**Содержание**

[Реферат 4](#_Toc516851513)

[Введение 5](#_Toc516851514)

[1. Теоретическое введение 7](#_Toc516851515)

[1.1. Объектно-ориентированное программирование 7](#_Toc516851516)

[1.2. Унифицированный язык моделирования 9](#_Toc516851517)

[2. Проектная часть 11](#_Toc516851518)

[2.1. Диаграмма вариантов использований 11](#_Toc516851519)

[2.2. Диаграмма последовательности действий 14](#_Toc516851520)

[2.3. Диаграмма классов 15](#_Toc516851521)

[3. Экспериментальная часть 18](#_Toc516851522)

[3.1. Тестирование 18](#_Toc516851523)

[3.2. Инструкция для пользователя 24](#_Toc516851524)

[Заключение 26](#_Toc516851525)

[Список литературы 26](#_Toc516851526)

[Приложения 27](#_Toc516851527)

[Приложение 1. Исходный код основной программы 27](#_Toc516851528)

[Приложение 2. Исходный код файла User.cpp 27](#_Toc516851529)

[Приложение 3. Исходный код заголовочного файла User 28](#_Toc516851530)

[Приложение 4. Исходный код файла ATM.cpp 29](#_Toc516851531)

[Приложение 4. Исходный код заголовочного файла ATM.h 35](#_Toc516851532)

**Реферат**

Курсовая работа состоит из 36 страниц. Работа включает в себя три главных раздела – теоретической, экспериментальной и проектной и содержит графический материал: 6 таблиц и 16 рисунков.

Раздел с теоретическими сведениями содержит сжатую информацию об объектно-ориентированном программировании. Приведены основные положения объектно-ориентированной парадигмы. Отличительные черты объектного-программирования: инкапсуляция, полиморфизм, объектная модель, наследование.

В проектной части были спроектированы, согласно заданию, классы.

Приведены диаграммы вариантов использований, диаграмма последовательности действий, диаграмма классов.

Экспериментальная часть включает описание тестирования. Результаты представлены в виде снимков экрана, подтверждающих правильную работу методов классов.

В приложении приведен исходный код приложения.

**Введение**

Как мы видим окружающий мир? Ответ зависит от нашего прошлого. Ученый может видеть мир как множество молекулярных структур. Художник видит мир как набор форм и красок. А кто-то может сказать, что мир – это множество вещей. Вероятно, первая мысль, которая пришла в голову, была: – какое значение имеет то, как кто-то видит мир? Однако это имеет большое значение для программиста, которому необходимо написать программу, моделирующую реальный мир.

Все мы видим мир как составляющие его вещи. Дом – вещь. Имущество, которое находится в доме, – вещи. То, что выкидываете, – вещи, покупаете вещи. Вещи, которые мы имеем, например, дом сделаны из других вещей, таких как окна или двери.

Технически говоря, любая вещь – это объект. То есть, дом – объект. Вещи, которые находятся в доме, – объекты. Вещи, которые вы выбрасываете, – объекты, и вещи, которые вы хотите ку­пить, – тоже объекты. Все мы, независимо от нашего прошлого, видим окружающий мир как множество объектов. Объект – это: человек, место, вещь, понятие, процесс и, возможно, событие.

Если проанализировать, каким образом человек решает разнообразные практические задачи в окружающем его мире, то можно понять, что этот мир также является объектно-ориентированным. Например, чтобы попасть на работу, человек, как правило, взаимодействует с таким объектом, как транспортное средство. Транспортное средство, в свою очередь, состоит из объектов, которые, взаимодействуя друг с другом, приводят его в движение, благодаря чему чело­век реализует свою задачу – добирается до нужного пункта. При этом ни водитель, ни пассажир не обязаны знать, каким образом взаимодействуют объекты, из которых состоит транспортное средст­во.

В объектно-ориентированной технологии, как и в реальном мире, пользователи про­грамм изолированы от логической схемы, необходимой для выполнения задач. Например, для пе­чати страницы в текстовом редакторе пользователь вызывает определенную функцию нажатием кнопки на панели инструментов, но не видит происходящих при этом внутренних процессов. При печати страницы во время работы программы происходит сложное взаимодействие объектов, ко­торые, в свою очередь, взаимодействуют с принтером.

1. **Теоретическое введение**
   1. **Объектно-ориентированное программирование**

Объектно-ориентированное программирование (ООП) – это программирование, сфокусированное на данных, причем данные и поведение неразрывно связаны между собой. Вместе данные и поведение представляют собой класс, а объекты являются экземплярами класса.

Составными частями объектно-ориентированного подхода являются:

* Инкапсуляция и расширяемость типов.
* Наследование.
* Полиморфизм.
* Перегрузка.
* Виртуальные функции.

В центре Объектно-ориентированное программирование находится понятие объекта. Объект — это сущность, которой можно посылать сообщения и которая может на них реагировать, используя свои данные. Объект — это экземпляр класса. Данные объекта скрыты от остальной программы. Сокрытие данных называется инкапсуляцией.

