

Лабораторные работы по программированию на языке Java

Оглавление

Лабораторная работа №1. Введение в программирование на языке Java.....	2
Лабораторная работа №2. ООП в Java. Документирование кода на языке Java с использованием утилиты javadoc	8
Лабораторная работа №3. Коллекции языка Java. Stream	17
Лабораторная работа №4. Создание программ с графическим интерфейсом пользователя на языке Java. Классы пакета Swing	26
Лабораторная работа №5. Модель обработки событий в Java.	33
Лабораторная работа №6. Создание меню, графика в языке Java.....	37
Лабораторная работа №7. Работа с файлами, JList, JScrollPane, HashMap. Создание запускового jar-файла.....	41
Лабораторная работа №8. Работа с БД	48
Лабораторная работа №9. Работа с MS Office	55

Лабораторная работа №1. Введение в программирование на языке Java.

Для создания проекта в среде программирования Eclipse выбираем пункт меню File – New – Java Project. В появившемся окне в поле Project name вводим название проекта (английскими буквами). Жмем Finish. В левой части окна Eclipse в области Package Explorer появится папка созданного проекта. Вызываем контекстное меню своего проекта, пункт New – Class, в появившемся окне в поле Name вводим имя создаваемого класса (английскими буквами). Желательно, чтобы был выбран пункт `public static void main(String[] args)` для создания метода, который будет запускать программу. Созданный класс сразу появится в рабочей области Eclipse.

Первая программа на языке Java:

```
class HelloWorld{  
    public static void main(String[] args){  
        System.out.println("Hello, XXI Century World!");  
    }  
}
```

На этом простом примере можно заметить целый ряд существенных особенностей языка Java.

- Всякая программа представляет собой один или несколько классов, в этом простейшем примере только один *класс* (class).
- Начало класса отмечается служебным словом `class`, за которым следует имя класса, выбираемое произвольно, в данном случае `HelloWorld`. Все, что содержится в классе, записывается в фигурных скобках и составляет *тело класса* (class body).
- Все действия производятся с помощью методов обработки информации, коротко говорят просто *метод* (method). Это название употребляется в языке Java вместо названия «функция», применяемого в других языках.
- Методы различаются по именам. Один из методов обязательно должен называться `main`, с него начинается выполнение программы. В нашей простейшей программе только один метод, а значит, имя ему `main`.
- Как и положено функции, метод всегда выдает в результате (чаще говорят, *возвращает* (return)) только одно значение, тип которого обязательно указывается перед именем метода. Метод может и не возвращать никакого значения, играя роль процедуры, как в нашем случае. Тогда вместо типа возвращаемого значения записывается слово `void`, как это и сделано в примере.
- После имени метода в скобках, через запятую, перечисляются *аргументы* (arguments) или *параметры* метода. Для каждого аргумента указывается его тип и, через пробел, имя. В примере только один аргумент, его тип – массив, состоящий из строк символов. Строка символов – это встроенный в Java тип `String`, а квадратные скобки – признак массива. Имя массива может быть произвольным, в примере выбрано имя `args`.
- Перед типом возвращаемого методом значения могут быть записаны *модификаторы* (modifiers). В примере их два: слово `public` означает, что этот метод доступен отовсюду; слово `static` обеспечивает возможность вызова метода `main()` в самом начале выполнения программы без создания экземпляра какого-либо класса. Модификаторы вообще необязательны, но для метода `main()` они необходимы.
- Все, что содержит метод, *тело метода* (method body), записывается в фигурных скобках.

Единственное действие, которое выполняет метод `main()` в примере, заключается в вызове другого метода со сложным именем `System.out.println` и передаче ему на обработку одного аргумента, текстовой константы `"Hello, 21th century world!"`. Текстовые константы записываются в кавычках, которые являются только ограничителями и не входят в состав текста.

Все основные алгоритмические конструкции языка Java (объявление/инициализация/присваивание переменных, ветвление (операторы if, switch, тернарный оператор), циклы) заимствованы из языка C/C++ и полностью соответствуют установленному там синтаксису.

Простые (или примитивные) типы данных похожи на применяемые в языках C и C++ (см. табл.1.1 и 1.2).

Таблица 1.1. Целые типы данных.

Тип	Разрядность (байт)	Диапазон
byte	1	от -128 до 127
short	2	от -32768 до 32767
int	4	от -2147483648 до 2147483647 ($-2^{31}..2^{31}-1$)
long	8	от -9223372036854775808 до 9223372036854775807 ($-2^{63}..2^{63}-1$)
char	2	от '\u0000' до '\uFFFF', в десятичной форме от 0 до 65535

Таблица 1.2. Вещественные типы данных

Тип	Разрядность	Диапазон	Точность
float	4	$-3.4 \cdot 10^{38}..3.4 \cdot 10^{38}$	7-8 цифр
double	8	$-1.8 \cdot 10^{308}..1.8 \cdot 10^{308}$	17 цифр

Примечание: В языке Java операции % (вычисление остатка от деления), ++ (инкремент) и -- (декремент) применяются и к вещественным типам.

Массивы

Массивы в языке Java относятся к ссылочным типам и описываются своеобразно, но характерно для ссылочных типов. Описание производится в три этапа.

Первый этап – *объявление* (declaration). На этом этапе определяется только переменная типа *ссылка* (reference) на массив, содержащая тип массива. Для этого записывается имя типа элементов массива, квадратными скобками указывается, что объявляется ссылка на массив, а не простая переменная, и перечисляются имена переменных типа ссылка, например,

```
double[] a, b;
```

Здесь определены две переменные – ссылки a и b на массивы типа double. Можно поставить квадратные скобки и непосредственно после имени. Это удобно делать среди определений обычных переменных:

```
int i = 0, ar[], k = -1;
```

Здесь определены две переменные целого типа i и k, и объявлена ссылка на целочисленный массив ar.

Второй этап – *определение* (installation). На этом этапе указывается количество элементов массива, называемое его *длиной*, выделяется место для массива в оперативной памяти, переменная-ссылка получает адрес массива. Все эти действия производятся еще одной операцией языка Java – операцией new, выделяющей участок в оперативной памяти для объекта указанного в операции типа и возвращающей в качестве результата адрес этого участка. Например,

```
a = new double[5];
b = new double[100];
ar = new int[50];
```

Индексы массивов всегда начинаются с 0. Массив `a` состоит из пяти переменных `a[0]`, `a[1]`, ..., `a[4]`. Элемента `a[5]` в массиве нет. Индексы можно задавать любыми целочисленными выражениями, кроме типа `long`, например, `a[i+j]`, `a[i%5]`, `a[++i]`. Исполняющая система Java следит за тем, чтобы значения этих выражений не выходили за границы длины массива.

Третий этап – *инициализация* (initialization). На этом этапе элементы массива получают начальные значения. Например,

```
a[0] = 0.01; a[1] = -3.4; a[2] = 2.89; a[3] = 4.5; a[4] = -6.7;
for (int i = 0; i < 100; i++) b[i] = 1.0/i;
for (int i = 0; i < 50; i++) ar[i] = 2 * i + 1;
```

Первые два этапа можно совместить:

```
double[] a = new double[5], b = new double[100];
int i = 0, ar[] = new int[50], k = -1;
```

Можно сразу задать и начальные значения, записав их в фигурных скобках через запятую в виде констант или константных выражений. При этом даже необязательно указывать количество элементов массива, оно будет равно количеству начальных значений:

```
double[] a = {0.01, -3.4, 2.89, 4.5, -6.7};
```

Можно совместить второй и третий этап:

```
a = new double[] {0.1, 0.2, -0.3, 0.45, -0.02};
```

Можно даже создать безымянный массив, сразу же используя результат операции `new`, например, так:

```
System.out.println(new char[] {'H', 'e', 'l', 'l', 'o'});
```

Ссылка на массив не является частью описанного массива, ее можно перебросить на другой массив того же типа операцией присваивания. Например, после присваивания `a = b` обе ссылки `a` и `b` указывают на один и тот же массив из 100 вещественных переменных типа `double` и содержат один и тот же адрес.

Ссылка может присвоить "пустое" значение `null`, не указывающее ни на какой адрес оперативной памяти:

```
ar = null;
```

После этого массив, на который указывала данная ссылка, теряется, если на него не было других ссылок.

Кроме простой операции присваивания, со ссылками можно производить еще только сравнения на равенство, например, `a==b`, и неравенство, `a!=b`. При этом сопоставляются адреса, содержащиеся в ссылках, мы можем узнать, не ссылаются ли они на один и тот же массив.

Кроме ссылки на массив, для каждого массива автоматически определяется целая константа с одним и тем же именем `length`. Она равна длине массива. Для каждого массива имя этой константы уточняется именем массива через точку. Так, после наших определений, константа `a.length` равна 5, константа `b.length` равна 100, а `ar.length` равна 50.

Последний элемент массива `a` можно записать так: `a[a.length-1]`, предпоследний – `a[a.length-2]` и т. д. Элементы массива обычно перебираются в цикле вида:

```
double aMin = a[0], aMax = aMin;
for (int i = 1; i < a.length; i++){
    if (a[i] < aMin) aMin = a[i];
    if (a[i] > aMax) aMax = a[i];
}
```

Консольный ввод-вывод данных

Для вывода данных можно использовать конструкцию типа:

```
System.out.println(данные);
```

При этом нужно помнить, что данные соединяются знаком +, например, блок кода

```
int n=5;

System.out.println("Задача"+" "+"#" +n);
```

выведет на экран

Задача #5

и переведет курсор на следующую строчку.

Для вывода пустой строки используется та же конструкция, но без параметров.

Если перевод строки не нужен, то можно использовать конструкцию типа

```
System.out.print(данные);
```

Для ввода данных с клавиатуры необходима более сложная конструкция:

```
Scanner in = new Scanner(System.in);
int m;
m = in.nextInt();
```

В первой строчке создается объект `in` класса `Scanner` – специального класса для ввода данных, причем в конструкторе класса указывается потока ввода `System.in`, означающий, что ввод будет с клавиатуры. Далее, объявляется переменная целого типа, которой потом присваивается целое значение, введенное в поток `in` (`in.nextInt()`). Для работы данной инструкции необходимо подключить класс `java.util.Scanner` (`import java.util.Scanner;`). Вводить можно разные типы данных, используя соответствующий метод (`nextInt`, `nextShort`, `nextFloat`, `nextDouble` и т.д., для всех примитивных типов). Например, программа, считывающая с клавиатуры массив и вычисляющая среднеарифметическое из его элементов:

```
import java.util.Scanner;

public class hi {
    public static void main (String [] args){
        int n, mas[];
        float s=0;
        Scanner in = new Scanner(System.in);
        System.out.print("Введите размерность массива: ");
        n = in.nextInt();
        mas = new int[n];
        for (int i=0;i<mas.length; i++){
            System.out.print("Введите "+i+"-й"+" элемент: ");
            mas[i]=in.nextInt();
            s+=mas[i];
        }
        s/=n;
        System.out.println("Ср. арифм. массива: "+s);
    }
}
```

После запуска программы увидим:

```
Введите размерность массива: 3
Введите 0-й элемент: 3
Введите 1-й элемент: 2
Введите 2-й элемент: 2
Ср. арифм. массива: 2.3333333
```

По умолчанию компилятор выводит 7 знаков после запятой для вещественных чисел. Если нужно вывести меньше, то нужно слегка подкорректировать наш код:

```
import java.util.Scanner;
import java.text.NumberFormat;//класс для форматирования представления чисел
```

```
public class hi {
    public static void main (String [] args){
        int n, mas[];
        float s=0;
        Scanner in = new Scanner(System.in);
        System.out.print("Введите размерность массива: ");
        n = in.nextInt();
        mas = new int[n];
        for (int i=0;i<mas.length; i++){
            System.out.print("Введите "+i+"-й"+" элемент: ");
            mas[i]=in.nextInt();
            s+=mas[i];
        }
        s/=n;
        NumberFormat nf = NumberFormat.getInstance(); //создаем объект nf класса
        //NumberFormat и приравниваем его к установленному в системе формату
        //представления чисел (NumberFormat.getInstance())
    }
}
```

```

        nf.setMaximumFractionDigits(2); //устанавливаем максимальное количество цифр после
                                         //запятой равным 2
        System.out.println("Ср. арифм. массива: "+nf.format(s));
//выводим переменную s с использованием установленного формата
    }
}

```

После запуска получим:

```

Введите размерность массива: 3
Введите 0-й элемент: 3
Введите 1-й элемент: 2
Введите 2-й элемент: 2
Ср. арифм. массива: 2,33

```

Но массивы удобнее заполнять с использованием генератора случайных чисел. Для этих целей в языке Java есть класс Random. Поэтому в программе можно создать объект данного класса и использовать его метод nextInt(*верхняя граница*), выдающий случайное число из диапазона 0..*(верхняя граница – 1)*. Для работы генератора нужно подключить java.util.Random. С учетом описанного наш пример можно переписать следующим образом:

```

import java.text.NumberFormat;
import java.util.*; //подключаем весь пакет java.util

public class hi {
    public static void main (String [] args){
        int n, mas[];
        float s=0;
        Scanner in = new Scanner(System.in);
        System.out.print("Введите размерность массива: ");
        n = in.nextInt();
        mas = new int[n];
        Random random = new Random(); //создаем объект-генератор случ. чисел
        for (int i=0; i<mas.length; i++){
            mas[i]=random.nextInt(5); //генерируем очередное случ. число из нужного нам
                                     //диапазона (от 0 до 4)
            System.out.println(i+"-й" + " элемент: "+mas[i]);
            s+=mas[i];
        }
        s/=n;
        NumberFormat nf = NumberFormat.getInstance();
        nf.setMaximumFractionDigits(2);
        System.out.println("Ср. арифм. массива: "+nf.format(s));
    }
}

```

После запуска получим:

```

Введите размерность массива: 10
0-й элемент: 4
1-й элемент: 3
2-й элемент: 4
3-й элемент: 2
4-й элемент: 3
5-й элемент: 4
6-й элемент: 2
7-й элемент: 3
8-й элемент: 1
9-й элемент: 1
Ср. арифм. массива: 2,7

```

Задачи (2 из каждой части, размер массива вводит пользователь, массив генерируется рандомно):

I. Одномерные массивы.

1. Дан массив. Удалить из него нули и после каждого числа, оканчивающего на 5, вставить 1.
2. Случайным образом генерируется массив чисел. Пользователь вводит числа а и b. Заменить элемент массива на сумму его соседей, если элемент массива четный и номер его лежит в промежутке от а до b.
3. В одномерном массиве удалить промежуток элементов от максимального до минимального.
4. Дан одномерный массив. Переставить элементы массива задом-наперед.
5. Сформировать одномерный массив случайным образом. Определить количество четных элементов массива, стоящих на четных местах.
6. задается массив. Определить порядковые номера элементов массива, значения которых содержат последнюю цифру первого элемента массива 2 раза (т.е. в массиве должны быть не только однозначные числа).
7. Сформировать одномерный массив из целых чисел. Вывести на экран индексы тех элементов, которые кратны трем и пяти.

8. Задается массив. Написать программу, которая вычисляет, сколько раз введенная с клавиатуры цифра встречается в массиве.
9. Задается массив. Узнать, какие элементы встречаются в массиве больше одного раза.
10. Даны целые числа a_1, a_2, \dots, a_n . Вывести на печать только те числа, для которых $a_i \geq i$.
11. Дан целочисленный массив с количеством элементов n . Напечатать те его элементы, индексы которых являются степенями двойки.
12. Задана последовательность из N чисел. Определить, сколько среди них чисел меньших K , равных K и больших K .
13. Задан массив действительных чисел. Определить, сколько раз меняется знак в данной последовательности чисел, напечатать номера позиций, в которых происходит смена знака.
14. Задана последовательность N чисел. Вычислить сумму чисел, порядковые номера которых являются простыми числами.
15. Дан массив чисел. Указать те его элементы, которые принадлежат отрезку $[c, d]$.
16. Массив состоит из нулей и единиц. Поставить в начало массива нули, а затем единицы.
17. Дан массив целые положительные чисел. Найти среди них те, которые являются квадратами некоторого числа x .
18. В массиве целых чисел найти наиболее часто встречающееся число. Если таких чисел несколько, то определить наименьшее из них.
19. Дан целочисленный массив с количеством элементов n . Сжать массив, выбросив из него каждый второй элемент.
20. Дан массив, состоящий из n натуральных чисел. Образовать новый массив, элементами которого будут элементы исходного, оканчивающиеся на цифру k .
21. Даны действительное число x и массив $A[n]$. В массиве найти два члена, среднее арифметическое которых ближе всего к x .
22. Даны два массива A и B . Найти, сколько элементов массива A совпадает с элементами массива B .

II. Двумерные массивы.

1. Дан двумерный числовой массив. Значения элементов главной диагонали возвести в квадрат.
2. Дан двумерный массив. Поменять местами значения элементов столбца и строки на месте стыка минимального значения массива (или первого из минимальных). Например, если индекс минимального элемента (3;1), т.е. он находится на пересечении 3 строки и 1 столбца, то 3 строку сделать 1 столбцом, а 1 столбец сделать 3 строкой.
3. Дан двумерный массив. Сформировать одномерный массив только из четных элементов двумерного массива.
4. Дан двумерный массив. Найти сумму элементов массива, начиная с элемента, индексы которого вводит пользователь, и заканчивая элементом, индексы которого вводит пользователь.
5. Дан двумерный массив. Сформировать одномерный массив только из элементов двумерного массива с четной суммой индексов.
6. Дан двумерный массив. Сделать из него 2 одномерных: в одном – четные элементы двумерного массива, в другом – нечетные.
7. Вычислить сумму и число положительных элементов матрицы $A[N, N]$, находящихся над главной диагональю.
8. В квадратной матрице определить максимальный и минимальные элементы. Если таких элементов несколько, то максимальный определяется по наибольшей сумме своих индексов, минимальный – по наименьшей.
9. Для заданной квадратной матрицы сформировать одномерный массив из ее диагональных элементов.
10. Заданы матрица порядка n и число k . Вычесть из элементов k -й строки диагональный элемент, расположенный в этой строке.
11. Заданы матрица порядка n и число k . Вычесть из элементов k -го столбца диагональный элемент, расположенный в этом столбце.
12. Дана прямоугольная матрица. Найти строку с наибольшей и наименьшей суммой элементов. Вывести на печать найденные строки и суммы их элементов.
13. Дана прямоугольная матрица. Найти столбец с наибольшей и наименьшей суммой элементов. Вывести на печать найденные столбцы и суммы их элементов.
14. Дан двумерный массив. Выяснить, в каких строках сумма элементов меньше введенного пользователем значения.
15. Дан двумерный массив. Выяснить, в каких столбцах произведение элементов меньше введенного пользователем значения.
16. Дан двумерный массив. Выяснить, есть ли столбец и строка с одинаковой суммой элементов. Если есть, напечатать их номера.
17. Дан двумерный массив. Выяснить, есть ли столбец и строка с одинаковым произведением элементов. Если есть, напечатать их номера.
18. Дан двумерный массив. Выяснить, есть ли строки с одинаковой суммой элементов. Если есть, вывести их номера.
19. Дан двумерный массив. Выяснить, есть ли столбцы с одинаковой суммой элементов. Если есть, вывести их номера.
20. Дан двумерный массив. Определить максимальный среди положительных элементов, минимальный среди отрицательных элементов и поменять их местами.
21. Дан двумерный массив. Заменить первый нуль в каждом столбце на количество нулей в этом столбце.
22. Дан двумерный массив. Заменить первый нуль в каждой строке на количество нулей в этой строке.

Лабораторная работа №2. ООП в Java. Документирование кода на языке Java с использованием утилиты javadoc

Объектно-ориентированное программирование в Java похоже на ООП в C++ (см. лаб. раб. по C++), но есть свои отличия. Нет нужды в объявлении класса и его методов в одном файле, а реализации – в другом, все осуществляется в рамках одного файла.

Например, опишем класс, представляющий автомобиль:

```
public class Auto {
    private String firm; //создаем закрытый член нашего класса с названием фирмы автомобиля
    private int maxSpeed; // закрытый член класса, содержащий максимальную скорость

    public void setFirm(String firma){ //открытая функция (метод класса) для задания
        firm=firma; //значения фирмы автомобиля
    }

    public void setMaxSpeed(int speed){ //открытая функция (метод класса) для задания
        maxSpeed=speed; //значения максимальной скорости автомобиля
    }

    public int getMaxSpeed(){ //открытая функция (метод класса) для вывода значения
        return maxSpeed; //максимальной скорости
    }

    public String getFirm(){ //открытая функция (метод класса) для вывода значения
        return firm; //заданной фирмы
    }

    public Auto(){ // конструктор класса (без параметров)
        firm="Без названия";
        maxSpeed=0;
    }
    public Auto(String firma, int speed){ //конструктор класса (с параметрами)
        firm=firma;
        maxSpeed=speed;
    }
}
```

Чтобы использовать созданный класс, лучше написать другой класс для тестирования нашего класса (в том же проекте):

```
public class test {
    public static void main(String[] args) {
        Auto myAutol=new Auto("Ford",180); // создаем объект типа нашего класса
        System.out.println(myAutol.getFirm()+" "+myAutol.getMaxSpeed()); //вывод сведений в
    } // консоль
}
```

Или, с вводом данных с клавиатуры:

```
import java.util.Scanner; //подключаем класс для ввода данных с клавиатуры в консоли

public class test {
    public static void main(String[] args) {
        Auto myAutol=new Auto(); //создаем объект типа нашего класса
        Scanner in = new Scanner(System.in); //создаем сканер для ввода данных из консоли
        System.out.print("Введите фирму: ");
        String nazv=in.next(); //считываем название из консоли !!!только 1 слово
        //т.к. in.next() считывает только символы до пробела, остальные символы отправляет
        //следующему оператору, связанному с консольным вводом, для ввода строки с пробелами
        //можно использовать in.nextLine()
        myAutol.setFirm(nazv); //задаем значение для параметра нашего класса
        System.out.print("Введите максимальную скорость: ");
        int s=in.nextInt();
        myAutol.setMaxSpeed(s);
        System.out.println(myAutol.getFirm()+" "+myAutol.getMaxSpeed());
    }
}
```

Теперь на основе созданного класса легко создать классы-наследники – класс Car (легковая) и класс Truck (грузовая):

```
public class Car extends Auto{//файл Car.java
    private String model;
    private int numDoors;
    private Boolean fullTime; //полный привод
}
```



```

public Car(){
    super();// вызываем конструктор класса-родителя без параметров (см. класс Auto)
    model=""; // добавляем инициализацию новых членов
    numDoors=4;
    fullTime=false;
}
public Car(String firma, int speed, String name, int n, Boolean f){
    super(firma,speed);//вызываем конструктор класса-родителя с параметрами (см. класс Auto)
    model=name; // добавляем инициализацию новых членов
    numDoors=n;
    fullTime=f;
}

public void setModel(String name){
    model=name;
}
public String getModel(){
    return model;
}

public void setNumDoors(int n){
    numDoors=n;
}
public int getNumDoors(){
    return numDoors;
}

public void setFullTime(Boolean b){
    fullTime=b;
}
public Boolean isFullTime(){
    return fullTime;
}
public String toString(){
    return getFirm()+" "+getMaxSpeed()+" "+model+" "+numDoors+" "+fullTime;
}
}

```

//файл Truck.java

```

import java.util.Scanner;

public class Truck extends Auto{
    private String model;
    private int power;
    private Boolean trailer; //с прицепом или без

    public Truck(){
        super();
        model="";
        power=0;
        trailer=false;
    }
    public Truck(String firma, int speed, String name, int n, Boolean f){
        super(firma,speed);
        model=name;
        power=n;
        trailer=f;
    }

    public void setModel(String name){
        model=name;
    }
    public String getModel(){
        return model;
    }

    public void setPower(int n){
        power=n;
    }
    public int getPower(){
        return power;
    }

    public void setTrailer(Boolean b){
        trailer=b;
    }
}

```

```

    }
    public Boolean isTrailer(){
        return trailer;
    }

    public void setAllInfo(){
        Scanner in = new Scanner(System.in);
        System.out.print("Введите фирму-производитель грузового авто: ");
        String nazv=in.next(); //метод next() позволяет вводить строки, но без пробелов
        setFirm(nazv);
        System.out.print("Введите максимальную скорость грузового авто: ");
        int s=in.nextInt();
        setMaxSpeed(s);
        System.out.print("Введите модель грузового авто: ");
        model=in.next();
        System.out.print("Введите мощность грузового авто: ");
        power=in.nextInt();
        System.out.print("Введите признак прицепа грузового авто (true/false): ");
        trailer=in.nextBoolean();
        System.out.println();
    }
    public String toString(){
        return "\n\tГрузовик"+ "\n\t" + "Фирма: " + getFirm() + "\n\t" + "Максимальная скорость: "
        + getMaxSpeed() + " \n\t" + "Модель: " + model + "\n\t" + "Мощность: " + power + "\n\t" + "Признак прицепа: "
        + trailer + "\n";
    }
}

```

Пример с использованием созданных классов (измененный test.java):

```

import java.util.Scanner;

public class test {

    public static void main(String[] args) {
        Auto myAuto1=new Auto();
        Scanner in = new Scanner(System.in);
        System.out.print("Введите фирму: ");
        String nazv=in.next();
        myAuto1.setFirm(nazv);
        System.out.print("Введите максимальную скорость: ");
        int s=in.nextInt();
        myAuto1.setMaxSpeed(s);
        System.out.println("Какой-то автомобиль: " + myAuto1.getFirm() + " " + myAuto1.getMaxSpeed());
        System.out.println();

        Car myCar1=new Car("Ford", 200, "Mustang", 2, false);
        Car myCar2=new Car();
        System.out.print("Введите фирму-производитель легкового авто: ");
        nazv=in.next();
        myCar2.setFirm(nazv);
        System.out.print("Введите максимальную скорость легкового авто: ");
        s=in.nextInt();
        myCar2.setMaxSpeed(s);
        System.out.print("Введите модель легкового авто: ");
        nazv=in.next();
        myCar2.setModel(nazv);
        System.out.print("Введите кол-во дверей легкового авто: ");
        s=in.nextInt();
        myCar2.setNumDoors(s);
        System.out.print("Введите признак полного привода легкового авто (true/false): ");
        Boolean f=in.nextBoolean();
        myCar2.setFullTime(f);
        System.out.println();
        System.out.println("Первый легковой автомобиль: " + myCar1.toString());
        System.out.println("Второй легковой автомобиль: " + myCar2.toString());

        Truck myTruck=new Truck();
        myTruck.setAllInfo();
        System.out.println(myTruck.toString());
    }
}

```

Пример работы с программой:

Введите фирму: Lada
Введите максимальную скорость: 130
Какой-то автомобиль: Lada 130

```
Введите фирму-производитель легкового авто: Nissan
Введите максимальную скорость легкового авто: 230
Введите модель легкового авто: Patrol
Введите кол-во дверей легкового авто: 5
Введите признак полного привода легкового авто (true/false): true
```

```
Первый легковой автомобиль: Ford 200 Mustang 2 false
Второй легковой автомобиль: Nissan 230 Patrol 5 false
Введите фирму-производитель грузового авто: Kamaz
Введите максимальную скорость грузового авто: 180
Введите модель грузового авто: Masters
Введите мощность грузового авто: 400
Введите признак прицепа грузового авто (true/false): false
```

```
Грузовик:
    Фирма: Kamaz
    Максимальная скорость: 180
    Модель: Masters
    Мощность: 400
    Признак прицепа: false
```

Мы можем использовать созданные классы не только в наследовании, но и для агрегации (когда один класс содержит в себе в качестве членов объекты других классов). Например, создадим класс Garage (Гараж), описывающий кол-во и состав машин в гараже:

```
import java.util.ArrayList; //нужно для работы с классом ArrayList

public class GarageAuto {

    private ArrayList<Auto> masAuto = new ArrayList<Auto>(); //массив с машинами

    public void addAuto(Auto m) { //метод для добавления машины в гараж
        masAuto.add(m);
    }

    public GarageAuto () {

    }

    public Boolean findAuto(Auto m) { //для выяснения - есть ли машина m в гараже
        return masAuto.contains(m);
    }

    public GarageAuto(ArrayList<Auto> n) { //конструктор для внесения существующего списка машин
        //в гараж
        masAuto = n;
    }

    public void printGarage() { //для вывода на экран списка машин в гараже
        System.out.println("В гараже: ");
        for (Auto a: masAuto) { //
            System.out.println("\t"+a.toString());
        }
    }
}
```

В этом примере использован объект типа ArrayList. Этот класс предназначен для работы с массивами объектов одного типа. Т.е. по сути это другой способ представления классического массива, но с удобными методами класса ArrayList. Синтаксис объявления и создания объекта типа ArrayList:

```
ArrayList<тип данных в массиве> имя_массива=new ArrayList<тип данных в массиве>();
```

После этого объекты в ArrayList можно добавлять при помощи метода add(элемент массива).

