ФЕДЕРАЛЬНОЕ АГЕНТСТВО МОРСКОГО И РЕЧНОГО ТРАНСПОРТА

ФЕДЕРАЛЬНОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«Морской государственный университет имени адмирала Г.И. Невельского»

Факультет электроники и информационных технологий

Кафедра Вычислительной техники

Дисциплина «Программирование»

**Курсовая работа**

«Векторный редактор. Объектно-ориентированное программирование.»

Выполнил: студент гр. *214.21*

*Гармышев Дмитрий Евгеньевич*

Проверил: преподаватель

Хованец В.А.

Оглавление

[Введение 4](#_Toc74568159)

[Задание 5](#_Toc74568160)

[План разработки проекта 6](#_Toc74568161)

[Набросок тестового проекта 6](#_Toc74568162)

[Разработка новой структуры данных 6](#_Toc74568163)

[Хранение точек в древовидной структуре. 7](#_Toc74568164)

[Хранение точек и фигур в списках, но фигуры состоят из рёбер. 13](#_Toc74568165)

[Хранение точек и фигур в списках, без использования рёбер. 15](#_Toc74568166)

[Разработка и тестирование программы 16](#_Toc74568167)

[Общее описание проекта 17](#_Toc74568168)

[Структура модулей 17](#_Toc74568169)

[Пакет активностей 17](#_Toc74568170)

[Пакет адаптеров 18](#_Toc74568171)

[Пакет компонентов 18](#_Toc74568172)

[Пакет фрагментов 19](#_Toc74568173)

[Пакет структуры данных 20](#_Toc74568174)

[Пакет визуальных компонентов приложения 21](#_Toc74568175)

[Класс Saver 21](#_Toc74568176)

[Типы данных и имена переменных 21](#_Toc74568177)

[Классы структуры данных 21](#_Toc74568178)

[Классы компонентов 24](#_Toc74568179)

[Классы активностей 26](#_Toc74568180)

[Классы фрагментов 26](#_Toc74568181)

[Классы адаптеров 28](#_Toc74568182)

[Классы view-элементов 29](#_Toc74568183)

[Структура меню проекта 30](#_Toc74568184)

[Главное меню 30](#_Toc74568185)

[Рабочая область 31](#_Toc74568186)

[Экран создания фигуры 31](#_Toc74568187)

[Экран создания шаблона 31](#_Toc74568188)

[Описания основных процедур 32](#_Toc74568189)

[Процедуры и методы структуры данных 32](#_Toc74568190)

[Процедуры классов, отвечающих за графику 34](#_Toc74568191)

[Процедуры классов фрагментов 36](#_Toc74568192)

[Процедуры компонентов 37](#_Toc74568193)

[Примеры графических построений в приложении 38](#_Toc74568194)

[Требования к оформлению кода и выбору имен 39](#_Toc74568195)

[Алгоритмы 40](#_Toc74568196)

[Тексты модулей 41](#_Toc74568197)

[Инструкция к приложению 42](#_Toc74568198)

[Заключение 43](#_Toc74568199)

[Список литературы 44](#_Toc74568200)

# Введение

Основная тема данной работы – векторный редактор, который позволяет быстро и удобно создавать изображения, основанные на геометрических фигурах. Его особенность заключается в том, что мы можем легко создавать и редактировать фигуры.

Для создания этого проекта было использовано Объектно-ориентированное программирование (далее ООП). Такой подход позволяет явно представлять объекты (например, объект точки или фигуры), указывать их свойства и функции. Помимо ООП, уделено внимание алгоритмам поиска фигур, точек и их обработки.

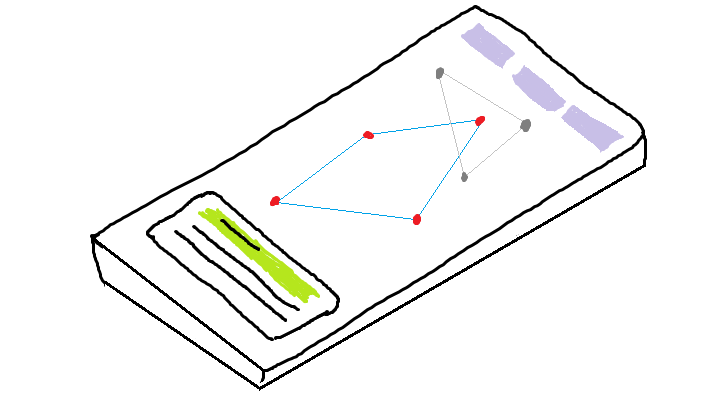
Проект написан для ОС Android, на языке Java, с использованием событийно-управляемого программирования (например, нажатие на экран, кнопки и т.д.).

Используемые инструменты включают в себя IDE Android Studio для написания кода, и браузер для поиска полезной информации в процессе разработки ПО.

Основная цель данного проекта – закрепить знания в области ООП, научиться создавать сложные структуры данных и алгоритмы для их обработки.

Структура данных включает в себя две части. Первая часть включает в себя все точки, которые отображаются на экране, и хранит только их координаты. Вторая часть хранит в себе все данные о фигурах, их шаблонах. Фигуры в данной структуре не содержат в себе объекты точек, а только ссылки на них в виде индексов.

P.S. Мои знания на момент написания программы и на момент написания отчёта очень рознятся, и сейчас я бы сделал всё по другому.



# Задание

1. Реализовать обработку типов геометрических фигур на плоскости, для чего создать структуру классов и необходимые структуры данных.

Использовать списки, связанные списки, и другие необходимые структуры.

1. Создать новую фигуру (из перечня фигур) щелчками мышью – суть этого задания заключается в том, чтобы при нажатии на экран, создавались точки и соединялись линиями, тем самым образуя фигуру.
2. Выбрать нужные фигуры или одну фигуру щелчками мышью на фигуре. По нажатию на экран, и по полученным координатам нажатия мы должны правильно распознать, какую фигуру мы выбрали.
3. Удалить выбранные фигуры. Данное задание вытекает из второго. Просто удаляем выбранные фигуры из списка, а также точки, которые принадлежат этой фигуре.
4. Копировать выбранные и поместить в нужное место. Данное задание вытекает из второго. Мы создаём копии выбранных фигур и их точек, но уже смещаем их в другое место, куда нажал пользователь.
5. Переместить выбранные фигуры. Данное задание вытекает из второго. Мы просто перемещаем выбранные фигуры, туда куда нажал пользователь.
6. Выполнить геометрические преобразования выбранных фигур из перечня меню.
7. Вести разработку на языке Java в среде Android Studio.
8. Протестировать приложение на мобильных устройствах.

# План разработки проекта

## Набросок тестового проекта

Для того чтобы приблизительно понять, что конкретно требуется разработать в этом проекте, было создано тестовое приложение (черновик), в котором использовались самые примитивные структуры данных (довольно неоптимизированные, и не пригодные для дальнейшего развития проекта), не самые лучшие алгоритмы, но зато примерно было понятно куда двигаться дальше. Из функций, это приложение включало в себя создание одной фигуры, редактирование её точек (простое перемещение и удаление их), и элементы событийно-управляемого программирования.

Пускай на это приложение было потрачено время, зато я смог понять с какими «подводными камнями» я могу столкнуться в процессе разработки основного приложения, смог определить для себя самые важные функции приложения, и продумал дальнейшую архитектуру (не структур данных, а моментов касаемо работы самого приложения), тем самым значительно снизив вероятность глобальных переработок в дальнейшем.

## Разработка новой структуры данных

Данный этап является тоже самым немаловажным. Изначально было очень много вариантов создания структуры данных, основанных на разных принципах.

1. Древовидная структура – в которой точки хранятся в виде дерева по 4 направлениям.
2. Структура, в которой точки хранятся в отдельном списке, но фигура состоит из рёбер.
3. Структура, в которой точки хранятся в отдельном списке, но фигура содержит в себе просто ссылки на рёбра.

Далее будут кратко описаны все эти принципы, и их реализация.

### Хранение точек в древовидной структуре.

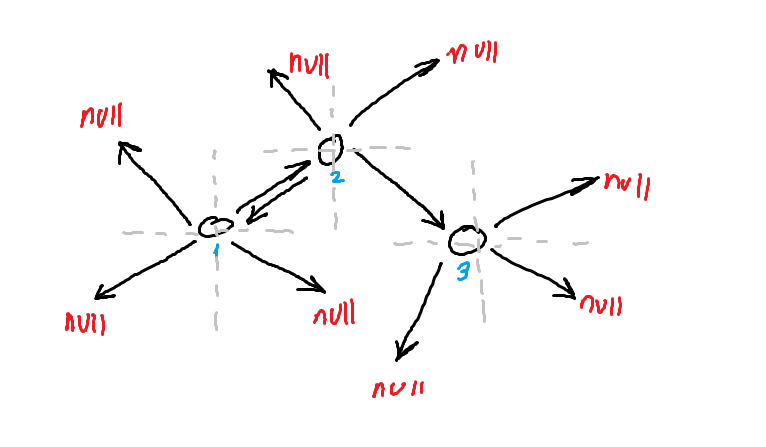


Рис 1. Образное представление древовидной структуры

Основная суть данной структуры заключается в том, что точки хранятся в виде ссылок друг на друга, и учитывается их положение относительно друг друга.