Наличие инкапсуляции достаточно для объектности языка программирования, но ещё не означает его объектной ориентированности — для этого требуется наличие наследования.

Но даже наличие инкапсуляции и наследования не делает язык программирования в полной мере объектным с точки зрения Объектно-ориентированное программирование. Основные преимущества Объектно-ориентированное программирование проявляются только в том случае, когда в языке программирования реализован полиморфизм подтипов — возможность единообразно обрабатывать объекты с различной реализацией при условии наличия общего интерфейса.

Объектно-ориентированное программирование позволяет легко создавать и использовать АТД (абстрактные типы данных). Объектно-ориентированное программирование использует механизм наследования. Наследование выгодно тем, что позволяет получать производные типы из уже определенных пользователем типов данных.

Объектно-ориентированное программирование имеет уже более чем сорокалетнюю историю, но, несмотря на это, до сих пор не существует чёткого общепринятого определения данной технологии. Основные принципы, заложенные в первые объектные языки и системы, подверглись существенному изменению (или искажению) и дополнению при многочисленных реализациях последующего времени. Кроме того, примерно с середины 1980-х годов термин «объектно-ориентированный» стал модным, в результате с ним произошло то же самое, что несколько раньше с термином «структурный» (ставшим модным после распространения технологии структурного программирования) — его стали искусственно «прикреплять» к любым новым разработкам, чтобы обеспечить им привлекательность. Бьёрн Страуструп в 1988 году писал, что обоснование «объектной ориентированности» чего-либо, в большинстве случаев, сводится к некорректному силлогизму: «X — это хорошо. Объектная ориентированность — это хорошо. Следовательно, X является объектно-ориентированным».

Появление в Объектно-ориентированное программирование отдельного понятия класса закономерно вытекает из желания иметь множество объектов со сходным поведением. Класс в Объектно-ориентированное программирование — это в чистом виде абстрактный тип данных, создаваемый программистом. С этой точки зрения объекты являются значениями данного абстрактного типа, а определение класса задаёт внутреннюю структуру значений и набор операций, которые над этими значениями могут быть выполнены. Желательность иерархии классов (а значит, наследования) вытекает из требований к повторному использованию кода — если несколько классов имеют сходное поведение, нет смысла дублировать их описание, лучше выделить общую часть в общий родительский класс, а в описании самих этих классов оставить только различающиеся элементы.

* 1. **Унифицированный язык моделирования**

Унифицированный язык моделирования (от англ. Unified Modeling Language) – язык графического описания для объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

UML является языком широкого профиля, это – открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML-моделью. UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. UML не является языком программирования, но на основании UML-моделей возможна генерация кода.

Все диаграммы UML можно условно разбить на две группы, первая из которых ‒ общие диаграммы. Общие диаграммы практически не зависят от предмета моделирования и могут применяться в любом программном проекте без оглядки на предметную область, область решений и т.д.

Существует два вида:

Статический вид диаграммы рассматривает логические взаимосвязи классов между собой;

Аналитический вид диаграммы рассматривает общий вид и взаимосвязи классов, входящих в систему.

Существуют разные точки зрения на построение диаграмм классов в зависимости от целей их применения:

Концептуальная точка зрения – диаграмма классов описывает модель предметной области, в ней присутствуют только классы прикладных объектов;

Точка зрения спецификации – диаграмма классов применяется при проектировании информационных систем;

Точка зрения реализации – диаграмма классов содержит классы, используемые непосредственно в программном коде (при использовании объектно-ориентированных языков программирования).

1. **Проектная часть**
   1. **Диаграмма вариантов использований**

Диаграмма вариантов использования указана на рисунке 1.

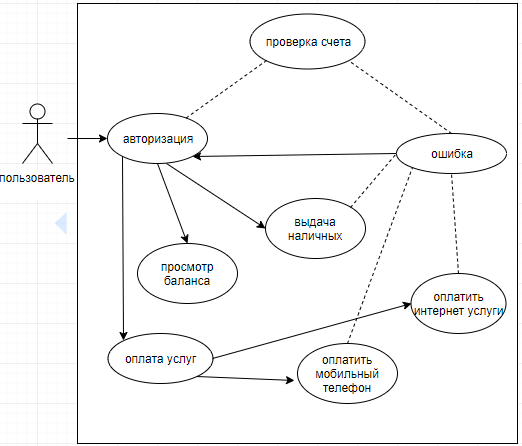


Рисунок 1. Диаграмма вариантов использования.

