style>
</head>
<body>

<h1 style="color:blue;">This is a heading</h1>
<p style="color:black;">This is a paragraph.</p>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

<script>
alert("alert");
</script>

</body>
</html>
```

Вот почему мы сначала сохраняем содержимое в переменной.

Также при переключении на обычный текст, исходный HTML код модифицируется.

В частности, удаляются inline css стили.

Чтобы сделать JEditorPane доступным только для чтения, нужно использовать метод setEditable (false).

При отображении HTML-документа, компонент JEditorPane может обнаруживать HTML-ссылки и отвечать на клики пользователей.

Чтобы обрабатывать событие click на ссылках, нужно обработать событие HyperlinkEvent, присоединив слушателя методом addHyperlinkListener.

Здесь показан пример реализации обработки гиперссы-

лок.

```
addHyperlinkListener(new HyperlinkListener() {

        public void hyperlinkUpdate(HyperlinkEvent e) {

          if (e.getEventType() == HyperlinkEvent.EventType.ACTIVATED){
            JEditorPane pane = (JEditorPane) e.getSource();
            if (e instanceof HTMLFrameHyperlinkEvent) {
              HTMLFrameHyperlinkEvent evt = (HTMLFrameHyperlinkEvent) e;
              HTMLDocument doc = (HTMLDocument) pane.getDocument();
              doc.processHTMLFrameHyperlinkEvent(evt);
            } else {
              try {
                pane.setPage(e.getURL());
              } catch (Throwable t) {
                t.printStackTrace();
              }
            }
          }
        }
      });
```

Сначала мы проверяем, является ли обрабатываемое событие событием активации, а не событием входа (клика) или выхода (готовности).

Затем мы проверяем, является ли обрабатываемое событие событием активации HTML фрема.

И если это так, тогда мы открываем фрейм с помощью HTMLDocument.

В другом случае, мы извлекаем URL адрес гиперссылки из события и открываем по этому адресу документ.

Обратите внимание, если тип содержимого является HTML документ, тогда относительные ссылки, например, для таких вещей, как изображения, не могут быть разреше-

ны, если не используется тег <base> или не указан абсолютный путь изображения.

```
<!DOCTYPE html>
<html>
<head>
<style>
body {background-color: yellow;}
h1  {color: green;}
p   {color: red;}
</style>
<base href="file:/C:/Users/user/Desktop/" target="_blank">
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

<img src='img.jpg' width=200></img>

</body>
</html>
```

То есть для включения изображения в HTML документ, нужно указать абсолютный путь к этому изображению.

JTextPane

Класс JTextPane – это подкласс класса JEditorPane.

Графические интерфейсы
пользователя Java

Лекция 24
JTextPane

```
final JTextPane myPane = new JTextPane();
myFrame.setContentPane(myPane);

DefaultStyledDocument document = new DefaultStyledDocument();
StyleContext context = new StyleContext();
// build a style
Style style = context.addStyle("test", null);
// set some style properties
StyleConstants.setForeground(style, Color.BLUE);
try {
   document.insertString(0, "Styled string \n", style);
} catch (BadLocationException e) {
   e.printStackTrace();
}
myPane.setStyledDocument(document);
```

Компонент JTextPane дополняет JEditorPane возможностью работать с объектом StyledDocument.

Реализация DefaultStyledDocument позволяет легко добавить новые стили методом addStyle.

Можно задать атрибуты для отдельных символов методом setCharacterAttributes.

Можно задать атрибуты сразу целому элементу методом setParagraphAttributes.

Можно вставить в текущую позицию текста изображение методом insertIcon.

Также можно вставить любой компонент методом insertComponent.

ImageIcon

Многие компоненты Swing, такие как ярлыки, кнопки и панели с вкладками, могут быть декорированы значком – изображением фиксированного размера.

Графические интерфейсы пользователя Java

Лекция 25
ImageIcon

```
Frame f=new Frame();

ImageIcon icon = new ImageIcon(this.getClass().getResource("icon.png"));

f.setIconImage(icon);
```

Значок – это объект, который реализует интерфейс Icon.

Swing обеспечивает реализацию интерфейса Icon – класс ImageIcon, который создает значок из изображения формата GIF, JPEG или PNG.

Надо отметить, что в библиотеке Swing нет класса, заменяющего AWT класс Image, есть только класс ImageIcon, который используется для декорирования компонентов значками.

Здесь показано создание значка из изображения, расположенного в пакете приложения, и установка этого значка на панели главного окна приложения.

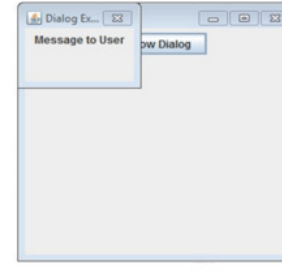
JDialog

Класс JDialog расширяет класс Dialog библиотеки AWT и является «тяжеловесным» компонентом.

Графические интерфейсы пользователя Java

Лекция 26
JDialog

```
JFrame frame = new JFrame("JDialog Example");
frame.setLayout( new FlowLayout() );
frame.setSize(300, 300);

final JDialog d = new JDialog(frame , "Dialog Example", true);
d.setLayout( new FlowLayout() );
d.setSize(100,100);
d.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
d.add( new JLabel ("Message to User"));


