Муниципальное бюджетное общеобразовательное учреждение

«Гимназия № 2» города Сарова

***Изучение принципов построения графических приложений с помощью библиотеки JavaFX на примере реализации разных математических задач.***

**Выполнил**: ученик 10 класса «Б»

МБОУ Гимназии №2

Аристов Сергей Антонович

**Руководитель**: Гудкова Евгения Александровна

учитель информатики

МБОУ Гимназии №2

**Оглавление**

[​ Введение 3](#__RefHeading___Toc263_2223766980)

[​ Теоретическая часть 4](#__RefHeading___Toc241_1050172711)

[​ Отбор технологий 5](#__RefHeading___Toc265_2223766980)

[​ AWT 5](#__RefHeading___Toc267_2223766980)

[​ Swing 5](#__RefHeading___Toc269_2223766980)

[​ JavaFX 7](#__RefHeading___Toc271_2223766980)

[​ Архитектура 7](#__RefHeading___Toc442_319222050)

[​ Архитектурное изучение 9](#__RefHeading___Toc273_2223766980)

[​ SOLID-принципы 9](#__RefHeading___Toc275_2223766980)

[​ MVC-паттерны 13](#__RefHeading___Toc277_2223766980)

[​ Обзор 13](#__RefHeading___Toc444_319222050)

[​ «Классическая» архитектура 13](#__RefHeading___Toc446_319222050)

[​ MVVM — архитектура 14](#__RefHeading___Toc448_319222050)

[​ Выбор задачи 15](#__RefHeading___Toc281_2223766980)

[​ Клеточные автоматы и игра «Жизнь» 15](#__RefHeading___Toc283_2223766980)

[​ Глава 2. Практическая часть 16](#__RefHeading___Toc243_1050172711)

[​ Система сборки 17](#__RefHeading___Toc531_969981342)

[​ Структура проекта 17](#__RefHeading___Toc533_969981342)

[​ Заключение 18](#__RefHeading___Toc535_969981342)

# Введение

Программирование одно из ведущих направлений трудовой деятельности в современном мире. Навыки математически-программного моделирования высоко оцениваются при найме на работу, особенно когда такие модели выглядят наглядно и понятно.

Решение задач программного проектирования и математического моделирования послужило идеей моего проекта по нескольким причинам:

* Навыки архитектурного проектирования ПО необходимы и высокооплачиваемы при найме в компанию на вакансию «программист»
* Понимание принципов модульного взаимодействия компонентов благоприятствует эффективности решения различных задач вообще, не только специфичных задач программирования
* Личный интерес, меня всегда интересовало, как можно красиво и не слишком сложно создавать красивую графику, на понятном мне языке, таком как Java, особенно мне интересно программировать необычные абстрактные математические концепции, такое как фазовое пространство

Задачи, которые я ставлю перед собой представлены:

* Изучить литературу и видеоматериалы по темам:
  + Общие принципы построения графического пользовательского интерфейса(далее – ГПИ)
  + Графические инструменты, доступные для языка Java
  + Материалы по некоторым математическим объектам, таких как клеточные автоматы, или фазовое пространство
* Провести отбор технологий
* Придумать несколько вариантов архитектуры, с последующим выбором наилучшего варианта
* Написать итоговое приложение
* Подвести итоги проделанной работы

# Теоретическая часть

## Отбор технологий

В этой части мы разберемся какие существуют технологии для вывода приложения на экран, и постараемся выбрать лучшую

### AWT

Начнём изучение этой темы в хронологическом порядке. Для языка Java, который был выпущен в 1995 году, с самого начала[[1]](#endnote-2) существовала стандартная реализация некоторых графических компонентов, на основе которых можно было написать графическое приложение. Она называлась Java AWT(Abstract Window Toolkit — англ. Абстрактный Оконный Инструментарий).

Она(библиотека) использовала связку java-кода с нативным кодом, написанным на C++.

