Учреждение Высшего Образования

«Университет управления «ТИСБИ»

Факультет Информационных Технологий

Кафедра Информационных Технологий

Курсовая работа по дисциплине объектно-ориентированное программирование на тему: «Разработка объектной программы для обработки данных о вкладах клиентов банка»

Проверил:

Кандидат технических наук, доцент

Козин. А. Н.

Выполнил:

Студент группы П-711

Мустафин К.Н

Казань 2019

**Оглавление**

[**Постановка задачи 2**](#_Toc10899825)

[**Описание используемых структур данных с алгоритмами выполнения основных операций 3**](#_Toc10899826)

[**Краткие сведения об объектном подходе 7**](#_Toc10899827)

[**Формализованное описание разработанных классов 12**](#_Toc10899828)

[**Демонстрационный модуль с характеристикой использованных стандартных компонентов и списком реализованных обработчиков 15**](#_Toc10899829)

[**Структура проекта в соответствии с использованным инструментом разработки 18**](#_Toc10899830)

[**Список литературы 19**](#_Toc10899831)

[**Полный листинг программы с краткими комментариями 20**](#_Toc10899832)

Постановка задачи

Подсистема управления денежными вкладами клиентов банка должна вести список лицевых счетов своих клиентов с хранением фамилии клиента и текущей суммы вклада. Для каждого лицевого счета должен создаваться список выполненных с ним операций с указанием уникальной дары операции и величины изменения вклада (пополнение или снятие денежных средств).

Разработка программы включает в себя определение необходимых объектов и описание их в виде классов, программную реализацию методов добавления и удаления операций со счетами с объема всех вкладов, всестороннее тестирование методов с помощью консольного (при разработке) и оконного (в окончательном варианте) приложения.

Для объединения клиентов используется структура данных в виде кольцевой очереди на основе обычного массива. Для объединения операций клиента используется структура данных в виде адресного замкнутого (кольцевого) упорядоченного двунаправленного списка без заголовка.

Стандартные контейнеры/коллекции (включая обобщенные классы) разрешается использовать только как дополнение к самостоятельно разработанным классам.

В качестве языка разработки разрешается использовать Java и соответствующий инструмент быстрой разработки приложений.

Описание используемых структур данных с алгоритмами выполнения основных операций

Требуется объединить операции каждого клиента (его историю) в адресный замкнутый (кольцевой) упорядоченного двунаправленного список без заголовка. Для этого создается приватизированная переменная history, которая является первым элементом в двунаправленном кольцевом списке.

Описание работы адресного замкнутого двунаправленного списка без заголовка: это список, в котором каждый элемент «знает» обоих своих соседей, как левого, так и правого. Для этого каждый элемент должен иметь не одно, а два связующих поля: указатель на элемент слева и указатель на элемент справа. В кольцевом списке: поле next у последнего элемента содержит адрес первого элемента, поле prev у первого – адрес последнего; здесь достаточно только одного основного указателя history.

history

n-ый элемент

Дата и операция

Next

Prev

…

Первый элемент

Дата и операция

Next

Prev

Второй элемент

Дата и операция

Next

Prev

**Алгоритм добавления:**

1. Поиск подходящего элемента куда мы хотим поставить (так как список упорядоченный: в консольном варианте мы ищем по операциям и ищем первую бОльшую сумму операций, и уже перед ним ставим, а в оконной реализации сравниваем сначала по датам, если даты равны, то сравниваем по операциям) и работаем через переменную curr.
2. Выделяем память для новой переменной temp
3. Формируем поля нового элемента, в частности:
   * В поле next заносится адрес заданного элемента  
     temp.setNext(curr.getNext());
   * В поле prev заносится адрес предшествующего элемента  
     temp.setPrev(curr);
4. Изменяем адресное поле prev у заданного элемента на адрес нового элемента
   * curr.setPrev(temp);
5. Изменяем адресно поле next у предшествующего элемента на адрес нового элемента
   * temp.getPrev().setNext(temp);

Заданный элемент

Дата и операция

Next

Prev

Предыдущ. элемент

Дата и операция

Next

Prev

шаг 5

Новый элемент

Дата и операция

Next

Prev

шаг 3

шаг 3

шаг 4

шаг 1

temp

шаг 2

curr

**Алгоритм удаления:**

1. Проверка наличия элементов

2. Поиск удаляемого элемента

3. Если элемент найден адресуем его findDel

4. Изменяем адресное поле prev у следующего за удаляемым элементом на адрес элемента, предшествующего удаляемому

findDel.getNext().setPrev(findDel.getPrev());

5. Изменяем адресное поле next у элемента, предшествующего удаляемому на адрес элемента, следующего за удаляемым

findDel.getPrev().setNext(findDel.getNext());

1. Обрабатываем удаляемый элемент, используя указатель findDel

findDel

шаг 3

шаг 2

шаг 4

Следующий элемент

Дата и операция

Next

Prev

Удаляемый элемент

Дата и операция

Next

Prev

Предыдущ. элемент

Дата и операция

Next

Prev

Для объединения клиентов используется структура данных в виде кольцевой очереди на основе обычного массива. Так как работа выполнена на языке программирования java, то массив я создавал из клиентов следующим образом: создавался приватизированный массив в главном классе(mainBank) bank.

