|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СОДЕРЖАНИЕ  ВВЕДЕНИЕ 6  1 ПОСТАНОВКА ЗАДАЧИ 7  1.1 Описание предметной области 7  1.2 Цели и задачи курсового проектирования 8  2 ВЫБОР МЕТОДА РЕШЕНИЯ 9  2.1 Описание ОО подхода 9  2.2 Выбор инструментальных средств 12  2.3 Описание входных и выходных данных 13  3 МЕТОД РЕШЕНИЯ 15  3.1 Объектно-ориентированный анализ 15  3.2 Объектно-ориентированное проектирование 17  3.2.1 Диаграмма вариантов использования 17  3.2.2 Диаграмма классов 19  3.2.3 Диаграмма объектов 20  3.2.4 Диаграмма состояний 21  3.2.5 Диаграмма активности 24  3.3 Объектно-ориентированное программирование 26  3.3.1 Диаграмма компонентов 26  3.3.2 Диаграмма развёртывания 26  3.3.3 Протоколы классов 27  ЗАКЛЮЧЕНИЕ 29  СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 30  ПРИЛОЖЕНИЕ А. ТЕХНИЧЕСКОЕ ЗАДАНИЕ 31  ПРИЛОЖЕНИЕ Б. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 36  ПРИЛОЖЕНИЕ В. ЭКРАННЫЕ ФОРМЫ 37  ПРИЛОЖЕНИЕ Г. ЛИСТИНГ ПРОГРАММНЫХ МОДУЛЕЙ 39 | | | | | |
|  |  |  |  |  |  |

# ВЕДЕНИЕ

В данной работе производится моделирование информационной системы «Кинотеатры».

Актуальность работы состоит в том, что она рассматривает модель информирования сотрудников справочной службы кинотеатров города. Имея доступ к базам сведений кинотеатров, они без проблем могут задать нужный запрос и получить на них ответ, что в свою обеспечивает больший для клиентов кинотеатров, а значит и повышает эффективность работы самих кинотеатров.

Объектом разработки является программа, написанная на языке С++,

эмулирующая работу справочной службы кинотеатров.

Предмет разработки – объектная модель информационной системы.

Целью курсового проектирования является разработка программного продукта – системы имитационного моделирования.

Задачи курсового проектирования:

* провести объектно-ориентированный анализ системы;
* построить объектную модель системы;
* спроектировать алгоритм работы системы;
* реализовать программу на языке С++, моделирующую данную систему.

Результат курсового проектирования: работающая программа, осуществляющая человеко-машинное взаимодействие согласно техническому заданию.

# 1 ПОСТАНОВКА ЗАДАЧИ

## 1.1 Описание предметной области

При помощи справочной службы кинотеатров города значительно повышается комфортабельность процесса поиска нужного фильма, сеанса или кинотеатра. Вместо неопределенности и траты времени, необходимо лишь сформировать нужный клиенту запрос в виде определенного набора данных и обратиться к работникам информационной службы. Получив необходимые данные, они за кратчайшие сроки с совершенной точностью найдут и предложат клиенту все подходящие варианты.

Изначально данная система имеет доступ ко всем данным кинотеатров города и фильмам в целом. Благодаря чему, возможна не только демонстрация необходимых сведений, но и работа с ними. Доступны такие возможности, как добавить, удалить или же редактировать ту или иную информацию, что обеспечивает динамичность данной системы.

Благодаря наличию легкодоступного и интуитивно понятного интерфейса, человек работающий с данной системой не будет иметь каких-либо трудностей или неопределенностей с взаимодействием с программой.

Запустив систему, на экране откроется консоль с дальнейшими инструкциями. Изучив их пользователь может вводить команды согласно инструкциям. Если пользователь программы случайно ошибется, то программа предусмотрит и укажет об этом, исключая неверный ход событий приложения.

Главные объекты данной системы — кинотеатры, фильмы и репертуары. Все они образуют целостную систему благодаря взаимосвязям друг с другом. Фильмы хранятся в репертуарах, а репертуары — кинотеатрах. Работникам дана возможность демонстрации как общих данных, так и отдельных благодаря поиску по нужным свойствам объектов.

При эксплуатации системы, пользователем не стоит заботиться о сохранности данных или их утечке. Программа автоматически загружает все данные при ее запуске и сохраняет при выходе из нее. Все эти и другие особенности данной системы обеспечивает наибольшую эффективность работы справочной службы

## 1.2 Цели и задачи курсового проектирования

Цель курсового проектирования – проанализировать, спроектировать и реализовать модель информациоонной системы «Турникет метрополитена».

При разработке системы следует выделить несколько задач:

* произвести объектно-ориентированнный анализ предметной области;
* спроектировать систему, осуществляющую взаимодействие между пользователем и её составными частями, построить соответсвующие UML-диаграммы;
* создать функциональный программный продукт на основе произведённого анализа и проектирования, соответствующий техническому заданию;
* оформить руководство пользователя согласно ГОСТам.

2 ВЫБОР МЕТОДА РЕШЕНИЯ

2.1 Описание ОО подхода

Определение классов и объектов – одна из самых сложных задач объектно-ориентированного проектирования.

Класс – это множество объектов, связанных общностью структуры и по­ведения. Любой объект является экземпляром класса.

Основополагающей идеей объектно-ориентированного подхода является объединение данных и действий, производимых над этими данными, в единое целое, которое называется объектом.

Объект имеет определённые свойства. Состояние объекта задаётся значениями его признаков. Объект «знает», как решать определённые задачи, то есть располагает методами решения. Программа, написанная с использованием ООП, состоит из объектов, которые могут взаимодействовать между собой. Ранее отмечалось, что программная реализация объекта представляет собой объединение данных и процедур их обработки. Переменные объектного типа называют экземплярами объекта. В отличие от типа «запись», объектный тип содержит не только поля, описывающие данные, но также процедуры и функции, описания которых содержится в описании объекта. Эти процедуры и функции называют методами. Методам объекта доступны его поля. Следует отметить, что методы и их параметры определяются в описании объекта, а их реализация даётся вне этого описания, в том месте программы, которое предшествует вызову данного метода. В описании объекта фактически содержаться лишь шаблоны обращения к методам, которые необходимы компилятору для проверки соответствия количества параметров и их типов при обращении к методам.

Если необходимо считать какие-либо данные объекта, нужно вызвать соответствующий метод, который выполнит считывание и возвратит требуемое значение.

Прямой доступ к данным невозможен. Данные сокрыты от внешнего воздействия, что защищает их от случайного изменения. Говорят, что данные и методы инкапсулированы.

Инкапсуляция является важнейшим свойством объектов, на котором строится объектно-ориентированное программирование. Инкапсуляция заключается в том, что объект скрывает в себе детали, которые несущественны для использования объекта. В традиционном подходе к программированию с использованием глобальных переменных программист не был застрахован от ошибок, связанных с использованием процедур, не предназначенных для обработки данных, связанных с этими переменными.

Наследование – это ещё одно базовое понятие объектно-ориентированного программирования. Наследование позволяет определять новые объекты, используя свойства прежних, дополняя или изменяя их. Объект-наследник получает все поля и методы «родителя», к которым он может добавить свои собственные поля и методы или заменить («перекрыть») их своими методами. Привлекательность наследования заключается в том, что если некий объект был уже определён и отлажен, он может быть использован и в других программах. При этом может оказаться, что новая задача отличается от предыдущей, и возникает необходимость некоторой модификации как данных, так и методов их обработки. Программисту приходится решать дилемму – создать объекты заново или использовать результаты предыдущей работы, применяя механизм наследования. Первый путь менее эффективен, так как требует дополнительных затрат времени на отладку и тестирование. Во втором случае часть этой работы оказывается выполненной, что сокращает время на разработку новой программы. Программист при этом может и не знать деталей реализации объекта-родителя. Наследование позволяет создавать иерархические, связанные отношениями подчинения, структуры данных.

Виртуальный метод(виртуальная функция) – в объектно-ориентированном программировании метод (функция) класса, который может быть переопределён в классах-наследниках так, что конкретная реализация метода для вызова будет определяться во время исполнения. Таким образом, программисту необязательно знать точный тип объекта для работы с ним через виртуальные методы: достаточно лишь знать, что объект принадлежит классу или наследнику класса, в котором метод объявлен.

Виртуальные методы – один из важнейших приёмов реализации полиморфизма. Они позволяют создавать общий код, который может работать как с объектами базового класса, так и с объектами любого его класса-наследника. При этом базовый класс определяет способ работы с объектами и любые его наследники могут предоставлять конкретную реализацию этого способа.

Переменные объектного типа могут быть динамическими, то есть размещаться в памяти только во время их использования. Для работы с динамическими объектами используются расширенный синтаксис процедур New и Delete. Обе процедуры в этом случае содержат в качестве второго параметра вызов конструктора или деструктора для выделения или освобождения памяти переменной объектного типа.

Полиморфизмзаключается в том, что одно и то же имя может соответствовать различным действиям в зависимости от типа объекта. Полиморфизм напрямую связан с механизмом позднего связывания. Решение о том, какая операция должна быть выполнена в конкретной ситуации, принимается во время выполнения программы. Когда существующая операция, например = или +, наделяется возможностью совершать действия над операндами нового типа, говорят, что такая операция является перегруженной. Перегрузка представляет собой частный случай полиморфизма и является важным инструментом ООП.

