Отчёт по проекту:

«Создание растрового графического редактора».

Выполнила:

Студентка 2 курса ФНБИК гр. 5103

Кулагина Екатерина.

Содержание.

Растровый редактор, его достоинства и недостатки.

Графический редактор – это программа создания, редактирования и просмотра графических изображений.

Основным элементом растрового изображения является точка. Файл растрового изображения содержит информацию о каждой точке этого изображения. Если изображение экранное, то эта точка называется пикселом. Растровые изображения получают при сканировании, с помощью цифровых фото- и видеокамер. Большинство графических редакторов, предназначенных для работы с растровой графикой, ориентированы не столько на создание изображений, сколько на их обработку.

Файл растровой графики содержит информацию об отдельных точках из которых складывается изображение. Размер изображения будет зависеть от общего числа точек и того, сколько точек размещается в одном дюйме изображения. Величина, показывающая число точек в одном дюйме изображения называется **разрешением изображения** и измеряется в **dpi** (dot per inch). Поскольку файл содержит информацию об определенном числе точек, то, увеличивая или уменьшая размеры изображения на экране или для печати, мы изменяем его разрешение. Однако это изменение может вступить в противоречие с физическими возможностями экрана или принтера отображать точки, то есть с их разрешающей способностью.

Для изображения, занимающего всю поверхность экрана, при разрешении монитора 800x600 пикселов нужно сохранить в файле информацию о $48*10^4$ точек, при этом в одном дюйме изображения будет около 75 точек, т.е. разрешение изображения составит 75 dpi.

Если в файле будут содержаться данные о большем числе точек, изображение уже не поместится на экране. Конечно, его всегда можно сжать до размеров экрана, но при этом часть точек не будет отображаться, и мелких деталей мы не увидим. Однако мы всегда можем посмотреть эти детали, просматривая фрагмент рисунка в увеличенном масштабе.

Если в файле будут содержаться данные о меньшем числе точек, изображение займет лишь часть экрана. Конечно, его можно растянуть до размеров экрана, но при этом каждая точка будет отображаться не одним пикселем, а несколькими. Никакой новой информации мы не получим, просто точки будут более крупными, а изображение более грубым.

При печати разрешение должно быть намного выше. Полиграфическая печать полноцветного изображения требует разрешения около 300 dpi. То изображение, которое занимает весь экран монитора, при такой печати будет иметь размер примерно 7*5 см. При попытке напечатать его в увеличенном размере, будет заметна зернистость изображения, то есть пострадает его качество.

Таким образом, если сканируемое изображение предполагается лишь рассматривать на экране, без увеличения, нет смысла устанавливать разрешение изображения больше 75 dpi. Если же предполагается печать изображения в том же масштабе, то разрешение должно быть 300 dpi.

Естественый размер изображения (actual size) это такой размер, при котором каждая точка файла отбражается на экране или бумаге, причем только одной точкой соответствующего устройства. Если размер изображения больше естественного - появляется зернистость, пикселизация. Если размер меньше естественного - часть информации не воспроизводится, можно сказать является излишней, неоправдано увеличивая размер файла. Поскольку разрешающая способность экрана и принтера различна, то, чтобы естественный размер изображения на бумаге был таким же как на

экране, графический файл для печати должен содержать информацию о значительно большем числе точек. Файл фотоснимка размером 10x15 см должен при этом содержать данные примерно 1,5 млн. точек.

Объем файла изображения зависит и от **цветового разрешения**, определяющего, сколько оттенков цветов может быть представлено в изображении. Для кодирования черно-белого изображения достаточно выделить по 1 биту на каждую точку (объем файла фотоснимка около 180 кбайт). Выделение 1 байта для точки позволяет закодировать 256 различных цветов (но объем файла составит уже 1,4 Мбайт).

А если на кодирование каждой точки использованы три байта (True Color, 24 разрядный рисунок), то возможно одновременное отображение 16,5 млн. цветовых оттенков, и объем информации обычного фотоснимка составит свыше 4 Мбайт. Современные цифровые камеры, оснащенные 10-мегапиксельной матрицей, нуждаются в 30MB на каждую фотографию. Сжатие иображений может немного помочь, но проблема остается. Понятно, что хранить черно-белое или малоцветное изображение в режиме True Color нет смысла.

Таким образом, можно отметить два недостатка растровой графики. Первый – это большой размер файлов, а второй – ограниченность увеличения растровых изображений для рассмотрения деталей. Поскольку изображение состоит из точек, то увеличение изображения приводит только к тому, что эти точки становятся крупнее. Это делает рисунок более грубым.

Формулировка задачи.

