**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторной работе № 3**

**по дисциплине «Технология объектно-ориентированного программирования и проектирования»**

Тема: Разработка интерфейса пользователя

|  |  |
| --- | --- |
| Студент гр. 4312 | Гущин К. Д. |
| Преподаватель | Лазарев Э. А. |

Санкт-Петербург

2025

**Цель работы:**

Разработать ПК для администратора выставки собак. В ПК должны храниться сведения о собаках, их владельцах и судьях. Администратор выставки может добавлять, изменять и удалять эти сведения.

**Задачи:**

1. Описать назначение экранной формы с указанием перечня вводимой

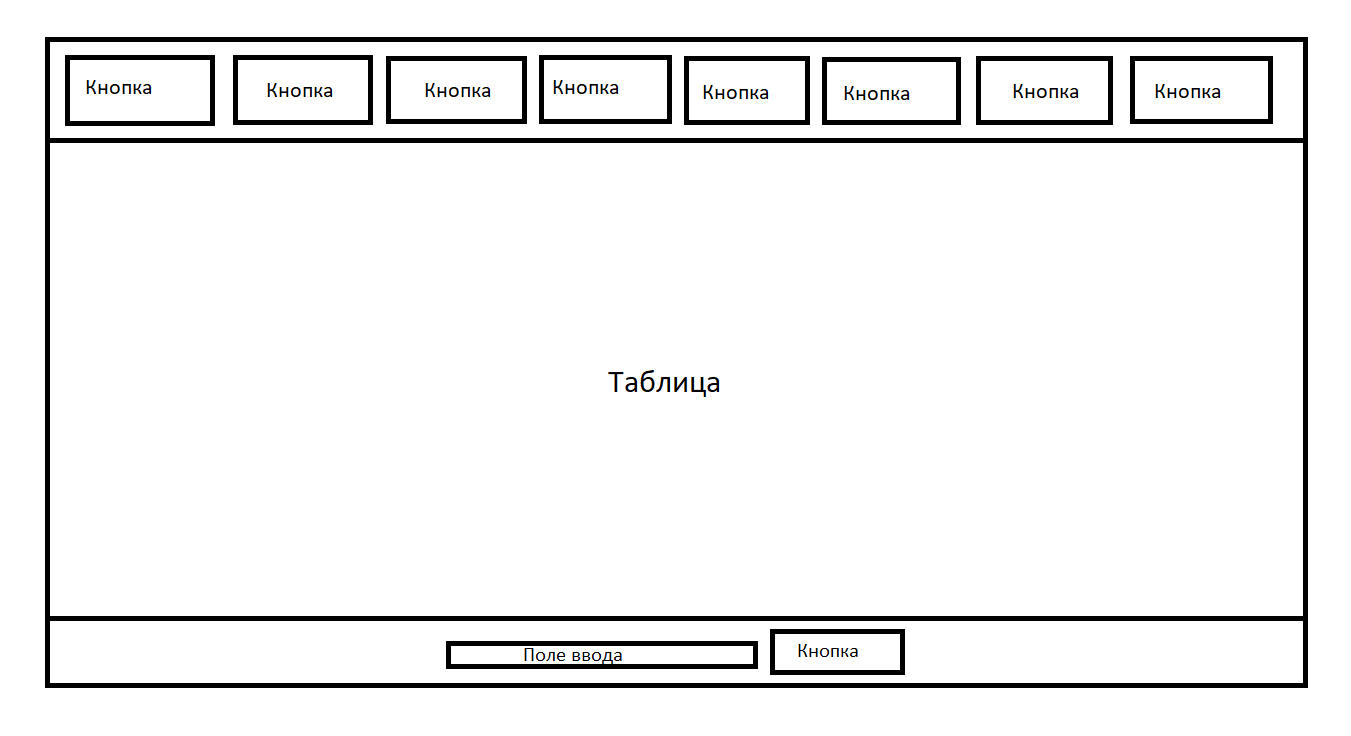
и выводимой информации, а также списка функций, доступных пользователю

