|  |
| --- |
| **МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**  **БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**  **МЕХАНИКО МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ**  **Кафедра веб-технологий и компьютерного моделирования**  **СОЗДАНИЕ КОНСОЛЬНОЙ/НАСТОЛЬНОЙ УТИЛИТЫ ДЛЯ ОБРАБОТКИ ИЗОБРАЖЕНИЙ**  Курсовая работа  Евсей Ирины Сергеевны  студентки 2 курса, специальность «математика и информационные технологии (направление математическое и программное обеспечение мобильных устройств)»  Научный руководитель:  Старший преподаватель,  Д. Ю. Дедков  Минск, 2017 |

**Oглавление**

1. Введение………………………………………………………….………3
2. Концепция и основные особенности пакета Swing.………......……….5
3. Описание и использование основных компонент
   1. Фреймы……………………………………………………………6
   2. Диалоги……………………………………………………………6
   3. Панели и менеджеры расположения…………………………….7
   4. Панель с управляющими кнопками………….…………………..9
   5. Кнопки…………………………………………………………….10
   6. Раскрывающиеся и простые списки…………………………….11
   7. Меню……………………………………………………………....11
   8. Ползунковый регулятор……………………………………….....12
   9. Метки……………………………………………………………...13
   10. Всплывающая подсказка………………………………………....13
4. Обработка реакций элементов управления с помощью специальных классов ....................……………………………………………...……....14
5. Работа с графикой

5.1. Графика 2D…………………………………………………….…...15

5.2. Пространства координат…………………………………………..16

1. Краткое описание функциональности ImageMaker (version 1.0)……..18
2. Основные компоненты программы.………………………………….....19

7.1. ImageMakerMainFrame……………………………………………....20

7.2. ImageMakerToolBar………………………………………………….21

7.3. ImageMakerMenu…………………………………………………….21

7.4. JFileChooser…………………………………………………………..22

7.5. ImagePanel……………………………………………………………23

7.6. BlurActionHelper……………………………………………………..24

7.7. BrightnessActionHelper………..……………………………………..24

7.8. NegativeActionHelper.………………………………………………..25

7.9. GrayActionHelper...…………………………………………………..25

7.10. SharpeningActionHelper.……………………………………………26

7.11. RotateActionHelper...………………………………………………..26

7.12. ZoomInActionHelper/ZoomOutActionHelper……...…………….....27

7.13. ResetActionHelper..………………………………………………....27

7.14. ExitActionHelper...………………………………………………….27

1. Реализация программы…...……………………………………….……..27
2. Заключение……………......……….………………………………...…...48

Используемая литература.……………….……………………………….....49

1. **Введение**

В настоящее время одной из самых передовых технологий в области программирования является Java технология, начало которой было положено компанией Sun Microsystems в июне 1995 г. Кроме того, что Java – это мощный современный объектно-ориентированный язык, это еще, главным образом, новая концепция создания программ, позволяющая обеспечивать простой и быстрый перенос программ на различные платформы. Девиз Java "write once - run everywhere" (напиши один раз - выполни где угодно) является главным преимуществом этой технологии.

Начальным этапом является написание программы на языке высокого уровня (на языке Java). Далее, с помощью компилятора Java создается промежуточный код, или так называемый байт-код (bytecodes) для некой абстрактной виртуальной машины Java (Java Virtual Machine - JVM). Реальная же виртуальная машина устанавливается на той платформе, на которой предполагается выполнение написанной программы. Так, компания Sun совместно с рядом независимых разработчиков создала программную версию JVM для большинства известных платформ: Solaris, Linux, Mac OS, Windows (подробности см. http://java.sun.com). Когда на компьютер загружается файл, содержащий байт-коды (так называемый файл класса - class file), он интерпретируется виртуальной машиной Java на данном компьютере, которая считывает файл класса и выполняет команды, написанные на языке Java. Поскольку ядро виртуальной машины легко переносится с одного типа компьютера на другой, можно ожидать, что для каждого нового типа процессора и для каждой новой операционной системы вскоре появится своя реализация JVM. Поэтому Java-программа сможет работать на любой известной платформе, для которой существует реализация JVM.

Кроме отмеченного главного преимущества Java (переносимость), следует привести еще ряд достоинств этой технологии:

* Совершенство языка. Java сочетает в себе положительные стороны многих языков, особенно C++. Являясь принципиально объектно-ориентированным языком, Java "приучает" программиста мыслить современными программистскими парадигмами ООП.
* Безопасность. В частности, в Java устранены некоторые небезопасные языковые средства, такие как указатели в C. Хотя на первый взгляд кажется, что использование указателей дает некоторую гибкость и свободу для программиста, но на практике это часто приводит к мучительной отладке больших и сложных программ, особенно, когда программа что-то пишет по ошибочному адресу. Следует отметить также повышенную безопасность апплетов (специальных Java программ для web-браузера).
* Широкий спектр возможностей. В Java реализованы мощные средства для разработки расчетных программ, GUI-интерфейсов, сетевых и web приложений, программ для работы с базами данных и графикой. Некоторые программы, реализующие сложную графику для web, по мнению автора вообще могут эффективно быть написаны только на Java.
* Открытая технология. Каждому, кто хочет работать на Java, не нужно платить для того, чтобы приобрести легальный пакет разработчика. Компания Sun предоставляет возможность всем приобрести его совершенно свободно (коммерческими могут быть лишь какие-то узкоспециальные программы и средства).
* Высокое качество документации и учебных материалов. Это достоинство на самом деле является немаловажным. Во "всемирной паутине" можно найти немало различных библиотек, поддерживаемых группами энтузиастов, документация которых часто страдает неполнотой и ошибочностью. Java же является государственным стандартом (в США) и поддерживается официально разработчиком (компанией Sun), а также постоянно увеличивающимся международным сообществом Java программистов.
* Возможность интеграции в Java откомпилированных кодов с других языков. Эта возможность реализована с помощью т.н. интерфейса JNI (Java Native Interface).

Как известно, за все надо платить, поэтому технология Java имеет также и недостатки. Основным недостатком обычно считают невысокую (по сравнению с машинно-зависимым кодом) скорость работы виртуальной машины. Однако нужно отметить, что этот недостаток за последние годы становится все менее и менее значимым вследствие постоянного усовершенствования виртуальной машины, в которой часто используемые интерпретирующие функции заменяются машинно-зависимыми. Как показали исследования, число таких функций в типовых приложениях часто превышает 50 %. Поэтому такая замена заметно увеличивает скорость работы JVM. Стремительность прогресса вычислительной техники также уменьшает значимость этого недостатка.

А если учесть тот факт, что за последние годы появились программные средства, позволяющие конвертировать байт-код непосредственно в машинно-зависимый, то отмеченный недостаток просто тает на глазах.[1]

**2. Концепция и основные особенности пакета Swing**

JFC (JavaTM Foundation Classes) представляет собой своего рода библиотеку или пакет классов, позволяющих пользователю программировать развитый GUI-интерфейс на современном уровне. Впервые JFC пакет был представлен общественности на "JavaOne developer" конференции в 1997 году и включал в себя следующие возможности:

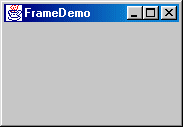
* Использование элементов: окон, кнопок т.д.
* Выбор произвольного стиля представления элементов (т.н. Looks & Feels) даже в процессе работы программы. В Swing существует пока три типа представления элементов: Windows, Metal, Motiff, но предполагается расширить их количество.
* Предоставляет возможность разработчикам легко интегрировать высокого качества 2D графику, тексты, рисунки в апплеты и приложения.
* Предоставляет возможность использовать технологию "drag and drop" между Java приложением и другими приложениями на конкретной платформе.

Надо отметить, что в первых версиях Java версий 1.0 и 1.1 (когда Swing еще не было) GUI интерфейс мог быть реализован с помощью базового пакета AWT (Abstact Windowing Toolkit), который не мог удовлетворить всех запросов программистов (в отличие от Swing). Хотя теперь Java продолжает поддерживать этот пакет, но Sun настоятельно рекомендует использовать только Swing. Для облегчения работы программистов названия Swing элементов начинаются с буквы J в отличие от названий в AWT (например, Button в AWT и JButton в Swing).

**3. Описание и использование основных компонент**

**3.1. Фреймы**

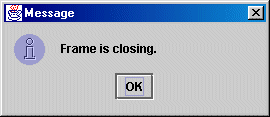
Фрейм, как экземпляр класса *JFrame,* является полноценным окном или контейнером, в котором происходит строительство всего GUI интерфейса. Фрейм имеет некоторые “прибамбасы”, такие как рамка, заголовок, кнопки закрытия, раскрытия и сворачивания окна. Все приложения с GUI, как правило, используют по крайней мере один фрейм, но и апплеты могут также использовать фреймы. Ниже показан пример простого фрейма.



**3.2. Диалоги**

В Swing имеется несколько классов поддерживающих диалоги. Диалоги - это окна, которые несколько ограничены по функциям по сравнению с фреймами и имеют некоторое специфичное назначение. Для создания самого простого информационного диалога следует использовать статический класс *JOptionPane.* Класс *ProgressMonitor* может сформировать диалог, показывающий индикатор полноты выполнения некого процесса. Два других класса *JColorChooser* и *JFileChooser* также представляют стандартные часто используемые диалоги, смысл которых ясен из названия и которые более подробно будут описаны ниже. Для создания произвольного диалога используется класс *JDialog.*

Код простейшего информационного диалога минимален. Ниже показан такой диалог и соответствующий ему исходный текст.



*Исходный текст:*

***JOptionPane.showMessageDialog(frame, "Frame is closing.");***

Первый параметр *frame* определяет родительский фрейм, в котором появляется данный диалог. Ниже показан ряд возможных вариантов использования метода *showMessageDialog():*

* ***showMessageDialog(frame, "Text.")*** - заголовок и значёк по умолчанию.
* ***showMessageDialog(frame,"Text.","Title",JOptionPane.WARNING\_MESSAGE)***- произвольный заголовок с предупреждающим значком.
* ***showMessageDialog(frame,"Text.","Title",JOptionPane.ERROR\_MESSAGE)***- произвольный заголовок со значком "ошибка".
* ***showMessageDialog(frame,"Text.","Title",JOptionPane.PLANE\_MESSAGE)***- произвольный заголовок без значка.
* ***showMessageDialog(frame,"Text.","Title",JOptionPane.INFORMATION\_MESSAGE)*** - произвольный заголовок со значком "информация".

Как и большинство классов, класс *JOptionPane* имеет еще множество других методов, дающих большую гибкость при программировании, среди которых можно еще отметить метод *showOptionDialog()*, позволяющий пользователю программировать в диалоге до 3-х кнопок (обычно Yes,No,Cancel) и определять реакцию на них.

**3.3. Панели и менеджеры расположения**

Класс *JPanel* пакета Swing представляет собой в чистом виде контейнер, предназначенный для размещения внутри себя других элементов, в том числе и других панелей. Все, что помещено в панель, можно рассматривать как группу никогда не расстающихся друзей. Если панель переместить или удалить, то та же участь постигнет и всех ее жителей. Панели - очень удобное средство для организации сложных GUI-интерфейсов, содержащих большое количество управляющих элементов, которые можно объединять в функциональные группы и потом размещать в удобном месте приложения.

По умолчанию сами панели визуально никак не проявляют себя на экране, кроме цвета непрозрачного по умолчанию фона (background). Однако к ним можно добавить видимую рамку (с заголовком или без). Класс *JPanel* имеет метод paint(), который позволяет рисовать в поле панели все, что пожелает пользователь. Панель можно сделать прозрачной, вызвав ее метод setOpaque(false). Для добавления компонентов в панель используется метод add().

С панелями (и другими контейнерами) тесно связано использование специальных классов, так называемых менеджеров расположения (layout managers), которые позволяют автоматически определять местоположение и размеры компонентов внутри панели. Существует несколько типов менеджеров расположения, каждый из которых растягивает (сжимает) и раскидывает добавляемые в панель элементы по определенному правилу. В Java используются следующие типы менеджеров расположения:

* ***BorderLayout.*** - предназначен для расположения 5-ти элементов, располагающихся, соответственно, в центре панели и по 4-м сторонам света (запад, восток, север, юг). Часто используется для приложений, выводящих основную информацию в центре, а боковые полосы использующих для различного рода кнопочек или строк состояния. Размеры каждой из этих 5-ти частей задаются автоматически пропорционально размеру панели.
* ***FlowLayout.*** - самый примитивный менеджер, который располагает, последовательно добавляемые элементы друг за другом в одну строчку с переносом на следующую, если уже не хватает места. Размеры элементов автоматически растягиваются так, чтобы заполнить всю панель.
* ***GridLayout.*** - решеточный менеджер, располагающий элементы в виде таблицы с заданным пользователем числом строк и столбцов. Размеры всех ячеек при этом одинаковые, а размер добавляемых элементов подгоняется под размер ячейки.
* ***GridBagLayout.*** - более "умный" решеточный менеджер, позволяющий, задавать индивидуальные свойства каждой ячейки (размер, поля), а также объединять несколько ячеек для расположения какого-нибудь большого элемента.
* ***CardLayout.*** - менеджер, который располагает элементы в виде колоды карт, при этом виден на экране только последний. Размер каждого элемента растягивается на всю панель. Для последовательного просмотра (открытия) каждого элемента используются его методы next() и previous().
* ***BoxLayout.*** - аналог FlowLayout, но располагающий элементы в направлении задаваемом пользователем (X-горизонталь или Y-вертикаль). В отличие от GridLayout, здесь каждый элемент может занимать ячейку различного размера. Этот менеджер используется в основном для специального контейнера типа Box, имеющегося только в Swing (а все предыдущие имеются также и в AWT).

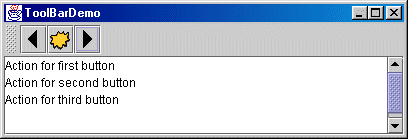
В Swing введено еще несколько специальных типов контейнеров, аналогичных JPanel, но предназначенных для узкоспециальных целей.

* ***ScrollPane***  - контейнерный класс, который применяет автоматически горизонтальную и/или вертикальную прокрутку (scrolling) единственного дочернего компонента. Пользователь может устанавливать специфическое поведение полос прокрутки. Для этого контейнера используется специальный менеджер расположения ScrollPaneLayout.
* ***JViewport.*** - контейнерный класс, представляющий собой некий "глазок видеокамеры", через который можно просматривать содержащуюся в контейнере информацию. Также имеет средства прокрутки и специфический менеджер расположения ViewportLayout.

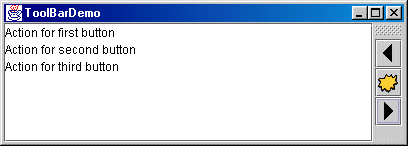
По умолчанию для всякой вновь создаваемой панели устанавливается менеджер *FlowLayout*, но пользователь может назначить любой другой с помощью метода панели setLayout(). Существуют ситуации, когда использование менеджеров расположения нежелательно, например, когда требуется задать конкретный размер элемента и его конкретное положение внутри панели. Для этого следует применить детерминированный подход и отменить менеджер путем вызова setLayout(null), затем использовать методы setSize() и setLocation() для каждого из добавляемых компонентов. При этом смешивать в рамках одной панели детерминированный подход с использованием менеджеров расположения нельзя, нужно использовать какой-либо один способ.[2]

**3.4. Панель с управляющими кнопками**

Класс *JToolBar* (панель с управляющими кнопками) является контейнером, который группирует несколько компонент (обычно кнопки со значками) в строку или столбец. Часто *JToolBar* обеспечивает легкое и удобное управление интерфейсом аналогично меню. Ниже показано приложение, содержащее панель с управляющими кнопками над текстовым полем.



По умолчанию пользователь может мышью перетаскивать панель с управляющими кнопками в другое место фрейма, например, из положения "север" в положение "восток", как показано на следующем рисунке:



Можно вообще перетащить панель за пределы главного фрейма. В этом случае генерируется новый фрейм (отдельное окно), в котором появляется наша панель:



Если закрыть этот вновь появившийся фрейм, то панель автоматически возвращается в свое начальное положение. Для правильной работы перетаскивания панели с управляющими элементами необходимо, чтобы ее родительский контейнер (в нашем случае главный фрейм) использовал менеджер расположения BorderLayout. Причем важно, что панель при этом не следует помещать в центр, а только по боковым сторонам ("запад", "восток", "север", "юг"). Перетаскивание панели в центр эквивалентно вытаскиванию его за пределы фрейма, т.е. панель появится тогда в новом окне.

