Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет «Высшая школа экономики»

Факультет информатики, математики и компьютерных наук

Образовательная программа «Программная инженерия»

Курсовая работа

«‎Разработка программной системы управления лифтами»

по направлению подготовки 09.03.04 "Программная инженерия"

Выполнила:

студентка 2 курса

Самсонова Карина Евгеньевна

Руководитель:

К.ф.м.н., доцент базовой кафедры "Мера" НИУ ВШЭ-НН

Лейкин Максим Валентинович

Нижний Новгород 2022

# Содержание

[Содержание 1](#_Toc105280649)

[Введение 2](#_Toc105280650)

[1. Постановка задачи 4](#_Toc105280651)

[2. Выбор инструментальных средств 5](#_Toc105280652)

[3. Проектирование и реализация 8](#_Toc105280653)

[3.1. Описание алгоритма работы лифтовой системы 9](#_Toc105280654)

[3.2. Принцип работы генератора пассажиропотока 10](#_Toc105280655)

[3.3. Разработка интерфейса программы 10](#_Toc105280656)

[4. Описание разработанной программы 13](#_Toc105280657)

[5. Заключение 15](#_Toc105280658)

[6. Использованные материалы 16](#_Toc105280659)

# Введение

На сегодняшний день лифт стал неотъемлемой частью жизни людей. Это подъемно-транспортное устройство, которое перевозит пассажиров лифта или грузы в вертикальном направлении по жестким стальным лифтовым направляющим. Он позволяет эффективно передвигаться между этажами, не затрачивая при этом большого количества времени на ожидание и путь до пункта назначения.

С каждым годом в мире появляется все больше многоэтажных зданий. Это обусловлено как развитием строительных технологий и материалов, так и стремительно растущей потребностью людей в подобных сооружениях, обладающих большим количеством помещений для проживания и/или деятельности людей. Такой увеличивающийся спрос можно объяснить множеством факторов, например, увеличение численности населения планеты или рост отдельных компаний, требующих обширные офисные пространства для обеспечения рабочих мест.

Сложно представить современное многоэтажное здание без лифтовой системы. Чаще всего достаточная этажность для установления лифтового оборудования в здании составляет 5 этажей, а передвижение по более высотным сооружениям или небоскребам, таким как «Бурдж-Халифа», этажность которого составляет 163 этажа, просто невозможно без высокоразвитой лифтовой системы.

С ростом средней этажности наземных сооружений и увеличением пассажиропотока в них, вопрос повышения эффективности работы системы лифтов становится все более насущным. Идеальный алгоритм работы системы лифтов должен быть универсален для разных типов сооружений, которые отличаются рядом параметров, таких как этажность здания, количество лифтов на этаже, грузоподъемность кабины и пр. К тому же первостепенной задачей является повышение эффективности работы системы. Минимизация времени ожидания пассажиром кабины, а также ожидания в пути до пункта назначения является одной из первостепенных задач, так как прежде всего лифт существует именно для увеличения уровня комфорта и скорости при перемещении человека между этажей здания. Помимо этого, важно минимизировать пробег лифта с целью пролонгации срока и удешевлении его эксплуатации. Тем не менее в настоящее время не существует универсального алгоритма, отвечающего вышеприведенным требованиям, чем и обусловлена **актуальность** выбранной темы курсовой работы.

# Постановка задачи

Таким образом, **цель** настоящей работы – разработка эффективного и универсального алгоритма работы лифтовой системы и создание наглядного интерфейса для демонстрации работы данного алгоритма.

Согласно указанной цели, возникает необходимость реализовать следующие **задачи**:

- Выбор инструментальных средств;

- Проектирование и реализация;

- Описание реализации.

Более того, итоговая программа должна отвечать следующим **требованиям**:

- Универсальность – возможность ручного ввода ряда параметров здания и лифтовой системы.

- Эффективность – алгоритм работы системы лифтов должен быть направлен на уменьшение ожидания пассажиров и пробега лифтов.

- Логичность – данная система должна оставаться актуальной в рамках реальных условий жизни людей, то есть учитывать такие факторы, как повышенный пассажиропоток на 1 этаже, возможность пассажиров заходить в проезжающую мимо кабину лифта и пр.

- Наглядность – интерфейс программы должен быть приятен глазу и понятно описывать поведение системы лифтов в различных условиях.

