**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №7

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

**приложений»**

на тему: **Обработка** **данных с использованием *XML***

Выполнил: студент гр. ИТП-31

Дашкевич Д.А.

Принял: ассистент

Белявский Е. В.

Гомель 2020

**Цель**: изучить методы обработки данных с использованием *XML* в *Java*.

**Ход работы**

**Вариант 2**

# **Задание:** выполнить задания, сохраняя объекты приложения в одном или нескольких файлах *XML*. Для работы изучить и использовать: *SAX*-парсер, *DOM*-парсер, *StAX*-парсер.

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:**

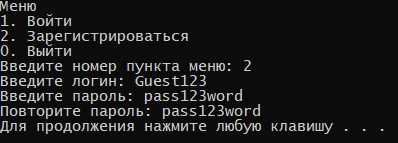
****

Рисунок 1 – Вывод в консоль процесса регистрации клиента

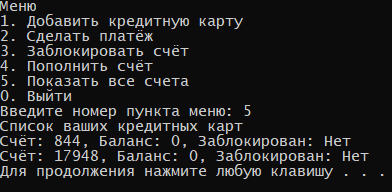


Рисунок 2 – Вывод в консоль результата всех счетов клиента

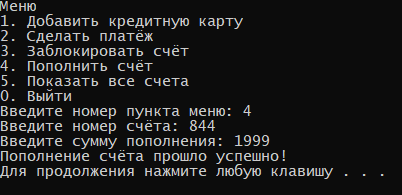


Рисунок 3 – Вывод в консоль результата пополнения счёта 844 на сумму 1999

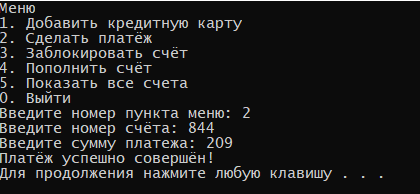


Рисунок 4 – Вывод в консоль результата совершения платежа на сумму 209 со счёта 844

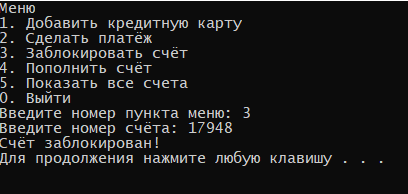


Рисунок 5 – Вывод в консоль результата блокировки счёта 17948

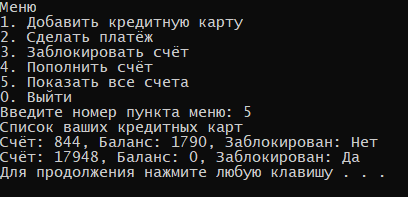


Рисунок 6 – Вывод в консоль значений счетов клиента после нескольких операций

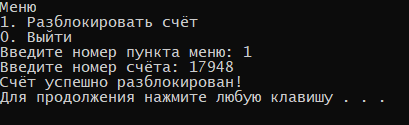


Рисунок 7 – Вывод в консоль результата разблокировки счёта 17948 администратором

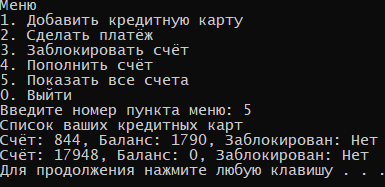


Рисунок 8 – Счета клиента после разблокировки счёта 17948

**Вывод:** в результате выполнения лабораторной работы были изучены методы обработки данных с использованием *XML* в *Java* и разработана программа, которая считывает данные с помощью трёх различных парсеров.