Описание работы кольцевой очереди на основе обычного массива: очередь-массив растёт от первых элементов массива к последним. Тогда индекс first – определяет первую занятую ячейку массива, а индекс last – первую свободную ячейку. Пустая очередь определяется как first = last = 0. При каждом добавлении нового элемента переменная last увеличивается на 1, а при удалении на 1 увеличивается переменная first. Суть кольцевой реализации очереди состоит в том, что при удалении элементов освобождаются ячейки в начале массива и разумно использовать из для добавления новых элементов, если в конце массива уже нет свободных элементов. Такое поведение легко реализуется за счет принудительной установки индексов first и last в 1, если они выходят за пределы массива.

Surname, budget, countOp, history

Surname, budget, countOp, history

Surname, budget, countOp, history

null

null

first

last

0

1

2

3

4

bank

**Алгоритм добавления:**

* Проверить возможность добавления (есть ли свободные ячейки)
* Добавить элемент в массив по индексу last
* Изменить индекс last на 1
* Если last выходит за пределы массива, то установить last в 0
* Увеличить счётчик числа элементов в очереди

**Алгоритм удаления:**

* Проверить возможность удаления
* Извлечь элемент из массива по индексу first и выполнить с ним необходимые действия
* Увеличить индекс first на 1
* Если first выходит за пределы массива, то установить first в 0
* Уменьшить счетчик числа элементов в очереди

…

шаг 5

шаг 5

Краткие сведения об объектном подходе

**Объект** – это достаточное общее и широкое понятие, которое может иметь разные трактовки. В широком смысле слова объект – это любая сущность, имеющая некоторый набор свойств и обладающая некоторым поведением.

Отсюда следует, что объектом может быть практически все, что угодно – материальные предметы (компьютер, автомобиль, человек), логические понятия (файл, документ, таблица, список, управляющая кнопка, окно в многооконной графической системе, форма, приложение), математические понятия (например, геометрические объекты типа отрезка или окружность).

Под объектом в узком смысле можно понимать некоторое формализованное описание рассматриваемой сущности, т.е. модель исходного объекта. Для такого описания нужен некоторый язык, например – язык программирования. В этом случае объект становится элементом языка, и именно в таком контексте он и будет рассматриваться далее.

Моделирование реальных объектов – это важнейший этап разработки объектных программ, во многом определяющий успех всего проекта. Необходимо понимать, что любая модель – это только часть исходного моделируемого объекта, отражающая те свойства и атрибуты обхекта, которые наиболее важны с точки зрения решаемой задачи. Изменение постановки задачи может привести к существенному изменению модели. Особенно это характерно для сложных объектов. Поэтому один из важнейших принципов построения моделей – это принцип абстрагирования, т.е. выделение наиболее существенных для решаемой задачи черт исходного объекта и отбрасывание второстепенных деталей. Отсюда следуетЮ что для одного и того же исходного объекта можно построить разные модели, каждая из которых будет отражать только наиболее важные в данный момент черты объекта.

Исходный объект

Описание 3

Описание 2

Описание 1

Использование объектов при разработке сложных информационных систем оказалось очень удачным решением.

Разработчики на основе общения с заказчиком и на основе анализа предметной области должны построить адекватные описания основных сущностей решаемой задачи, перевести эти описания в необходимый формальный вид и создать объектную программу как набор взаимодействующих объектов.

Достоинством объектных программ является их четкая структуризация – каждый объект выполняет свою строго определенную задачу, а их совместная работа позволяет достичь поставленных целей.

Программный объект – это условное понятие, с которым связывается набор некоторых данных и программный ко обработки этих даннных

**Объект: данные + программный код**

Учитывая огромную важность методов-конструкторов, рассмотрим отдельно их назначение и особенности использования.

Конструктор отвечает за динамическое создание нового объекта при выполнении программ, которое включает в себя:

* Выделение памяти, необходимой для хранения значений свойств создаваемого объекта
* Занесение в выделенную область начальных значений свойств создаваемого объекта

Важно понимать, что пока объект не создан, он не может представлять никакие услуги и поэтому конструктор должен вызываться раньше всех остальных методов. Для одного и того же объекта можно предусмотреть несколько различных конструкторов, которые по-разному инициализируют свойства создаваемого объекта. Начальные значения свойств часто передаются конструктору как входные параметры.

Деструктор отвечает за уничтожение объекта, т.е. освобождение памяти, выделенной объекту. В отличие от конструкторов, механизм деструкторов реализован не во всех языках.

Ещё одна группа очень часто используемых методов – это методы доступа к закрытым свойствам объекта. Введение таких методово позволяет организовать контролируемый доступ к внутренним данным объекта. В общем случае для закрытого свойства можно ввести два метода доступа:

* Метод для чтения хранящегося в свойстве значения (get-метод)
* Метод для изменения значения свойства (set-метод)

В итоге объектная программа представляет собой набор взаимодействующих объектов, которые обращаются друг к другу за выполнением необходимых действий.

Конструкторы  
методы доступа  
открытые методы

Закрытые данные   
Закрытые методы

Объект C

Конструкторы  
методы доступа  
открытые методы

Закрытые данные   
Закрытые методы

Объект B

Объект А

Конструкторы  
методы доступа  
открытые методы

Закрытые данные   
Закрытые методы

Очевидно, что при работе объектной программы одновременно может существовать множество однотипных программных объектов. Поэтому необходим инструмент формального описания таких однотипных объектов и в качестве такого инструмента выступает однотипных объектов и в качестве такого инструмента выступает следующее важнейшее понятие объектного подхода – класс.