Таблица 1. Диаграмма вариантов использования (авторизация).

|  |  |
| --- | --- |
| Название варианта | Авторизация |
| Цель | Авторизоваться в системе |
| Действующие лица | Пользователь |
| Краткое описание | Пользователь входит в систему для дальнейшней эксплуатации, если введены неверные данные,то авторизация не произойдет |
| Тип варианта | Основной |

Таблица 2. Диаграмма вариантов использования (просмотр баланса).

|  |  |
| --- | --- |
| Название Варианта | Просмотр баланса |
| Цель | Узнать текущий баланс пользователя |
| Действующие лица | Пользователь |
| Краткое описание | Система предоставляет узнать счет пользователя |
| Тип варианта | Основной |

Таблица 3. Диаграмма вариантов использования (выдача наличных).

|  |  |
| --- | --- |
| Название Варианта | Выдача наличных |
| Цель | Снять средства |
| Действующие лица | Пользователь |
| Краткое описание | Пользователь снимает наличные со счета |
| Тип варианта | Основной |

Таблица 4. Диаграмма вариантов использования (оплата услуг).

|  |  |
| --- | --- |
| Название Варианта | Оплата услуг |
| Цель | Оплата мобильного телефона и интернета |
| Действующие лица | Пользователь |
| Краткое описание | Пользователь оплачивает услуги на свое усмотрение |
| Тип варианта | дополнительный |

Таблица 5. Диаграмма вариантов использования (пополнение мобильного телефона).

|  |  |
| --- | --- |
| Название Варианта | Пополнение мобильного телефона |
| Цель | Произвести оплату мобильного телефона |
| Действующие лица | Пользователь |
| Краткое описание | Пользователь вводит номер мобильного телефона и сумму для пополнения |
| Тип варианта | Основной |

Таблица 6. Диаграмма вариантов использования (пополнение интернет услуг).

|  |  |
| --- | --- |
| Название Варианта | Пополнение интернет услуг |
| Цель | Пополнить оплату интернет услуг |
| Действующие лица | Пользователь |
| Краткое описание | Пользователь вводит номер интернет счета и сумму для пополнения |
| Тип варианта | Основной |

* 1. **Диаграмма последовательности действий**

Диаграмма последовательности действий (от англ. sequence diagram) – это вид диаграммы взаимодействия, в котором внимание акцентируется на временной упорядоченности сообщений во времени. С помощью диаграмм последовательности действий удобно моделировать простые потоки управления, не содержащие сложных ветвлений и циклов.

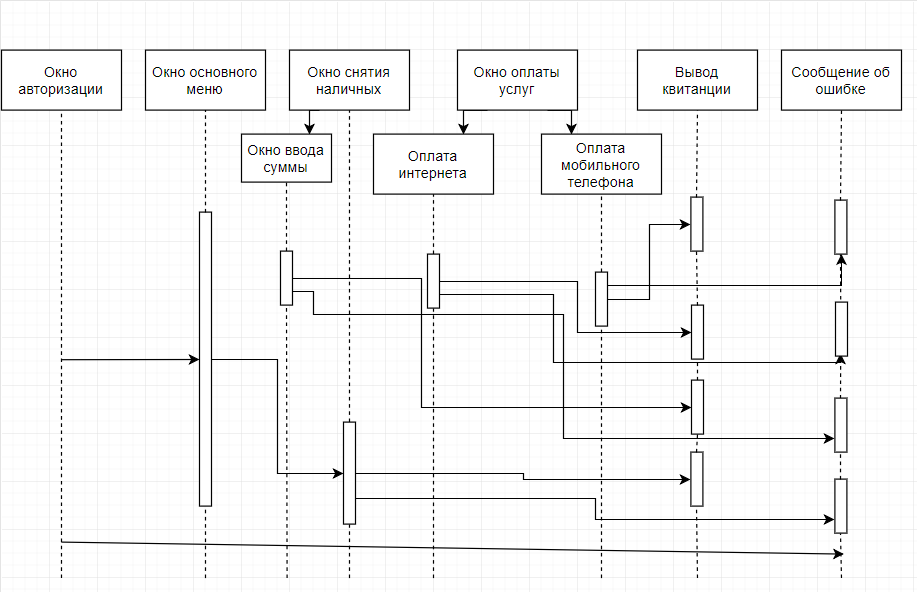


Рисунок 2. Диаграмма последовательности действий

* 1. **Диаграмма классов**

Для выполнения задачи я создал класс пользователя (User) и класс банкомата (ATM).

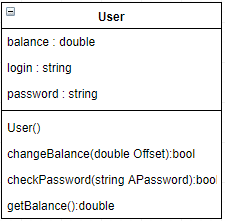


Рисунок 3. Диаграмма класса User.

Класс User обладает 3 полями : balance, login, password.

Также имеет 2 конструктора User(), User(string ALogin, string APassword, double ABalance), 3 метода ChangeBalance(double Offset), CheckPassword(string APassword), GetBalance().

Два конструктора отвечают за определение переменной в классе.

Метод ChangeBalance(double Offset) контролирует сумма средств запрошенных для снятия со счета, а именно если сумма не будет превышать текущий баланс, то баланс будет изменен в соответствии с запрошенными данными.

Метод CheckPassword(string Apassword) отвечает за проверку введенного и считанного с файла пароля.

Метод GetBalance() возвращает баланс текущего пользователя.