Эта библиотека является очень тяжеловесной, и часто не удобной в использовании, к тому же, в силу того, что она не сама рисует компоненты на экране, а лишь использует системные вызовы, программы, написанные с использованием AWT работают медленно, и нестабильно.

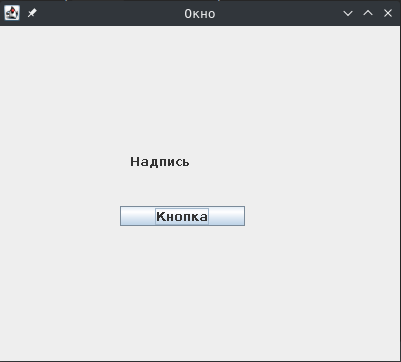
На данный момент она считается устаревшей, и начинать новые проекты на ней не стоит.

### Swing

На смену Java AWT почти сразу пришел Java Swing.[[2]](#endnote-3)(Версия языка 1.2). У Swing очень похожая архитектура на AWT, но с достаточно серьезными отличиями, компоненты Java Swing «Легковесны», т. е для отрисовки себя используют исключительно безопасный java-код, не задействуя на прямую системные вызовы, из-за этого графические программы стали быстрее и стабильнее, а так же такая архитектура позволила писать более гибкие программы, потому что весь код был написан на java, так что любой разработчик мог свободно подстраивать даже уже написанные компоненты под себя, к тому же, теперь программы на java стали выглядеть унифицированно, похоже друг на друга, не сильно завися от данной оконной системы.

Приведен пример кода с использованием Java Swing, и результат его работы

*package ru.cmr;  
import javax.swing.\*;  
public class SwingExample {  
 public static void main(String[] args) {  
 JFrame window = new JFrame("Окно");  
 JLabel label = new JLabel("Надпись");  
 JButton btn = new JButton("Кнопка");  
  
 label.setBounds(130, 110, 100, 50);  
 btn.setBounds(120, 180, 125, 20);  
  
 window.setLayout(null);  
 window.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
 window.setBounds(300,300, 400,360);  
 window.add(label);  
 window.add(btn);  
 window.setVisible(true);  
 }  
}*

 *Результат работы кода, приведенного выше.*

Так выглядит типичное приложение Java Swing, видно что код достаточно понятный, и результат можно получить достаточно быстро, но внешний вид достаточно устаревший.

### JavaFX

Хоть Swing был во многом лучше AWT, но все равно он быстро устаревал, и было решено разработать альтернативную технологию, которая будет и удобной, и красивой, и производительной. Эту технологию назвали JavaFX. Согласно сайту компании Oracle[[3]](#endnote-4) JavaFX имеет ряд преимуществ:

* **Java APIs** – JavaFX – это Java-библиотека, так что её собственный код хорошо взаимодействует с другими Java-библиотеками, а так же с JVM-языками, вроде Groovy, Kotlin, JRuby, Jython, и тд..
* **FXML, SceneBuilder** – Внешний вид JavaFX-приложения можно описывать в специальном формате, называемым FXML, а не прописывать все детали прямо в коде программы.
* **WebView** – в стандартную поставку JavaFX входит компонент, использующий WebKitHTML технологии для отрисовки внутри себя web-страницы. Что делает возможным с легкостью сделать, допустим, свою реализацию браузера, или сделать всё приложение — как комплекс web-страниц, а часть приложения на Java, запускает эти страницы.
* **Swing-совместимость** – JavaFX может спокойно работать со Swing, использовать уже написанные на Swing компоненты, а существующие приложения на Swing могут быть постепенно перестроены на JavaFX.
* **3D Графика** – существенное различие между Swing и JavaFX — это возможность использовать 3D графику.
* **Аппаратно-ускоренный процесс отрисовки** – JavaFX использует технологию, которая использует графический ускоритель, для отрисовки компонентов на экране.
* **Самодостаточная модель распространения приложения –** приложения на JavaFX распространяются в виде так называемых «слепков», другими словами, приложение распространяется в виде запускаемого файла для конкретной системы, и при запуске, запускается виртуальная машина, которая запускает само приложение, и эта виртуальная машина не связана с работой других виртуальных машин в системе(если, к примеру, работает сервер, он не будет затронут нашим приложением).