Например, если в структуре уже есть точки, и я хочу добавить новую, то я прохожусь по каждой точке и проверяю, в каком направлении лежит новая точка относительно старой. Если оно правее и выше, то я иду по ветке дальше, и опять сверяю направление. Как только я нахожу пустое место, оно заполняется ссылкой на новую точку. При этом у каждой новой точке остаётся ссылка на родительскую точку.

У этого способа есть свои плюсы и недостатки.

Плюсы:

* Очень быстрый поиск точек. Нам не нужно проходиться по каждой точке в структуре (как например при использовании списков), мы просто идём по ветвям в поисках ближайших координат и затрагиваем только интересующие нас элементы.

Минусы:

* Редактирование точек. Дело в том, что при редактировании (перетаскивании точек), меняется их положение относительно друг друга. И поэтому, при каждой смене координат приходилось удалять точку из списка и создавать новую, но уже с новыми координатами, которая встанет на своё место. Из-за этого сбивались все ссылки на точки в структуре фигур.
* Сложный контроль над точками. Мы не можем в любой момент и быстро обратиться к элементу по индексу (как например в списках или массивах), поэтому постоянно приходилось искать точку снова по всей структуре. (Именно на момент написания программы у меня были некие пробелы в знаниях, и сейчас я понимаю, что можно было просто обращаться к точке по ссылке на неё).
* Использование более сложных алгоритмов для обработки точек и фигур.
* Более сложный алгоритм удаления точек из структуры, т.к. нужно учитывать ссылки дочерних и родительских точек друг на друга.

Реализация данной структуры на Java.

Вот такой наполнение у класса Point (скриншот без процедур и методов).

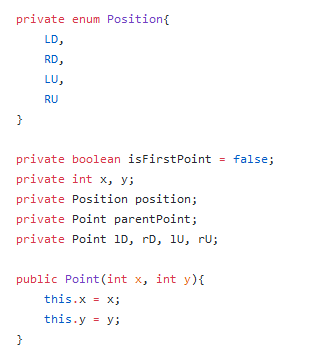


Рис 2. Наполнение класса Point в древовидной структуре.

Тут у нас хранится флаг, дающий знать, какая это точка в структуре, первая или нет, её координаты, текущее положение точки относительно родительской точки, ссылка на родительскую точку, 4 ссылки на дочерние точки, где каждая ссылка отвечает за своё направление.

Давайте рассмотрим самые основные процедуры в этом классе.

Процедура добавления новых точек в структуру:

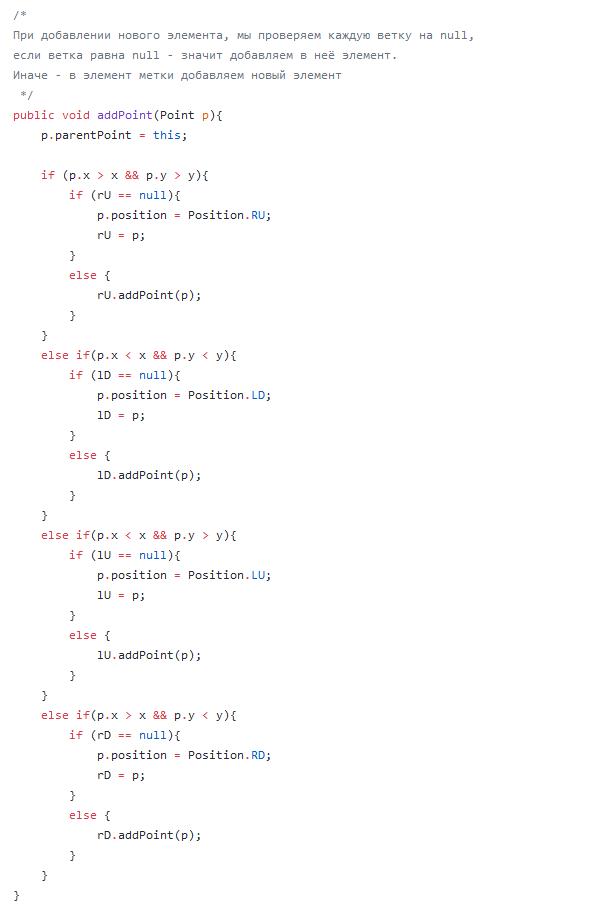


Рис 3. Процедура добавления новых точек в древовидную структуру.

Поиск точек в структуре. То о чём говорилось выше. Мы не просто перебираем каждый элемент, а идём по ветвям и ищем нужный элемент.

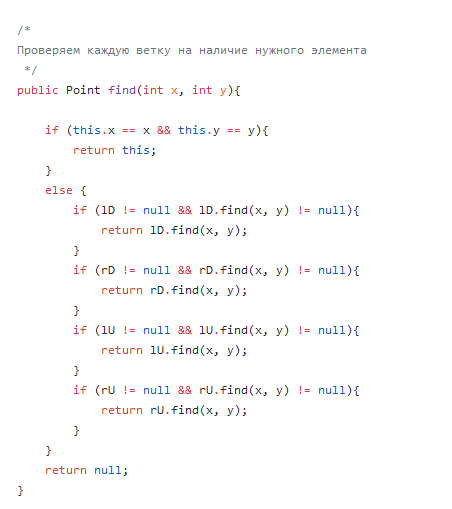


Рис 4. Реализация поиска точек в древовидной структуре.

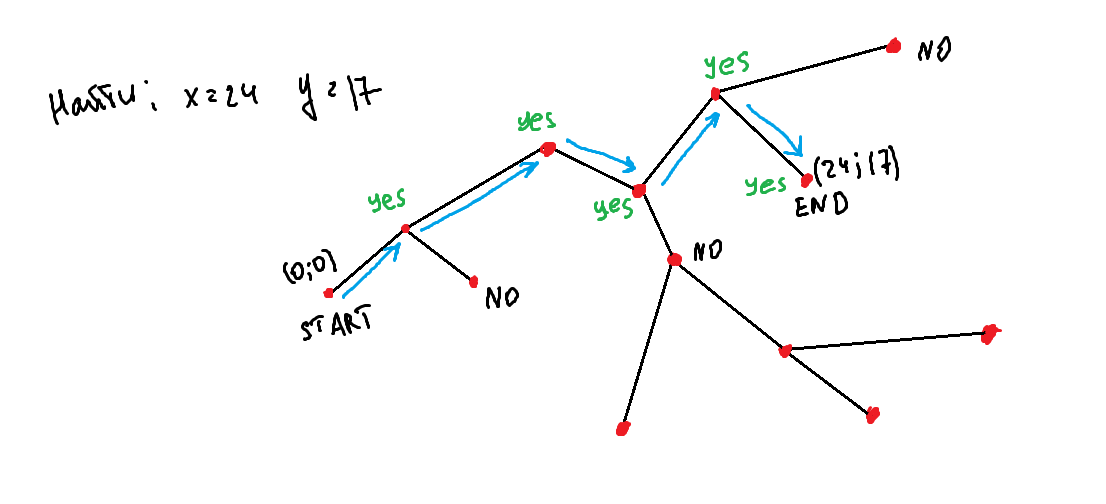


Рис 4. Визуальное представление поиска точек в древовидной структуре.

Удаление точки из структуры. Мы должны перекинуть ссылки с родительских и дочерних элементов друг на друга.

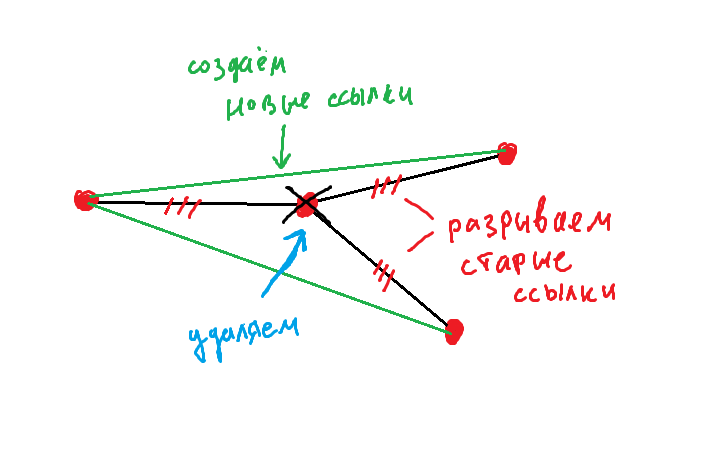
 

Рис 5 и 6. Алгоритм удаления точек из структуры, а также его визуальное представление.

Также стоит отметить то, как именно использовалась данная структура данных, именно на этапе разработки этой структуры.