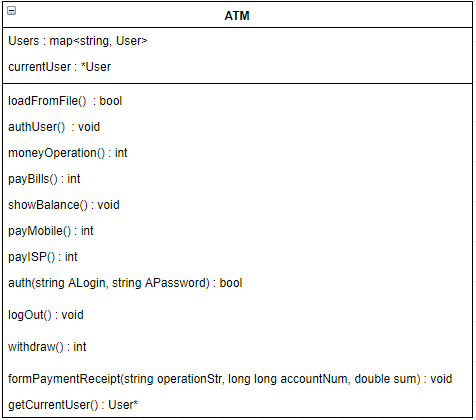


Рисунок 4. Диаграмма класса ATM.

Класс ATM имеет 2 поля: User\* currentUser и map<string, User> Users.

Также обладает 12 методами : LoadFromFile(), AuthUser(), MoneyOperation(), PayBills(), ShowBalance(), PayMobile(), PayISP(), Auth(string ALogin, string APassword), LogOut(), Withdraw().

Метод LoadFromFile() выполняет загрузку из файла(файл с данными пользователя).

Метод AuthUser() выполняет авторизацию пользователя.

Метод MoneyOperation() выполняет снятие средств с баланса.

Метод PayBills() вызывает меню оплаты услуг.

Метод PayMobile() выполняет денежное пополнение мобильного телефона.

Метод PayISP() выполняет денежное пополнение интернет услуги.

Метод Auth(string ALogin, string APassword) выполняет проверку введенных данных пользователя(логин и пароль).

Метод LogOut() выполняет выход из авторизовавшегося аккаунта.

Метод Withdraw() вызывает метод MoneyOperation().

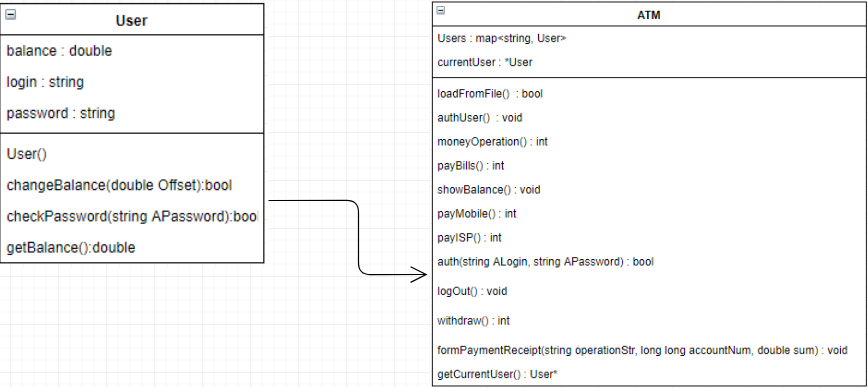
Ниже, на рисунке 5 представлена диаграмма реализованных классов.

Рисунок 5. Диаграмма классов.

1. **Экспериментальная часть**
   1. **Тестирование**

Тестирование программы необходимо производить для того чтобы найти ошибки, проверить полную функциональность и уже затем предоставить программу в пользование.

Пользователь вводит данные (логин и пароль), записанные в файле, как показано на рисунке 6.

Рисунок 6. Удачная авторизация

При неправильном вводе авторизации не будет, как показано на рисунке 7.



Рисунок 7. Неудачная авторизация.

Перед пользователем появляется меню выбора, показанном на рисунке 8.



Рисунок 8. Меню выбора.

В случае выбора пользователем 2й команды – Withdraw, произойдет снятие наличных со счета (необходимо ввести сумму), что показано на рисунке 9.



Рисунок 9. Снятие наличных.

Пользователь получит квитанцию операции.

При некорректном вводе пользователь получит уведомление, показанное на рисунке 10.

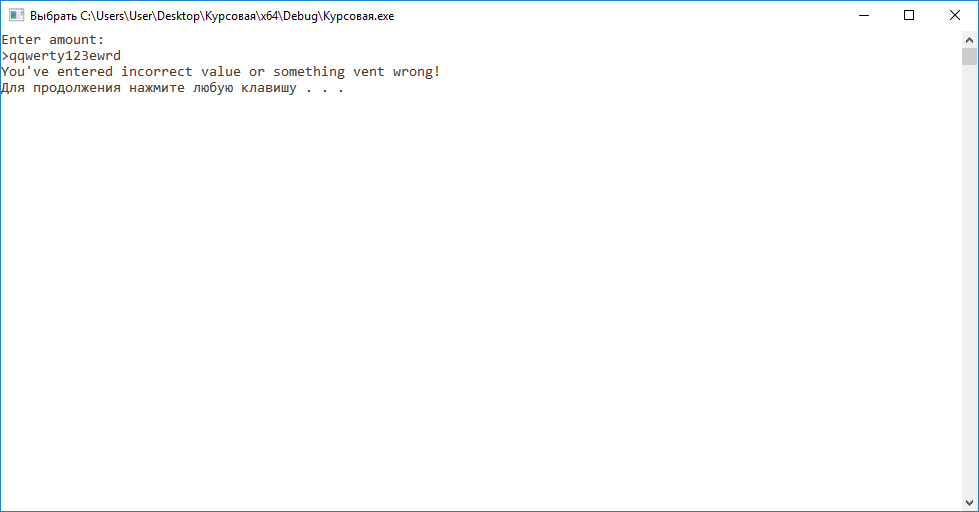


Рисунок 10. Некорректный ввод.