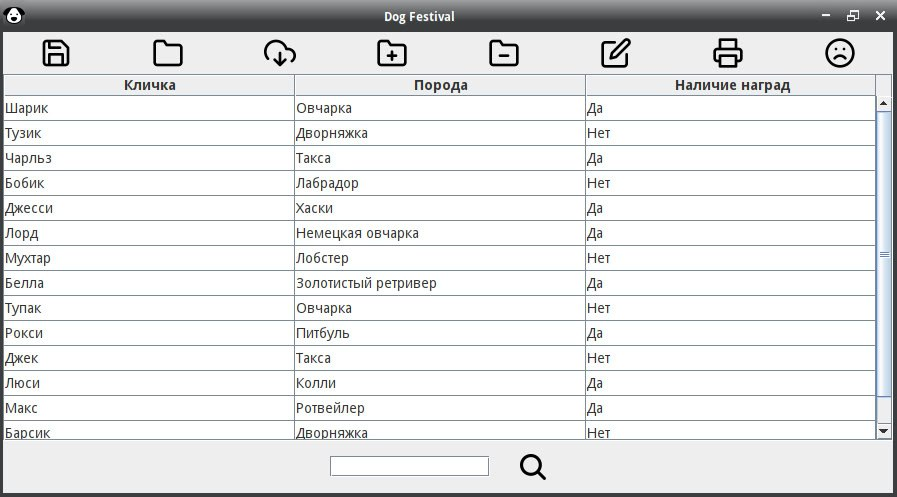
1. Спроектировать макет экранной формы, где определяется вид графических элементов и их размещение
2. Написать класс, визуализирующий экранную форму

**Описание назначения экранной формы**

Экранная форма предназначена для отображения списка собак, участвующих на выставке собак, она может менять свой размер на экране. Форма должна реализовывать следующие функции: загрузку списка собак из файла, сохранение списка собак в файле, добавление собаки в список, редактирование данных о собаке, удаление собаки из списка, поиск собаки по кличке или

породе, вывод списка собак.

**Макет экранной формы**

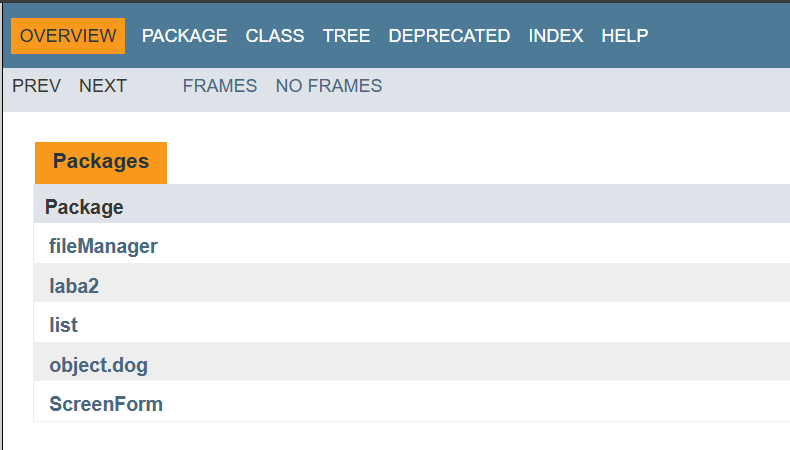
**Описание проверки работоспособности приложения**