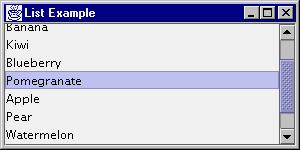
**3.5. Кнопки**

Кнопки являются самыми распространенными элементами управления в любом интерфейсе. В Swing имеется большой выбор разнообразных кнопок, организованных в четкую структуру. Все кнопки имеют один и тот же родительский класс AbstractButton. К кнопкам относятся следующие классы:

* ***JButton*** - Простая кнопка. Остается нажатой пока не отпущена левая кнопка мыши. В кнопку можно помещать надпись и/или значок, подключать к ней всплывающую подсказку, активировать и деактивировать ее.
* ***JCheckBox*** - Всем известный "флажок" или "галка".
* ***JRadioButton*** – Радио-кнопка. Обычно несколько таких кнопок объединяют в группу с помощью специального класса ButtonGroup, который в общем случае может применятся не только к радио-кнопкам, но и к другим объектам.
* ***JMenuItem*** - Пункт меню. Как ни странно, но это тоже кнопка.
* ***JCheckBoxMenuItem*** - Флажок во всплывающем меню.
* ***JRadioButtonMenuItem*** - Радиокнопка во всплывающем меню.
* ***JToggleButton*** - Кнопка аналогичная JButton, с той разницей, что остается нажатой после первого щелчка мышью. Повторный щелчок приводит ее в исходное состояние. По своей функциональности совершенно не отличается от JCheckBox и JRadioButton.

**3.6. Раскрывающиеся и простые списки**

Класс *JList* (простой список) позволяет пользователю объединять в группу ряд элементов, представляемых на экране в виде списка-столбца. Пользователь может выбирать любой элемент с помощью мыши, при этом селектор передвигается на выбранный пункт. Количество пунктов в списке не ограничивается и может быть достаточно велико, поэтому список, как правило, помещают в окно с прокруткой. С помощью метода setSelectionMode() пользователь может назначить для списка режим одиночного или множественного способа выбора пунктов одновременно. Кроме простого списка аналогичные функции (выбор из множества) могут обеспечивать и другие объекты пакета Swing, а именно: *JComboBox* (раскрывающийся список), *JMenu* (меню), *JTable* (таблицы) и группы кнопок *JCheckBox* или *JRadioButton.* Причем только простые списки, таблицы и группы кнопок *JCheckBox* могут обеспечивать множественный выбор элементов одновременно. Ниже показан пример простого списка в окне с прокруткой.[3]



Класс *JComboBox* (раскрывающийся список) имеется в двух существенно разных видах: *редактируемый* и *нередактируемый*. По умолчанию раскрывающийся список *нередактируемый*. *Нередактируемый* список внешне похож на простую кнопку, пока пользователь не щелкнет по нему мышью. После щелчка список раскрывается и показывает содержащийся в нем список элементов для выбора. Для решения задачи выбора из множества разработчики Sun рекомендуют использовать именно *нередактируемый* список, когда множество элементов велико, а пространство ограничено, или когда список может меняться в процессе выполнения программы. Ниже показан примеры *нередактируемого* раскрывающегося списка в раскрытом состоянии.

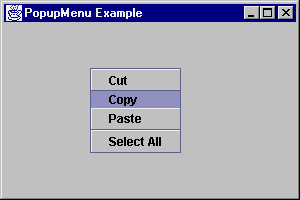
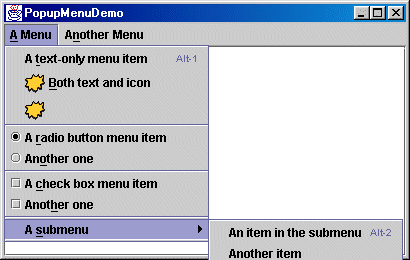
**3.7. Меню**

Как уже было отмечено выше, меню относится к классу кнопок, но в этом разделе меню будет описано более подробно. Меню - очень распространенный, удобный и занимающий очень мало места способ осуществлять процедуру выбора из некоторого множества в GUI-интерфейсах. Меню уникальны в том отношении, что они не размещаются вместе с другими компонентами в пользовательском интерфейсе. Вместо этого меню обычно или встроены в верхнюю часть главного фрейма в виде линейки кнопочек (*menu bar*) или появляются как всплывающие меню. Всплывающее меню - это меню, которое невидимо до тех пор, пока пользователь не сделает определенное действие мышью, например, пока не щелкнет правой кнопкой мыши или пока не щелкнет по определенному месту интерфейса. Меню при этом сразу появляется под курсором. В пункты меню, кроме текста, можно добавлять значок, подсказку для "горячей" клавиши и разделитель.

В Swing используются следующие классы для создания различных меню:

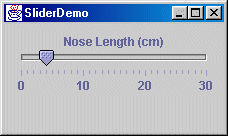
* ***JMenuBar*** - линейка меню, которая включается во фрейм.
* ***JMenu*** - кнопка меню в линейке *JMenuBar*
* ***JMenuItem*** - пункт меню-кнопка под кнопкой *JMenu* .
* ***JCheckBoxMenuItem*** - пункт меню-флажок под кнопкой *JMenu*
* ***JRadioButtonMenuItem*** - пункт меню-радиокнопка под кнопкой *JMenu*
* ***JPopupMenu*** - пункт всплывающего меню.

Класс *JMenu* имеет метод *add()* , с помощью которого в него можно добавлять пункты меню *JMenuItem* . Если в качестве пункта меню добавлен сам класс *JMenu* , то в этом случае формируется подменю, в которое уже можно добавлять свои пункты. Степень вложения меню не ограничивается. Ниже показаны два примера меню: раскрытой линейки меню (*menu bar*) и всплывающего меню (*popup menu*).



**3.8. Ползунковый регулятор**

Использование класса *JSlider* (ползунковый регулятор) дает возможность пользователю вводить численную величину, ограниченную заданными минимальным и максимальным значением. Используя *JSlider* вместо ввода в текстовое поле, пользователь гарантируется от ошибок. Кроме того, операция передвижения ползунка с помощью мыши



осуществляется быстрее, чем набивка руками на клавиатуре. Класс *JSlider* имеет множество методов, позволяющих пользователю формировать подходящий вид регулятора, включая размеры, шкалу и оцифровку. Ниже показан пример типичного ползункового регулятора.

**3.9. Метки**

Класс *JLabel* (метка) позволяет пользователю отображать небольшой текст и/или значок (картинку). Текст при этом не может быть выделен мышью. Если необходимо создать компонент, который отображает строчку и/или картинку с более сложными функциями, то следует создать новый класс, расширяющий *JLabel* , и наделить его данными и методами, которые необходимы. Иногда бывает удобнее просто использовать, какую-нибудь кнопку, которая может также включать в себя текст и картинки. Класс *JLabel* имеет ряд методов, позволяющих пользователю располагать и выравнивать текст и значок относительно друг друга и относительно границ самой метки. Ниже показано небольшое приложение с тремя метками: с текстом и значком, только с текстом и только со значком. Здесь окно разделено на три горизонтальные полосы равной высоты, в каждую из которых помещена метка, имеющая максимально возможную ширину.



Все содержимое метки по умолчанию имеет вертикальное выравнивание по центру. Но можно задать и другое выравнивание. Так, на приведенном рисунке для верхней метки, содержащей и текст и значок, задано горизонтальное выравнивание по центру. Для второй метки, содержащей только текст, задано выравнивание к левому краю (по умолчанию для меток только с текстом). Для третьей метки, содержащей только значок, задано горизонтальное выравнивание по центру (по умолчанию для меток только со значком).

**3.10. Всплывающая подсказка**

Все классы Swing, наследующие класс JComponent, могут использовать вплывающую подсказку, представляющую собой маленькое окно с некоторым поясняющим текстом, которое появляется всякий раз, когда пользователь удерживает курсор мыши некоторое малое время на данном компоненте. Создать всплывающую подсказку очень легко. Нужно просто использовать метод *setToolTipText(String s)* с аргументом, содержащим нужный поясняющий текст.

Ниже показано приложение, использующее всплывающую подсказку для кнопки *JButton.*



**4. Обработка реакций элементов управления с помощью специальных классов**

Всякий раз, когда пользователь нажимает на клавиши или делает какие-либо манипуляции с мышью, происходит событие. Любой объект может быть оповещен об этом событии. Все что надо сделать для такого оповещения, это наследовать соответствующий интерфейс к объекту и зарегистрироваться как "слушатель события" (event listener). Swing компоненты могут генерировать много видов событий. Вот некоторые из них:

|  |  |
| --- | --- |
| ***Действие, генерирующее событие*** | ***Интерфейс "слушателя"*** |
| Пользователь нажимает на кнопку, нажимает клавишу Enter после ввода текста или выбирает пункт меню | ***ActionListener*** |
| Пользователь закрывает фрейм (главное окно) | ***WindowListener*** |
| Пользователь нажимает кнопки мыши, когда курсор находится поверх объекта. | ***MouseListener*** |
| Пользователь движет мышь поверх компонента | ***MouseMotionListener*** |
| Компонент становится видимым | ***ComponentListener*** |
| Компонент получает фокус ввода от клавиатуры | ***FocusListener*** |
| Изменяется выбор в таблице или списке. | ***ListSelectionListener*** |

Каждое событие представляется объектом, который дает информацию об этом событии и идентифицирует источник события. Источники событий - это компоненты, однако и другие виды объектов могут быть источниками событий. Каждый источник события может иметь много зарегистрированных у него "слушателей". И напротив, один "слушатель" может регистрировать много разных событий.

Пусть, например, пользователю необходимо обработать определенное событие (например, нажатие на кнопку) в некотором пользовательском классе MyClass . Для этого необходимо реализовать следующие три шага:

* В объявлении класса MyClass наследовать соответствующий интерфейс "слушателя" или какой-то другой класс, который уже имеет этот интерфейс в качестве предка:

***public class MyClass implements ActionListener {***

* Зарегистрироваться, как "слушатель" события. Это достигается с помощью одной строчки кода, который обычно помещают в конструктор класса MyClass :

***addActionListener(this);***

* В классе MyClass определить (обязательно все) функции, унаследованные от интерфейса. Для нашего примера это только одна функция:

***public void actionPerformed(ActionEvent e)***

***{...//код обработки события...}***

**5. Работа с графикой**

**5.1. Графика 2D**

Начиная с версии JDK 1.2 компания Sun включила в состав пакета разработчика более мощный графический пакет java.awt.Graphics2D. Это достаточно полный и гибкий набор инструментов для графического вывода, поддерживающий дизайн различных линий, текста, картинок, имеющий более богатый и полнофункциональный интерфейс для пользователя. В состав API пакета входит:

* Обобщенная модель графического отображения для устройств вывода (мониторов, принтеров).
* Широкий круг геометрических примитивов, таких как кривые, прямоугольники, эллипсы, а также механизм для рисования практически любых геометрических форм.
* Механизм для выявления отклонений (дефектов) графических форм, текста и картинок.
* Механизм управления перекрывающимися объектами.
* Улучшенная цветовая поддержка, облегчающая управление цветом.
* Поддержка печати сложных документов.

Основной механизм рисования в java.awt.*Graphics2D* - графическая система управляет тем, когда и что программа должна рисовать. Когда некоторый компонент должен отобразить себя на экране, его метод paint() или update() автоматически вызывается с соответствующим ему контекстом *Graphics*. В новой версии JDK в API введен новый тип графического контекста - класс *Graphics2D*, который расширяет класс *Graphics*, обеспечивая улучшенную графику и свойства рисования.

Использование графики 2D в методах paint() осуществляется теперь следующим образом:

public void paint (Graphics g) {

Graphics2D g2 = (Graphics2D) g;

}

**5.2. Пространства координат**

Одним из осложняющих моментов при работе с графическими объектами является тот факт, что не все графические устройства имеют одну и ту же систему координат или разрешающую способность. Эта проблема особенно остро ощутима в языке Java, претендующем на девиз "напиши один раз - выполни где угодно". Для поддержки подобной репутации язык Java нуждается в таком способе представления графических объектов, благодаря которому будет обеспечено приемлемое качество изображения независимо от характеристик используемых устройств.

В определенных случаях может потребоваться написать программу, позволяющую создавать обычные рисунки и воспроизводить их как некоторую последовательность кадров. Если характеристики устройства, на котором воспроизводится данная последовательность, будут отличаться от характеристик устройства, на котором выполнялось рисование, вполне вероятно появление каких-либо ошибок.

В связи с этим обстоятельством в Java поддерживаются два координатных пространства:

* ***Пространство пользователя*** - пространство, в котором определены все графические примитивы.
* ***Пространство устройства*** - координатная система выходного устройства, такого как экран, окно или принтер.

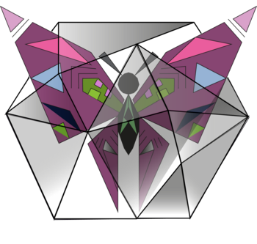
***Пространство пользователя*** - это независимая от устройства локальная система координат, используемая программой пользователя. Все геометрические объекты, рисуемые методами Java , определены именно в пространстве пользователя.

Если используется преобразование координат, *пространство пользователя - пространство устройства*, заданное по умолчанию, то начало координат принимается находящимся в верхнем левом углу области рисования. Координата **x** возрастает слева направо, а координата **y** - сверху вниз.

**Пространство устройства** - это зависимое от устройства координатная система, которая изменяется в зависимости от конкретного устройства графического вывода. Хотя координатные системы для окна или экрана может существенно отличаться от системы принтера, эти различия невидимы для Java программ. Все необходимые преобразования между пространством пользователя и пространством устройства выполняются автоматически во время процесса рисования.

**6. Краткое описание функциональности программы**

***ImageMaker (version 1.0)***

******

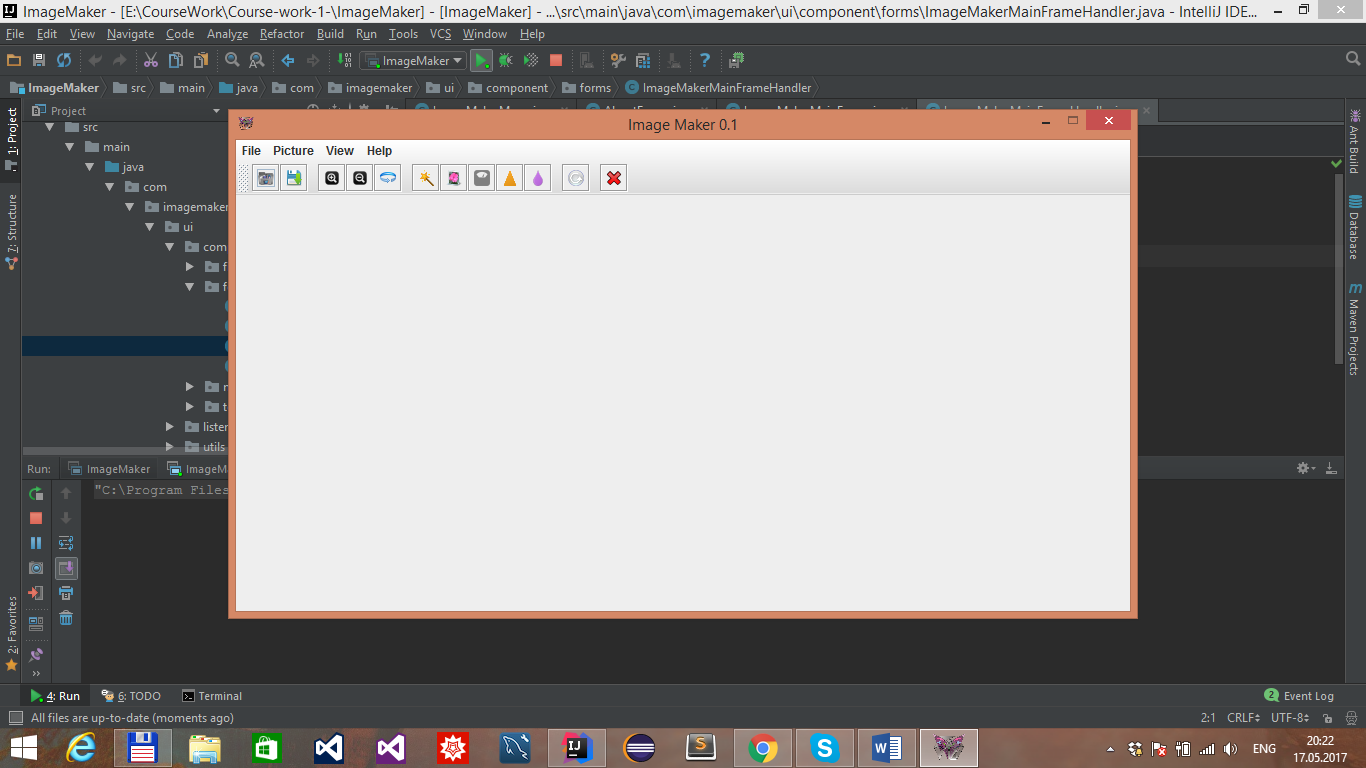
Эта утилита - инструмент для обработки изображений (.jpg и .png файлов), Результат обработки автоматически представляются в панели. Предоставляет возможность применения к изображению таких фильтров, как черно-белый и негатив,а так же осуществление поворота, увеличение/уменьшение яркости, смягчение и увеличение резкости, приближение и отдаление. Вы можете отменить изменения произошедшие с файлом.