**Класс** представляет собой формализованный способ описания однотипных объектов, т.е. объектов с одинаковым набором свойств и методов. Именно при описании класса перечисляются свойства и реализуются методы соответствующих объектов. Разработка объектной программы начинается с описания необходимых классов. На основе одного класса можно создать любое разумное число объектов, называемых экземплярами этого класса. Все используемые в программе объекты должны быть экземплярами некоторых классов, стандартных или аналогично соответствию между понятиями «переменная» и «тип данных». Описание класса включает в себя:

* Заголовок класса, включающий специальную директиву class и имя класса; очень часто в заголовке задается ещё и некоторая дополнительная информация о классе;
* Тело класса, содержащее перечень свойств, заголовки методов и их программную реализацию;

**Основные принципы структурирования** в случае объектного подхода связаны с различными аспектами базового понимания предметной задачи, которое требуется для оптимального управления соответствующей моделью:

* Абстрагирование для выделения в моделируемом предмете важного для решения конкретной задачи по предмету, в конечном счёте — контекстное понимание предмета, формализуемое в виде класса;
* Инкапсуляция для быстрой и безопасной организации собственно иерархической управляемости: чтобы было достаточно простой команды «что делать», без одновременного уточнения как именно делать, так как это уже другой уровень управления;
* Наследование для быстрой и безопасной организации родственных понятий: чтобы было достаточно на каждом иерархическом шаге учитывать только изменения, не дублируя всё остальное, учтённое на предыдущих шагах;
* Полиморфизм для определения точки, в которой единое управление лучше распараллелить или наоборот — собрать воедино.

Формализованное описание разработанных классов

public class bank {

static class mainBank { //объект банк

private klient[] bank; //кольцевая очередь для объединения клиентов на основе массива

private int first, last; //указатели на клиентов первого и последнего в очереди

private int maxClients; //ограничение на количество клиентов

private int count = 0;//счётчик клиентов

public void mainBank(int maxCount){

//конструктор

}

public boolean addClient(String newSur){

//фун-ция для добавления клиентов

}

public klient getKlient(String newSur){

//фун-кция для получение клиента

}

public boolean deleteFirstClient() {

//удаление первого

}

public String getInfo(klient tempInfo) {

//получение информации о клиенте

}

public String getInfoAboutAll(){

//получение информации обо всех

}

public boolean loadFromFile(Scanner lFin) {

//загрузка из файла всей структуры

}

public void saveToFile(FileWriter sFout) {

//сохранение в файл всей стркутуры

}

}

public class klient {//класс для клиента

private String surname;

private int budget, countOp;

private operation history; //двунаправленный список без заголовка для операций (а это первый элемент)

public void klient(String sur){

//конструктор для создания нового клиента

}

public void setNewSurname(String newSur){

//функция которая устанавливает новую фамилию

}

public boolean setHistory(String newData, int newOp) {

//функция для добавления действий в список history(дата и +число или -число) а так же изменения значения бюджета

}

public operation findOp(String findData, int findOp){

//функция для поиска операции

}

public boolean deleteHistory(String deletingData, int deletingOp){

//функция для удаление операции

}

public int getBudget(){

}

public String getSurname(){

}

public int getCountOp(){

}

public String getInfo(int space) {

//возвращает всю инфу про клиента (все его операции и даты)

}

}

public class operation {//класс для операций

private String data; //дата

private int oper; //операция

private operation next, prev; //указатель на следующую и предыдущую операцию

public void operation(operation first, String newData, int newOp) {

//конструктор для создания новой операции

data = newData;

oper = newOp;

next = first;

prev = first;

}

public operation getNext() {

return next;

}

public void setNext(operation newNext) {

next = newNext;

}

public void setPrev(operation newPrev) {

prev = newPrev;

}

public void setOper(int newOp) {

oper = newOp;

}

public void setData(String newData) {

data = newData;

}

public String getData() {

return data;

}

public int getOper() {

return oper;

}

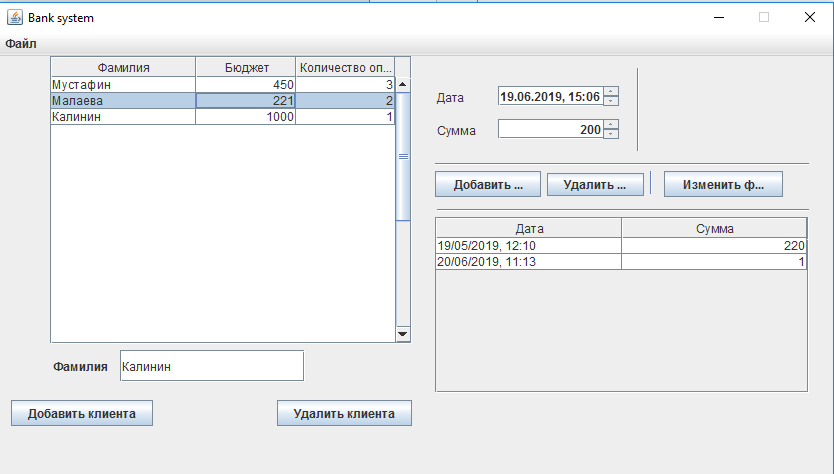
public operation getPrev() {

return prev;

}

}

Демонстрационный модуль с характеристикой использованных стандартных компонентов и списком реализованных обработчиков



При первом запуске программы открывается вот такое окно:

Здесь мы видим таблицу в которой есть Фамилия, Бюджет и Количество операций, а также текстовое поле для ввода Фамилии, кнопки Добавить и Удалить и второе текстовое поле с полям Дата операции и Сумма.

