Буреев Фёдор Григорьевич P3107

**Лабораторная работа №8  
Вариант 26254**

1. Интерфейс должен быть реализован с помощью библиотеки Swing
2. Графический интерфейс клиентской части должен поддерживать **русский**, **португальский**, **каталанский** и **испанский (Сальвадор)** языки / локали. Должно обеспечиваться корректное отображение чисел, даты и времени в соответстии с локалью. Переключение языков должно происходить без перезапуска приложения. Локализованные ресурсы должны храниться в **файле свойств**.

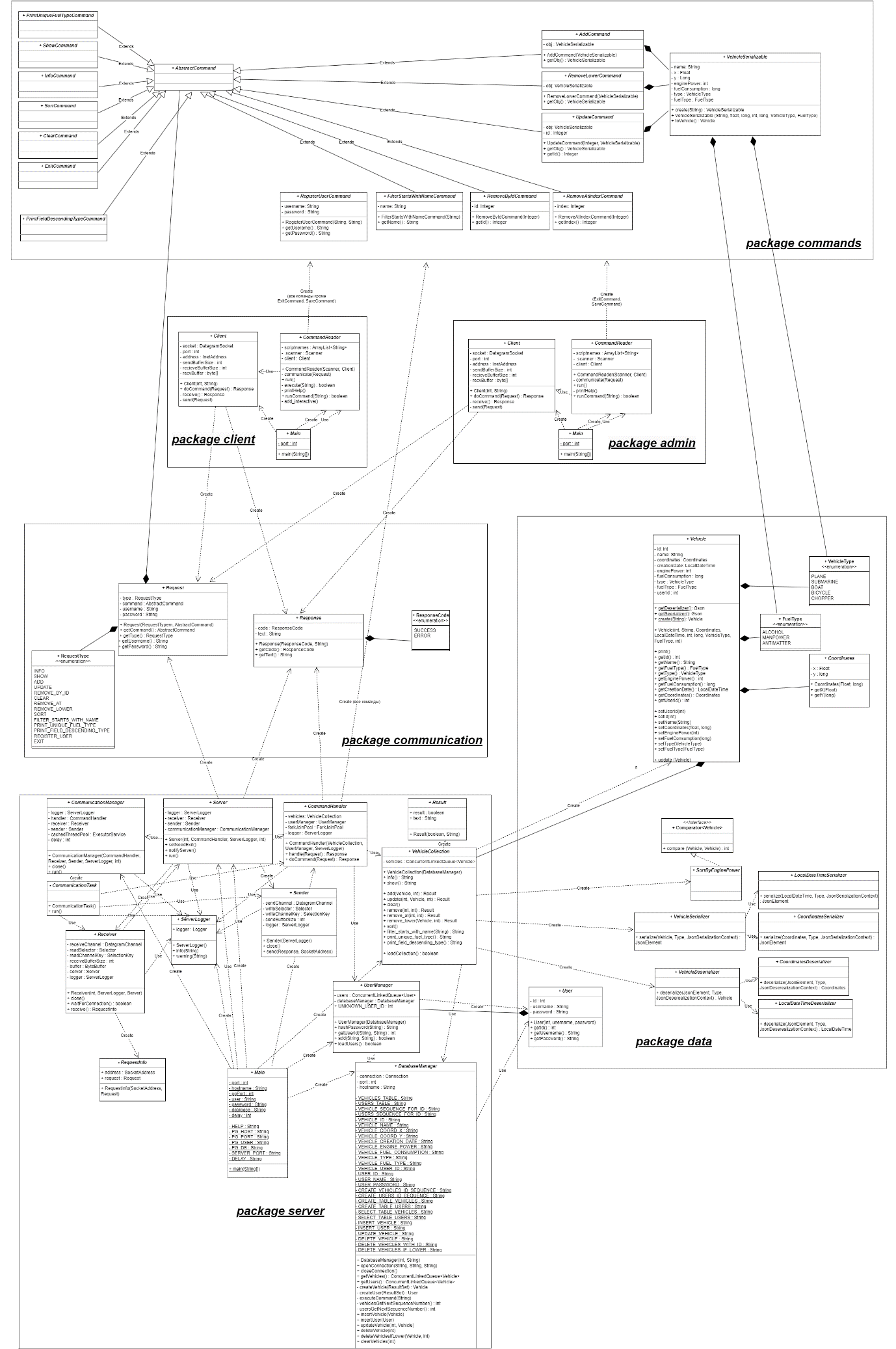
Доработать программу из [лабораторной работы №7](https://se.ifmo.ru/courses/programming#lab7) следующим образом:

Заменить консольный клиент на клиент с графическим интерфейсом пользователя(GUI). В функционал клиента должно входить:

1. Окно с авторизацией/регистрацией.
2. Отображение текущего пользователя.
3. Таблица, отображающая все объекты из коллекции
   1. Каждое поле объекта - отдельная колонка таблицы.
   2. Строки таблицы можно фильтровать/сортировать по значениям любой из колонок. Сортировку и фильтрацию значений столбцов реализовать с помощью Streams API.
4. Поддержка всех команд из предыдущих лабораторных работ.
5. Область, визуализирующую объекты коллекции
   1. Объекты должны быть нарисованы с помощью графических примитивов с использованием [Graphics](https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics.html), [Canvas](https://docs.oracle.com/javase/8/javafx/api/javafx/scene/canvas/Canvas.html) или аналогичных средств графической библиотеки.
   2. При визуализации использовать данные о координатах и размерах объекта.
   3. Объекты от разных пользователей должны быть нарисованы разными цветами.
   4. При нажатии на объект должна выводиться информация об этом объекте.
   5. При добавлении/удалении/изменении объекта, он должен **автоматически** появиться/исчезнуть/измениться  на области как владельца, так и всех других клиентов.
   6. При отрисовке объекта должна воспроизводиться согласованная с преподавателем **анимация**.
6. Возможность редактирования отдельных полей любого из объектов (принадлежащего пользователю). Переход к редактированию объекта возможен из таблицы с общим списком объектов и из области с визуализацией объекта.
7. Возможность удаления выбранного объекта (даже если команды remove ранее не было).

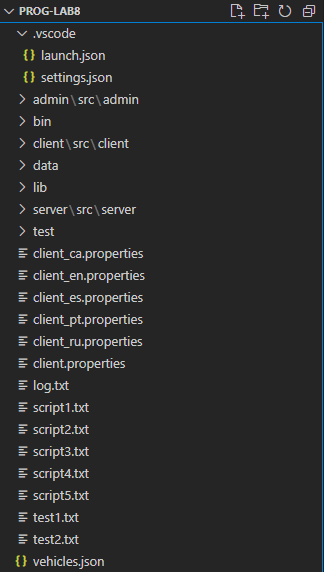
Перед непосредственной разработкой приложения **необходимо** согласовать прототип интерфейса с преподавателем. Прототип интерфейса должен быть создан с помощью средства для построения прототипов интерфейсов(mockplus, draw.io, etc.)