**Приложение А**

**Исходный код программы**

**Main.java:**

**public class** Main {  
 **public static void** main(String[] args) {  
 **new** Menu().openMenu();  
 }  
}

**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.***STAX\_XML***);  
 }  
  
 **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 static int** *counter* = 0;  
  
 **private int id** = *counter*++;  
 **private long account**;  
 **private long balance** = 0;  
 **private boolean freezed** = **false**;  
  
 CreditCard() {  
 **this**.**account** = Math.*abs*(**new** Random().nextInt(100000));  
 }  
  
 **public** CreditCard(**int** id, **long** account, **long** balance, **boolean** freezed) {  
 **this**.**id** = id;  
 **this**.**account** = account;  
 **this**.**balance** = balance;  
 **this**.**freezed** = freezed;  
 }  
  
 **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 static int** *counter* = 0;  
  
 **private int id** = *counter*++;  
 **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(CreditCard creditCard) {  
 **creditCards**.add(creditCard);  
 }  
  
 **public void** addCreditCard() {  
 **creditCards**.add(**new** 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** List<Client> **clients** = **new** ArrayList<>();  
  
 **protected abstract void** loadClients();  
  
 **protected abstract void** saveClients();  
  
 **public** AbstractPaymentSystem() {  
 loadClients();  
 }  
  
 @Override  
 **public void** addClient(Client client) {  
 **clients**.add(client);  
 }  
  
 @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**;  
 }  
}

**PaymentSystemFactory.java:**

**public class** PaymentSystemFactory {  
  
 **public enum** PaymentSystemType {  
 ***SAX\_XML***, ***DOM\_XML***, ***STAX\_XML*** }  
  
 **public static** PaymentSystem getPaymentSystem(PaymentSystemType type) {  
 **switch** (type) {  
 **case *SAX\_XML***:  
 **return new** SaxXmlPaymentSystem();  
 **case *DOM\_XML***:  
 **return new** DomXmlPaymentSystem();  
 **case *STAX\_XML***:  
 **return new** StaxXmlPaymentSystem();  
 **default**:  
 **return null**;  
 }  
 }  
}

**XmlPaymentSystem.java:**

**public abstract class** XmlPaymentSystem **extends** AbstractPaymentSystem {  
  
 **static final** String ***FILENAME*** = **"storage.xml"**;  
  
 **static final** String ***CLIENT\_LIST\_ELEMENT*** = **"Clients"**;  
 **static final** String ***CLIENT\_ELEMENT*** = **"Client"**;  
 **static final** String ***CREDIT\_CARD\_LIST\_ELEMENT*** = **"CreditCards"**;  
 **static final** String ***CREDIT\_CARD\_ELEMENT*** = **"CreditCard"**;  
  
 **static final** String ***ID\_ATTRIBUTE*** = **"id"**;  
  
 **static final** String ***LOGIN\_ATTRIBUTE*** = **"login"**;  
 **static final** String ***PASSWORD\_ATTRIBUTE*** = **"password"**;  
  
 **static final** String ***ACCOUNT\_ATTRIBUTE*** = **"account"**;  
 **static final** String ***BALANCE\_ATTRIBUTE*** = **"balance"**;  
 **static final** String ***FREEZED\_ATTRIBUTE*** = **"freezed"**;  
}

**SaxXmlPaymentSystem.java:**

**public class** SaxXmlPaymentSystem **extends** XmlPaymentSystem {  
 **private static final** Logger ***logger*** = LoggerFactory.*getLogger*(SaxXmlPaymentSystem.**class**);  
  
 @Override  
 **protected void** loadClients() {  
 SAXParserFactory factory = SAXParserFactory.*newInstance*();  
 **try** {  
 SAXParser parser = factory.newSAXParser();  
 XmlHandler handler = **new** XmlHandler();  
 parser.parse(**new** File(***FILENAME***), handler);  
 } **catch** (ParserConfigurationException | SAXException | IOException e) {  
 ***logger***.error(e.getMessage());  
 }  
 }  
  