Создать графический растровый редактор. В нём реализовано рисование линий и геометрических фигур (прямоугольник, эллипс) с помощью карандаша и кисти, можно выбирать цвет, также возможно использование ластика для стирания ошибочно нарисованных фрагментов. Пользователь может создавать новый рисунок или загружать уже готовое изображение для последующего редактирования. По окончании работы изображение сохраняется.

Написание проекта.

Проект реализовывался на Java с использованием библиотеки Swing.

Графический интерфейс в Java прошел весьма тернистый путь развития и становления. Долгое время его обвиняли в медленной работе, жадности к ресурсам системы и ограниченной функциональности.

Первой попыткой создать графический интерфейс для Java была библиотека **AWT** (Abstract Window Toolkit) — инструментарий для работы с различными оконными средами. Sun сделал прослойку на Java, которая вызывает методы из библиотек, написанных на C. Библиотечные методы AWT создают и используют графические компоненты операционной среды. С одной стороны, это хорошо, так как программа на Java похожа на остальные программы в рамках одной ОС. Но при запуске ее на другой платформе могут возникнуть различия в размерах компонентов и шрифтов, которые будут портить внешний вид программы.

Чтобы обеспечить мультиплатформенность **AWT** интерфейсы вызовов компонентов были унифицированы, вследствии чего их функциональность получилась немного урезанной. Да и набор компонентов получился довольно небольшой. Так например, в AWT нет таблиц, а в кнопках не поддерживается отображение иконок. Тем не менее пакет **java.awt** входит в Java с самого первого выпуска и его можно использовать для создания графических интерфейсов.

Таким образом, компоненты **AWT** не выполняют никакой "работы". Это просто «Java-оболочка» для элементов управления той операционной системы, на которой они работают. Все запросы к этим компонентам перенаправляются к операционной системе, которая и выполняет всю работу.

Использованные ресурсы **AWT** старается освобождать автоматически. Это немного усложняет архитектуру и влияет на производительность. Написать что-то серьезное с использованием AWT будет несколько затруднительно. Сейчас ее используют разве что для апплетов.

Вслед за **AWT** Sun разработала графическую библиотеку компонентов **Swing**, полностью написанную на Java. Для отрисовки используется 2D, что принесло с собой сразу несколько преимуществ. Набор стандартных компонентов значительно превосходит AWT по разнообразию и функциональности. Swing позволяет легко создавать новые компоненты, наследуясь от существующих, и поддерживает различные стили и скины.

Создатели новой библиотеки пользовательского интерфейса **Swing** не стали «изобретать велосипед» и в качестве основы для своей библиотеки выбрали AWT. Конечно, речь не шла об использовании конкретных тяжеловесных компонентов AWT (представленных классами Button, Label и им подобными). Нужную степень гибкости и управляемости обеспечивали только легковесные компоненты.

Для создания графического интерфейса приложения необходимо использовать специальные компоненты библиотеки Swing, называемые контейнерами высшего уровня (top level containers). Они представляют собой окна операционной системы, в которых размещаются компоненты пользовательского интерфейса. К контейнерам высшего уровня относятся окна JFrame и JWindow, диалоговое окно JDialog, а также апплет JApplet (который не является окном, но тоже предназначен для вывода интерфейса в браузере, запускающем этот апплет). Контейнеры высшего уровня Swing представляют собой тяжеловесные компоненты и являются исключением из общего правила. Все остальные компоненты Swing являются легковесными.

Каждое приложение, которое имеет графический интерфейс пользователя не может обходиться без кнопок. В Java Swing кнопка представлена классом JButton. У кнопки имеются различные методы для ее конфигурирования — установка надписи на JButton, установка иконки, выравнивание текста, установка размеров и так далее. Кроме всего прочего разработчику необходимо навесить на JButton слушателя, который будет выполняться как только пользователь нажмет на кнопок.

Все взаимодействия пользователя с приложением основано на событиях. Не является исключением и JButton. Как только пользователь нажимает кнопку, создается ActionEvent событие, которое передается слушателям кнопки. Для того, чтобы организовать слушателя Swing предоставляет интерфейс ActionListener, который необходимо реализовать. Интерфейс ActionListener требует только реализации одного метода — actionPerformed. После того, как обработчик создан, его необходимо добавить к кнопке. Делается это при помощи метода addActionListener.

MouseListener используется для прослушивания событий мыши компонента. Для того, чтобы прослушивать события мыши необходимо реализовать интерфейс MouseListener, который располагается в пакете java.awt.event. По аналогии с другими слушателями swing, MouseListener —

это интерфейс, методы которого необходимо реализовать. Рассмотрим методы, которые требуют реализации. Всего их пять. Условно методы можно разделить на две части. Первые два метода, говоря простым языком, отвечают за движение курсора, то есть будут вызываться при движении курсора мыши (навели и убрали курсор с компонента). Остальные три будут вызываться при нажатии кнопок.