Диаграмма разработанной программы:

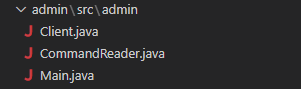
****

**3. Исходный код программы**

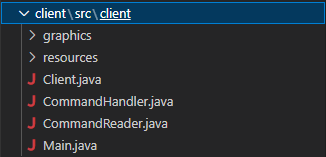
***Список пакетов:***

****

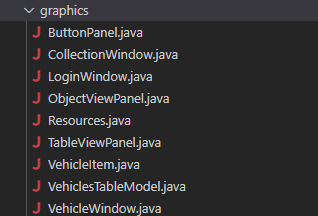
*package admin :*

**

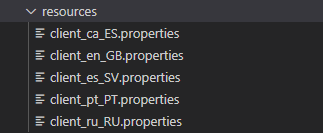
*package client:*

**

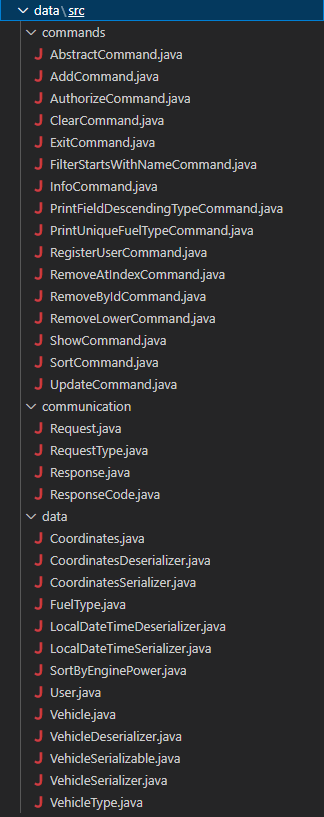
*package client.graphics:*

**

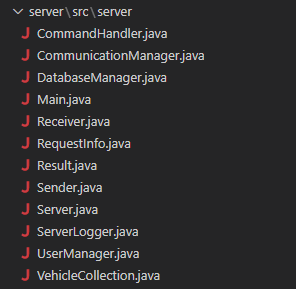
*package client.resources:*

**

*packages commands, communication, data:*

**

*package server:*

**

***Код:***

# package admin :

## Client.java

package admin;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.net.SocketException;

import java.net.UnknownHostException;

import java.net.SocketTimeoutException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.io.ByteArrayInputStream;

import java.io.ByteArrayOutputStream;

import communication.Request;

import communication.Response;

public class Client {

private DatagramSocket socket;

private int port;

private InetAddress address;

private int sendBufferSize = 10000;

private int receiveBufferSize = 10000;

private byte[] recvBuffer = new byte[receiveBufferSize];

public Client(int port, String hostname) throws SocketException, UnknownHostException {

socket = new DatagramSocket();

socket.setSoTimeout(1000);

this.port = port;

address = InetAddress.getByName(hostname);

}

public Response doCommand(Request request) {

send(request);

return receive();

}

private Response receive() {

try {

DatagramPacket packet = new DatagramPacket(recvBuffer, recvBuffer.length);

socket.receive(packet);

ObjectInputStream stream = new ObjectInputStream(new ByteArrayInputStream(packet.getData(), packet.getOffset(), packet.getLength()));

Response response = (Response) stream.readObject();

stream.close();

return response;

} catch (SocketTimeoutException e) {

System.out.println("Похоже, связь с сервером прервалась.");

} catch (Exception e) {

System.out.println("Не получилось прочитать ответ из сокета : " + e);

}

return null;

}

private void send(Request request) {

try {

ByteArrayOutputStream byteStream = new ByteArrayOutputStream(sendBufferSize);

ObjectOutputStream stream = new ObjectOutputStream(byteStream);

stream.writeObject(request);

stream.flush();

byte[] sendBuffer = byteStream.toByteArray();

DatagramPacket packet = new DatagramPacket(sendBuffer, sendBuffer.length, address, port);

socket.send(packet);

stream.close();

}

catch (Exception e) {

System.out.println("Не получилось записать запрос в сокет : " + e);

}

}

}

## CommandReader.java

package admin;

import java.util.Scanner;

import commands.\*;

import communication.Response;

import communication.Request;

import communication.RequestType;

public class CommandReader {

private Scanner scanner;

private Client client;

public CommandReader(Scanner scanner, Client client) {

this.scanner = scanner;

this.client = client;

}

private void communicate(Request request) {

if(request != null) {

try {

Response response = client.doCommand(request);

if(response != null) {

System.out.println("Получили ответ (" + response.getCode() + ") : \n" + response.getText());

}

} catch (Exception e) {

System.out.println(e);

}

}

}

public void run()

{

String command = "";

while(true) {

System.out.println("Введите команду управления сервером:");

try {

command = scanner.nextLine();

if(command != null) {

command = command.trim();

}

if (runCommand(command)) {

break;

}

} catch (Exception e) {

System.out.println("Неправильная команда");

}

}

}

private void printHelp() {

System.out.println("help : вывести справку по доступным командам");

System.out.println("exit : завершить программу");

System.out.println("exitServer : завершить работу сервера");

System.out.println("save : сохранить коллекцию в файл");

}

public boolean runCommand (String command)

{

boolean needExit = false;

if(command == null) {

return needExit;

}

if(command.isEmpty()) {

return needExit;

}

String[] args;

args = command.split(" ");

if(args.length == 0) {

return needExit;

}

switch (args[0]) {

case "exit":

needExit = true;

break;

case "help":

printHelp();

break;

case "exitServer":

communicate(new Request(RequestType.EXIT, new ExitCommand()));

break;

case "save":

communicate(new Request(RequestType.SAVE, new SaveCommand()));

break;

default:

System.out.println("Неопознанная команда. Наберите 'help' для справки.");

break;

}

return needExit;

}

}

## Main.java

package admin;

import java.util.Scanner;

public class Main {

private static int port = 2000;

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

try {

System.out.println("Запускаю админ.");

Client client = new Client(port, "localhost");

CommandReader reader = new CommandReader(scanner, client);

reader.run();

} catch (Exception e) {

System.out.println("Аварийное завершение работы админа : " + e);

}

scanner.close();

}

}

# package client.graphics:

## ButtonPanel.java

package client.graphics;

import javax.swing.JPanel;

import javax.swing.JButton;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.GridLayout;

import java.awt.Color;

import java.awt.Font;

import javax.swing.JOptionPane;

import java.lang.Exception;

import client.CommandHandler;

public class ButtonPanel extends JPanel {

static private int FONT\_SIZE = 20;

static private String FONT\_NAME = "Arial";

JButton helpButton;

JButton infoButton;

JButton addButton;

JButton updateButton;

JButton removeByIdButton;

JButton removeAtIndexButton;

