**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №1

по дисциплине: **«Визуальные средства разработки программных**

**приложений»**

на тему: **Потоки ввода-вывода**

Выполнил: студент гр. ИТП-31

Дашкевич Д.А.

Принял: ассистент

Белявский Е. В.

Гомель 2019

**Цель**: изучить потоки ввода-вывода.

**Ход работы**

**Вариант 2**

# **Задание:**

1. Выполнить задания, сохраняя объекты приложения в одном или нескольких файлах с применением механизма сериализации. Объекты должны содержать одно или больше полей, помеченных как *static*, а также *transient*.

2. Реализовать примеры использования *InputStreamReader*, *BufferedReader* для считывания строки с русскоязычными символами с использованием потока *System*.*in*.

Система Платежи. Клиент имеет одну или несколько Кредитных Карт, каждая из которых соответствует некоторому Счету в системе платежей. Клиент может при помощи Счета сделать Платеж, заблокировать Счет и пополнить Счет. Администратор снимает блокировку.

**Результат работы программы:**

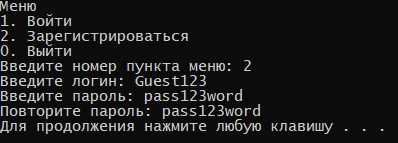
****

Рисунок 1 – Вывод в консоль процесса регистрации клиента

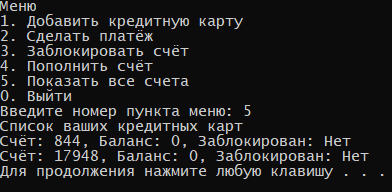


Рисунок 2 – Вывод в консоль результата всех счетов клиента

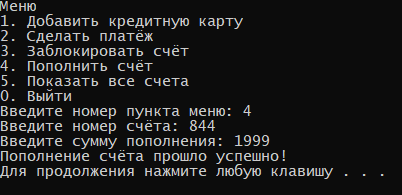


Рисунок 3 – Вывод в консоль результата пополнения счёта 844 на сумму 1999

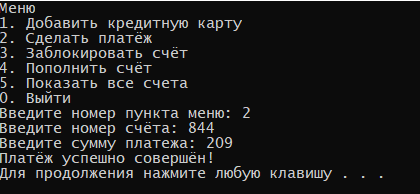


Рисунок 4 – Вывод в консоль результата совершения платежа на сумму 209 со счёта 844

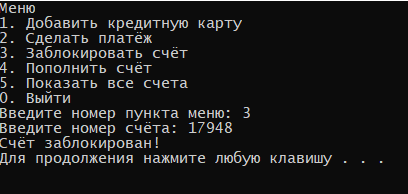


Рисунок 5 – Вывод в консоль результата блокировки счёта 17948

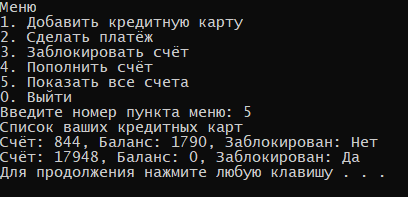


Рисунок 6 – Вывод в консоль значений счетов клиента после нескольких операций

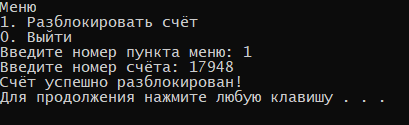


Рисунок 7 – Вывод в консоль результата разблокировки счёта 17948 администратором

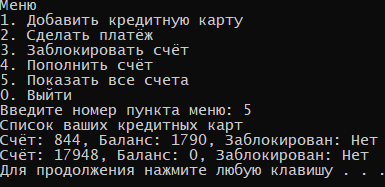


Рисунок 8 – Счета клиента после разблокировки счёта 17948

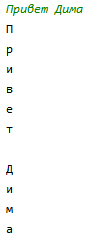


Рисунок 9 – Считывание русского текста с помощью *InputStreamReader* и вывод в консоль



Рисунок 10 – Считывание русского текста с помощью *BufferedReader* и вывод в консоль

**Вывод:** в результате выполнения лабораторной работы были изучены потоки ввода-вывода в *Java* и разработана программа, которая работает с потоками ввода-вывода и стандартной сериализацией языка программирования *Java*.