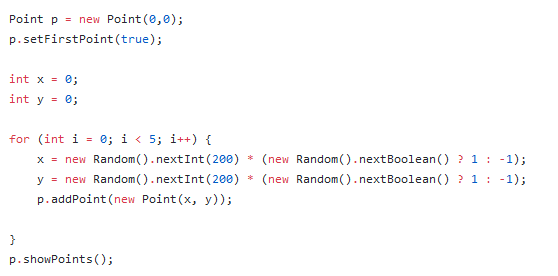


Рис 7. Использование древовидной структуры в программе.

Добавляли новые точки мы в переменную *p,* тем самым создавая целую древовидную структуру в дальнейшем. (Просто закидывали точку в точку).

И на тот момент предполагалось, что одна такая переменная типа *Point* – есть готовая фигура, т.к. она сразу хранит в себе структуру точек. Т.е. мы могли создать список из *Point*’ов, и использовать его как список фигур. Отсюда и вытекали все дальнейшие сложности работы с такой структурой. Правда появлялись они из-за того, что, на тот момент я не знал, как правильно это всё организовать.

По итогу я вынужден был отказаться от такой структуры и пойти по более простому пути. Но я планирую доработать эту структуру и избавиться от всех её недостатков, т.к. считаю, что у неё есть потенциал.

### Хранение точек и фигур в списках, но фигуры состоят из рёбер.

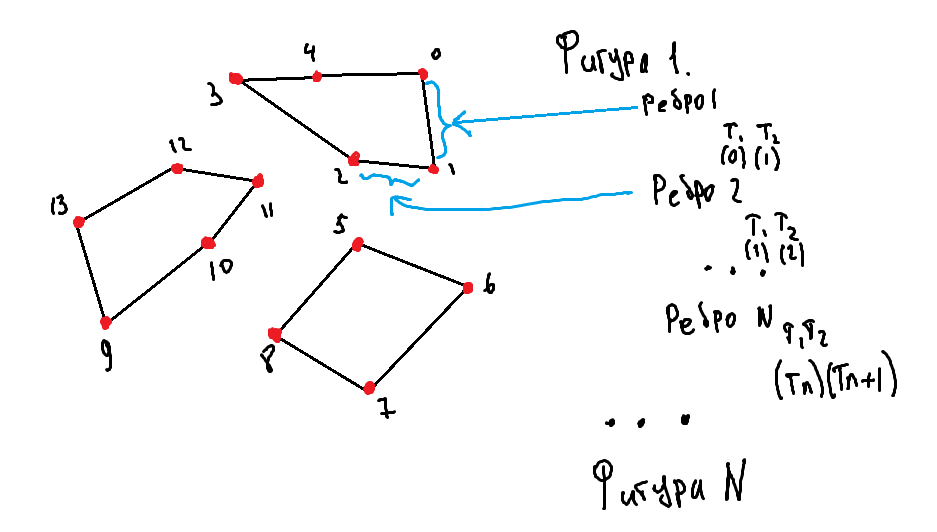


Рис 8. Визуальное представление структуры с рёбрами.

Суть данной структуры заключается в том, что все точки хранятся в отдельном списке, как и фигуры. Однако фигуры состоят из рёбер. Ребро тем самым ссылается на две точки, начало и конец ребра.

С такой структурой работать уже проще, чем с первой. Появляется больше контроля над всей структурой благодаря индексам, становятся проще алгоритмы. Но всё же нашлись свои весомые недостатки.

Плюсы данной структуры:

1. Более просыте алгоритмы поиска, удаления и редактирования точек и фигур в структуре.
2. Больше контроля над всей структурой благодаря индексам.
3. Т.к. фигуры хранятся отдельными объектами, появилось обширное поле для модернизации этих фигур (настройка их параметров, изменение цвета и т.д.).

Минусы:

1. По сравнению с первой структурой поиск точки представляет собой проход по каждой точке, из-за чего скорость поиска более медленная. Однако это спорный вопрос. Ибо если брать маленькое количество точек, то скорость будет мало чем отличаться и особой разницы тут не увидеть.

Но заметное преимущество первой структуры можно увидеть при использовании более миллиона точек (условно), где проход по каждой будет занимать достаточно много времени.

1. Из-за того что использовались рёбра, а в одном ребре хранились две ссылки на точки, то у нас бессмымленно удваивалась работа при редактирвоании фигур и точек.

Реализация данной структуры на Java.

К сожалению, исходного кода той структуры не осталось. Но от неё тоже пришлось отказаться, из-за её несовершенства и использования рёбер, что в какой-то мере бессмысленно удваивало работу.

### Хранение точек и фигур в списках, без использования рёбер.

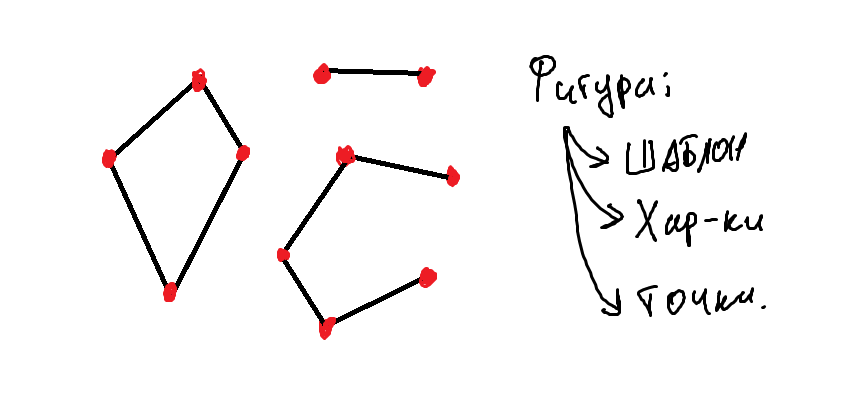


Рис 8. Визуальное представление структуры без рёбер.

Данная структура полностью наследовалась от предыдущей, и всё что в неё поменялось – использование индексов точек вместо рёбер, для построения фигуры. Именно этот шаг упростил дальнейшую работу, и позволил реализовать шаблоны фигур, выполнение каких-либо операций с ними и т.д.

От предыдущей структуры ей передались и плюсы, и минусы (из минусов только один, связанный с производительностью).

Описывать её реализацию в этом разделе смысла нет, потому что всё это ещё будет расписано далее.

## Разработка и тестирование программы

Разработка полноценного приложения началась после выполнения двух предыдущих пунктов, что вполне объяснимо. Сначала была подготовлена графическая часть приложения, затем разработана отдельно структура данных, и по итогу эти две системы слились воедино.

Сначала приложение имело скудный функционал, позволяющий создавать одну фигуру и редактировать её точки, затем, по мере продвижения наращивались новые «фичи». Например, разные режимы работы с фигурами, различные новые окна в приложении т.д.

Тестирование производилось на разных этапах и на разных уровнях.

Т.е. тестировалась как структура данных отдельно, во время разработки, так и графическая часть приложения. И со временем, оно всё переросло в состояние, когда тестирование этих обоих частей происходило одновременно.

Например, добавление точек на экран производится через касание (графическая часть), и через дальнейшее занесение этой точки в структуру данных (часть данных).

Единственное, что не вошло в этот план – построение архитектуры самого приложения. И я очень об этом жалею. Если бы изначально я изучил нюансы разработки этого приложения, продумал как внедрить структуру данных в него, то сейчас это приложение выглядело бы гораздо лучше, как внутри, так и снаружи. Это бесценный опыт для меня, когда я на живом примере увидел, почему важно планирование и проектирование.

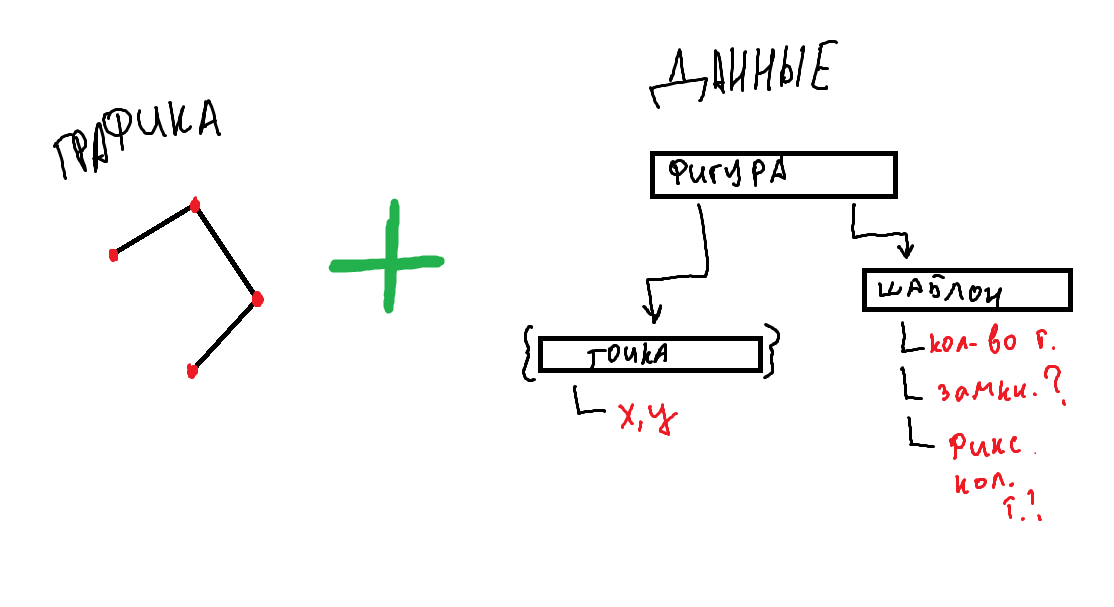


Рис 9. Визуальное представление смешения двух частей приложения.