Некоторые другие полезные методы класса ArrayList (после двоеточия указан тип возвращаемого функцией значения, void – нет возвращаемого значения):

```
contains(m): Boolean - возвращает true, если содержится элемент m, иначе false
add (int index, m) : void - добавляет элемент m в конкретную позицию index
clear() : void - удаляет все элементы из списка
get(int index) : Тип_элементов_массива - возвращает элемент, стоящий на позиции index
indexOf(m) : int - возвращает номер позиции элемента m, если есть одинаковые элементы, то первую позицию
isEmpty() : boolean - возвращает true, если массив пустой, иначе false
lastIndexOf(m) : int - возвращает номер последней позиции элемента m
remove(int index) : Тип_элементов_массива - удаляет элемент с индексом index, остальные элементы сдвигаются на позицию, сам удаленный элемент возвращается функцией
remove(m) : boolean - удаляет элемент m, возвращает true, если элемент был в списке
```

set(int index, m) : Тип_элементов_массива - заменяет элемент на позиции index на новый элемент m, старый элемент возвращается функцией
size() : int - кол-во элементов, содержащихся в массиве
subList(int i1, int i2) : List<Тип_элементов_массива> - выдает список элементов от элемента с номером i1 до номера i2
toArray() : Object[] - преобразует в обычный массив

В нашем примере создается массив объектов типа Auto, т.е. мы сможем в нем хранить как элементы типа Car, так и элементы типа Truck.

Если посмотреть на цикл for, использованный в методе printGarage(), то можно увидеть, что он используется не совсем обычно:

```
for (Auto a:masAuto){  
    System.out.println("\t"+a.toString());  
}
```

Этот цикл носит название «для каждого» (foreach). В качестве параметра цикла выступает переменная a типа Auto, при этом указывается, что переменная a будет каждую итерацию цикла заменяться элементом из masCar, который является представителем класса ArrayList, т.е. для каждого элемента из созданного массива будет выполняться действие в теле цикла. Этот цикл удобно использовать, если мы не хотим зависеть от размеров массива (класса ArrayList).

Напишем программу с использованием созданного гаража:

```
public class testGarage {  
  
    public static void main(String[] args) {  
        GarageAuto myGarage=new GarageAuto(); //создаем новый гараж  
        Car myCar1=new Car("Ford", 200,"Mustang",2,false); //создаем легковую машину  
        myGarage.addAuto(myCar1); // добавляем ее в гараж  
        myGarage.addAuto(new Car("LADA", 140, "Kalina", 4,false)); //добавляем еще одну машину  
        Truck myTruck=new Truck("Dove",160,"DTS",700,true); //создаем грузовик  
        myGarage.addAuto(myTruck); //добавляем его в гараж  
        myGarage.printGarage(); //выводим на экран содержимое гаража  
        if (myGarage.findAuto(myCar1)) { //ищем машину  
            System.out.println("Да");  
        }  
        else {  
            System.out.println("Нет");  
        }  
    }  
}
```

Пример работы программы:

В гараже:

```
Ford 200 Mustang 2 false  
LADA 140 Kalina 4 false
```

```
Грузовик  
Фирма: Dove  
Максимальная скорость: 160  
Модель: DTS  
Мощность: 700  
Признак прицепа: true
```

Да

Есть еще один интересный аспект при работе с классом ArrayList. Можно узнать класс объекта, который является текущим при обработке в цикле, для этого используется оператор instanceof – оператор сравнения на принадлежность к определенному классу или типу, т.е. можно написать

```
for (Auto a:masAuto){  
    if (a instanceof Car) {System.out.println("Это легковая машина");}
```

и текст *Это легковая машина* будет выведен столько раз, сколько содержится объектов типа Car в нашем массиве, то же самое можно сделать и для объектов типа Truck. Оператор instanceof понадобится для решения задач.

В среду программирования Eclipse встроена поддержка утилиты javadoc для языка Java. Утилита javadoc входит в состав JDK – Java Development Kit (комплект разработчика приложений на языке Java). Она позволяет генерировать html-страницы с документацией созданных классов, если код этих классов размечен специальными комментариями. Эти комментарии должны начинаться со знаков /**, а заканчиваться */

Внутри спецкомментариев допустимы следующие специальные дескрипторы:

Список дескрипторов <i>Javadoc</i>		
Дескриптор	Описание	Применим к
@author	Автор	класс, интерфейс
@version	Версия. Не более одного дескриптора на класс	класс, интерфейс
@since	Указывает, с какой версии доступно	класс, интерфейс, поле, метод
@see	Ссылка на другое место в документации	класс, интерфейс, поле, метод
@param	Входной параметр метода	метод
@return	Описание возвращаемого значения	метод
@exception <i>имякласса</i> описание @throws <i>имякласса</i> описание	Описание исключения, которое может быть обработано внутри метода	метод
@deprecated	Описание устаревших блоков кода	метод
{@link reference}	Ссылка	класс, интерфейс, поле, метод
{@value}	Описание значения переменной	статичное поле

Например, ранее созданный класс Auto можно было бы задокументировать следующим образом:

```
/**
 * Класс Автомобиль - базовый класс для объектов транспорта
 * @author Слива М.В.
 */
public class Auto {
    /**Поле для хранения названия фирмы автомобиля */
    private String firm;

    /**Поле для хранения максимальной скорости автомобиля */
    private int maxSpeed;

    /**
     * Устанавливает значение поля {@link Auto#firm}
     * @param firma - название фирмы автомобиля */
    public void setFirm(String firma){
        firm=firma;
    }

    /**
     * Устанавливает значение поля {@link Auto#maxSpeed}
     * @param speed - значение максимальной скорости автомобиля */
    public void setMaxSpeed(int speed) {
        maxSpeed=speed;
    }

    /**
     * Возвращает значение поля {@link Auto#maxSpeed}
     * @return целое значение максимальной скорости автомобиля */
    public int getMaxSpeed(){
        return maxSpeed;
    }

    /**
     * Возвращает значение поля {@link Auto#firm}
     * @return строку с названием фирмы автомобиля */
    public String getFirm(){
        return firm;
    }

    /**
     * Создает автомобиль с фирмой "Без названия" и максимальной скоростью, равной 0*/
```

```

public Auto() {
    firm="Без названия";
    maxSpeed=0;
}
/**
 * Создает автомобиль с заданными значениями фирмы и максимальной скорости
 * @param firma - название фирмы автомобиля
 * @param speed - значение максимальной скорости автомобиля*/
public Auto(String firma, int speed){
    firm=firma;
    maxSpeed=speed;
}
}

```

Созданные описания можно предварительно посмотреть во вкладке Javadoc нижней панели среды Eclipse (рис. 1):

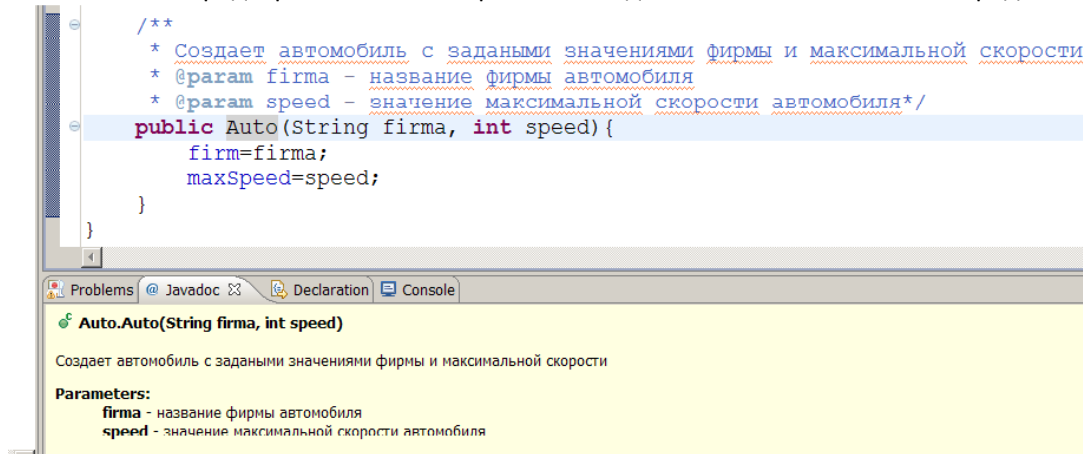


Рис. 1. Предварительный просмотр созданной документации.

Для этого нужно просто установить курсор на названии описанного метода, поля или класса.

Чтобы сгенерировать документацию по классу в виде html-страниц, нужно выбрать команды меню Project\Generate Javadoc. Появится диалоговое окно с параметрами (рис. 2.1), в котором нужно выбрать в своем проекте классы, для которых нужно сгенерировать документацию (**поставить галочки возле нужных файлов!**), выбрать путь к утилите Javadoc, если он не выбран (как правило, это что-то похожее на C:\Program Files\Java\jdk1.6.0_27\bin\javadoc.exe), и путь для сохранения документации (по умолчанию, это текущий проект). **И выбрать Private в пункте Create Javadoc for members with visibility.** Это позволит в документации осуществлять переход к полям класса, которые являются private.

После этого нажимаем Next, в следующем окне тоже Next, потом в итоговом окне (рис. 2.2) добавляем в поле VM options строку -encoding utf8 для правильной кодировки кириллических символов и жмем Finish.

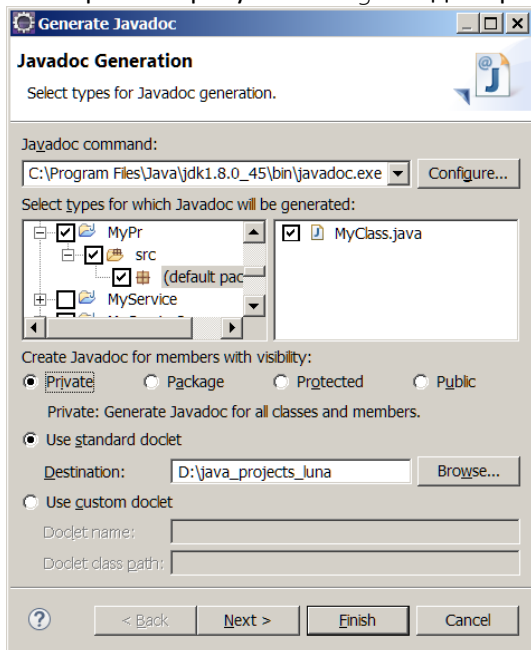


Рис. 2.1. Начальное диалоговое окно Project\Generate Javadoc.

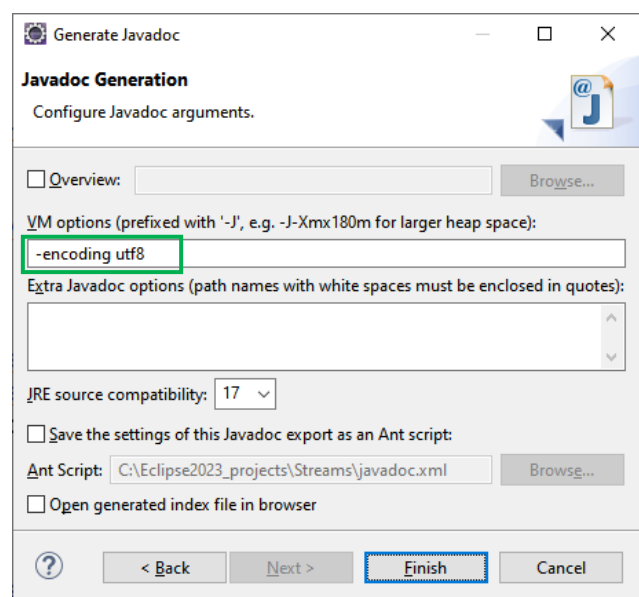


Рис. 2.2. Итоговое диалоговое окно Project\Generate Javadoc.

Каталог с документацией будет примерно следующего вида (рис. 3):

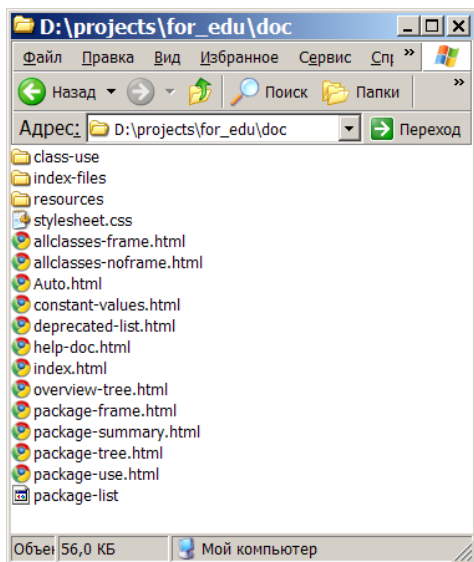


Рис. 3. Каталог с созданной документацией.

И, запустив файл `index.html`, в браузере можно увидеть документацию стандартного для Java вида (рис. 4):

Class Auto
`java.lang.Object`
 Auto

public class Auto
 extends `Object`

Класс Автомобиль - базовый класс для объектов транспорта

Author:
 Слива М.В.

Field Summary

Fields	Modifier and Type	Field	Description
	private	<code>String</code> <code>firm</code>	Поле для хранения названия фирмы автомобиля
	private	<code>int</code> <code>maxSpeed</code>	Поле для хранения максимальной скорости автомобиля

Constructor Summary

Constructors	Constructor	Description
	<code>Auto()</code>	Создает автомобиль с фирмой "Без названия" и максимальной скоростью.
	<code>Auto(String firm, int speed)</code>	Создает автомобиль с заданными значениями фирмы и максимальной скор

Method Summary

All Methods	Instance Methods	Concrete Methods	Modifier and Type	Method	Description
			<code>String</code>	<code>getFirm()</code>	Возвращает значение поля <code>firm</code>

Рис. 4. Документация в сгенерированном файле.

Задания:

- По вариантам (создать классы, в них предусмотреть различные члены классов и методы для работы):**
 - Базовый класс – учащийся. Производные – школьник и студент. Создать класс Конференция, который может содержать оба вида учащихся. Предусмотреть метод подсчета участников конференции отдельно по школьникам и по студентам (использовать оператор `instanceof`).
 - Базовый класс – работник. Производные – работник на почасовой оплате и на окладе. Создать класс Предприятие, который может содержать оба вида работников. Предусмотреть метод подсчета работников отдельно на почасовой оплате и на окладе (использовать оператор `instanceof`).
 - Базовый класс – компьютер. Производные – ноутбук и смартфон. Создать класс РемонтСервис, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно ремонтируемых ноутбуков и смартфонов (использовать оператор `instanceof`).
 - Базовый класс – печатные издания. Производные – книги и журналы. Создать класс КнижныйМагазин, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно книг и журналов (использовать оператор `instanceof`).
 - Базовый класс – помещения. Производные – квартира и офис. Создать класс Дом, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно квартир и офисов (использовать оператор `instanceof`).
 - Базовый класс – файл. Производные – звуковой файл и видео-файл. Создать класс Каталог, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно звуковых и видео-файлов (использовать оператор `instanceof`).
 - Базовый класс – летательный аппарат. Производные – самолет и вертолет. Создать класс Авиакомпания, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно самолетов и вертолетов (использовать оператор `instanceof`).
 - Базовый класс – соревнование. Производные – командные соревнования и личные. Создать класс Чемпионат, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно командных соревнований и личных (использовать оператор `instanceof`).
 - Базовый класс – мебель. Производные – диван и шкаф. Создать класс Комната, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно диванов и шкафов (использовать оператор `instanceof`).

10. Базовый класс – оружие. Производные – огнестрельное и холодное. Создать класс ОружейнаяПалата, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно огнестрельного и холодного оружия (использовать оператор `instanceof`).
 11. Базовый класс – оргтехника. Производные – принтер и сканер. Создать класс Офис, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно принтеров и сканеров (использовать оператор `instanceof`).
 12. Базовый класс – СМИ. Производные – телеканал и газета. Создать класс Холдинг, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно телеканалов и газет (использовать оператор `instanceof`).
 13. Свой вариант по согласованию с преподавателем (можно по теме курсовой).
- II. Сделать документацию по всем созданным в своем варианте классам.**

Лабораторная работа №3. Коллекции языка Java. Stream API

В Java существует множество коллекций, все они образуют стройную и логичную систему. В основе всех коллекций лежит применение того или иного интерфейса, который определяет базовый функционал:

- **Collection**: базовый интерфейс для всех коллекций и других интерфейсов коллекций, расширяет интерфейс `Iterable`, а у этого интерфейса есть единственный метод `iterator()`
- **Queue**: наследует интерфейс `Collection` и представляет функционал для структур данных в виде очереди
- **Deque**: наследует интерфейс `Queue` и представляет функционал для двунаправленных очередей
- **List**: наследует интерфейс `Collection` и представляет функциональность простых списков
- **Set**: также расширяет интерфейс `Collection` и используется для хранения множеств уникальных объектов
- **SortedSet**: расширяет интерфейс `Set` для создания сортированных коллекций
- **NavigableSet**: расширяет интерфейс `SortedSet` для создания коллекций, в которых можно осуществлять поиск по соответствию
- **Map**: предназначен для созданий структур данных в виде словаря, где каждый элемент имеет определенный ключ и значение. Не наследуется от интерфейса `Collection`

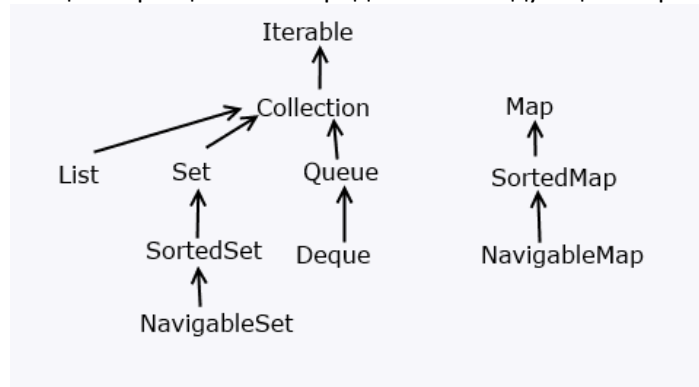
Эти интерфейсы частично реализуются абстрактными классами:

- **AbstractCollection**: базовый абстрактный класс для других коллекций, который применяет интерфейс `Collection`
- **AbstractList**: расширяет класс `AbstractCollection` и применяет интерфейс `List`, предназначен для создания коллекций в виде списков
- **AbstractSet**: расширяет класс `AbstractCollection` и применяет интерфейс `Set` для создания коллекций в виде множеств
- **AbstractQueue**: расширяет класс `AbstractCollection` и применяет интерфейс `Queue`, предназначен для создания коллекций в виде очередей и стеков
- **AbstractSequentialList**: также расширяет класс `AbstractList` и реализует интерфейс `List`. Используется для создания связанных списков
- **AbstractMap**: применяет интерфейс `Map`, предназначен для создания наборов по типу словаря с объектами в виде пары "ключ-значение"

С помощью применения вышеописанных интерфейсов и абстрактных классов в Java реализуется широкая палитра классов коллекций - списки, множества, очереди, отображения и другие:

- **ArrayList**: простой список объектов
- **LinkedList**: представляет связанный список
- **ArrayDeque**: класс двунаправленной очереди, в которой мы можем произвести вставку и удаление как в начале коллекции, так и в ее конце
- **HashSet**: набор объектов или хеш-множество, где каждый элемент имеет ключ - уникальный хеш-код
- **TreeSet**: набор отсортированных объектов в виде дерева
- **LinkedHashSet**: связанное хеш-множество
- **PriorityQueue**: очередь приоритетов
- **HashMap**: структура данных в виде словаря, в котором каждый объект имеет уникальный ключ и некоторое значение
- **TreeMap**: структура данных в виде дерева, где каждый элемент имеет уникальный ключ и некоторое значение

Схематично всю систему коллекций вкратце можно представить следующим образом:



С классом `ArrayList` мы уже работали в предыдущей лабораторной. Разберем другие классы. Начнем с класса `HashSet`. Его конструктор выглядит следующим образом:

```
Set<Obj> set = new HashSet<Obj>();
```

где `Obj` – это класс объектов, которые будут храниться в множестве.

Например, следующий код создает множество из целых чисел:

```
Set<Integer> set = new HashSet<Integer>();
set.add(1);
set.add(2);
set.add(3);
set.add(1);
set.add(2);
System.out.println(set);
```

В консоль будет выведено

[1, 2, 3]

что логично, учитывая, что в множестве не может быть повторений.

При заполнении множества при помощи генератора случайных чисел нужно также учитывать вышеозвученный факт. Например, код:

```
Set<Integer> set = new HashSet<Integer>();
Random rand = new Random();
for(int i=0; i<5; i++) {
    set.add(rand.nextInt(0, 20));
}
System.out.println(set);
```

может вывести в консоль

[3, 19, 5, 11]

т.е. 4 числа вместо 5, в зависимости от выданных генератором случайных чисел (хотя 5 чисел тоже можно получить, если повезет).

Соответственно, чтобы заполнить множество нужным количеством значений, удобнее использовать следующий код:

```
Set<Integer> set = new HashSet<Integer>();
Random rand = new Random();
while (set.size()<5) {
    set.add(rand.nextInt(0, 20));
}
System.out.println(set);
```

Также необходимо учитывать особенность добавления в множество объектов созданных нами классов:

```
class Person {
    String name;

    Person(String name) { this.name = name; }

    public String toString() { return name; }
}

public class ex1 {
    public static void main(String[] args) {
        HashSet<Person> set = new HashSet<>();
        Person p1 = new Person("Иван");
        Person p2 = new Person("Мария");
        Person p3 = new Person("Пётр");
        Person p4 = new Person("Мария");
        set.add(p1);
        set.add(p2);
        set.add(p3);
        set.add(p4);
        System.out.println(set);
        HashSet<String> set2 = new HashSet<>();
        String s1 = new String("Иван");
        String s2 = new String("Мария");
        String s3 = new String("Пётр");
        String s4 = new String("Мария");
        set2.add(s1);
        set2.add(s2);
        set2.add(s3);
        set2.add(s4);
        System.out.println(set2);
    }
}
```

При выполнении данного кода в консоль будет выведено

[Пётр, Мария, Иван, Мария]

[Пётр, Мария, Иван]

Т.е. в первое множество добавился объект с существующим значением, а во второе нет. Почему?

Потому что при добавлении объекта нашего созданного класса его хеш сравнивается с хешами уже имеющихся в множестве объектов. а сам хеш вычисляется из адреса объекта, а он у всех разный, даже если все поля одинаковые. А в строках хеш вычисляется по хранимому значению, т.е. у строк с одинаковым текстом хеш одинаковый. В этом легко убедиться, добавив в конце предыдущего кода следующие строки:

```
System.out.println("p2 hash = "+p2.hashCode());
System.out.println("p4 hash = "+p4.hashCode());
System.out.println("s2 hash = "+s2.hashCode());
System.out.println("s4 hash = "+s4.hashCode());
```

В консоль будет выведено

```
p2 hash = 1365202186
p4 hash = 1586600255
s2 hash = 1004560195
s4 hash = 1004560195
```

Но мы можем и к своему классу добавить такую фишку – равенство хешей при равенстве значений полей объектов. Для этого добавим к нашему классу в конце следующий код:

```
@Override
public int hashCode() {
    return name.hashCode();
}

@Override
public boolean equals(Object o) {
    return true;
}
```

Этим кодом мы перегружаем стандартный метод высчитывания хеша, делая его равным хешу поля со строкой, и метод сравнения объектов.