#### Архитектура

Многие возможности этой библиотеки существуют благодаря её архитектуре.

Это изображение взято с официального сайта компании Oracle[[4]](#endnote-5). Как видно за т. н. «публичном интерфейсом» скрыта целая сложная система, позволяющая создавать красивые и производительные приложения.



## Архитектурное изучение

Успешность ПО во многом зависит от выбранной архтектуры. Главный критерий — архитектура должна быть гибкой, и продуманной, то есть, удобной и легкорасширяемой.

### SOLID-принципы

Начнём с наиболее важной стороны разработки ПО, а именно SOLID-принципов. Акроним «SOLID» означает 5 основополагающих принципов, соблюдать которые по хорошему должны все разработчики любых програм.[[5]](#endnote-6)

Начнём разбор.

* SRP — (англ. Single Responsibility Principle, Принцип Единственной Ответственности)
  + Принцип Единой Ответственности заключается в том, что каждая програмная сущность должна отвечать за какую то одну вещь, и ни за что больше. Таким способом допускается меньше ошибок в коде, ведь когда нам надо изменить один функционал, нам не надо трогать другие, в противном случае придется вносить изменения каждый раз в один и тот же файл, даже если с точки зрения поставленной задачи это совершенно не нужно.
* OCP — (англ. Open-Closed Principle, Принцип открытости-закрытости)
  + Этот принцип говорит о том, что написанный код должен быть открыть для расширения и закрыт для модификации. К примеру, если мы пишем библиотеку, читающую данные из файла или из интернета, и в первой версии мы написали «в лоб», тогда все работало медленно, потому что требовалось постоянно ожидать ответ от операционной системы, и работать можно было только с текущим байтом, в то время как хотелось бы работать с полноценным объектом, состоящим из множества байтов. И вот была поставлена задача в новой версии добавить возможность чтения объектов. Первая мысль — переписать существующий код. Но тогда перестанет работать уже существующий код у клиентов нашей библиотеки. И на такой запрос принцип открытости-закрытости отвечает, что нам надо дописать расширение.

Ниже представлен утрированный пример.

Пример 1



*Первая версия библиотеки, читающая файлы по байтам.*

Пример 2.

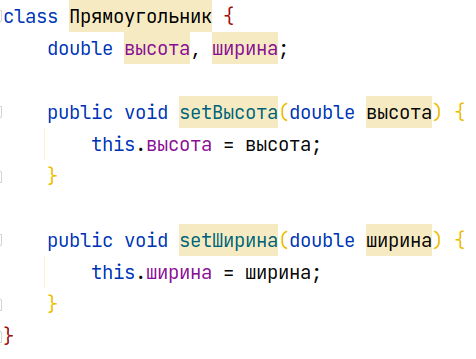


*Вторая версия, читающая файлы построчно.*

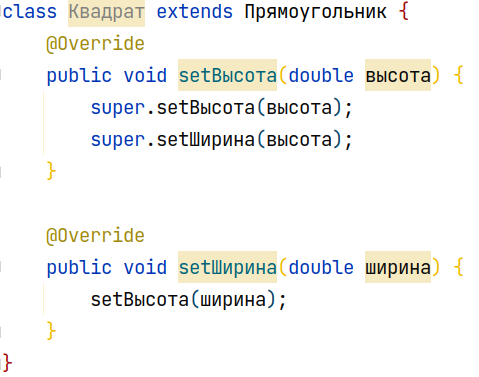
* LSP — (англ. Liskov Substitution Principle, Принцип Подстановки Барбары Лисков)
  + Подстановка Лисков говорит о том, что все потомки должны соблюдать интерфейс своего супер-класса. Другими словами, клиент не должен знать, о том, что он использует какую то конкретную реализацию того интерфейса, который он ожидает от нас, и использует.