# Общее описание проекта

## Структура модулей

Этот проект работает на ОС Android, поэтому в структуре классов встречается не мало классов, которые связаны с этой системой. Например, активности, фрагменты и т.д.

Поэтому, сначала я покажу все пакеты, которые содержатся в проекте, а затем разберу каждый из них, включая вложенные в них классы.

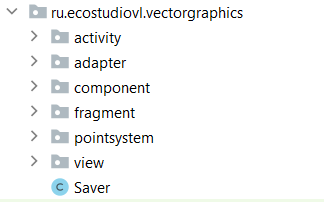


Рис 9. Структура пакетов программы.

### Пакет активностей

В системе Android, приложение состоит из окон, которые могут переключаться между собой, они называются активностями (Activity).

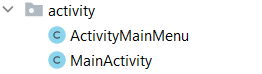


Рис 10. Структура пакета активностей

В данном пакете хранятся рабочий экран (MainActivity), и главное меню (ActivityMainMenu).

Рабочий экран представляет собой область, где мы строим изображения.

Главное меню отвечает за навигацию по приложению.

### Пакет адаптеров

Адаптеры (Adapter, в данном случае используются RecyclerView адаптеры), нужны для того, чтобы преобразовать данные (допустим из списка) в графическое представление, и вывести их на экран.

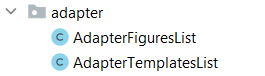


Рис 11. Структура пакета адаптеров

Конкретно в этом пакете хранятся адаптеры для списка фигур, и для списка шаблонов. Адаптеры так-же позволяют реализовать выбор данных, проводить с ними какие-либо операции (например, удаление или изменение данных).

### Пакет компонентов

В данном пакете хранятся глобальные компоненты программы, к которым можно обратиться из любого места программы, записать или взять из них информацию.

Обращение к ним из любого места программы возможно, благодаря тому что, эти классы глобальные, как и их данные, и все эти классы реализуют паттерн Singleton.

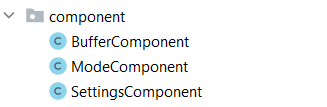


Рис 12. Структура пакета компонентов

BufferComponent – позволяет проводить операции связанные с выбором фигур. Его суть заключается в том, чтобы хранить индексы на выбранные пользователем фигуры, которые мы потом сможем прочитать в любой удобный момент.

ModeComponent – хранит в себе доступные и текущие режимы работы программы, режимы работы с фигурами и т.д.

SettingsComponent – обеспечивает доступ к глобальным настройкам приложения, для того чтобы мы смогли узнать интересующие нас параметры в любой момент работы программы.

### Пакет фрагментов

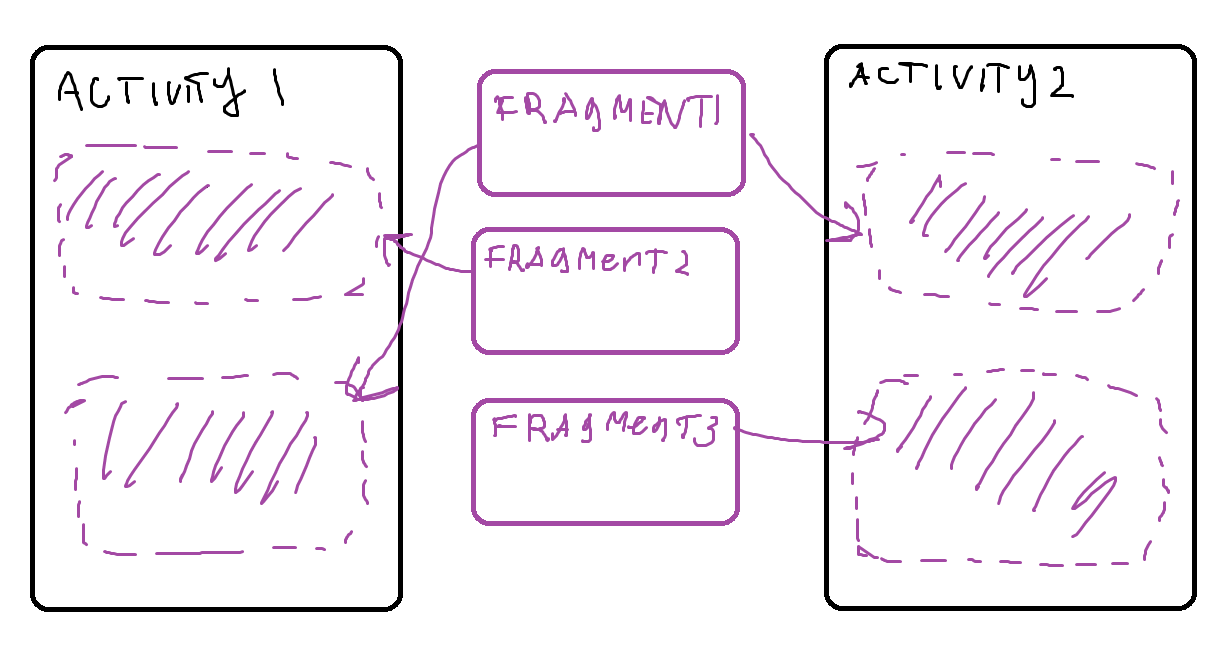


Рис 13. Визуализация фрагментов

Фрагменты (Fragment) в Android представляют собой панели, элементы экрана, которые уже встраиваются в активность. Т.е. активность это, целое окно программы, а фрагменты это, маленькие окошки которые встраиваются в активность. Преимущество фрагментов в том, что их можно много раз пере использовать, например, один и тот же фрагмент использовать в пяти активностях.

Также, основная суть фрагментов заключается в том, что мы большую часть логики разбиваем на эти самые фрагменты, и каждый из них выполняет свою роль. Мы избегаем строительства гигантской и запутанной системы благодаря им.

Данный пакет уже состоит из нескольких дочерних пакетов, с вложенными в них классами фрагментов.

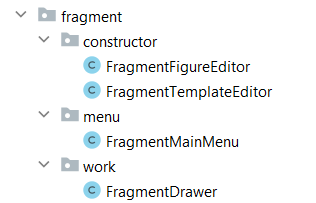


Рис 14. Структура пакета с фрагментами

Пакет ***constructor,*** содержит в себе фрагменты для создания фигур и шаблонов фигур. ***FragmentFigureEditor*** и ***FragmentTemplateEditor*** соответственно.

Пакет ***menu,*** содержит в себе фрагмент ***FragmentMainMenu***, который отвечает за отображение главного меню.

Пакет ***work,*** содержит в себе фрагмент ***FragmentDrawer***, являющийся той самой рабочей областью, где мы выполняем операции с фигурами. Отсюда и название пакета, от слова «рабочий».

### Пакет структуры данных

В этом пакете хранятся все классы и под-пакеты, которые относятся к структуре данных точек и фигур.

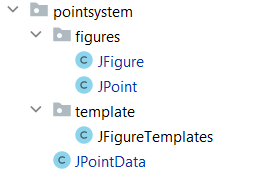


Рис 15. Структура пакета с данными о точках и фигурах

Пакет ***figures*** включает в себя класс ***JFigure*** (Фигура) и ***JPoint*** (Точка). Собственно, это два самых важных классах, на которых строится вся структура.

Пакет ***template*** включает в себя класс ***JFigureTemplates*** (Шаблон фигуры). В этом классе хранятся первоначальные данные о фигуре (замкнутая ли фигура, кол-во точек и т.д.).

Далее идёт просто класс ***JPointData***, в котором уже хранятся все данные о точках, фигурах и шаблонах. Также он содержит в себе основные операции с фигурами и точками. Т.е. это уже класс всей структуры данных.

### Пакет визуальных компонентов приложения

В этом пакете хранится класс элемента, который выводит графику на экран, и класс потока, который эту графику обрабатывает (Работает с Canvas).

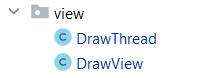


Рис 15. Структура пакета обработчиков графики

### Класс Saver

Данный класс у нас лежит просто в главном пакете, и в будущем пригодится для реализации работы с файлами.