**Приложение А**

**Исходный код программы**

**Main.java:**

**public class** Main {  
  
 **private static final** Logger ***logger*** = LoggerFactory.*getLogger*(Main.**class**);  
  
 **public static void** main(String[] args) {  
 **new** Menu().openMenu();  
*// input1();  
// input2();* }  
  
 **public static void** input1() {  
 InputStreamReader inputReader = **new** InputStreamReader(System.***in***);  
 **while** (**true**) {  
 **char** ch = 0;  
 **try** {  
 ch = (**char**) inputReader.read();  
 } **catch** (IOException e) {  
 ***logger***.warn(e.getMessage());  
 }  
 System.***out***.println(ch);  
 }  
 }  
  
 **public static void** input2() {  
 BufferedReader bufferedReader = **new** BufferedReader(**new** InputStreamReader(System.***in***));  
 **while** (**true**) {  
 **try** {  
 System.***out***.println(bufferedReader.readLine());  
 } **catch** (IOException e) {  
 ***logger***.warn(e.getMessage());  
 }  
 }  
 }  
}

**Menu.java:**

**public class** Menu {  
 **private static final** String ***ADMIN\_LOGIN*** = **"admin"**;  
 **private static final** String ***ADMIN\_PASSWORD*** = **"admin"**;  
  
 **private static final** String ***ENTER\_LOGIN\_STR*** = **"Введите логин: "**;  
 **private static final** String ***ENTER\_PASSWORD\_STR*** = **"Введите пароль: "**;  
 **private static final** String ***ENTER\_ACCOUNT\_STR*** = **"Введите номер счёта: "**;  
  
 **private** Scanner **in** = **new** Scanner(System.***in***);  
  
 **private** PaymentSystem **paymentSystem**;  
 **private** Client **currentClient** = **null**;  
  
 **public** Menu() {  
 **paymentSystem** = PaymentSystemFactory.*getPaymentSystem*(PaymentSystemFactory.***SIMPLEFILE***);  
 }  
  
 **public void** openMenu() {  
 startMenuCycle();  
 }  
  
 **private long** inputLongValue() {  
 **long** k;  
 **try** {  
 k = **in**.nextLong();  
 } **catch** (InputMismatchException ex) {  
 ex.printStackTrace();  
 k = -1;  
 }  
 **return** k;  
 }  
  
 **private void** startMenuCycle() {  
 **while** (**true**) {  
 showStartMenu();  
 **int** k = (**int**) inputLongValue();  
 **switch** (k) {  
 **case** 0:  
 **paymentSystem**.saveChanges();  
 **return**;  
 **case** 1:  
 enter();  
 **break**;  
 **case** 2:  
 register();  
 **break**;  
 **default**:  
 System.***out***.println(**"Неправильный номер пункта меню!"**);  
 }  
 }  
 }  
  
 **private void** enter() {  
 System.***out***.print(***ENTER\_LOGIN\_STR***);  
 String login = **in**.next();  
 System.***out***.print(***ENTER\_PASSWORD\_STR***);  
 String password = **in**.next();  
 **if** (login.equals(***ADMIN\_LOGIN***) && password.equals(***ADMIN\_PASSWORD***)) {  
 adminMenuCycle();  
 **return**;  
 }  
 Client client = **paymentSystem**.findClientByLogin(login);  
 **if** (client == **null**) {  
 System.***out***.println(**"Клиента с таким логином не существует!"**);  
 } **else if** (client.getPassword().equals(password)) {  
 **currentClient** = client;  
 clientMenuCycle();  
 } **else** {  
 System.***out***.println(**"Неправильный пароль!"**);  
 }  
 }  
  
 **private void** register() {  
 String login, password, repeatedPassword;  
 **do** {  
 System.***out***.print(***ENTER\_LOGIN\_STR***);  
 login = **in**.next();  
 } **while** (**paymentSystem**.findClientByLogin(login) != **null**);  
 **do** {  
 System.***out***.print(***ENTER\_PASSWORD\_STR***);  
 password = **in**.next();  
 System.***out***.print(**"Повторите пароль: "**);  
 repeatedPassword = **in**.next();  
 } **while** (!password.equals(repeatedPassword));  
 **currentClient** = **new** Client(login, password);  
 **paymentSystem**.addClient(**currentClient**);  
 }  
  