Хрестоматийный пример с Квадратом и Прямоугольником.

Сначала определим класс Прямоугольник, у которого будет два измерения и методы для их задания(т.н Геттеры и Сеттеры)

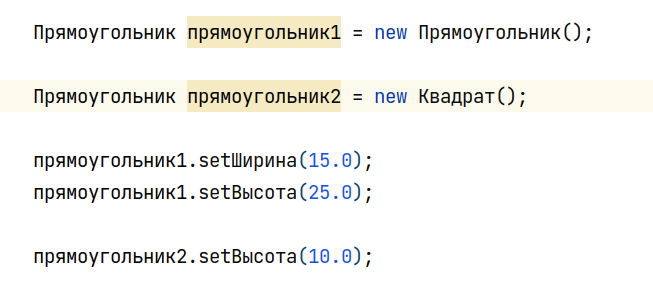


*Определение класса «Прямоугольник». Обратим внимание на сигнатуры(определения) методов, и их количество*



*У потомка должен сохранятся интерфейс ( совокупность всех методов ).*

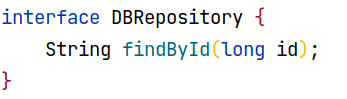
Благодаря такому подходу клиент может не знать, что использует конкретную реализацию прямоугольника — квадрат, благодаря чему код становится более гибким и переиспользуемым.

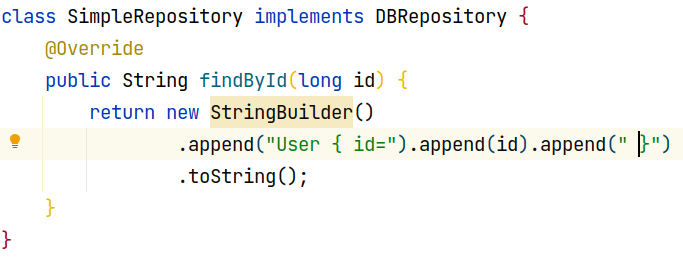


* ISP — (англ. Interface Segregation Principle, Принцип Разделения Интерфейсов)
  + Принцип говорит, что вместо объединения функциональности в один большой интерфейс, лучше создавать много маленьких интерфейсов. Пример, вместо одного интерфейса «СлушательРабочий» написать два маленьких: «Слушатель», «Рабочий». Потому что не всегда нам нужна и та, и та функциональность, гораздо лучше когда, мы можем выбирать какой интерфейс реализовывать, какой нет, в нашем примере все очевидно, но когда таких интерфейсов около 500 штук, соблюдения этого принципа становится очень критично.
* DIP — (англ. Dependency Inversion Principle, Принцип Инверсии Зависимостей)
  + Принцип Инверсии Зависимостей говорит о том, что высокоуровневые модули, не должны зависеть от низкоуровневых, а должны зависеть от абстракций.

Пример. Сервис по поиску аккаунтов в базе данных по уникальному идентификатору, и главная проблема в том, что у каждый базы данных свой протокол взаимодействия, поэтому мы объявили интерфейс, унифицирующий этот протокол.







### MVC-паттерны

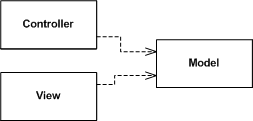
#### Обзор

Создание графического интерфейса для программы – это типовая задача в программировании, и для этого уже придуманы различные стандартные приемы, такие как архитектура «MVC»

#### «Классическая» архитектура

Стандартный вариант решения «MVC» расшифровывается так: «Модель — Внешний Вид(Внешнее представление) — Контроллер»

Такой подход заключается в том, что приложение разбивается на 3 связных уровня, каждый из которых ответственнен за что то одно.