***Возможности:***

* ***Поддерживает автоматическое изменение файла непосредственно в программе.***
* ***Поддерживает возможность выбора и открытия необходимого файла.***
* ***Присутствует фильтрация при выборе файла.***
* ***Поддерживает возможность сохранения обработанного изображения с нужным расширением.***
* ***Поддерживает возможно отмены произведенного действия.***
* ***Поддерживает возможно закрытия рабочего окна с помощью кнопки «Exit».***

**7.** **Основные компоненты программы**

Первое, что мы видим при запуске программы, это её главное окно, которое выглядит следующим образом:



Для расположения панелей в главном окне используется полярное расположение *BorderLayout.* Для этого окно разбивается на области, или полюса, а всё оставшееся место заполняется компонентом выполняющим основную функцию приложения. Надо сказать, что работает этот менеджер немного не так, как все остальные — чтобы добавить с его помощью компонент, в методе *add()* необходимо указать дополнительный параметр, который показывает, в какую область контейнера следует поместить компонент. Ниже я приведу пример значений этого параметра (который я использовала в главном окне своей программы) с частью кода:

* Значение ***BorderLayout.NORTH*** — компонент располагается вдоль верхней (северной) границы окна и растягивается на всю его ширину.

В программе *ImageMaker* на “север” помещен toolBar: ***getContentPane().add(toolBar, BorderLayout.NORTH);***

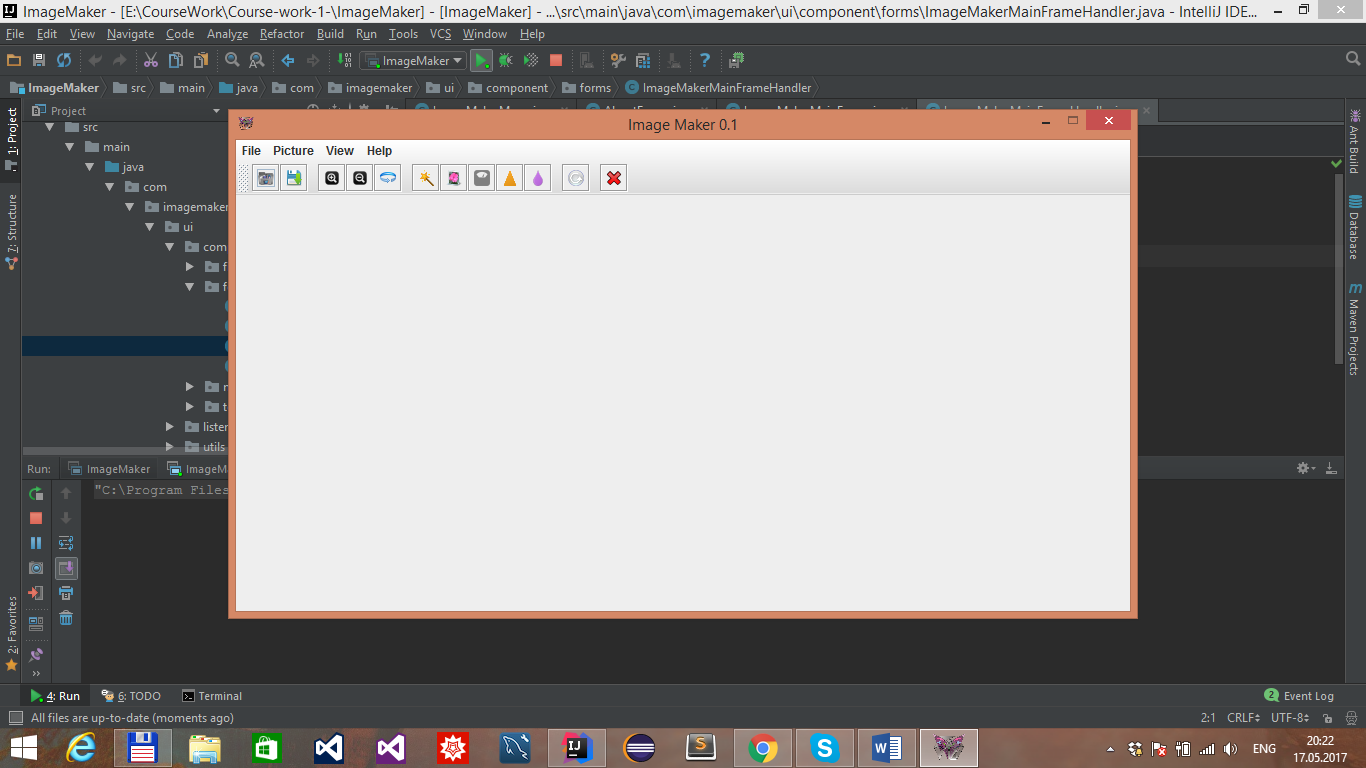
Используя метод *getContentPane()* панель *imagePanel* была установлена на главное окно программы.

***image\_frame.getContentPane().add(imagePanel);  
this.add(image\_frame)***

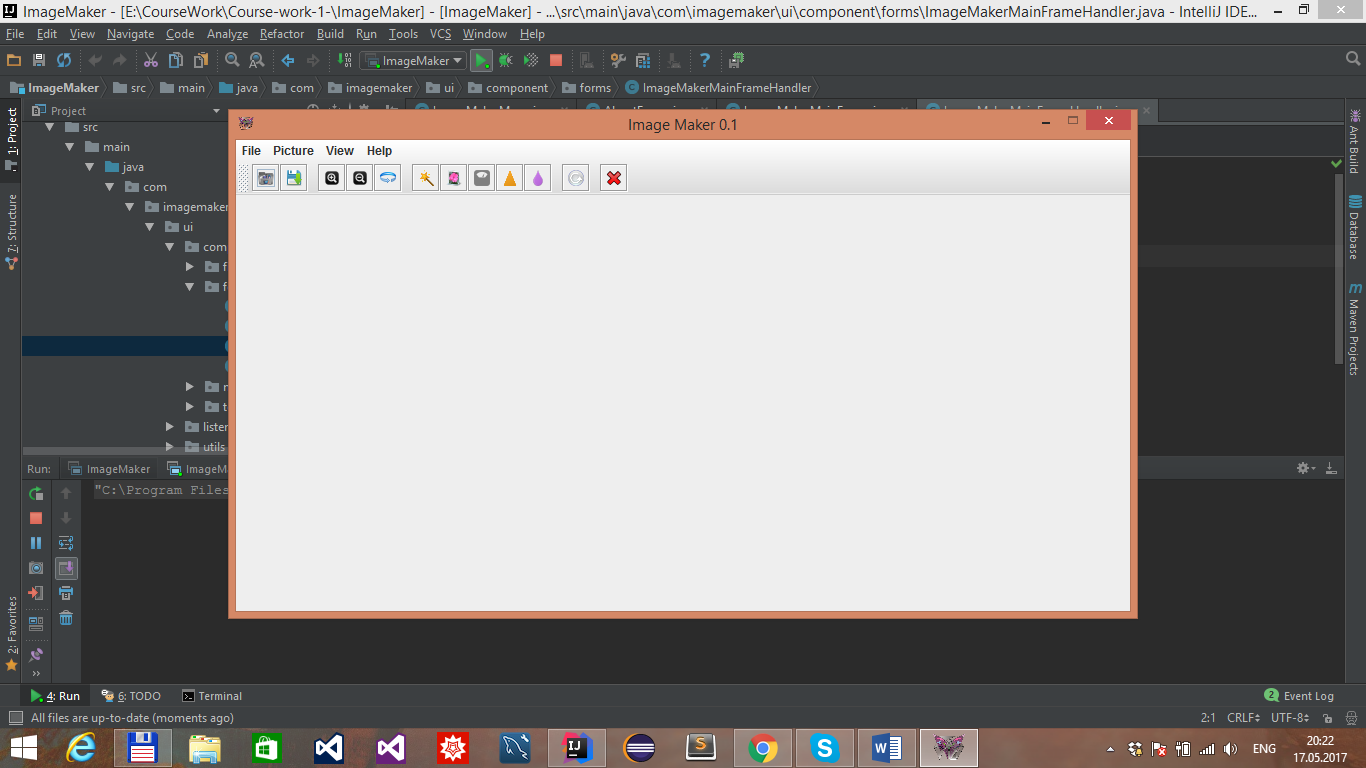
Далее рассмотрим каждую панель программы *ImageMaker* поподробнее.

**7.1. ImageMakerMainFrame**

Как вы уже успели заметить в моей программе основную часть окна ImageMakerMainFrame занимает большая главная панель ImagePanel, а так же панели ImageMakerToolBar и ImageMakerMenu в верхней части окна.

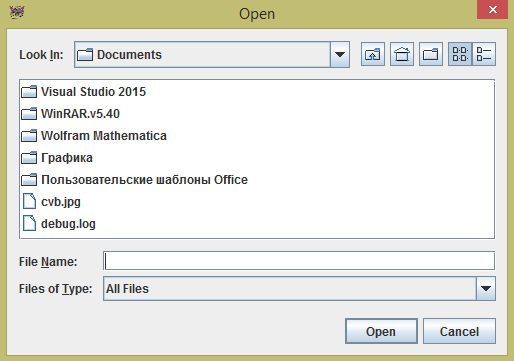


**7.2. ImageMakerToolBar**



На этой панели находятся JButton-ы со встроенными компонентами ImageIcon:

***openButton*** — это кнопка, предназначенная для вызова OpenActionListener(), который осуществляет открытие файла. Также, она содержит путь (где хранится файл, с которым работаете).



Это *JFileChooser* — диалог для выбора файлов. Кроме того диалог JFileChooser позволяет производить навигацию по файловой системе.

***saveButton*** — это кнопка, предназначенная для вызова SaveActionListener(), который осуществляет сохранение файла.

***zoomInButton / zoomOutButton*** — это кнопки, предназначены для вызова ZoonInActionListener() / ZoomOutActionListener(), которые осуществляют увеличение и уменьшение изображения соответственно.

***rotateButton*** – это кнопка предназначена для вызова RotateActionListener(), который осуществляет поворот изображения.

***brightnessButton*** – это кнопка предназначена для вызова BrightnessActionListener(), который осуществляет увеличение и уменьшение яркости изображения.

***negativeButton*** – это кнопка предназначена для вызова NegativeActionListener(), который осуществляет перевод изображения в негатив.

***grayscaleButton*** – это кнопка, предназначенная для вызова GrayscaleActionListener(), который осуществляет перевод изображения в черно-белый цвет.

***sharpeningButton*** – это кнопка предназначена для вызова SharpeningActionListener(), который осуществляет увеличение резкости изображения.

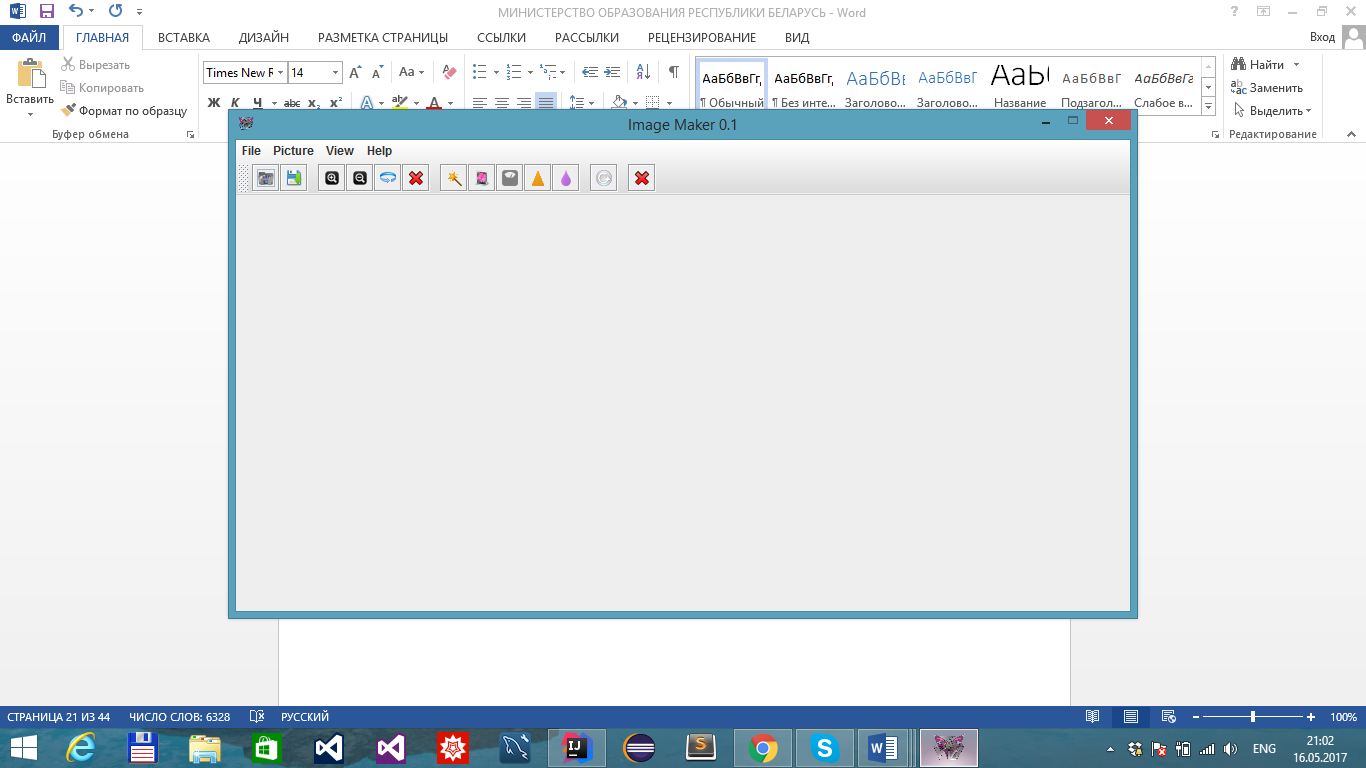
***blurButton*** – это кнопка предназначена для вызова BlurActionListener(), который осуществляет размытие изображения.

***resetButton*** – это кнопка предназначена для вызова ResetActionListener(), который осуществляет возвращение к исходному изображению.