 @Override  
 **protected void** saveClients() {  
 XMLOutputFactory xmlOutputFactory = XMLOutputFactory.*newInstance*();  
 XMLStreamWriter writer = **null**;  
 **try** {  
 writer = xmlOutputFactory.createXMLStreamWriter(**new** FileOutputStream(***FILENAME***));  
 writer.writeStartDocument();  
 writer.writeStartElement(***CLIENT\_LIST\_ELEMENT***);  
 **for** (Client client : **clients**) {  
 writer.writeStartElement(***CLIENT\_ELEMENT***);  
 writer.writeAttribute(***ID\_ATTRIBUTE***, String.*valueOf*(client.getId()));  
 writer.writeAttribute(***LOGIN\_ATTRIBUTE***, client.getLogin());  
 writer.writeAttribute(***PASSWORD\_ATTRIBUTE***, client.getPassword());  
 writer.writeStartElement(***CREDIT\_CARD\_LIST\_ELEMENT***);  
 **for** (CreditCard creditCard : client.getCreditCards()) {  
 writer.writeStartElement(***CREDIT\_CARD\_ELEMENT***);  
 writer.writeAttribute(***ID\_ATTRIBUTE***, String.*valueOf*(creditCard.getId()));  
 writer.writeAttribute(***ACCOUNT\_ATTRIBUTE***, String.*valueOf*(creditCard.getAccount()));  
 writer.writeAttribute(***BALANCE\_ATTRIBUTE***, String.*valueOf*(creditCard.getBalance()));  
 writer.writeAttribute(***FREEZED\_ATTRIBUTE***, String.*valueOf*(creditCard.isFreezed()));  
 writer.writeEndElement();  
 }  
 writer.writeEndElement();  
 writer.writeEndElement();  
 }  
 writer.writeEndElement();  
 writer.writeEndDocument();  
 } **catch** (FileNotFoundException | XMLStreamException e) {  
 ***logger***.error(e.getMessage());  
 } **finally** {  
 **if** (writer != **null**) {  
 **try** {  
 writer.close();  
 } **catch** (XMLStreamException e) {  
 ***logger***.error(e.getMessage());  
 }  
 }  
 }  
 }  
  
  
 **private class** XmlHandler **extends** DefaultHandler {  
  
 **private** Client **client** = **null**;  
  
 @Override  
 **public void** startElement(String uri, String localName, String qName, Attributes attributes) {  
 **switch** (qName) {  
 **case *CLIENT\_ELEMENT***:  
 **try** {  
 **int** id = Integer.*parseInt*(attributes.getValue(***ID\_ATTRIBUTE***));  
 String login = attributes.getValue(***LOGIN\_ATTRIBUTE***);  
 String password = attributes.getValue(***PASSWORD\_ATTRIBUTE***);  
 **if** (login == **null** || password == **null**) {  
 **return**;  
 }  
 **client** = **new** Client(id, login, password);  
 } **catch** (NumberFormatException e) {  
 ***logger***.warn(e.getMessage());  
 }  
 **break**;  
 **case *CREDIT\_CARD\_ELEMENT***:  
 **try** {  
 **int** id = Integer.*parseInt*(attributes.getValue(***ID\_ATTRIBUTE***));  
 **long** account = Long.*parseLong*(attributes.getValue(***ACCOUNT\_ATTRIBUTE***));  
 **long** balance = Long.*parseLong*(attributes.getValue(***BALANCE\_ATTRIBUTE***));  
 **boolean** freezed = Boolean.*parseBoolean*(attributes.getValue(***FREEZED\_ATTRIBUTE***));  
 **if** (**client** != **null**) {  
 **client**.addCreditCard(**new** CreditCard(id, account, balance, freezed));  
 }  
 } **catch** (NumberFormatException e) {  
 ***logger***.warn(e.getMessage());  
 }  
 **break**;  
 }  
 }  
  
 @Override  
 **public void** endElement(String uri, String localName, String qName) {  
 **if** (***CLIENT\_ELEMENT***.equals(qName)) {  
 **if** (**client** != **null**) {  
 **clients**.add(**client**);  
 **client** = **null**;  
 }  
 }  
 }  
 }  
}

**StaxXmlPaymentSystem.java:**

**public class** StaxXmlPaymentSystem **extends** SaxXmlPaymentSystem {  
 **private static final** Logger ***logger*** = LoggerFactory.*getLogger*(StaxXmlPaymentSystem.**class**);  
  