Если запрашиваемая сумма превышает баланс, то перед пользователем появляется сообщение с уведомлением, показанное на рисунке 11.

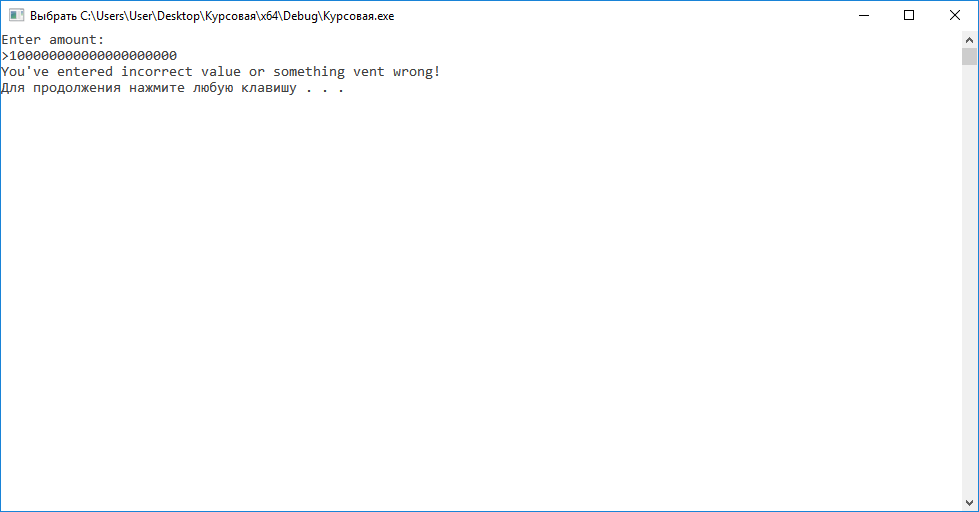


Рисунок 11. Сумма превышает баланс.

Пользователь может узнать свой баланс, что показано на рисунке 12.



Рисунок 12. Баланс пользователя.

Пользователь может оплачивать услуги интернета и мобильного телефона, как на рисунке 13.

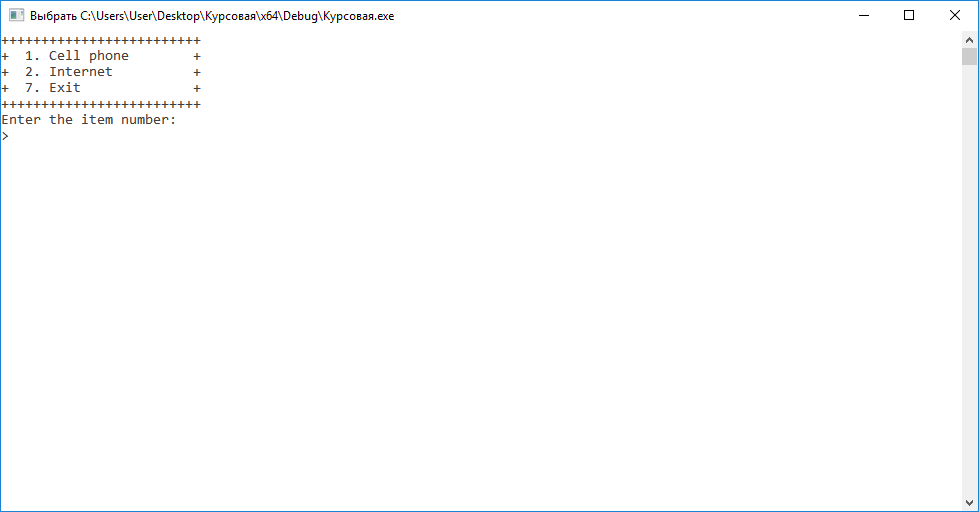


Рисунок 13. Оплата услуги интернета и мобильного телефона.

Оплата мобильного телефона требует ввод номер, что показано на рисунке 14.



Рисунок 14. Оплата мобильного телефона.

Окно ввода сумы пополнения представлено на рисунке 15.