## Типы данных и имена переменных

В этом проекте используется очень много переменных, и зачастую они выполняют не самую передовую роль в выполнении задач, например, переменные для кнопок, интерфейсы обратной связи и т.д. Поэтому тут я затрону самые важные и основные модули и переменные в программе.

Также хочется отметить, что я не буду показывать геттеры и сеттеры к членам класса, т.к. считаю, что это не заслуживает особого внимания и они просто предоставляют данные к скрытым членам класса.

### Классы структуры данных

Я думаю, что начать можно с этих модулей, т.к. они самые основные.

Класс ***JPoint*** (точка) – самый основной и примитивный класс, который просто хранит в себе две переменные, это позиция ***x*** и ***y***. Больше в нём ничего не предусмотрено, потому что по сути от него большее и не требуется.

private float x; *//Позиция точки по X*private float y; *//Позиция точки по Y  
  
//Конструктор класса*public JPoint*(*float x, float y*){* this.x = x;  
 this.y = y;  
*}*

Класс ***JFigure*** (фигура) – класс на основе которого строятся сами фигуры. Он включает в себя название, крайние точки фигуры, центр, точки и т.д.

private String name; *//Название фигуры*private int templateIndex; *//Индекс используемого шаблона*private float centerX; *//Центр фигуры по X*private float centerY; *//Центр фигуры по Y*private float scale = 1; *//Размер фигуры*private float rotate = 0; *//Поворот фигуры*private float minX, maxX; *//Крайние точки по X*private float minY, maxY; *//Крайние точки по Y*protected List*<*Integer*>* points; *//Индексы точек, которая хранит в себе фигура  
  
//Конструктор фигуры*public JFigure*(*String name, int templateIndex*){* this.points = new LinkedList*<>()*;  
 this.name = name;  
 this.centerX = 0;  
 this.centerY = 0;  
 this.templateIndex = templateIndex;  
*}*

Класс ***JFigureTemplates*** (шаблон фигуры) – данный класс содержит в себе информацию о шаблоне фигуры. А именно, какое кол-во точек в фигуре, фиксированное ли кол-во точек, замкнутая фигура или нет, и название шаблона. Основное преимущество этого класса в том, что мы можем в любой момент изменить его, не затрагивая остальные части программы.

private int pointsCount; *//Количество точек в фигуре*private boolean isClosePointNumber; *//Ограниченное ли кол-во точек*private boolean isClosedFigure; *//амкнутая фигура или нет*private String name; *//Название шаблона  
  
//онструктор класса*public JFigureTemplates*(*int pointsCount, boolean isClosePointNumber, boolean isClosedFigure, String name*){* this.pointsCount = pointsCount;  
 this.isClosePointNumber = isClosePointNumber;  
 this.isClosedFigure = isClosedFigure;  
 this.name = name;  
  
*}*

Класс ***JPointData*** (основная структура данных) – данный класс содержит в себе все точки, фигуры, и шаблоны, используемые в программе. По части данных от него больше ничего не требуется, всю остальную часть класса на себя берут различные процедуры.

private List*<*JFigure*>* figures; *//Все фигуры созданные пользователем*private List*<*JFigureTemplates*>* templates; *//Все шаблоны в программе*private List*<*JPoint*>* points;*/\*Список абсолютно всех точек, которые есть на экране и которые задействуются  
 в структуре данных, описывающих фигуры.\*/*private static JPointData *instance*; */\*Т.к. этот класс реализует паттерн  
 Singleton, внутри храним объект самого класса\*/  
  
/\*  
Вспомогательная структура данных, которая используется в процедурах  
и методах этого класса.  
Она собержит в себе 4 направления: ВВЕРХ, ВНИЗ, ВЛЕВО, ВПРАВО  
 \*/*private enum PointDirection*{  
 UP*,  
 *DOWN*,  
 *LEFT*,  
 *RIGHT  
}  
  
  
//Конструктор класса*private JPointData*(){* figures = new ArrayList*<>()*;  
 templates = new ArrayList*<>()*;  
 *//Тут мы добавляем системные шаблоны в программу* templates.add*(*new JFigureTemplates*(*0, false, false, "Кривая"*))*;  
 templates.add*(*new JFigureTemplates*(*0, false, true, "Замкнутая фигура"*))*;  
 templates.add*(*new JFigureTemplates*(*3, true, true, "Треугольник"*))*;  
 templates.add*(*new JFigureTemplates*(*4, true, true, "Четырёхугольник"*))*;  
 templates.add*(*new JFigureTemplates*(*5, true, true, "Пятиугольник"*))*;  
 templates.add*(*new JFigureTemplates*(*6, true, true, "Шестиугольник"*))*;  
 templates.add*(*new JFigureTemplates*(*3, true, false, "Угол"*))*;  
 points = new LinkedList*<>()*;  
*}  
  
//Реализация паттерна Singleton*public static JPointData getInstance*(){* if *(instance* == null*) {  
 instance* = new JPointData*()*;  
 *}* return *instance*;  
*}*

### Классы компонентов

Класс ***BufferComponent*** (компонент буффера) – данный класс реализует паттерн Singleton, и содержит в себе просто «мапу» (TreeMap) как буфер, в которой содержатся все выбранные нами фигуры. Суть класса в том, чтобы запомнить выбранные фигуры, и чтобы можно было получить эти данные из любого места программы (засчёт Singleton).

private TreeMap*<*Integer, Integer*>* selectedMap;*//Мапа в которой хранятся все выбранные фигуры*private static BufferComponent *instance*; *//Объект этого класса, нуэжен для реализации паттерна Singleton  
  
//Конструктор класса*private BufferComponent*(){* selectedMap = new TreeMap*<>()*;  
*}*public static BufferComponent getInstance*(){* if *(instance* == null*){  
 instance* = new BufferComponent*()*;  
 *}* return *instance*;  
*}*

Класс ***ModeComponent*** (компонент доступных и текущих режимов работы программы) – данный класс реализует паттерн Singleton, и содержит в себе маленькие enum структуры, которые определяют, какие в программе есть режимы работы, режимы выбора фигур. Этот класс сделан отдельно и с ипользованием Singleton, для того чтобы можно было обратиться к нему из любого места программы.

*//Режим работы программы*public enum Mode *{  
 CREATE*, *//Создание фигуры   
 EDIT*, *//Редактирование фигуры  
 DELETE*, *//Удаление точек  
 VIEW*, *//Просмотр фигур (холста)  
 COPY*, *//Копирование фигуры  
 MOVE //Перемещение фигуры  
}  
  
//Режим выбора фигуры*public enum SelectionMode*{  
 ONE*, *//Одиночный выбор  
 MORE //ножественный выбор  
}*

*//Режим создания фигуры  
//Пока не используется*public enum CreationMode*{  
 ADDITIVE*, *//Аддитивный, на основе уже созданных точек  
 CREATIVE //Креативный, с созданием новых точек   
}*public Mode currentMode; *//Текущий режим работы программы*public SelectionMode selectionMode; *//Текущий режим выбора фигуры*public CreationMode creationMode; *//Текущий режим создания фигуры (не используется)*private static ModeComponent *instance*;  
  
*//Конструктор класса*private ModeComponent*(){* currentMode = Mode.*VIEW*;  
*}*public static ModeComponent getInstance*(){* if *(instance* == null*){  
 instance* = new ModeComponent*()*;  
 *}* return *instance*;  
*}*

Класс ***SettingsComponent*** (компонент настроек) – данный класс реализует паттерн Singleton, и содержит в себе набор переменных, которые тем или иным образом влияют на работу программы. Например, отображать точки у фигуры или нет. Но в текущей версии программы этот класс ещё не нашёл себе применения, но всё ещё в будущем.

private static SettingsComponent *instance*; *//Инстанс класса*private boolean showPoints; *//Отображать точки фигур или нет*private SettingsComponent*(){* showPoints = true;  
*}*public static SettingsComponent getInstance*(){* if *(instance* == null*){  
 instance* = new SettingsComponent*()*;  
 *}* return *instance*;  
*}*

### Классы активностей

Класс ***ActivityMainMenu*** (активность главного меню) – данный класс является активностью Android программы, и его зада состоит в том, чтобы просто отобразить фрагмент с главным меню. В нём не хранится переменных, и поэтому тут нечего показывать.