JButton removeLowerButton;

JButton clearButton;

JButton executeScriptButton;

JButton sortButton;

JButton filterStartsWithNameButton;

JButton printUniqueFuelTypeButton;

JButton printFieldDescendingTypeButton;

public ButtonPanel(VehiclesTableModel model, CommandHandler handler) {

setBackground(Color.lightGray);

setBounds(25,25,300,850);

setLayout(new GridLayout(13, 1, 0, 0));

//setBorder(BorderFactory.createLineBorder(Color.black));

helpButton = new JButton(Resources.getString(Resources.BUTTON\_HELP));

helpButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

handler.doHelp();

}

});

helpButton.setFont(new Font(FONT\_NAME, Font.PLAIN, FONT\_SIZE));

infoButton = new JButton(Resources.getString(Resources.BUTTON\_INFO));

infoButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

handler.doInfo();

}

});

infoButton.setFont(new Font(FONT\_NAME, Font.PLAIN, FONT\_SIZE));

addButton = new JButton(Resources.getString(Resources.BUTTON\_ADD));

addButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

VehicleWindow w = new VehicleWindow();

if (w.run()) {

handler.doAdd(w.getVehicleItem().createSerializable());

}

}

});

addButton.setFont(new Font(FONT\_NAME, Font.PLAIN, FONT\_SIZE));

updateButton = new JButton(Resources.getString(Resources.BUTTON\_UPDATE));

updateButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

String str = JOptionPane.showInputDialog(Resources.getString(Resources.TEXT\_ENTER\_ID\_UPDATE));

try {

if(str != null) {

int id = Integer.parseInt(str);

VehicleItem item = model.getObjById(id);

if(item == null) {

JOptionPane.showMessageDialog(null, Resources.getString(Resources.TEXT\_ID\_IS\_NOT\_FOUND),

Resources.getString(Resources.ERROR\_TITLE), JOptionPane.ERROR\_MESSAGE);

} else {

VehicleWindow w = new VehicleWindow(item, true);

if (w.run()) {

handler.doUpdate(id, w.getVehicleItem().createSerializable());

}

}

}

} catch (Exception ex) {

System.out.println(ex);

JOptionPane.showMessageDialog(null, Resources.getString(Resources.TEXT\_INVALID\_ID),

Resources.getString(Resources.ERROR\_TITLE), JOptionPane.ERROR\_MESSAGE);

}

}

});

updateButton.setFont(new Font(FONT\_NAME, Font.PLAIN, FONT\_SIZE));

removeByIdButton = new JButton(Resources.getString(Resources.BUTTON\_REMOVE\_BY\_ID));

removeByIdButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

String str = JOptionPane.showInputDialog(Resources.getString(Resources.TEXT\_ENTER\_ID\_REMOVE));

try {

if(str != null) {

int id = Integer.parseInt(str);

handler.doRemoveItemById(id);

}

} catch (Exception ex) {

JOptionPane.showMessageDialog(null, Resources.getString(Resources.TEXT\_INVALID\_ID),

Resources.getString(Resources.ERROR\_TITLE), JOptionPane.ERROR\_MESSAGE);

}

}

});

removeByIdButton.setFont(new Font(FONT\_NAME, Font.PLAIN, FONT\_SIZE));

removeAtIndexButton = new JButton(Resources.getString(Resources.BUTTON\_REMOVE\_BY\_INDEX));

removeAtIndexButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

String str = JOptionPane.showInputDialog(Resources.getString(Resources.TEXT\_ENTER\_INDEX\_REMOVE));

try {

if(str != null) {

int id = Integer.parseInt(str);

handler.doRemoveItemByIndex(id);

}

} catch (Exception ex) {

JOptionPane.showMessageDialog(null, Resources.getString(Resources.TEXT\_INVALID\_INDEX),

Resources.getString(Resources.ERROR\_TITLE), JOptionPane.ERROR\_MESSAGE);

}

}

});

removeAtIndexButton.setFont(new Font(FONT\_NAME, Font.PLAIN, FONT\_SIZE));

removeLowerButton = new JButton(Resources.getString(Resources.BUTTON\_REMOVE\_LOWER));

removeLowerButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

String str = JOptionPane.showInputDialog(Resources.getString(Resources.TEXT\_ENTER\_ID\_REMOVE\_LOWER));

try {

if(str != null) {

int id = Integer.parseInt(str);

VehicleItem item = model.getObjById(id);

if(item == null) {

JOptionPane.showMessageDialog(null, Resources.getString(Resources.TEXT\_ID\_IS\_NOT\_FOUND),

Resources.getString(Resources.ERROR\_TITLE), JOptionPane.ERROR\_MESSAGE);

} else {

handler.doRemoveLowerItems(item.createSerializable());

}

}

} catch (Exception ex) {

JOptionPane.showMessageDialog(null, Resources.getString(Resources.TEXT\_INVALID\_ID),

Resources.getString(Resources.ERROR\_TITLE), JOptionPane.ERROR\_MESSAGE);

}

}

});

removeLowerButton.setFont(new Font(FONT\_NAME, Font.PLAIN, FONT\_SIZE));

clearButton = new JButton(Resources.getString(Resources.BUTTON\_CLEAR));

clearButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

handler.doClear();

}

});

clearButton.setFont(new Font(FONT\_NAME, Font.PLAIN, FONT\_SIZE));

executeScriptButton = new JButton(Resources.getString(Resources.BUTTON\_EXECUTE\_SCRIPT));

executeScriptButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

String str = JOptionPane.showInputDialog(Resources.getString(Resources.TEXT\_ENTER\_FILE\_NAME));

if(str != null) {

handler.doExecuteScript(str);

}

}

});

executeScriptButton.setFont(new Font(FONT\_NAME, Font.PLAIN, FONT\_SIZE));

sortButton = new JButton(Resources.getString(Resources.BUTTON\_SORT));

sortButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

//new TestTable();

}

});

sortButton.setFont(new Font(FONT\_NAME, Font.PLAIN, FONT\_SIZE));

filterStartsWithNameButton = new JButton(Resources.getString(Resources.BUTTON\_FILTER\_STARTS\_WITH\_NAME));

filterStartsWithNameButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

//dispose();

}

});

filterStartsWithNameButton.setFont(new Font(FONT\_NAME, Font.PLAIN, FONT\_SIZE));

printUniqueFuelTypeButton = new JButton(Resources.getString(Resources.BUTTON\_PRINT\_UNIQUE\_FUEL\_TYPES));

printUniqueFuelTypeButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

handler.doPrintUniqueFuelType();

}

});

printUniqueFuelTypeButton.setFont(new Font(FONT\_NAME, Font.PLAIN, FONT\_SIZE));

printFieldDescendingTypeButton = new JButton(Resources.getString(Resources.BUTTON\_PRINT\_FIELD\_DESCENDING\_TYPE));

printFieldDescendingTypeButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

//dispose();