JButton b = new JButton ("Show Dialog");
b.addActionListener ( new ActionListener()
{
   public void actionPerformed( ActionEvent e )
   {
      d.setVisible(true);
   }
});

frame.add(b);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
```



Он создает модальные или немодальные диалоговые окна.

Напомним, что модальное окно блокирует графический интерфейс пользователя, пока не будет закрыто.

Каждое диалоговое окно обязательно связано с родительским AWT окном Window, Dialog или Frame.

Диалоговое окно снабжено рамкой и строкой заголовка, в которую помещается строка, записанная в конструкторе.

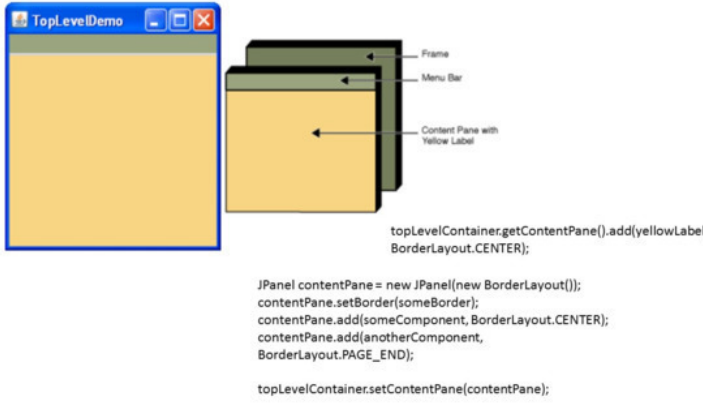
В строке заголовка есть кнопка Закрыть, реакцию на которую, а заодно и реакцию на нажатие комбинации клавиш <Alt> + <F4>, можно установить методом setDefaultCloseOperation.

Модальность окна и его заголовок можно изменить методами setModalityType и setTitle.

По умолчанию диалоговое окно может изменять свои раз-

меры, но это правило можно поменять унаследованным методом setResizable.

Как уже говорилось ранее, Swing предоставляет контейнеры верхнего уровня: JFrame, JDialog и JWindow.



При использовании этих классов, чтобы отображаться на экране, каждый компонент GUI должен быть частью иерархии компонентов, в которой в качестве корня используется контейнер верхнего уровня.

При этом каждый GUI компонент может содержаться только один раз.

Если компонент уже находится в контейнере, и вы пытаетесь добавить его в другой контейнер, компонент будет удален из первого контейнера, а затем добавлен ко второму.

Каждый контейнер верхнего уровня имеет панель содержимого contentPane, которая содержит видимые компоненты контейнера верхнего уровня.

Вы можете дополнительно добавить панель меню в контейнер верхнего уровня.

Панель меню расположена в контейнере верхнего уровня, но вне панели содержимого.

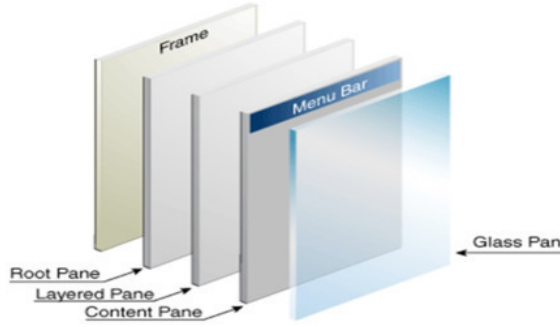
Получить панель содержимого контейнера верхнего уровня можно с помощью метода getContentPane.

Панель содержимого по умолчанию – это простой промежуточный контейнер, который наследуется от JComponent и использует BorderLayout в качестве менеджера компоновки.

Однако можно создать панель JPanel и установить ее как панель содержимого, используя метод setContentPane контейнера верхнего уровня.

Вы можете напрямую добавлять компоненты в контейнер верхнего уровня методом add, при этом они будут добавляться в панель содержимого.

Устройство контейнера верхнего уровня еще более сложное, чем это кажется на первый взгляд.



Каждый контейнер верхнего уровня использует промежуточный контейнер, называемый корневой панелью root pane.

Корневая панель управляет панелью содержимого и панелью меню вместе с несколькими другими контейнерами.

Здесь слоистая панель layered pane содержит панель меню и панель содержимого и обеспечивает упорядочивание других компонентов по оси Z.

Стеклянная панель glass pane часто используется для перехвата входных событий, проходящих через контейнер верхнего уровня, а также может использоваться для рисования поверх нескольких компонентов.

Здесь показан пример создания пользовательского диалога.

```
public class MyJDialog extends JDialog {
    public MyJDialog(JFrame parent, String title, String message, int x, int y) {
      super(parent, title);
      setLocation(x, y);
      JPanel messagePane = new JPanel();
      messagePane.add(new JLabel(message));
      getContentPane().add(messagePane);
      setDefaultCloseOperation(DISPOSE_ON_CLOSE);
      pack();
      setVisible(false);
    }
     public JRootPane createRootPane() {
      JRootPane rootPane = new JRootPane();
      KeyStroke stroke = KeyStroke.getKeyStroke("ESCAPE");
      Action action = new AbstractAction() {
        public void actionPerformed(ActionEvent e) {
          setVisible(false);
          dispose();
        } };
      InputMap inputMap = rootPane.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW);
      inputMap.put(stroke, "ESCAPE");
      rootPane.getActionMap().put("ESCAPE", action);
      return rootPane;
    }  }
```

В конструкторе класса используется панель содержимого для создания интерфейса диалога.

Также переопределяется метод createRootPane, который создает корневую панель, для закрытия диалога при нажатии кнопки «Escape».

В этом методе используется метод getKeyStroke класса KeyStroke для установки нажатия клавиши ESCAPE.

Конец ознакомительного фрагмента.