\*Результат выполнения контрольного примера\*

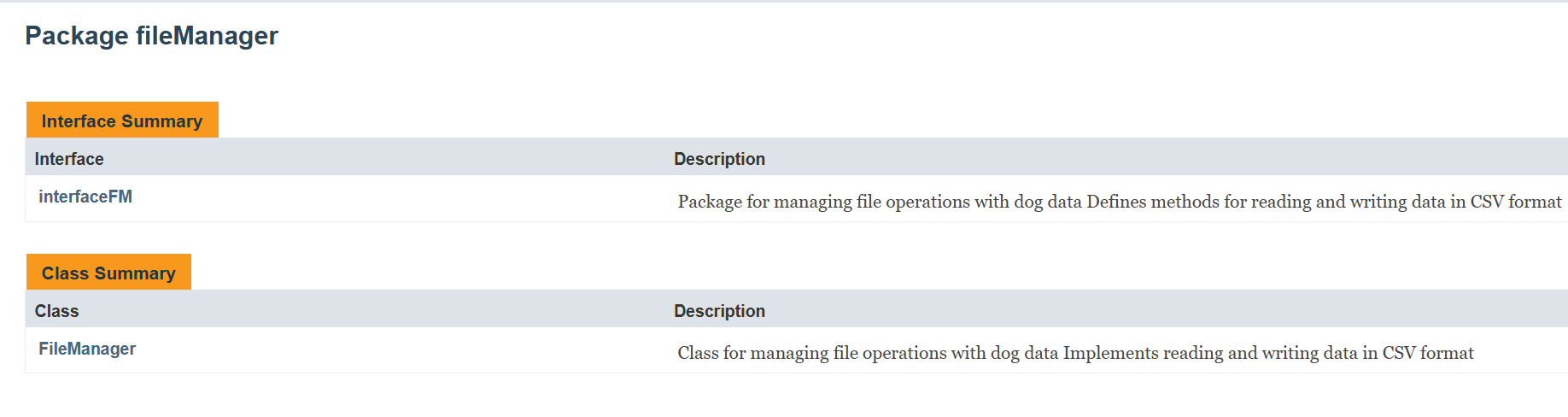
Анализ работоспособности:

* Приложение успешно выводит экранную форму на экран
* Данные из файла успешно выводятся в таблице
* Программа не требует отладки

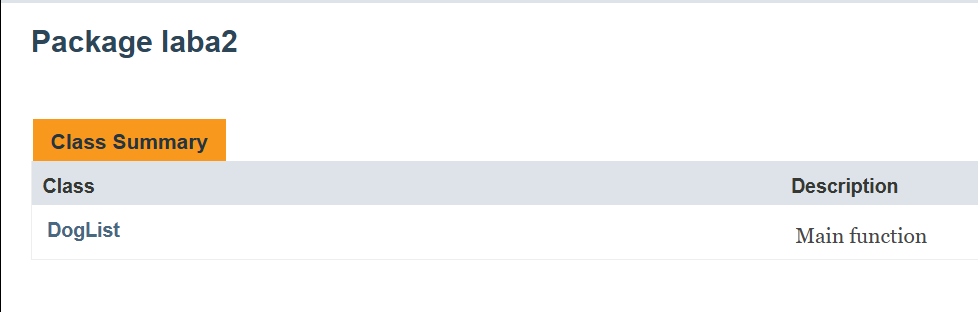
**Текст документации, сгенерированный Javadoc**