}

});

printFieldDescendingTypeButton.setFont(new Font(FONT\_NAME, Font.PLAIN, FONT\_SIZE));

add(helpButton);

add(infoButton);

add(addButton);

add(updateButton);

add(removeByIdButton);

add(removeAtIndexButton);

add(removeLowerButton);

add(clearButton);

add(executeScriptButton);

add(sortButton);

add(filterStartsWithNameButton);

add(printUniqueFuelTypeButton);

add(printFieldDescendingTypeButton);

}

public void updateLocale() {

helpButton.setText(Resources.getString(Resources.BUTTON\_HELP));

infoButton.setText(Resources.getString(Resources.BUTTON\_INFO));

addButton.setText(Resources.getString(Resources.BUTTON\_ADD));

updateButton.setText(Resources.getString(Resources.BUTTON\_UPDATE));

removeByIdButton.setText(Resources.getString(Resources.BUTTON\_REMOVE\_BY\_ID));

removeAtIndexButton.setText(Resources.getString(Resources.BUTTON\_REMOVE\_BY\_INDEX));

removeLowerButton.setText(Resources.getString(Resources.BUTTON\_REMOVE\_LOWER));

clearButton.setText(Resources.getString(Resources.BUTTON\_CLEAR));

executeScriptButton.setText(Resources.getString(Resources.BUTTON\_EXECUTE\_SCRIPT));

sortButton.setText(Resources.getString(Resources.BUTTON\_SORT));

filterStartsWithNameButton.setText(Resources.getString(Resources.BUTTON\_FILTER\_STARTS\_WITH\_NAME));

printUniqueFuelTypeButton.setText(Resources.getString(Resources.BUTTON\_PRINT\_UNIQUE\_FUEL\_TYPES));

printFieldDescendingTypeButton.setText(Resources.getString(Resources.BUTTON\_PRINT\_FIELD\_DESCENDING\_TYPE));

}

}

## CollectionWindow.java

package client.graphics;

import javax.swing.JFrame;

import javax.swing.JPanel;

import java.awt.Color;

import client.Client;

import client.CommandHandler;

import java.awt.event.ActionListener;

import java.awt.event.ActionEvent;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

import javax.swing.JComboBox;

public class CollectionWindow extends JFrame{

private CommandHandler handler;

private ObjectViewPanel objectView;

private TableViewPanel tableView;

private ButtonPanel buttonPanel;

public CollectionWindow(VehiclesTableModel model, Client client, CommandHandler handler) {

this.handler = handler;

//setTitle("VEHICLE COLLECTION (User: " + client.getUsername() + ")");

setTitle(Resources.getString(Resources.WINDOW\_TITLE) +

" (" + Resources.getString(Resources.WINDOW\_TITLE\_USER) + ": " + client.getUsername() + ")");

setExtendedState(JFrame.MAXIMIZED\_BOTH);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

this.addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent evt) {

//System.out.println("Закрываем окно");

handler.doStop();

objectView.stop();

dispose();

System.exit(0);

}

});

JComboBox<String> langBox = new JComboBox<String>(Resources.languages);

langBox.setEditable(false);

langBox.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

Resources.setLanguage(langBox.getSelectedIndex());

setTitle(Resources.getString(Resources.WINDOW\_TITLE) +

" (" + Resources.getString(Resources.WINDOW\_TITLE\_USER) + ": " + client.getUsername() + ")");

tableView.updateLocale();

buttonPanel.updateLocale();

}

});

JPanel p = new JPanel();

p.setBackground(Color.LIGHT\_GRAY);

buttonPanel = new ButtonPanel(model, handler);

tableView = new TableViewPanel(model, handler, p, this);

objectView = new ObjectViewPanel(model, handler);

getContentPane().add(buttonPanel);

getContentPane().add(tableView);

getContentPane().add(objectView);

p.add(langBox);

getContentPane().add(p);

}

public void run() {

handler.doStart();

//pack();

setVisible(true);

}

}

## LoginWindow.java

package client.graphics;

import javax.swing.JFrame;

import javax.swing.JDialog;

import javax.swing.JLabel;

import javax.swing.JTextField;

import javax.swing.SpringLayout;

import javax.swing.JButton;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.KeyListener;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

import java.awt.event.KeyEvent;

import java.awt.Container;

import java.awt.Component;

public class LoginWindow extends JFrame{

private String username;

private String password;

private JDialog dialog;

public String getUsername() {

return username;

}

public String getPassword() {

return password;

}

public LoginWindow() {

username = "";

password = "";

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

dialog = new JDialog(this, Resources.getString(Resources.AUTH\_TITLE), true);

dialog.setResizable(false);

dialog.addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent evt) {

//System.out.println("Закрываем окно");

dispose();

System.exit(0);

}

});

dialog.setDefaultCloseOperation(JDialog.DISPOSE\_ON\_CLOSE);

dialog.setSize(300,300);

dialog.setLocationRelativeTo(null);

Container contentPane = dialog.getContentPane();

SpringLayout layout = new SpringLayout();

contentPane.setLayout(layout);

JLabel label1 = new JLabel(Resources.getString(Resources.LABEL\_LOGIN) + ":");

JTextField tf1 = new JTextField(15);

tf1.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

username = tf1.getText();

Component c = (Component)e.getSource();

c.transferFocus();

}

});

contentPane.add(label1);

contentPane.add(tf1);

layout.putConstraint(SpringLayout.WEST, label1, 5, SpringLayout.WEST, contentPane);

layout.putConstraint(SpringLayout.NORTH, label1, 5, SpringLayout.NORTH, contentPane);

layout.putConstraint(SpringLayout.WEST, tf1, 100, SpringLayout.WEST, contentPane);

layout.putConstraint(SpringLayout.NORTH, tf1, 5, SpringLayout.NORTH, contentPane);

JLabel label2 = new JLabel(Resources.getString(Resources.LABEL\_PASSWORD) + ":");

JTextField tf2 = new JTextField(15);

tf2.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

password = tf2.getText();

Component c = (Component)e.getSource();

c.transferFocus();

}

});

contentPane.add(label2);

contentPane.add(tf2);

layout.putConstraint(SpringLayout.WEST, label2, 5, SpringLayout.WEST, contentPane);

layout.putConstraint(SpringLayout.NORTH, label2, 5, SpringLayout.SOUTH, label1);

layout.putConstraint(SpringLayout.WEST, tf2, 100, SpringLayout.WEST, contentPane);

layout.putConstraint(SpringLayout.NORTH, tf2, 5, SpringLayout.SOUTH, tf1);

JButton button = new JButton(Resources.getString(Resources.BUTTON\_ENTER));

button.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

username = tf1.getText();

password = tf2.getText();

if(username.isEmpty() || password.isEmpty()) {

System.out.println(Resources.getString(Resources.TEXT\_ENTER\_LOGIN\_AND\_PASSWORD));