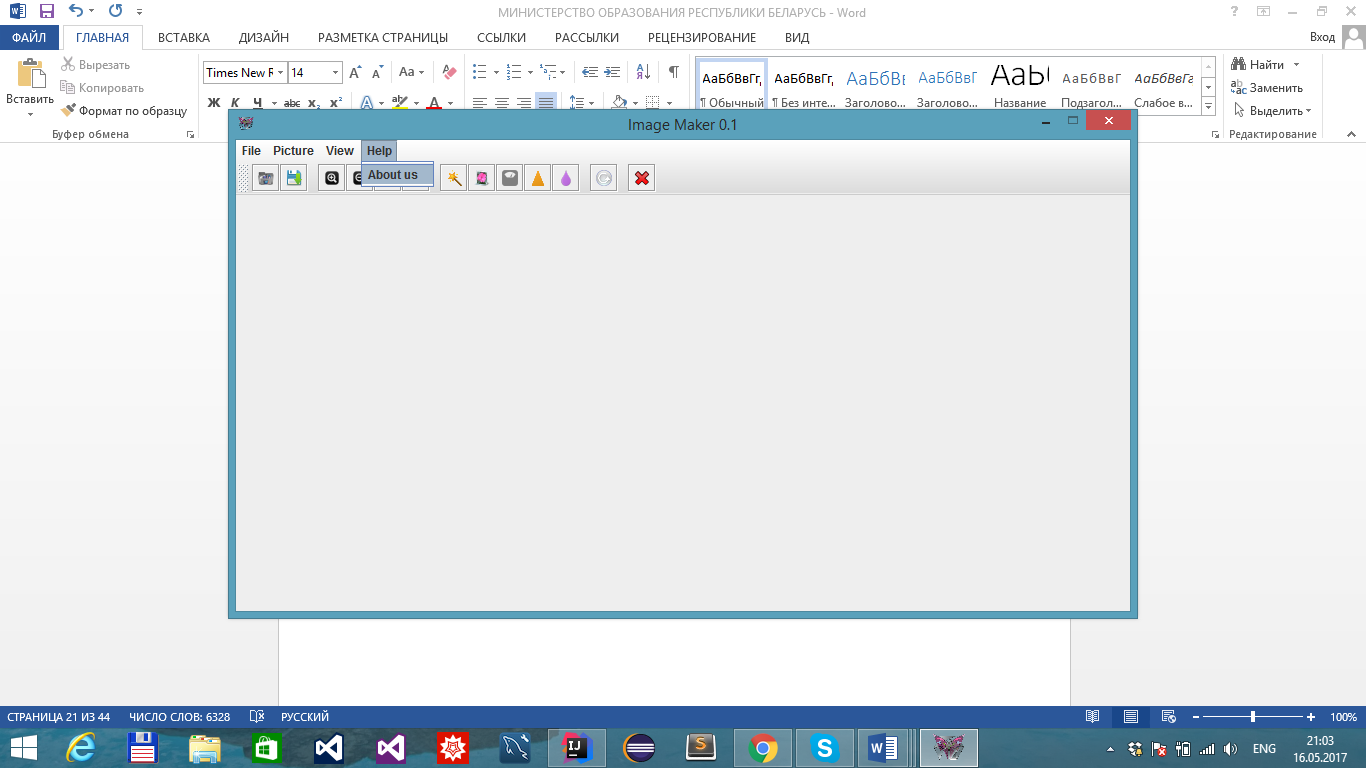
***exitButton*** – это кнопка предназначена для вызова ExitActionListener(), который осуществляет закрытие рабочего окна.

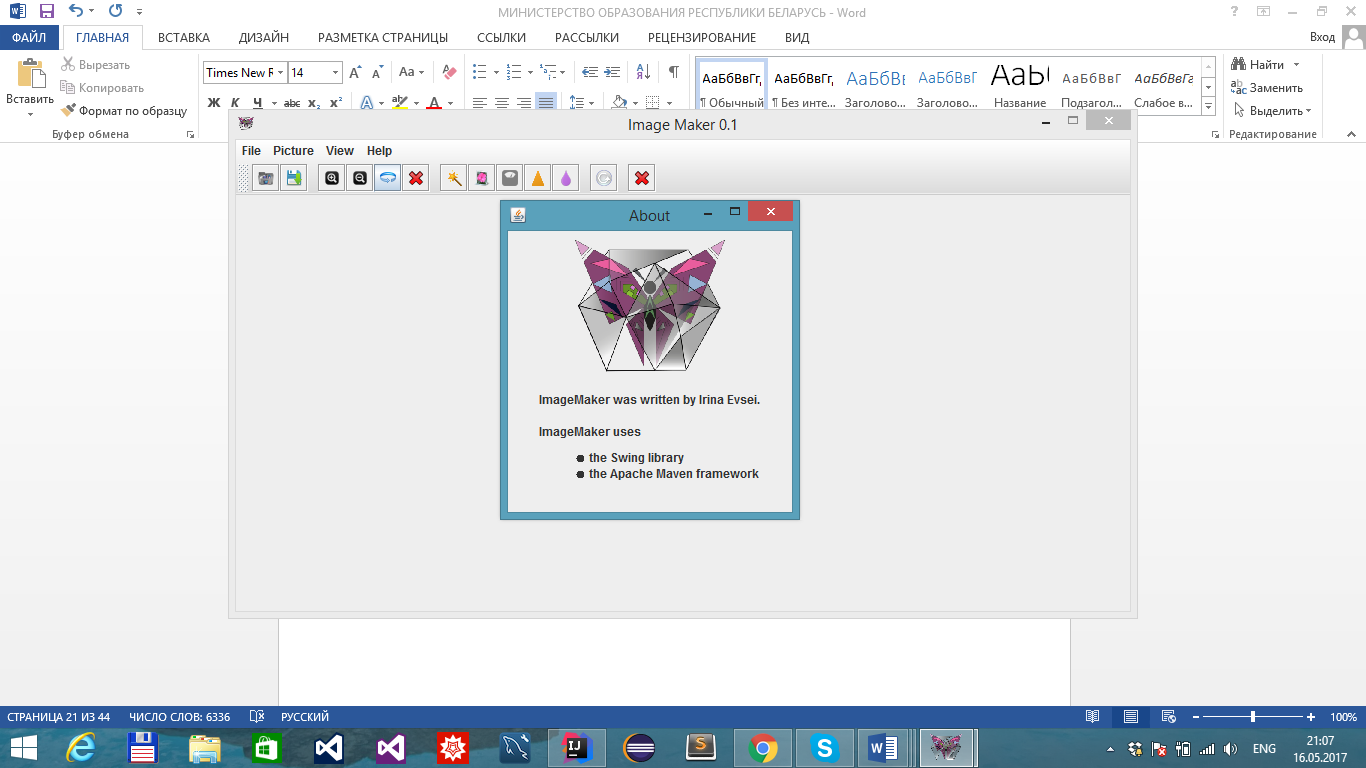
**7.3. ImageMakerMenu**

В данном меню продублированы все те кнопки и действия, которые присутствуют в ImageMakerToolBar.



Отличие составляет только пункт *Help,* которыйсодержит в себе информацию о программе:





**7.4. JFileChooser**

Во фрагменте, представленном ниже предполагается, что пользователь в диалоге JFileChooser выбрал файл и открыл его. При помощи метода getSelectedFile мы получаем ссылку на объект File и затем продолжаем с ним работать:

*FileChooser fileChooser = (FileChooser) e.getSource();*

*if (JFileChooser.APPROVE\_SELECTION.equals(e.getActionCommand())){*

*File file = fileChooser.getSelectedFile();*

*ImageMakerMainFrameHandler.getInstance().openImage(file);*

*}*

Для вывода диалогового окна открытия файлов служит следующий метод:

*public void actionPerformed(ActionEvent e) {*

*FileChooser fileChooser = new FileChooser(true);*

*fileChooser.showOpenDialog(ImageMakerMainFrameHandler.getInstance());*

*}*

По умолчанию в списке файлов, который компонент *JFileChooser* предоставляет пользователю, содержатся все файлы из текущего каталога (того, что пользователь просматривает в данный момент). Зачастую это бывает очень неудобно, поэтому я использовалa *фильтр файлов.*

*public static final FileNameExtensionFilter FILTER\_JPG = new FileNameExtensionFilter("\*.JPG", "jpg");*

*public static final FileNameExtensionFilter FILTER\_PNG = new FileNameExtensionFilter("\*.PNG", "png");*

Здесь анализируется имя файла и делается вывод о его пригодности для включения в список файлов, который увидит пользователь. В данный метод попадают абсолютно *все* файлы. Для обычных файлов я провожу анализ имени: мой фильтр должен отбирать файлы изображений, а их имена (точнее расширения) заканчиваются символами ***«jpg» / «png».***

Фильтр файлов к компоненту *JFileChooser* присоединён при помощи метода *addChoosableFileFilter()****.***

*private void buildComponent(boolean openDialog) {  
 addChoosableFileFilter(FILTER\_JPG);  
 addChoosableFileFilter(FILTER\_PNG);  
 /\*………..\*/  
}*

В моей программе создано два фильтра файлов: *FILTER\_JPG, FILTER\_PNG.*

**7.5. ImagePanel**

Класс ImagePanel предназначен для создания панели, в которую будет загружаться картинка. Переопределила метод public void paint(Graphics g). Реализация этого метода будет производить прорисовку картинки на панели. Более того, мы будем управлять размерами изображения, и подстраивать (вписывать) его под размеры ImagePanel.

*static BufferedImage image;*

*static BufferedImage copyImage;*

*public void paint(Graphics g) {*

*g.drawImage(image, 0, 0, null);*

*repaint();*

*}*

*public ImagePanel(File img) {*

*try {*

*image = ImageIO.read(img);*

*copyImage = image;*

*} catch (Exception ex) {*

*MessageDialogUtils.showErrorMessage(ImageMakerMainFrameHandler.getInstance().getRootPane(), "ERROR! Cannot open the file.");*

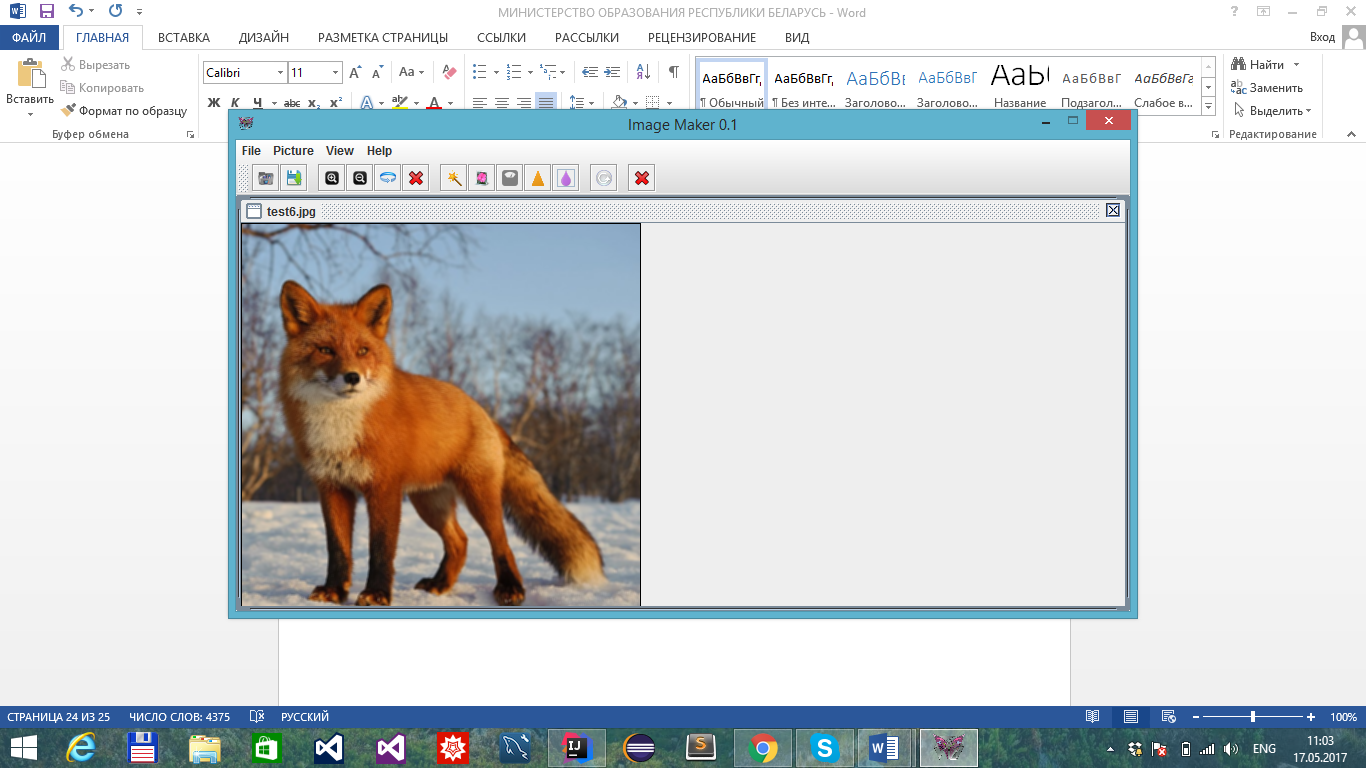
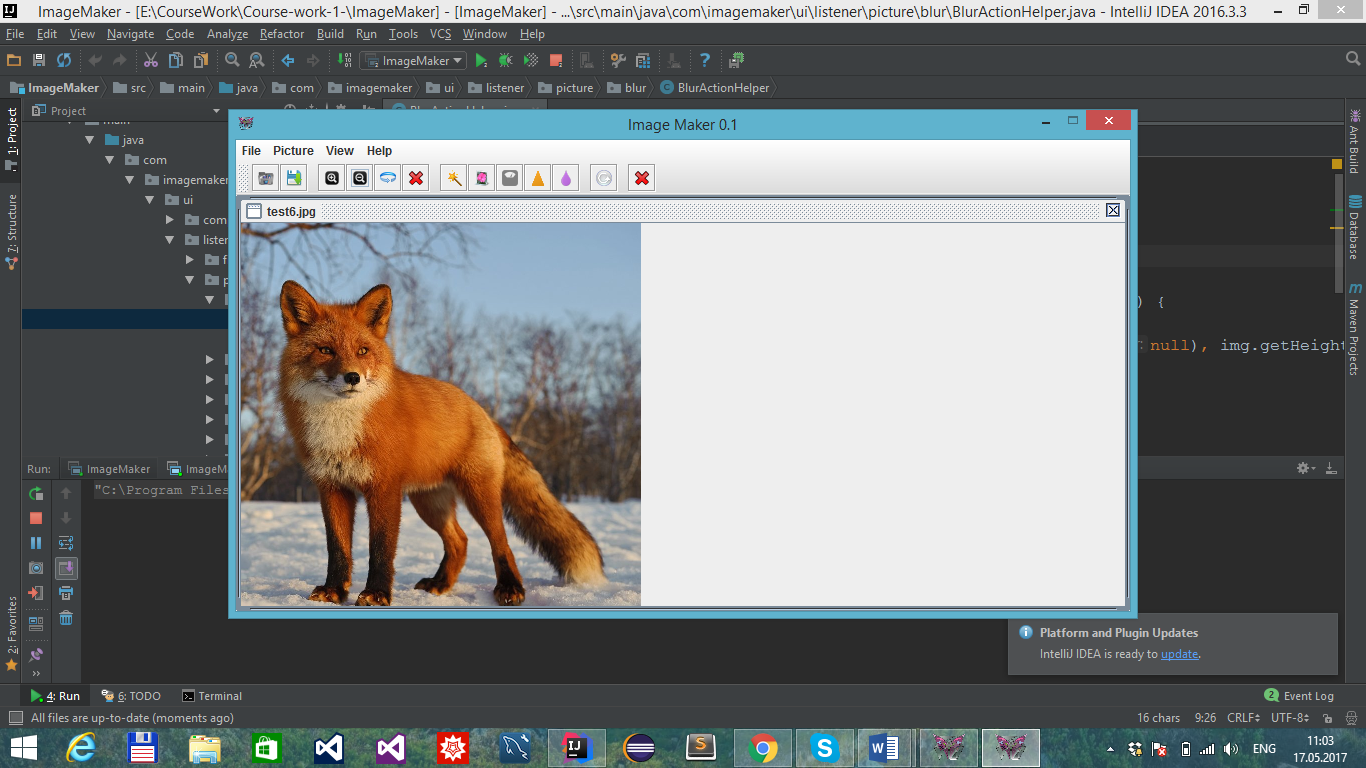
*}*

}

**7.6. BlurActionHelper**

Если говорить простым языком, то этот класс предназначен для обычного размытие картинки. Данный эффект подойдет, если вам нужно сделать края более мягкими. За счет размытия создается ощущение фона, который не в фокусе.