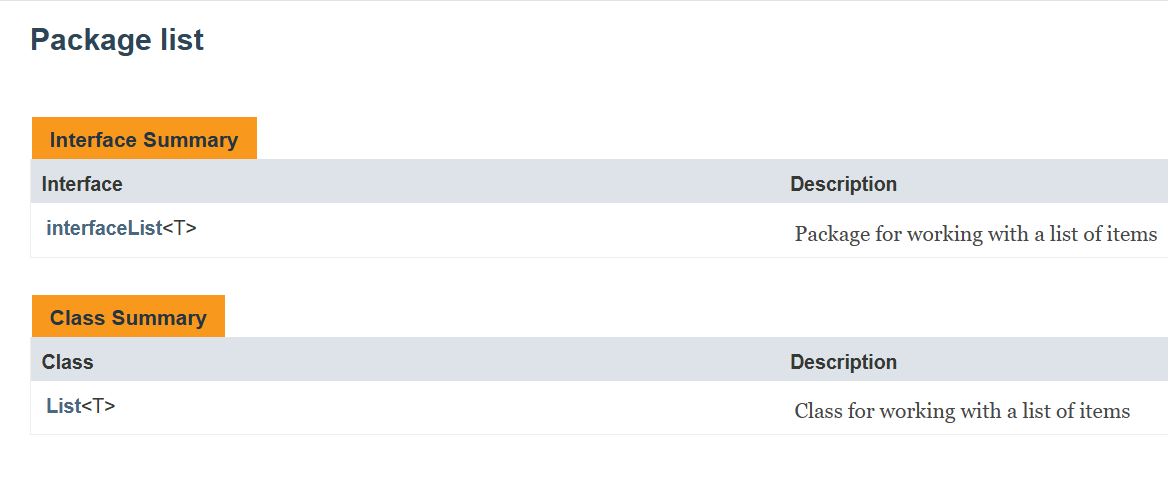
\*используемые пакеты\*



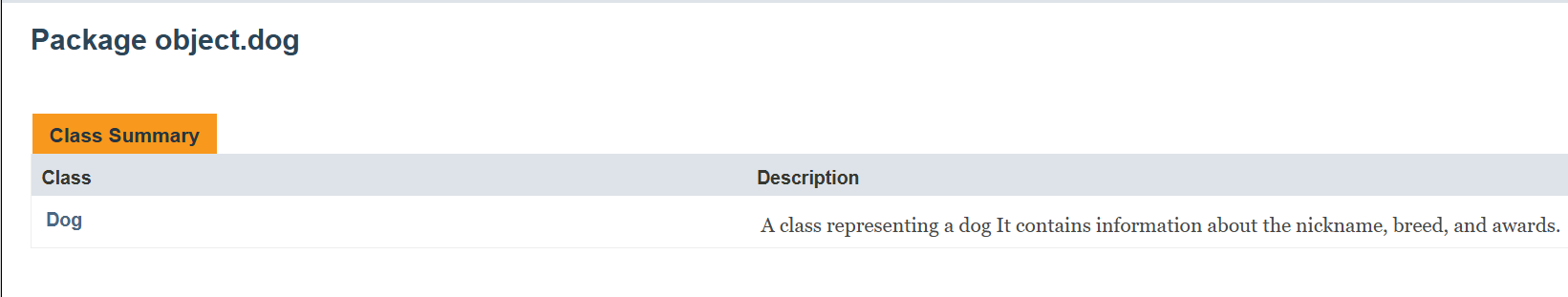
\*пакет для работы с файлом CSV\*

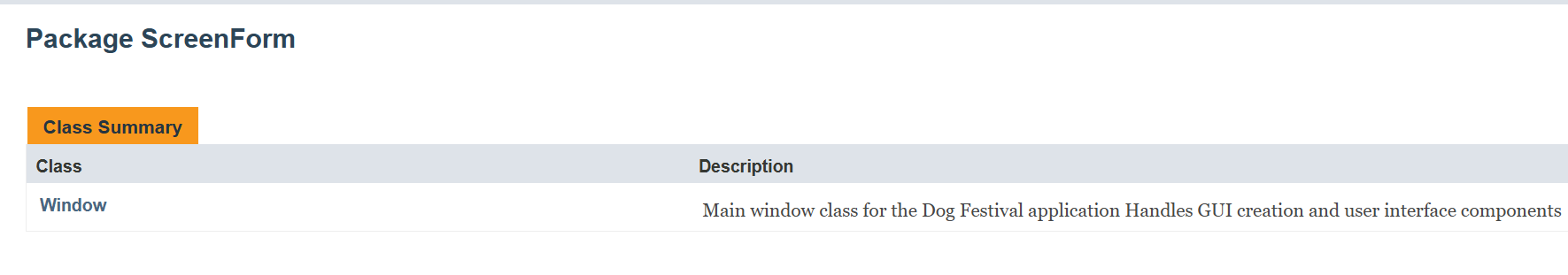


\*функция main\*



\*пакет для работы со списками\*

\*пакет, хранящий данные о собаке\*



\*главная экранная форма программы\*

**Тексты программы**

1) FileManager.java

package fileManager;

import list.List;

import object.dog.Dog;

import java.io.IOException;

import java.nio.file.\*;

import java.util.Arrays;

/\*\*

\* Class for managing file operations with dog data

\* Implements reading and writing data in CSV format

\* **@author** Vadim Ustinov

\* **@version** 1.0

\*/

public class FileManager implements interfaceFM

{

public FileManager() {

}

@Override

public List<Dog> inputFromCSV(String filePath) throws IOException

{

// Create a new empty list to store Dog objects

List<Dog> dogList = new List<>();