# Выбор инструментальных средств

Для разработки программного проекта был выбран язык программирования Java.

Java — высокоуровневый объектно-ориентированный язык программирования общего назначения. Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре, для которой существует реализация виртуальной Java-машины (JVM).

Данный язык программирования существует более двух десятилетий, но не теряет своей актуальности по сей день. Использование Java в программном проекте было обусловлено рядом причин:

- Объектно-ориентированный подход.

Java позволяет создавать модульные программы, исходный код которых может использоваться многократно. Объектно-ориентированное программирование - методология программирования, основанная на представлении программы в виде совокупности взаимодействующих объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования. Такой подход не только является актуальным с точки зрения удобства реализации, изменения и понимания самой программы, но и идеологически соответствует проекту, так как основной целью является разработка системы эффективного взаимодействия отдельных сущностей (объектов), таких как пассажир и лифт.

- Встроенные в язык средства создания многопоточных приложений.

При помощи данных средств можно выделить в приложении несколько потоков, которые будут выполнять различные задачи одновременно. Это позволяет системе работать более эффективно, так как каждая отдельная кабина лифта сможет выполнять свои функции независимо от других. Поток заявок пассажиров, желающих добраться до какого-либо этажа, так же работает отдельно, что позволяет ему быть более непредсказуемым и, соответственно, более правдоподобным, приближенным к реальности. К тому же принцип многопоточности позволяет одновременно осуществлять работу алгоритма лифтовой системы и поддержку интерфейса.

- Автоматическое управление памятью.

Сборка мусора — одна из форм автоматического управления памятью, при которой происходит специальный процесс, периодически освобождающий память, удаляя из нее объекты, ставшие ненужными. Автоматическая сборка мусора позволяет повысить безопасность доступа к памяти. Разработка симуляции лифтовой системы предполагает ее бесконечную (или продолжительную) непрерывную работу, что напрямую связано с созданием большого количества переменных и использованием значительных массивов памяти. В таком случае автоматическая сборка мусора позволяет избежать неожиданных утечек памяти, которые могли бы привести к неисправной работе или краху программы.

- Набор стандартных коллекций.

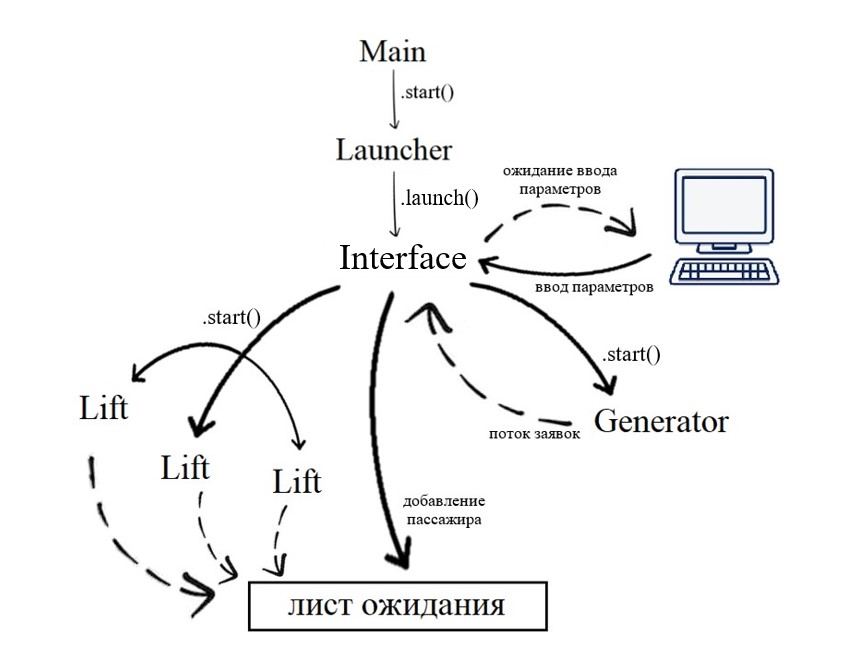
Пакет java.util включает в себя внушительный набор различных коллекций для удобной и эффективной обработки и хранения данных. Среди базовых интерфейсов в Java существуют такие, как Queue, List, Set, Map и тд. С помощью данных интерфейсов и абстрактных классов в Java реализуется широкая палитра классов коллекций. Использование списков, множеств, очередей и прочих коллекций, зачастую значительно повышает скорость работы с данными, что делает программу в целом заметно быстрее.