И теперь, если запустить проект, в консоли работа множества с объектами нашего класса будет аналогична работе со стандартными строками:

```
[Пётр, Мария, Иван]
[Пётр, Мария, Иван]
p2 hash = 1004560195
p4 hash = 1004560195
s2 hash = 1004560195
s4 hash = 1004560195
```

И хеши наших объектов с одинаковыми строками тоже теперь равны.

`boolean add(element)` – добавляет элемент в мн-во. Возвращает `true`, если объект добавлен, и `false`, если нет (т.е. уже есть такой объект в мн-ве).

`addAll(collection)` – добавляет все объекты из указанной коллекции в мн-во.

`clear()` – удаляет все эл-ты из мн-ва.

`boolean contains(element)` – проверяет наличие указанного эл-та в мн-ве.

`containsAll(collection)` – проверяет наличие всех эл-ов из указанной коллекции в мн-ве. Вернет `true`, только если все эл-ты коллекции присутствуют в мн-ве, иначе вернет `false`.

`isEmpty()` – возвращает `true`, если мн-во пустое.

`iterator()` – возвращает итератор для мн-ва.

`remove(element)` – удаляет указанный объект из мн-ва. Возвращает `true`, если объект был во мн-ве, и `false`, если нет.

`removeAll(collection)` – удаляет из мн-ва все объекты указанной в параметрах коллекции. Возвращает `true`, если что-то было удалено, и `false`, если нет.

`retainAll(collection)` – пересечение с указанной коллекцией. Возвращает `true`, если мн-во было изменено, и `false`, если нет.

`size()` – возвращает размер мн-ва (кол-во эл-ов).

`toArray()` – возвращает обычный массив из эл-ов мн-ва.

Рассмотрим небольшой пример для демонстрации работы метода `retainAll`:

```
Set<Integer> intSet1 = new HashSet<>();
for (int i = 0; i < 6; i++)
    intSet1.add(i);
Set<Integer> intSet2 = new HashSet<>();
for (int i = 3; i < 8; i++)
    intSet2.add(i);
System.out.println("set1 = "+intSet1);
System.out.println("set2 = "+intSet2);
intSet1.retainAll(intSet2);
System.out.println("set1 = "+intSet1);
```

В консоль будет выведено

```
set1 = [0, 1, 2, 3, 4, 5]
set2 = [3, 4, 5, 6, 7]
set1 = [3, 4, 5]
```

Далее разберем класс HashMap. Этот класс представляет собой хранилище связанных пар <ключ, значение>. Т.е. по ключу можно найти нужное значение. Например, создадим хранилище связанных пар <Фамилия, Номер телефона>:

```
import java.util.HashMap;

public class SampleHash {
    public static void main(String[] args) {
        //создадим объект типа HashMap, в котором оба значения будут строкового типа
        HashMap<String, String> myHash=new HashMap<String, String>();
        myHash.put("Иванов", "891234567"); //записываем телефон Иванова
        myHash.put("Петров", "892233345"); //записываем телефон Петрова
        myHash.put("Сидоров", "893137567"); //записываем телефон Сидорова
        System.out.println(myHash); //выводим на экран все записи
        System.out.println(myHash.get("Иванов")); //выводим значение, связанное с Ивановым
    }
}
```

При запуске получим:

```
{Иванов=891234567, Петров=892233345, Сидоров=893137567}
891234567
```

Еще один пример – связанные пары <Город, Кол-во населения>:

```
HashMap<String, Long> cities = new HashMap<>();
cities.put("Москва", (long) 13149803);
cities.put("Санкт-Петербург", (long) 5597763);
cities.put("Ханты-Мансийск", (long) 111772);
cities.put("Нижневартовск", (long) 283256);
System.out.println(cities);
```

В консоль будет выведено

```
{Нижневартовск=283256, Москва=13149803, Ханты-Мансийск=111772, Санкт-Петербург=5597763}
```

Можно сделать вывод значений в нужном нам виде:

```
for (String city : cities.keySet() ) {
    System.out.println(city+": "+cities.get(city));
}
```

В консоль будет выведено

```
Нижневартовск: 283256
Москва: 13149803
Ханты-Мансийск: 111772
Санкт-Петербург: 5597763
```

А можно для тех же целей использовать лямбда-выражения. Лямбда-выражения – это анонимные функции. Проще говоря, это метод без объявления, т.е. без модификаторов доступа, возвращающие значение и имя.

Они позволяют написать метод и сразу же использовать его. Особенно полезно в случае однократного вызова метода, т.к. сокращает время на объявление и написание метода без необходимости создавать класс. Особенность лямбда-выражений в том, что их можно передавать в другие методы в качестве аргумента.

Основу лямбда-выражения составляет лямбда-оператор, который представляет стрелку ->. Этот оператор разделяет лямбда-выражение на две части: левая часть содержит список параметров выражения, а правая собственно представляет тело лямбда-выражения, где выполняются все действия.

Лямбда-выражение не выполняется само по себе, а образует реализацию метода, определенного в функциональном интерфейсе. При этом важно, что функциональный интерфейс должен содержать только один единственный метод без реализации.

Применительно к коллекциям, можно привести несколько примеров использования лямбда-выражений:

```
List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7); // список
System.out.println("list: ");
// Старый способ вывода на экран эл-ов списка:
for(Integer a: list) {
    System.out.print(a+" ");
}
System.out.println();
System.out.println("list: ");
// Новый способ:
list.forEach(a -> System.out.print(a+" "));
```

В консоль будет выведено

```
list:
1 2 3 4 5 6 7
```

```
list:
1 2 3 4 5 6 7
```

Т.е. оба способа идентичны. Можно использовать многострочный код внутри лямбда-выражения:

```
list.forEach(a -> {
    System.out.print(a);
    System.out.print(";");
});
```

В консоль будет выведено

```
1;2;3;4;5;6;7;
```

Если вернуться к выводу значений, хранящихся в HashMap, то вместо кода

```
for (String city : cities.keySet() ) {
    System.out.println(city+": "+cities.get(city));
}
```

можно использовать лямбда-выражение

```
cities.forEach( (k,v) -> System.out.println("Город " + k + " с населением " + v));
```

где (k,v) – это и есть пары ключ-значение, хранящиеся в HashMap. И встроенный цикл forEach при помощи лямбда-выражения проходит по таким парам и выводит их в нужном формате.

Попробуем сделать что-то более полезное. Выберем из HashMap с городами все объекты, отвечающие некоторому условию, например, все города с населением больше 300000:

```
cities.forEach( (k,v) -> {
    if (v > 300000) System.out.println("Город " + k + " с населением " + v);
});
```

В консоль будет выведено

```
Город Москва с населением 13149803
```

```
Город Санкт-Петербург с населением 5597763
```

И еще один вариант такой выборки:

```
cities.entrySet().stream().filter(k-> k.getValue().longValue() > 300000).forEach(System.out::println);
```

В консоли будет

```
Москва=13149803
```

```
Санкт-Петербург=5597763
```

В этот раз используется еще одна возможность ~~улучшить свою жизнь~~ современной Java – Stream (потокковая обработка коллекций). Расшифруем всю строку по частям:

cities.entrySet() – получаем набор пар значений из HashMap в виде множества;

.stream() – организуем из этого набора поток значений;

.filter(k-> k.getValue().longValue() > 300000) – фильтруем его: берем только пары, у которых поле с значением населения больше 300000;

.forEach(System.out::println) – для полученных после фильтра значений организуем цикл с вызовом для каждой пары метода println системного потока вывода System.out.

На самом деле каждая рассмотренная часть возвращает отдельный объект, который поступает на вход следующей части.

Stream API состоит из набора компонентов и концепций, которые работают вместе, чтобы обеспечить потоковую обработку данных:

- Источник (Source) – откуда приходят данные. Это может быть коллекция, массив, файл, генератор или любой другой источник данных.
- Операции – преобразовывают и/или обрабатывают данные.
- Поток (Stream) – последовательность элементов, подлежащих параллельной или последовательной обработке.
- Пайплайн (Pipeline) – последовательность операций в потоке, применяемых к данным.
- Терминал (Terminal) – место выхода данных из потока. Терминальная операция означает окончание обработки потока и возвращает результат.

Основные промежуточные методы Stream API:

- filter – Этот метод используется для создания нового потока, включающего только элементы, которые удовлетворяют определенному условию. В качестве аргумента метод принимает функциональный интерфейс Predicate (условие), задающий условие фильтрации. Как в нашем примере
.filter(k-> k.getValue().longValue() > 300000) – фильтруем все эл-ты k, такие что значения в парах «ключ-значение» больше 300000.

Можно отдельно сохранить результат выполнения потока с фильтром, например:

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
Stream<Integer> evenNumbersStream = numbers.stream().filter(n -> n % 2 == 0);
evenNumbersStream.forEach(a-> System.out.print(a+", ")); // печатает 2, 4, 6, 8, 10
```

- `map()` – Метод принимает в качестве аргумента функциональный интерфейс `Function`, задающий преобразование, применяемое к каждому элементу. Возвращаемый поток содержит преобразованные элементы.

Метод `map()` возвращает новый поток. Он не изменяет исходный поток и коллекцию. Обычно он используется для выполнения операций, таких как преобразование элементов из одного типа в другой. Например:

```
List<String> words = Arrays.asList("apple", "banana", "orange", "peach");
// заменяем строки количеством символов в строке, используя короткую запись лямбды (String::length),
// так называемую ссылку на метод
Stream<Integer> lengthsStream = words.stream().map(String::length);
lengthsStream.forEach(System.out::println); // печатает 5 6 6 5
```

Можно использовать для вывода данных свой метод:

```
public class StreamWork {
    public static void main(String[] args) {
        List<String> words = Arrays.asList("apple", "banana", "orange", "peach");
        Stream<Integer> lengthsStream = words.stream().map(String::length);
        lengthsStream.forEach(StreamWork::myPrint); // вызываем свой метод для вывода каждого эл-та
    }

    public static void myPrint(int a) {
        System.out.print(a+" ");
    }
}
```

- `distinct()` – возвращает новый поток, содержащий только уникальные элементы исходного потока. Дубликаты определяются на основе их естественного порядка или с использованием переданного компаратора.

Пример, в котором функция `distinct()` используется для удаления дубликатов из потока целых чисел:

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 2, 1, 4, 5, 3, 5);
List<Integer> uniqueNumbers = numbers.stream().distinct().toList();
System.out.println(uniqueNumbers); // печатает [1, 2, 3, 4, 5]
```

- `limit(n)` – возвращает новый поток, содержащий не более `n` элементов исходного потока. Если исходный поток содержит меньше `n` элементов, новый поток будет содержать все элементы исходного потока.

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
List<Integer> limitedNumbers = numbers.stream().limit(4).toList();
System.out.println(limitedNumbers); // печатает [1, 2, 3, 4]
```

- `skip(n)` – возвращает новый поток, который содержит все элементы исходного потока, исключая первые `n` элементов. Если исходный поток содержит меньше `n` элементов, новый поток будет пустым.

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
List<Integer> withoutSkippedNumbers = numbers.stream().skip(5).toList();
System.out.println(withoutSkippedNumbers); // печатает [6, 7, 8, 9, 10]
```

- `sorted()` – создает новый поток, содержащий элементы исходного потока, отсортированные в порядке возрастания.

```
List<Integer> numbers = Arrays.asList(9, 2, 3, 1, 5, 8);
System.out.println(numbers.stream().sorted().toList()); // печатает [1, 2, 3, 5, 8, 9]
```

Если элементы исходного потока не реализуют интерфейс `Comparable`, может возникнуть исключение `ClassCastException`. Чтобы избежать этого, можно предоставить собственный компаратор в качестве аргумента метода `sorted()`. Например, следующий код вызовет ошибку:

```
class Person {
    String name;
    int age;
    Person(String name, int age) { this.name = name; this.age = age; }
    public String toString() { return name+" "+age; }
}

public class StreamWork {
    public static void main(String[] args) {
        List<Person> people = Arrays.asList( new Person("Иван", 25 ),
                                             new Person("Мария", 35), new Person("Андрей", 30) );
        System.out.println(people.stream().sorted().toList());
    }
}
```

В консоли будет

```
Exception in thread "main" java.lang.ClassCastException: class Person cannot be cast to class
java.lang.Comparable
```

Для устранения ошибки мы можем передать в метод `sorted` правило сравнения объектов нашего класса:

```
System.out.println(people.stream().sorted( (p1, p2) -> ((Integer)p1.age).compareTo(p2.age) ).toList());
```


Само правило `(p1, p2) -> ((Integer)p1.age).compareTo(p2.age)` представляет из себя лямбда-выражение, в котором участвуют 2 объекта (`p1` и `p2`), типы которых не указываются, т.к. они берутся из потока. После стрелки указывается, как сравнивать объекты – по полю `age`, при этом первое сравниваемое значение приводим к типу `Integer`, чтобы вызвать метод `compareTo` (у обычного типа `int` такого метода нет, у него вообще никаких методов нет). Результатом будет вывод в консоль отсортированного по возрасту списка:
[Иван 25, Андрей 30, Мария 35]

Можно сделать сортировку по убыванию, просто поменяв `p1` и `p2` местами (в выражении до стрелки).

Также можно сделать сортировку по имени:

```
System.out.println(people.stream().sorted((p1, p2)->(p1.name.compareTo(p2.name))).toList());
```

Также можно упростить описание правила сравнения, если добавить в наш класс `Person` геттеры для полей `age` и `name`:

```
class Person {
    String name;
    int age;
    Person(String name, int age) { this.name = name; this.age = age; }
    int getAge() {return age;}
    String getName() {return name; }
    public String toString() { return name+" "+age; }
}
```

Тогда оба метода сортировки можно представить следующим образом:

```
System.out.println(people.stream().sorted( Comparator.comparing(Person::getAge) ).toList());
System.out.println(people.stream().sorted( Comparator.comparing(Person::getName) ).toList());
```

Т.е. мы используем в качестве выражения сравнения специальный класс `Comparator`, у которого вызываем метод `comparing` с указанием геттера нужного поля, по которому мы хотим сделать сортировку.

- `takeWhile(условие в виде лямбды)` – создает новый поток, содержащий элементы исходного потока до тех пор, пока они удовлетворяют указанному условию. Если первый элемент потока не соответствует предикату, новый поток будет пустым.

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
List<Integer> takenNumbers = numbers.stream().takeWhile(a -> a < 5).toList();
System.out.println(takenNumbers); // выводит [1, 2, 3, 4]
```

- `dropWhile(условие в виде лямбды)` – возвращает новый поток, который включает все элементы исходного потока, начиная с первого элемента, не удовлетворяющего указанному условию. В момент, когда предикат возвращает `false`, все последующие элементы из исходного потока включаются в новый поток.

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
List<Integer> takenNumbers = numbers.stream().dropWhile(a -> a < 5).toList();
System.out.println(takenNumbers); // выводит [5, 6, 7, 8, 9, 10]
```

- `reduce()` – применяется для комбинирования элементов потока в одно значение. Например, для суммирования всех эл-ов списка:

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
int sum = numbers.stream().reduce((a, b) -> a + b).get();//получаем сумму всех эл-ов
System.out.println(sum);
```

В консоль будет выведено 15.

- `boolean allMatch(условие в виде лямбды)` – возвращает `true`, если все элементы потока удовлетворяют предикату. Например, проверка, что все эл-ты списка положительны:

```
List<Integer> numbers = Arrays.asList(1, 2, -3, 4, 5);
boolean result = numbers.stream().allMatch(a -> (a>0));
System.out.println(result); // выведет false
```

Рассмотрим пример работы с коллекциями и `Stream API` на примере классов из Лаб.раб.№2. Добавим к классу `GarageAuto` метод `getMas`:

```
public ArrayList<Auto> getMas() {
    return masAuto; // возвращает массив
}
```

И теперь в запускном классе можем написать следующий код:

```
//ищем легковой автомобиль с 2 дверьми
ArrayList<Auto> mas = myGarage.getMas();
List<?> res = mas.stream().filter(a -> (a instanceof Car && ((Car) a).getNumDoors()==2)).toList();
System.out.println("Подходящие объекты: ");
res.stream().forEach(t -> System.out.println(t));
```

В консоль будет выведено:

Подходящие объекты:

Car [model=Mustang, numDoors=2, fullTime=false, MaxSpeed=200, Firm=Ford]

Задания

I. Задачи на класс Set (2 штуки через 5):

1. Сформировать множество, элементами которого являются буквы от a до f и от x до z. Требуется ввести с клавиатуры некую последовательность символов, и выяснить, какие из них входят в заданное множество.
2. Сформировать множество, содержащее натуральные числа из некоторого диапазона. Сформировать два множества, первое из которых содержит все четные числа из данного множества, а второе — все нечетные.
3. Сформировать множество из строчных латинских букв. Определить, каких букв — гласных (a, e, i, o, u) или согласных — больше.

Дан текст из строчных латинских букв, за которым следует точка. Напечатать (4-6):

4. все буквы, входящие в текст не менее двух раз;
5. все буквы, входящие в текст по одному разу;
6. первые вхождения букв в текст, сохраняя их исходный взаимный порядок.
7. Дана последовательность целых чисел. Определить, является ли эта последовательность перестановкой заданного отрезка элементов натурального ряда.
8. Подсчитать количество чётных цифр в исходной символьной строке и распечатать все, кроме пробелов, знаков операций и знаков препинания.
9. Сформировать множество, в которое входят только латинские буквы, встретившиеся во входной строке, и множество знаков препинания из входной строки.
10. Сформировать множество, в которое входят только цифры, встретившиеся во входной строке.
11. Сформировать множество, в которое входят только большие латинские буквы, встретившиеся во входной строке.
12. Дан текст на русском языке. Напечатать в алфавитном порядке все гласные буквы, которые входят в каждое слово.
13. Дан текст на русском языке. Напечатать в алфавитном порядке все звонкие согласные буквы, которые входят более чем в одно слово.

II. Задачи на класс HashMap (2 штуки через 5):

1. Задать хешмап вида (английское слово)-(набор русских слов), т.е. словарь для перевода. Организовать перевод введенного пользователем с консоли слова.
2. Задать хешмап вида (страна)-(кол-во населения). Организовать поиск стран с количеством населения, меньшим и большим значения, введенного пользователем.
3. Задать хешмап вида (группа)-(список студентов). Организовать поиск группы, где больше всего студентов на определённую букву.
4. Задать хешмап вида (человек)-(возраст). Организовать поиск людей с определенным возрастом, введенным пользователем.
5. Задать хешмап вида (дата)-(список покупок). Определить даты, в которые среди покупок были товары, введенные пользователем.
6. Задать хешмап вида (вид деят-ти)-(список фирм). Определить фирмы, которые соответствуют нескольким видам деятельности, введенным пользователем.
7. Задать хешмап вида (магазин)-(список товаров). Найти магазины, у которых в списке кол-во товаров больше числа, введенного пользователем.
8. Задать хешмап вида (преподаватель)-(список предметов). Найти преподавателей, у которых в списке есть предметы, введенные пользователем.
9. Задать хешмап вида (человек)-(место работы). Организовать поиск людей с местом работы, введенным пользователем. Посчитать людей, работающих в одном месте.
10. Задать хешмап вида (каталог)-(список его файлов). Организовать поиск нужного файла с выводом названия его каталога.
11. Задать хешмап вида (сайт)-(кол-во страниц). Отсортировать по кол-ву страниц. Вывести сайты, у которых кол-во страниц меньше введенного пользователем числа.
12. Задать хешмап вида (актер)-(список его фильмов). Организовать поиск фильмов, в которых снимались актеры, введенные пользователем (т.е. все введенные актеры должны быть вместе в этом фильме).
13. Задать хешмап вида (ресторан)-(список блюд). Найти рестораны, у которых в списке блюд есть те, которые ввел пользователь.

III. Добавить к созданным в лаб.раб.№2 классам демонстрацию работы с использованием Stream API (подсчет объектов по какому-либо параметру, сортировка, фильтр по определенному значению и т.д.).

Лабораторная работа №4. Создание программ с графическим интерфейсом пользователя на языке Java. Классы пакета Swing

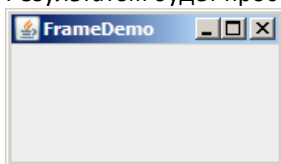
Для создания графического интерфейса пользователя в языке Java есть несколько графических пакетов библиотек (AWT, Swing, SWT и др.). Пакет Swing является одним из наиболее простых в применении и содержит классы для реализации большинства современных элементов GUI (Graphical User Interface – графический интерфейс пользователя).

Базовым объектом при создании пользовательского интерфейса является окно. В классификации языка Java им является класс JFrame. Для создания окна достаточно следующего кода:

```
//файл MyFrame.java
import javax.swing.*;

public class MyFrame {
    public static void main(String s[]) {
        JFrame frame = new JFrame("FrameDemo");// создаем окно с заголовком FrameDemo
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);//делаем базовым действием при
        //закрытие окна выход из приложения, иначе окно закроется, а программа в памяти останется
        frame.setSize(175,100);//задаем размер окна
        frame.setVisible(true);//делаем его видимым
    }
}
```

Результатом будет простое окно:



Дальнейшая работа с окном предусматривает использование менеджеров компоновки или компоновщиков. Это специальные классы, которые позволяют упаковывать содержимое окна и задавать нужное поведение всех элементов окна в зависимости от изменения размеров окна. В языке Java существует несколько компоновщиков:

- **BorderLayout** – размещает элементы в один из пяти регионов (см. рис. 5), указанный при добавлении элемента в контейнер: наверх, вниз, влево, вправо, в центр (или север, юг, запад, восток, центр); при этом, если в какой-либо регион не был добавлен элемент, то этот регион не отображается в окне;
- **FlowLayout** – размещает элементы по порядку в том же направлении, что и ориентация контейнера (слева направо по умолчанию), применяя один из пяти видов выравнивания, указанного при создании менеджера (по центру по умолчанию). Данный менеджер используется по умолчанию в большинстве контейнерах;
- **GridLayout** – размещает элементы таблично. Количество столбцов и строк указывается при создании менеджера. По умолчанию одна строка, а число столбцов равно числу элементов;
- **BoxLayout** – размещает элементы по вертикали или по горизонтали. Обычно он используется не напрямую, а через контейнерный класс **Box**, который имеет дополнительные возможности;
- **CardLayout** – размещает элементы в виде колоды карт, т.е. в текущий момент может быть активным и видимым только один элемент (который лежит наверху колоды).

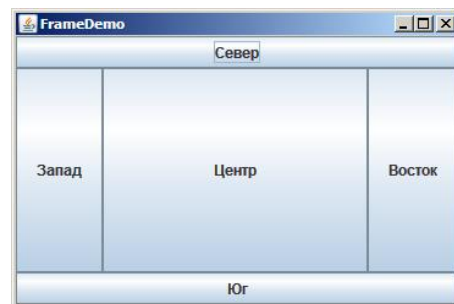


Рис. 5. BorderLayout.

Рекомендуется в качестве основного компоновщика для созданного окна использовать BorderLayout, т.к. это позволяет максимально удобно использовать все доступное пространство формы: в верхней части можно создать меню и панели инструментов, в нижней – строку состояния, в центральной – основные рабочие элементы, а боковые области использовать для дополнительных элементов приложения. Все элементы графического интерфейса пользователя можно располагать на специальных панелях с нужным компоновщиком, а эти панели уже помещать в нужные области окна (верх, низ и т.д.).

Таким образом, необходимо рассмотреть работу с панелями и базовыми элементами пользовательского интерфейса.

JPanel – класс для работы с панелями. Панель – это своеобразный контейнер, к которому можно применить нужный компоновщик и поместить туда необходимые элементы, причем саму панель видно не будет, если только не установить специальные параметры.

JButton – кнопка.

JCheckBox – кнопка-флажок;

JComboBox – выпадающий список;

JLabel – метка, надпись;

JList – список;

JPasswordField – текстовое поле для скрытого ввода;

JProgressBar – компонент для отображения числа в некотором диапазоне;

JRadioButton – переключатели, радио-кнопки, обычно используются с компонентом **ButtonGroup**;

JSlider – компонент, позволяющий выбрать значение из заданного диапазона;

JSpinner – компонент, позволяющий выбрать значение из указанной последовательности;

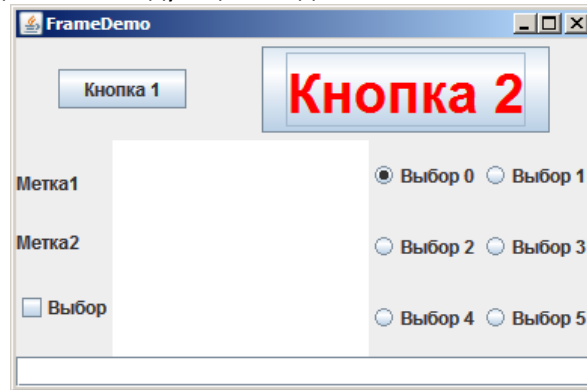
JTable – таблица;

TextField – однострочное текстовое поле;

TextArea – многострочное текстовое поле;

JTree – дерево элементов (иерархически упорядоченных).

Например, программа, создающая окно следующего вида:



```
import java.awt.*;
import java.util.ArrayList;
import javax.swing.*;

public class MyFrame {
    public static void main(String[] args) {
        JFrame frame = new JFrame("FrameDemo"); // создаем окно с заголовком FrameDemo
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // делаем базовым действием при
        // закрытие окна выход из приложения, иначе окно закроется, а программа в памяти останется
        frame.setSize(500,400); // задаем размер окна

        JPanel myPanel1=new JPanel(); // создаем панель
        myPanel1.setLayout(new FlowLayout()); // устанавливаем для нее компоновщик
        myPanel1.add(new JButton("Кнопка 1")); // добавляем кнопку
        // вторую кнопку делаем по-другому, отдельным объектом, причем оформляем ее через html-код
        JButton myButton2 = new JButton
            ("<html><b><font color='red' size=14>Кнопка 2</font></b></html>");
        // создаем распорку, которая будет стоять между кнопками
        Component horizontalStrut = Box.createHorizontalStrut(40); // с расстоянием в 40 точек
        myPanel1.add(horizontalStrut); // добавляем распорку
        myPanel1.add(myButton2); // добавляем вторую кнопку