Класс ***MainActivity*** (главная рабочая активность) – данный класс является активностью Android программы, и он же есть главная рабочая активность. Его задача состоит в том, чтобы переключать фрагменты экрана, во время работы программы, через колбэки (callback).

public class MainActivity extends AppCompatActivity implements FragmentTemplateEditor.FragmentTemplateEditorCallback, FragmentDrawer.FragmentDrawerCallback, FragmentFigureEditor.FragmentFigureEditorCallback *{* private NavController navController; *//Навигационный контроллер, для переключения экранов*}

### Классы фрагментов

Класс ***FragmentFigureEditor*** (фрагмент для создания фигуры) – данный класс является фрагментом Android программы. В нём мы создаём фигуры, указываем их название и шаблон.

private View view; *//Корневой View элемент фрагмента*private RecyclerView rvTemplates; *//Графичесакий элемент списка шаблонов*private EditText etFigureName; *//Поле ввода названия фигуры*private int selectedTemplate = -1; *//Индекс выбранного шаблона фигуры  
  
//Интерфейс для колбэка в активность*public interface FragmentFigureEditorCallback*{* void onCreatedFigure*()*; *//Говорим активности что мы создали фигуру* void onBackPressedFigure*()*; *//Говорим активности, что мы не хотим создавать фигуру  
}  
  
//Объект коллбэка, чтобы вызывать его процедуры*private FragmentFigureEditorCallback fragmentFigureEditorCallback;  
  
public FragmentFigureEditor*() {  
 // Required empty public constructor  
}*

Класс ***FragmentTemplateEditor*** (фрагмент для создания шаблонов) – данный класс является фрагментом Android программы. В нём мы создаём шаблон, указываем его характеристики.

private View view;*//Корневой view элемент фрагмента*private RecyclerView rvTemplates; *//Элемент отображающий все имеющиеся шаблоны*private EditText etTemplateName; *//Поле ввода имени нового шаблона*private EditText etPointsNumber; *//Поле ввода количества точек шаблона*private CheckBox cbIsClosedFigure; *//Чек-бокс, закрытая ли фигура*private CheckBox cbIsClosedFigureNumber;*//Чек-бокс, закрытое ли кол-во точек*private int selectedIndex = 0; *//Индекс выбранного шаблона для удаления  
  
//Интерфейс коллбэка для активности*public interface FragmentTemplateEditorCallback*{* void onBackPressedTemplate*()*;  
*}  
  
//Объект интерфейса коллбэка чтобы к нему обратиться*private FragmentTemplateEditorCallback fragmentTemplateEditorCallback;

Класс ***FragmentMainMenu*** (фрагмент главного меню) – данный класс является фрагментом Android программы. В нём мы просто отображаем кнопку начала работы, и передаём коллбэк активности.

private View view; *//Корневой элемент фрагмента  
  
//Объект для колбэка, чтобы к нему обращаться*private FragmentMainMenuCallback fragmentMainMenuCallback;  
  
*//нтерфейс коллбэка для активности*public interface FragmentMainMenuCallback*{* void onStartWork*()*;  
*}*

Класс ***FragmentDrawer*** (фрагмент рисования) – данный класс является фрагментом Android программы. Он принимает на себя всю работу по рисованию и работе с фигурами.

private View view; *//Корневой view элемент фрагмента*private ImageButton btnHide; *//Кнопка скрытия списка фигур*private ImageButton btnSelectionMode; *//Кнопка режима выбора фигуры*private ImageButton btnDeleteSelection; *//Кнопка удаления выбраных фигур*private ImageButton btnCopy; *//Кнопка копирования фигуры*private ImageButton btnMove; *//Кнопка перемещения фигуры*private CardView lnList; *//Обхект на экране со списком фигур*private RecyclerView rvFigures; *//Список фигур*private DrawView drawView; *//Элемент отображающий сами фигуры на экране (линии и точки)*private ImageButton btnChangeMode; *//Кнопка смены режима рисования*private TextView tvInfoLabel; *//Информационное сообщение о текущих действияъ*private LinearLayout lnProperties; *//Не используется*private SeekBar sbScale, sbRotate; *//Не используются  
  
//Интерфейс для коллбэка в активность*public interface FragmentDrawerCallback*{* void onCreateFigureClicked*()*;  
 void onCreateTemplateClicked*()*;  
 void onMainMenuClicked*()*;  
*}  
  
//Объект интерфейса, для обращения к нему*private FragmentDrawerCallback fragmentDrawerCallback;  
  
public FragmentDrawer*() {  
 // Required empty public constructor  
}*

### Классы адаптеров

Класс ***AdapterFiguresList*** (адаптер фигур) – данный класс является адаптером для отображения фигур в Android программе. В нём хранится локальный список фигур, которые потом отображаются в графическом виде. Также реализован колбэк выбора фигуры.

private List*<*JFigure*>* mData; *//Локальный список фигур*private FigureSelect figureSelect; *//Объект интерфейса колбэка*private Context context; *//Контекст в котором работает данный список  
  
//Конструктор класса*public AdapterFiguresList*(*List*<*JFigure*>* figures, FigureSelect figureSelect, Context context*){* mData = figures;  
 this.figureSelect = figureSelect;  
 this.context = context;  
*}  
  
//Интерфейс колбэка*public interface FigureSelect*{* void onSelectFigure*(*int index*)*;  
*}*

Класс ***AdapterTemplatesList*** (адаптер шаблонов) – данный класс является адаптером для отображения шаблонов в Android программе. В нём хранится локальный список шаблонов, которые потом отображаются в графическом виде. Также реализован колбэк выбора шаблона.

private List*<*JFigureTemplates*>* mData; *//Локальный список шаблонов для отображения*private TemplateSelect figureSelect; *//Объект интерфейса колбэка*private Context context; *//Контекст в котором работает данный адаптер*private int selectedTemplate; *//Выбранный шаблон, для посветки  
  
//Конструктор адаптера*public AdapterTemplatesList*(*TemplateSelect figureSelect, Context context, int selectedTemplate*){* mData = JPointData.*getInstance()*.getTemplates*()*;  
 this.figureSelect = figureSelect;  
 this.context = context;  
 this.selectedTemplate = selectedTemplate;  
*}  
  
//Интерфейс колбэка выбранной фигуры*public interface TemplateSelect*{* void onSelectTemplate*(*int index*)*;  
*}*

### Классы view-элементов

Класс ***DrawView*** (элемент для отображения графики) – данный класс является view элементом в Android программе. Этот элемент помещается на экран, и способен выводить «картинку». За обработку самой графики отвечает другой поток.

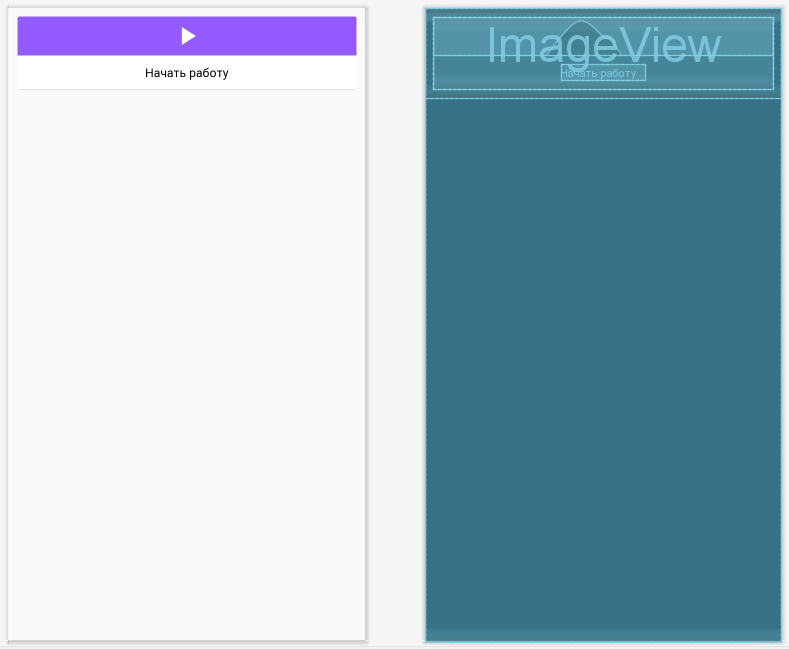
private int touchedPoint; *//Индекс точки, на которую мы тыкнули пальцем.*private boolean isPointTouched; *//Переменная нужна при перемещении точки, чтобы не цеплять точки которые очень рядом.*private TreeMap*<*Float, List*<*JPoint*>>* pairsMap; */\*Мапа, которая используется для поиска пар точек  
Ключ - расстояние между точками  
Значение - список с самими точками\*/  
  
  
  
/\*  
TODO: реализовать нормальное перемещение по холсту, и масштабирование  
Вспомогательные перменные которые на данный момент не используются.  
Нужны для перемещения по холсту, и для масштабирования изображения.  
 \*/*private int scaleMultiplier = 1;  
private float xDelta;  
private float yDelta;  
  
private DrawThread drawThread;*//Главный поток, отвечающий за постоянную отрисовку фигур и точек (рендеринг).  
  