 **private void** clientMenuCycle() {  
 **while** (**true**) {  
 showClientMenu();  
 **int** k = (**int**) inputLongValue();  
 **switch** (k) {  
 **case** 0:  
 **currentClient** = **null**;  
 **return**;  
 **case** 1:  
 **currentClient**.addCreditCard();  
 **break**;  
 **case** 2:  
 makePayment();  
 **break**;  
 **case** 3:  
 freezeAccount();  
 **break**;  
 **case** 4:  
 makeDeposit();  
 **break**;  
 **case** 5:  
 showAccounts();  
 **break**;  
 **default**:  
 System.***out***.println(**"Неправильный номер пункта меню!"**);  
 }  
 }  
 }  
  
 **private void** makePayment() {  
 System.***out***.print(***ENTER\_ACCOUNT\_STR***);  
 **long** account = inputLongValue();  
 CreditCard creditCard = **currentClient**.getCreditCardByAccount(account);  
 **if** (creditCard != **null**) {  
 System.***out***.print(**"Введите сумму платежа: "**);  
 **long** amount = inputLongValue();  
 **if** (creditCard.makePayment(amount)) {  
 System.***out***.println(**"Платёж успешно совершён!"**);  
 } **else** {  
 System.***out***.println(**"Платёж не совершён по неизвестной причине!"**);  
 }  
 } **else** {  
 System.***out***.println(**"Счёт не существует!"**);  
 }  
 }  
  
 **private void** freezeAccount() {  
 System.***out***.print(***ENTER\_ACCOUNT\_STR***);  
 **long** account = inputLongValue();  
 CreditCard creditCard = **currentClient**.getCreditCardByAccount(account);  
 **if** (creditCard != **null**) {  
 creditCard.freeze();  
 System.***out***.println(**"Счёт заблокирован!"**);  
 } **else** {  
 System.***out***.println(**"Счёт не существует!"**);  
 }  
 }  
  
 **private void** makeDeposit() {  
 System.***out***.print(***ENTER\_ACCOUNT\_STR***);  
 **long** account = inputLongValue();  
 CreditCard creditCard = **currentClient**.getCreditCardByAccount(account);  
 **if** (creditCard != **null**) {  
 System.***out***.print(**"Введите сумму пополнения: "**);  
 **long** amount = inputLongValue();  
 **if** (creditCard.makeDeposit(amount)) {  
 System.***out***.println(**"Пополнение счёта прошло успешно!"**);  
 } **else** {  
 System.***out***.println(**"Ошибка пополнения счёта!"**);  
 }  
 } **else** {  
 System.***out***.println(**"Счёт не существует!"**);  
 }  
 }  
  
 **private void** showAccounts() {  
 ArrayList<CreditCard> cards = **currentClient**.getCreditCards();  
 System.***out***.println(**"Список ваших кредитных карт"**);  
 **for** (CreditCard card : cards) {  
 System.***out***.printf(**"Счёт: %d, Баланс: %d, Заблокирован: %s\n"**, card.getAccount(),  
 card.getBalance(), card.isFreezed() ? **"Да"** : **"Нет"**);  
 }  
 }  
  
 **private void** adminMenuCycle() {  
 **while** (**true**) {  
 showAdminMenu();  
 **int** k = (**int**) inputLongValue();  
 **switch** (k) {  
 **case** 0:  
 **return**;  
 **case** 1:  
 unfreezeAccount();  
 **break**;  
 **default**:  
 System.***out***.println(**"Неправильный номер пункта меню!"**);  
 }  
 }  
 }  
  
 **private void** unfreezeAccount() {  
 System.***out***.print(***ENTER\_ACCOUNT\_STR***);  
 **long** account = inputLongValue();  
 CreditCard card = **paymentSystem**.findCreditCardByAccount(account);  
 **if** (card != **null**) {  
 card.unfreeze();  
 System.***out***.println(**"Счёт успешно разблокирован!"**);  
 } **else** {  
 System.***out***.println(**"Счёт не существует!"**);  
 }  
 }  
  