Конструктор – функция-член класса, определяющая способ создания объекта. Имя конструктора должно совпадать с именем класса.

Существует несколько форм конструктора. Наиболее часто используется конструктор с параметрами. Эти параметры используются для одновременной инициализации создаваемых объектов.

Деструктор – специальная функция класса, которая отвечает за уничтожение объекта. Имя деструктора совпадает с именем класса, перед которым ставится символ «тильда»: **~** .

Типичная программа на языке C++ состоит из совокупности объектов, взаимодействующих между собой посредством вызова методов друг друга. Структура программы на C++ приводится на рисунке 2.1.

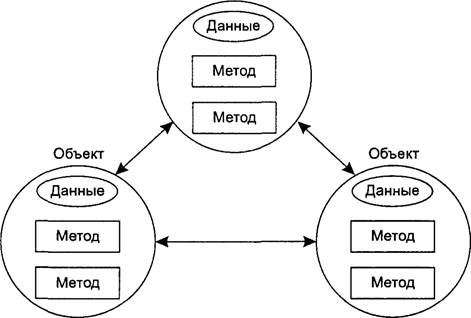


Рисунок 2.1 – Структура программы

## 2.2 Выбор инструментальных средств

Для построения моделей системы с рассмотрением различных её свойств целесообразно использовать язык разметки UML. В качестве программы, осуществляющей построение UML-диаграмм выбран Visual Paradigm.

В качестве IDE для разработки программы была выбрана Miscrosoft Visual Studio 2019, поскольку данная система программирования обладает следующими необходимыми функциями:

* подсветка синтаксиса;
* автодополнение кода;
* автоматическое выявление синтаксических и семантических ошибок;
* возможность использования объектно-ориентированного подхода;
* удобный интерфейс;
* наличие предкомпилированных модулей и библиотек.

## 2.3 Описание входных и выходных данных

Данные, фигурирующие в системе, можно разделить на входные, выходные и промежуточные. Также на эти данные накладываются определённые ограничения, возникающие вследствие технических характеристик компьютерной техники, представления данных в компьютере или вызванные непосредственно архитектурой и функционалом самой системы.

Входными данными является любая информация, поступающая в систему извне.

Выходные данные – это форматированные данные, выводимые программой в каком-либо виде (в файл, на устройство вывода и т.п.).

Промежуточные данные – набор данных, существующих лишь во время работы системы и использующиеся только системой на стадии обработки входных и формирования выходных данных.

Для данной предметной области перечень входных, выходных и промежуточных данных приведен в таблице 2.1.

Таблица 2.1 – Входные и выходные данные

|  |  |
| --- | --- |
| Данные | Составляющие |
| Входные данные | 1) номер команды – целочисленное число. Ограничение – 1 цифра, которая принадлежит диапозону чисел от 0 до максимального номера команды;  2) свойства объектов – строка или целочисленное число. Ограничение – перечисление нескольких значений через запятую. |
| Выходные данные | 1) нумерованные команды и инструкции – строка, если команда, то с номеров в начале.;  2) данные объектов, сведения – строка;  3) файлы cinemas.db, films.db, reperoires.db, содержащие структурированные строки с данными об объектах |
| Промежуточные данные | Данные объектов, которые добавляли, удаляли или изменяли, существуют лишь во время работы системы до тех пор, пока не выйти из нее с помощью инструкций, иначе все измененные данные будут утеряны |

# 3 МЕТОД РЕШЕНИЯ

## 3.1 Объектно-ориентированный анализ

Выполним объектно-ориентированный анализ предметной области. В результате анализа предметной области был составлен словарь предметной области приведенный в таблице 3.1.

Таблица 3.1 – Словарь предметной области

|  |  |  |
| --- | --- | --- |
| Существительное | Глагол | Прочее |
| Кинотеатр  Фильм  Репертуар | Ввести  Сохранить  Найти  Обнаружить  Закрыть  Открыть  Показать  Добавить  Удалить  Изменить  Сравнить  Очистить  Считать данные  Записать данные | Тип  Ключ  Название  Киностудия  Жанр  Операторы  Продюсеры  Актеры  Место  Зал  Цена  Дата  Состояние  Адрес  Категория Сеансы |

На основе словаря предметной области составим объектно-ориентированный словарь – таблица 3.2.

Таблица 3.2 – ОО словарь предметной области

|  |  |  |
| --- | --- | --- |
| Класс/Сущность | Свойство/Состояние | Метод/Функция |
| Кинотеатр (Cinema) | Ключ - id  Название – name  Адрес – address  Категория – category  Кол-во мест – placec  Кол-во залов – halls  Состояние – state | Сохранить ключ -set\_id()  Ввести – input()  Найти – find()  Показать – show()  Сохранить – save() |

Продолжение таблицы 3.2

|  |  |  |
| --- | --- | --- |
| Класс/Сущность | Свойство/Состояние | Метод/Функция |
| Кинотеатр (Cinema) | Репертуары - reps |  |
| Фильм (Film) | Ключ - id  Название – name  Продюсеры – producers  Операторы – opers  Жанры – genres  Актеры - actors | Сохранить ключ -set\_id()  Ввести – input()  Найти – find()  Показать – show()  Сохранить – save()  Обнаружить – search() |
| Репертуар (Repertoire) | Ключ - id  Дата – date  Цена – price  Кол-во свободных мест – free\_places  Фильм - Film | Сохранить ключ -set\_id()  Ввести – input()  Найти – find()  Показать – show()  Сохранить – save()  Обнаружить – search() |

## 3.2 Объектно-ориентированное проектирование

### 3.2.1 Диаграмма вариантов использования

Диаграмма вариантов использования (Use Case Diagram) позволяет описать список операций, которые выполняет система. Действующее лицо представляет собой любую внешнюю по отношению к моделируемой системе сущность, которая взаимодействует с системой и использует её функциональные возможности для достижения определённых целей или решения частных задач.

На рисунке 3.1 изображена диаграмма вариантов использования для системы «Кинотеатры». В качестве актёров выступают объекты системы: информационная модель кинотеатра, фильма и репертуара.

Действиями являются функции системы, такие как: работа с картой, перекрытие доступа, управление индикаторами и вывод информации о количестве оставшихся на карте поездок.

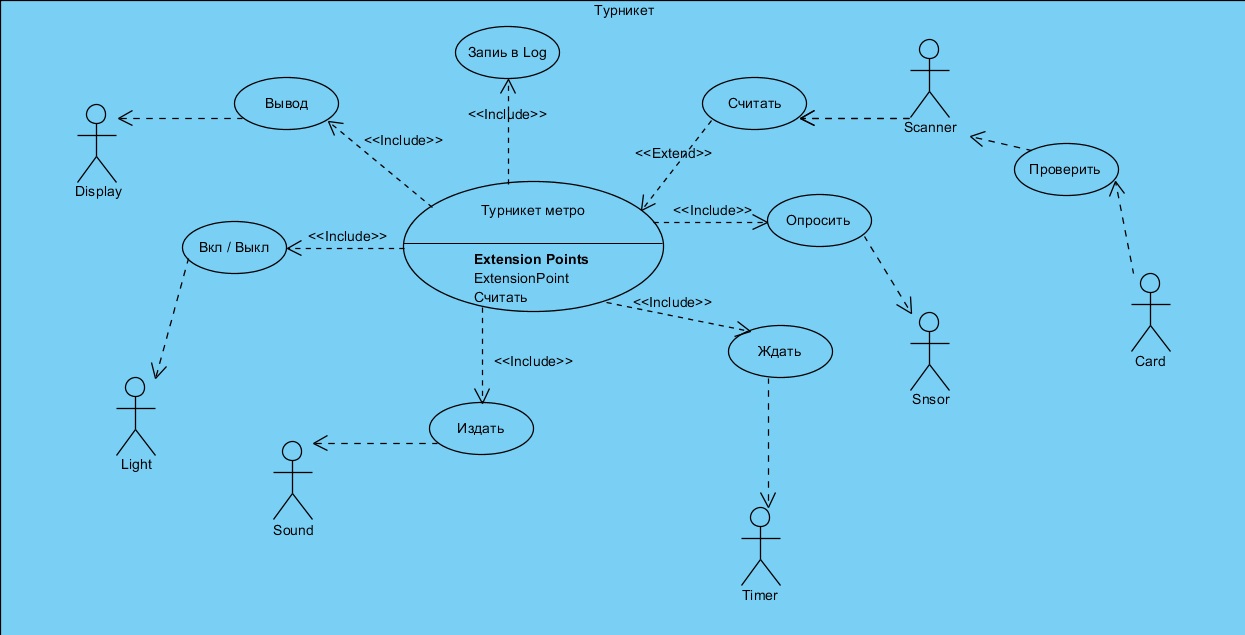
****

Рисунок 3.1 – Диаграмма вариантов использования

### 3.2.2 Диаграмма классов

Диаграмма классов (Class Diagram) служит для представления статической структуры модели систеы в терминах ООП. Диаграмма классов может отражать взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывать их внутреннюю структуру и типы отношений.