*Так схематично выглядит архитектура MVC*

Модель – этот слой инкапсулирует в себе данные о модели из предметной области(примеры: Аккаунт клиента, Пункт продажи, Документы определённого расширения, и т. д.)

Внешний вид(представление) – слой, который прямо взаимодействует с пользователем, т. е. показывается на экране, обрабатывает кнопки и т. п., так же, обновляет свой контент на основе данных из модели.

Контроллер – тот слой, который занимается логикой предметной области(прим. Посчитать баланс на аккаунте, расстояние между городами, отправить уведомление на телефон, и т. д.)

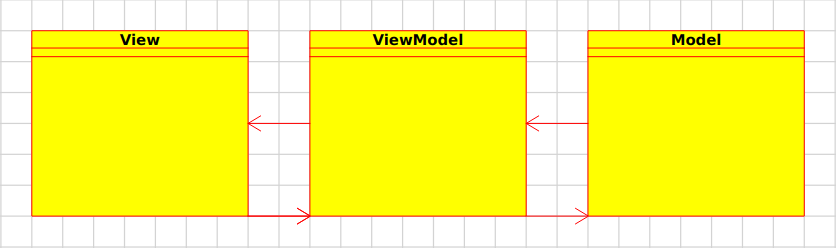
#### MVVM — архитектура

MVVM (Model-View-ViewModel) – расширение классического шаблона MVC.

Шаблон состоит из слоев:

* Model – отвечает за основную логику программы
* View – отвечает за отображение программы на экране
* ViewModel – слой-связь между абстракциями предметной области и уровнем представления

Архитектура такого приложения схематично выглядит так:



То есть, слой отображения, к примеру веб-страничка, при разработке или проектировании использует(зависит) от слоя модели-представления, который в свою очередь является оберткой для данных из слоя модели.

## Выбор задачи

Чтобы реализовать названные концепции нужна задача. Вместо того, чтобы придумывать абстрактную ситуацию вроде интернет магазина, или социальной сети я решил взять известную математическую задачу-игру.

### Клеточные автоматы и игра «Жизнь»

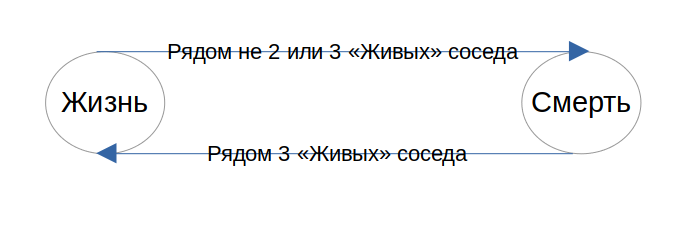
Клеточный автомат (КА) — дискретная модель, изучаемая в математике, состоит из бесконечной решеткой из элементов(клеток), у которых есть определенное множество состояний(к примеру «Жизнь» и «Смерть») и множество правил перехода клетки из одного состояния в другое.

Самый известный клеточный автомат это игра «Жизнь» от Джона Конвея.

Игра заключается в том, что мы определяем двумерный КА с состояниями: «Жизнь»(1) и «Смерть»(0); и правила:

1. Если клетка жива, и рядом с ней есть 2 или 3 тоже живых клеток, то она остаётся живой
2. Если клетка мертва, и рядом с ней 3 живых клетки, то она становится живой
3. В остальных случая клетка умирает

Этот автомат выражается в категорию:



# Глава 2. Практическая часть

В этой части мы реализуем поставленную задачу (Игра «Жизнь») с помощью выбранных технологий.

### Система сборки

Я буду использовать систему сборки для этого проекта «Gradle», она позволяет

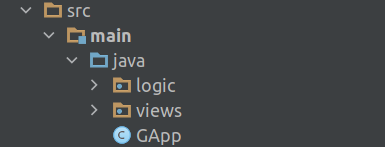
удобно и гибко подключать зависимости для проекта, а также т.н. «Инкрементальную сборку»,

когда подлежат компиляции и обработке только те компоненты, которые действительно в этом нуждаются.