 @Override  
 **protected void** loadClients() {  
 Client client = **null**;  
 XMLInputFactory factory = XMLInputFactory.*newInstance*();  
 **try** {  
 XMLEventReader eventReader = factory.createXMLEventReader(**new** FileReader(***FILENAME***));  
 **while** (eventReader.hasNext()) {  
 XMLEvent event = eventReader.nextEvent();  
 **switch** (event.getEventType()) {  
 **case** XMLStreamConstants.***START\_ELEMENT***:  
 StartElement startElement = event.asStartElement();  
 String qName = startElement.getName().getLocalPart();  
 **switch** (qName) {  
 **case *CLIENT\_ELEMENT***:  
 **try** {  
 **int** id = Integer.*parseInt*(startElement.getAttributeByName(QName.*valueOf*(***ID\_ATTRIBUTE***)).getValue());  
 String login = startElement.getAttributeByName(QName.*valueOf*(***LOGIN\_ATTRIBUTE***)).getValue();  
 String password = startElement.getAttributeByName(QName.*valueOf*(***PASSWORD\_ATTRIBUTE***)).getValue();  
 **if** (login == **null** || password == **null**) {  
 **return**;  
 }  
 client = **new** Client(id, login, password);  
 } **catch** (NumberFormatException e) {  
 ***logger***.warn(e.getMessage());  
 }  
 **break**;  
 **case *CREDIT\_CARD\_ELEMENT***:  
 **try** {  
 **int** id = Integer.*parseInt*(startElement.getAttributeByName(QName.*valueOf*(***ID\_ATTRIBUTE***)).getValue());  
 **long** account = Long.*parseLong*(startElement.getAttributeByName(QName.*valueOf*(***ACCOUNT\_ATTRIBUTE***)).getValue());  
 **long** balance = Long.*parseLong*(startElement.getAttributeByName(QName.*valueOf*(***BALANCE\_ATTRIBUTE***)).getValue());  
 **boolean** freezed = Boolean.*parseBoolean*(startElement.getAttributeByName(QName.*valueOf*(***FREEZED\_ATTRIBUTE***)).getValue());  
 **if** (client != **null**) {  
 client.addCreditCard(**new** CreditCard(id, account, balance, freezed));  
 }  
 } **catch** (NumberFormatException e) {  
 ***logger***.warn(e.getMessage());  
 }  
 **break**;  
 }  
 **break**;  
 **case** XMLStreamConstants.***END\_ELEMENT***:  
 EndElement endElement = event.asEndElement();  
 **if** (***CLIENT\_ELEMENT***.equals(endElement.getName().getLocalPart())) {  
 **if** (client != **null**) {  
 **clients**.add(client);  
 client = **null**;  
 }  
 }  
 **break**;  
 }  
 }  
 } **catch** (XMLStreamException | FileNotFoundException e) {  
 ***logger***.error(e.getMessage());  
 }  
 }  
}

**DomXmlPaymentSystem.java:**

**public class** DomXmlPaymentSystem **extends** XmlPaymentSystem {  
 **private static final** Logger ***logger*** = LoggerFactory.*getLogger*(DomXmlPaymentSystem.**class**);  
  