В качестве классов выделены актёры рассматриваемой системы – рисунок 3.2.

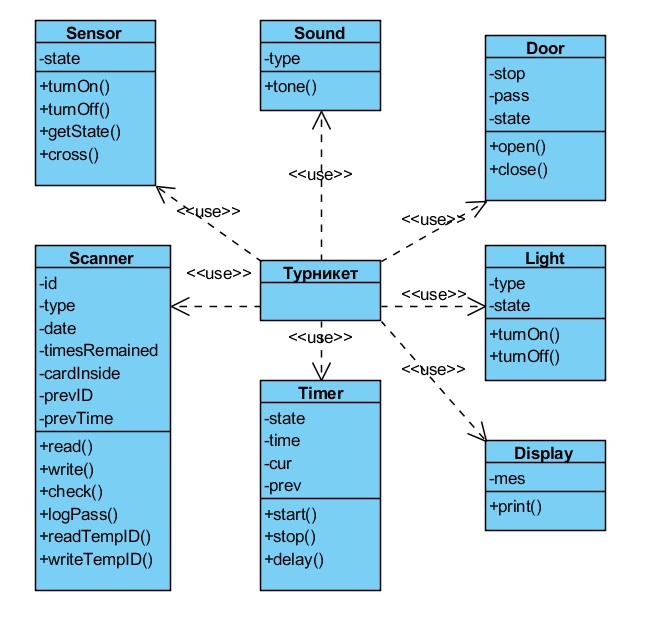


Рисунок 3.2 – Диаграмма классов

### 3.2.3 Диаграмма объектов

Диаграмма объектов строится на основе диаграммы классов, построенной ранее.

Данную диаграмму можно использовать для отображения одного из вариантов конфигурации объектов. Последний вариант очень полезен, когда допустимые связи между объектами могут быть сложными.

Объектам присваиваются уникальные имена, соответствующие своему назначению. При создании объектов инициализируются определённые свойства и выполняются определённые действия – рисунок 3.3.

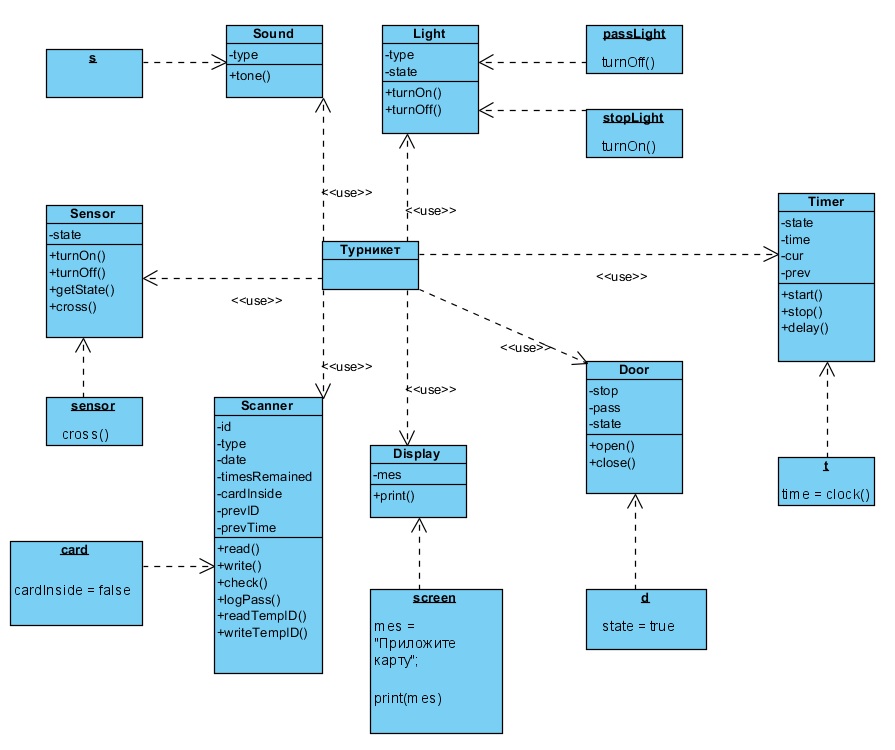


Рисунок 3.3 – Диаграмма объектов

### 3.2.4 Диаграмма состояний

Диаграмма состояний описывает всевозможные состояния системы и её частей во время работы. Переход от одного состояния к другому осуществляется при помощи специальных действий – методов, вызываемых в определённый момент времени выполнения программы.

Для данной предметной области было составлено 7 диаграмм приведенных на рисунках.

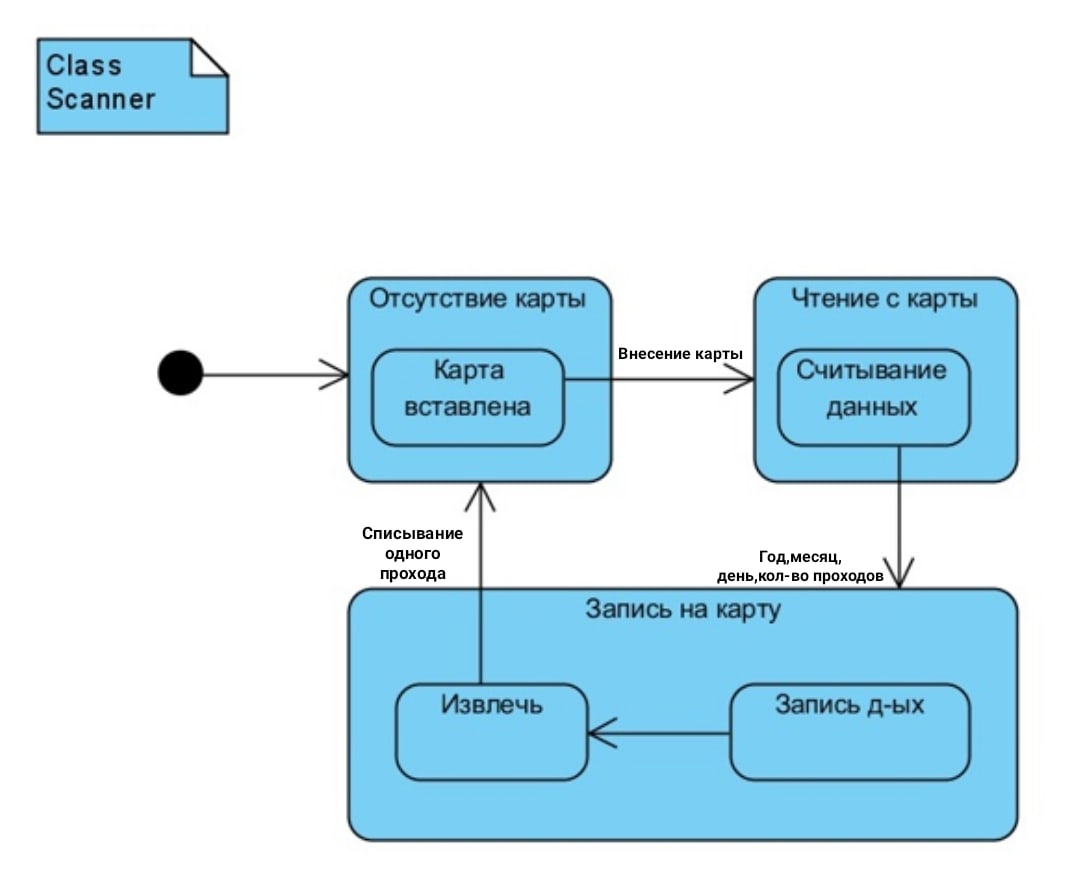


Рисунок 3.4 – Диаграмма состояний для класса Scanner (Считыватель карт)

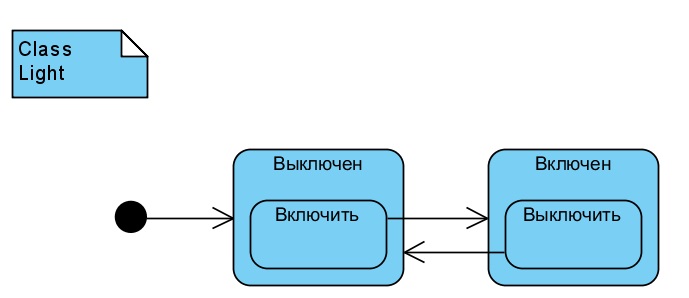


Рисунок 3.5 – Диаграмма состояний для класса Light (Индикаторы)

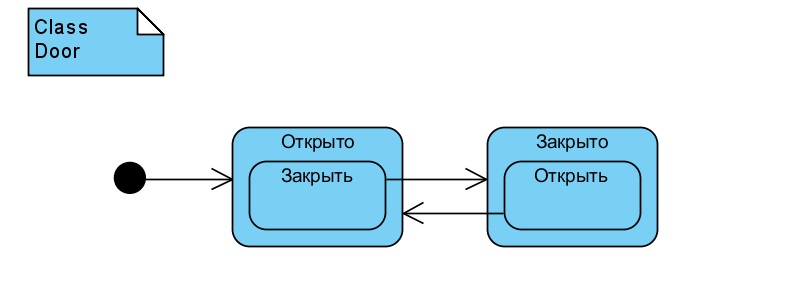


Рисунок 3.6 – Диаграмма состояний для класса Door (Устройство перекрытия доступа)