// Read all lines from the file into a list of strings

java.util.List<String> lines = Files.readAllLines(Paths.get(filePath));

// Variable to store award information

boolean hasAward;

// Process each line from the file

for(String line:lines)

{

// Split the line by ";" delimiter into an array of strings

String[] dogInfo = line.split(";");

// Convert the third element (index 2) to boolean: 1 = true, 0 = false

hasAward = Integer.parseInt(dogInfo[2]) == 1;

// Create a new Dog object and add it to the list

// dogInfo[0] - name, dogInfo[1] - breed, hasAward - award status

dogList.push\_back(new Dog(dogInfo[0], dogInfo[1], hasAward));

}

// Return the populated list of Dog objects

return dogList;

}

@Override

public void outputToCSV(String filePath, List<Dog> dogs) throws IOException

{

// Convert the list of Dog objects to an array of strings

// Each Dog object is converted to string representation

String[] listString = dogs.convToStr();

// Write the array of strings to the file, converting array to List

Files.write(Paths.get(filePath), Arrays.asList(listString));

}

}

2)DogList.java

package laba2;

import ScreenForm.Window;

import java.io.IOException;

/\*\*

\* Main function

\* @author Vadim Ustinov

\* @version 1.0

\*/

public class DogList

{

public static void main(String[] args) throws IOException

{

Window.show();

}

}

3)List.java

package list;

/\*\*

\* Class for working with a list of items

\* @param <T> type of list items

\* @author Vadim Ustinov

\* @version 1.0

\*/

public class List<T> implements interfaceList<T>

{

private static class Node<T>

{

T val;

Node<T> next;

Node(T data)

{

this.val = data;

this.next = null;

}

}

private Node<T> head;

private Node<T> end;

private int sz;

public List()

{

head = null;

end = null;

sz=0;

}

/\*\*

\* Checks if the index is valid for access operations

\* @param index index to check

\* @throws IndexOutOfBoundsException if index is out of range

\*/

private void checkIndex(int index)

{

if(index<0 || index>=sz)

{

throw new IndexOutOfBoundsException("Index " + index + " is out of range (list size: " + sz + ")\n");

}

}

/\*\*

\* Checks if the index is valid for insert operations

\* @param index index to check

\* @throws IndexOutOfBoundsException if index is out of range

\*/

private void checkIndexForInsert(int index)

{

if(index<0 || index>sz)

{

throw new IndexOutOfBoundsException("Index " + index + " is out of range (list size: " + sz + ")\n");

}

}

/\*\*

\* Adds an element to the end of the list

\* @param value value to add

\*/

@Override

public void push\_back(T value)

{

Node<T> newNode = new Node<>(value);

if(head==null)

{

head = newNode;

end = newNode;

head.next = null;

end.next = null;

}

else

{

end.next = newNode;

end = newNode;

}

sz++;

}

/\*\*

\* Adds an element to the beginning of the list

\* @param value value to add

\*/

@Override

public void push\_front(T value)

{

Node<T> newNode = new Node<>(value);

if(head == null)

{

head = newNode;

end = newNode;

head.next = null;

end.next = null;

}

else

{

Node<T> temp = head;

head = newNode;

head.next = temp;

}

sz++;

}

/\*\*

\* Inserts an element at the specified position

\* @param value value to insert

\* @param index insertion position

\*/

@Override

public void insert(T value, int index)

{

checkIndexForInsert(index);

if(index==0)

{

this.push\_front(value);

return;

}

if(index==sz)

{

this.push\_back(value);

return;

}

Node<T> newNode = new Node<>(value);

Node<T> curr = head;

for(int i=0;i<index-1;i++)

{

curr=curr.next;

}

newNode.next = curr.next;

curr.next = newNode;

sz++;

}

/\*\*

\* Returns the number of elements in the list

\* @return size of the list

\*/

@Override

public int getSize() {return sz;}

/\*\*

\* Returns the element at the specified position

\* @param index position of the element

\* @return element at the specified position

\*/

@Override

public T at(int index)