} else {

dialog.dispose();

dispose();

}

}

});

button.addKeyListener( new KeyListener() {

public void keyPressed(KeyEvent e) {

if (e.getKeyCode()==KeyEvent.VK\_ENTER){

username = tf1.getText();

password = tf2.getText();

if(username.isEmpty() || password.isEmpty()) {

System.out.println(Resources.getString(Resources.TEXT\_ENTER\_LOGIN\_AND\_PASSWORD));

} else {

dialog.dispose();

dispose();

}

}

}

public void keyTyped(KeyEvent k) {}

public void keyReleased(KeyEvent k) {}

});

contentPane.add(button);

layout.putConstraint(SpringLayout.WEST, button, 100, SpringLayout.WEST, contentPane);

layout.putConstraint(SpringLayout.NORTH, button, 5, SpringLayout.SOUTH, tf2);

// layout constraints

layout.putConstraint(SpringLayout.EAST, contentPane, 50, SpringLayout.EAST, tf2);

layout.putConstraint(SpringLayout.SOUTH, contentPane, 50, SpringLayout.SOUTH, tf2);

}

public void run() {

dialog.pack();

dialog.setVisible(true);

}

}

## ObjectViewPanel.java

package client.graphics;

import javax.swing.JPanel;

import javax.swing.BorderFactory;

import javax.swing.JOptionPane;

import javax.swing.Timer;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.event.ActionListener;

import java.awt.event.MouseEvent;

import java.awt.event.ActionEvent;

import java.awt.event.MouseListener;

import javax.swing.event.TableModelEvent;

import javax.swing.event.TableModelListener;

import java.lang.Math;

import java.util.concurrent.ConcurrentLinkedQueue;

import client.CommandHandler;

public class ObjectViewPanel extends JPanel {

private static int minSize = 20;

private static int maxSize = 50;

private static int maxX = 1325 - maxSize - 5;//662;

private static int maxY = 420 - maxSize - 5;//210;

private static int maxEnginePower = 100;

private static float maxOrigX = 1000;

private static long maxOrigY = 1000;

private static Color[] colors = {Color.RED, Color.GREEN, Color.BLUE, Color.YELLOW, Color.MAGENTA, Color.ORANGE, Color.CYAN, Color.PINK};

VehiclesTableModel model;

ConcurrentLinkedQueue<GItem> gItems;

Timer t;

private class GItem {

private int id;

private int x;

private int y;

private int size;

private Color color;

private float scale;

private float delta;

public GItem(VehicleItem item) {

id = item.getId();

x = scaleX(item.getX());

y = scaleY(item.getY());

size = scaleSize(item.getEnginePower());

color = (item.getUserId().equals(model.getUserId())) ? Color.WHITE : getColor(item.getUserId());

scale = 0.1f;

delta = 0.1f;

System.out.println("gItem id=" + id + " name=" + item.getName() + " x=" + item.getX() + "->" + x);

System.out.println("gItem id=" + id + " name=" + item.getName() + " y=" + item.getY() + "->" + y + "\n");

}

public int getId() {

return id;

}

public float getScale() {

return scale;

}

public int getX() {

return x;

}

public int getY() {

return y;

}

public int getSize() {

return size;

}

public void setX(Float x) {

this.x = scaleX(x);

}

public void setY(Long y) {

this.y = scaleY(y);

}

public void setSize(Integer enginePower) {

this.size = scaleSize(enginePower);

}

public void setDelta(float delta) {

this.delta = delta;

}

public void update() {

if (delta != 0.0f) {

float newScale = scale + delta;

if (newScale > 1.0f) {

delta = 0.0f;

scale = 1.0f;

} else if (newScale < 0.0f) {

delta = 0.0f;

scale = 0.0f;

} else {

scale = newScale;

}

}

}

public void draw(Graphics2D graphics) {

if (scale > 0.0f) {

int diameter = (int) (size \* scale);

graphics.setColor(color);

graphics.fillOval( x, y, diameter, diameter);

graphics.setColor(Color.BLACK);

graphics.drawOval( x, y, diameter, diameter);

}

}

}

public ObjectViewPanel(VehiclesTableModel model, CommandHandler handler) {

this.model = model;

gItems = new ConcurrentLinkedQueue<GItem>();

model.setObjectView(this);

setBackground(Color.lightGray);

setBounds(350,455,1325,420);

setBorder(BorderFactory.createLoweredBevelBorder());

t = new Timer(100, null);

t.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

//System.out.println("timer");

for(GItem gItem : gItems) {

gItem.update();

if (gItem.getScale() == 0.0f) {

System.out.println("Removing gItem id=" + gItem.getId());

gItems.remove(gItem);

System.out.println("GItem removed.");

}

}

repaint();

}

});

t.start();

model.addTableModelListener(new TableModelListener() {

@Override

public void tableChanged(TableModelEvent e) {

int column = e.getColumn();

int firstRow = e.getFirstRow();

int lastRow = e.getLastRow();

boolean updateGItem = (column == 2) || (column == 3) || (column == 5);

System.out.println("tablemodellistener : " + e.getType() + " (" + firstRow + " - " + lastRow + ")");

if (e.getType() == TableModelEvent.UPDATE) {

for(int row = firstRow; row <= lastRow; row++) {

System.out.println("UPDATE ROW " + row);

VehicleItem item = model.getObjByIndex(row);

if(updateGItem) {

GItem gItem = get(item.getId());

if (gItem != null) {

if (column == 2) {

gItem.setX(item.getX());

} else if(column == 3) {

gItem.setY(item.getY());

} else if(column == 5) {

gItem.setSize(item.getEnginePower());

}

}

}

//handler.doUpdate(item.getId(), item.createSerializable());

/\*

GItem gItem = get((Integer) model.getValueAt(row, 0));

if (gItem != null) {

if (column == 2) {

gItem.setX((Float) model.getValueAt(row, column));

} else if(column == 3) {

gItem.setY((Long) model.getValueAt(row, column));

} else if(column == 5) {

gItem.setSize((Integer) model.getValueAt(row, column));

}

}\*/

}

}

}

});

addMouseListener(new MouseListener() {

public void mousePressed(MouseEvent e) {

//System.out.println("Mouse pressed; # of clicks: " + e.getClickCount() + " " + e);

}

public void mouseReleased(MouseEvent e) {

//System.out.println("Mouse released; # of clicks: " + e.getClickCount() + " " + e);

}

public void mouseEntered(MouseEvent e) {

//System.out.println("Mouse entered " + e);

}

public void mouseExited(MouseEvent e) {

//System.out.println("Mouse exited " + e);

}

public void mouseClicked(MouseEvent e) {

//System.out.println("X: " + e.getX() + " Y: " + e.getY());

int x = e.getX();

int y = e.getY();

for(GItem gItem : gItems) {

if( (x >= gItem.getX()) && (x <= (gItem.getX() + gItem.getSize())) && (y >= gItem.getY()) && (y <= (gItem.getY() + gItem.getSize())) ) {