        Box myBox1=new Box(BoxLayout.Y_AXIS); // создаем объект Box для компоновки BoxLayout
        // с расположением объектов по вертикали (BoxLayout.Y_AXIS)
        myBox1.add(Box.createVerticalStrut(20)); // добавляем распорку от верхнего края окна
        myBox1.add(new JLabel("Метка1")); // добавляем метку
        myBox1.add(Box.createVerticalGlue()); // добавляем пружину - она будет увеличиваться с
        // увеличением окна и уменьшаться с уменьшением окна, тем самым изменяя расстояние между объектами
        myBox1.add(new JLabel("Метка2")); // добавляем еще одну метку
        myBox1.add(Box.createVerticalGlue()); // добавляем еще одну пружину
        myBox1.add(new JCheckBox("Выбор")); // добавляем чекбокс
        myBox1.add(Box.createVerticalStrut(20)); // добавляем распорку от нижнего края окна

        ButtonGroup myGroup=new ButtonGroup(); // создаем группу, в которой будут радиокнопки
        JPanel myPanel2=new JPanel(); // создаем панель для радиокнопок
        // Создаем массив радиокнопок
        ArrayList<JRadioButton> masRB=new ArrayList<JRadioButton>();
        myPanel2.setLayout(new GridLayout(3,2)); // устанавливаем компоновщик для табличного
        // размещения объектов в 3 строки и 2 столбца
        // в цикле будем добавлять радиокнопки и в массив, и в группу, и на панель
        for (int i=0;i<6;i++){
            masRB.add(new JRadioButton("Выбор "+i)); // добавляем радиокнопку в массив
```

```

        //masRB.get(i) возвращает i-ю радиокнопку
        myGroup.add(masRB.get(i)); //вставляем ее в группу
        myPanel2.add(masRB.get(i)); //и добавляем на панель
    }
    masRB.get(0).setSelected(true); //устанавливаем выбранной 0-ю радиокнопку
    //теперь можно добавить все на форму в нужные области компоновки BorderLayout
    frame.add(myPanel1, BorderLayout.NORTH);
    frame.add(myBox1, BorderLayout.WEST);
    frame.add(new JTextArea(), BorderLayout.CENTER); // создаем текстовую область и добавляем ее
                                                    //в центр окна
    frame.add(myPanel2, BorderLayout.EAST);
    frame.add(new JTextField(), BorderLayout.SOUTH); // создаем текстовое поле и добавляем его
                                                    //в нижнюю область окна

    frame.setVisible(true); // делаем окно видимым
    frame.pack(); //упаковываем окно, чтобы привести его к оптимальному размеру, при котором
    //все элементы видны
    frame.setMinimumSize(frame.getSize()); // и делаем этот размер минимальным
}
}

```

Или можно создать отдельные методы для заполнения каждого из регионов окна:

```

import java.awt.*;
import java.util.ArrayList;
import javax.swing.*;

public class MyFrame {
    public static void main(String[] args) {
        JFrame frame = new JFrame("FrameDemo"); // создаем окно с заголовком FrameDemo
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(500, 400); //задаем размер окна

        setNorth(frame); //вызываем метод для заполнения верхней области
        setWest(frame); //вызываем метод для заполнения левой области
        setEast(frame); //вызываем метод для заполнения правой области
        setCenter(frame); //вызываем метод для заполнения центральной области
        setSouth(frame); //вызываем метод для заполнения нижней области
        frame.setVisible(true); //делаем окно видимым
        frame.pack(); //упаковываем его
        frame.setMinimumSize(frame.getSize());
    }

    public static void setNorth(JFrame fr) { //метод для заполнения верхней области
        JPanel myPanel1 = new JPanel();
        myPanel1.setLayout(new FlowLayout());
        myPanel1.add(new JButton("Кнопка 1"));
        JButton myButton2 = new JButton(
            "<html><b><font color='red' size=14>Кнопка 2</font></b></html>");
        Component horizontalStrut = Box.createHorizontalStrut(40);
        myPanel1.add(horizontalStrut);
        myPanel1.add(myButton2);
        fr.add(myPanel1, BorderLayout.NORTH);
    }

    public static void setWest(JFrame fr) { //метод для заполнения левой области
        Box myBox1 = new Box(BoxLayout.Y_AXIS);
        myBox1.add(Box.createVerticalStrut(20));
        myBox1.add(new JLabel("Метка1"));
        myBox1.add(Box.createVerticalGlue());
        myBox1.add(new JLabel("Метка2"));
        myBox1.add(Box.createVerticalGlue());
        myBox1.add(new JCheckBox("Выбор"));
        myBox1.add(Box.createVerticalStrut(20));
        fr.add(myBox1, BorderLayout.WEST);
    }

    public static void setEast(JFrame fr) { //метод для заполнения правой области
        ButtonGroup myGroup = new ButtonGroup();
        JPanel myPanel2 = new JPanel();
        ArrayList<JRadioButton> masRB = new ArrayList<JRadioButton>();
        myPanel2.setLayout(new GridLayout(3, 2));
        for (int i = 0; i < 6; i++) {
            masRB.add(new JRadioButton("Выбор " + i));
            myGroup.add(masRB.get(i));
            myPanel2.add(masRB.get(i));
        }
        masRB.get(0).setSelected(true);
    }
}

```

```

fr.add(myPanel2, BorderLayout.EAST);
}

public static void setCenter(JFrame fr){ //метод для заполнения центральной области
    fr.add(new JTextArea(), BorderLayout.CENTER);
}

public static void setSouth(JFrame fr){ //метод для заполнения нижней области
    fr.add(new JTextField(), BorderLayout.SOUTH);
}
}

```

В Java есть возможность установки стиля отображения GUI. Для этого перед любыми манипуляциями с графическим интерфейсом (перед созданием самого окна) необходимо вызвать команды для установки нужного стиля:

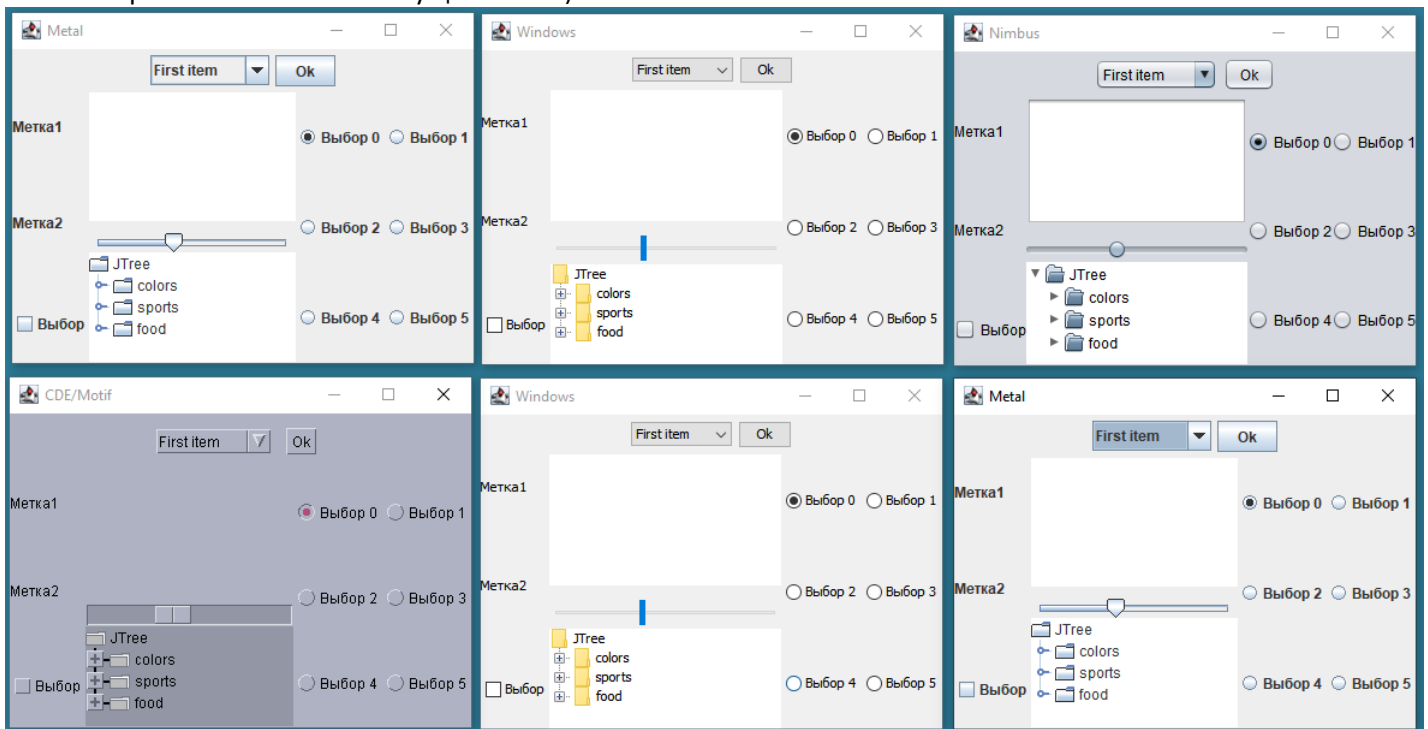
//в блоке try...catch вызываем метод для установки нужного стиля оформления окна

```

try {
    UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel"); //текущий стиль
    // UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel"); //остальные стили
    // UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel"); //закомментированы, их можно
    // по одному проверять, только нужно учитывать, что действующим может быть только один
    // UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");
    // UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    // UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
} catch (ClassNotFoundException | InstantiationException | IllegalAccessException
        | UnsupportedLookAndFeelException e) {
    e.printStackTrace();
}

```

Результат работы программы с использованием каждого стиля можно увидеть на рисунке (в заголовке окна отображается название текущего стиля):



Создать комбобокс можно строками

```

String[] items = { "First item", "Second item", "Third item", "Fourth item" };
JComboBox<String> myCombo = new JComboBox<>(items);

```

Из рисунка видно, что некоторые стили совпадают (1 и 6, 2 и 5), хотя в коде у них разные названия. Это произошло из-за того, что установка стиля MetalLookAndFeel и вызов кода

```

UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());

```

это одно и то же, т.к. стиль MetalLookAndFeel является кроссплатформенным стилем GUI в Java.

Похожая ситуация в случае со стилем WindowsLookAndFeel и вызовом кода

```

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

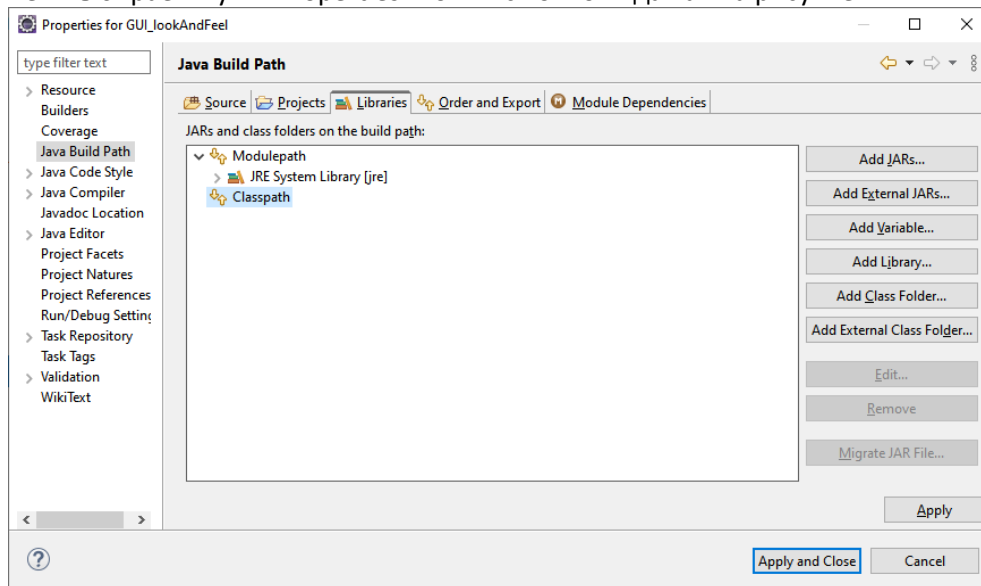
```

т.к. код запускался в ОС Windows, и стиль WindowsLookAndFeel является системным для нее.

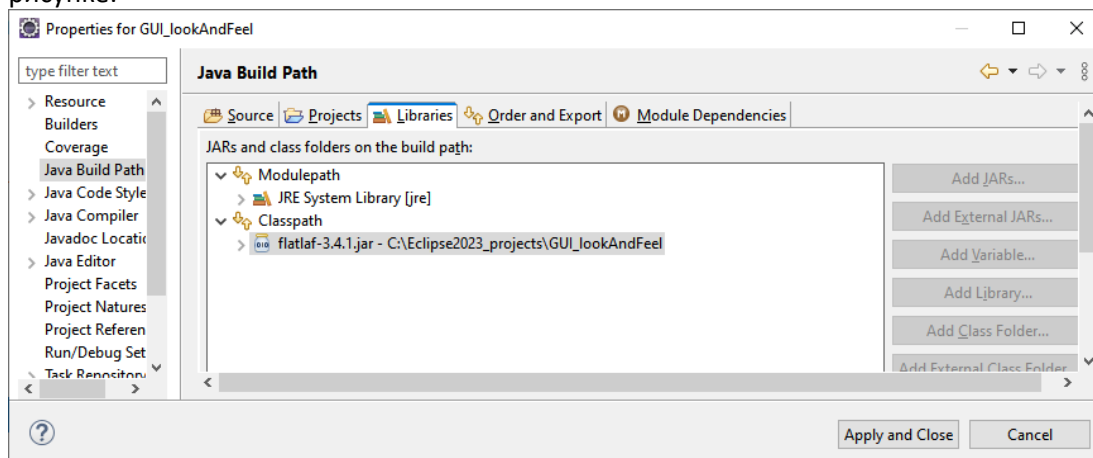
Кроме встроенных стилей, можно для своей программы установить любой доступный, скачав его отдельно. Например, на сайте <https://www.formdev.com/flatlaf/> есть стиль FlatLaf – Flat Look and Feel. Скачать его с этого сайта можно в разделе Download или напрямую по ссылке

<https://search.maven.org/remotecontent?filepath=com/formdev/flatlaf/3.4.1/flatlaf-3.4.1.jar>

Сохранить полученный файл можно в папке своего проекта. Далее нам нужно подключить этот файл к нашему проекту. Для этого в правой части рабочего окна Eclipse вызываем контекстное меню нашего проекта, в нем выбираем пункт Properties. Появится окно вида как на рисунке:



В нем слева выбираем раздел Java Build Path, потом вкладку Libraries, внутри вкладки выбираем Classpath и далее справа нажимаем кнопку Add External JARs. В появившемся диалоговом окне выбираем загруженный файл flatlaf-3.4.1.jar и жмем Открыть. Выбранный файл должен отобразиться в окне как на рисунке:

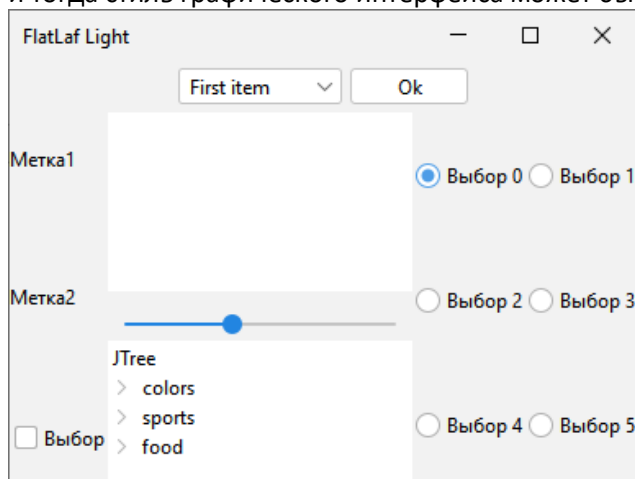


Жмем Apply and Close.

После добавления файла к нашему проекту можно установить этот стиль в своей программе, для этого нужно в блоке try...catch из предыдущего кода использовать строку

```
UIManager.setLookAndFeel("com.formdev.flatlaf.FlatLightLaf");
```

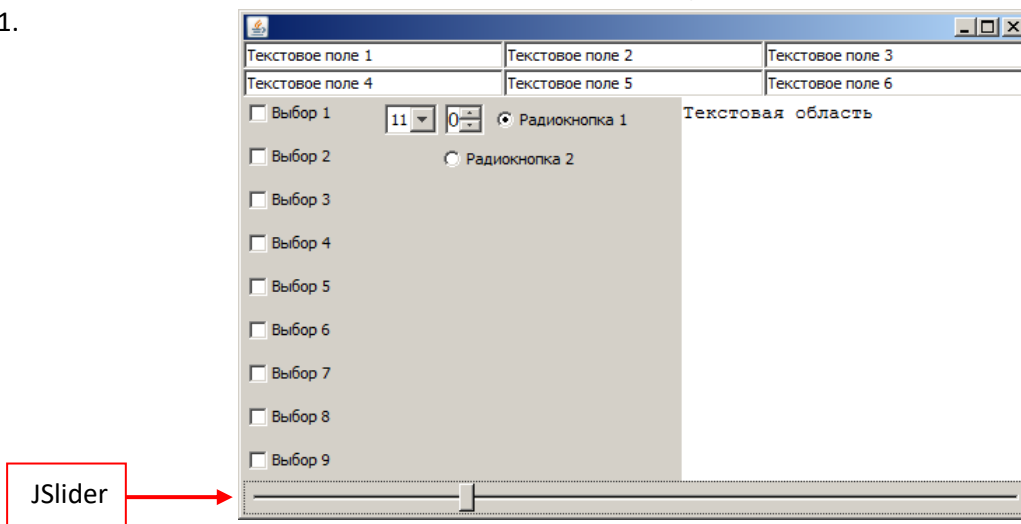
и тогда стиль графического интерфейса может быть примерно следующим:



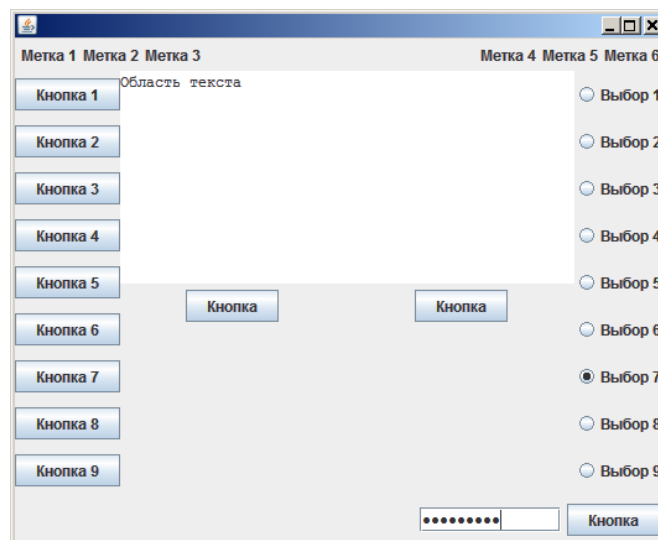
Задачи:

Создать интерфейс по изображению (использовать разные виды компоновок, если одинаковых объектов больше 3, то использовать массив объектов):

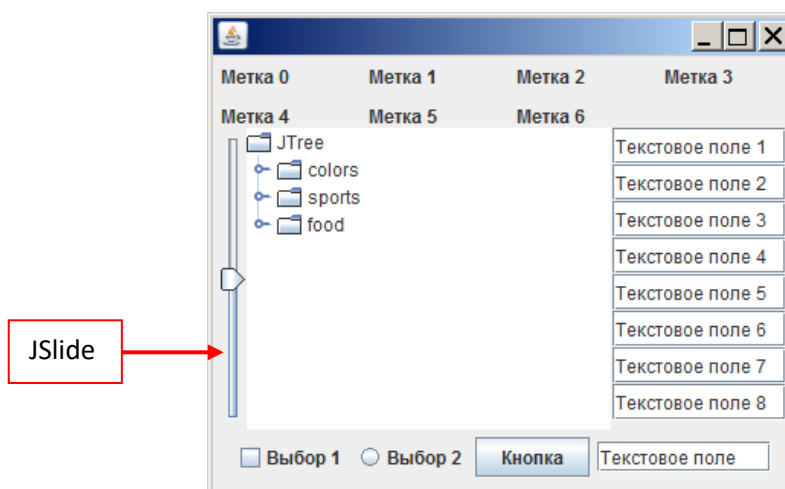
1.



2.



3.



4.

5.

6.

Лабораторная работа №5. Модель обработки событий в Java.

Обработка событий в языке Java организована с использованием специальных классов-слушателей (listeners). Т.е., если мы хотим обработать нажатие на кнопку, то мы регистрируем для этой кнопки объект-слушатель, в одном из методов которого прописываем действия на произошедшее событие. Причем сделать это можно как минимум двумя способами: 1 – с помощью анонимного вложенного класса; 2 – с помощью реализации нужного интерфейса. Разберем оба эти способа.

1. Обработка события с помощью анонимного вложенного класса. Создадим форму с двумя текстовыми полями и кнопкой, и по нажатию кнопки текст из одного поля будет копироваться во второе (рис. 6).

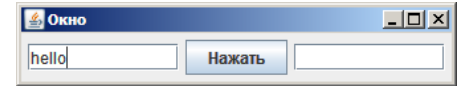


Рис. 6. Обработка события.

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent; //класс для обработки событий
import java.awt.event.ActionListener; //класс-интерфейс слушателя
import javax.swing.*;

public class MyAction {
    public static JFrame myFrame; //объявляем окно приложения
    public static JButton myButton; //объявляем кнопку
    public static JTextField myText1; //объявляем текстовое поле
    public static JTextField myText2; //объявляем другое текстовое поле

    public static void main(String[] args){
        initWindow(); //вызываем функцию инициализации нашего приложения, которую написали ниже
    }

    private static void initWindow(){ //функция для инициализации программы
        myFrame=new JFrame("Окно"); //создаем окно программы
        myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //см. Лаб.раб №4
        myButton=new JButton("Нажать"); //создаем кнопку
        myText1=new JTextField(); //создаем первое текстовое поле
        myText1.setColumns(10); //задаем ему ширину в 10 символов
        myText2=new JTextField(); //создаем второе текстовое поле
        myText2.setColumns(10); //задаем ему ширину в 10 символов
        JPanel myPanel=new JPanel(); //создаем панель
        myPanel.add(myText1); //ставим на панель первое текстовое окно
        myPanel.add(myButton); //ставим на панель кнопку
        myPanel.add(myText2); //ставим на панель второе текстовое окно
        myButton.addActionListener( new ActionListener() { //добавляем к кнопке слушатель событий
            //и тут же его и создаем в виде анонимного вложенного класса
            public void actionPerformed(ActionEvent e) { //описываем процедуру обработки события
                myText2.setText(myText1.getText()); //копируем текст из первого поля во второе
            }
        }); //!!!не забудьте закрыть нужные скобки и поставить точку с запятой
        myFrame.add(myPanel, BorderLayout.NORTH); //добавляем панель в верхний регион окна
        myFrame.pack(); //упаковываем
        myFrame.setMinimumSize(myFrame.getSize()); //задаем минимальный размер
        myFrame.setVisible(true); //показываем
    }
}
```

2. Обработка события с помощью реализации нужного интерфейса. Для этого свой класс создается с реализацией нужного класса-интерфейса (например, ActionListener – обработка нажатия кнопок). При этом, в созданном классе нужно переопределить метод actionPerformed, в котором прописать нужные действия. Создадим форму с тремя кнопками и одним текстовым полем, в которое будет выводиться цифра, написанная на нажатой кнопке (рис. 7).

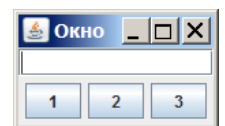


Рис. 7. Обработка события (способ 2).