{

checkIndex(index);

Node<T> curr = head;

for(int i=0;i<index;i++)

{

curr = curr.next;

}

return curr.val;

}

/\*\*

\* Removes the element at the specified position

\* @param index position of the element to remove

\*/

@Override

public void remove(int index)

{

checkIndex(index);

if(index==0)

{

head = head.next;

if(head==null)

{

end=null;

}

sz--;

return;

}

Node<T> curr = head, temp;

for(int i=0;i<index-1;i++)

{

curr=curr.next;

}

temp = curr.next;

curr.next = temp.next;

if(curr.next == null)

{

end = curr;

}

temp = null;

sz--;

}

/\*\*

\* Checks if the list is empty

\* @return true if the list is empty, false otherwise

\*/

@Override

public boolean isEmpty()

{

if(head!=null)

{

return false;

}

return true;

}

/\*\*

\* Checks if the list contains the specified value

\* @param value value to search for

\* @return true if the value is found, false otherwise

\*/

@Override

public boolean contains(T value)

{

Node<T> curr = head;

while(curr!=null)

{

if(curr.val==value)

{

return true;

}

curr=curr.next;

}

return false;

}

/\*\*

\* Removes all elements from the list

\*/

@Override

public void clear()

{

sz=0;

head=null;

end=null;

}

/\*\*

\* Replaces the element at the specified position with a new value

\* @param value new value

\* @param index position of the element to replace

\*/

@Override

public void replace(T value, int index)

{

checkIndex(index);

Node<T> curr = head;

for(int i=0;i<index;i++)

{

curr = curr.next;

}

curr.val = value;

}

/\*\*

\* Converts the list to an array of strings

\* @return array of string representations of elements, or null if list is empty

\*/

@Override

public String[] convToStr()

{

if(sz==0)

{

return null;

}

String[] res = new String[sz];

Node<T> curr = head;

for(int i=0;i<sz;i++)

{

res[i] = curr.val.toString();

curr = curr.next;

}

return res;

}

}

4)Dog.java

package object.dog;

/\*\*

\* A class representing a dog

\* It contains information about the nickname, breed, and awards.

\* @author Vadim Ustinov

\* @version 1.0

\*/

public class Dog

{

private String name;

private String poroda;

private boolean hasAward;

/\*\*

\* Constructor for creating a Dog object

\* @param inpName is the dog's nickname

\* @param inpBreed dog breed

\* @param inpHasAward availability of awards

\*/

public Dog(String inpName, String inpBreed, boolean inpHasAward)

{

name = inpName;

poroda = inpBreed;

hasAward = inpHasAward;

}

/\*\*

\* Function returns the dog's name

\* @return dog's name

\*/

public String getName(){return name;}

/\*\*

\* Function returns the dog's breed

\* @return dog's breed

\*/

public String getBreed(){return poroda;}

/\*\*

\* Function checks for the dog's awards

\* @return true if there are rewards, otherwise false

\*/

public boolean hasAward(){return hasAward;}

/\*\*

\* Function sets a new dog name

\* @param newName new nickname

\*/

public void setName(String newName){name = newName;}

/\*\*

\* Function sets a new breed of dog

\* @param newPoroda new breed

\*/

public void setBreed(String newPoroda){poroda = newPoroda;}

/\*\*

\* Function sets the dog's awards

\* @param newHasAward availability of awards

\*/

public void setAwards(boolean newHasAward){hasAward = newHasAward;}

@Override

public String toString()

{

int hasAwardInt;

if(hasAward)

{

hasAwardInt = 1;

}

else

{

hasAwardInt = 0;

}

return name + ";" + poroda + ";" + hasAwardInt;

}

}

5)Window.java

package ScreenForm;

import fileManager.FileManager;

import java.io.IOException;

import list.List;

import object.dog.Dog;

import java.io.\*;

import javax.swing.\*;

import javax.swing.table.DefaultTableModel;

import java.awt.Image;

import java.awt.BorderLayout;

import java.awt.Font;

import java.awt.GridLayout;

import java.awt.FlowLayout;