//System.out.println("mouse clicked on gItem " + gItem.getId());

int id = gItem.getId();

VehicleItem item = model.getObjById(id);

if(item == null) {

JOptionPane.showMessageDialog(null, "Элемент с Id=" + id + " не найден.", "Error", JOptionPane.ERROR\_MESSAGE);

} else {

boolean isEdit = item.getUserId() == model.getUserId();

VehicleWindow w = new VehicleWindow(item, isEdit);

if (w.run()) {

if(isEdit) {

handler.doUpdate(id, w.getVehicleItem().createSerializable());

}

};

}

}

}

}

});

}

public void stop() {

t.stop();

}

private GItem get(int id) {

for (GItem gItem : gItems) {

if (gItem.getId() == id) {

return gItem;

}

}

return null;

}

public void add(VehicleItem item) {

System.out.println("Add gItem " + item.getId());

gItems.add(new GItem(item));

}

public void remove(VehicleItem item) {

System.out.println("Remove gItem " + item.getId());

GItem gItem = get(item.getId());

if (gItem != null) {

gItem.setDelta(-0.1f);

}

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2d = (Graphics2D) g;

//System.out.println("paintComponent");

for(GItem gItem : gItems) {

gItem.draw(g2d);

}

}

private static int scaleX(Float x) {

//return maxX + (x.intValue() % maxX);

float newX = x;

newX = (newX > 0) ? Math.min(newX, maxOrigX) : Math.max(newX, -maxOrigX);

newX += maxOrigX;

newX = newX / (2 \* maxOrigX) \* (float)maxX;

return (int) newX + 2;

}

private static int scaleY(Long y) {

//return maxY + (y.intValue() % maxY);

long newY = y;

newY = (newY > 0) ? Math.min(newY, maxOrigY) : Math.max(newY, -maxOrigY);

newY = -newY;

newY += maxOrigY;

newY = (newY \* (long)maxY) / (2 \* maxOrigY);

return (int) newY + 2;

}

private static Color getColor(int id) {

return colors[id % colors.length];

}

private static int scaleSize(int enginePower) {

if(enginePower > maxEnginePower) {

return maxSize;

} else {

return minSize + enginePower / (maxEnginePower / (maxSize - minSize));

}

}

}

## Resources.java

package client.graphics;

import java.util.Locale;

import java.util.ResourceBundle;

import java.util.Properties;

import java.io.FileOutputStream;

import java.io.OutputStream;

public class Resources {

    public static String name = "client.resources.client";

    public static String[] languages = {"русский (RU)", "portuguesa (PT)", "español (ES)", "català (CA)", "english (EN)"};

    public static String WINDOW\_TITLE = "windowTitle";

    public static String WINDOW\_TITLE\_USER = "windowTitleUser";

    public static String FILTER\_LABEL = "filterLabel";

    public static String BUTTON\_HELP = "buttonHelp";

    public static String BUTTON\_INFO = "buttonInfo";

    public static String BUTTON\_ADD = "buttonAdd";

    public static String BUTTON\_UPDATE = "buttonUpdate";

    public static String BUTTON\_REMOVE\_BY\_ID = "buttonRemoveById";

    public static String BUTTON\_REMOVE\_BY\_INDEX = "buttonRemoveByIndex";

    public static String BUTTON\_REMOVE\_LOWER = "buttonRemoveLower";

    public static String BUTTON\_CLEAR = "buttonClear";

    public static String BUTTON\_EXECUTE\_SCRIPT = "buttonExecuteScript";

    public static String BUTTON\_SORT = "buttonSort";

    public static String BUTTON\_FILTER\_STARTS\_WITH\_NAME = "buttonFilterStartsWithName";

    public static String BUTTON\_PRINT\_UNIQUE\_FUEL\_TYPES = "buttonPrintUniqueFuelTypes";

    public static String BUTTON\_PRINT\_FIELD\_DESCENDING\_TYPE = "buttonPrintFieldDescendingType";

    public static String TEXT\_HELP\_HELP = "textHelpHelp";

    public static String TEXT\_HELP\_INFO = "textHelpInfo";

    public static String TEXT\_HELP\_ADD = "textHelpAdd";

    public static String TEXT\_HELP\_UPDATE = "textHelpUpdate";

    public static String TEXT\_HELP\_REMOVE\_BY\_ID = "textHelpRemoveById";

    public static String TEXT\_HELP\_REMOVE\_BY\_INDEX = "textHelpRemoveByIndex";

    public static String TEXT\_HELP\_REMOVE\_LOWER = "textHelpRemoveLower";

    public static String TEXT\_HELP\_CLEAR = "textHelpClear";

    public static String TEXT\_HELP\_EXECUTE\_SCRIPT = "textHelpExecuteScript";

    public static String TEXT\_HELP\_SORT = "textHelpSort";

    public static String TEXT\_HELP\_FILTER\_STARTS\_WITH\_NAME = "textHelpFilterStartsWithName";

    public static String TEXT\_HELP\_PRINT\_UNIQUE\_FUEL\_TYPES = "textHelpPrintUniqueFuelTypes";

    public static String TEXT\_HELP\_PRINT\_FIELD\_DESCENDING\_TYPE = "textHelpPrintFieldDescendingType";

    public static String LABEL\_VEHICLE\_NAME = "labelVehicleName";

    public static String LABEL\_VEHICLE\_X = "labelVehicleX";

    public static String LABEL\_VEHICLE\_Y = "labelVehicleY";

    public static String LABEL\_VEHICLE\_ENGINE\_POWER = "labelVehicleEnginePower";

    public static String LABEL\_VEHICLE\_FUEL\_CONSUMPTION = "labelVehicleFuelConsumption";

    public static String LABEL\_VEHICLE\_TYPE = "labelVehicleType";

    public static String LABEL\_VEHICLE\_FUEL\_TYPE = "labelVehicleFuelType";

    public static String ELEMENT\_TITLE = "elementTitle";

    public static String ADD\_ELEMENT\_TITLE = "addElementTitle";

    public static String UPDATE\_ELEMENT\_TITLE = "updateElementTitle";

    public static String ERROR\_TITLE = "errorTitle";

    public static String AUTH\_TITLE = "authTitle";

    public static String LABEL\_LOGIN = "labelLogin";

    public static String LABEL\_PASSWORD = "labelPassword";

    public static String BUTTON\_ENTER = "buttonEnter";

    public static String TEXT\_ENTER\_LOGIN\_AND\_PASSWORD = "textEnterLoginAndPassword";

    public static String TEXT\_ENTER\_ID\_UPDATE = "textEnterIdUpdate";

    public static String TEXT\_ENTER\_ID\_REMOVE = "textEnterIdRemove";

    public static String TEXT\_ENTER\_INDEX\_REMOVE = "textEnterIndexRemove";