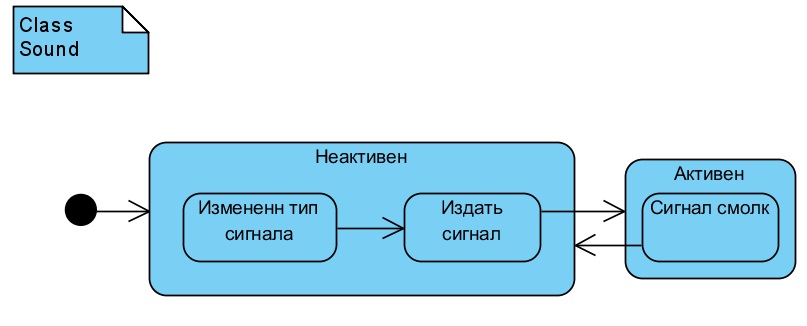


Рисунок 3.7 – Диаграмма состояний для класса Sound (Устройство подачи звукового сигнала)

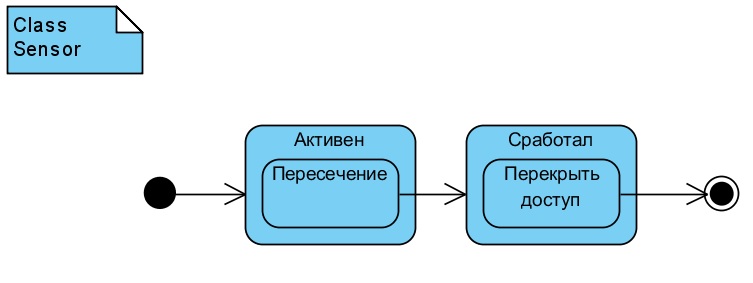


Рисунок 3.8 – Диаграмма состояний для класса Sensor (Датчик движения)

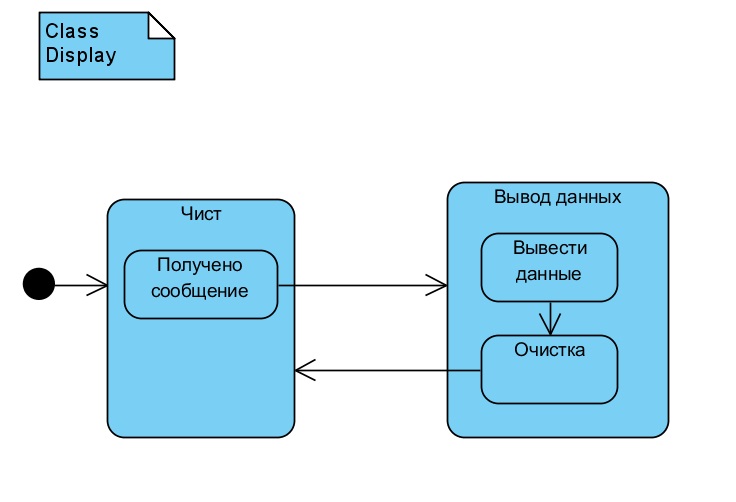


Рисунок 3.9 – Диаграмма состояний для класса Display (Индикатор количества оставшихся поездок)

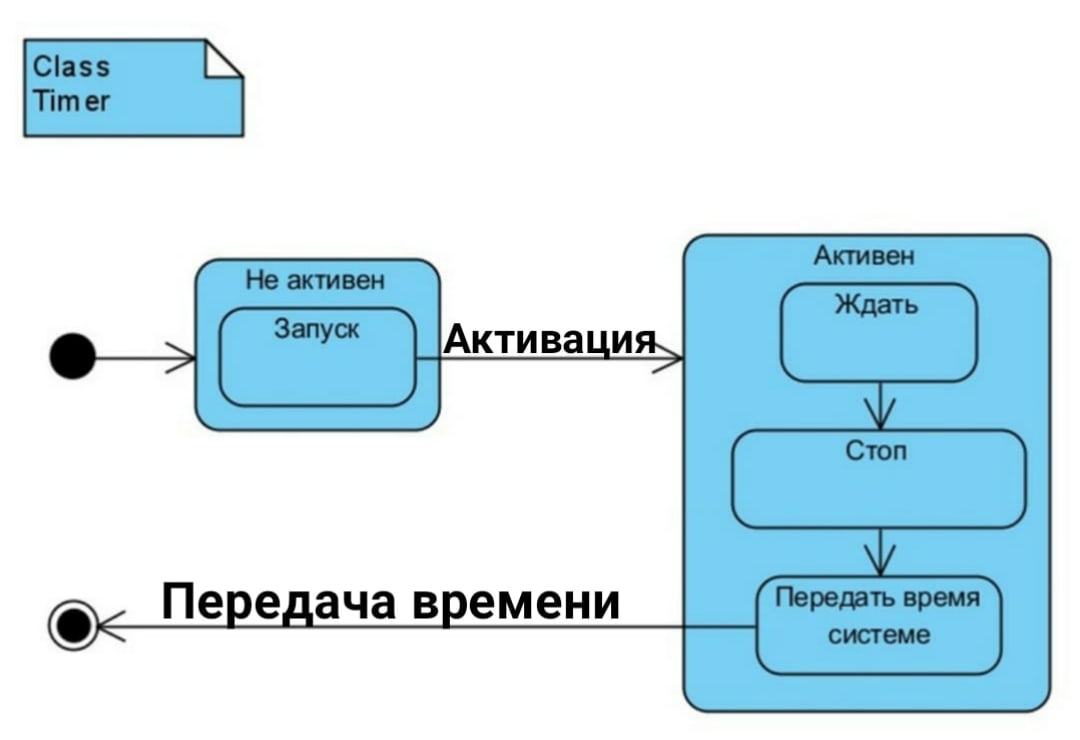


Рисунок 3.10 – Диаграмма состояний для класса Display (Индикатор количества оставшихся поездок)

### 3.2.5 Диаграмма активности

При моделировании поведения проектируемой системы возникает необходимость не только представить процесс изменения её состояний, но и детализировать особенности алгоритмической и логической реализации выполняемых системой операций.

Для этого используется диаграмма активности (Activity diagram) – она описывает алгоритмы вычислений или потоков управления в программных системах. Такая диаграмма изображена на рисунке 3.11.

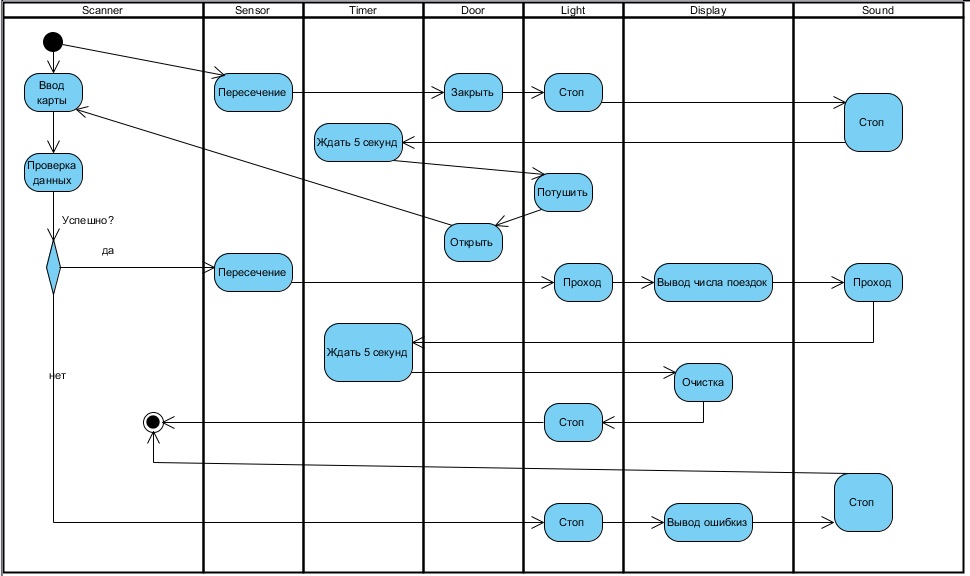


Рисунок 3.11 – Диаграмма активности

3.2.6 Диаграмма последовательности

Для представления временных особенностей передачи и приёма сообщений между объектами используется диаграмма последовательности. (Sequence diagram). На ней изображаются только те объекты, которые непосредственно участвуют во взаимодействии и не показываются возможные статические ассоциации с другими объектами – Рисунок 3.12.