#### Структура проекта

Для языка Java типичная ситуация, когда исходный код расфасован по «пакетам».

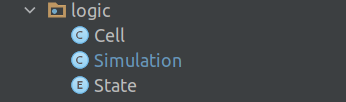
Пакет в этом контексте — это сущность, в которой находятся классы, служит своего рода пространством имен для классов, и отображается в виде директории в файловой системе.



В данном случае есть 3 пакета, т. н. «безымянный», «logic», «views», в каждом из которых лежат классы.

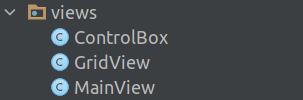
В пакете «logic» хранятся классы, отвечающие за основу работы приложения. А именно

Клетка(Cell), Состояние(State), Симуляция(Simulation)



Cell отвечает за конкретную клетку с её конкретным состоянием, которое определяется перечислением(константой) State. А Simulation в свою очередь использует объекты класса Cell, как опору для своей работы( Этот класс целиком отвечает за изменение состояний всех клеток на поле).

В пакете «views» лежат классы, ответственные за показ приложения на экране( элементы интерфейса)



Здесь «ControlBox» - это полоса управления, на ней расположены кнопки.

«GridView» – игровое поле, рисующее клетки в виде квадратиков на экране.

«MainView» – агрегирует эти 2 класса в себе, и показывает их в системном окне.

Остался один класс, это класс «App». Как понятно из названия, это точка входа в наше приложение( при запуске система вызывает особый метод в этом классе, после чего стартует само приложение).

## **Заключение**

В ходе работы над индивидуальным проектом была достигнута поставленная цель, которая заключалась в освоении технологий построения ГПИ. Для реализации поставленной цели были достигнуты задачи:

1. Были изучены основные способы создания ГПИ
2. Были исследованы графические инструментарии доступные для Java-платформы
3. Был проведен отбор технологий для реализации проекта
4. Была выбрана и реализована математическая задача

В ходе работы над индивидуальным проектом была достигнута поставленная цель, которая заключалась в создании электронных викторин в среде MS PowerPoint.

Для реализации поставленной цели мною были достигнуты задачи:

1) При выполнении работы были рассмотрены основные виды компьютерных викторин, представленные в среде Интернет;

2) Проанализировано программное обеспечение для создания компьютерных викторин, этапы их разработки.

3) Описана технология разработки компьютерных викторин.

4) Созданы компьютерные викторины для учащихся начальной школы.

Применение компьютерных викторин позволяет дифференцировать процесс обучения школьников с учетом их индивидуальных особенностей, дает возможность творчески работающему учителю расширить спектр способов предъявления учебной информации, позволяет осуществлять гибкое управление учебным процессом, является социально значимым и актуальным.

1. Oracle: официальный сайт компании-владельца языка Java [Электронный ресурс] – URL (Дата обращения 12.04.21)

   https://docs.oracle.com/javase/8/docs/api/index.html [↑](#endnote-ref-2)
2. Oracle: страница документации [Электронный ресурс] – URL (Дата обращения 12.04.21)

   https://docs.oracle.com/javase/8/docs/api/index.html?overview-summary.html [↑](#endnote-ref-3)
3. Oracle: официальная статья по JavaFX [Электронный ресурс] — URL (Дата обращения 18.04.21)

   https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm [↑](#endnote-ref-4)
4. Oracle: статья по архитектуре JavaFX [Электронный ресурс] – URL (Дата обращения 18.04.21)

   https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-architecture.htm#CHDFDAFF [↑](#endnote-ref-5)
5. Jrebel: статья на тему SOLID принципов[Электронный ресурс] — URL(Дата обращения 08.07.21)

   https://www.jrebel.com/blog/solid-principles-in-java [↑](#endnote-ref-6)