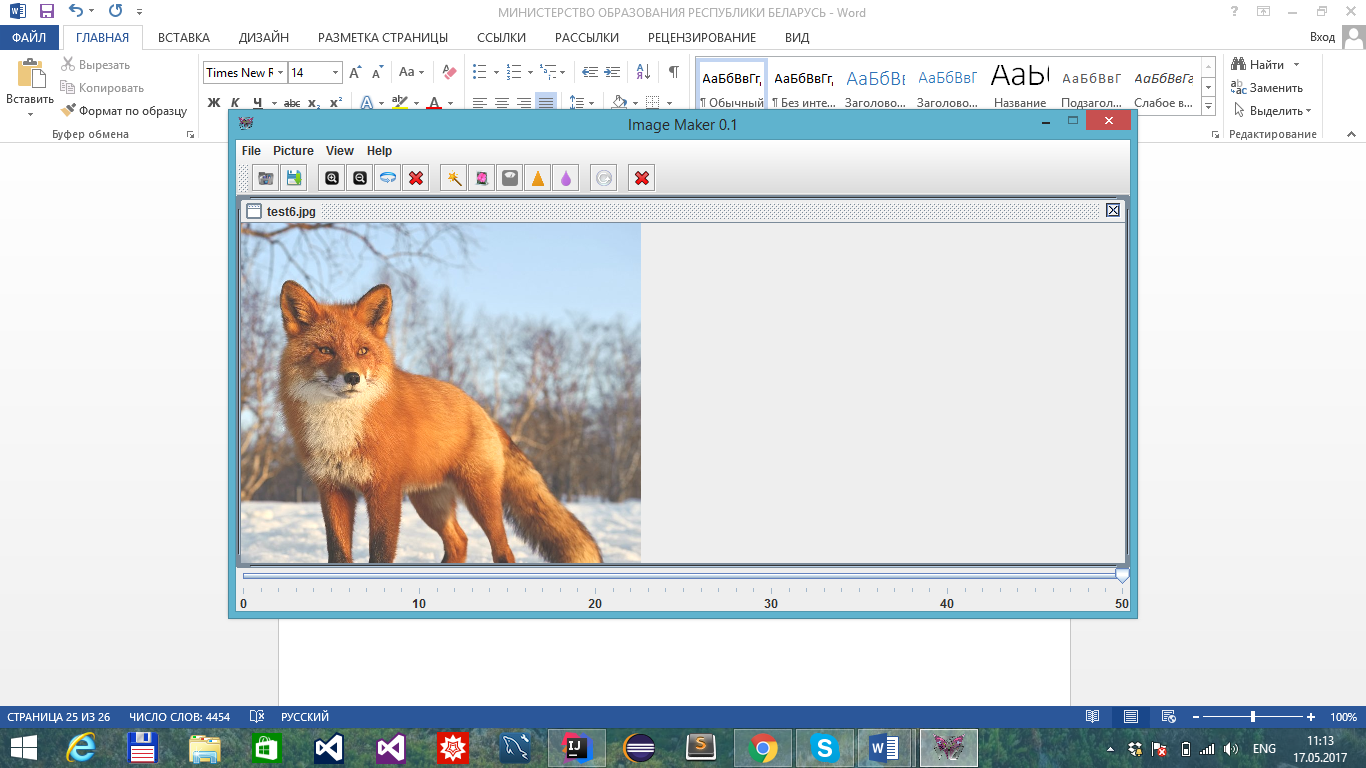
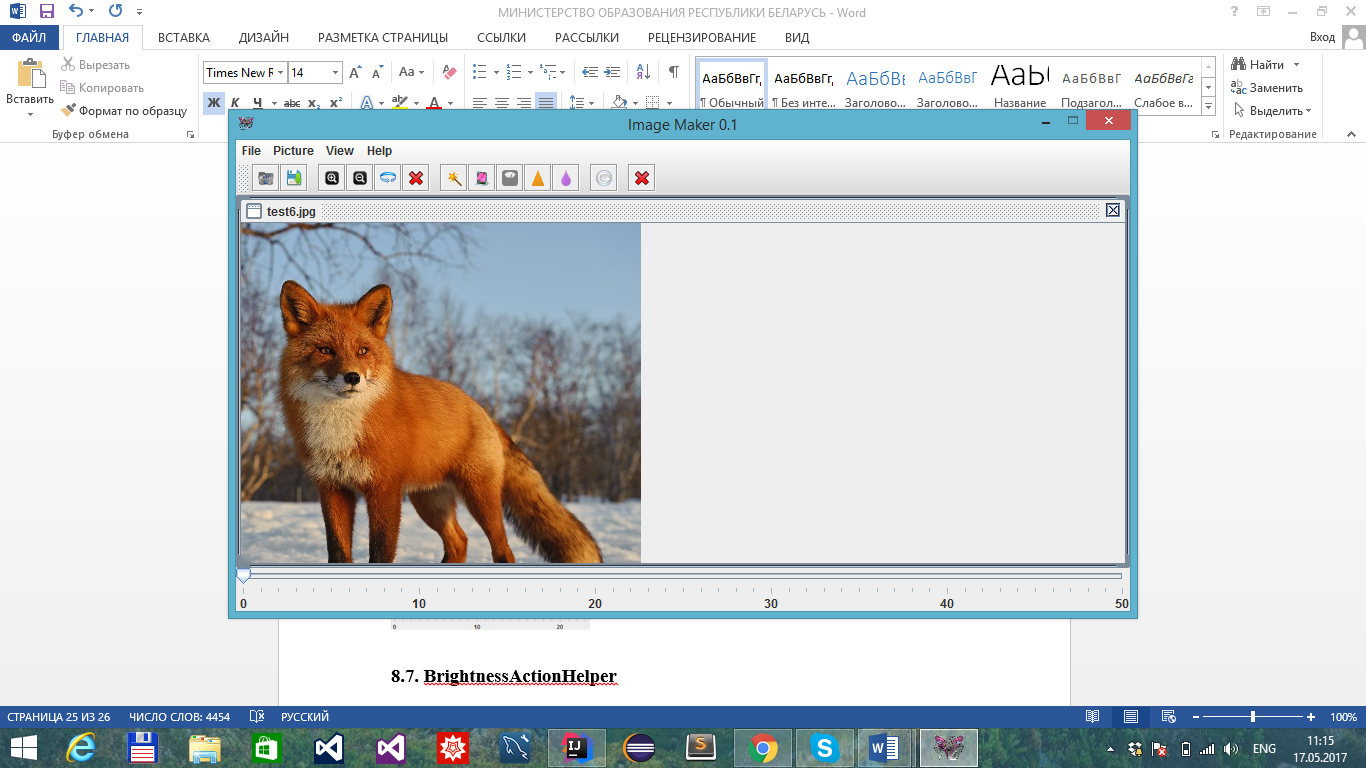
Слева, для наглядности, я поставилa обычную картинку, а вот справа то же самое изображение, только изменённое при помощи фильтра Blur.



**7.7. BrightnessActionHelper**

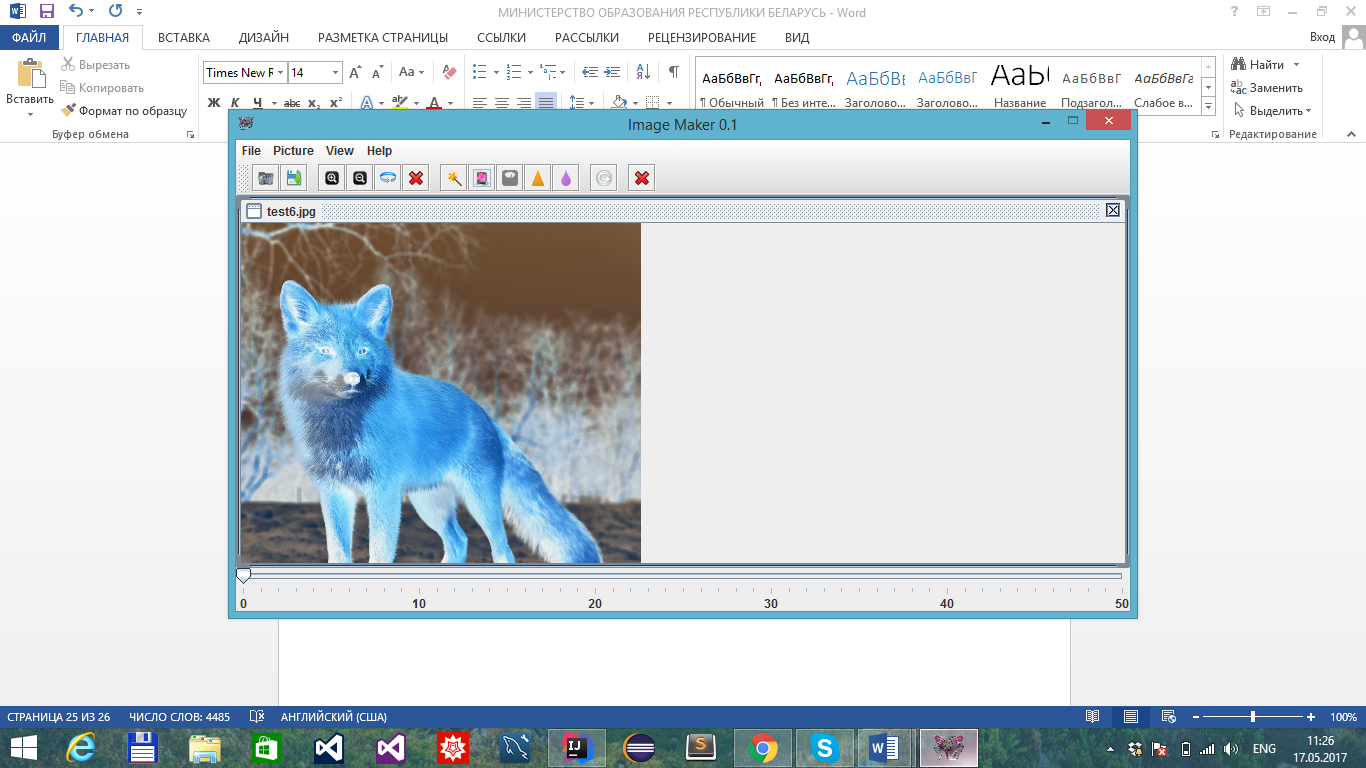
Класс представляет собой фильтр, который напоминает изменение яркости экрана телевизора.

В данном случае регулируется цвет между черным и оригинальным цветом по мере добавления параметров. Регулировать вы можете от 0% и более. При 0% изображение будет темнее, при 25% - оригинальным, а при 50% - станет светлее в два раза. Это очень хороший эффект, особенно для темных изображений.



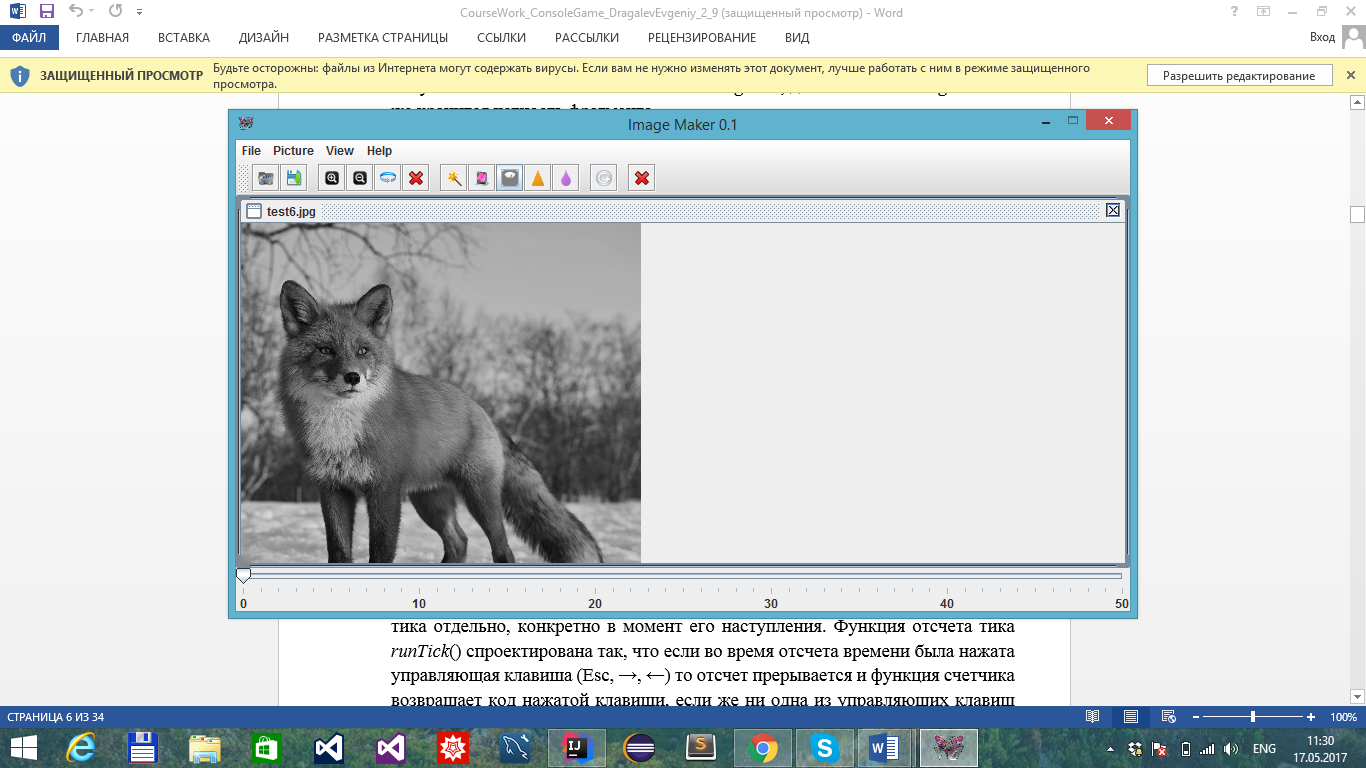
**7.8.** **NegativeActionHelper**

Данный класс представляет собой фильтр, который меняет тона и цвета изображения на прямо противоположные цвета и тона оригинала. Например, светлые области изображения в негативе темные, а желтый цвет — синий.



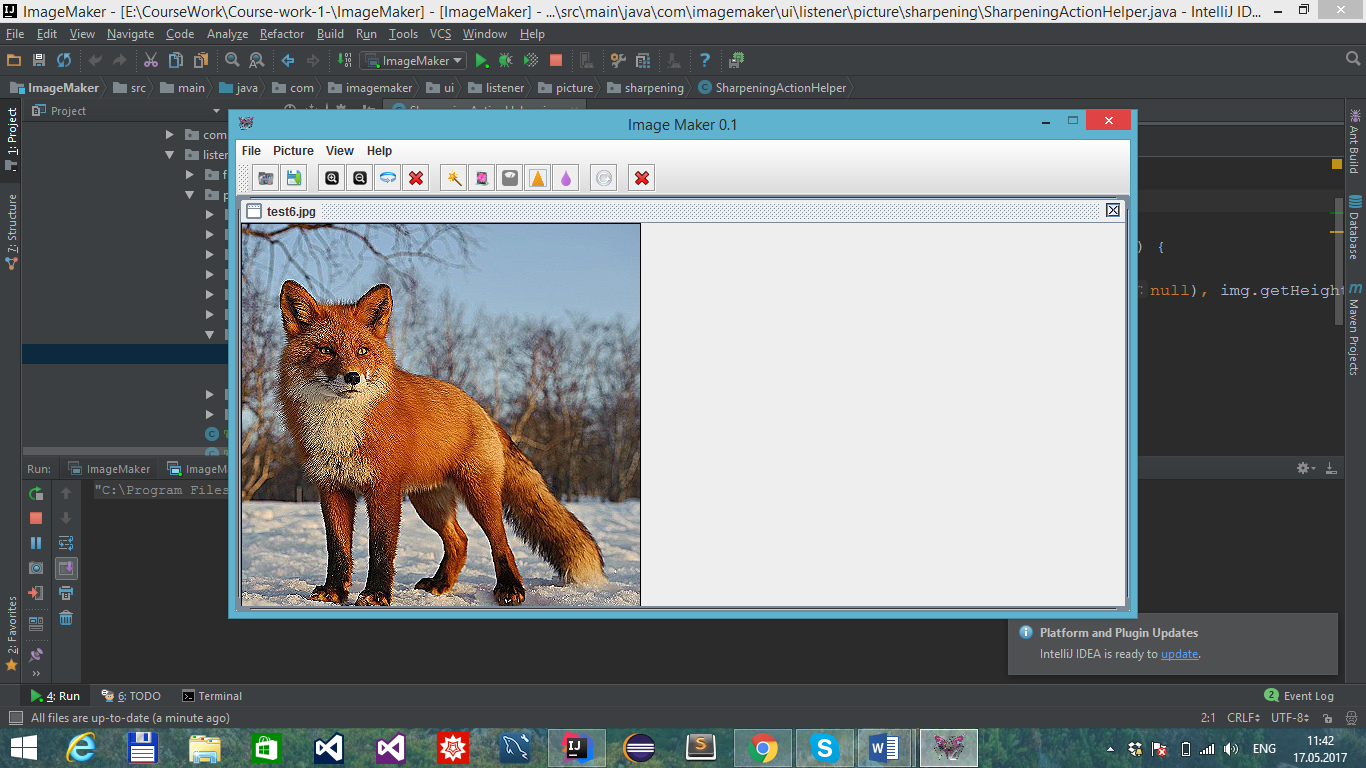
**7.9. GrayActionHelper**

Класс представляет собой фильтр, который позволяет нам превращать цвета в оттенки серого.