/\*\*

\* Main window class for the Dog Festival application

\* Handles GUI creation and user interface components

\* @author Vadim Ustinov

\* @version 1.0

\*/

public class Window

{

private JFrame dogFestival;

private DefaultTableModel model;

private JScrollPane scroll;

private JTable dogsTable;

private JPanel buttonsPanel;

private JPanel inputPanel;

private JTextField textField;

private JButton[] buttons;

/\*\*

\* Displays the main application window with dog data

\* @throws IOException if there's an error reading/writing files

\*/

public static void show() throws IOException

{

// Initialize file manager and load dog data from CSV

FileManager fm = new FileManager();

List<Dog> dogs = new List<>();

dogs = fm.inputFromCSV("src/data/dogs.csv");

// Create main application frame

JFrame frame = new JFrame("Dog Festival");

ImageIcon icon = new ImageIcon("src/picts/dogIcon.png");

frame.setIconImage(icon.getImage());

// Create panels for buttons and input

JPanel buttonsPanel = new JPanel();

buttonsPanel.setLayout(new GridLayout(1, 8, 10, 10));

JPanel inputPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 10, 5));

// Define table column names

String[] columnNames = {

"Кличка", "Порода", "Наличие наград"

};

// Convert dog data to table format

String[][] data = new String[dogs.getSize()][];

String[] strListDogs = dogs.convToStr();

// Process each dog's data for display

for(int i=0;i<dogs.getSize();i++)

{

data[i] = strListDogs[i].split(";");

// Convert award indicator to readable format

if(Integer.parseInt(data[i][2]) == 1)

{

data[i][2] = "Yes";

}

else

{

data[i][2] = "No";

}

}

// Create table model and table

DefaultTableModel tableModel = new DefaultTableModel(data, columnNames);

JTable dogsTable = new JTable(tableModel);

// Configure table appearance

dogsTable.setFont(new Font("Arial", Font.PLAIN, 14));

dogsTable.getTableHeader().setFont(new Font("Arial", Font.BOLD, 14));

dogsTable.setRowHeight(25);

JScrollPane tableScrollPane = new JScrollPane(dogsTable);

// Create search text field

JTextField textField = new JTextField(12);

textField.setFont(new Font("Arial", Font.PLAIN, 14));

inputPanel.add(textField);

// Define button images and tooltips

String[] imagePaths = {

"src/picts/save.png",

"src/picts/folder\_documents.png",

"src/picts/cloud.png",

"src/picts/plus.png",

"src/picts/minus.png",

"src/picts/edit.png",

"src/picts/print.png",

"src/picts/exit.png",

"src/picts/search.png"

};

String[] tooltips = {

"Save",

"Open",

"Backup",

"Add",

"Remove",

"Edit",

"Print",

"Exit",

"Search"

};

JButton[] buttons = new JButton[9];

// Create and configure buttons

for(int i = 0; i < 9; i++)

{

ImageIcon buttonIcon = new ImageIcon(imagePaths[i]);

Image scaledImage = buttonIcon.getImage().getScaledInstance(32, 32, Image.SCALE\_SMOOTH);

ImageIcon scaledIcon = new ImageIcon(scaledImage);

buttons[i] = new JButton(scaledIcon);

buttons[i].setToolTipText(tooltips[i]);

buttons[i].setBorderPainted(false);

buttons[i].setContentAreaFilled(false);

buttons[i].setFocusPainted(false);

// Add search button to input panel, others to buttons panel

if(i == 8)

{

inputPanel.add(buttons[i]);

}

else

{

buttonsPanel.add(buttons[i]);

}

}

// Add components to frame

frame.add(buttonsPanel, BorderLayout.NORTH);

frame.add(tableScrollPane, BorderLayout.CENTER);

frame.add(inputPanel, BorderLayout.SOUTH);

// Configure frame properties

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setSize(900, 500);

frame.setLocationRelativeTo(null);

frame.setVisible(true);

// Save data back to CSV file

fm.outputToCSV("src/data/dogs.csv", dogs);

}

}