**Добавление клиента**

Прежде чем добавить нового клиента текстовое поле проходит проверку на вводимые значения: можно вводить только латинские буквы, кириллицу и цифры, затем, если система не создана, то создается система, и добавляется новый клиент.

**Вывод информации**

Выделив клиента вызвался обработчик private void infoTableMouseClicked(java.awt.event.MouseEvent evt), который в свою очередь сделал видим панель справа в которой есть: дата и сумма операции, кнопки добавить, удалить и изменить фамилию, а также таблица с информативной частью об операциях клиента. Берется фамилия, производится её поиск в системе и выводится вся его информация в таблицу.

**Добавление операции**

Для добавления операции нужно выбрать дату в поле даты и сумму (>0) и нажать на кнопку Добавить операцию. Нажав на кнопку добавить операцию вызвался обработчик кнопки addOperButtonActionPerformed(java.awt.event.ActionEvent evt), который соответственно, упорядочив операции по датам, добавил их, а также изменил значения в основной таблице и вывел слово Добавлено.

**Удалении операции**

При нажатии на кнопку Удалить операцию вызывается обработчик deleteOperButtonActionPerformed(java.awt.event.ActionEvent evt), который совершает поиск указанной даты и суммы и при успешном нахождении удаляет его, а также выводит надпись Удалено и меняет значения бюджета и количества операций в таблице.

**Удаление клиента**

Так как клиенты объединены в кольцевую очередь, нажав на кнопку Удалить клиента вызвался обработчик deleteClientButtonActionPerformed(java.awt.event.ActionEvent evt), который удаляет первого в очереди. И панель с информативной частью закроется.

**Изменение фамилии**

При нажатии на кнопку Изменить фамилию вызвался обработчик кнопки changeSurnameButtonActionPerformed(java.awt.event.ActionEvent evt), который сделав все необходимые проверки и при успешной смене фамилии, вывел слово Изменено и поменял фамилию в таблице.

**Алгоритм сохранения в файл**

Вызвав кнопку сохранить из верхнего меню «Файл» появилось окно (ВАЖНО, ЧТОБЫ ФАЙЛ БЫЛ УЖЕ СОЗДАН), для указания пути сохранения. Это вызывается обработчиком кнопки saveMenuItemActionPerformed(java.awt.event.ActionEvent evt), который выделив выбранный путь файла сохраняет туда всю систему. Сперва идёт количество клиентов с системе, потом фамилия и количество операций самого клиента, а затем даты и операции и так для всех клиентов.

**Алгоритм загрузки с файла**

Нажав кнопку загрузить мы вызываем обработчик кнопки private void loadMenuItemActionPerformed(java.awt.event.ActionEvent evt), который взяв файл с указанного пути загружает все данные, а так же заново создает систему. Так же как и в сохранении: сначала количество клиентов, потом фамилия первого клиента и количество его операций и затем сами операции: дата и операции; итак для всех клиентов.

**Контроль фамилии**

За это отвечает обработчик private void newClientTextFieldKeyTyped(java.awt.event.KeyEvent evt), который смотрит, какую буквы ввели последней. Если эта буква из алфавита, или цифра, или кнопка удаления, или пробел, или цифра, то всё хорошо, иначе выходит ошибка, что вводимые символы не допустимы.

Структура проекта в соответствии с использованным инструментом разработки

NetBeans IDE — свободная интегрированная среда разработки приложений (IDE) на языках программирования Java, Python, PHP, JavaScript, C, C++, Ада и ряда других.

Создается главный файл kursach.java, где описывается вся система, и файл NewJFrame.java, где описывается внешняя часть оконного приложения. При сборке всего проекта создается файл kursach.jar.

Главные файлы kursach.java и NewJFrame.form / NewJFrame.java лежат в папке kursach\src\kursach.

Исполняемый файл kursach.jar лежит в папке kursach\dist\. При его запуске, запускает оконное приложение проекта.

Если нужен тестовый вариант клиентов, то для этого есть уже готовый вариант save.txt, который лежит в папке kursach\файлы\. Загрузив его, можно увидеть готовый вариант работы системы банка.

Комментарии к файлам Kursach.java и NewJFrame.java лежат в папке kursach\файлы и называются «комментарий к Kursach.java» и «комментарий к NewJFrame.java».