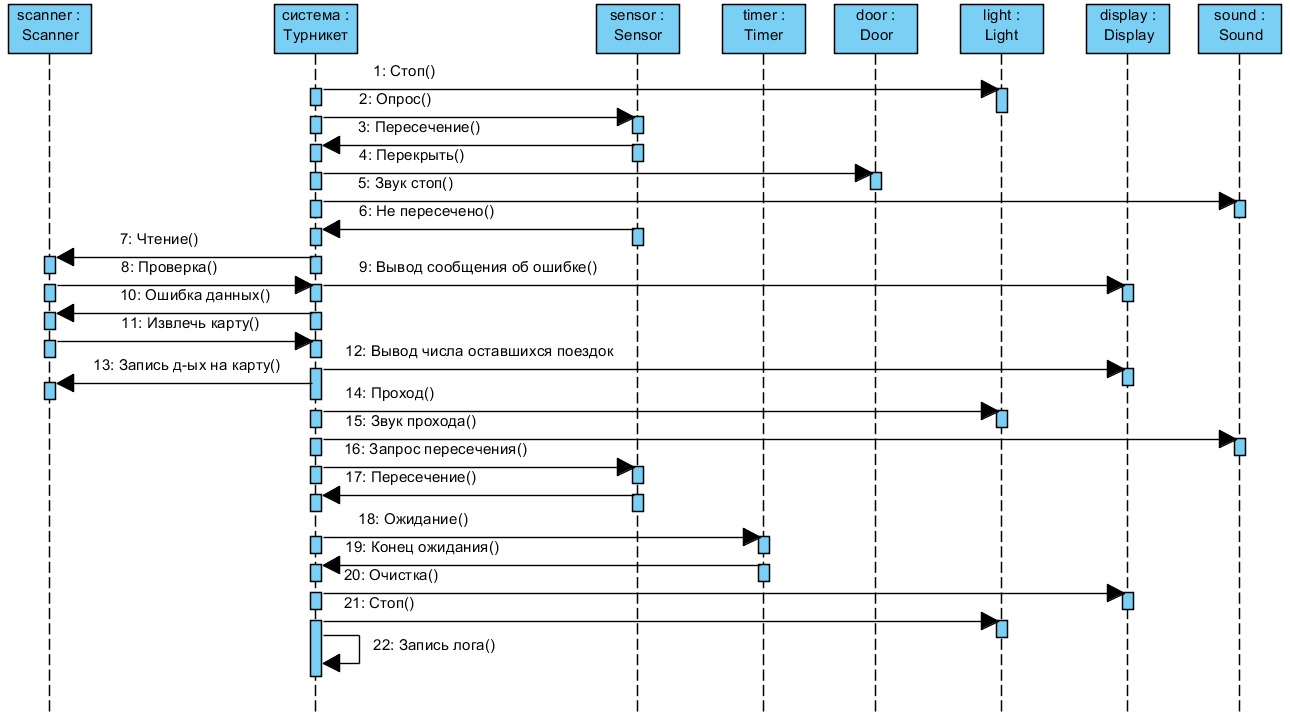


Рисунок 3.12 – Диаграмма последовательности

# 3.3 Объектно-ориентированное программирование

### 3.3.1 Диаграмма компонентов

Диаграмма компонентов описывает особенности физического представления системы. Она позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых выступает исходный, бинарный и исполняемый код – Рисунок 3.13.

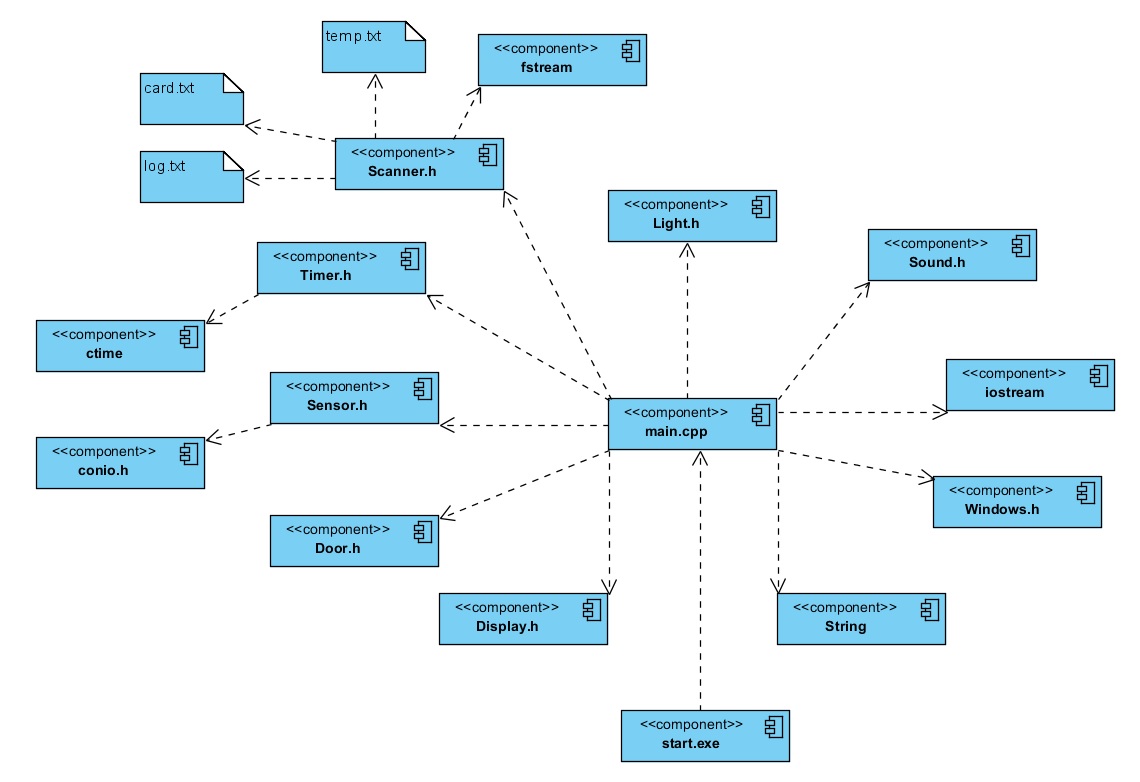


Рисунок 3.13 – Диаграмма компонентов

### 3.3.2 Диаграмма развёртывания

Диаграмма развёртывания (Deployment diagram) предназначена для визуализации элементов и компонентов программы, существующих лишь на этапе её выполнения. Она содержит графические изображения процессов, устройств и связей между ними – Рисунок 3.14.

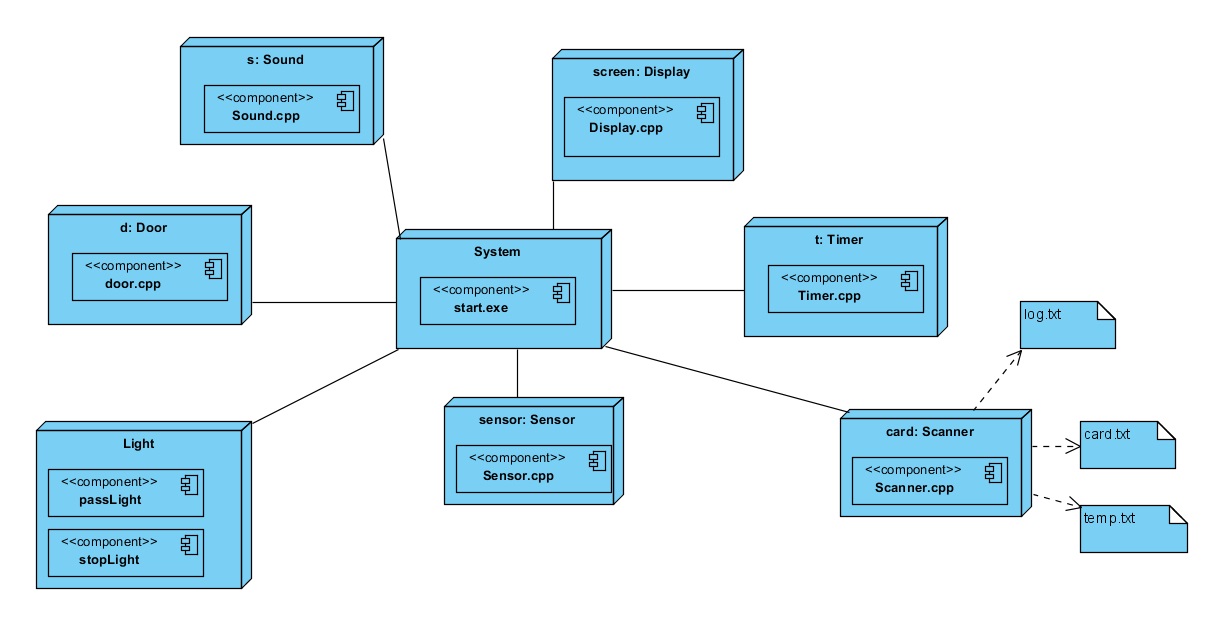


Рисунок 3.14 – Диаграмма развёртывания

### 3.3.3 Протоколы классов

Протокол класса определяет его допустимое поведение.В рамках предемнтной области было выделнно 7 классов (Timer, sound, display, door,

light, scanner, sensor).Протоколы этих классов приведенны в таблице 3.1.