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

public class MyAction2 extends JFrame implements ActionListener{ //наш класс наследуется от JFrame
    //и реализует интерфейс ActionListener

    public JFrame myFrame;
    public JButton myButton;
    public JTextField myText1;
    public JButton myButton2;
    public JButton myButton3;
```

```

public static void main(String[] args) {
    new MyAction2(); //создаем безымянный объект нашего класса
}

public MyAction2(){ //конструктор нашего класса
    myFrame=new JFrame("Окно");
    myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    myButton=new JButton("1"); //создаем кнопки
    myButton2=new JButton("2");
    myButton3=new JButton("3");
    myText1=new JTextField();
    JPanel myPanel=new JPanel();
    myPanel.add(myButton);
    myPanel.add(myButton2);
    myPanel.add(myButton3);
    myButton.addActionListener(this); //регистрируем слушатель для кнопок - метод нашего класса,
    myButton2.addActionListener(this); // для этого используем указатель this - указатель на
    myButton3.addActionListener(this); // текущий класс - myAction2, т.е. для обработки нажатия
    //на любую из трех кнопок будет использоваться метод actionPerformed, определенный ниже
    myFrame.add(myText1, BorderLayout.NORTH);
    myFrame.add(myPanel, BorderLayout.CENTER);
    myFrame.pack();
    myFrame.setMinimumSize(myFrame.getSize());
    myFrame.setVisible(true);
}

public void actionPerformed(ActionEvent e) { //переопределяем метод actionPerformed из интерфейса
    myText1.setText(e.getActionCommand()); // ActionListener
}
}

```

Кроме классов `ActionEvent` и `ActionListener`, существуют другие классы и соответствующие интерфейсы для обработки событий:

Класс события (который подключается через <code>import java.awt.event.*</code>)	Интерфейс слушателя (через <code>implements</code>)	Методы слушателя (которые нужно переопределить для реализации обработки соответствующего события)	Описание метода, т.е. что с его помощью можно обработать
<code>ActionEvent</code>	<code>ActionListener</code>	<code>actionPerformed()</code>	Нажатие на кнопку, нажатие клавиши Enter при вводе текста (для компонентов типа <code>JButton</code> , <code>JList</code> , <code>JTextField</code>)
<code>AdjustmentEvent</code>	<code>AdjustmentListener</code>	<code>adjustmentValueChanged()</code>	Изменение значения для компонента типа <code>JScrollbar</code>
<code>ComponentEvent</code> (события любого компонента)	<code>ComponentListener</code>	<code>componentHidden()</code>	Компонент стал невидимым
		<code>componentMoved()</code>	Компонент переместился
		<code>componentResized()</code>	Компонент изменил размер
		<code>componentShown()</code>	Компонент стал виден
<code>ContainerEvent</code> (события контейнера, т.е. объекта типа <code>JFrame</code> , <code>JPanel</code> , <code>JScrollPane</code> , <code>JDialog</code> , <code>JFileDialog</code> ...)	<code>ContainerListener</code>	<code>componentAdded()</code>	Добавлен объект в контейнер
		<code>componentRemoved()</code>	Убран объект из контейнера
<code>FocusEvent</code> (события фокуса)	<code>FocusListener</code>	<code>focusGained()</code>	Объект получает фокус
		<code>focusLost()</code>	Объект теряет фокус
<code>ItemEvent</code>	<code>ItemListener</code>	<code>itemStateChanged()</code>	Изменение состояния элемента в списке (для <code>JCheckbox</code> , <code>JList</code> , <code>JComboBox</code> и др., в которых есть объекты типа <code>Item</code>)
<code>KeyEvent</code> (события клавиатуры)	<code>KeyListener</code>	<code>keyPressed()</code>	Была нажата кнопка клавиатуры
		<code>keyReleased()</code>	Была отпущена кнопка клавиатуры
		<code>keyTyped()</code>	Была нажата и отпущена кнопка клавиатуры

MouseEvent (события мыши)	MouseListener	mouseClicked()	Была нажата и отпущена кнопка мыши на указанном объекте
		mouseEntered()	Курсор мыши переместился на объект
		mouseExited()	Курсор мыши покинул область объекта
		mousePressed()	Была нажата кнопка мыши на указанном объекте
		mouseReleased()	Была отпущена кнопка мыши на указанном объекте
	MouseMotionListener	mouseDragged()	Во время перетаскивания (т.е. когда зажата клавиша мыши и курсор перемещается)
		mouseMoved()	Курсор мыши переместился в пределах объекта
TextEvent (события текста)	TextListener	textValueChanged()	(для компонентов типа JTextArea, JTextField)
WindowEvent (события окна приложения)	WindowListener	windowActivated()	Окно становится активным
		windowClosed()	Окно закрылось
		windowClosing()	Окно закрывается (только нажат крестик закрытия, например)
		windowDeactivated()	Окно перестало быть активным
		windowDeiconified()	Окно восстановлено из свернутого состояния
		windowIconified()	Окно минимизировано (свернуто)
		windowOpened()	Окно появилось (впервые стало видимым)

Кроме соответствующих интерфейсов слушателей есть и специальные классы-адаптеры, отличающиеся от слушателей тем, что можно переопределить не все методы, а только нужные. Например, напомним программу, в которой при наведении курсора мыши на объект будет выводиться имя класса этого объекта:

1-й способ, с созданием собственного класса-слушателя для обработки события с реализацией интерфейса

MouseListener.

```
import java.awt.BorderLayout;
import java.awt.event.*;
import javax.swing.*;
```

```
public class MyAction2 extends JFrame{
    public JFrame myFrame;
    public JButton myButton;
    public JTextField myText1;
    public JCheckBox myCheck;
    public JRadioButton myRButton;
    public static JLabel myLabel; //static нужно, чтобы этот объект был доступен для других классов
```

```
    public static void main(String[] args) {
        new MyAction2();
    }
```

```
    public MyAction2(){
        myFrame=new JFrame("Окно");
        myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        myButton=new JButton("1"); //создаем разные объекты
        myCheck=new JCheckBox("2");
        myRButton=new JRadioButton("3");
        myText1=new JTextField();
        myLabel=new JLabel(" ");
        JPanel myPanel=new JPanel();
        myPanel.add(myButton);
        myPanel.add(myCheck);
        myPanel.add(myRButton);
        myButton.addMouseListener(new MyMouseListener()); //регистрируем для всех объектов
        myCheck.addMouseListener(new MyMouseListener()); //в качестве обработчика событий мыши
        myRButton.addMouseListener(new MyMouseListener()); //безымянный объект типа MyMouseListener
        myText1.addMouseListener(new MyMouseListener());
        myFrame.add(myText1, BorderLayout.NORTH);
        myFrame.add(myPanel, BorderLayout.CENTER);
        myFrame.add(myLabel, BorderLayout.SOUTH); //сюда будем выводить информацию
```

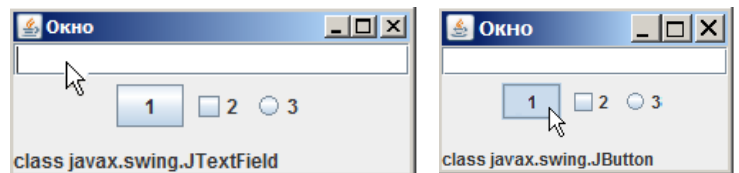


Рис. 8. При наведении мыши выводится название класса объекта.

```

myFrame.setMinimumSize(new Dimension(200,100)); //минимальный размер делаем 200 на 100
myFrame.setVisible(true);
}
}

class MyMouseListener implements MouseListener{//создаем свой класс-слушатель, реализующий
//интерфейс MouseListener и поэтому нужно переопределить все методы
public void mouseClicked(MouseEvent e) { } // интерфейса MouseListener, даже если нам нужен
// всего один из них

public void mouseEntered(MouseEvent e) {
//обращаемся к объекту myLabel класса MyAction2, и устанавливаем на него текст, который
//состоит из названия класса компонента, который вызвал обрабатываемое событие
MyAction2.myLabel.setText(e.getComponent().getClass().toString());
}

//ненужные нам методы просто оставляем пустыми
public void mouseExited(MouseEvent e) { }

public void mousePressed(MouseEvent e) { }

public void mouseReleased(MouseEvent e) { }
}

```

Из примера видно, что неудобством данного способа является необходимость переопределять все методы реализуемого интерфейса, даже если нам реально нужен всего один.

2-й способ, с созданием собственного класса-слушателя для обработки события с наследованием от класса

MouseAdapter.

```

import java.awt.BorderLayout;
import java.awt.event.*;
import javax.swing.*;

public class MyAction2 extends JFrame{
    public JFrame myFrame;
    public JButton myButton;
    public JTextField myText1;
    public JCheckBox myCheck;
    public JRadioButton myRButton;
    public static JLabel myLabel;

    public static void main(String[] args) {
        new MyAction2();
    }

    public MyAction2(){
        myFrame=new JFrame("Окно");
        myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        myButton=new JButton("1");
        myCheck=new JCheckBox("2");
        myRButton=new JRadioButton("3");
        myText1=new JTextField();
        myLabel=new JLabel(" ");
        JPanel myPanel=new JPanel();
        myPanel.add(myButton);
        myPanel.add(myCheck);
        myPanel.add(myRButton);
        myButton.addMouseListener(new MyMouseAdapter());
        myCheck.addMouseListener(new MyMouseAdapter());
        myRButton.addMouseListener(new MyMouseAdapter());
        myText1.addMouseListener(new MyMouseAdapter());
        myFrame.add(myText1, BorderLayout.NORTH);
        myFrame.add(myPanel, BorderLayout.CENTER);
        myFrame.add(myLabel, BorderLayout.SOUTH);
        myFrame.setMinimumSize(new Dimension(200,100));
        myFrame.setVisible(true);
    }
}

class MyMouseAdapter extends MouseAdapter{ //создаем свой класс-слушатель, наследуя его от
//MouseListener, что позволяет нам переопределить только нужный нам метод
    public void mouseEntered(MouseEvent e) {
        myAction2.myLabel.setText(e.getComponent().getClass().toString());
    }
}

```

Классы-адаптеры есть для всех интерфейсов слушателей, в которых методов больше одного, в названии соответствующего интерфейса достаточно поменять `Listener` на `Adapter`.

Задачи:

I. Обязательная задача для всех:

Найти описание следующих классов-слушателей и выписать основное:

ChangeListener, MouseWheelListener.

- II. **Индивидуальное задание.** К созданному интерфейсу из лаб. раб. №4 добавить функциональность – как минимум 3 различных слушателя (например, при нажатии кнопки выводить текст в текстовое поле, при наведении курсора мыши выводить соответствующее сообщение, при изменении ползунка типа JSlider выводить его значение, при нажатии клавиши Enter после набора текста перемещать его куда-нибудь и т.д.). **У студентов с одним вариантом интерфейса не должны быть одинаковые слушатели!**

Лабораторная работа №6. Создание меню, графика в языке Java.

Для создания меню в языке Java есть удобный класс JMenuBar, который представляет собой строку меню, куда можно вставлять пункты меню с нужными подпунктами. Для пунктов меню и подпунктов можно использовать класс JMenuItem, вставляя подпункты в соответствующие пункты меню. Чтобы установить созданное меню на нужное окно, нужно использовать метод setJMenuBar для выбранного объекта типа JFrame.

Например, создадим окно с меню и текстовой строкой, в которую будут выводиться названия выбранных подпунктов меню (рис. 9):

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

```
import javax.swing.*;
```

```
public class MyMenu extends JFrame implements ActionListener{
    private JTextField myText;
```

```
    public static void main(String[] args) {
        new MyMenu("Пример окна с меню");//создаем окно с меню
    }
```

```
    public MyMenu(String name){//конструктор окна с меню, параметр – заголовок окна
        super(name); //передаем заголовок окна в конструктор класса-родителя – в JFrame
        JMenuBar myMenuBar=new JMenuBar();// создаем строку меню

        JMenu menu1=new JMenu("Пункт1");//создаем первый пункт меню
        JMenu first=new JMenu("Пункт1_1");//создаем подпункт меню
        menu1.add(first);//добавляем подпункт в первый пункт меню
        JMenuItem[] first_1=new JMenuItem[3]; // создаем массив из трех подпунктов меню
        for (int i=0;i<3;i++){ //в цикле создается каждый подпункт, добавляется в нужное место меню
            first_1[i]=new JMenuItem("Пункт1_1_"+(i+1));
            first.add(first_1[i]);
            first_1[i].addActionListener(this);//и к нему подключаем слушатель, описанный в конце
        } //класса
        JMenu second=new JMenu("Пункт1_2");//создаем подпункт меню
        menu1.add(second); //добавляем его в меню
        JMenuItem[] second_1=new JMenuItem[3]; // создаем массив из трех подпунктов меню
        for (int i=0;i<3;i++){ // и т.д.
            second_1[i]=new JMenuItem("Пункт1_2_"+(i+1));
            second.add(second_1[i]);
            second_1[i].addActionListener(this);
        }
    }
```

```
    JMenu menu2=new JMenu("Пункт2");//создаем второй пункт меню и далее аналогично первому
    JMenu first2=new JMenu("Пункт2_1");
    menu2.add(first2);
    JMenuItem[] first2_1=new JMenuItem[3];
    for (int i=0;i<3;i++){
        first2_1[i]=new JMenuItem("Пункт2_1_"+(i+1));
        first2.add(first2_1[i]);
        first2_1[i].addActionListener(this);
    }
    JMenu second2=new JMenu("Пункт2_2");
    menu2.add(second2);
    JMenuItem[] second2_1=new JMenuItem[3];
    for (int i=0;i<3;i++){
        second2_1[i]=new JMenuItem("Пункт2_2_"+(i+1));
        second2.add(second2_1[i]);
    }
```

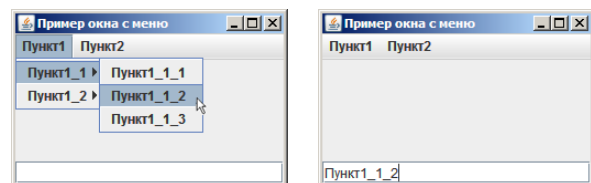


Рис. 9. При выборе пункта меню его название выводится в текстовую строку.

```

        second2_1[i].addActionListener(this);
    }

    myMenuBar.add(menu1); //в строку меню добавляем главные пункты меню
    myMenuBar.add(menu2);
    myText=new JTextField();
    setJMenuBar(myMenuBar); //устанавливаем для окна созданное меню
    add(myText, BorderLayout.SOUTH);
    setSize(300, 200);
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

@Override
public void actionPerformed(ActionEvent e) { //описываем метод слушателя, отвечающий за действия
    myText.setText(e.getActionCommand()); //при выборе пункта меню
    // e.getActionCommand() возвращает название пункта меню, который был выбран
}

```

Для использования графических возможностей языка Java необходимо создать свой компонент, на котором будет происходить рисование, унаследовав его от стандартного компонента пакета Swing, и переопределить в нем метод `paintComponent`. Например, создадим панель, на которой будет происходить рисование:

```

//файл JMyPanel.java
import java.awt.*;
import javax.swing.JPanel;

public class JMyPanel extends JPanel{ // наш класс является наследником класса JPanel
    //создаем перечисление используемых параметров
    public static enum Figure {LINE, OVAL, RECT, ROUNDRRECT, CLEAR};
    private Figure vibor=Figure.CLEAR; //объявляем переменную типа созданного перечисления
    //и присваиваем ей значение CLEAR

    public JMyPanel() { } //конструктор нашего класса

    public void ris(String s) { //метод, вызов которого приводит к перерисовке панели
        //параметр s принимает значение во время вызова данного метода (см. MyGraph.java)
        vibor=Figure.valueOf(s); //устанавливаем, что нужно нарисовать
        repaint(); //перерисовываем нашу панель, т.е. вызываем метод paintComponent
    }

    public void paintComponent(Graphics gr){ //переопределяемый метод с параметром типа Graphics
        super.paintComponent(gr); // вызов такого же метода родительского класса
        // и передача ему параметра типа Graphics
        Graphics2D g = (Graphics2D)gr; //преобразование параметра к типу Graphics2D
        //задание параметров для сглаживания графики (антиалиасинг)
        g.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
        switch (vibor){ //в зависимости от установленного методом ris значения для vibor
            case LINE: g.drawLine(20, 20, 100, 100); break; //рисует линию
            case OVAL: g.drawOval(20, 20, 100, 100); break; // круг
            case RECT: g.drawRect(20, 20, 100, 100); break; // прямоугольник
            case ROUNDRRECT: g.drawRoundRect(20, 20, 100, 100, 60, 60); break; // прямоугольник со
            //скругленными краями, первые четыре параметра - такие же, как у прямоугольника,
            //еще 2 - скругление углов по x и по y
            case CLEAR: g.clearRect(0, 0, getSize().width, getSize().height); break; //очищаем
        }
    }
}

```

В этом примере использован тип данных `enum` – перечисление. Этот тип данных используется для хранения константных значений в виде их перечисления. При этом всем значениям присваивается порядковый номер в зависимости от порядка, в котором эти константы перечислены. В нашем примере значение `LINE` имеет порядковый номер 0, `OVAL` – 1, `RECT` – 2, `ROUNDRRECT` – 3, `CLEAR` – 4. Узнать порядковый номер текущего значения переменной перечисляемого типа можно, вызвав метод `ordinal()`, например, `vibor.ordinal()` вернет нам порядковый номер текущего значения переменной `vibor` (0, 1, 2, 3 или 4). Также можно узнать текстовое значение переменной перечисляемого типа, вызвав метод `toString()`. В нашем случае `vibor.toString()` выдаст одно из значений: `LINE`, `OVAL`, `RECT`, `ROUNDRRECT` или `CLEAR`. Все эти особенности позволяют удобно использовать переменные перечисляемого типа (в нашем случае – для организации выбора рисуемой фигуры).

Также в приведенном примере используется метод `paintComponent(Graphics gr)` для перерисовки компонента. Его параметр `gr` типа `Graphics` в дальнейшем используется при создании объекта `g` типа `Graphics2D`, методами которого и рисуются все фигуры. `Graphics2D` является расширением класса `Graphics` и

позволяет выполнять различные преобразования графики, например, сглаживание, чего нет в классе `Graphics`, который использовался для рисования раньше.

После описанных манипуляций можно использовать созданный класс для рисования. Создадим окно, в котором, в зависимости от нажатой кнопки, рисуется нужная фигура или все стирается (рис. 10):

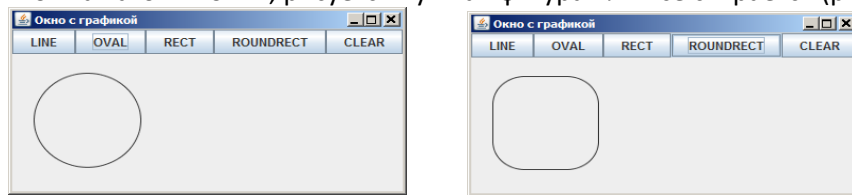


Рис. 10. Рисование фигуры в зависимости от нажатой кнопки.

```
// файл MyGraph.java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class MyGraph extends JFrame implements ActionListener{
    private JMyPanel myPanel=new JMyPanel();//объявляем и создаем нашу панель для рисования

    public static void main(String[] args) {
        new MyGraph("Окно с графикой");//создаем окно
    }

    public MyGraph(String s){ //конструктор с параметром для заголовка окна
        super(s);//вызываем конструктор суперкласса и передаем ему параметр
        Box myBox=new Box(BoxLayout.X_AXIS);//создаем компоновку в виде горизонтального ящика

        JButton[] figs=new JButton[5]; //массив кнопок
        for (int i=0;i<5;i++){
            //каждая кнопка создается с параметром надписи на ней, надпись берется из перечисления,
            //объявленного в классе JMyPanel, values()[i].toString() переводит в текст название i-го
            //параметра из Figure
            figs[i]=new JButton(JMyPanel.Figure.values()[i].toString());
            figs[i].addActionListener(this); //добавляем слушатель, который реализуется в конце
            //описания класса
            myBox.add(figs[i]); //добавляем кнопку в компоновку
            if (i!=4){ //для всех кнопок кроме последней вставляем пружину после кнопки
                myBox.add(Box.createHorizontalGlue());
            }
        }
        myBox.setAlignmentX(CENTER_ALIGNMENT); //устанавливаем для компоновки выравнивание по центру
        //хотя в нашем случае это не важно, т.к. мы используем пружины
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        add(myBox, BorderLayout.NORTH);
        add(myPanel, BorderLayout.CENTER);
        Dimension size=getSize();//записываем в переменную size текущий размер окна
        size.setSize(size.width, size.height+200); //устанавливаем новый размер окна, увеличивая
        //текущий по высоте на 200
        setMinimumSize(size);
        pack();
        setVisible(true);
    }

    public void actionPerformed(ActionEvent e) { //при нажатии на одну из кнопок
        myPanel.ris(e.getActionCommand()); //вызываем метод ris нашей панели (см. JMyPanel.java)
    } //и передаем в качестве параметра название нажатой кнопки (e.getActionCommand())
}
```

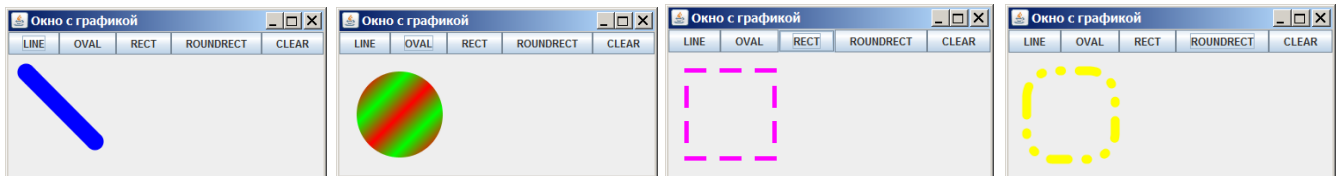
Добавим немного разнообразия в созданный класс `JMyPanel`, изменив его метод `paintComponent`:

```
public void paintComponent(Graphics gr){
    super.paintComponent(gr);
    Graphics2D g = (Graphics2D)gr;
    BasicStroke pen;//создаем перо, параметры которого будут определять стиль линий
    g.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
    switch (vibor){
        case LINE:
            //определяем перо толщиной 20 точек, с закругленными концами линий и закругленными стыками линий
            pen=new BasicStroke(20,BasicStroke.CAP_ROUND,BasicStroke.JOIN_ROUND);
            g.setStroke(pen); //делаем текущим пером созданное нами
            g.setColor(Color.blue); //задаем цвет пера
            g.drawLine(20, 20, 100, 100); break;
        case OVAL:
            //задаем массив, определяющий вид линии
    }
```

```

// элементы массива с четными индексами задают длину штриха в пикселах, элементы с нечетными
// индексами – длину промежутка; массив перебирается циклически;
float[] dash = {10, 30};
// определяем перо толщиной 10 точек, с квадратными концами линий, закругленными стыками линий,
// расстоянием в 10 точек, с которого начинает действовать закругление, массив, определяющий вид
// линии, и с какого элемента массива начинать узор
pen=new BasicStroke(10,BasicStroke.CAP_SQUARE,BasicStroke.JOIN_ROUND,10, dash,0 );
g.setStroke(pen);
g.setColor(Color.red);
// устанавливаем стиль заливки, в качестве параметра задаем градиент от красного к зеленому,
// 30, 30 – начальная точка первого цвета, 50, 50 – начальная точка второго цвета, true –
// цикличность градиента
g.setPaint(new GradientPaint(30, 30, Color.red, 50, 50, Color.green, true));
// g.fill – создание объекта с заливкой, в качестве параметра задается объект из пакета Graphics2D,
// в нашем случае – эллипс
g.fill(new Ellipse2D.Double(20, 20, 100, 100)); break;
case RECT:
float[] dash2 = {20, 20};
pen=new BasicStroke(5,BasicStroke.CAP_SQUARE,BasicStroke.JOIN_BEVEL,1, dash2,0 );
g.setStroke(pen);
g.setColor(Color.magenta);
g.drawRect(20, 20, 100, 100); break;
case ROUNDRECT:
float[] dash3 = {20, 20,2,20,2,20};
pen=new BasicStroke(10,BasicStroke.CAP_ROUND,BasicStroke.JOIN_BEVEL,1, dash3,0 );
g.setStroke(pen);
g.setColor(Color.yellow);
g.drawRoundRect(20, 20, 100, 100,60,60); break;
case CLEAR: g.clearRect(0, 0, getSize().width, getSize().height);break;
}
}

```



Задачи:

I. Обязательная задача для всех:

Переделать пример с рисованием фигур так, чтобы фигуры рисовались выбором соответствующего пункта меню.

II. Индивидуальное задание.

Добавить к предыдущему заданию пункт меню для рисования своей фамилии линиями разной толщины, цвета и стиля (и вообще – проявить фантазию).

Лабораторная работа №7. Работа с файлами, JList, JScrollPane, HashMap. Создание запускового jar-файла.

Компонент JList в Java используется для отображения данных в виде списка. При этом используется парадигма Модель-Вид-Контроллер. Она позволяет не сваливать весь код приложения в кучу, а разделять его на три большие блока. Модель – обработка данных и всё, что с ними связано. Вид – внешность приложения. Определяет то, как будет выглядеть приложение, и что будет отображаться пользователю. Вид показывает данные, которые ему предоставляет модель. Контроллер – обработка всего, что приходит от пользователя.

Для списка работа этой парадигмы выглядит следующим образом: за Модель, т.е. обработку данных, отвечает класс DefaultListModel, за Вид – класс JList, а в качестве Контроллера могут выступать слушатели. Например, напомним программу, в которой создается список с числами от 0 до 9, в этот список можно добавлять и удалять элементы, при этом сам список расположим на панели с прокруткой (JScrollPane) (рис. 11):

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ListWork extends JFrame{
    public static void main(String[] args) {
        ListWork window= new ListWork("Работа со списком");
        window.setVisible(true);
        window.pack();
        window.setMinimumSize(window.getSize());
    }

    public ListWork(String s){
        super(s); //задаем название окна
        final DefaultListModel myListModel = new DefaultListModel(); //создаем модель для нашего
        for (int i=0;i<10;i++){ //списка и заполняем ее элементами, модификатор final делает
            myListModel.addElement(""+i); //объект доступным для вложенных слушателей
        }
        final JList myList=new JList(); //создаем объект, отвечающий за вид нашего списка
        JScrollPane myScroll = new JScrollPane(myList); //создаем панель с прокруткой
        myList.setModel(myListModel); //задаем для списка созданную модель

        Box myBox1=new Box(BoxLayout.Y_AXIS); //создаем компоновку
        final JTextField myText=new JTextField(); //текстовое поле для ввода значений элементов
        myBox1.add(myText); //добавляем поле на компоновку
        Box box1=new Box(BoxLayout.X_AXIS); //создаем еще одну компоновку
        JButton button1=new JButton("Добавить в список"); //создаем кнопку
        box1.add(button1); //добавляем кнопку на компоновку
        button1.addActionListener(new ActionListener() { //создаем слушатель,
            public void actionPerformed(ActionEvent e) { //отвечающий за добавление
                myListModel.addElement(myText.getText()); //элемента списка
            }
        });
        JButton button2=new JButton("Убрать из списка"); //создаем кнопку
        button2.addActionListener(new ActionListener() { //создаем слушатель, отвечающий
            public void actionPerformed(ActionEvent e) { //за удаление элементов списка
                while (myListModel.contains(myText.getText())){ //равных значению
                    myListModel.removeElement(myText.getText()); //в текстовой строке
                }
            }
        });
        box1.add(button2); //добавляем кнопку на компоновку
        JButton buttonClear=new JButton("Очистить список"); //еще кнопка для очистки списка
        buttonClear.addActionListener(new ActionListener() { //к ней слушатель
            @Override
            public void actionPerformed(ActionEvent e) {
                myListModel.clear(); //очищаем список
            }
        });
        box1.add(buttonClear); //добавляем кнопку на компоновку
        myBox1.add(box1); //вставляем компоновку в другую компоновку

        add(myScroll, BorderLayout.CENTER); //добавляем панель с прокруткой в центр окна
        add(myBox1, BorderLayout.NORTH); //добавляем компоновку в верхнюю часть окна
    }
}
```

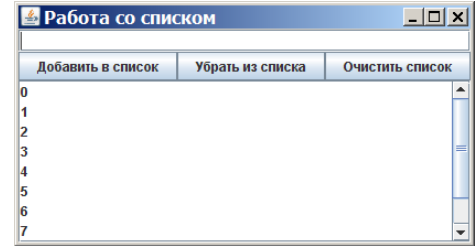


Рис. 11. Работа со списком.