 **private void** show(MenuItems t) {  
 *pause*();  
 *clear*();  
 System.***out***.println(**"Меню"**);  
 t.showItems();  
 System.***out***.println(**"0. Выйти"**);  
 System.***out***.print(**"Введите номер пункта меню: "**);  
 }  
  
 **private void** showStartMenu() {  
 show(() -> {  
 System.***out***.println(**"1. Войти"**);  
 System.***out***.println(**"2. Зарегистрироваться"**);  
 });  
 }  
  
 **private void** showClientMenu() {  
 show(() -> {  
 System.***out***.println(**"1. Добавить кредитную карту"**);  
 System.***out***.println(**"2. Сделать платёж"**);  
 System.***out***.println(**"3. Заблокировать счёт"**);  
 System.***out***.println(**"4. Пополнить счёт"**);  
 System.***out***.println(**"5. Показать все счета"**);  
 });  
 }  
  
 **private void** showAdminMenu() {  
 show(() -> System.***out***.println(**"1. Разблокировать счёт"**));  
 }  
  
 **private static void** clear() {  
 **try** {  
 **new** ProcessBuilder(**"cmd"**, **"/c"**, **"cls"**).inheritIO().start().waitFor();  
 } **catch** (IOException | InterruptedException ex) {  
 System.***out***.println(ex.getMessage());  
 }  
 }  
  
 **private static void** pause() {  
 **try** {  
 **new** ProcessBuilder(**"cmd"**, **"/c"**, **"pause"**).inheritIO().start().waitFor();  
 } **catch** (IOException | InterruptedException ex) {  
 System.***out***.println(ex.getMessage());  
 }  
 }  
  
  
 **private interface** MenuItems {  
 **void** showItems();  
 }  
}

**CreditCard.java:**

**public class** CreditCard **implements** Serializable {  
 **private int id** = -1;  
 **private long account**;  
 **private long balance** = 0;  
 **private boolean freezed** = **false**;  
  
 CreditCard() {  
 **this**.**account** = Math.*abs*(**new** Random().nextInt(100000));  
 }  
  
 **private** CreditCard(**long** account, **long** balance, **boolean** freezed) {  
 **this**.**account** = account;  
 **this**.**balance** = balance;  
 **this**.**freezed** = freezed;  
 }  
  
 **public** CreditCard(**int** id, **long** account, **long** balance, **boolean** freezed) {  
 **this**(account, balance, freezed);  
 **this**.**id** = id;  
 }  
  
 **public boolean** makePayment(**long** amount) {  
 **if** (**balance** - amount < 0 || amount <= 0 || **freezed**) {  
 **return false**;  
 }  
 **balance** -= amount;  
 **return true**;  
 }  
  
 **public void** freeze() {  
 **freezed** = **true**;  
 }  
  
 **public void** unfreeze() {  
 **freezed** = **false**;  
 }  
  
 **public void** setId(**int** id) {  
 **this**.**id** = id;  
 }  
  
 **public int** getId() {  
 **return id**;  
 }  
  
 **public long** getAccount() {  
 **return account**;  
 }  
  
 **public boolean** makeDeposit(**long** amount) {  
 **if** (amount <= 0) {  
 **return false**;  
 }  
 **balance** += amount;  
 **return true**;  
 }  
  
 **public long** getBalance() {  
 **return balance**;  
 }  
  
 **public boolean** isFreezed() {  
 **return freezed**;  
 }  
}

**Client.java:**

**public class** Client **implements** Serializable {  
 **private int id** = -1;  
 **private** String **login**;  
 **private** String **password**;  
 **private** ArrayList<CreditCard> **creditCards**;  
  
 **public** Client(String login, String password) {  
 **this**.**login** = login;  
 **this**.**password** = password;  
 **this**.**creditCards** = **new** ArrayList<>();  
 }  
  