  
  
//Интерфейс коллбэка, который нужен для обращения к фраменту*public interface DrawViewCallback*{* void onSelectFigure*(*int index*)*;  
 void onCopyFigure*()*;  
*}  
  
//Объект колбэка*private DrawViewCallback drawViewCallback;  
  
public DrawView*(*Context context, DrawViewCallback drawViewCallback*) {* super*(*context*)*;  
 this.drawViewCallback = drawViewCallback;  
 getHolder*()*.addCallback*(*this*)*;  
 initializeVariables*()*;  
*}*

Класс ***DrawThread*** - поток который обрабатывает графику, работает с canvas, и результат нарисованного выводит на экран.

private boolean running = false; *//Флаг который показывает, работает поток или нет*private SurfaceHolder surfaceHolder; *//Холдер для возврата результата рисования на экран*private Paint p; *//Кисть, которая выполняет рисование  
  
//Конструктор класса*public DrawThread*(*SurfaceHolder surfaceHolder*){* this.surfaceHolder = surfaceHolder;  
 this.p = new Paint*()*;  
*}*

## Структура меню проекта

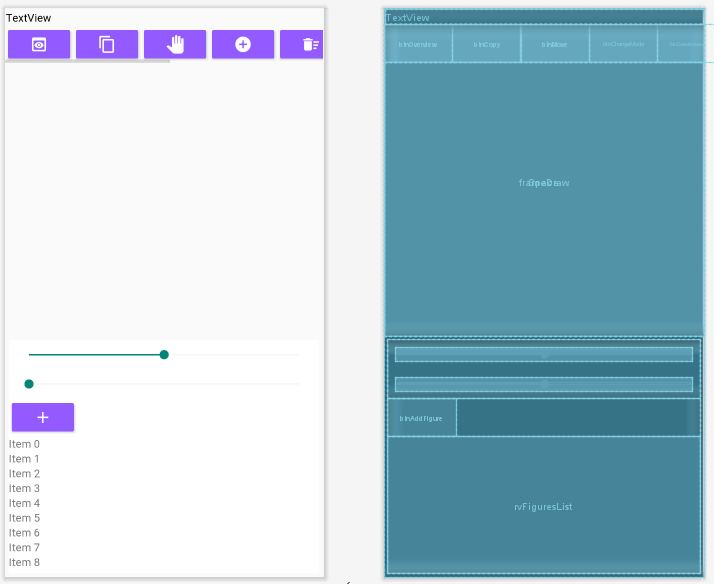
### Главное меню



На этом экране не происходит ничего особого, мы просто отображаем кнопку начала работы.

Рис 16. Экран главного меню

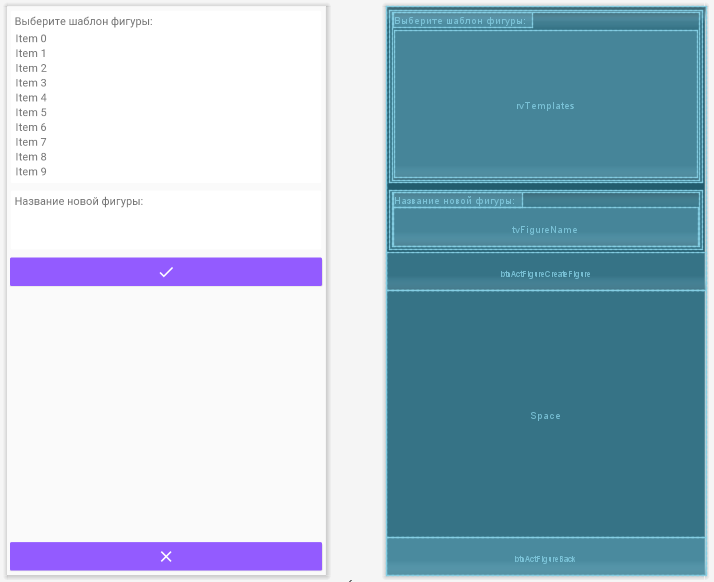
### Рабочая область



На этом экране происходит сама работа по построению и редактированию фигур, выбираются различные режимы работы программы.

Рис 17. Экран рабочей области

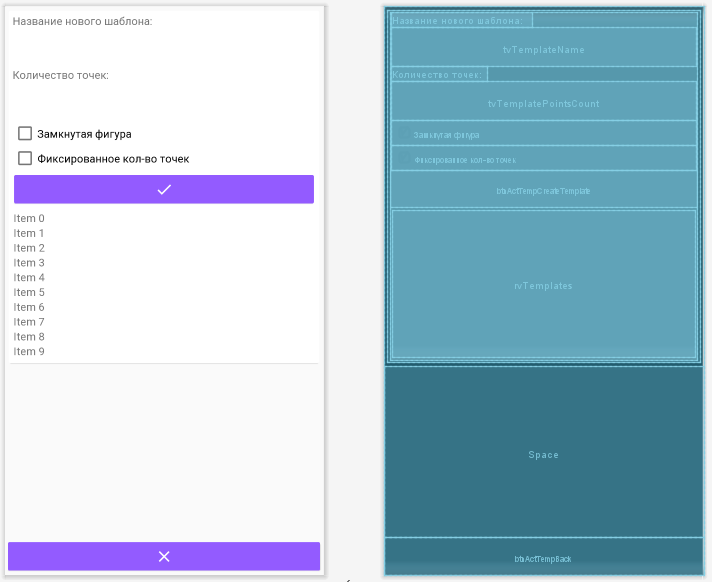
### Экран создания фигуры



На этом экране мы выбираем шаблон фигуры и вводим её название. Присутствуют кнопки подтверждения и отмены действий.

Рис 18. Экран создания фигуры

### Экран создания шаблона



На этом экране мы создаём шаблоны, или удаляем их. При создании шаблона указываем его свойства.

Рис 19. Экран создания шаблона

## Описания основных процедур

В данном модуле отчета я приведу только самые основные процедуры, которые отвечают за саму структуру данных и фигуры, ибо не вижу смысла занимать много места процедурами вывода информации на экран и т.д. То же самое касается стандартных методов и процедур во фрагментах, активностях и адаптерах, они не несут особой смысловой нагрузки.

### Процедуры и методы структуры данных

Класс JFigure

*/\*  
Процедура добавляющая точку в фигуру  
Сначала мы проверяем по шаблону, можем ли мы её добавить  
Затем добавляем её в список  
Добавляется именно индекс самой точки из общего списка точек  
На вход подаётся индекс точки, и шаблон на который мы ссылаемся  
 \*/*public void addPoint*(*int index, JFigureTemplates template*){…}*

*/\*  
Процедура просчитывает центр масс фигуры на основе всех имеющихся точек   
у этой фигуры  
Ищется самая центральная точка, относительно самых крайних точек фигуры  
 \*/*public void recalculateCenter*(){…}*

*/\*  
Процедура удаляет точку из списка точек в фигуре  
На вход подаётся индекс точки из обшего списка точек  
 \*/*public void deletePoint*(*int index*){…}*

*/\*  
Процедура возвращает индекс точки в локальном списке фигуры  
На основе индекса из общего списка  
 \*/*public int getLocalIndexOfPoint*(*int index*){…}*

*/\*  
Процедура возвращающая булевое значение  
Говорящее о том, хранится точка в фигуре или нет  
На вход подаётся индекс из общего списка точек  
 \*/*public boolean isContainsPoint*(*int index*){…}*

Класс JPointData

*/\*  
Процедура копирует указанную фигуру в указанные координаты  
На вход подаётся индекс фигуры, и позиция куда её копировать  
 \*/*public void copyFigureToPosition*(*int figure, float posX, float posY*){…}*

*/\*  
Процедура перемещает указанную фигуру в указанные координаты  
На вход подаётся индекс фигуры, и позиция куда её переместить  
 \*/*public void moveFigureToPosition*(*int figure, float posX, float posY*){…}*

*/\*  
Процедура поворачивает указанную фигуру по часовой стрелке на указаное кол-во градусов  
 \*/*public void rotateFigurePlus*(*int figure, float angle*){…}*

*/\*  
Процедура поворачивает указанную фигуру против часовой стрелки на указаное кол-во градусов  
 \*/*public void rotateFigureMinus*(*int figure, float angle*){…}*

*/\*  
Находит положение координаты X одной точки, относительно другой  
первый входной параметр - точка относительно которой находим  
второй входной параметр - точка положение которой нужно найти  
 \*/*public PointDirection findHorizontalDirection*(*float sX, float eX*){…}  
  
/\*  
Находит положение координаты Y одной точки, относительно другой  
первый входной параметр - точка относительно которой находим  
второй входной параметр - точка положение которой нужно найти  
 \*/*public PointDirection findVerticalDirection*(*float sY, float eY*){*…*}*

*/\*  
Процедура находит длину между двумя точками в одной плоскости  
 \*/*private float getLengthByOneAxis*(*float startX, float endX*){*…*}*

*/\*  
Процедура находит расстояние между двумя точками в двумерном пространстве  
 \*/*private float getLength*(*float startX, float startY, float endX, float endY)*{*…*}*

### Процедуры классов, отвечающих за графику

Класс DrawThread

*/\*  
Процедура отрисовывает линии и точки фигур  
На вход передаётся холст для рисования  
 \*/*private void render*(*Canvas canvas*) {…}*

*/\*  
Процедура рабоыт потока отрисовки графика  
Вызывает процедуру render, возвращает холст на экран  
 \*/*@Override  
public void run*() {…}*

Класс DrawView

*/\*  
Процедура останавливает поток рисования  
 \*/*public void stopDrawThread*() {*…*}  
  