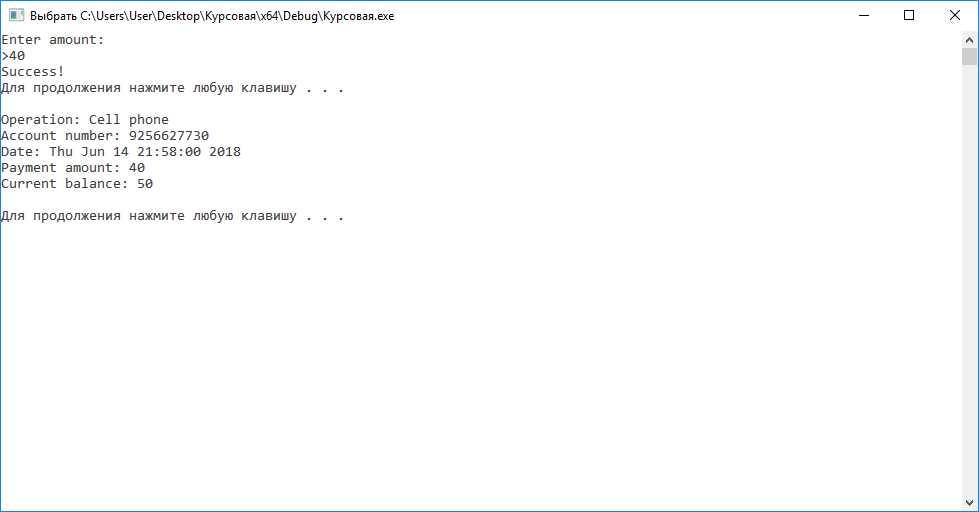


Рисунок 15. Окно успешной оплаты мобильного телефона.

Оплата интернета требует ввода номера аккаунта и суммы, как показано на рисунках 16, 17.



Рисунок 16. Ввод номера аккаунта.

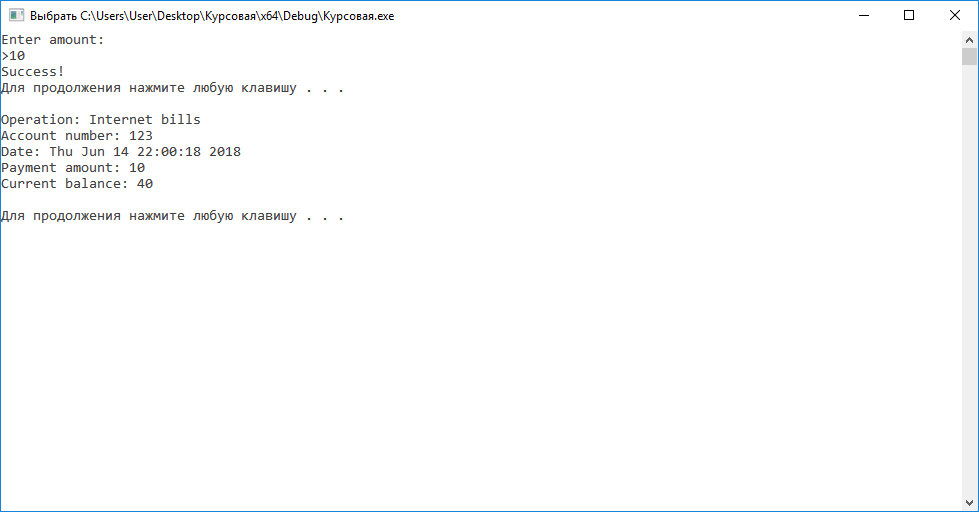


Рисунок 17. Ввод суммы пополнения и получение квитанции с номером аккаунта.

Пользователь может выйти со своего счета с помощью команды под номером 7.

* 1. **Инструкция для пользователя**

Последовательность действий:

1. Авторизоваться.
2. Выбрать желаемую операцию.
3. Совершить выбранную операцию
4. Получить квитанцию.

Подробное описание действий для пользователя:

1. Пользователь запускает программу и перед появляется меню с двумя выборами: авторизация и выход из программы. Пользователь вводит свои данные, записанные в отдельном файле, а именно логин и пароль. Если пользователь вводит неправильный пароль или логин, то получает уведомление о неуспешной авторизации.
2. Перед пользователем появляется консольное меню с выбором операций. Снятие наличных требует от пользователя указать сумму для проведения операции, в случае не хватки денежных средств пользователь получит оповещение о проблеме и операция не будет выполнена. Просмотр баланса позволяет пользователю увидеть баланс его счета. Операция оплата услуг переводит пользователя к другому меню с двумя командами: оплата интернет услуги и мобильного телефона. Для оплаты интернет услуги пользователь должен указать номер аккаунта и сумму для пополнения, если текущий баланс пользователя меньше суммы пополнения операция будет не выполнена. Для оплаты мобильного телефона пользователь должен ввести номер телефона и количество средств для пополнения, если текущий баланс пользователя меньше суммы пополнения операция будет не выполнена.
3. При каждой успешной транзакции пользователь будет получать квитанцию, которая включает в себя время транзакции, количество потраченных средств и остаток своего баланса.

**Заключение**

В выполненной курсовой работе было разработано приложение для работы с банкомат. Банкомат имеет функцию авторизации с проверкой пароля и выдачи ошибки при неправильной авторизации. Было сделано меню с выбором оплаты услуг интернета или мобильной связи. Добавлена поддержка баланса пользователя с возможностью снятия средств со счёта.

Программа реализована на языке программирования C++, с использованием объектно-ориентированного программирования, а именно использованием классов, инкапсуляции и полиморфизмов.

Был проведён анализ предметной области и применены на полученные знания. Поставленная задача была выполнена полностью.