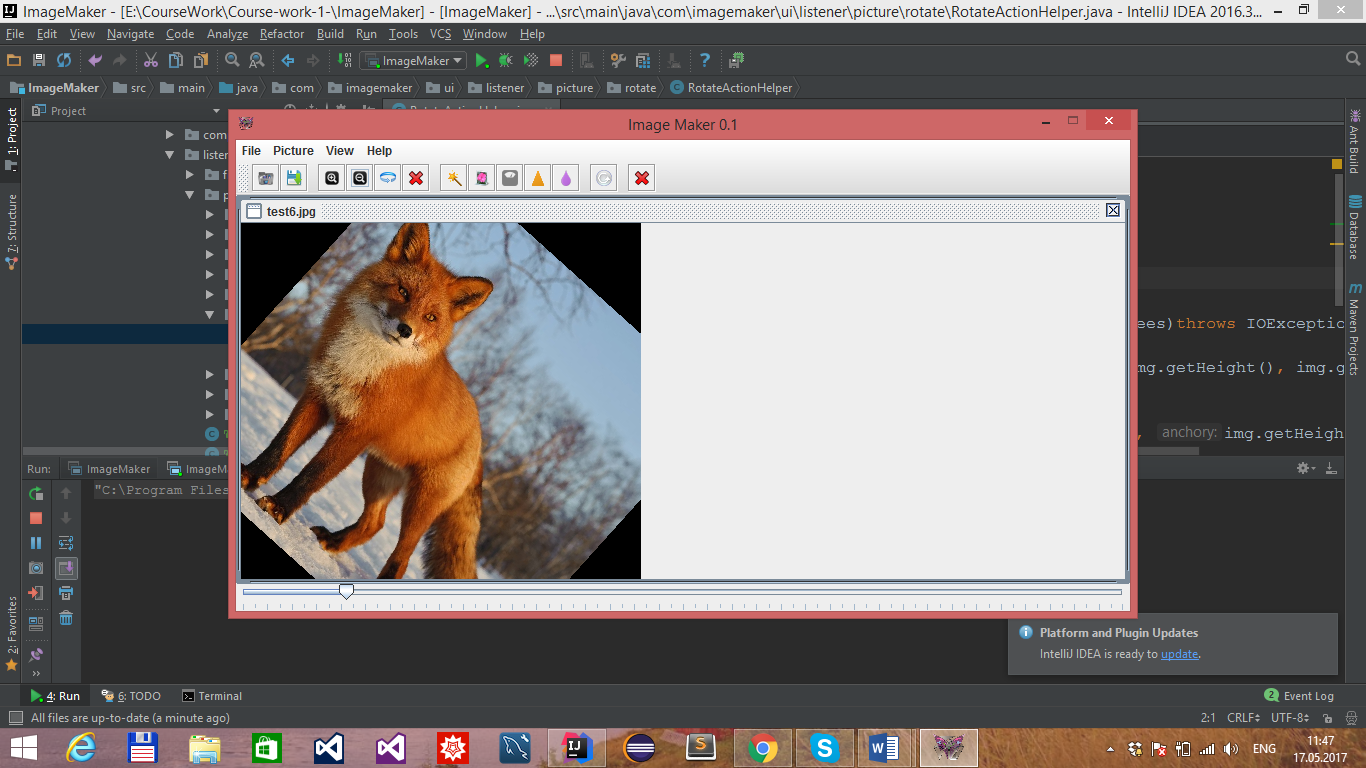
**7.10. SharpeningActionHelper**

Класс представляет собой фильтр, который увеличивает резкость, однако это автоматический фильтр, в котором не предусмотрены элементы управления. Увеличивать резкость можно для всего изображения, но не для отдельной его части, заданной выделением.



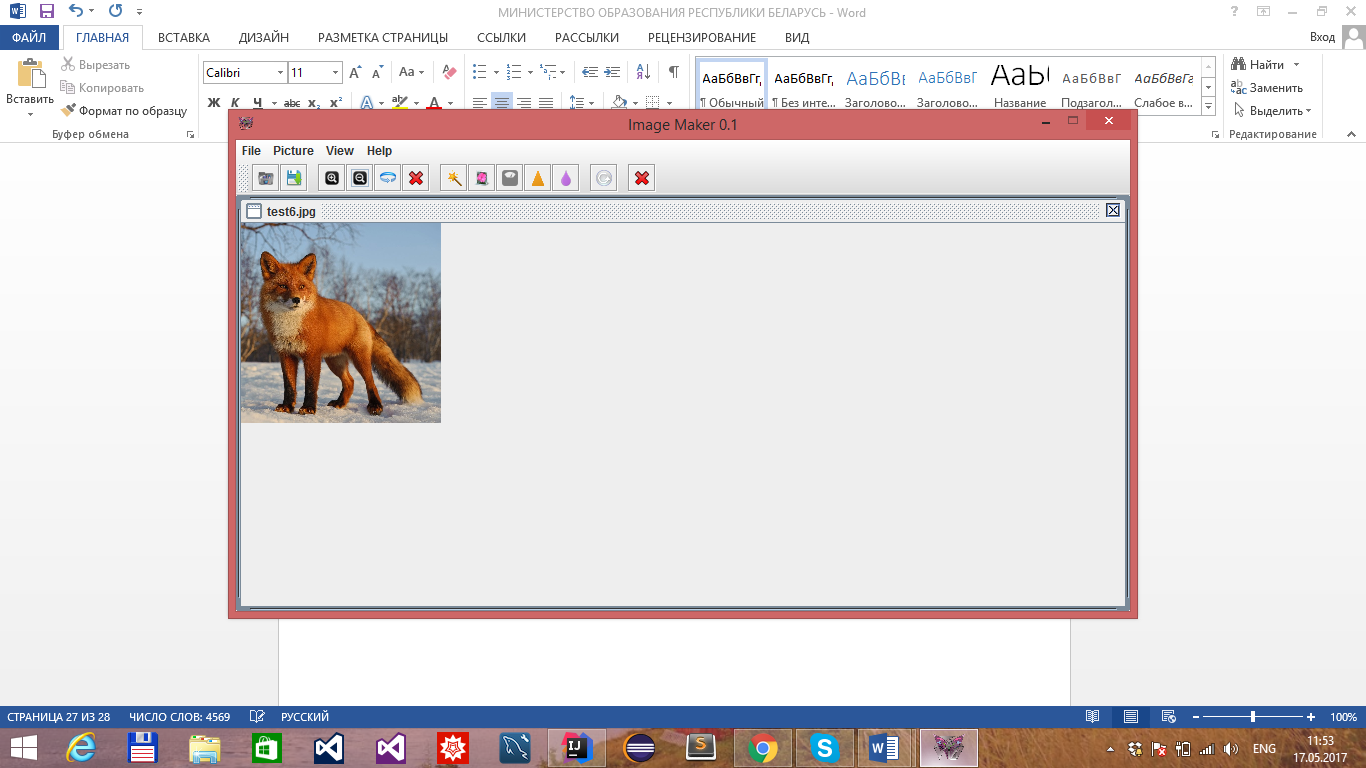
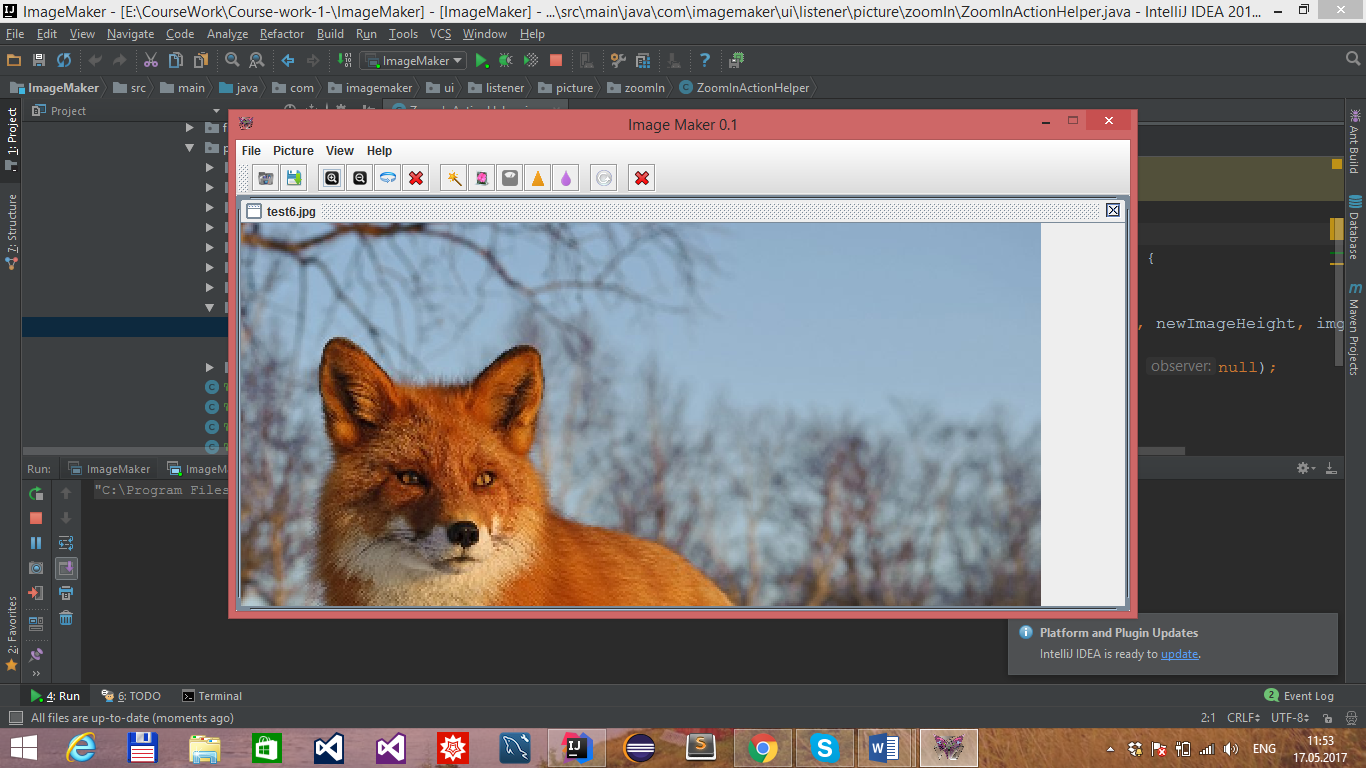
**7.11. RotateActionHelper**

Класс для осуществления поворота изображения. Включает панель с ползунком, который позволяет регулировать угол поворота.



**7.12. ZoomInActionHelper / ZoomOutActionHelper**

Данные класс позвляют приближать или отдалять изображение. При этом размер оригинала увеличивается либо уменьшается в 2 раза.



**7.13. ResetActionListener**

Класс ResetActionListener предоставляет пользователю применять отмену произведенных с изображением действий.

**7.14. ExitActionListener**

Этот класс позволяет пользователю завершить работу с программой и закрыть рабочее окно.

**8.Реализация**

*ImageMaker.java*

package com.imagemaker;  
  
import com.imagemaker.ui.component.forms.ImageMakerMainFrame;  
import com.imagemaker.ui.component.forms.ImageMakerMainFrameHandler;  
  
public class ImageMaker {  
  
 public static void main(String[] args) {  
 ImageMakerMainFrame frame = ImageMakerMainFrameHandler.*getInstance*();  
 frame.setLocationRelativeTo(null);  
 frame.setVisible(true);  
 }  
}

*FileChooser.java*

package com.imagemaker.ui.component.file;  
  
import com.imagemaker.ui.listener.file.OpenFileChooserListener;  
import com.imagemaker.ui.listener.file.SaveFileChooserListener;  
  
import javax.swing.JFileChooser;  
import javax.swing.filechooser.FileNameExtensionFilter;  
  
public class FileChooser extends JFileChooser {  
  
 public static final FileNameExtensionFilter *FILTER\_JPG* = new FileNameExtensionFilter("\*.JPG", "jpg");  
 public static final FileNameExtensionFilter *FILTER\_PNG* = new FileNameExtensionFilter("\*.PNG", "png");  
  
 public FileChooser(boolean openDialog) {  
 super();  
 buildComponent(openDialog);  
 }  
  
 private void buildComponent(boolean openDialog) {  
 addChoosableFileFilter(*FILTER\_JPG*);  
 addChoosableFileFilter(*FILTER\_PNG*);  
 if (openDialog){  
 addActionListener(new OpenFileChooserListener());  
 }else {  
 addActionListener(new SaveFileChooserListener());  
 }  
 }  
}

AboutFrame.java

package com.imagemaker.ui.component.forms;  
  
import java.awt.\*;  
import java.awt.image.BufferedImage;  
import javax.swing.\*;  
  
public class AboutFrame extends JFrame {  
 private static final String *ABOUT\_TITLE* = "About";  
 private BufferedImage pic;  
 ImageIcon iconOpen = new ImageIcon(ClassLoader.*getSystemResource*("img/logo.jpg"));  
  
 public AboutFrame() throws HeadlessException {  
 super(*ABOUT\_TITLE*);  
 buildComponent();  
 }  
  
  
 private void buildComponent() {  
 JPanel panel = new JPanel();  
 JLabel label = new JLabel("<html>ImageMaker was written by <b>Irina Evsei</b>." +  
 "<br><br>ImageMaker uses <ul>" +  
 "<li>the Swing library" +  
 "<li>the Apache Maven framework");  
  
  
 ImageIcon logo = new ImageIcon(iconOpen.getImage().getScaledInstance(150, 150, 1));  
  
 JLabel jLabel = new JLabel("", logo, JLabel.*LEFT*);  
  
 panel.add(jLabel);  
 panel.add(label);  
  
 add(panel);  
 setSize(300, 320);  
 setLocation(500, 200);  
 }  
}

*ImageMakerMainFrame.java*

package com.imagemaker.ui.component.forms;  
  
import com.imagemaker.ui.component.menu.ImageMakerMenu;  
import com.imagemaker.ui.component.toolbar.ImageMakerToolBar;  
import com.imagemaker.ui.utils.BuildInformation;  
  
import java.awt.BorderLayout;  
import java.awt.Dimension;  
import java.awt.HeadlessException;  
import java.awt.image.BufferedImage;  
import java.io.File;  
import javax.swing.ImageIcon;  
import javax.swing.JFrame;  
import javax.swing.JInternalFrame;  
  
public class ImageMakerMainFrame extends JFrame{  
  
 ImageIcon logo = new ImageIcon(ClassLoader.*getSystemResource*("img/logo.jpg"));  
  
 static ImagePanel *imagePanel*;  
  
 public ImageMakerMainFrame() throws HeadlessException {  
 super();  
 buildComponent();  
 }  
  
 private void buildComponent() {  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 setTitle(BuildInformation.*getWindowFixTitle*());  
 setIconImage(logo.getImage());  
 setSize(new Dimension(900, 500));  
 setResizable(false);  
 setLocation(200, 100);  
  
 ImageMakerMenu menu = new ImageMakerMenu();  
 setJMenuBar(menu);  
  
 ImageMakerToolBar toolBar = new ImageMakerToolBar();  
 getContentPane().add(toolBar, BorderLayout.*NORTH*);  
 }  
  
 public void openImage(File file)  
 {  
 JInternalFrame imageFrame = new JInternalFrame(file.getName(), true, true);  
 *imagePanel* = new ImagePanel(file);  
 imageFrame.getContentPane().add(*imagePanel*);  
 imageFrame.setVisible(true);  
 imageFrame.pack();  
 this.add(imageFrame);  
 }  
  
 public static BufferedImage getImage() {  
 if (*imagePanel* != null) {  
 return *imagePanel*.*getImage*();  
 }  
 return null;  
 }  
}