 **public** Client(**int** id, String login, String password) {  
 **this**(login, password);  
 **this**.**id** = id;  
 }  
  
 **public void** addCreditCard() {  
 **creditCards**.add(**new** CreditCard());  
 }  
  
 **public void** addCreditCard(CreditCard creditCard) {  
 **creditCards**.add(creditCard);  
 }  
  
 **public** CreditCard getCreditCardByAccount(**long** account) {  
 **for** (CreditCard creditCard : **creditCards**) {  
 **if** (creditCard.getAccount() == account) {  
 **return** creditCard;  
 }  
 }  
 **return null**;  
 }  
  
 **public void** setId(**int** id) {  
 **this**.**id** = id;  
 }  
  
 **public int** getId() {  
 **return id**;  
 }  
  
 **public** String getLogin() {  
 **return login**;  
 }  
  
 **public** String getPassword() {  
 **return password**;  
 }  
  
 **public** ArrayList<CreditCard> getCreditCards() {  
 **return creditCards**;  
 }  
}

**PaymentSystem.java:**

**public interface** PaymentSystem {  
 Client findClientByLogin(String login);  
  
 CreditCard findCreditCardByAccount(**long** account);  
  
 **void** addClient(Client client);  
  
 **void** saveChanges();  
}

**AbstractPaymentSystem.java:**

**public abstract class** AbstractPaymentSystem **implements** PaymentSystem {  
 **protected** ArrayList<Client> **clients**;  
  
 **protected abstract void** loadClients();  
  
 **protected abstract void** saveClients();  
  
 **public** AbstractPaymentSystem() {  
 **this**.**clients** = **new** ArrayList<>();  
 }  
  
 @Override  
 **public void** saveChanges() {  
 saveClients();  
 }  
  
 @Override  
 **public** Client findClientByLogin(String login) {  
 **for** (Client client : **clients**) {  
 **if** (client.getLogin().equals(login)) {  
 **return** client;  
 }  
 }  
 **return null**;  
 }  
  
 @Override  
 **public** CreditCard findCreditCardByAccount(**long** account) {  
 **for** (Client client : **clients**) {  
 CreditCard card = client.getCreditCardByAccount(account);  
 **if** (card != **null**) {  
 **return** card;  
 }  
 }  
 **return null**;  
 }  
}

**SimpleFilePaymentSystem.java:**

**public class** SimpleFilePaymentSystem **extends** AbstractPaymentSystem {  
  
 **private** String **filename**;  
  
 SimpleFilePaymentSystem() {  
 **super**();  
 **filename** = **"client\_database.txt"**;

loadClients();  
 }  
  
 @Override  
 **public void** addClient(Client client) {  
 **clients**.add(client);  
 }  
  
 @Override  
 **protected void** loadClients() {  
 File file = **new** File(**filename**);  
 **if** (file.exists()) {  
 **try** {  
 ObjectInputStream stream = **new** ObjectInputStream(**new** FileInputStream(file));  
 **int** size = stream.readInt();  
 **clients** = **new** ArrayList<>(size);  
 **for** (**int** i = 0; i < size; i++) {  
 **clients**.add((Client) stream.readObject());  
 }  
 stream.close();  
 } **catch** (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 @Override  
 **protected void** saveClients() {  
 File file = **new** File(**filename**);  
 **if** (!file.exists()) {  
 **try** {  
 **if** (!file.createNewFile()) {  
 **throw new** IOException(**"Ошибка при создании файла!"**);  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 **try** {  
 ObjectOutputStream stream = **new** ObjectOutputStream(**new** FileOutputStream(file));  
 stream.writeInt(**clients**.size());  
 **for** (Client client : **clients**) {  
 stream.writeObject(client);  
 }  
 stream.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

**PaymentSystemFactory.java:**

**public class** PaymentSystemFactory {  
  
 **public enum** PaymentSystemType {  
 ***SIMPLEFILE*** }  
  
 **public static** PaymentSystem getPaymentSystem(PaymentSystemType type) {  
 **if** (type == PaymentSystemType.***SIMPLEFILE***) {  
 **return new** SimpleFilePaymentSystem();  
 }  
 **return null**;  
 }  
}