Список литературы

1. Вайсфельд М. Объектно-ориентированное мышление, — СПб.: Питер, 2015 г.
2. Гаст Х. Объектно-ориентированное проектирование: концепции и программный код, — СПб.: Питер 2019 г.
3. Кнут Д. Искусство программирования, том 1. Оснвоные алгоритмы. – М.:Вильямс, 2014г.
4. Кнут Д. Искусство программирования, том 3. Сортировка и поиск. – М.:Вильямс
5. Кормен Т. Алгоритмы. Вводный курс. – М.:Вильямс, 2016 г.
6. Лафоре Р. Структуры данных и алгоритмы в Java. – СПб.: Питер, 2017г.
7. Седжквик Р. Алгоритмы на Java. – M.:Вильямс, 2016 г.
8. 2014 г.
9. Стивенс Р. Алгоритмы. Теория и практическое применения – М.: Издательство «Э», 2016 г.
10. Шилдт Г. Java 8: полное руководство. – М.: Вильямс, 2015 г.
11. Система ответов и вопросов о программировании Stack Overflow. – Режим доступа: https://stackoverflow.com/http://venec.ulstu.ru/lib/disk/2015/149.pdf
12. О.Ф.Власенко. Основы программирования на Java. Основные управляющие конструкции. 2015.
13. Джошуа блох, пер. Стрельцов В., ред. Усманов Р.Java. Эффективное программирование. Профобразование. 2017.
14. Мухаметзянов Р.Р. 2017.Основы программирования на Java. Учебное пособие. Набережночелнинский государственный педагогический университет.

Полный листинг программы с краткими комментариями

Файл kursach.java(главный файл):

package kursach;

import java.io.BufferedReader;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.FileWriter;

import java.io.IOException;

import java.io.InputStreamReader;

import static java.lang.System.in;

import java.util.ArrayList;

import java.util.Scanner;

public class Kursach {

static mainBank koyash1; //переменная для работы с «банком»

public static void main(String[] args) {

NewJFrame frame = new NewJFrame();

frame.setVisible(true); //запускаем окно

}

static class mainBank { //объект банк

private klient[] bank; //кольцевая очередь для объединения клиентов на основе массива

private int first, last; //указатели на клиентов первого и последнего в очереди

private int maxClients; //ограничение на количество клиентов

private int count = 0;//счётчик клиентов

public void mainBank(int maxCount){

//конструктор

first = 0;

last = 0;

maxClients = maxCount;

bank = new klient[maxClients];

}

public int getMaxClients(){ return maxClients;}

public int getCount(){return count;}

klient[] getBank(){return bank; }

public int getFirst(){return first; }

public int getLast(){return last; }

public boolean addClient(String newSur){

//фун-ция для добавления клиентов

if (count < maxClients) {

if (!checkForSur(newSur)){

return false;

}

count++;

bank[last] = new klient();

bank[last].klient(newSur);

last++;

if (last >= maxClients)

last = 0;

return true;

} else {

return false;

}

}

public boolean checkForSur(String checkSur){

if (count > 0){

int i = first;

int j = 0;

while (j < count){

if (bank[i].getSurname().equals(checkSur)){

return false;

}

i++;

j++;

if (i >= maxClients)

i = 0;

}

return true;

} return true;

}

public klient getKlient(String newSur){

//фун-кция для получение клиента

int i = first;

int j = 0;

while (j < count){

if (bank[i].getSurname().equals(newSur)){

return bank[i];

}

i++;

j++;

if (i >= maxClients)

i = 0;

}

return null;

}

public boolean deleteFirstClient() {

//удаление первого

if (count > 0) {

bank[first] = null;

first++;

if (first == maxClients)

first = 0;

count--;

return true;

}else{

return false;

}

}

public String[][] getInfo(klient tempInfo) {

//получение информации о клиенте

String[][] result;

result = tempInfo.getInfo(tempInfo.getCountOp());

return result;

}

public boolean loadFromFile(String path) throws FileNotFoundException, ParseException{

//загрузка из файла всей структуры

BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStream(path)));

String n;

try {

if (null != (n = br.readLine())){

koyash1 = new mainBank();

koyash1.mainBank(Integer.valueOf(n));

for (int i = 0; i < Integer.valueOf(n); i++) {

String name = br.readLine();

koyash1.addClient(name);

String countOfOp = br.readLine();

for (int j = 0; j < Integer.valueOf(countOfOp); j++) {

String dateFromfile = br.readLine();

String opFromFile = br.readLine();

klient loadClientTemp = koyash1.getKlient(name);

loadClientTemp.setHistory (dateFromfile, Integer.valueOf(opFromFile));

}

}

br.close();

return true;

} else {

br.close();

return false;

}

} catch (IOException e){

return false;

}

}

public void saveToFile(FileWriter sFout) {

//сохранение в файл всей стркутуры

try {

sFout.write(String.valueOf(count) + "\n");

for (int i = 0; i < count; i++) {

sFout.write(bank[i].getSurname() + "\n" + bank[i].getCountOp() + "\n");

String[][] info = bank[i].getInfo(bank[i].getCountOp());

if (!info.equals("") && info != (null))

for (int j = 0; j < info.length;j++){

sFout.write(info[j][0] + "\n" + info[j][1] + "\n");

}

}

sFout.close();

} catch (IOException e){

System.out.println("Some error!");

}

}

}

static public class klient { //класс клиента

private String surname;

private int budget, countOp = 0;

private operation history; //двунаправленный список без заголовка для операций (а это первый элемент)

public void klient(String sur){

//конструктор для создания нового клиента

surname = sur;

budget = 0;

}

public void setNewSurname(String newSur){

//функция которая устанавливает новую фамилию

surname = newSur;

}

public boolean setHistory(String newData, int newOp) throws ParseException {

//функция для добавления действий в список history(дата и +число или -число) а так же изменения значения бюджета

if (newOp < 0 && Math.abs(newOp) > budget) { //если на счету не достаточно денег при снятии

return false;