*ImageMakerMainFrameHandler.java*

package com.imagemaker.ui.component.forms;  
  
public class ImageMakerMainFrameHandler {  
  
 private static ImageMakerMainFrame *instance*;  
  
 public static ImageMakerMainFrame getInstance() {  
 if (*instance* == null) {  
 *instance* = new ImageMakerMainFrame();  
 }  
 return *instance*;  
 }  
}

*ImagePanel.java*

package com.imagemaker.ui.component.forms;  
  
import com.imagemaker.ui.utils.MessageDialogUtils;  
  
import java.awt.Component;  
import java.awt.Graphics;  
import java.awt.image.BufferedImage;  
import java.io.File;  
import javax.imageio.ImageIO;  
  
public class ImagePanel extends Component {  
 static BufferedImage *image*;  
 static BufferedImage *copyImage*;  
  
 public void paint(Graphics g) {  
 g.drawImage(*image*, 0, 0, null);  
 repaint();  
 }  
  
 public ImagePanel(File img) {  
 try {  
 *image* = ImageIO.*read*(img);  
 *copyImage* = *image*;  
 } catch (Exception ex) {  
 MessageDialogUtils.*showErrorMessage*(ImageMakerMainFrameHandler.*getInstance*().getRootPane(), "ERROR! Cannot open the file.");  
 }  
  
 }  
  
 public static BufferedImage getImage() {  
 return *image*;  
 }  
  
 public static void setImage(BufferedImage img) {  
 ImagePanel.*image* = img;  
 }  
  
 public static BufferedImage getCopyImage() {  
 return *copyImage*;  
 }  
  
 public static void setCopyImage(BufferedImage copyImage) {  
 ImagePanel.*copyImage* = copyImage;  
 }  
  
 public static void resetImage(){  
 *image* = *copyImage*;  
 }  
}

*ImageMakerMenu.java*

package com.imagemaker.ui.component.menu;  
  
import com.imagemaker.ui.listener.AboutActionListener;  
import com.imagemaker.ui.listener.ExitActionListener;  
import com.imagemaker.ui.listener.OpenFileActionListener;  
import com.imagemaker.ui.listener.SaveFileActionListener;  
import com.imagemaker.ui.listener.picture.blur.BlurActionListener;  
import com.imagemaker.ui.listener.picture.brightness.BrightnessActionListener;  
import com.imagemaker.ui.listener.picture.gray.GrayActionListener;  
import com.imagemaker.ui.listener.picture.negative.NegativeActionListener;  
import com.imagemaker.ui.listener.picture.reset.ResetActionListener;  
import com.imagemaker.ui.listener.picture.rotate.RotateActionListener;  
import com.imagemaker.ui.listener.picture.sharpening.SharpeningActionListener;  
import com.imagemaker.ui.listener.picture.zoomIn.ZoomInActionListener;  
import com.imagemaker.ui.listener.picture.zoomOut.ZoomOutActionListener;  
  
import javax.swing.\*;  
  
public class ImageMakerMenu extends JMenuBar {  
  
 ImageIcon iconOpen = new ImageIcon(ClassLoader.*getSystemResource*("img/icon/open.png"));  
 ImageIcon iconSave = new ImageIcon(ClassLoader.*getSystemResource*("img/icon/save.png"));  
 ImageIcon iconExit = new ImageIcon(ClassLoader.*getSystemResource*("img/icon/cancel.png"));  
  
 public ImageMakerMenu() {  
 JMenu fileMenu = fileMenu();  
 JMenu pictureMenu = pictureMenu();  
 JMenu view = viewMenu();  
 JMenu help = helpMenu();  
  
 add(fileMenu);  
 add(pictureMenu);  
 add(view);  
 add(help);  
 }  
  
 private JMenu fileMenu() {  
 JMenu fileMenu = new JMenu("File");  
  
 JMenuItem open = new JMenuItem("Open", iconOpen);  
 open.addActionListener(new OpenFileActionListener());  
  
 JMenuItem save = new JMenuItem("Save", iconSave);  
 save.addActionListener(new SaveFileActionListener());  
  
 JMenuItem exit = new JMenuItem("Exit", iconExit);  
 exit.addActionListener(new ExitActionListener());  
  
  
 fileMenu.add(open);  
 fileMenu.add(save);  
 fileMenu.addSeparator();  
 fileMenu.add(exit);  
  
 return fileMenu;  
 }  
  
 private JMenu pictureMenu() {  
 JMenu picture = new JMenu("Picture");  
 JMenu effects = new JMenu("Effects");  
  
 JMenuItem reset = new JMenuItem("Reset Image");  
 reset.addActionListener(new ResetActionListener());  
  
 JMenuItem grayScale = new JMenuItem("Grayscale");  
 grayScale.addActionListener(new GrayActionListener());  
  
 JMenuItem negative = new JMenuItem("Negative");  
 negative.addActionListener(new NegativeActionListener());  
  
 JMenuItem brightness = new JMenuItem("Brightness");  
 brightness.addActionListener(new BrightnessActionListener());  
  
 JMenuItem sharpen = new JMenuItem("Sharpening");  
 sharpen.addActionListener(new SharpeningActionListener());  
  
 JMenuItem rotation = new JMenuItem("Rotation");  
 rotation.addActionListener(new RotateActionListener());  
  
  
 JMenuItem blur = new JMenuItem("Blur");  
 blur.addActionListener(new BlurActionListener());  
  
  
 effects.add(grayScale);  
 effects.add(negative);  
 effects.add(sharpen);  
 effects.add(blur);  
 effects.add(rotation);  
 effects.add(brightness);  
  
  
 picture.add(effects);  
 picture.add(reset);  
 return picture;  
 }  
  
 private JMenu viewMenu() {  
 JMenu view = new JMenu("View");  
  
 JMenuItem zoomIn = new JMenuItem("Zoom In");  
 //zoomIn.addActionListener(new ZoomInActionListener();  
  
 JMenuItem zoomOut = new JMenuItem("Zoom Out");  
 //zoomOut.addActionListener(new ZoomOutActionListener();  
  
 view.add(zoomIn);  
 view.add(zoomOut);  
  
 return view;  
 }  
  
 private JMenu helpMenu() {  
 JMenu help = new JMenu("Help");  
  
 JMenuItem about = new JMenuItem("About us");  
 about.addActionListener(new AboutActionListener());  
  
 help.add(about);  
  
 return help;  
 }  
}

*ImageMakerToolBar.java*

package com.imagemaker.ui.component.toolbar;  
  
  
import com.imagemaker.ui.listener.ExitActionListener;  
import com.imagemaker.ui.listener.OpenFileActionListener;  
import com.imagemaker.ui.listener.SaveFileActionListener;  
import com.imagemaker.ui.listener.picture.blur.BlurActionListener;  
import com.imagemaker.ui.listener.picture.brightness.BrightnessActionListener;  
import com.imagemaker.ui.listener.picture.gray.GrayActionListener;  
import com.imagemaker.ui.listener.picture.negative.NegativeActionListener;  
import com.imagemaker.ui.listener.picture.reset.ResetActionListener;  
import com.imagemaker.ui.listener.picture.rotate.RotateActionListener;  
import com.imagemaker.ui.listener.picture.sharpening.SharpeningActionListener;  
import com.imagemaker.ui.listener.picture.zoomIn.ZoomInActionListener;  
import com.imagemaker.ui.listener.picture.zoomOut.ZoomOutActionListener;  
  
import javax.swing.\*;  
  
public class ImageMakerToolBar extends JToolBar {  
  
 ImageIcon iconOpen = new ImageIcon(ClassLoader.*getSystemResource*("img/icon/open.png"));  
 ImageIcon iconSave = new ImageIcon(ClassLoader.*getSystemResource*("img/icon/save.png"));  
 ImageIcon iconZoomIn = new ImageIcon(ClassLoader.*getSystemResource*("img/icon/zoomIn.png"));  
 ImageIcon iconZoomOut = new ImageIcon(ClassLoader.*getSystemResource*("img/icon/zoomOut.png"));  
 ImageIcon iconRotate = new ImageIcon(ClassLoader.*getSystemResource*("img/icon/rotate.png"));  
 ImageIcon iconBrightness = new ImageIcon(ClassLoader.*getSystemResource*("img/icon/brightness.png"));  
 ImageIcon iconNegative = new ImageIcon(ClassLoader.*getSystemResource*("img/icon/negative.png"));  
 ImageIcon iconGrayscale = new ImageIcon(ClassLoader.*getSystemResource*("img/icon/grayscale.png"));  
 ImageIcon iconSharpening = new ImageIcon(ClassLoader.*getSystemResource*("img/icon/sharpening.png"));  
 ImageIcon iconBlur = new ImageIcon(ClassLoader.*getSystemResource*("img/icon/blur.png"));  
 ImageIcon iconReset = new ImageIcon(ClassLoader.*getSystemResource*("img/icon/reset.png"));  
 ImageIcon iconExit = new ImageIcon(ClassLoader.*getSystemResource*("img/icon/cancel.png"));  
  
 public ImageMakerToolBar() {  
 super("ToolBar",*HORIZONTAL*);  
  
 JButton openButton = new JButton(iconOpen);  
 openButton.setToolTipText("Open");  
 openButton.addActionListener(new OpenFileActionListener());  
  
 JButton saveButton = new JButton(iconSave);  
 saveButton.setToolTipText("Save");  
 saveButton.addActionListener(new SaveFileActionListener());  
  
 JButton zoomInButton = new JButton(iconZoomIn);  
 zoomInButton.setToolTipText("Zoom In");  
 zoomInButton.addActionListener(new ZoomInActionListener());  
  
 JButton zoomOutButton = new JButton(iconZoomOut);  
 zoomOutButton.setToolTipText("Zoom Out");  
 zoomOutButton.addActionListener(new ZoomOutActionListener());  
  
 JButton rotateButton = new JButton(iconRotate);  
 rotateButton.setToolTipText("Rotate");  
 rotateButton.addActionListener(new RotateActionListener());  
  
 JButton brightnessButton = new JButton(iconBrightness);  
 brightnessButton.setToolTipText("Brightness");  
 brightnessButton.addActionListener(new BrightnessActionListener());  
  
 JButton negativeButton = new JButton(iconNegative);  
 negativeButton.setToolTipText("Negative");  
 negativeButton.addActionListener(new NegativeActionListener());  
  
 JButton grayscaleButton = new JButton(iconGrayscale);  
 grayscaleButton.setToolTipText("Grayscale");  
 grayscaleButton.addActionListener(new GrayActionListener());  
  
 JButton sharpeningButton = new JButton(iconSharpening);  
 sharpeningButton.setToolTipText("Sharpening");  
 sharpeningButton.addActionListener(new SharpeningActionListener());  
  
 JButton blurButton = new JButton(iconBlur);  
 blurButton.setToolTipText("Blur");  
 blurButton.addActionListener(new BlurActionListener());  
  
 JButton resetButton = new JButton(iconReset);  
 resetButton.setToolTipText("Reset");  
 resetButton.addActionListener(new ResetActionListener());  
  
 JButton exitButton = new JButton(iconExit);  
 exitButton.setToolTipText("Exit");  
 exitButton.addActionListener(new ExitActionListener());  
  
 add(openButton);  
 add(saveButton);  
 addSeparator();  
 add(zoomInButton);  
 add(zoomOutButton);  
 add(rotateButton);  
 addSeparator();  
 add(brightnessButton);  
 add(negativeButton);  
 add(grayscaleButton);  
 add(sharpeningButton);  
 add(blurButton);  
 addSeparator();  
 add(resetButton);  
 addSeparator();  
 add(exitButton);  
 }  
}

*OpenFileChooserListener.java*

package com.imagemaker.ui.listener.file;  
  
import com.imagemaker.ui.component.file.FileChooser;  
import com.imagemaker.ui.component.forms.ImageMakerMainFrameHandler;  
import com.imagemaker.ui.utils.MessageDialogUtils;  
  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.io.File;  
import javax.swing.JFileChooser;  
  
public class OpenFileChooserListener implements ActionListener {  
  
 public OpenFileChooserListener() {  
 super();  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 try {  
 FileChooser fileChooser = (FileChooser) e.getSource();  
 if (JFileChooser.*APPROVE\_SELECTION*.equals(e.getActionCommand())){  
 File file = fileChooser.getSelectedFile();  
 ImageMakerMainFrameHandler.*getInstance*().openImage(file);  
 }  
 } catch (Exception ex) {  
 MessageDialogUtils.*showErrorMessage*(ImageMakerMainFrameHandler.*getInstance*().getRootPane(), "ERROR! Cannot open the file.");  
 }  
 }  
  
}

*SaveFileChooserListener.java*

package com.imagemaker.ui.listener.file;  
  
import com.imagemaker.ui.component.file.FileChooser;  
import com.imagemaker.ui.component.forms.ImageMakerMainFrameHandler;  
import com.imagemaker.ui.component.forms.ImagePanel;  
import com.imagemaker.ui.utils.MessageDialogUtils;  
  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.io.File;  
import javax.imageio.ImageIO;  
import javax.swing.JFileChooser;  
  
public class SaveFileChooserListener implements ActionListener {  
  
 public SaveFileChooserListener() {  
 super();  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 try {  
 FileChooser fileChooser = (FileChooser) e.getSource();  
 if (JFileChooser.*APPROVE\_SELECTION*.equals(e.getActionCommand())) {  
 File file = fileChooser.getSelectedFile();  
 if (fileChooser.getFileFilter() == FileChooser.*FILTER\_JPG*) {  
 ImageIO.*write*(ImagePanel.*getImage*(), "jpg", new File(file.getPath() + ".jpg"));  
 } else if (fileChooser.getFileFilter() == FileChooser.*FILTER\_PNG*) {  
 ImageIO.*write*(ImagePanel.*getImage*(), "png", new File(file.getPath() + ".png"));  
 }  
 }  
 } catch (Exception ex) {  
 MessageDialogUtils.*showErrorMessage*(ImageMakerMainFrameHandler.*getInstance*().getRootPane(), "ERROR! Cannot open the file.");  
 }  
  
 }  
}

*AboutActionListener.java*

package com.imagemaker.ui.listener;  
  
import com.imagemaker.ui.component.forms.AboutFrame;  
  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
  
public class AboutActionListener implements ActionListener {  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 AboutFrame about = new AboutFrame();  
 about.setVisible(true);  
 }  
}

*ExitActionListener.java*

package com.imagemaker.ui.listener;  
  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
public class ExitActionListener implements ActionListener {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 System.*exit*(1);  
 }  
}

*OpenFileActionListener.java*

package com.imagemaker.ui.listener;  
  
import com.imagemaker.ui.component.forms.ImageMakerMainFrameHandler;  
import com.imagemaker.ui.component.file.FileChooser;  
  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
public class OpenFileActionListener implements ActionListener {  
  
 public OpenFileActionListener() {  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 FileChooser fileChooser = new FileChooser(true);  
 fileChooser.showOpenDialog(ImageMakerMainFrameHandler.*getInstance*());  
 }  
}

*SaveFileActionListener.java*

package com.imagemaker.ui.listener;  
  
import com.imagemaker.ui.component.forms.ImageMakerMainFrameHandler;  
import com.imagemaker.ui.component.file.FileChooser;  
  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
  
public class SaveFileActionListener implements ActionListener {  
  
 public SaveFileActionListener() {  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 FileChooser fileChooser = new FileChooser(false);  
 fileChooser.showSaveDialog(ImageMakerMainFrameHandler.*getInstance*());  
 }  
}

*BlurActionHelper.java*

package com.imagemaker.ui.listener.picture.blur;  
  
import java.awt.Graphics;  
import java.awt.image.BufferedImage;  
import java.awt.image.BufferedImageOp;  
import java.awt.image.ConvolveOp;  
import java.awt.image.Kernel;  
  
public class BlurActionHelper {  
  
 public static BufferedImage createBlurredImage(BufferedImage img) {  
  
 BufferedImage newimg = new BufferedImage(img.getWidth(null), img.getHeight(null),BufferedImage.*TYPE\_INT\_BGR*);  
  
 Graphics g = newimg.getGraphics();  
 g.drawImage(img, 455, 255, null);  
  
 float[] blurKernel = {  
 1 / 9f, 1 / 9f, 1 / 9f,  
 1 / 9f, 1 / 9f, 1 / 9f,  
 1 / 9f, 1 / 9f, 1 / 9f  
 };  
  
 BufferedImageOp blur = new ConvolveOp(new Kernel(3, 3, blurKernel));  
 img = blur.filter(img, new BufferedImage(img.getWidth(), img.getHeight(), img.getType()));  
 g.dispose();  
 return img;  
 }  
}