Таблица 3.1 – Протоколы классов

|  |  |  |
| --- | --- | --- |
| Класс | Свойства | Методы |
| Timer | Public:  bool state; //состояние  unsigned int time; //время (пр-к)  unsigned int cur; //тек. время системы  unsigned int prev; //пред. время | Public:  void start(); //пуск таймера  void stop(); //стоп таймера  void delay(unsigned int millis);  //ф-ия ожидания (в миллисекундах) |
| Sound | Public:  char type; //тип | Public:  void tone(); //издать сигнал |
| Display | public:  string mes; //сообщение для вывода | Public:  void print(); //ф-ия вывода |
| Продолжение таблицы 3.1 | | |
| Класс | Свойства | Методы |
| Door | Public:  bool state; //состояние  string stop[5]; //"закрыто"  string pass[5]; //"открыто" | Public:  void open(); //открыть дверцы  void close(); //закрыть дверцы |
| Light | Public:  char type; //тип  private:  bool state; //состояние | Public:  void turnOn(); //вкл  void turnOff(); //выкл |
| Scanner | Public:  string id; //номер карты  char type; //тип  string date; //дата  char timesRemained[4]; //кол-во поездок  bool cardInside; //карта внутри?  string prevID; //врем. ID беск. карты  int prevTime; //время беск. карты | Public:  void read(); //чтение  void write(); //запись  bool check(); //проверка д-ых  void logPass(string s); //запись лога в файл  //чтение из врем. файла  void Scanner::readTempID(string &id, int &time);  //перезапись врем. файла  void Scanner::writeTempID(string id, int t); |
| Sensor | private:  string state; //состояние | public:  void turnOn(); //вкл  void turnOff(); //выкл  string getState(); //получить состояние  int cross(); //пересечь |

ЗАКЛЮЧЕНИЕ

В ходе работы была изучена предметная область «Турникет метрополитена», был произведен объектно-ориентированный анализ. На основании выделеных классов были построены UML-диаграммы (Диаграмма вариантов использования диаграмма классов диаграмма объектов диаграмма состояний диаграмма активности диаграмма последовательности диаграмма компонентов диаграмма развёртывания), характеризующие структуру системы и происходящие в ней процессы.

Результатом работы является рабочая программа, написанная на языке С++. Программа позволяет производить чтение и запись данных в файл, эмулирующий карту метрополитена. Программа имеет высокое быстродействие и надёжность. Данные на карте хранятся в определённом формате, обработка производится согласно постановке задачи.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Шлеер, С. Объектно-ориентированный анализ: моделирование мира в состояниях / С. Шлеер, Меллор С. – К.: Диалектика, 1993. – 240 с.
2. Фаулер, М. UML. Основы / М. Фаулер, К. Скотт – СПб.: Символ-Плюс, 2002. – 2-е изд. – 192 с.
3. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Дж. Рамбо, А. Якобсон. – 2007. – 2-е изд. – 496 с.
4. Якобсон, А. Унифицированный процесс разработки программного обеспечения / А. Якобсон, Г. Буч, Дж. Рамбо – СПб.: Питер, 2002. – 496 с.
5. Арлоу, Д. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование / Д. Арлоу, И. Нейштадт – 2007. – 2-е изд. – 624 с Издательство Символ-Плюс.
6. Кватрани, Т. Rational Rose 2000 и UML. Визуальное моделирование / Т. Кватрани – М.: ДМК Пресс, 2001. – 176 с.
7. Мацяшек, Л. Анализ требований и проектирование систем. Разработка информационных систем с использованием UML / Лешек А. Мацяшек. – М.: Вильямс, 2002. – 432 с.
8. Леоненков, А.В. Самоучитель UML / А.В. Леоненков. – СПб.: БХВ, 2004. – 2-е изд. – 158 с.
9. Леоненков, А.В. Язык UML 2 в анализе и проектировании программных систем и бизнес-процессов [Электронный ресурс]. – Режим доступа : http://www.intuit.ru/department/se/uml2/
10. Страуструп, Б. Язык программирования C++. Специальное издание / Бьерн Страуструп. – М.: Бином, 2008. – 3-е изд. – 1054 с.
11. Холзнер, С. Учебный курс Visual C++ 6 / С. Холзнер– СПб.: Питер, 2007. – 570 с.
12. Horton, I. Beginning Visual C++ 2008 / Ivor Horton. – Indianapolis: Wiley, 2008. – 1394 p.
13. Пахомов, Б.И. C/C++ и MS Visual C++ 2008 для начинающих / Б.И. Пахомов – СПб.: БХВ-Петербург, 2009. – 624 с.
14. Норкене, Е.А. Методические указания по оформлению студенческих работ / Е.А. Норкене, Н.П. Пулинец – Донецк: ДонНТУ, 2017. – 32 c.

ПРИЛОЖЕНИЕ А

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

А.1 Общие сведения

Тема курсового проекта: «Имитационное моделирование динамических систем и процессов с использованием объектно-ориентированного подхода. Программное обеспечение встроенного процессора турникета для метрополитена»

Система проектируется студентом 1-го курса Донецкого Национального Технического Университета, факультета КНТ, группы ПИ-19 г Федоровым Алексеем Дмитриевичем.

Основанием для разработки ПП является задание, выданное кафедрой ИИСА. Плановый срок начала работы по созданию системы имитационного моделирования: 5.02.2020 г., срок окончания: 21.05.2020 г. Курсовой проект должен выполняться согласно графику, приведенному в таблице А.1.

Таблица А.1 – Этапы, результаты и сроки разработки ПП

|  |  |  |  |
| --- | --- | --- | --- |
| № | Этапы работы | Результаты работы | Срок |
| 1 | Получение задания | План решения поставленной задачи | 06.02.2021 –  11.02.2021 |
| 2 | Оформление технического задания | Техническое задание (4–5 л.) | 12.02.2021 –  13.02.2021 |
| 3 | Постановка задачи | Описание предметной области. Формулировка задачи. Выбор средств реализации | 14.02.2021 –  28.02.2021 |
| 4 | Выбор метода решения | Описание исходных, выходных данных и ограничений | 28.03.2021 –  14.03.2021 |
| 5 | Выбор языка программирования | Выбор среды разработки | 14.03.2021 –  21.03.2021 |
| 6 | Построение диаграмм | Диаграмма классов, объектов, состояний и переходов, взаимодействий, модулей, процессов | 21.03.2021 –  05.04.2021 |

Продолжение таблицы А.1

|  |  |  |  |
| --- | --- | --- | --- |
| № | Этапы работы | Результаты работы | Срок |
| 7 | Разработка программного обеспечения | Написание кода. Создание формы полей и вывода данных | 06.04.2021 –  06.05.2021 |
| 8 | Тестирование | Результат тестирования | 04.05.2021 –  12.05.2021 |
| 9 | Оформление программного обеспечения по ГОСТу | Руководство пользователя. Описание алгоритма решения поставленной задачи | 13.05.2021 –  24.05.2021 |
| 10 | Защита |  | 25.05.2021 |

А.2 Назначения и цели создания программы

Данный программный продукт предназначен для имитации работы динамической системы реального мира – справочная служба кинотеатров города и получения статистических данных, таких как: скорость работы системы, надёжность, пропускная способность, универсальность для оптимизации работы подобных существующих систем.

А.3 Характеристика объекта автоматизации

При помощи турникета контролируется проход пассажиров в метро и взимается входная плата. Турникет имеет приемник карт, устройство для перекрывания доступа, таймер, три оптических датчика для определения прохода пассажира, устройство подачи звуковых сигналов, индикаторы **«**Проход**»** и **«**Стоп**»**, индикатор количества оставшихся поездок.

В начальном состоянии турникета зажжен индикатор **«**Стоп**»**, индикатор **«**Проход**»** потушен. Если один из датчиков посылает сигнал, то проход через турникет сразу же перекрывается, и подается предупредительный звуковой сигнал. Для прохода пассажир должен поместить карту в приемник карт. Каждая карта имеет срок годности, по истечении которого она не может быть использована для прохода. Карты бывают двух типов: с фиксированным количеством поездок и с неограниченным количеством поездок. Турникет считывает с карты данные: срок годности карты, номер карты, тип карты и количество поездок. Если данные не удается считать, или карта просрочена, или количество поездок нулевое, то карта возвращается пассажиру, и турникет остается в исходном состоянии. Иначе с карты с фиксированным количеством поездок списывается одна поездка, карта возвращается из приемника, индикатор **«**Стоп**»** гаснет, зажигается индикатор **«**Проход**»**, индикатор количества оставшихся поездок высвечивает текущее значение и пассажир может пройти через турникет. Получив от одного из датчиков сигнал, турникет ожидает время, отведенное на проход пассажира (5 секунд), после чего он возвращается в начальное состояние. Если карта имеет неограниченное количество поездок, то ее номер запоминается, чтобы в течение пяти минут после прохода пассажира с этой картой блокировать попытки прохода с ней через все турникеты данной станции метро. При проходе с картой такого типа индикатор количества оставшихся поездок высвечивает символ ∞ (бесконечность).

Наличие трех оптических датчиков в турникете гарантирует, что при проходе пассажира хотя бы один из них подаст сигнал (датчики невозможно перешагнуть, перепрыгнуть и т. д.). Во время прохода пассажира возможна ситуация, когда два или три датчика одновременно посылают сигналы. В этом случае принимается только первый сигнал и от момента его приема отсчитывается положенное время. Остальные сигналы игнорируются.