Теперь добавим возможность сохранения в файл и загрузки из файла нашего списка.

Для работы с файлами в языке Java используется несколько классов. Стратегия работы с текстовыми файлами такая:

- создаем объект для записи в файл или чтения из файла (класс `FileWriter` или `FileReader`), соединяя его с нужным файлом;
- создаем объект для потока буферизации записи или чтения (класс `BufferedWriter` или `BufferedReader`), связывая его с объектом для записи в файл или чтения из файла из предыдущего пункта;
- записываем или читаем с использованием предыдущего объекта;
- все вышеописанные пункты помещаем в конструкцию:

```
try {
    //наши действия с файлами
}
catch (IOException e1) {
    e1.printStackTrace();
}
```

Данная конструкция позволяет безопасно использовать код внутри `try`, а все возникающие ошибки (исключения) обрабатывать в секции `catch`, а не выдавать на обработку операционной системе.

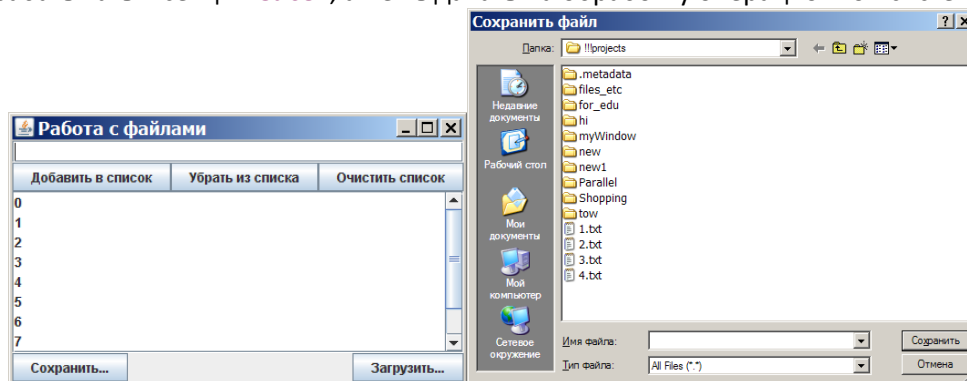


Рис. 12. Загрузка и сохранение списка в файл.

Добавим следующий код к предыдущему примеру (результат на рис. 12):

```
Box myBox2=new Box(BoxLayout.X_AXIS); //новая компоновка
JButton button3=new JButton("Сохранить..."); //кнопка для сохранения списка
myBox2.add(button3);
final FileDialog fdlg=new FileDialog(this, ""); //создаем диалоговое окно для чтения и записи
//файла
button3.addActionListener(new ActionListener() { //слушатель для сохранения
    public void actionPerformed(ActionEvent e) {
        fdlg.setMode(FileDialog.SAVE); //делаем созданный диалог диалогом сохранения
        fdlg.setTitle("Сохранить файл"); //задаем ему заголовок
        fdlg.setVisible(true); //делаем видимым
        FileWriter myWriter = null; //создаем объект типа FileWriter и приравниваем его к null
        try { //секция, в которой можно выполнять небезопасные действия
            // созданному объекту типа FileWriter задаем новый объект с параметрами каталога и файла,
            //выбранного пользователем в диалоге сохранения файла
            myWriter=new FileWriter(fdlg.getDirectory()+fdlg.getFile());
            //создаем объект типа BufferedWriter, соединяя его с созданным объектом myWriter
            BufferedWriter myBWriter=new BufferedWriter(myWriter);
            for(int i=0;i<myListModel.getSize();i++){ //в цикле сохраняем каждый элемент в файл
                myBWriter.write(""+myListModel.getElementAt(i)); //используя BufferedWriter
                myBWriter.newLine(); //и вставляем символ перехода на новую строку
            }
            myBWriter.close(); //закрываем все соединения
            myWriter.close();
        } catch (IOException e1) {
            e1.printStackTrace(); //если произойдет ошибка, будет выведено сообщение
        }
    }
});
myBox2.add(Box.createHorizontalGlue()); //вставляем «пружину», чтобы кнопки были по краям окна
JButton button4=new JButton("Загрузить..."); //кнопка для загрузки списка из файла
button4.addActionListener(new ActionListener() { //слушатель для загрузки из файла
    public void actionPerformed(ActionEvent e) {
        fdlg.setMode(FileDialog.LOAD); //делаем созданный диалог диалогом загрузки
        fdlg.setTitle("Загрузить файл"); //задаем ему заголовок
        fdlg.setVisible(true); //делаем видимым
        FileReader myReader = null;
        try { //секция, в которой можно выполнять небезопасные действия
            // созданному объекту типа FileReader задаем новый объект с параметрами каталога и файла,
            //выбранного пользователем в диалоге загрузки файла
            myReader=new FileReader(fdlg.getDirectory()+fdlg.getFile());
            myListModel.clear(); //очищаем список, т.к. в него будут помещены новые данные
```

```
//создаем объект типа BufferedReader, соединяя его с созданным объектом myReader
BufferedReader myBReader=new BufferedReader(myReader);
String s; //строка для временного хранения данных
//в s записываем строку из файла, и если она не пустая (а пустой она будет, если файл закончился
//или пустой), то добавляем в список новый элемент с параметром s
while ((s=myBReader.readLine())!=null){
    myListModel.addElement(s);
}
myBReader.close();//закрываем все соединения
myReader.close();
} catch (IOException e1) {
    e1.printStackTrace();
}
}
});
myBox2.add(button4);//добавляем кнопку на компоновку

add(myBox2, BorderLayout.SOUTH);//вставляем компоновку в нижнюю область окна
```

Можно добавить к проекту загрузку рисунков из файлов, связанных с названиями элементов списка. Самый простой способ – когда имя файла совпадает с названием элемента списка. Для этого нужно добавить панель, на которую можно будет выводить изображения. Примерный код, который нужно добавить к нашему проекту, может быть следующим:

```
JPanel panell=null;
private static Image image;
static File f1;
```

} Эта часть вставляется перед **public static void** main(String[] args) {
в ней описываются панель для изображений, объект типа Image для хранения изображения и объект типа File для доступа к файлу

Для списка добавляем слушателя:

```
myList.addListSelectionListener(new ListSelectionListener() { //слушатель выделения элемента списка
    public void valueChanged(ListSelectionEvent e) {
        JList tempList=(JList)e.getSource();//создаем временный объект, равный объекту, который
        //вызвал слушателя во время выполнения программы
        f1=new File(tempList.getSelectedValue().toString()+".jpg"); //файл f1 связываем с
        //именем выбранного элемента списка и с расширением jpg (если не задан
        // каталог – значит файл находится в текущем каталоге проекта), таким
        //образом, создаем связь с физическим файлом в каталоге проекта
        try { //секция для небезопасных операций с файлами
            image=ImageIO.read(f1); //объект типа Image связываем с файлом
            loadImage(image); //вызываем функцию для загрузки изображения на панель
        } catch (IOException e1) { //в случае ошибки
            showDialog(); //вызываем функцию с сообщением об ошибке
            panell.repaint(); //перерисовываем панель
        }
    }
});
```

Создаем панель для вывода изображения и меняем компоновку центральной части окна:

```
panell=new JPanel();

Box centerBox=new Box(BoxLayout.X_AXIS); //создаем компоновку
centerBox.add(myScroll); //вставляем в компоновку панель с прокруткой
centerBox.add(panell); //и панель для изображений
add(centerBox, BorderLayout.CENTER); //добавляем компоновку в центральную область окна
```

В конце класса описываем новые функции:

```
public void loadImage(Image im){ //функция для загрузки изображения на панель
    Graphics2D g = (Graphics2D)panell.getGraphics(); //получаем графический контекст панели
    g.drawImage(im, 0, 0, null); //и выводим на него изображение
}

public void showDialog(){ //функция с сообщением об ошибке
    JDialog myDialog=new JDialog(); //создаем окно для вывода сообщения
    myDialog.setModal(true); //делаем его модальным, т.е. пока не закрыть его, основное
    //окно будет недоступным
    myDialog.add(new JLabel("No file!!!")); //вставляем в окно метку с текстом
    myDialog.pack(); //упаковываем окно по размерам выведенного текста
    //задаем расположение окна – посередине основного окна
    myDialog.setLocation(getLocation().x+getWidth()/2, getLocation().y+getHeight()/2);
    myDialog.setVisible(true); //делаем окно видимым
}
```

Таким образом, в каталоге с проектом должны быть файлы с названиями 1.jpg, 2.jpg и т.д., если файла для какого-либо числа из списка не будет, то появится окно с сообщением (рис. 13).

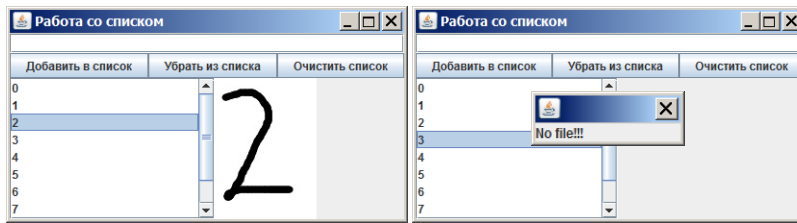


Рис. 13. Список, связанный с графическими файлами.

Но описанный выше способ подойдет, если название файла совпадает с названием элемента списка. А если нам нужно связать с элементом списка произвольную картинку, то необходимо где-то хранить эту связь. Для этих целей можно использовать класс `HashMap`.

Добавим к ранее созданному проекту возможность связывания произвольных изображений с нашим списком, реализовав это при помощи контекстного меню (класс `JPopupMenu`) и `HashMap` (рис. 14):

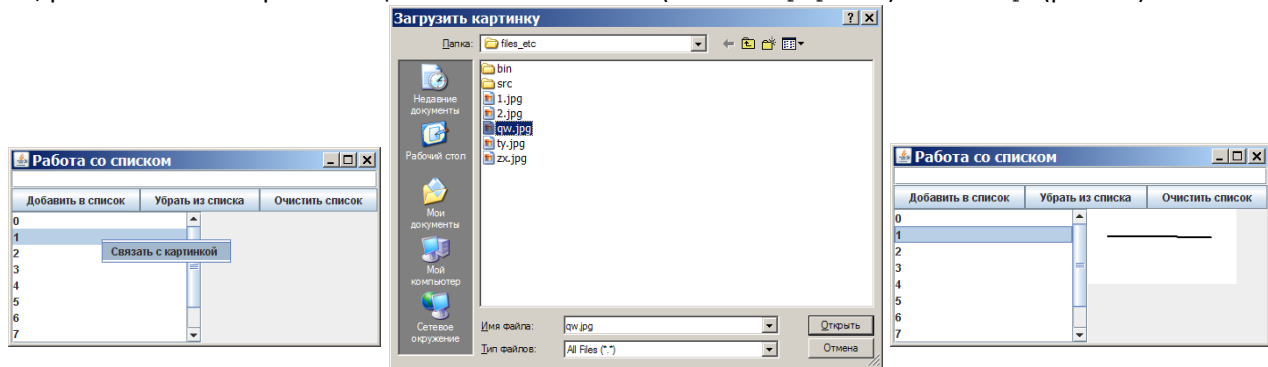


Рис. 14. Связывание произвольных изображений с элементами списка.

Добавим 2 новых члена класса

```
private static HashMap<String, String> myHash=new HashMap<String, String>(); //наш HashMap
static JPopupMenu myPopup; //контекстное меню
```

Добавим создание контекстного меню и его обработку

```
myPopup=new JPopupMenu(); //создаем контекстное меню
JMenuItem myItem1=new JMenuItem("Связать с картинкой"); //создаем пункт контекстного меню
myItem1.addActionListener(new ActionListener() { //добавляем к нему слушателя
    public void actionPerformed(ActionEvent e) {
        loadFromFile(myList.getSelectedValue().toString()); //вызываем функцию, которую создадим ниже,
        //ей передаем название выбранного пункта списка
    }
});
myPopup.add(myItem1); //добавляем созданный пункт к контекстному меню
myList.setComponentPopupMenu(myPopup); //устанавливаем созданное контекстное меню для списка

myList.addMouseListener(new MouseAdapter() { //добавляем к списку слушатель событий мыши
    public void mousePressed(MouseEvent e) { //обрабатываем нажатие на клавишу мыши
        myList.setSelectedIndex(myList.locationToIndex(e.getPoint())); //устанавливаем текущим
        //элементом списка тот, который ближе к позиции курсора мыши в момент нажатия кнопки мыши
    }
});
```

Упростим `myList.addListSelectionListener`:

```
myList.addListSelectionListener(new ListSelectionListener() {
    public void valueChanged(ListSelectionEvent e) {
        //запишем в переменную path значение, связанное в myHash с текущим выделенным элементом списка
        String path=myHash.get(myList.getSelectedValue().toString());
        loadImage(path); //вызов loadImage с путем к графическому файлу
    }
});
```

Перепишем метод `loadImage`:

```
public void loadImage(String path){
    try {
        if (path!=null) { //если переменная не нулевая
            fl=new File(path); //создаем файл
            image=ImageIO.read(fl); //считываем из него изображение
            Graphics2D g = (Graphics2D)panell1.getGraphics(); //получаем графический контекст панели
            g.setColor(panell1.getBackground()); //устанавливаем цвет текущим цветом фона панели
            g.clearRect(0, 0, panell1.getWidth(), panell1.getHeight()); //и закрашиваем панель, чтобы
            //стереть предыдущее изображение
        }
    } catch (IOException e) {
        //обработка исключения
    }
}
```

```

        g.drawImage(image, 0, 0, null); //выводим картинку на панель
    } else throw new IOException(); //иначе кидаем исключение
} catch (IOException e1) { // и обрабатываем его,
    panell.repaint(); //перерисовывая панель
}
}

```

Добавим новую функцию `loadFromFile`, которая принимает в качестве параметра строку с текстом выбранного элемента списка, вызывает диалог открытия файла и соединяет через `myHash` элемент списка с выбранным пользователем графическим файлом:

```

public void loadFromFile(String s){
    FileDialog fdlg=new FileDialog(this, "Загрузить картинку",FileDialog.LOAD);
    fdlg.setFile("*.jpg"); //задаем видимость только файлам с расширением *.jpg
    fdlg.setVisible(true);
    myHash.put(s, fdlg.getDirectory()+fdlg.getFile()); //записываем в myHash текст выбранного
        //элемента списка и путь к выбранному файлу
    loadImage(fdlg.getDirectory()+fdlg.getFile()); //показываем выбранное изображение
}

```

Полностью новый класс будет выглядеть следующим образом:

```

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.HashMap;

import javax.imageio.ImageIO;
import javax.swing.*;
import javax.swing.event.*;

public class ListWithHashPopUp extends JFrame{
    JPanel panell=null;
    private static Image image;
    static File fl;
    private static HashMap<String, String> myHash=new HashMap<String, String>();
    static JPopupMenu myPopup;
    public static void main(String[] args) {
        ListWithHashPopUp window= new ListWithHashPopUp("Работа со списком");
        window.setVisible(true);
        window.pack();
        window.setMinimumSize(window.getSize());
    }

    public ListWithHashPopUp(String s){
        super(s);
        final DefaultListModel myListModel = new DefaultListModel();
        for (int i=0;i<10;i++){
            myListModel.addElement(""+i);
        }
        final JList myList=new JList();
        JScrollPane myScroll = new JScrollPane(myList);
        myList.setModel(myListModel);
        myPopup=new JPopupMenu();
        JMenuItem myItem1=new JMenuItem("Связать с картинкой");
        myItem1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                loadFromFile(myList.getSelectedValue().toString());
            }
        });
        myPopup.add(myItem1);

        myList.setComponentPopupMenu(myPopup);

        myList.addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent e) {
                myList.setSelectedIndex(myList.locationToIndex(e.getPoint()));
            }
        });

        myList.addListSelectionListener(new ListSelectionListener() {
            public void valueChanged(ListSelectionEvent e) {
                String path=myHash.get(myList.getSelectedValue().toString());
                loadImage(path);
            }
        });
    }
}

```

```

Box myBox1=new Box(BoxLayout.Y_AXIS);
    final JTextField myText=new JTextField();
    myBox1.add(myText);
    Box box1=new Box(BoxLayout.X_AXIS);
        JButton button1=new JButton("Добавить в список");
        box1.add(button1);
        button1.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                myListModel.addElement(myText.getText());
            }
        });
        JButton button2=new JButton("Убрать из списка");
        button2.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                while (myListModel.contains(myText.getText())){
                    myListModel.removeElement(myText.getText());
                }
            }
        });
        box1.add(button2);
        JButton buttonClear=new JButton("Очистить список");
        buttonClear.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                myListModel.clear();
            }
        });
        box1.add(buttonClear);
        myBox1.add(box1);

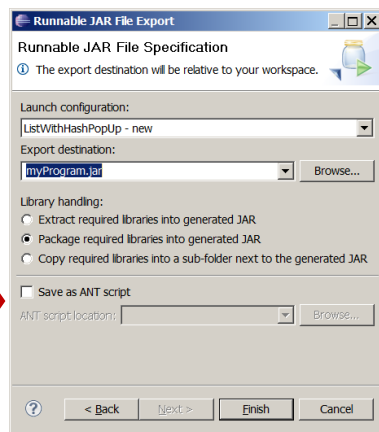
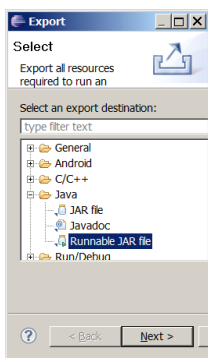
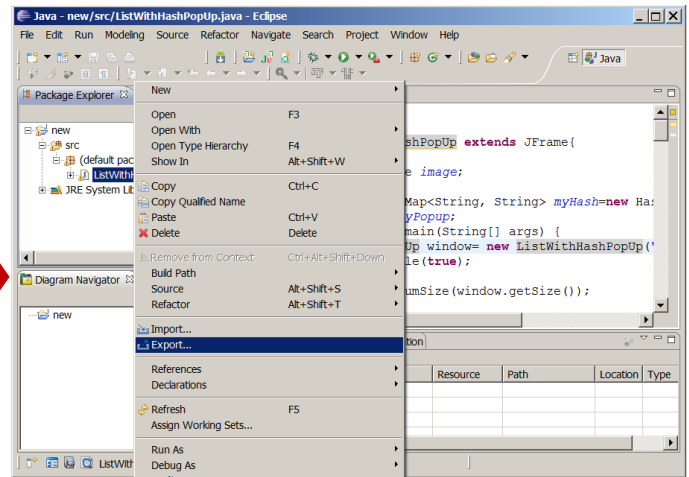
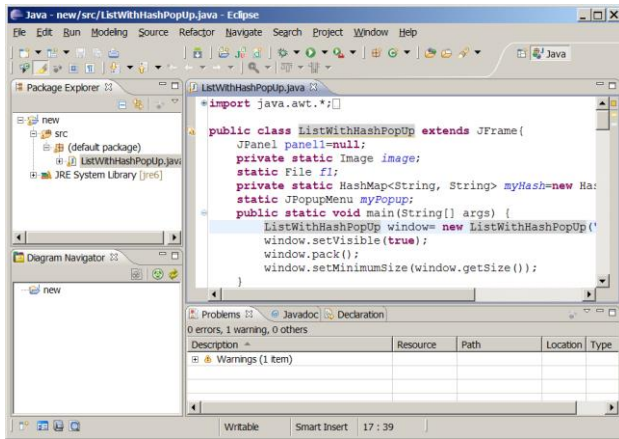
    panell=new JPanel();

    Box centerBox=new Box(BoxLayout.X_AXIS);
    centerBox.add(myScroll);
    centerBox.add(panell);
    add(centerBox, BorderLayout.CENTER);
    add(myBox1, BorderLayout.NORTH);
}
public void loadImage(String path){
    try {
        if (path!=null) {
            f1=new File(path);
            image=ImageIO.read(f1);
            Graphics2D g = (Graphics2D)panell.getGraphics();
            g.setColor(panell.getBackground());
            g.clearRect(0, 0, panell.getWidth(), panell.getHeight());
            g.drawImage(image, 0, 0, null);
        }
        else throw new IOException();
    } catch (IOException e1) {
        panell.repaint();
    }
}

public void loadFromFile(String s){
    FileDialog fdlg=new FileDialog(this, "Загрузить картинку", FileDialog.LOAD);
    fdlg.setFile("*.jpg");
    fdlg.setVisible(true);
    myHash.put(s, fdlg.getDirectory()+fdlg.getFile());
    loadImage(fdlg.getDirectory()+fdlg.getFile());
}
}

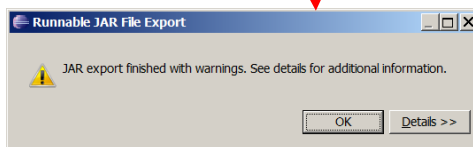
```

До этого мы запускали наши проекты только в среде Eclipse, но для распространения программы нужно, чтобы она запускалась и вне среды создания Java классов. Для этого нужно создать запускной jar-файл (аналог exe-файла). Сделаем это на примере последнего проекта:



В первый раз эти строчки пустые, в первой из списка выбирается запускной класс, в котором есть метод main, во второй записываем название запускного файла (или выбираем путь к нему по кнопке Browse).

После нажатия кнопки Finish может появиться окно, в котором сказано, что экспорт завершился с предупреждениями. Нажимаем OK и запускной файл готов.



Задачи:

I. Обязательная задача для всех:

Переделать пример со списком, чтобы в нем хранился список группы, а выбрав студента, в другом поле появлялись его данные (возраст, адрес). Обязательно использовать HashMap, запись в файл и чтение из файла.

II. Создать запускные jar-файлы из всех своих лабораторных, начиная с 4-ой.

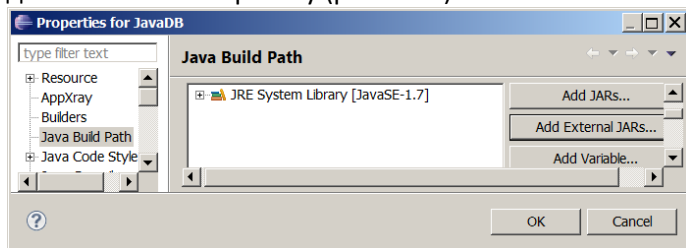
Лабораторная работа №8. Работа с БД

Для демонстрации возможностей языка Java при работе с БД будем использовать H2 – открытую кроссплатформенную СУБД, полностью написанную на языке Java. Она может работать как в клиент-серверном, так и во встроенном режиме. Мы будем использовать встроенный режим, т.е. сама БД будет лежать там же, где и программа, ее использующая.

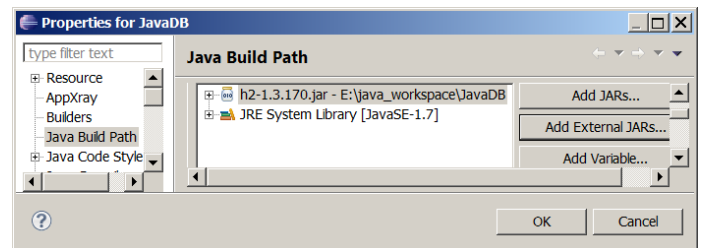
Создадим БД для хранения информации о студентах и преподавателях университетов.

Перед созданием нам понадобится драйвер нужной БД. Т.к. мы будем использовать H2, то нам понадобится файл **h2-1.3.170.jar** (или подобный, другой версии, есть на сервере). Перепишите этот файл себе в папку нового проекта.

Добавим библиотеку БД H2 в настройки созданного проекта. Для этого в контекстном меню проекта выбираем Properties (Свойства), в появившемся окне выбираем Java Build Path (рис. 15 а), в правой части жмем Add External JARs... В появившемся окне выбираем скопированный в свою папку файл **h2-1.3.170.jar** и добавляем его к проекту (рис. 15 б).



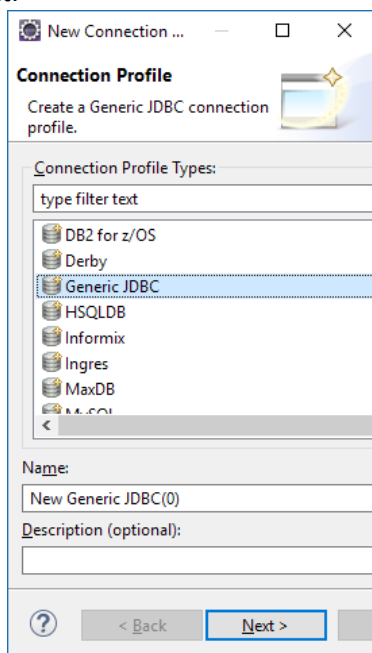
а)



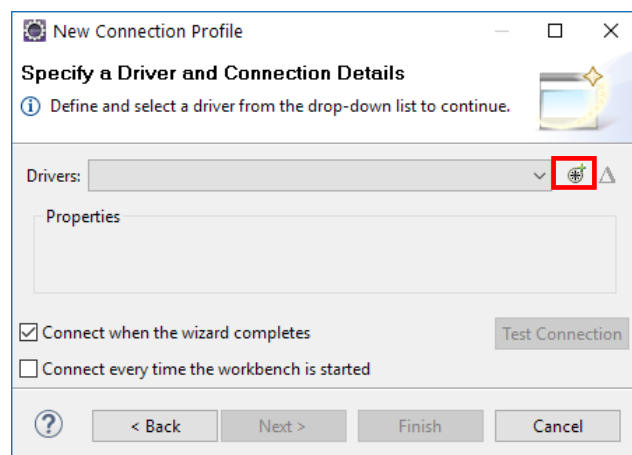
б)

Рис. 15. Добавление библиотеки H2 в проект.

Переходим к созданию БД. В среде программирования Eclipse переключаемся на режим Database Development (Window – Perspective – Open Perspective - Other, в появившемся окне выбираем Database Development). В левой части окна теперь панель Data Source Explorer, в котором отображаются подключения к БД. В контекстном меню Database Connections выбираем New..., в появившемся окне новых подключений выбираем Generic JDBC (рис. 16 а). В поле Name можно задать свое имя для подключения к БД (например, H2). Жмем Next.