Помимо вышеупомянутых преимуществ, Java также является высокоуровневым, удобным языком программирования, имеющим большое количество функций для обеспечения безопасности данных и обладающим простым и понятным синтаксисом.

Для разработки интерфейса был выбран инструментарий JavaFX, предназначенный для создания кроссплатформенных графических приложений на платформе Java. Он позволяет создавать приложения с богатой насыщенной графикой благодаря использованию аппаратного ускорения графики и возможностей GPU. JavaFX предоставляет большие возможности по сравнению с рядом других подобных платформ. Это и большой набор элементов управления, и возможности по работе с мультимедиа, двухмерной и трехмерной графикой, декларативный способ описания интерфейса с помощью языка разметки FXML, возможность стилизации интерфейса с помощью CSS и многое другое.

Процесс разработки происходил в интегрированной среде разработки от JetBrains - IntelliJ IDEA (Community-версия). Это удобная и функциональная IDE для разработки приложений на языках, работающих под управлением виртуальной машины Java. Помимо эргономичного дизайна, она предлагает такие достоинства, как умное завершение кода, автоматизация однообразных задач и надежные инструменты рефакторинга, а также интегрированные системы контроля версий, широкий спектр поддерживаемых языков и фреймворков.

# Проектирование и реализация

Общая схема работы программы с момента запуска наглядно представлена на рисунке:

Из класса Main происходит старт потока класса Launcher. Данный класс предназначен для работы интерфейса приложения в отдельном потоке, именно в потоке Launcher запускается класс Interface, отвечающий за графику программы. Далее происходит ожидание программой ввода параметров симуляции (число этажей, число лифтов, вместимость кабины). После ввода параметров из класса Interface происходит старт потока генератора заявок и старт потоков лифтов. После получения заявки о новом пассажире Interface добавляет его в лист ожидания, обращаясь к которому лифты узнают, куда им двигаться.

Это было общее и краткое представление работы программы. Далее будут приведены более подробные описания отдельных аспектов проекта.

# Описание алгоритма работы лифтовой системы

Каждый лифт в системе работает в отдельном бесконечном потоке, придерживаясь единого алгоритма работы. В бесконечном цикле лифт сначала спит около одной секунды (потому что постоянный поиск пассажиров только перегружает программу и не имеет практического смысла), а далее обращается к листу ожидания в поиске свободных пассажиров.

Каждый пассажир имеет определенный статус в зависимости от того, свободен ли он, ожидает ли он определенный лифт, который уже едет до его этажа или он едет внутри кабины лифта. Если пассажир приходит на этаж, где другой пассажир, который собирается ехать в том же направлении (вверх/вниз), уже ждет определенный лифт, то этот пассажир так же получает статус ожидания. Если кабина лифта приехала, но пассажир в нее не поместился из-за ограниченной вместимости, то он обретает статус «свободен». Это повышает уровень правдоподобности симуляции, ведь в жизни человек, пришедший на этаж, не нажимает кнопку вызова лифта, если она уже нажата, а если пассажир не смог вместиться в кабину лифта, его ожидание «возобновляется».

Когда лифт находит свободного пассажира, он проверяет, нет ли более близкого к данному пассажиру свободного лифта. Эта проверка была сделана для того, чтобы минимизировать пробег лифтов, особенно при большой этажности здания. Если данный лифт действительно является оптимальным, то он присваивает своему пассажиру статус ожидания и отправляется на его этаж.

Если за время, пока лифт едет на нужный этаж, его пассажир был перехвачен другим лифтом, проезжающим мимо, то данный лифт теряет свою задачу и переходит в статус «свободен», текущая итерация бесконечного цикла заканчивается.

По прибытии на нужный этаж, запускается алгоритм работы лифта: он забирает всех, кто едет в нужном направлении на данном этаже и, пока загруженность данного лифта составляет больше 0 пассажиров, он едет в своем направлении, проверяя на каждом этаже, может ли он кого-либо высадить или принять. Если загруженность кабины в какой-то момент времени становится равна нулю, лифт останавливается и переходит в статус «свободен», текущая итерация бесконечного цикла заканчивается.