    public static String TEXT\_INVALID\_INDEX = "textInvalidIndex";

    public static String TEXT\_ENTER\_ID\_REMOVE\_LOWER = "textEnterIdRemoveLower";

    public static String TEXT\_ID\_IS\_NOT\_FOUND = "textIdIsNotFound";

    public static String TEXT\_INVALID\_ID = "textInvalidId";

    public static String TEXT\_ENTER\_FILE\_NAME = "textEnterFileName";

    public static String TEXT\_CANT\_CONNECT\_TO\_SERVER = "textCantConnectToServer";

    public static String TEXT\_CANT\_READ\_FROM\_SOCKET = "textCantReadFromSocket";

    public static String TEXT\_ERROR = "textError";

    public static String TEXT\_AUTH\_ERROR = "textAuthError";

    public static String TEXT\_NO\_UNIQUE\_FUEL\_TYPE = "textNoUniqueFuelType";

    public static String TEXT\_REMOVED = "textRemoved";

    private static String[] serverRTexts = {

        "R001", "R002", "R003", "R004", "R005", "R006", "R007", "R008", "R009", "R010", "R011", "R012", "R013", "R014", "R015", "R016", "R017", "R018", "R019"

    };

    private static String[] serverDTexts = {

        "D001", "D002", "D003", "D004", "D005", "D006", "D007", "D008", "D009", "D010", "D011", "D012", "D013"

    };

    /\*

        "R001: Объект успешно добавлен в коллекцию."

        "R002: Не получилось добавить объект в коллекцию."

        "R003: Объект принадлежит другому пользователю"

        "R004: Объект с этим id не был найден"

        "R005: Объект успешно обновлён"

        "R006: Ошибка обновления объекта"

        "R007: Объект принадлежит другому пользователю"

        "R008: Объект с этим id не был найден"

        "R009: Объект успешно удалён"

        "R010: Ошибка удаления объекта"

        "R011: Объект по этому индексу не был найден : " + index

        "R012: Объект принадлежит другому пользователю"

        "R013: Объект успешно удалён из коллекции."

        "R014: Ошибка удаления объекта"

        "R015: Объект(ы) успешно удалены из коллекции"

        "R016: Ошибка удаления объектов"

        "R017: Коллекция отсортирована по параметру EnginePower."

        "R018: Пользователь не зарегистрирован"

        "R019: Неизвестный запрос."

        "D001: Ошибка чтения коллекции vehicles из базы данных : " + e

        "D002: Ошибка чтения коллекции users из базы данных : " + e

        "D003: Ошибка создания объекта Vehicle : " + e

        "D004: Ошибка создания объекта User : " + e

        "D005: Ошибка выполнения команды : " + command + " " + e

        "D006: Не удалось получить следующий элемент последовательности для базы данных vehicles : " + e

        "D007: Не удалось получить следующий элемент последовательности для базы данных users: " + e

        "D008: Не удалось добавить Vehicle в базу данных : " + e

        "D009: Не удалось добавить User в базу данных : " + e

        "D010: Не удалось обновить Vehicle в базу данных : " + e

        "D011: Не удалось удалить Vehicle из базы данных : " + e

        "D012: Не удалось удалить Vehicle из базы данных : " + e

        "D013: Не удалось удалить vehicles данного пользователя из базы данных : " + e

    \*/

    public static void setLanguage(int index) {

        switch (index) {

            case 0:

                Locale.setDefault(new Locale("ru", "RU"));

                break;

            case 1:

                Locale.setDefault(new Locale("pt", "PT"));

                break;

            case 2:

                Locale.setDefault(new Locale("es", "SV"));

                break;

            case 3:

                Locale.setDefault(new Locale("ca", "ES"));

                break;

            case 4:

                Locale.setDefault(new Locale("en", "GB"));

                break;

            default:

                break;

        }

        ResourceBundle.clearCache();

    }

    public static String getString(String key) {

        try {

            return ResourceBundle.getBundle(Resources.name).getString(key);

        } catch ( Exception e) {

            return "";

        }

    }

    public static void init() {

        //saveProperties();

        Locale.setDefault(new Locale("ru", "RU"));

    }

    public static String getServerInfo(String str) {

        //"I001: Тип коллекции: ConcurrentLinkedQueue \n" + "Объекты коллекции: Vehicle \n" + "Количество элементов: " + vehicles.size() + "\n"

        String res = "";

        if (str.contains("I001:")) {

            res = str;

            res = res.replace("I001: ", "");

            res = res.replace("Тип коллекции", Resources.getString("srvI001"));

            res = res.replace("Объекты коллекции", Resources.getString("srvI002"));

            res = res.replace("Количество элементов", Resources.getString("srvI003"));

        }

        return res.isEmpty() ? str : res;

    }

    public static String getServerText(String str) {

        String res = "";

        for (int i = 0; i < serverRTexts.length; i++) {

            if (str.contains(serverRTexts[i] + ":")) {

                try {

                    String text = Resources.getString("srv" + serverRTexts[i]);

                    res += text + "\n";

                } catch (Exception e) {}

            }

        }

        for (int i = 0; i < serverDTexts.length; i++) {

            if (str.contains(serverDTexts[i] + ":")) {

                try {

                    String text = Resources.getString("srv" + serverDTexts[i]);

                    res += text + "\n";

                } catch (Exception e) {}

            }

        }

        return res.isEmpty() ? str : res;

    }

    public static void saveProperties() {

        try (OutputStream output = new FileOutputStream("client\_ru.properties")) {

            Properties prop = new Properties();

            prop.setProperty("windowTitle", "КОЛЛЕКЦИЯ ТРАНСПОРТА");

            prop.setProperty("windowTitleUser", "Пользователь");

            prop.setProperty("filterLabel", "Фильтр");

            prop.setProperty("buttonHelp", "СПРАВКА");

            prop.setProperty("buttonInfo", "ИНФОРМАЦИЯ");

            prop.setProperty("buttonAdd", "ДОБАВИТЬ");

            prop.setProperty("buttonUpdate", "ОБНОВИТЬ");

            prop.setProperty("buttonRemoveById", "УДАЛИТЬ ПО ID");

            prop.setProperty("buttonRemoveByIndex", "УДАЛИТЬ ПО ИНДЕКСУ");

            prop.setProperty("buttonRemoveLower", "УДАЛИТЬ МЕНЬШЕ");

            prop.setProperty("buttonClear", "УДАЛИТЬ ВСЁ");

            prop.setProperty("buttonExecuteScript", "СКРИПТ");

            prop.setProperty("buttonSort", "СОРТИРОВАТЬ");

            prop.setProperty("buttonFilterStartsWithName", "ФИЛЬТР ПО ИМЕНИ");

            prop.setProperty("buttonPrintUniqueFuelTypes", "УНИКАЛЬНЫЕ ТИПЫ ТОПЛИВА");