а)



б)

Рис. 16. Настройка соединения с БД.

Появится окно выбора драйвера СУБД и параметров подключения (рис. 16 б). Жмем кнопку New Driver Definition (выделена красным прямоугольником). В появившемся окне выбираем Generic JDBC Driver и переходим на вкладку JAR List (рис. 17).

Жмем Add JAR/Zip и в появившемся диалоговом окне находим и выбираем драйвер СУБД H2 (h2-1.3.170.jar или h2-1.3.176.jar, должен быть в папке проекта).

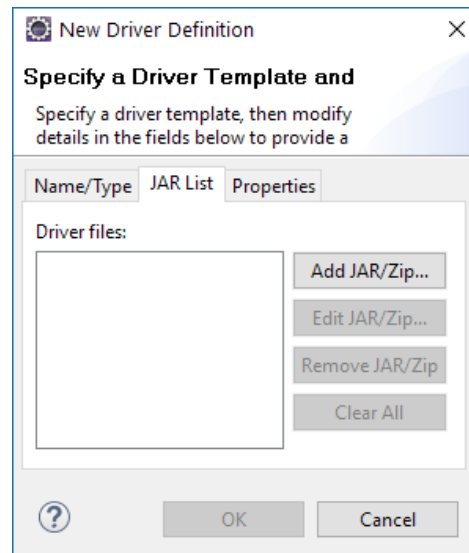
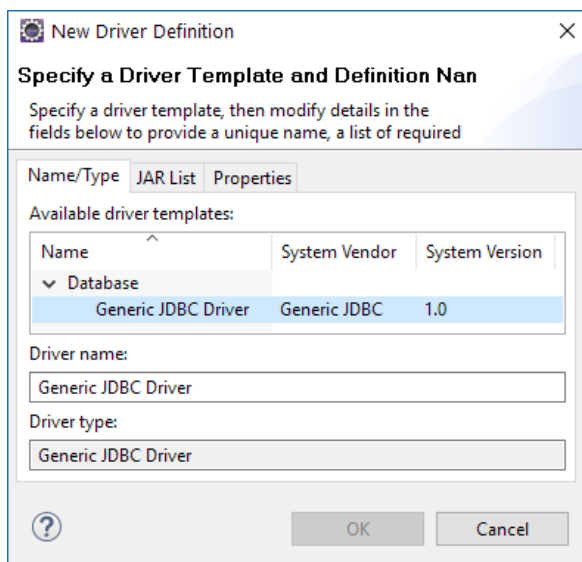
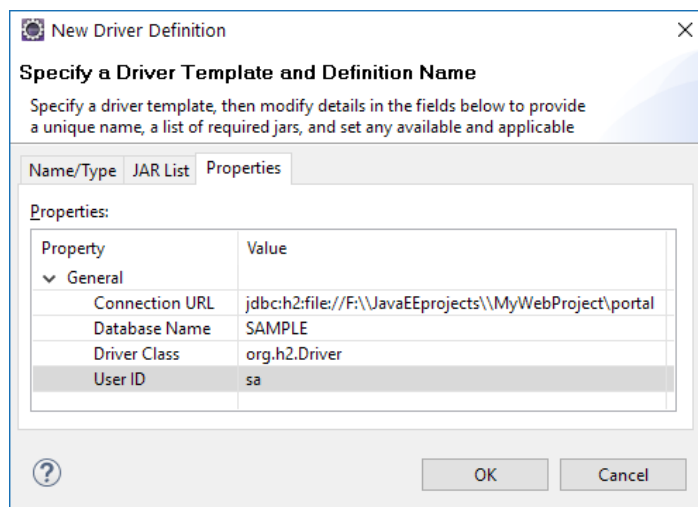
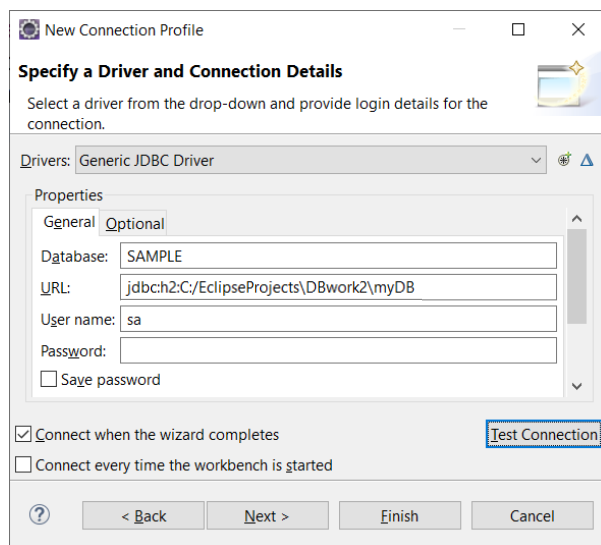


Рис. 17. Выбор драйвера СУБД.

Идем на вкладку Properties (рис. 18), в поле Driver Class жмем кнопку с троеточием, в появившемся окне жмем на радиокнопку Browse for class, ждем, когда в текстовом поле появится org.h2.Driver, выбираем его и жмем Ok. В поле Connection URL прописываем путь к БД, который должен начинаться с `jdbc:h2:file://`, а далее полный путь к БД с указанием всех каталогов и названия самой БД (для версий 1.x см. рис. 18а). **ВНИМАНИЕ!!!** Для версий 2.x путь к файлу БД должен быть вида `jdbc:h2:путь к файлу\имя_бд` (см. рис. 18б). В User ID пишем sa (дефолтное имя пользователя).



а) для версий 1.x (драйвер вида h2-1.x.xxx)



б) для версий 2.x (драйвер вида h2-2.x.xxx)

Рис. 18. Параметры подключения к БД.

Жмем Ok. В появившемся окне жмем Finish. Если все было сделано правильно, то во вкладке Data Source Explorer можно будет увидеть подключение к БД (рис. 19).

Теперь можно создать таблицы БД. Нажимаем кнопку *Open scrapbook to edit SQL statements* (на рис. 20а выделена красным). Появляется текстовая область для ввода SQL-команд (рис. 20б). В верхней части в выпадающем списке Type выбираем Generic JDBC_1.x, в списке Name выбираем имя нашего подключения (в примере – H2_2), в списке Database – свою БД (в примере - MYDB).

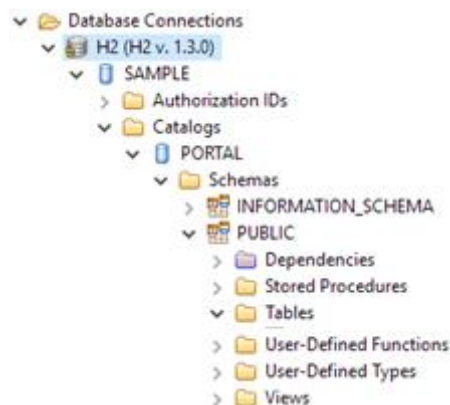
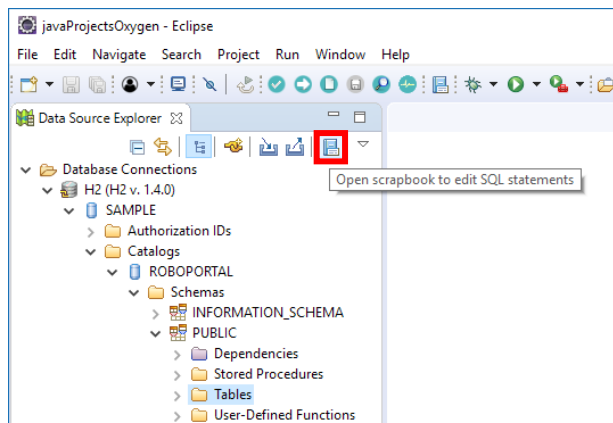
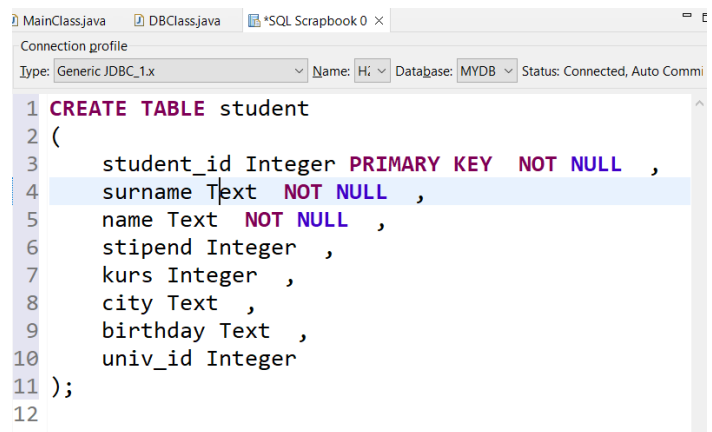


Рис. 19. Подключение к БД.



а) кнопка для вызова области



б) сама область с выбранными параметрами и SQL-кодом

Рис. 20. Область для ввода SQL-команд.

Введем следующую команду для создания таблицы с данными о студентах:

```
CREATE TABLE student
(
    student_id Integer PRIMARY KEY NOT NULL ,
    surname Text NOT NULL ,
    name Text NOT NULL ,
    stipend Integer ,
    kurs Integer ,
    city Text ,
    birthday Text ,
    univ_id Integer );
```

После того, как весь текст введен, выделяем его и в контекстном меню выбираем *Execute all* (Выполнить всё). Ждем окончания выполнения и, если ошибок не было, то таблица будет создана. Переходим к левой панели (Data Source Explorer), ждем F5 или команду Refresh (Обновить) и в раскрытом списке Tables можем увидеть нашу таблицу.

Для заполнения всей оставшейся информации в БД понадобятся следующие команды SQL:

```
INSERT INTO student VALUES (1, 'Иванов', 'Иван', 150, 1, 'Орел', '3/12/1982', 10);
INSERT INTO student VALUES (3, 'Петров', 'Петр', 200, 3, 'Курск', '1/12/1980', 10);
INSERT INTO student VALUES (6, 'Сидоров', 'Вадим', 150, 4, 'Москва', '7/06/1979', 22);
INSERT INTO student VALUES (10, 'Кузнецов', 'Борис', 0, 2, 'Брянск', '8/12/1981', 10);
INSERT INTO student VALUES (12, 'Зайцева', 'Ольга', 250, 2, 'Липецк', '1/05/1981', 10);
INSERT INTO student VALUES (32, 'Котов', 'Павел', 150, 5, 'Белгород', '', 14);
```

```
CREATE TABLE lecturer( lecturer_id Integer Primary Key NOT NULL ,
    surname Text ,
    name Text ,
    city Text ,
    univ_id Integer );
INSERT INTO lecturer VALUES (24, 'Колесников', 'Борис', 'Воронеж', 10);
INSERT INTO lecturer VALUES (48, 'Никонов', 'Иван', 'Воронеж', 10);
INSERT INTO lecturer VALUES (74, 'Лагутин', 'Павел', 'Москва', 22);
INSERT INTO lecturer VALUES (108, 'Струков', 'Николай', 'Москва', 22);
INSERT INTO lecturer VALUES (276, 'Николаев', 'Виктор', 'Воронеж', 10);
```

```
CREATE TABLE subject( subj_id Integer Primary Key NOT NULL ,
    subj_name Text ,
    hours Integer ,
    semester Integer );
INSERT INTO subject VALUES (10, 'Информатика', 56, 1);
INSERT INTO subject VALUES (22, 'Физика', 34, 1);
INSERT INTO subject VALUES (43, 'Математика', 56, 2);
INSERT INTO subject VALUES (46, 'История', 34, 4);
INSERT INTO subject VALUES (73, 'Физкультура', 34, 5);
```

```
CREATE TABLE university(univ_id Integer Primary Key NOT NULL ,
    univ_name Text ,
    rating Integer ,
    city Text );
INSERT INTO university VALUES (10, 'ВГУ', 296, 'Воронеж');
INSERT INTO university VALUES (11, 'НГУ', 345, 'Новосибирск');
INSERT INTO university VALUES (14, 'БГУ', 326, 'Белгород');
INSERT INTO university VALUES (15, 'ТГУ', 368, 'Томск');
INSERT INTO university VALUES (18, 'ВГМА', 327, 'Воронеж');
INSERT INTO university VALUES (22, 'МГУ', 606, 'Москва');
```

```
CREATE TABLE exam_marks ( exam_id Integer Primary Key NOT NULL ,
    student_id Integer ,
    subj_id Integer ,
    mark Integer ,
```

```

        exam_date Text );
INSERT INTO exam_marks VALUES (34, 32, 10, 4, '23/01/2000');
INSERT INTO exam_marks VALUES (43, 6, 22, 4, '18/01/2000');
INSERT INTO exam_marks VALUES (75, 55, 10, 5, '05/01/2000');
INSERT INTO exam_marks VALUES (145, 12, 10, 5, '12/01/2000');
INSERT INTO exam_marks VALUES (238, 12, 73, 3, '17/06/1999');
INSERT INTO exam_marks VALUES (639, 55, 22, 2, '22/06/1999');

CREATE TABLE subj_lect (lecturer_id Integer ,
        subj_id Integer );
INSERT INTO subj_lect VALUES (24, 22);
INSERT INTO subj_lect VALUES (48, 46);
INSERT INTO subj_lect VALUES (74, 73);

```

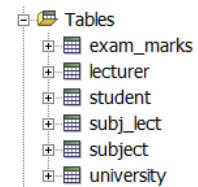


Рис. 21. Все таблицы БД.

Если после выполнения этих команд все прошло успешно, и не было ошибок, то должна получиться структура таблиц БД как на рис. 21. Для просмотра таблицы в её контекстном меню выбираем Data – Sample Contents. В правом нижнем углу появится содержимое таблицы. Если выбрать Data – Edit, то можно будет редактировать содержимое таблицы. В некоторых версиях Eclipse почему-то таблицы не отображаются, поэтому для проверки созданных таблиц можно использовать следующий минимальный код проекта:

Класс для запуска проекта:

```

public class MainClass {
    public static void main(String[] args) {
        String curPath = System.getProperty("user.dir"); //переменная для пути к папке проекта
        System.out.println("Working Directory = " + curPath);
        DBClass myDB = new DBClass(curPath); //объект для работы с БД
        myDB.showTable("student"); //показываем содержимое нужной таблицы - student
    }
}

Класс для работы с БД:
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.logging.Level;
import java.util.logging.Logger;

public class DBClass {
    Connection conn = null; // объект для связи с БД
    Statement stmt = null; // объект для создания SQL-запросов

    public DBClass(String curPath) {
        super();
        try { // нужен блок try ... catch для работы с БД
            Class.forName("org.h2.Driver"); // подгружаем драйвер для H2
            // получаем доступ к БД jdbc:h2:путь к папке проекта\\myDB
            conn = DriverManager.getConnection("jdbc:h2:" + curPath + "\\myDB", "sa", "");
            // получаем объект для выполнения команд SQL
            stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                    ResultSet.CONCUR_UPDATABLE);
        } catch (ClassNotFoundException | SQLException e) { // обрабатываем возможные ошибки
            //подключаем логгер для вывода понятных сообщений в случае ошибок
            Logger.getLogger(MainClass.class.getName()).log(Level.SEVERE, null, e);
            System.out.println("Trouble with connection!!");
        }
        finally { //финальный блок для вывода сообщения о подключении
            if (conn!=null) System.out.println("DB connected!!");
        }
    }

    public void showTable(String tableName) { //метод для вывода в консоль содержимого нужной таблицы
        System.out.println("Table = "+tableName); //выводим имя таблицы
        ResultSet localRS=null; //локальный объект для результатов запроса
        int rowCount=0; //кол-во записей в таблице
        try {
            localRS=stmt.executeQuery("select * from "+tableName); //выполняем запрос для нужной таблицы
            localRS.last(); //переходим на последнюю запись
            rowCount = localRS.getRow(); //получаем номер последнего ряда
            System.out.println("RowsCount="+rowCount); //выводим его, это кол-во записей в таблице
            ResultSetMetaData rsmd = localRS.getMetaData(); //получаем доп.данные о таблице
            int columnsNumber = rsmd.getColumnCount(); //и берем из них кол-во полей
            localRS.beforeFirst(); //переходим в начало списка записей
        }
    }
}

```



```

        mainFrame.add(scrollPane, BorderLayout.CENTER);
        mainFrame.pack(); // подстраиваем размеры окна под размеры таблицы
    });
    mainMenu.add(studentsButton);
    return mainMenu;
}
}

И в класс DBClass добавляем метод getTable:
public JTable getTable(String tableName) {
    ResultSet localRS=null;
    ArrayList<String> colNames = new ArrayList<>(); //массив для названий полей таблицы
    String[][] data = null; //двумерный массив для записей
    try {
        localRS=stmt.executeQuery("select * from "+tableName);
        ResultSetMetaData rsmd = localRS.getMetaData(); //получаем доп.данные о таблице
        int columnsNumber = rsmd.getColumnCount(); //и берем из них кол-во полей
        for (int i = 1; i <= columnsNumber; i++) { //цикл по всем полям таблицы
            colNames.add(rsmd.getColumnName(i)); //добавляем название поля в массив
        }
        localRS.last(); //переходим на последнюю запись результата SQL-запроса
        int rowCount = localRS.getRow(); //считаем кол-во записей в результате SQL-запроса
        data=new String[rowCount][colNames.size()]; //задаем размер двумерного массива записей
        localRS.beforeFirst(); //переходим в начало списка записей
        while (localRS.next()) { //цикл пока есть записи
            for (int i = 1; i <= columnsNumber; i++) { //цикл по всем полям таблицы
                //в нужный элемент двумерного массива вставляем элемент записи
                data[localRS.getRow()-1][i-1]=localRS.getString(i);
                System.out.print(data[localRS.getRow()-1][i-1]+" "); //выводим его для проверки
            }
            System.out.println("");
        }
    } catch (SQLException e) {
        System.out.println("Trouble with query!!");
        e.printStackTrace();
    }
    //создаем таблицу с полученными выше данными
    JTable tempTable = new JTable(data, colNames.toArray());
    return tempTable; //возвращаем таблицу как результат метода
}
}

```

Результат нажатия кнопки должен быть как на рис. 22.

Но показывать id вместо реальных данных – ~~mainvais~~ ~~топ~~ не очень правильно. Поэтому нужно улучшить отображение данных, заодно сделать возможным показ разных таблиц из БД.

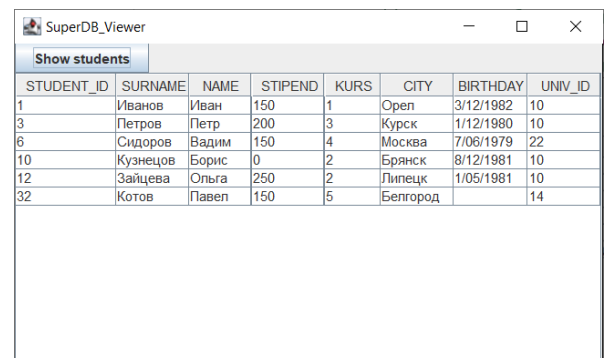
Для этого изменим метод для создания кнопок и добавим улучшенный метод для выбора данных из БД:

В классе **MainGUI** перепишем метод main в следующем виде:

```

public static void main(String[] args) {
    mainFrame = new JFrame("SuperDB_Viewer");
    mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    scrollPane = new JScrollPane();
    String curPath = System.getProperty("user.dir");
    myDB = new DBClass(curPath);
    //добавляем массив названий для кнопок
    String buttonNames[] = { "Show students", "Show lecturers", "Show subjects", "Show universities",
        "Show exam_marks", "Show subj_lect" };
    //и массив названий таблиц в БД
    String tableNames[] = { "student", "lecturer", "subject", "university", "exam_marks", "subj_lect" };
    //создаем хеш-мап для сопоставления названия кнопки и таблицы
    HashMap<String, String> mapForTables = new HashMap<>();
    for (int i = 0; i < buttonNames.length; i++) { //в цикле сопоставляем кнопки и таблицы
        mapForTables.put(buttonNames[i], tableNames[i]);
    }
    mainFrame.add(setMenu(buttonNames, mapForTables), BorderLayout.NORTH);
    mainFrame.setSize(600, 400);
    mainFrame.setMinimumSize(mainFrame.getSize());
    mainFrame.setVisible(true);
    mainFrame.pack();
}

```



The screenshot shows a window titled "SuperDB_Viewer" with a button labeled "Show students". Below the button is a table with the following data:

STUDENT_ID	SURNAME	NAME	STIPEND	KURS	CITY	BIRTHDAY	UNIV_ID
1	Иванов	Иван	150	1	Орел	3/12/1982	10
3	Петров	Петр	200	3	Курск	1/12/1980	10
6	Сидоров	Вадим	150	4	Москва	7/06/1979	22
10	Кузнецов	Борис	0	2	Брянск	8/12/1981	10
12	Зайцева	Ольга	250	2	Липецк	1/05/1981	10
32	Котов	Павел	150	5	Белгород		14

Рис. 22. Результат нажатия кнопки.


```

mainFrame.addWindowListener(new WindowAdapter() { //слушатель закрытия окна, чтобы отключиться от БД
    @Override
    public void windowClosing(WindowEvent e) {
        System.out.println("Exit");
        myDB.closeConnection();
        e.getWindow().dispose();
    }
});
}

```

И переписываем метод `setMenu`:

```

//метод setMenu принимает в качестве параметров массив названий кнопок и хеш-мап
private static Component setMenu(String[] buttonNames, HashMap<String, String> mapForTables) {
    Box mainMenu = new Box(BoxLayout.X_AXIS);
    for (int i = 0; i < buttonNames.length; i++) { //цикл по всем названиям кнопок
        JButton tempButton = new JButton(buttonNames[i]); //создаем временную кнопку
        tempButton.addActionListener(e -> { //слушатель нажатия кнопки
            //создаем таблицу, получая ее из БД вызовом метода getTableWithJoin,
            //которому передаем имя таблицы, связанной с названием нажатой кнопки
            JTable tempTable = myDB.getTableWithJoin(mapForTables.get(e.getActionCommand()));
            mainFrame.remove(scrollPane); // убираем предыдущую панель с таблицей
            // даже если не было еще панели с таблицей
            scrollPane = new JScrollPane(tempTable); // создаем новую панель с прокруткой
            // нужно для корректного отображения больших таблиц
            tempTable.setFillViewportHeight(true);
            // вставляем панель с таблицей в центр окна
            mainFrame.add(scrollPane, BorderLayout.CENTER);
            mainFrame.pack(); // подстраиваем размеры окна под размеры таблицы
        });
        mainMenu.add(tempButton);
    }
    return mainMenu;
}

```

Теперь изменения для класса `DBClass`:

Перед конструктором добавляем новое поле – хеш-мап для запросов

```
HashMap<String, String> selectsForTables;
```

а в сам конструктор, в раздел `finally`, добавляем строки

```

selectsForTables = new HashMap();
selectsForTables.put("student", "select student.SURNAME, student.NAME, student.STIPEND, student.KURS, "
    + "student.CITY, student.BIRTHDAY, university.univ_name from student, university "
    + "where student.univ_id=university.univ_id");
selectsForTables.put("lecturer", "select lecturer.SURNAME, lecturer.NAME, lecturer.CITY,"
    + " university.univ_name from lecturer, university where lecturer.univ_id=university.univ_id");

```

Эти строки вставляют в хеш-мап пары «ключ-значение» для таблицы и соответствующего ей запроса для выбора записей таблиц с учетом внешних ключей и нужных для отображения полей.

Добавляем к этому классу метод `getTableWithJoin`, который возвращает объект класса `JTable` (визуальное отображение таблицы) для соответствующей таблицы БД (параметр-строка):

```

//данный метод является немного измененным вариантом метода getTable
public JTable getTableWithJoin(String tableName) {
    ResultSet localRS = null;
    ArrayList<String> colNames = new ArrayList<>(); // массив для названий полей таблицы
    String[][] data = null; // двумерный массив для записей
    try {
        localRS = stmt.executeQuery(selectsForTables.get(tableName));
        ResultSetMetaData rsmd = localRS.getMetaData(); // получаем доп.данные о таблице
        int columnsNumber = rsmd.getColumnCount(); // и берем из них кол-во полей
        for (int i = 1; i <= columnsNumber; i++) { // цикл по всем полям таблицы
            colNames.add(rsmd.getColumnName(i)); // добавляем название поля в массив
        }
        localRS.last(); // переходим на последнюю запись результата SQL-запроса
        int rowCount = localRS.getRow(); // считаем кол-во записей в результате SQL-запроса
        data = new String[rowCount][colNames.size()]; // задаем размер двумерного массива записей
        localRS.beforeFirst(); // переходим в начало списка записей
        while (localRS.next()) { // цикл пока есть записи
            for (int i = 1; i <= columnsNumber; i++) { // цикл по всем полям таблицы
                // в нужный элемент двумерного массива вставляем элемент записи
                data[localRS.getRow() - 1][i - 1] = localRS.getString(i);
                System.out.print(data[localRS.getRow() - 1][i - 1] + " "); //выводим его для проверки
            }
            System.out.println("");
        }
    }
}

```

```

} catch (SQLException e) {
    System.out.println("Trouble with query!!");
    e.printStackTrace();
}
JTable tempTable = new JTable(data, colNames.toArray());
return tempTable; // возвращаем таблицу как результат метода
}

```

И еще добавляем метод для закрытия соединения с БД:

```

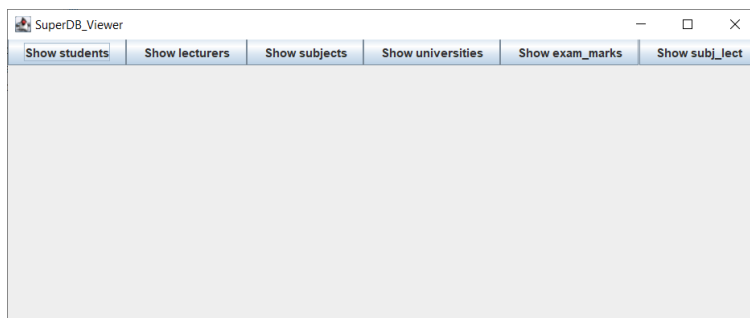
public void closeConnection() {
    if (conn!=null)
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
}

```

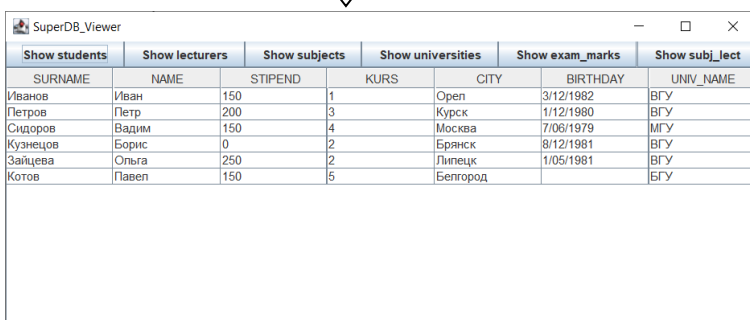
Он нужен, чтобы корректно закрывать связь с БД в конце программы или при срабатывании исключения при обработке ошибок. И теперь программа приобретает более-менее завершенный вид и работает примерно как на рис. 23.