# Принцип работы генератора пассажиропотока

Поток генератора запускается из класса Interface после ввода параметров и начала симуляции. Генератор работает в бесконечном цикле и, отсекая некий определенный с помощью рандомайзера промежуток времени от 1 до 10 секунд, вызывает в классе интерфейса функцию addPassenger, которая так же рандомно генерирует пассажира и добавляет его в список ожидания. Для каждого пассажира генерируются этажи отправления и назначения, причем вероятность того, что один из этажей будет первым равна 50%. Это так же было введено для повышения правдоподобности симуляции, ведь на первом этаже всегда самый плотный трафик людей, потому что именно на нем находится выход из здания. Также для каждого пассажира генерируется иконка, это может быть изображение либо мужчины, либо женщины, для большей наглядности и приближенности к реальности.

# Разработка интерфейса программы

Графика приложения полностью создана инструментами JavaFX. Интерфейс создается в классе Interface. Изначально это окно с ползунками для настройки параметров симуляции. После нажатия кнопки «Далее» запускается симуляция с указанными параметрами.

Создаются отдельные иконки для каждой лифтовой кабины, а также для каждого этажа. Они располагаются на экране так, чтобы кабинки изначально были на 1 этаже. Каждый этаж имеет тип StackPane, это модель компоновки, которая позволяет располагать одни объекты поверх других в виде стека. Каждый этаж хранит простой прямоугольник для наглядности, номер данного этажа, а также кнопки «вверх» и «вниз», которые загораются, если данная кнопка находится в статусе «нажата».

Управление кабинками лифтов происходит при помощи TranslateTransition, что позволяет плавно перемещать их вверх и вниз на определенное расстояние. Иконки пассажиров так же перемещаются при помощи данной анимации. Перемещение человека вовнутрь кабины, выход из кабины, а также движение лифта вместе с его пассажирами осуществляются при помощи отдельных функций, прописанных в классе интерфейса:

*/\*\*  
 \* Процедура передвижения иконки кабины лифта  
 \** ***@param*** *direction - направление передвижения  
 \** ***@param*** *cabin - иконка кабины лифта  
 \*/*public static void moveCabin(int direction, StackPane cabin){  
 TranslateTransition trans\_l = new TranslateTransition(Duration.*millis*(800), cabin);  
 trans\_l.setByY(-1 \* direction \* 51.3);  
 trans\_l.play();  
}  
  
*/\*\*  
 \* Процедура передвижения иконки пассажира  
 \** ***@param*** *direction - направление передвижения  
 \** ***@param*** *fellow - иконка пассажира  
 \*/*public static void moveFellow(int direction, Passenger fellow) {  
 TranslateTransition trans\_f = new TranslateTransition(Duration.*millis*(800), fellow.image);  
 trans\_f.setByY(-1 \* direction \* 51.3);  
 trans\_f.play();  
}  
  
*/\*\*  
 \* Процедура отображения посадки пассажира в кабину лифта  
 \** ***@param*** *pass - пассажир  
 \** ***@param*** *index - индекс лифта  
 \*/*public static void getInCabin(Passenger pass, int index) {  
 TranslateTransition transition = new TranslateTransition(Duration.*millis*(200), pass.image);  
 transition.setByX(-51\*(*lifts\_num*-index));  
 transition.play();  
}  
  
*/\*\*  
 \* Процедура отображения высадки пассажира из кабины лифта  
 \** ***@param*** *pass - пассажир  
 \*/*public static void dropOff(Passenger pass) {  
 TranslateTransition transition = new TranslateTransition(Duration.*millis*(500), pass.image);  
 transition.setByX(50\**lifts\_num*+150);  
 transition.play();  
}

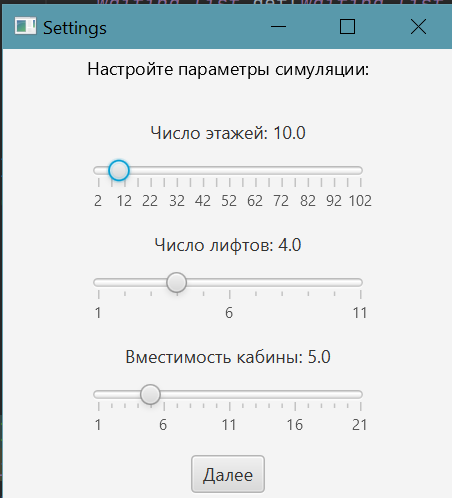
Данные функции статические и вызываются из класса Lift при необходимости.