Турникет заносит в свою память время всех проходов. В конце рабочего дня он передает всю информацию, накопленную за день, в АСУ метрополитена.

А.4 Требования к программному продукту

А.4.1 Требования к системе в целом

В целом к системе предъявляются следующие требования:

а) имитация работы турникета метрополитена;

б) вывод статистики: номер карты, прошедшей через турникет, а также дата прохода в формате «ДД.ММ.ГГГГ ЧЧ:ММ:СС».

А.4.2 Требования к задачам и функциям программного продукта

В процессе работы необходимо обеспечить выполнение следующих функций:

1. ввод начальных параметров для моделирования;
2. вывод характеристик объектов на каждом шаге моделирования на экран;
3. вывод результатов моделирования на экран и в файл.

А.4.3 Требования к техническому обеспечению

К техническому обеспечению предъявляются следующие требования:

1. процессор – 32-битный x86-совместимый (уровня Pentium и выше);
2. объем оперативной памяти – не менее 32Мб;
3. свободное дисковое пространство – около 10 Мб. Не менее 5 Мб свободного дискового пространства для временных файлов;
4. графический адаптер – VGA-совместимый;
5. монитор –VGA-совместимый;
6. клавиатура.

А.4.4 Требования к программному обеспечению

Для стабильной работы к программному обеспечению предъявляются следующие требования:

* 1. обеспечить удобный и понятный пользовательский интерфейс;
  2. реализовать программу в виде отдельных классов;
  3. организовать защиту от некорректного ввода начальных параметров;
  4. обеспечить надежное хранение информации.

Программным обеспечением для проектирования программы является CASE-средство Visual Paradigm, а для разработки – Microsoft Visual Studio 2013. Для запуска программы необходимо наличие операционной системы Windows NT или старше и соответствующих библиотек Microsoft Visual C++ Redistributable.

А.4.5 Требования к организационному обеспечению

В программную документацию должны входить:

1. пояснительная записка;
2. приложения: техническое задание, руководство пользователя, экранные формы, листинг программы.

А.4.6 Требования к комплекту поставки ПП

Программный продукт поставляется с пояснительной запиской к курсовому проекту в файле ПЗ.doc, руководством пользователя в файле Help.doc, исходными кодами в виде проекта среды разработки в папке SRC, исполнительным файлом программы start.exe и файлом с данными о разработчике readme.txt.

# ПРИЛОЖЕНИЕ Б

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для корректной работы программы необходимо следовать следующим шагам:

1. запустите программу через файл start.exe;
2. следуя инструкциям выведенным на экране, выберите область с которой вам нужно работать и введите ее номер
3. после выбора выберите действие с данным типом объекта и введите его номер
4. чтобы вернуться или выйти из программы введите число 0
5. чтобы отменить ввод свойств введите число -1
6. чтобы выйти из программы следуйте инструкциям, иначе потеряются несохраненные данные
7. по истечению 5-ти секунд прозвучит сигнал «Стоп», на дисплее появится надпись «Заберите карту»;
8. извлеките файл card.txt из папки с проектом;
9. если карта не прошла проверку, появится надпись «Карта недействительна». Пользователю будет предложено вставить новую карту;
10. не пытайтесь пройти дважды с одной картой с бесконечным числом поездок – сработает блокировка данной карты на 5 минут.

# ПРИЛОЖЕНИЕ В

# ЭКРАННЫЕ ФОРМЫ

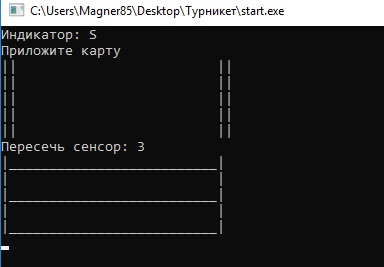


Рисунок В.1 - Пересечение сенсора и перекрытие доступа

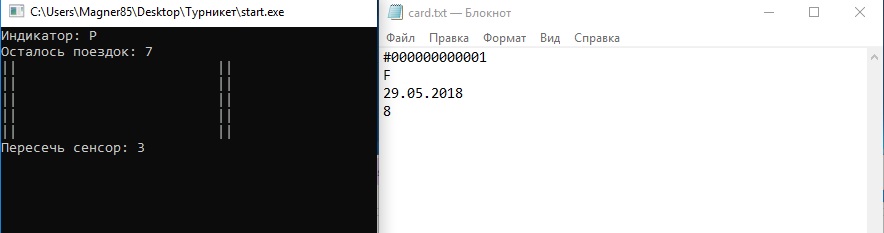


Рисунок В.2 - Проход (карта с фиксированным числом поездок)

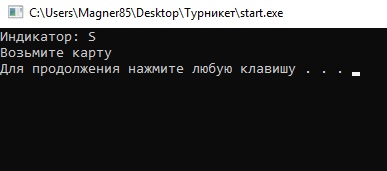


Рисунок В.3 – Операция успешна, возврат карты

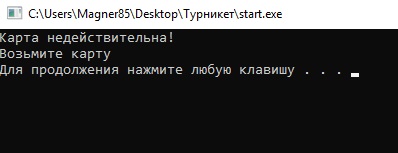


Рисунок В.4 - Карта не прошла проверку

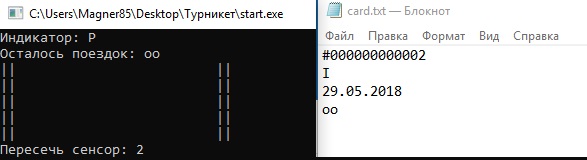


Рисунок В.5 - Проход (карта с неограниченным числом поездок)

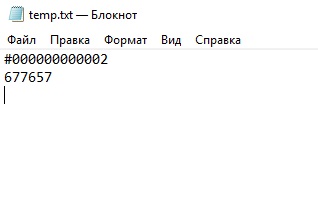


Рисунок В.6 – Временный файл

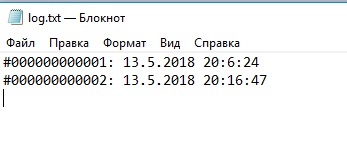


Рисунок В.7 - Отчёт обо всех проходах

# ПРИЛОЖЕНИЕ Г

# ЛИСТИНГ ПРОГРАММНЫХ МОДУЛЕЙ

Г.1 Файл main.cpp

1. #include <Windows.h>
2. #include <iostream>
3. #include <String>
4. #include «Display.h»
5. #include «Door.h»
6. #include «Light.h»
7. #include «Scanner.h»
8. #include «Sensor.h»
9. #include «Sound.h»
10. #include «Timer.h»
11. using namespace std;
12. int main(){
13. Scanner card; //считыватель (роль карты)
14. card.writeTempID(«00», 300000); //сброс врем. файла
15. while (true){
16. SetConsoleCP(1251);
17. SetConsoleOutputCP(1251);
18. Light passLight('P'); //индикатор стоп
19. Light stopLight('S'); //индикатор проход
20. stopLight.turnOn();
21. Display screen; //дисплей
22. Timer t; //таймер
23. Sound s; //звук. сигнал
24. Door d; //дверцы
25. d.open();
26. Sensor sensor; //датчики движения
27. int allow = sensor.cross(); //хранит пересечение датчиков
28. if (allow == 1 || allow == 2 || allow == 3){ //если пересечен хоть 1
29. d.close();
30. s.type = 'S';
31. s.tone();
32. t.delay(5000);
33. continue;
34. }
35. card.read(); //попытка чтения карты
36. system(«cls»);
37. if (card.check()){ //карта прочитана. Проверка данных
38. stopLight.turnOff();
39. passLight.turnOn();
40. card.write();
41. string str(card.timesRemained);
42. screen.mes = «Осталось поездок: « + str;
43. s.type = 'P';
44. s.tone();
45. screen.print();
46. d.open();
47. do{ //ожидание пересечения сенсоров
48. allow = sensor.cross();
49. } while (allow != 1 && allow != 2 && allow != 3);
50. t.delay(5000);
51. s.type = 'S';
52. s.tone();
53. passLight.turnOff();
54. stopLight.turnOn();
55. screen.mes = «Возьмите карту»;
56. screen.print();
57. }
58. else{ //данные неверные / не удалось считать
59. screen.mes = «Карта недействительна!»;
60. s.type = 'S';
61. s.tone();
62. screen.print();
63. screen.mes = «Возьмите карту»;
64. screen.print();
65. }
66. system(«pause»);
67. }
68. return 0;
69. }

Г.2 Файл timer.h

1. #include <iostream>
2. using namespace std;
3. class Timer{
4. public:
5. Timer(); //конструктор
6. bool state; //состояние
7. unsigned int time; //время (пр-к)
8. unsigned int cur; //тек. время системы
9. unsigned int prev; //пред. время
10. void start(); //пуск таймера
11. void stop(); //стоп таймера
12. void delay(unsigned int millis);
13. //ф-ия ожидания (в миллисекундах)
14. };