**Список литературы**

1. Бьёрн Страуструп «Программирование. Принципы и практика с использованием C++». – М.: Вильямс, 2015. – 1386 с.
2. Герберт Шилдт «C++. Базовый курс». – М.: Вильямс, 2014. – 62 с.
3. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений, 3-е издание, – М.: ООО «И. Д. Вильямс», 2008. – 720 с.
4. Фаулер, М. Язык UML. Основы / Мартин Фаулер. – 3-е изд. – СПб.: Символ, 2005. – 192 с.
5. Бьерн Страуструп. Язык программирования С++. Второе дополнительное издание. – СПб: Питер. 2013. – 369 с.

**Приложения**

**Приложение 1. Исходный код основной программы**

|  |
| --- |
| #include <iostream>  #include "ATM.h"  using namespace std;  ATM sysATM;  int main(int argc, char\* argv[])  {  // Запускаем процедуру показа меню  sysATM.ShowMenu();  } |

Листинг 1. Исходный код программы Menu.cpp.

**Приложение 2. Исходный код файла User.cpp**

|  |
| --- |
| #pragma once  #include "User.h"  using namespace std;  User::User()  {  login = "";  password = "";  balance = 0.0;  }  User::User(string ALogin, string APassword, double ABalance)  {  login = ALogin;  password = APassword;  balance = ABalance;  }  bool User::ChangeBalance(double Offset)  {  if (balance + Offset >= 0)  {  balance += Offset;  return true;  }  else  {  return false;  }  }  bool User::CheckPassword(string APassword)  {  return APassword == password;  }  double User::GetBalance()  {  return balance;  } |

Листинг 2. Исходный код User.cpp.

**Приложение 3. Исходный код заголовочного файла User**

|  |
| --- |
| #pragma once  #include <iostream>  #include <string>  using namespace std;  class User  {  public:  User();  User(string ALogin, string APassword, double ABalance);  bool ChangeBalance(double Offset);  bool CheckPassword(string APassword);  double GetBalance();  private:  double balance;  string login; |

Листинг 3. Исходный код заголовочного файла User.h.