*/\*\*  
 \* Процедура отображения высадки пассажира из кабины лифта  
 \** ***@param*** *pass - пассажир  
 \*/*public static void dropOff(Passenger pass) {  
 TranslateTransition transition = new TranslateTransition(Duration.*millis*(500), pass.image);  
 transition.setByX(50\**lifts\_num*+150);  
 transition.play();  
}

Управление кнопками так же происходит из класса Lift.

# Описание разработанной программы

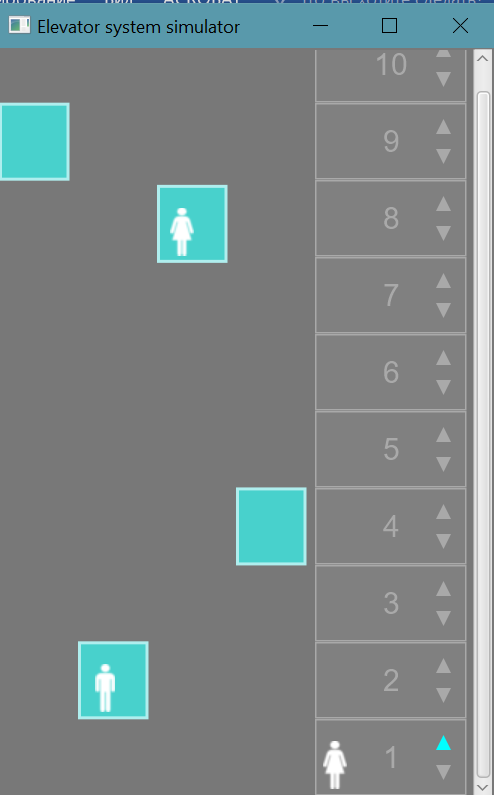
При запуске программы пользователь может наблюдать окно ввода параметров:



По умолчанию число этажей, число лифтов и вместимость кабины равны соответственно 10, 4 и 5.

После нажатия кнопки «Далее» текущее окно закрывается, и появляется симуляция системы.

По истечении некоторого времени начинают появляться новые пассажиры, а кабинки лифтов передвигаются каждая в нужном направлении.



Приложение хорошо показывает себя при продолжительной работе, а также демонстрирует эффективную работу алгоритма, так как даже при активном потоке пассажиров лифтовой системе удается быстро развезти всех по пунктам назначения.

# Заключение

Таким образом, в ходе работы был создан гибкий и эффективный алгоритм работы системы лифтов, который не теряет своего качества в условиях, приближенных к реальности. Работа данного алгоритма была продемонстрирована приятным и ясным графическим интерфейсом. Все пункты технического задания были реализованы, были выбраны подходящие инструменты реализации, а также проведена успешная разработка программного проекта, результат которого был описан.

В будущем можно поработать над повышением скорости работы алгоритма и над улучшением реализации проекта за счет выбора других инструментальных средств и библиотек, например, java.util.concurrent для более надежной и эффективной работы с потоками. Также можно провести доработки по интерфейсу, например, чтобы сделать его сложнее и интереснее для наблюдения.

Код проекта хранится на репозитории: <https://github.com/karina-samsonova/course_work>

# Использованные материалы

1. Патрик Ноутон, Герберт Шилдт. Java 2 [наиболее полн. рук.: пер. с англ.]. – Спб.: БХВ-Петербург, 2008. – 1050 с.: ил. ISBN:9785941570126, 5941570120
2. Кей Хорстманн, Гари Корнелл. Java. Библиотека профессионала, т. 1. Основы, пер. с англ. — М.: Вильямс, 2008. — 813 с.: ил. ISBN:9785845913784, 5845913780
3. Кей Хорстманн, Гари Корнелл. Java. Библиотека профессионала, т. 2. Расширенные средства, пер. с англ. — М.: Вильямс, 2014. — 1008 с.: ил. ISBN:9785845903457, 5845903459
4. JavaFX 2 — Справочная документация — [Электронный ресурс]. — URL: <https://docs.oracle.com/javafx/2/>
5. JavaFX — Documentation Project — Version Unspecified [Электронный ресурс]. — URL: <https://fxdocs.github.io/docs/html5/>
6. Processes and Threads – Справочная документация — [Электронный ресурс]. — URL: <https://docs.oracle.com/javase/tutorial/essential/concurrency/procthread.html>