Г.3 Файл timer.cpp

1. #include «Timer.h»
2. #include <ctime>
3. Timer::Timer(){
4. state = false;
5. }//конструктор
6. void Timer::start(){
7. state = true;
8. prev = clock();
9. cur = prev;
10. }//пуск
11. void Timer::stop(){
12. state = false;
13. cur = clock();
14. time = cur - prev;
15. }//стоп
16. void Timer::delay(unsigned int millis){
17. start();
18. do{
19. cur = clock();
20. time = cur - prev;
21. } while (time < millis);
22. stop();
23. }//ф-ия ожидания

Г.4 Файл sound.h

1. #include <iostream>
2. using namespace std;
3. class Sound{
4. public:
5. char type; //тип
6. void tone(); //издать сигнал
7. };

Г.5 Файл sound.cpp

1. #include «Sound.h»
2. #include <Windows.h>
3. void Sound::tone(){ //подача сигнала
4. if (type == 'P') //тип проход
5. MessageBeep(MB\_COMPOSITE);
6. else if (type == 'S') //тип стоп
7. printf(«\a»);
8. }

Г.6 Файл sensor.h

1. #include <iostream>
2. using namespace std;
3. class Sensor{
4. public:
5. Sensor(); //Конструктор по умолчанию
6. void turnOn(); //вкл
7. void turnOff(); //выкл
8. string getState(); //получить состояние
9. int cross(); //пересечь
10. private:
11. string state; //состояние
12. };

Г.7 Файл sensor.cpp

1. #include «Sensor.h»
2. #include <conio.h>
3. void Sensor::turnOn(){
4. state = «Online»;
5. }//включить сенсор
6. void Sensor::turnOff(){
7. state = «Offline»;
8. }//откллючить
9. string Sensor::getState(){
10. return (\*this).state;
11. }//получить состояние
12. // ^ не используется
13. int Sensor::cross(){
14. cout << «Пересечь сенсор: «;
15. int a;
16. cin >> a;
17. return a;
18. }//пересечение датчика
19. Sensor::Sensor(){
20. (\*this).state = «Online»;
21. }//Конструктор по умолчанию

Г.8 Файл scanner.h

1. #include <iostream>
2. #include <time.h>
3. using namespace std;
4. class Scanner{
5. public:
6. Scanner(); //конструктор
7. string id; //номер карты
8. char type; //тип
9. string date; //дата
10. char timesRemained[4]; //кол-во поездок
11. bool cardInside; //карта внутри?
12. string prevID; //врем. ID беск. карты
13. int prevTime; //время беск. карты
14. void read(); //чтение
15. void write(); //запись
16. bool check(); //проверка д-ых
17. void logPass(string s); //запись лога в файл
18. //чтение виз рем. файла
19. void Scanner::readTempID(string &id, int &time);
20. //перезапись врем. файла
21. void Scanner::writeTempID(string id, int t);
22. };

Г.9 Файл scanner.cpp

1. #define \_CRT\_SECURE\_NO\_WARNINGS
2. #include «Scanner.h»
3. #include <fstream>
4. #include <string>
5. Scanner::Scanner(){
6. cardInside = false;
7. }//конструктор
8. void Scanner::logPass(string inner){
9. ofstream l;
10. l.open(«log.txt», ios::app);
11. l << inner;
12. l.close();
13. }//запись log`a
14. void Scanner::readTempID(string &id, int &time){
15. ifstream file;
16. file.open(«temp.txt»);
17. file >> id;
18. file >> time;
19. file.close();
20. }
21. void Scanner::writeTempID(string id, int t){
22. ofstream file;
23. file.open(«temp.txt»);
24. file << id << endl;
25. file << t << endl;
26. file.close();
27. }
28. void Scanner::read(){
29. FILE \*f;
30. do{ //ожидание ввода карты
31. f = fopen(«card.txt»,»r»);
32. } while (f == NULL);
33. fclose(f);
34. cardInside = true;
35. ifstream card(«card.txt»);
36. card >> id;
37. card >> type;
38. card >> date;
39. card >> timesRemained;
40. card.close();
41. }//чтение
42. void Scanner::write(){
43. if (type == 'F'){
44. ofstream card(«card.txt», ios\_base::out);
45. int num = atoi(timesRemained) - 1;
46. \_itoa(num, timesRemained, 10);
47. card << id << endl;
48. card << type << endl;
49. card << date << endl;
50. card << timesRemained;
51. card.close();
52. }//уменьшение числа поездок на фикс. карте
53. cardInside = false;
54. }//запись
55. bool Scanner::check(){
56. struct tm \*tim;
57. bool flag = false; //флаг правильности
58. time\_t tt = time(NULL);
59. //получение текущей даты
60. tim = localtime(&tt);
61. int theDay = tim->tm\_mday;
62. int theMonth = tim->tm\_mon+1;
63. int theYear = tim->tm\_year + 1900;
64. //формирование лога
65. string dateToday= id+»: «+to\_string(theDay)+».»+ to\_string(theMonth)+
66. «.»+to\_string(theYear)+» «+to\_string(tim->tm\_hour)+»:»+
67. to\_string(tim->tm\_min) + «:» + to\_string(tim->tm\_sec)+»\n»;
68. //обработка прочитанной даты
69. char dayEx[3], monthEx[3], yearEx[5];
70. char buf[11];
71. for (int i = 0; i < 10; i++){
72. buf[i] = date[i];
73. }
74. buf[10] = '\0';
75. char \*pch = strtok(buf, «.»);
76. strcpy(dayEx, pch);
77. pch = strtok(NULL, «.»);
78. strcpy(monthEx, pch);
79. pch = strtok(NULL, «.»);
80. strcpy(yearEx, pch);
81. readTempID(prevID, prevTime); //чтение из врем. файла
82. int timeInf = clock() - prevTime;
83. if (prevID == id && type == 'I' && timeInf < 300000){ //проверка беск. карты
84. return false; //^ 5минут
85. }
86. if (atoi(yearEx) > theYear){
87. flag = true;
88. }
89. else if (atoi(yearEx) == theYear){
90. if (atoi(monthEx) > theMonth)
91. flag = true;
92. else if (atoi(monthEx) == theMonth){
93. if (atoi(dayEx) >= theDay){
94. flag = true;
95. }
96. }
97. }
98. else flag = false;
99. if (flag){
100. if (type == 'I'){
101. logPass(dateToday); //запись лога
102. strcpy(timesRemained, «oo»); //на всякий случай
103. prevTime = clock(); //обновить время для беск. карты
104. prevID = id;
105. writeTempID(prevID, prevTime); //перезапись врем. файла
106. return true;
107. }//для бесконечной карты
108. else if (type == 'F' && atoi(timesRemained) > 0){
109. logPass(dateToday); //запись лога
110. return true; //для фикс. карты
111. }
112. }
113. return false;
114. }//проверка данных

Г.10 Файл light.h

1. #include <iostream>
2. using namespace std;
3. class Light{
4. public:
5. Light(char type); //констр-р с парам-ом
6. char type; //тип
7. void turnOn(); //вкл
8. void turnOff(); //выкл
9. private:
10. bool state; //состояние
11. };

Г.11 Файл light.cpp

1. #include «Light.h»
2. void Light::turnOn(){ //зажечь
3. state = true;
4. system(«cls»);
5. cout << «Индикатор: « << type << endl;
6. }
7. void Light::turnOff(){ //потушить
8. state = false;
9. }
10. Light::Light(char type){
11. if (type == 'P'){
12. (\*this).type = type;
13. (\*this).turnOn();
14. }
15. else if (type == 'S'){
16. (\*this).type = type;
17. (\*this).turnOn();
18. }
19. }//конструктор с параметром

Г.12 Файл door.h

1. #include <iostream>
2. #include <string>
3. using namespace std;
4. class Door{
5. public:
6. Door(); //конструктор по умолчанию
7. void open(); //открыть дверцы
8. void close(); //закрыть дверцы
9. bool state; //состояние
10. string stop[5]; //»закрыто»
11. string pass[5]; //»открыто»
12. };

Г.13 Файл door.cpp

1. #include «Door.h»
2. Door::Door(){
3. state = true;
4. stop[0] = «|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|»;
5. stop[1] = «| |»;
6. stop[2] = «|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|»;
7. stop[3] = «| |»;
8. stop[4] = «|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|»;
9. //»закрытая» дверь
10. pass[0] = «|| ||»;
11. pass[1] = «|| ||»;
12. pass[2] = «|| ||»;
13. pass[3] = «|| ||»;
14. pass[4] = «|| ||»;
15. //»открытая» дверь
16. } //конструктор по умолчанию
17. void Door::open(){
18. state = true;
19. for (int i = 0; i < 5; i++){
20. cout << pass[i] << endl;
21. }
22. }//открытие
23. void Door::close(){
24. state = false;
25. for (int i = 0; i < 5; i++){
26. cout << stop[i] << endl;
27. }
28. }//закрытие

Г.14 Файл display.h

1. #include <iostream>
2. using namespace std;
3. class Display{
4. public:
5. Display(); //конструктор
6. string mes; //сообщение для вывода
7. void print(); //ф-ия вывода
8. };

Г.15 Файл display.cpp

1. #include «Display.h»
2. #include <string>
3. Display::Display(){ //конструктор
4. mes = «Приложите карту»;
5. print();
6. }
7. void Display::print(){ //ф-ия вывода
8. cout << mes << endl;
9. }