 @Override  
 **protected void** loadClients() {  
 DocumentBuilderFactory factory = DocumentBuilderFactory.*newInstance*();  
 DocumentBuilder builder;  
 **try** {  
 builder = factory.newDocumentBuilder();  
 Document document = builder.parse(**new** File(***FILENAME***));  
 NodeList clientNodes = document.getElementsByTagName(***CLIENT\_ELEMENT***);  
 **for** (**int** i = 0; i < clientNodes.getLength(); i++) {  
 Node node = clientNodes.item(i);  
 NamedNodeMap attributes = node.getAttributes();  
 **int** id = Integer.*parseInt*(attributes.getNamedItem(***ID\_ATTRIBUTE***).getNodeValue());  
 String login = attributes.getNamedItem(***LOGIN\_ATTRIBUTE***).getNodeValue();  
 String password = attributes.getNamedItem(***PASSWORD\_ATTRIBUTE***).getNodeValue();  
 **if** (login == **null** || password == **null**) {  
 **break**;  
 }  
 Client client = **new** Client(id, login, password);  
 NodeList cardsNodes = ((Element) node).getElementsByTagName(***CREDIT\_CARD\_ELEMENT***);  
 **for** (**int** j = 0; j < cardsNodes.getLength(); j++) {  
 Node node2 = cardsNodes.item(j);  
 NamedNodeMap attributes2 = node2.getAttributes();  
 **int** id2 = Integer.*parseInt*(attributes2.getNamedItem(***ID\_ATTRIBUTE***).getNodeValue());  
 **long** account = Long.*parseLong*(attributes2.getNamedItem(***ACCOUNT\_ATTRIBUTE***).getNodeValue());  
 **long** balance = Long.*parseLong*(attributes2.getNamedItem(***BALANCE\_ATTRIBUTE***).getNodeValue());  
 **boolean** freezed = Boolean.*parseBoolean*(attributes2.getNamedItem(***FREEZED\_ATTRIBUTE***).getNodeValue());  
 client.addCreditCard(**new** CreditCard(id2, account, balance, freezed));  
 }  
 **clients**.add(client);  
 }  
 } **catch** (ParserConfigurationException | IOException | SAXException | NumberFormatException e) {  
 ***logger***.error(e.getMessage());  
 }  
 }  
  
 @Override  
 **protected void** saveClients() {  
 DocumentBuilderFactory factory = DocumentBuilderFactory.*newInstance*();  
 DocumentBuilder builder;  
 **try** {  
 builder = factory.newDocumentBuilder();  
 Document document = builder.newDocument();  
 Element root = document.createElement(***CLIENT\_LIST\_ELEMENT***);  
 document.appendChild(root);  
 **for** (Client client : **clients**) {  
 Element clientElement = document.createElement(***CLIENT\_ELEMENT***);  
 clientElement.setAttribute(***ID\_ATTRIBUTE***, String.*valueOf*(client.getId()));  
 clientElement.setAttribute(***LOGIN\_ATTRIBUTE***, String.*valueOf*(client.getLogin()));  
 clientElement.setAttribute(***PASSWORD\_ATTRIBUTE***, String.*valueOf*(client.getPassword()));  
 Element cardListElement = document.createElement(***CREDIT\_CARD\_LIST\_ELEMENT***);  
 **for** (CreditCard creditCard : client.getCreditCards()) {  
 Element cardElement = document.createElement(***CREDIT\_CARD\_ELEMENT***);  
 cardElement.setAttribute(***ID\_ATTRIBUTE***, String.*valueOf*(creditCard.getId()));  
 cardElement.setAttribute(***ACCOUNT\_ATTRIBUTE***, String.*valueOf*(creditCard.getAccount()));  
 cardElement.setAttribute(***BALANCE\_ATTRIBUTE***, String.*valueOf*(creditCard.getBalance()));  
 cardElement.setAttribute(***FREEZED\_ATTRIBUTE***, String.*valueOf*(creditCard.isFreezed()));  
 cardListElement.appendChild(cardElement);  
 }  
 clientElement.appendChild(cardListElement);  
 root.appendChild(clientElement);  
 }  
 TransformerFactory transformerFactory = TransformerFactory.*newInstance*();  
 Transformer transformer = transformerFactory.newTransformer();  
 transformer.setOutputProperty(OutputKeys.***ENCODING***, **"UTF-8"**);  
 transformer.setOutputProperty(OutputKeys.***INDENT***, **"yes"**);  
 transformer.setOutputProperty(**"{http://xml.apache.org/xslt}indent-amount"**, **"4"**);  
 DOMSource domSource = **new** DOMSource(document);  
 StreamResult streamResult = **new** StreamResult(**new** File(***FILENAME***));  
 transformer.transform(domSource, streamResult);  
 } **catch** (ParserConfigurationException | TransformerException e) {  
 ***logger***.error(e.getMessage());  
 }  
 }  
}