} else {

budget += newOp; //изменение бюджета

if (history == null) { //если история операций ещё не была создана

history = new operation(); //то выделяем место для новой переменной

history.operation(history, newData, newOp); //и строим(через конструктор) её

} else {//если же история уже была создана

int i = 0; //вспомогательный счётчик для поиска места

operation curr = history; //вспомогательная переменная для поиска куда ставить новое значение

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy, HH:mm");

Date dateCurr = null, dateTemp = null;

dateCurr = sdf.parse(curr.getData());

dateTemp = sdf.parse(newData);

while (i < countOp - 1 && dateCurr.compareTo(dateTemp) == -1){ //пока мы не пробежались по всем элементам И не нашли бОльший элемент

i++;//идём дальше

curr = curr.getNext();

dateCurr = sdf.parse(curr.getData());

}

operation temp = new operation();//когда мы нашли ПРИМЕРНОЕ подходяещее место создаем вспомгательную переменную

temp.operation(temp, newData, newOp); //определяем её

switch (dateCurr.compareTo(dateTemp)) {

case 0: //если даты равны

while (i < countOp - 1 && curr.getOper() < newOp && dateCurr.compareTo(dateTemp) == 0){ //пока мы не пробежались по всем элементам И не нашли бОльший элемент

i++;//идём дальше

curr = curr.getNext();

dateCurr = sdf.parse(curr.getData());

}

if (dateCurr.compareTo(dateTemp) == 0){ //если мы не вышли за диапазон

if (curr.getOper() >= newOp){

temp.setNext(curr);

temp.setPrev(curr.getPrev());

temp.getPrev().setNext(temp);

curr.setPrev(temp);

} else {

temp.setNext(curr.getNext());

temp.setPrev(curr);

temp.getNext().setPrev(temp);

curr.setNext(temp);

}

if (temp.getOper() < history.getOper() && i == 0)

history = temp;

} else { //если всё таки вышли

curr = curr.getPrev();

temp.setNext(curr.getNext());

temp.setPrev(curr);

temp.getNext().setPrev(temp);

curr.setNext(temp);

}

break;

case 1: //если новая дата меньше

temp.setNext(curr);

temp.setPrev(curr.getPrev());

temp.getPrev().setNext(temp);

curr.setPrev(temp);

if (i == 0)

history = temp;

break;

default: //а иначе

temp.setNext(curr.getNext());

temp.setPrev(curr);

temp.getNext().setPrev(temp);

curr.setNext(temp);

break;

}

}

countOp++; //увеличиваем счётчик

return true;

}

}

public operation findOp(String findData, int findOp){

operation curr = history;

int i = 0;

while (i < countOp) {

if (curr.getData().equals(findData) && curr.getOper() == findOp) {

return curr;

}

curr = curr.getNext();

i++;

}

return null;

}

public boolean deleteHistory(String deletingData, int deletingOp){

//функция для удаление операции

if (history != null) {

operation findDel = findOp(deletingData, deletingOp);

if (findDel != null){

if (countOp == 1) {

history = null;

budget -= deletingOp;

history = null;

countOp--;

return true;

}

if (history == findDel)

history = findDel.getNext();

findDel.getPrev().setNext(findDel.getNext());

findDel.getNext().setPrev(findDel.getPrev());

budget -= deletingOp;

countOp--;

return true;

}

} return false;

}

public int getBudget(){

return budget;

}

public String getSurname(){

return surname;

}

public int getCountOp(){return countOp;}

public String[][] getInfo(int size) {

//возвращает всю инфу про клиента (все его операции и даты)

String[][] result = new String[size][2];

if (history != null) {

operation tempInfo = history;

result[0][0] = tempInfo.getData();

result[0][1] = String.valueOf(tempInfo.getOper());

tempInfo = tempInfo.getNext();

int j = 1;

while (j < countOp) {

result[j][0] = tempInfo.getData();

result[j][1] = String.valueOf(tempInfo.getOper());

j++;

tempInfo = tempInfo.getNext();

}

}

return result;

}

}

static public class operation { //класс операций

private String data; //дата

private int oper; //операция

private operation next, prev; //указатель на следующую и предыдущую операцию

public void operation(operation first, String newData, int newOp) {

//конструктор для создания новой операции

data = newData;

oper = newOp;

next = first;

prev = first;

}

public operation getNext() { return next; }

public void setNext(operation newNext) {next = newNext; }

public void setPrev(operation newPrev) {prev = newPrev; }

public void setOper(int newOp) {oper = newOp; }

public void setData(String newData) {data = newData; }

public String getData() {return data; }

public int getOper() {return oper; }

public operation getPrev() {return prev; }

}

}

Файл NewJFrame.java (файл для работы с формой):

package kursach;

//Необходимые библиотеки

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

import java.io.File;

import java.io.FileWriter;

import java.text.SimpleDateFormat;

import java.util.Scanner;

import javax.swing.JOptionPane;

import javax.swing.Timer;

import javax.swing.filechooser.FileFilter;

import javax.swing.filechooser.FileNameExtensionFilter;

import javax.swing.table.DefaultTableModel;

public class NewJFrame extends javax.swing.JFrame {

DefaultTableModel infoModeTable; //эта переменная нужна для обработки таблицы(не спрашивайте, так надо)) )