**Приложение 4. Исходный код файла ATM.cpp**

|  |
| --- |
| #include "ATM.h"  #include "User.h"  using namespace std;  ATM::ATM()  {  currentUser = NULL;  if (!LoadFromFile())  {  cout << "Failed to load user list" << endl;  system("pause");  }  }  void ATM::AuthUser()  {  // Очищаем экран, запрашиваем данные для авторизации  system("cls");  string login, password;  cout << "Enter login:\n>";  getline(cin >> ws, login);  cout << "Enter password:\n>";  getline(cin >> ws, password);  // Проверяем, есть ли пользователь с такими данными в базе  if (Auth(login, password))  {  cout << "Success!" << endl;  system("pause");  }  else  {  cout << "Error!" << endl;  system("pause");  }  }  bool ATM::Auth(string ALogin, string APassword)  {  // Поиск пользователя по имени в базе  if (Users.find(ALogin) != Users.end())  {  // Проверка пароля  if (Users[ALogin].CheckPassword(APassword))  {  currentUser = &Users[ALogin];  return true;  }  }  return false;  }  void ATM::LogOut()  {  if (currentUser != NULL) {  system("cls");  currentUser = NULL;  cout << "Success" << endl;  system("pause");  }  }  int ATM::MoneyOperation()  {  system("cls");  int amount;  try  {  if (currentUser == NULL) {  return -1;  }  cout << "Enter amount:\n>";  cin >> amount;  if (!cin.fail())  {  if (amount > 0 && amount > currentUser->GetBalance())  {  cout << "You have no such means! Request a smaller amount." << endl;  system("pause");  return -1;  }  else  {  currentUser->ChangeBalance(-1.0 \* amount);  cout << "Success!" << endl;  system("pause");  return amount;  }  }  else  {  cout << "You've entered incorrect value or something vent wrong!" << endl;  system("pause");  return -1;  }  cin.clear();  }  catch (char ex)  {  cout << "Requires Input Numbers! " << endl;  system("pause");  }  return -1;  }  int ATM::PayBills()  {  if (currentUser == NULL) {  return -1;  }  int ind = 0;  do {  system("cls");  cout << "+++++++++++++++++++++++++" << endl;  cout << "+ 1. Cell phone +" << endl;  cout << "+ 2. Internet +" << endl;  cout << "+ 7. Exit +" << endl;  cout << "+++++++++++++++++++++++++" << endl;  cout << "Enter the item number:\n> ";  cin >> ind;  if (!cin.fail())  {  switch (ind)  {  case 1:  PayMobile();  ind = 7;  break;  case 2:  PayISP();  ind = 7;  break;  }  }  else  {  cin.clear();  ind = 0;  cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  }  } while (ind != 7);  return 0;  }  int ATM::Withdraw()  {  // Очищаем экран  system("cls");  try  {  // Проверяем, авторизован ли пользователь  if (currentUser == NULL) {  return -1;  }  // Выполняем процедуру для операций над балансом  int res = MoneyOperation();  if (res >= 0) {  FormPaymentReceipt("Withdraw", -1, res);  system("pause");  return 0;  }  }  catch (char ex)  {  cout << "Requires Input Numbers! " << endl;  system("pause");  }  return -1;  }  int ATM::PayMobile()  {  system("cls");  long long mobileNumber;  try  {  if (currentUser == NULL) {  return -1;  }  cout << "Enter phone number:\n>";  cin >> mobileNumber;  if ((mobileNumber >= 9000000000) && (mobileNumber <= 9999999999))  {  int res = MoneyOperation();  if (res >= 0) {  FormPaymentReceipt("Cell phone", mobileNumber, res);  system("pause");  }  }  else  {  cout << "Requires correct number! " << endl;  system("pause");  }  }  catch (char ex)  {  cout << "Wrong input! " << endl;  system("pause");  }  return -1;  }  int ATM::PayISP()  {  system("cls");  long long accountNum;  try  {  if (currentUser == NULL) {  return -1;  }  cout << "Enter account number:\n>";  cin >> accountNum;  int res = MoneyOperation();  if (res >= 0) {  FormPaymentReceipt("Internet bills", accountNum, res);  system("pause");  return 0;  }  }  catch (char ex)  {  cout << "Requires Input Numbers! " << endl;  system("pause");  }  return -1;  }  void ATM::ShowBalance()  {  if (currentUser != NULL) {  system("cls");  cout << currentUser->GetBalance() << endl;  system("pause");  }  }  User\* ATM::GetCurrentUser()  {  if (currentUser != NULL) {  return currentUser;  }  return nullptr;  }  void ATM::ShowMenu()  {  int ind = 0;  do  {  // Очищаем экран  system("cls");  // Выводим меню  cout << "+++++++++++++++++++++++++" << endl;  cout << "+ 1. Authentication +" << endl;  // Выводим дополнтиельные пункты, если пользователь авторизовался  if (currentUser != NULL) {  cout << "+ 2. Withdraw +" << endl;  cout << "+ 3. Show balance +" << endl;  cout << "+ 4. Pay bills +" << endl;  cout << "+ ---------------------+" << endl;  cout << "+ 5. Log out +" << endl;  }  cout << "+ 6. Exit +" << endl;  cout << "+++++++++++++++++++++++++" << endl;  // Запрашиваем и считываем номер пункта  cout << "Enter the item number:\n> ";  cin >> ind;  // Если считывание прошло без ошибок, выполняем операцию, соответствующую выбранному пункту  if (!cin.fail())  {  switch (ind)  {  case 1:  // Авторизация  AuthUser();  break;  case 2:  // Вывод средств  Withdraw();  break;  case 3:  // Показ баланса  ShowBalance();  break;  case 4:  // Оплата счетов  PayBills();  break;  case 50:  //Search(0);  break;  case 5:  // Выход из учётной записи  LogOut();  break;  }  }  else  {  // Если при считывании произошли какие-то ошибки  // Очищаем флаг ошибки, сбрасываем выбранный пункт и очищаем буффер  cin.clear();  ind = 0;  cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  }  } while (ind != 6); // Если выбран пункт 6 - выходим из программы  }  void ATM::FormPaymentReceipt(string operationStr, long long accountNum, double sum)  {  time\_t \_tm = time(NULL);  char dateStr[256];  struct tm curtime;  if (!localtime\_s(&curtime, &\_tm)) {  asctime\_s(dateStr, &curtime);  cout << endl << "Operation: " << operationStr << endl;  if (accountNum > 0) {  cout << "Account number: " << accountNum << endl;  }  cout << "Date: " << string(dateStr) <<  "Payment amount: " << sum << endl <<  "Current balance: " << currentUser->GetBalance() << endl << endl;  }  }  bool ATM::LoadFromFile()  {  ifstream myfile("users.txt");  string line;  try {  if (myfile) // same as: if (myfile.good())  {  while (!myfile.eof()) // same as: while (getline( myfile, line ).good())  {  string login, password;  float balance;  getline(myfile, login);  getline(myfile, password);  getline(myfile, line);  balance = stof(line);  User tmpUser(login, password, balance);  Users[login] = tmpUser;  }  myfile.close();  }  }  catch (const ifstream::failure& ex)  {  return false;  }  return true;} |

Листинг 4. Исходный код файла ATM.cpp.

**Приложение 4. Исходный код заголовочного файла ATM.h**

|  |
| --- |
| #pragma once  #include <string>  #include <map>  #include <time.h>  #include <fstream>  #include "User.h"  using namespace std;  struct Date  {  unsigned short day;  unsigned short month;  unsigned short year;  };  class ATM  {  public:  ATM();  void ShowMenu();  private:  bool LoadFromFile();  void AuthUser();  int MoneyOperation();  int PayBills();  void ShowBalance();  int PayMobile();  int PayISP();  bool Auth(string ALogin, string APassword);  void LogOut();  int Withdraw();  User\* GetCurrentUser();  map<string, User> Users;  User\* currentUser;  void FormPaymentReceipt(string operationStr, long long accountNum,double sum);  };string password;}; |

Листинг 5. Исходный код заголовочного файла ATM.h.