Задание

Согласно выбранному варианту в [лаб.раб.№2](#) сделать БД и реализовать ее отображение в программе с графическим интерфейсом. Часть таблиц сделать встроенными в главное окно, часть – в отдельных окнах. Во всех показываемых колонках должны быть реальные данные, а не id (внешние ключи).



Нажали кнопку Show students



Нажали кнопку Show lecturers

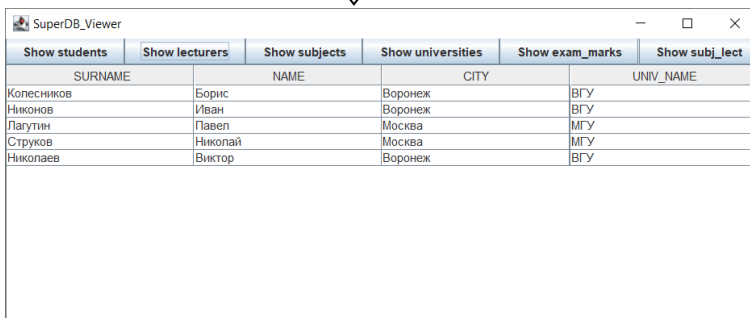


Рис. 23. Результат нажатия на первые 2 кнопки

Лабораторная работа №9. Работа с MS Office

Часто появляется необходимость быстро создать документ MS Office с данными из своего Java-приложения. Для этого есть средства по автоматизации многих стандартных действий в MS Office. Например, библиотека [Apache POI](#). Рассмотрим ее использование на примере сохранения данных с таблицы из предыдущей лабораторной в документ MS Office.

Для начала настроим проект для работы с пакетами библиотеки Apache POI. Для этого в папке проекта создадим каталог POI_libs, куда поместим следующие файлы (можно взять на сервере или по ссылке на [GitHub](#)):

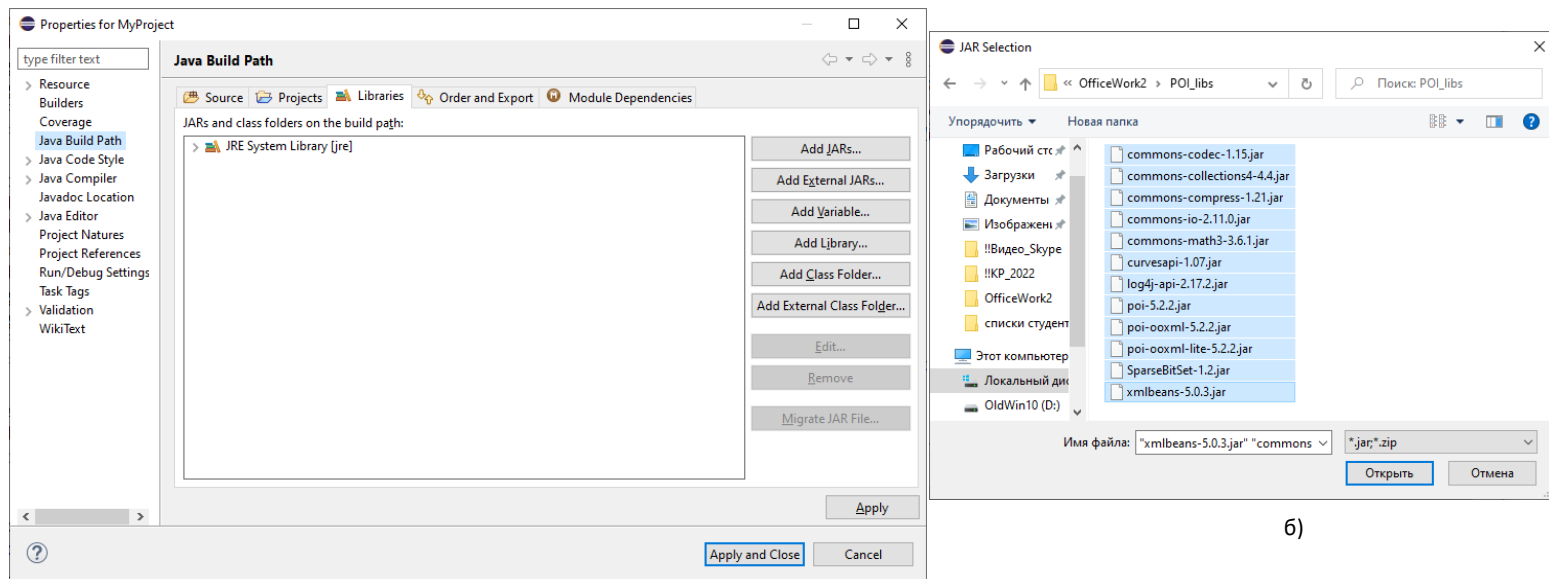
```

commons-codec-1.15.jar
commons-collections4-4.4.jar
commons-compress-1.21.jar
commons-io-2.11.0.jar
commons-math3-3.6.1.jar
curvesapi-1.07.jar
log4j-api-2.17.2.jar
poi-5.2.2.jar

```

poi-ooxml-5.2.2.jar
poi-ooxml-lite-5.2.2.jar
SparseBitSet-1.2.jar
xmlbeans-5.0.3.jar

Далее добавим эти библиотеки в настройках проекта: контекстное меню проекта, пункт Properties. В появившемся окне в списке слева выбираем Java Build Path, в правой части выбираем вкладку Libraries и нажимаем кнопку Add External JARs (см. рис. 24а).



а) Рис. 24. Добавление файлов для Apache POI в проект

В появившемся диалоге выбора файлов отмечаем все файлы папки POI_libs нашего проекта (рис. 24б) и нажимаем Открыть. Выбранные файлы появятся в первом окне, жмем Apply and Close. Теперь можно писать код с использованием классов библиотеки Apache POI.

Нам понадобятся следующие изменения в проекте (весь код доступен по ссылке на [GitHub](#)):

В классе **MainGUI** добавляем поля

```
static FileDialog fdlg;  
static String curTableName = "";
```

Первое – это диалог сохранения файла, второе – имя текущей таблицы, которая отображается в окне.

Метод main этого класса перепишем в следующем виде (серым выделен новый код):

```
public static void main(String[] args) {  
    mainFrame = new JFrame("SuperDB_Viewer");  
    mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    scrollPane = new JScrollPane();  
    String curPath = System.getProperty("user.dir");  
    myDB = new DBClass(curPath);  
    String buttonNames[] = { "Show students", "Show lecturers", "Show subjects",  
                             "Show universities", "Show exam_marks", "Show subj_lect" };  
    String tableNames[] = { "student", "lecturer", "subject", "university", "exam_marks",  
                             "subj_lect" };  
    String tableNamesRus[] = { "Инфо о студентах", "Инфо о преподавателях", "Инфо о предметах",  
                               "Инфо об университетах", "Инфо об экзаменах", "Инфо кто что ведет" };  
    HashMap<String, String> mapForTables = new HashMap<>();  
    // создаем хеш-мап для сопоставления таблицы и ее русского названия  
    HashMap<String, String> mapForTablesRus = new HashMap<>();  
    for (int i = 0; i < buttonNames.length; i++) { // в цикле сопоставляем кнопки и таблицы  
        mapForTables.put(buttonNames[i], tableNames[i]);  
        mapForTablesRus.put(buttonNames[i], tableNamesRus[i]); //и русские названия  
    }  
    //добавляем в метод setMenu параметр mapForTablesRus для передачи ему русских названий таблиц  
    mainFrame.add(setMenu(buttonNames, mapForTables, mapForTablesRus), BorderLayout.NORTH);  
    //добавляем метод для показа кнопок внизу окна, через метод setBottom  
    mainFrame.add(setBottom(), BorderLayout.SOUTH);  
    mainFrame.setSize(600, 400);  
    mainFrame.setMinimumSize(mainFrame.getSize());  
    mainFrame.setVisible(true);  
    mainFrame.pack();  
    mainFrame.addWindowListener(new WindowAdapter() {  
        @Override
```



```

        public void windowClosing(WindowEvent e) {
            System.out.println("Exit");
            myDB.closeConnection();
            e.getWindow().dispose();
        }
    });
    fdlg = new FileDialog(mainFrame, ""); //создаем диалог работы с файлами
    fdlg.setMode(FileDialog.SAVE); //делаем созданный диалог диалогом сохранения
}

```

Далее немного переписываем метод setMenu (серым выделен новый код):

```

private static Component setMenu(String[] buttonNames, HashMap<String, String> mapForTables,
                                HashMap<String, String> mapForTablesRus) {
    Box mainMenu = new Box(BoxLayout.X_AXIS);
    for (int i = 0; i < buttonNames.length; i++) { // цикл по всем названиям кнопок
        JButton tempButton = new JButton(buttonNames[i]); // создаем временную кнопку
        tempButton.addActionListener(e -> { // слушатель нажатия кнопки
            JTable tempTable = myDB.getTableWithJoin(mapForTables.get(e.getActionCommand()));
            curTableName = mapForTablesRus.get(e.getActionCommand());
            mainFrame.remove(scrollPane);
            scrollPane = new JScrollPane(tempTable);
            tempTable.setFillViewportHeight(true);
            // вставляем панель с таблицей в центр окна
            mainFrame.add(scrollPane, BorderLayout.CENTER);
            mainFrame.pack();

            //для активации нижних кнопок (они по умолчанию не активны)
            //получаем компоновку окна
            BorderLayout layout = (BorderLayout) mainFrame.getContentPane().getLayout();
            //получаем компоновку окна и берем нижнюю область с новыми кнопками
            Box southBox = (Box) layout.getLayoutComponent(BorderLayout.SOUTH);
            //в цикле проходим по всем объектам и активируем их
            for (int j = 0; j < southBox.getComponentCount(); j++) {
                southBox.getComponent(j).setEnabled(true);
            }
        });
        mainMenu.add(tempButton);
    }
    return mainMenu;
}

```

И добавляем новые методы для нового функционала (сохранение в формат Word и Excel) :

```

private static Component setBottom() { //метод для создания нижних кнопок
    Box bottom = new Box(BoxLayout.X_AXIS); //компоновка для кнопок
    JButton toWord = new JButton("toWord"); //кнопка для сохранения в Word
    toWord.addActionListener(e -> { //обрабатываем нажатие на кнопку toWord
        String[] columnNames = getTableHeader(); //получаем заголовки столбцов таблицы (см. функцию)
        String[][] data = getTableData(); //получаем записи таблицы (см. функцию)
        File file = getFile("Save to Word file", "docx"); //получаем сохраняемый файл
        //если имя файла не содержит nullnull, т.е. файл реально был выбран для сохранения,
        //то вызываем метод класса ToOffice (будет создан ниже) для создания Word файла с текущей таблицей
        if (!file.getName().contains("nullnull")) ToOffice.toWordDocx(columnNames, data, file,
                                                                    curTableName);
    });
    //делаем кнопку неактивной (она будет активироваться только при выборе таблицы)
    toWord.setEnabled(false);
    bottom.add(toWord); //добавляем кнопку в компоновку
    bottom.add(Box.createHorizontalGlue()); //добавляем пружину
    JButton toExcel = new JButton("toExcel"); //кнопка для сохранения в Excel
    toExcel.addActionListener(e -> {
        String[] columnNames = getTableHeader();
        String[][] data = getTableData();
        File file = getFile("Save to Excel file", "xls");
        if (!file.getName().contains("nullnull")) ToOffice.toExcel(columnNames, data, file,
                                                                    curTableName);
    });
    toExcel.setEnabled(false); //делаем кнопку неактивной
    bottom.add(toExcel); //добавляем кнопку в компоновку
    return bottom; //возвращаем компоновку
}

```

```

private static File getFile(String caption, String ext) { //метод получения файла для сохранения
    System.out.println(caption);
    fdlg.setTitle(caption); //задаем заголовок для диалога
    fdlg.setFile("*. "+ext); //делаем фильтр для отображения только файлов с нужным расширением
}

```

```

    fdlg.setVisible(true); //показываем диалог
    String fileName = fdlg.getDirectory()+fdlg.getFile(); //получаем полный путь выбранного файла
    //если пользователь не добавил к файлу нужное расширение, то сами его добавляем
    if (!fileName.contains(". "+ext)) fileName = fileName.concat(". "+ext);
    File file = new File(fileName); //создаем файл
    System.out.println("file = "+file);
    return file; //возвращаем файл
}

private static String[] getTableHeader() { //функция для получения названия столбцов таблицы
    JViewport viewPort = (JViewport) scrollPane.getComponent(0); //получаем внутреннюю область панели
    JTable tempTable = (JTable) viewPort.getComponent(0); //из нее получаем таблицу (объект с номером 0)
    TableModel tableModel = tempTable.getModel(); //у таблицы берем модель (в ней находятся данные)
    int colCount = tableModel.getColumnCount(); //узнаем кол-во полей (столбцов)
    String[] columnNames = new String[colCount]; //создаем массив такого размера
    for (int i = 0; i < colCount; i++) { //в цикле проходим по полям (столбцам) таблицы
        columnNames[i] = tableModel.getColumnName(i); //и записываем их в массив
    }
    System.out.println(" " + Arrays.asList(columnNames)); //вывод массива для проверки
    return columnNames; //возвращаем массив
}

private static String[][] getTableData() { // функция для получения содержимого таблицы
    JViewport viewPort = (JViewport) scrollPane.getComponent(0); //получаем внутреннюю область панели
    JTable tempTable = (JTable) viewPort.getComponent(0); //из нее получаем таблицу (объект с номером 0)
    TableModel tableModel = tempTable.getModel(); //у таблицы берем модель (в ней находятся данные)
    int colCount = tableModel.getColumnCount(); //узнаем кол-во полей (столбцов)
    int rowCount = tableModel.getRowCount(); //узнаем кол-во записей (содержимое таблицы)
    String[][] data = new String[rowCount][colCount]; //создаем двумерный массив такого размера
    for (int i = 0; i < rowCount; i++) { //во вложенных циклах
        for (int j = 0; j < colCount; j++) { //проходим по записям таблицы
            data[i][j] = (String) tableModel.getValueAt(i, j); //и записываем их в массив
        }
    }
    for (int i = 0; i < data.length; i++) { //вывод массива для проверки
        System.out.println(Arrays.asList(data[i]));
    }
    return data; //возвращаем массив
}

```

И теперь осталось описать класс ToOffice, который собственно и создает файлы формата docx и xls:

```

public class ToOffice {
    static Workbook book; // книга в Excel

    //метод для записи данных в файл Excel в формате xls, первый параметр - строковый массив с заголовками колонок
    //второй параметр - список с данными, третий параметр - файл для сохранения, четвертый - название таблицы
    public static void toExcel(String[] tableTitles, String[][] data, File file, String curTableName) {
        book = new HSSFWorkbook(); // создаем книгу
        Sheet sheet = book.createSheet(curTableName); // создаем лист
        // создадим объединение из нескольких ячеек, чтоб сделать шапку таблицы,
        // кол-во ячеек зависит от кол-ва столбцов таблицы
        sheet.addMergedRegion(new CellRangeAddress( // добавляем в книгу объединенный диапазон ячеек
            0, // начальный ряд диапазона
            0, // конечный ряд диапазона
            0, // начальная колонка диапазона
            tableTitles.length // конечная колонка диапазона
        ));
        Row row = sheet.createRow(0); // создаем новый ряд
        Cell cell = row.createCell(0); // создаем ячейку
        cell.setCellValue(curTableName); // задаем текст объединенных ячеек
        HSSFCellStyle cellStyle = (HSSFCellStyle) book.createCellStyle(); // создаем стиль для ряда
        cellStyle.setAlignment(HorizontalAlignment.CENTER); // задаем выравнивание по центру
        Font font = book.createFont(); // создаем шрифт для объединенных ячеек
        font.setFontHeightInPoints((short) 24); // задаем размер шрифта
        font.setColor(HSSFColor.HSSFColorPredefined.RED.getIndex()); // задаем цвет шрифта
        cellStyle.setFont(font); // добавляем шрифт к стилю
        cell.setCellStyle(cellStyle); // устанавливаем стиль на ячейку

        headCreate(sheet, tableTitles); // вызываем метод для заголовка таблицы (описан ниже)
        dataToSheet(sheet, data); // вызываем метод для заполнения данных таблицы (описан ниже)

        for (int i = 0; i < tableTitles.length; i++) { // цикл для установки ширины ячеек по содержимому
            sheet.autoSizeColumn(i);
        }

        try { //секция для записи в файл
            book.write(new FileOutputStream(file)); // пишем книгу в файл через потоковый объект
            book.close(); // закрываем книгу
        }
    }
}

```

```

    } catch (IOException e) {
        e.printStackTrace();
    }
    System.out.println("xls file was written successfully");
}

private static void headCreate(Sheet sheet, String[] tableTitles) { // метод для создания заголовка таблицы
    // Нумерация рядов начинается с нуля
    Row row = sheet.createRow(1); // создаем новый ряд
    HSSFCellStyle cellStyle = (HSSFCellStyle) book.createCellStyle(); // создаем стиль для ряда
    cellStyle.setAlignment(HorizontalAlignment.CENTER); // задаем выравнивание по центру
    cellStyle.setBorderBottom(BorderStyle.THICK); // задаем нижнюю жирную границу
    cellStyle.setBorderLeft(BorderStyle.THICK); // и все остальные
    cellStyle.setBorderRight(BorderStyle.THICK);
    cellStyle.setBorderTop(BorderStyle.THICK);
    Font font = book.createFont(); // создаем объект для параметров шрифта
    font.setBold(true); // делаем шрифт жирным
    cellStyle.setFont(font); // устанавливаем шрифт в стиль
    for (int i = 0; i < tableTitles.length; i++) { // цикл по заголовкам
        Cell temp = row.createCell(i); // создаем ячейку
        temp.setCellStyle(cellStyle); // задаем ей стиль
        temp.setCellValue(tableTitles[i]); // задаем ей значение
    }
}

private static void dataToSheet(Sheet sheet, String[][] data) { // метод для заполнения данных таблицы
    HSSFCellStyle cellStyle = (HSSFCellStyle) book.createCellStyle(); // создаем стиль для ряда
    cellStyle.setBorderBottom(BorderStyle.THIN); // задаем нижнюю тонкую границу
    cellStyle.setBorderLeft(BorderStyle.THIN); // и все остальные
    cellStyle.setBorderRight(BorderStyle.THIN);
    cellStyle.setBorderTop(BorderStyle.THIN);
    for (int i = 0; i < data.length; i++) { //цикл по массиву с данными
        Row row = sheet.createRow(i+2); // создаем новый ряд
        for (int j = 0; j < data[i].length; j++) {
            Cell temp = row.createCell(j); // создаем ячейку
            temp.setCellValue(data[i][j]); // вставляем туда очередные данные
            temp.setCellStyle(cellStyle); //задаем стиль ячейки
        }
    }
}

//метод для записи данных в docx файл, параметры - массив с заголовками для столбцов, массив с данными, файл для записи
//и имя таблицы
public static void toWordDocx(String[] tableTitles, String[][] data, File file, String curTableName) {
    XWPFDocument document = new XWPFDocument(); // создаем документ Word
    XWPFPparagraph paragraph = document.createParagraph(); // создаем абзац в документе
    XWPFRun run = paragraph.createRun(); // создаем объект для записи в полученный ранее абзац
    run.setText(curTableName); // текст перед таблицей
    XWPFTable table = document.createTable(); // создаем таблицу
    XWPFTableRow tableHead = table.getRow(0); // создаем первый ряд - заголовок таблицы
    for (int i = 0; i < tableTitles.length; i++) { // цикл по заголовкам
        XWPFPparagraph paragraph1 = null; // объявляем объект для параграфа
        if (i == 0) { // если первая ячейка, то сразу берем у нее параграф
            paragraph1 = tableHead.getCell(0).getParagraphs().get(0);
        } else { // иначе создаем новую ячейку и берем у нее параграф
            XWPFTableCell cell = tableHead.addNewTableCell();
            paragraph1 = cell.getParagraphs().get(0);
        }
        paragraph1.setAlignment(ParagraphAlignment.CENTER); // устанавливаем выравнивание по центру
        XWPFRun tableHeadRun = paragraph1.createRun(); //// создаем объект для записи в абзац
        tableHeadRun.setBold(true); // устанавливаем жирный шрифт
        tableHeadRun.setText(tableTitles[i]); // задаем текст
    }
    for (int i = 0; i < data.length; i++) { //цикл по массиву с данными
        XWPFTableRow tableRow = table.createRow(); // создаем ряд
        for (int j = 0; j < data[i].length; j++) {
            tableRow.getCell(j).setText(data[i][j]); // вставляем туда очередную ячейку
        }
    }
}

try { //секция для записи в файл
    FileOutputStream out = new FileOutputStream(file); //создаем файловый поток вывода с новым файлом
    document.write(out); // пишем в файл из созданного объекта
    out.close(); // закрываем поток вывода
    document.close(); //и документ
} catch (IOException e) {
    e.printStackTrace();
}
System.out.println("docx file was written successfully");
}
}

```

Если всё сделано правильно, то при запуске программы должно получиться примерно как на рис. 25:

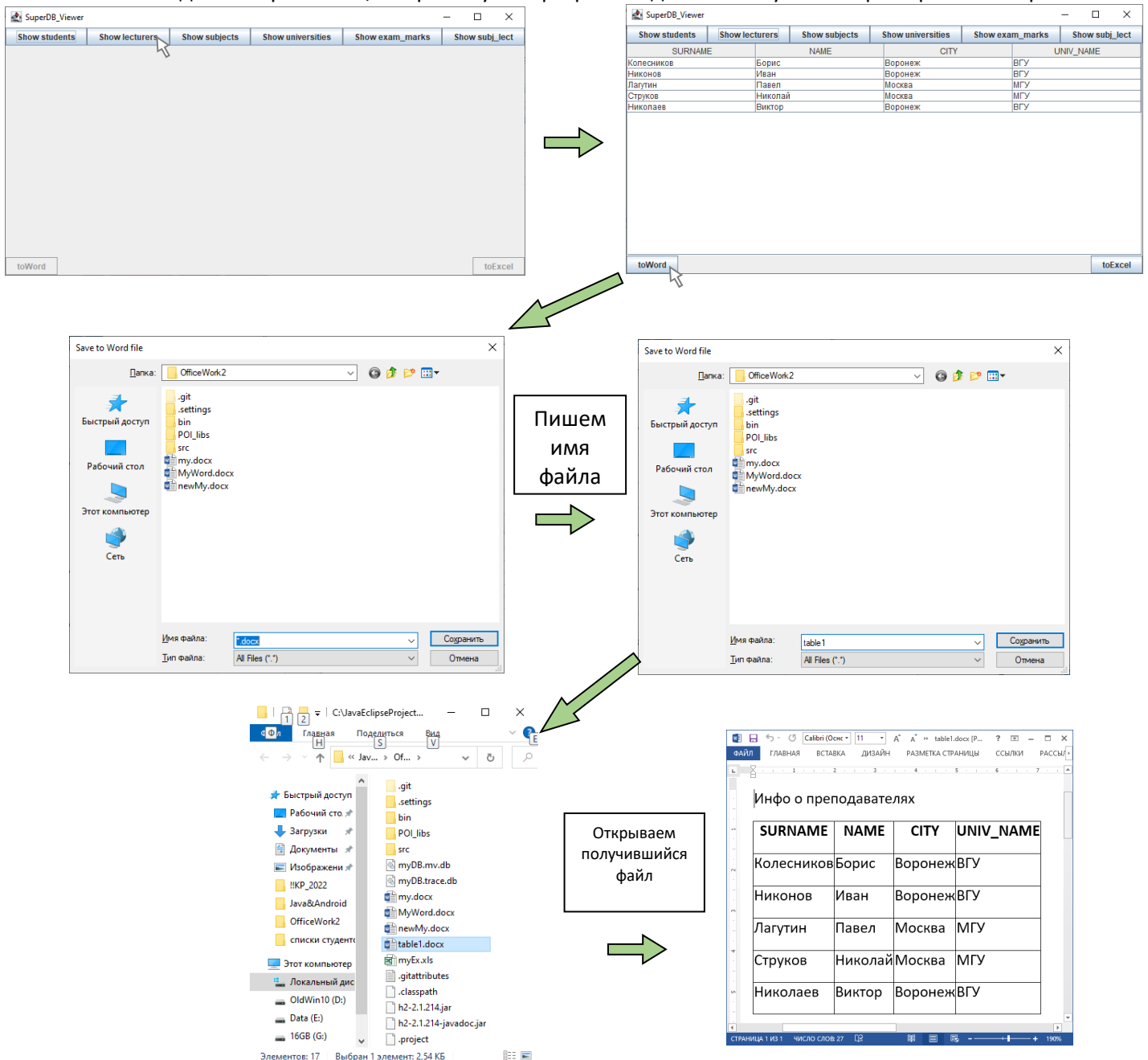


Рис. 25. Результат выполнения программы при нажатии кнопки toWord

Задание

Добавить к программе из предыдущей лабораторной возможность сохранять выбранную таблицу в Word и в Excel.