DefaultTableModel dateOperInfoModeTable; //для обработки таблицы с датами и операциями

Timer timerForOperLabel; //таймер для появления и исчезновения надписей

public NewJFrame() { //конструктор

initComponents();

operLabel.setText("");

operPanel.setVisible(false);

infoModeTable = (DefaultTableModel) infoTable.getModel();

dateOperInfoModeTable = (DefaultTableModel) dateOperInfoTable.getModel();

timerForOperLabel = new Timer(3000, new ActionListener(){ //каждые 3 секунды, исчезает надпись

@Override

public void actionPerformed(ActionEvent e) {

operLabel.setText("");

}

});

}

private void infoTableMouseClicked(java.awt.event.MouseEvent evt) { //обработчик если на таблицу нажали

// TODO add your handling code here:

for (int i = 0; i < infoTable.getRowCount(); i++){

if (infoTable.isRowSelected(i)){ //при выборе конкретной строки в таблицу

operPanel.setVisible(true); //делаем панель с редактирование видимым

String infoSurname = String.valueOf(infoModeTable.getValueAt(infoTable.getSelectedRow(), 0)); //берем фамилию

String[][] info = Kursach.koyash1.getInfo(Kursach.koyash1.getKlient(infoSurname)); //получаем инфу

while (dateOperInfoModeTable.getRowCount() > 0){ //очищаем таблицу

dateOperInfoModeTable.removeRow(0);

}

for (int j = 0; j < info.length;j++){ //добавляем в таблицу

dateOperInfoModeTable.addRow(new Object[]{info[j][0], info[j][1]});

}

}

}

}

private void deleteClientButtonActionPerformed(java.awt.event.ActionEvent evt) { //обработчик кнопки для удаления

if (Kursach.koyash1 != null){ //если система существует

if (Kursach.koyash1.deleteFirstClient()){ //если смогли удалить

infoModeTable.removeRow(0);//убираем строку

operPanel.setVisible(false); //скрываем панельку с редактированием

} else {

JOptionPane.showMessageDialog(null, "В системе никого нет!"); //иначе

}

} else {

JOptionPane.showMessageDialog(null, "Система не создана!"); //иначе

}

}

private void changeSurnameButtonActionPerformed(java.awt.event.ActionEvent evt) { //обработчик кнопки для смены фамилии

String s = newClientTextField.getText();

if (s.length() > 1){

if (checkForSurname(s)){

String newSurname = String.valueOf(s);

String oldSurname = String.valueOf(infoModeTable.getValueAt(infoTable.getSelectedRow(), 0));

if (Kursach.koyash1.checkForSur(newSurname)){

Kursach.koyash1.getKlient(oldSurname).setNewSurname(newSurname);

infoModeTable.setValueAt(newSurname, infoTable.getSelectedRow(), 0);

operLabel.setText("Изменено");

timerForOperLabel.stop();

timerForOperLabel.start();

} else {

JOptionPane.showMessageDialog(null, "Такая фамилия уже существует, пожалуйста немного её измените!");

}

}

} else {

JOptionPane.showMessageDialog(null, "Фамилия должна иметь больше букв");

} }

private void addOperButtonActionPerformed(java.awt.event.ActionEvent evt) { //обработчик кнопки для добавления операции

String selectedClient;

String newDate;

int newOp;

try {

selectedClient = String.valueOf(infoModeTable.getValueAt(infoTable.getSelectedRow(), 0)); //выбранный клиент

newDate = new SimpleDateFormat("dd/MM/yyyy, HH:mm").format(dateOperSpinner.getValue());//новая дата

newOp = Integer.valueOf(operTextField.getText());//новая операция

if (newOp != 0) { //если сумма в новой операции не 0

if (Kursach.koyash1.getKlient(selectedClient).setHistory(newDate, newOp)) { //если добавилось

infoModeTable.setValueAt(Kursach.koyash1.getKlient(selectedClient).getBudget(), infoTable.getSelectedRow(), 1); //изменяем в таблице бюджет

infoModeTable.setValueAt(Kursach.koyash1.getKlient(selectedClient).getCountOp(), infoTable.getSelectedRow(), 2); //изменяем в таблице количество операций

operLabel.setText("Добавлено");

timerForOperLabel.stop();

timerForOperLabel.start();

} else {

JOptionPane.showMessageDialog(null, "Недостаточно средств");

}

} catch (Exception e){

JOptionPane.showMessageDialog(null, "Нужно ввести дату в поле даты и сумму в поле суммы");

}

} else { JOptionPane.showMeesageDialog(null, “Новая сумма не может быть 0”);

String infoSurname = String.valueOf(infoModeTable.getValueAt(infoTable.getSelectedRow(), 0));

String[][] info = Kursach.koyash1.getInfo(Kursach.koyash1.getKlient(infoSurname));

while (dateOperInfoModeTable.getRowCount() > 0){

dateOperInfoModeTable.removeRow(0);

}

for (int j = 0; j < info.length;j++){

dateOperInfoModeTable.addRow(new Object[]{info[j][0], info[j][1]});

}

}

private void deleteOperButtonActionPerformed(java.awt.event.ActionEvent evt) { //обработчик кнопки удаления операции