/\*  
Процедура запускает поток рисования  
 \*/*public void startDrawThread*() {…}  
  
  
/\*  
Эвент реагирующий на нажатия по экрану  
В нём, в зависимости от того какой режим работы программы  
происходят нужные действия с фигурами  
 \*/*@Override  
public boolean onTouchEvent*(*MotionEvent event*) {…}*

*/\*  
Метод возвращает индекс точки, на которую мы нажали  
На вход подаются координаты точки нажатия на экран  
 \*/*private int getTouchedPoint*(*float centerX, float centerY*) {…}*

*/\*  
Процедура находит фигуру, на которую мы нажали  
На вход передаётся координаты x и y, где было произведено нажатие  
1. Находим все пары точек, в зоне которых находятся координаты  
2. Находим пару точек, расстояние между которыми минимальное  
3. Если такая пара найдена, находим угол поворота между первой точкой, и координатами  
4. Далее, находим угол поворота между точками в паре  
5. Если угол координат, совпадает с углом второй точки (с учётом погрешности),  
 то мы нашли нужное нам пересечение с линией  
6. Находим фигуру, благодаря процедуре которая находит фигуру по точке (используем первую точку в паре)  
 \*/*private void findFigureByLine*(*float x, float y*){…}*

*/\*  
Данная процедура минимизирует индексы точек которые ссылаются  
Для чего это нужно?  
Когда мы удаляем точку из общего списка, все точки после неё автоматически скатываются по индексу вниз  
Поэтому, чтобы не было несостыковок в структуре данных, мы в каждой фигуре,  
где есть индексы точек большие чем удалённая точка - понижаем эти индексы  
 \*/*private void minimize*(*int index*){*…}

*/\*  
Процедура удаляет фигуру, а так-же все точки которые ей принадлежат  
 \*/*public void deletePointsWithFigure*(*int figureIndex*){…}*

*//Процедура находит длину отрезка между двумя точками*private float getLength*(*float startX, float startY, float endX, float endY*){…}*

*//Процедура находит угол поворота одной точки, относительно другой*private float getDegrees*(*float aX, float aY, float bX, float bY*){*…*}*

*/\*  
Метод находит самую ближайшую точку к переданным координатам  
 \*/*private int findNearPoint*(*float startX, float startY*){…}*

*/\*  
Процедура находит все пары точек, в зоне области которых находятся переданные координаты  
 \*/*private void findCollisedPairs*(*float x, float y*){…}*

*/\*  
Процедура находит пару точек, расстояние между которыми минимальное  
 \*/*private List*<*JPoint*>* findMinimalPair*(){…}*

*/\*  
Процедура проверяет нахождение точки в прямоугольной области  
Учитывается то, в каком направлении идёт фигура  
Вверх, вниз, влево, вправо  
 \*/*private boolean isPointInZone*(*float x, float y, JPoint startPoint, JPoint endPoint*){…}*

*/\*  
Процедура находит фигуру, которой принадлежит индекс переданной точки  
 \*/*private int findFigureByPoint*(*int point*){…}*

### Процедуры классов фрагментов

*/\*  
Процедура инициализирует все view-элементы этого фрагмента  
 \*/*private void initializeViewElements*(){…}*

*/\*  
Процедура, которая меняет режим рабоыт программы, на основе текущего режима  
 \*/*private void onChangedMode*(){…}*

*/\*  
Процедура которая меняет текущий режим работы программы  
и подгоняет интерфейс под него  
НА вход подаётся новый режим работы программы  
 \*/*private void switchMode*(*ModeComponent.Mode mode*){…}*

*/\*  
Процедура смены режима выбора фигур  
На вход подаётся новый режим выбора фигуры  
 \*/*private void switchSelectionMode*(*ModeComponent.SelectionMode mode*){…}*

*/\*  
Процедура обновляет список фигур на экране  
 \*/*public void updateList*(){*…}

*/\*  
Процедура колбэка адаптера списка фигур  
Срабатывает при выборе фигуры в списке  
НА вход подаётся индекс выбранной фигуры  
 \*/*@Override  
public void onSelectFigure*(*int index*) {…}*

*/\*  
Колбэк от графического элемента, гвоорящий что фигуры была скопирвоана  
 \*/*@Override  
public void onCopyFigure*() {*…}

### Процедуры компонентов

Класс BufferComponent

*/\*  
Метод возвращает булевое значение, которое даёт знать  
есть ли в этом компоненте выбранные фигуры  
 \*/*public boolean hasSelectedFigures*(){…}*

*/\*  
Процедура добавляет индекс фигуры в мапу  
На вход подаётся индекс фигуры  
 \*/*public void addFigure*(*int index*){*…}

*/\*  
Процедура удаляет индекс фигуры из мапы  
 \*/*public void deleteFigure*(*int index*){*…}

*/\*  
Процедура очищает мапу выбранных фигур  
 \*/*public void clearSelected*(){*…}

*/\*  
Метод возвращает одну единственную выбранную фигуру в мапе  
Возвращается именно индекс  
 \*/*public int getCurrentSelectedObject*(){…}*

*/\*  
Процедура проверяет, содержит ли мапа индекс фигуры  
 \*/*public boolean isContainsFigure*(*int index*){*…}

*/\*  
Процедура минимизирует индексы всех выбранных фигур  
относительно переданной переменной  
  
Если текущий индекс фигуры больше чем переданная переменная - понижаем её индекс на 1  
 \*/*public void minimize*(*int index*){…}*

## Примеры графических построений в приложении

В данном разделе я добавлю лишь один скриншот работы программы, т.к. на нём видно основные типы фигур (замкнутые, не замкнутые, с ограниченным кол-вом точек и с неограниченным кол-вом точек). Также, можно увидеть, что был скопирован треугольник.



Рис 20. Пример графического построения фигур в программе

# Требования к оформлению кода и выбору имен

Очень важный раздел. Здесь должны быть приведены основные требования к оформлению кода, способам формирования имен и всё достаточно подробно. Ищите то, что я давал в лекциях и что написано в методических указаниях, ищите информацию в Интернет, чем больше вы напишете, тем лучше. Напишите какие префиксы в именах вы используете, укажите в какой нотации записываются ваши имена в проекте. Также можете объяснить, почему соглашения об оформлении кода и выборе имен так важны.

# Алгоритмы

Здесь размещаются словесные описания алгоритмов для каждого из ваших индивидуальных заданий. Как показывает опыт, написание алгоритмов для вас немалая проблема, поэтому, тренируйтесь, пробуйте, пишите. Обычно пытаются сразу начинать кодирование, а в голове нет даже и понимания, как надо решать задачу. Это важный блок вашего отчета.

# Тексты модулей

Здесь вы должны разместить тексты всех модулей проекта. Предлагается вставлять в отчет тексты модулей в виде копий экрана в приложении Lazarus. Т.е., в редакторе кода выбираете модуль, пролистываете до нужного мета, копируете активное окно при помощи Alt-PrtScr или другим способом (в Win 10, например, есть функция копии в буфер произвольной области экрана) и вставляете копии в отчет так, чтобы не было пропусков в тексте программы. Подсвеченные Лазарусом тексты удобнее читать, да и копии экранов не сканируются Антиплагиатом.

# Инструкция к приложению

Напишите инструкцию, позволяющую освоить разработанный вами векторный редактор.

# Заключение

Сформулируйте, что вы сделали в ходе исполнения проекта. Что разработано, кратко функционал и, по-вашему, какие достоинства имеет этот проект.

# Список литературы

В списке литературы обязательно указываются нижеперечисленные источники. В продолжение списка добавляете ту литературу и те ссылки в Интернете, которые вы использовали в своей работе в дополнение к основным.

1. Хованец В.А. Технология программирования и абстрактные типы данных// Методические указания – Владивосток: Изд-во МГУ им. адм. Г.И. Невельского, 2011. – 36 с.
2. Хованец В.А. Событийно-управляемое программирование// Методические указания – Владивосток: Изд-во МГУ им. адм. Г.И. Невельского, 2011. – 26 с.
3. Хованец В.А. Курс лекций «Программирование» [Текст]: учеб-метод, пособие / В.А. Хованец. – Владивосток: Изд-во МГУ им. адм. Г.И. Невельского, 2014. – 90 с. (электронный вариант).
4. Хованец В.А. Курс «Программирование» в системе электронного обучения МГУ им. адм. Г.И. Невельского, 2 курс ФЭИТ ИВТ, осенний семестр 2019-2020, <http://course.msun.ru/course/view.php?id=10179>.
5. Хованец В.А. Курс «Программирование» в системе электронного обучения МГУ им. адм. Г.И. Невельского, 2 курс ФЭИТ ИВТ, весенний семестр 2019-2020, <http://course.msun.ru/course/view.php?id=10178>.
6. Форум по курсовому проекту «Векторный редактор», <http://course.msun.ru/mod/forum/view.php?id=52933>
7. https://metanit.com/java/tutorial/5.8.php