*BlurActionListener.java*

package com.imagemaker.ui.listener.picture.blur;  
  
import com.imagemaker.ui.component.forms.ImagePanel;  
  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.image.BufferedImage;  
  
  
public class BlurActionListener implements ActionListener {  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 BufferedImage image = BlurActionHelper.*createBlurredImage*(ImagePanel.*getImage*());  
 ImagePanel.*setImage*(image);  
 }  
}

*BrightnessActionHelper.java*

package com.imagemaker.ui.listener.picture.brightness;  
  
  
import java.awt.Color;  
import java.awt.image.BufferedImage;  
  
public class BrightnessActionHelper {  
  
 public static void changeBrightness(int increasingFactor, BufferedImage img) {  
 //size of input image  
  
 int w = img.getWidth();  
 int h = img.getHeight();  
 //Pixel by pixel navigation loop  
 for (int i = 0; i < w; i++) {  
 for (int j = 0; j < h; j++) {  
 //get the RGB component of input imge pixel  
 Color color = new Color(img.getRGB(i, j));  
 int r, g, b;  
 //change the value of each component  
 r = color.getRed() + increasingFactor;  
 g = color.getGreen() + increasingFactor;  
 b = color.getBlue() + increasingFactor;  
 //r,g,b values which are out of the range 0 to 255 should set to 0 or 255  
 if (r >= 256) {  
 r = 255;  
 } else if (r < 0) {  
 r = 0;  
 }  
  
 if (g >= 256) {  
 g = 255;  
 } else if (g < 0) {  
 g = 0;  
 }  
  
 if (b >= 256) {  
 b = 255;  
 } else if (b < 0) {  
 b = 0;  
 }  
  
 //set output image pixel component  
 img.setRGB(i, j, new Color(r, g, b).getRGB());  
 }  
 }  
 }  
}

*BrightnessActionListener.java*

package com.imagemaker.ui.listener.picture.brightness;  
  
import com.imagemaker.ui.component.forms.ImageMakerMainFrameHandler;  
import com.imagemaker.ui.component.forms.ImagePanel;  
  
import java.awt.BorderLayout;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import javax.swing.JSlider;  
  
  
public class BrightnessActionListener implements ActionListener {  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 final JSlider slider = new JSlider(JSlider.*HORIZONTAL*, 0, 50,25);  
 slider.setMinorTickSpacing(1);  
 slider.setMajorTickSpacing(10);  
 slider.setPaintTicks(true);  
 slider.setPaintLabels(true);  
 ImageMakerMainFrameHandler.*getInstance*().add(slider, BorderLayout.*SOUTH*);  
 ImageMakerMainFrameHandler.*getInstance*().setVisible(true);  
 slider.addChangeListener(new javax.swing.event.ChangeListener() {  
 public void stateChanged(javax.swing.event.ChangeEvent evt) {  
  
 BrightnessActionHelper.*changeBrightness*((slider.getValue()) / 10, ImagePanel.*getImage*());  
  
 }  
 });  
  
 }  
}

*GrayActionHelper.java*

package com.imagemaker.ui.listener.picture.gray;  
  
import java.awt.Color;  
import java.awt.image.BufferedImage;  
import java.io.IOException;  
  
public class GrayActionHelper {  
  
 public static void createGrayImage(BufferedImage image)throws IOException  
 {  
  
 int w=0,h=0,width,height;  
 width=image.getWidth();  
 height=image.getHeight();  
 for(w = 0; w <width ; w++)  
 {  
 for(h = 0 ; h <height ; h++)  
 {  
 // BufferedImage.getRGB() saves the colour of the pixel as a single integer.  
 // use Color(int) to grab the RGB values individually.  
 Color color = new Color(image.getRGB(w, h));  
 // get the luminosity value as an integer  
 int luminosity = (int) (0.2126 \* color.getRed() + 0.7152 \* color.getGreen() + 0.0722 \* color.getBlue());  
 // create a new Color object using the luminosity colour as the red, green and blue  
 // colour values  
 Color lum = new Color(luminosity, luminosity, luminosity);  
  
 // set the pixel at that position to the new Color object using Color.getRGB().  
 image.setRGB(w, h, lum.getRGB());  
 }  
 }  
  
 }  
}

*GrayActionListener.java*

package com.imagemaker.ui.listener.picture.gray;  
  
import com.imagemaker.ui.component.forms.ImageMakerMainFrameHandler;  
import com.imagemaker.ui.component.forms.ImagePanel;  
import com.imagemaker.ui.utils.MessageDialogUtils;  
  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
  
public class GrayActionListener implements ActionListener {  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 try {  
 GrayActionHelper.*createGrayImage*(ImagePanel.*getImage*());  
 } catch (Exception ex) {  
 MessageDialogUtils.*showErrorMessage*(ImageMakerMainFrameHandler.*getInstance*().getRootPane(), "ERROR! This action cannot be performed.");  
 ImagePanel.*resetImage*();  
 }  
 }  
}

*NegativeActionHelper.java*

package com.imagemaker.ui.listener.picture.negative;  
  
import java.awt.image.BufferedImage;  
  
public class NegativeActionHelper {  
  
 public static BufferedImage getNegativeImage(BufferedImage img) {  
 int w1 = img.getWidth();  
 int h1 = img.getHeight();  
 // int value[][] = new int[w1][h1];  
 BufferedImage neg = new BufferedImage(w1, h1, 1);  
 int value, alpha, r, g, b;  
 for (int i = 0; i < w1; i++) {  
 for (int j = 0; j < h1; j++) {  
 value = img.getRGB(i, j); // store value  
 alpha = *getAlpha*(value);  
 r = 255 - *getRed*(value);  
 g = 255 - *getGreen*(value);  
 b = 255 - *getBlue*(value);  
  
 value = *createRGB*(alpha, r, g, b);  
 neg.setRGB(i, j, value);  
 }  
 }  
 return neg;  
 }  
  
 public static int createRGB(int alpha, int r, int g, int b) {  
 int rgb = (alpha << 24) + (r << 16) + (g << 8) + b;  
 return rgb;  
 }  
  
 public static int getAlpha(int rgb) {  
 return (rgb >> 24) & 0xFF;  
 }  
  
 public static int getRed(int rgb) {  
 return (rgb >> 16) & 0xFF;  
 }  
  
 public static int getGreen(int rgb) {  
 return (rgb >> 8) & 0xFF;  
 }  
  
 public static int getBlue(int rgb) {  
 return rgb & 0xFF;  
 }  
}

*NegativeActionListener.java*

package com.imagemaker.ui.listener.picture.negative;  
  
import com.imagemaker.ui.component.forms.ImagePanel;  
  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.image.BufferedImage;  
  
  
public class NegativeActionListener implements ActionListener {  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 BufferedImage image = NegativeActionHelper.*getNegativeImage*(ImagePanel.*getImage*());  
 ImagePanel.*setImage*(image);  
 }  
}

*ResetActionListener.java*

package com.imagemaker.ui.listener.picture.reset;  
  
import com.imagemaker.ui.component.forms.ImagePanel;  
  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
  
public class ResetActionListener implements ActionListener {  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 ImagePanel.*resetImage*();  
 }  
}

*RotateActionHelper.java*

package com.imagemaker.ui.listener.picture.rotate;  
  
  
import java.awt.Graphics2D;  
import java.awt.geom.AffineTransform;  
import java.awt.image.BufferedImage;  
import java.io.IOException;  
  
public class RotateActionHelper {  
  
 public static BufferedImage rotate(BufferedImage img,double degrees)throws IOException  
 {  
 BufferedImage newImage = new BufferedImage(img.getWidth(), img.getHeight(), img.getType());  
  
 AffineTransform tx = new AffineTransform();  
 tx.rotate(Math.*toRadians*(degrees), img.getWidth() / 2, img.getHeight() / 2);  
  
 Graphics2D g2 = newImage.createGraphics();  
 g2.drawImage(img, tx, null);  
  
 img=newImage;  
 g2.dispose();  
 return img;  
 }  
}

*RotateActionListener.java*

package com.imagemaker.ui.listener.picture.rotate;  
  
import com.imagemaker.ui.component.forms.ImageMakerMainFrameHandler;  
import com.imagemaker.ui.component.forms.ImagePanel;  
import com.imagemaker.ui.utils.MessageDialogUtils;  
  
import java.awt.BorderLayout;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import javax.swing.\*;  
import javax.swing.event.ChangeEvent;  
import javax.swing.event.ChangeListener;  
  
  
public class RotateActionListener implements ActionListener {  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 final JSlider slider = new JSlider(JSlider.*HORIZONTAL*, 0, 360, 0);  
 slider.setPaintTicks(true);  
 slider.setMinorTickSpacing(5);  
 slider.setMajorTickSpacing(10);  
 slider.setValue(0);  
  
 ImageMakerMainFrameHandler.*getInstance*().add(slider, BorderLayout.*SOUTH*);  
 ImageMakerMainFrameHandler.*getInstance*().setVisible(true);  
 slider.addChangeListener(new ChangeListener() {  
 @Override  
 public void stateChanged(ChangeEvent e) {  
 try {  
 ImagePanel.*setImage*(RotateActionHelper.*rotate*(ImagePanel.*getCopyImage*(), slider.getValue()));  
 } catch (Exception ei) {  
 MessageDialogUtils.*showErrorMessage*(ImageMakerMainFrameHandler.*getInstance*().getRootPane(), "ERROR! This action cannot be performed.");  
 ImagePanel.*resetImage*();  
 }  
 }  
 });  
 }  
}

*SharpeningActionHelper.java*

package com.imagemaker.ui.listener.picture.sharpening;  
  
import java.awt.Graphics;  
import java.awt.image.BufferedImage;  
import java.awt.image.BufferedImageOp;  
import java.awt.image.ConvolveOp;  
import java.awt.image.Kernel;  
  
public class SharpeningActionHelper {  
  
 public static BufferedImage createSharpenImage(BufferedImage img) {  
  
 BufferedImage newimg = new BufferedImage(img.getWidth(null), img.getHeight(null),BufferedImage.*TYPE\_INT\_BGR*);  
  
 Graphics g = newimg.getGraphics();  
 g.drawImage(img, 455, 255, null);  
  
 float[] sharpKernel = {0.0f, -1.0f, 0.0f,  
 -1.0f, 5.0f, -1.0f,  
 0.0f, -1.0f, 0.0f};  
  
 BufferedImageOp blur = new ConvolveOp(new Kernel(3, 3, sharpKernel));  
 img = blur.filter(img, new BufferedImage(img.getWidth(), img.getHeight(), img.getType()));  
 g.dispose();  
 return img;  
 }  
}

*SharpeningActionListener.java*

package com.imagemaker.ui.listener.picture.sharpening;  
  
import com.imagemaker.ui.component.forms.ImagePanel;  
  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.image.BufferedImage;  
  
  
public class SharpeningActionListener implements ActionListener {  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 BufferedImage image = SharpeningActionHelper.*createSharpenImage*(ImagePanel.*getImage*());  
 ImagePanel.*setImage*(image);  
 }  
}

*ZoomInActionHelpe.java*

package com.imagemaker.ui.listener.picture.zoomIn;  
  
import java.awt.\*;  
import java.awt.image.BufferedImage;  
  
public class ZoomInActionHelper {  
 public static BufferedImage createZoomInImage(BufferedImage img) {  
 int newImageWidth = img.getWidth()\*2;  
 int newImageHeight = img.getHeight()\*2;  
 BufferedImage zoomInImage = new BufferedImage(newImageWidth , newImageHeight, img.getType());  
 Graphics2D g = zoomInImage.createGraphics();  
 g.drawImage(img, 0, 0, newImageWidth , newImageHeight , null);  
 g.dispose();  
  
 return zoomInImage;  
 }  
}

*ZoomInActionListener.java*

package com.imagemaker.ui.listener.picture.zoomIn;  
  
import com.imagemaker.ui.component.forms.ImagePanel;  
  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.image.BufferedImage;  
  
public class ZoomInActionListener implements ActionListener {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 BufferedImage image = ZoomInActionHelper.*createZoomInImage*(ImagePanel.*getImage*());  
 ImagePanel.*setImage*(image);  
 }  
}

*ZoomOutActionHelper.java*

package com.imagemaker.ui.listener.picture.zoomOut;  
  
import com.imagemaker.ui.component.forms.ImagePanel;  
  
import java.awt.\*;  
import java.awt.image.BufferedImage;  
  
public class ZoomOutActionHelper {  
 public static BufferedImage createZoomOutImage(BufferedImage img) {  
 int newImageWidth1 = img.getWidth()/2;  
 int newImageHeight1 = img.getHeight()/2;  
 BufferedImage zoomInImage1 = new BufferedImage(newImageWidth1 , newImageHeight1, img.getType());  
 Graphics2D g = zoomInImage1.createGraphics();  
 g.drawImage(img, 0, 0, newImageWidth1 , newImageHeight1 , null);  
 g.dispose();  
  
 return zoomInImage1;  
 }  
}

*ZoomOutActionListener.java*

package com.imagemaker.ui.listener.picture.zoomOut;  
  
import com.imagemaker.ui.component.forms.ImagePanel;  
  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.image.BufferedImage;  
  
public class ZoomOutActionListener implements ActionListener {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 BufferedImage image = ZoomOutActionHelper.*createZoomOutImage*(ImagePanel.*getImage*());  
 ImagePanel.*setImage*(image);  
 }  
}

*BuildInformation.java*

package com.imagemaker.ui.utils;  
  
  
public class BuildInformation {  
  
 public static final String *VERSION\_NUMBER* = "0.1";  
 private static String *fixTitle* = "Image Maker ";  
  
 public static String getWindowFixTitle() {  
 return *fixTitle* + BuildInformation.*VERSION\_NUMBER*;  
 }  
}

*ImageUtils.java*

package com.imagemaker.ui.utils;  
  
  
import java.awt.Color;  
import java.awt.Image;  
import java.awt.image.BufferedImage;  
  
public class ImageUtils {  
  
 public static BufferedImage scaleImage(BufferedImage image, double panelWidth){  
 if (image == null){  
 return null;  
 }  
 double widthScale = *calculateImageScale*(image.getWidth(), panelWidth);  
  
 int scaledWidth = (int) (image.getWidth() \* widthScale);  
 int scaledHeight = (int) (image.getHeight() \* widthScale);  
  
 BufferedImage newImage = *scaledImage*(image, scaledWidth, scaledHeight);  
  
 return newImage;  
 }  
  
 private static double calculateImageScale(double imageWidth, double frameWidth){  
 if(imageWidth < frameWidth){  
 return 1;  
 }else{  
 return frameWidth/imageWidth;  
 }  
 }  
  
 private static BufferedImage scaledImage(BufferedImage image, int scaledWidth, int scaledHeight) {  
 Image scaledImage = image.getScaledInstance(scaledWidth, scaledHeight, Image.*SCALE\_SMOOTH*);  
  
 BufferedImage newImage = new BufferedImage(scaledWidth, scaledHeight, BufferedImage.*TYPE\_INT\_RGB*);  
 newImage.createGraphics().drawImage(scaledImage, 0, 0, Color.*WHITE*, null);  
  
 return newImage;  
 }  
}

*MessageDialogUtils.java*

package com.imagemaker.ui.utils;  
  
import javax.swing.JComponent;  
import javax.swing.JOptionPane;  
  
public class MessageDialogUtils {  
  
 public static void showErrorMessage(JComponent component, String message){  
 JOptionPane.*showMessageDialog*(component, message);  
 }  
}

**9.Заключение**

Целью данного проекта являлась разработка на языке программирования Java утилиты обработки изображений. Подводя итоги, можно сказать, что поставленная цель была успешно достигнута. ImageMaker представляет собой оконное приложение, позволяющее открыть изображение в формате JPG и PNG и выбрать из меню некоторые эффекты обработки, например, размытие, масштабирование, поворот, цветовые преобразования и т.д.

**Список используемой литературы**

1. Thinking in Java, 4nd edition, Revision by Bruce Eckel.
2. Swing: Эффектные пользовательские интерфейсы. Библиотека программиста. Портянкин И. А. СПб: "Питер", 2005 г
3. Java. Промышленное программирование : практ. пособие / И.Н. Блинов, В.С. Романчик. – Минск : УниверсалПресс, 2007. – 704 с.
4. <http://java.sun.com/developer/onlineTraining/GUI/Swing1/shortcourse.html#JFCButton>
5. <https://stackoverflow.com/documentation>