String delStr = String.valueOf(infoModeTable.getValueAt(infoTable.getSelectedRow(), 0));

String selectedClient = String.valueOf(infoModeTable.getValueAt(infoTable.getSelectedRow(), 0));

Kursach.klient tempDel = Kursach.koyash1.getKlient(delStr);

if (tempDel != null){

try {

if (tempDel.deleteHistory(new SimpleDateFormat("dd/MM/yyyy, HH:mm").format(dateOperSpinner.getValue()), Integer.valueOf(operTextField.getText()))){ //если смогли удалить

infoModeTable.setValueAt(Kursach.koyash1.getKlient(selectedClient).getBudget(), infoTable.getSelectedRow(), 1); //меняем бюджет

infoModeTable.setValueAt(Kursach.koyash1.getKlient(selectedClient).getCountOp(), infoTable.getSelectedRow(), 2); //меняем количество операций

operLabel.setText("Удалено");

timerForOperLabel.stop();

timerForOperLabel.start();

} else {

JOptionPane.showMessageDialog(null, "Такой операции не существует!");

}

} catch (Exception e){

JOptionPane.showMessageDialog(null, "Нужно ввести дату в поле даты и сумму в поле суммы");

}

}

String infoSurname = String.valueOf(infoModeTable.getValueAt(infoTable.getSelectedRow(), 0));

String[][] info = Kursach.koyash1.getInfo(Kursach.koyash1.getKlient(infoSurname));

while (dateOperInfoModeTable.getRowCount() > 0){

dateOperInfoModeTable.removeRow(0);

}

for (int j = 0; j < info.length;j++){

dateOperInfoModeTable.addRow(new Object[]{info[j][0], info[j][1]});

}

}

private void loadMenuItemActionPerformed(java.awt.event.ActionEvent evt) { //обработчик кнопки загрузки с файла данных

FileFilter filter = new FileNameExtensionFilter("txt", "txt"); //ставим фильтер на поиск файлов txt

jFileChooser1.setFileFilter(filter);

int replyOpen = jFileChooser1.showOpenDialog(null);

if (replyOpen == jFileChooser1.APPROVE\_OPTION){ //если пользователь выбрал кнопку «открыть»

try {

String pathLoad = jFileChooser1.getSelectedFile().getAbsolutePath();//получаем путь к файлу\

int reply = JOptionPane.showConfirmDialog(null, "Вы уверены? Все не сохраненные данные будут утеряны.", null, JOptionPane.YES\_NO\_OPTION);

if (reply == JOptionPane.YES\_OPTION){ //если пользователь согласился удалить старые не сохраненные данные

Kursach.koyash1 = new Kursach.mainBank();//грузим также как и в консольном варианте

Kursach.koyash1.loadFromFile(pathLoad);

Kursach.klient[] newBank = Kursach.koyash1.getBank();//для работы с таблицей

while (infoTable.getRowCount() > 0){ //убираем строки с таблицы

infoModeTable.removeRow(0);

}

int i = Kursach.koyash1.getFirst();

while (i != Kursach.koyash1.getLast()){//теперь добавляем все строки

infoModeTable.addRow(new Object[]{newBank[i].getSurname(), newBank[i].getBudget(), newBank[i].getCountOp()});

i++;

if (i > Kursach.koyash1.getMaxClients())

i = 0;

}

}

} catch(Exception e){

}

}

}

private void saveMenuItemActionPerformed(java.awt.event.ActionEvent evt) { //обработчик кнопки для сохранения

if (Kursach.koyash1 != null) {

FileFilter filter = new FileNameExtensionFilter("txt", "txt");

jFileChooser1.setFileFilter(filter);

jFileChooser1.showSaveDialog(null);

try {

String pathSave = jFileChooser1.getSelectedFile().getAbsolutePath();

JOptionPane.showMessageDialog(null, pathSave);

FileWriter fout = new FileWriter(pathSave);

Kursach.koyash1.saveToFile(fout);

} catch(Exception e){

}

} else {

JOptionPane.showMessageDialog(null, "Система не создана");

}

}

private void newClientTextFieldKeyTyped(java.awt.event.KeyEvent evt) {//GEN-FIRST:event\_newClientTextFieldKeyTyped

char c = evt.getKeyChar();

if (!(Character.isDigit(c) || Character.isAlphabetic(c)

|| (c == KeyEvent.VK\_DELETE)

|| (c == KeyEvent.VK\_BACK\_SPACE)

|| Character.isWhitespace(c))){

evt.consume();

JOptionPane.showMessageDialog(null, "Можно вводить только латинские буквы, кириллицу и цифры");

}

}//GEN-LAST:event\_newClientTextFieldKeyTyped

private void addNewClientButtonActionPerformed(java.awt.event.ActionEvent evt) { //обработчик кнопки для добавления нового клиента

String s = newClientTextField.getText();

if (s.length() > 1){

if (checkForSurname(s)){

if (Kursach.koyash1 == null){

Kursach.koyash1 = new Kursach.mainBank();

Kursach.koyash1.mainBank(5);

}

if (Kursach.koyash1.addClient(s))

infoModeTable.addRow(new Object[]{s, 0, 0});

else

JOptionPane.showMessageDialog(null, "Нет места или клиент с такой фамилей уже существует!");

}

}else {

JOptionPane.showMessageDialog(null, "Фамилия должна иметь больше букв");

}

}