Методы очень простые и подробного описания думаю им не надо. Начнем рассмотрение с mouseEntered. Данный метод будет вызываться системой у слушателя каждый раз, когда курсор мыши будет оказываться над компонентом. В противоположность этому методу — mouseExited. Он срабатывает, когда убираем курсор мыши с компонента. Пример — у нас есть компонент. Мы добавили к нему слушателя MouseListener. Начинаем водить мышкой. Как только «залезли» курсором на компонент — вызвался mouseEntered, уводим курсор с компонента — вызвался mouseExited.

Идем дальше. Каждый раз при нажатии одной из кнопок мыши будет срабатывать mousePressed. Навели на компонент, зажали кнопку — система вызвала mousePressed. Отпускаем кнопку — mouseReleased. Здесь всё просто. И самое интересное — это mouseClicked. По идее клик (click) — это когда пользователь нажал и отпустил одну из кнопок, но тут тоже есть свой момент. Если позиция курсора не меняется между зажатием и отпусканием кнопки, то mouseClicked срабатывает, если же зажали кнопку, сменили положение курсора — передвинули его куда-нибудь, но не убрали с компонента а затем отпустили, то mouseClicked не вызовется.

Интерфейс MouseListener реализован, что дальше? Дальше нужно добавить слушателя к компоненту при помощи метода addMouseListener и прослушивать события. В случае необходимости можно удалить слушателя при помощи removeMouseListener.

Работая с приложением, которое имеет графический интерфейс, пользователь прибегает к помощи не только мыши, но и клавиатуры. Java Swing даёт возможность разработчику приложения обработать различные события, которые поступают от клавиатуры в то время, когда пользователь нажимает клавиши. Давайте посмотрим, что необходимо сделать, чтобы иметь возможность слушать события клавиатуры. Для этого рассмотрим интерфейс KeyListener из пакета java.awt.event.

Как в случае и с обработкой других событий, для обработки событий клавиатуры необходимо реализовать специальный интерфейс, а затем добавить получившегося слушателя к интересуемому компоненту. Интерфейс, который нужно реализовать для обработки клавиш был упомянут выше – это KeyListener из пакета java.awt.event. KeyListener имеет три метода: keyTyped, keyPressed и keyReleased.

Когда вызывается каждый из методов системой? Метод keyTyped вызывается системой каждый раз, когда пользователь нажимает на клавиатуре клавиши символы Unicode. Метод keyPressed вызывается системой в случае нажатия любой клавиши на клавиатуре. Метод keyReleased вызывается при отпускании любой клавиши на клавиатуре. Чтобы добавить слушателя KeyListener к интересуемому компоненту для прослушивания событий клавиатуры, используется метод addKeyListener. В качестве параметра методу передается ссылка на слушателя. Для удаления слушателя используется метод removeKeyListener.

Как можно добавить слушателя KeyListener к компоненту? Создается текстовое поле JTextField. Затем при помощи метода addKeyListener добавляется анонимный слушатель, который реализует все методы интерфейса KeyListener.
Как правило не всегда нужно реализовывать все три метода интерфейса KeyListener. Однако если мы делаем implements KeyListener, то обязаны сделать реализацию каждого метода интерфейса, даже если они будут пустыми. На такой случай есть специальный абстрактный класс KeyAdapter,

который содержит все три метода но с пустыми методами keyTyped, keyPressed и keyReleased. Тогда достаточно будет пронаследоваться от KeyAdapter и переопределить в нем только требуемый метод.

Каждый раз, когда пользователь нажимает клавиши на клавиатуре и система вызывает методы keyTyped, keyPressed и keyReleased, в качестве параметра им передается объект KeyEvent, который содержит всю необходимую информацию о произошедшем событии. Отсюда можно узнать код клавиши, которая была нажата – метод getKeyCode. Были ли зажаты при этом такие клавиши, как Alt, Shift или Ctrl. Проверить это можно вызвав соответственно методы isAltDown, isShiftDown и isControlDown. Класс KeyEvent содержит большой набор констант. Каждая константа содержит код соответствующей клавиши. Поэтому нет необходимости коды всех клавиш. Достаточно использовать какую-то из констант. По названиям констант можно легко определить, какой клавише она соответствует. Например KeyEvent. VK_ENTER или KeyEvent.VK_F.

Стоит сказать, что события от клавиатуры будут генерироваться системой только тогда, когда компонент, который мы слушаем, находится в фокусе.

Список используемой литературы.

- Г.Шилдт « Java 8 Полное руководство»
- Cay S. Horstmann “Big Java Early Objects”
- https://docs.oracle.com/javase/tutorial/uiswing/
- http://www.java2s.com/Tutorial/Java/0240__Swing/Catalog0240__Swing.htm