            prop.setProperty("buttonPrintFieldDescendingType", "ОБРАТНО ПО ТИПУ");

            prop.setProperty("textHelpHelp", "Вывести справку по доступным командам.");

            prop.setProperty("textHelpInfo", "Вывести информацию о коллекции.");

            prop.setProperty("textHelpAdd", "Добавить новый элемент в коллекцию.");

            prop.setProperty("textHelpUpdate", "Обновить значение элемента коллекции, id которого равен заданному.");

            prop.setProperty("textHelpRemoveById", "Удалить элемент из коллекции по его id.");

            prop.setProperty("textHelpRemoveByIndex", "Удалить элемент, находящийся в заданной позиции коллекции.");

            prop.setProperty("textHelpRemoveLower", "Удалить из коллекции все элементы, меньшие, чем заданный.");

            prop.setProperty("textHelpClear", "Очистить коллекцию.");

            prop.setProperty("textHelpExecuteScript", "Выполнить скрипт из указанного файла. В скрипте содержатся команды в таком же виде, в котором их вводит пользователь в интерактивном режиме.");

            prop.setProperty("textHelpSort", "Отсортировать коллекцию в естественном порядке.");

            prop.setProperty("textHelpFilterStartsWithName", "Вывести элементы, значение поля name которых начинается с заданной подстроки.");

            prop.setProperty("textHelpPrintUniqueFuelTypes", "Вывести уникальные значения поля fuelType всех элементов в коллекции.");

            prop.setProperty("textHelpPrintFieldDescendingType", "Вывести значения поля type всех элементов в порядке убывания.");

            prop.setProperty( "labelVehicleName", "Название");

            prop.setProperty( "labelVehicleX", "Координата X");

            prop.setProperty( "labelVehicleY", "Координата Y");

            prop.setProperty( "labelVehicleEnginePower", "Мощь двигателя");

            prop.setProperty( "labelVehicleFuelConsumption", "Потребление топлива");

            prop.setProperty( "labelVehicleType", "Тип");

            prop.setProperty( "labelVehicleFuelType", "Тип топлива");

            prop.setProperty( "elementTitle", "Элемент");

            prop.setProperty( "addElementTitle", "Добавление элемента");

            prop.setProperty( "updateElementTitle", "Обновление элемента");

            prop.setProperty( "errorTitle", "Ошибка");

            prop.setProperty( "authTitle", "Авторизация");

            prop.setProperty( "labelLogin", "Имя" );

            prop.setProperty( "labelPassword", "Пароль" );

            prop.setProperty( "buttonEnter", "Войти" );

            prop.setProperty( "textEnterLoginAndPassword", "Введите имя и пароль !" );

            prop.setProperty( "textEnterIdUpdate", "Введите Id обновляемого объекта" );

            prop.setProperty( "textEnterIdRemove", "Введите Id удаляемого объекта" );

            prop.setProperty( "textEnterIndexRemove", "Введите индекс удаляемого объекта" );

            prop.setProperty( "textInvalidIndex", "Неправильное значение индекса" );

            prop.setProperty( "textEnterIdRemoveLower", "Введите id объекта, для которого программа должна удалить все объекты с меньшим значением enginePower, чем у него." );

            prop.setProperty( "textIdIsNotFound", "Элемент не найден." );

            prop.setProperty( "textInvalidId", "Неправильное значение id" );

            prop.setProperty( "textEnterFileName", "Введите имя файла." );

            prop.setProperty( "textCantConnectToServer", "Не удалось подключиться к серверу." );

            prop.setProperty( "textCantReadFromSocket", "Не получилось прочитать ответ из сокета" );

            prop.setProperty( "textError", "Ошибка" );

            prop.setProperty( "textAuthError", "Не удалось авторизоваться" );

            prop.setProperty( "textNoUniqueFuelType", "В коллекции нет элементов с уникальным типом топлива." );

            prop.setProperty( "textRemoved", "Данные успешно удалены." );

            prop.setProperty( "srvR001", "R001: Объект успешно добавлен в коллекцию." );

            prop.setProperty( "srvR002", "R002: Не получилось добавить объект в коллекцию." );

            prop.setProperty( "srvR003", "R003: Объект принадлежит другому пользователю" );

            prop.setProperty( "srvR004", "R004: Объект с этим id не был найден" );

            prop.setProperty( "srvR005", "R005: Объект успешно обновлён" );

            prop.setProperty( "srvR006", "R006: Ошибка обновления объекта" );

            prop.setProperty( "srvR007", "R007: Объект принадлежит другому пользователю" );

            prop.setProperty( "srvR008", "R008: Объект с этим id не был найден" );

            prop.setProperty( "srvR009", "R009: Объект успешно удалён" );

            prop.setProperty( "srvR010", "R010: Ошибка удаления объекта" );

            prop.setProperty( "srvR011", "R011: Объект по этому индексу не был найден" );

            prop.setProperty( "srvR012", "R012: Объект принадлежит другому пользователю" );

            prop.setProperty( "srvR013", "R013: Объект успешно удалён из коллекции" );

            prop.setProperty( "srvR014", "R014: Ошибка удаления объекта" );

            prop.setProperty( "srvR015", "R015: Объект(ы) успешно удалены из коллекции" );

            prop.setProperty( "srvR016", "R016: Ошибка удаления объектов" );

            prop.setProperty( "srvR017", "R017: Коллекция отсортирована по параметру EnginePower" );

            prop.setProperty( "srvR018", "R018: Пользователь не зарегистрирован" );

            prop.setProperty( "srvR019", "R019: Неизвестный запрос" );

            prop.setProperty( "srvD001", "D001: Ошибка чтения коллекции vehicles из базы данных" );

            prop.setProperty( "srvD002", "D002: Ошибка чтения коллекции users из базы данных" );

            prop.setProperty( "srvD003", "D003: Ошибка создания объекта Vehicle" );

            prop.setProperty( "srvD004", "D004: Ошибка создания объекта User" );

            prop.setProperty( "srvD005", "D005: Ошибка выполнения команды" );

            prop.setProperty( "srvD006", "D006: Не удалось получить следующий элемент последовательности для базы данных vehicles" );

            prop.setProperty( "srvD007", "D007: Не удалось получить следующий элемент последовательности для базы данных users" );

            prop.setProperty( "srvD008", "D008: Не удалось добавить Vehicle в базу данных" );

            prop.setProperty( "srvD009", "D009: Не удалось добавить User в базу данных" );

            prop.setProperty( "srvD010", "D010: Не удалось обновить Vehicle в базу данных" );

            prop.setProperty( "srvD011", "D011: Не удалось удалить Vehicle из базы данных" );

            prop.setProperty( "srvD012", "D012: Не удалось удалить Vehicle из базы данных" );

            prop.setProperty( "srvD013", "D013: Не удалось удалить vehicles данного пользователя из базы данных" );

            prop.setProperty( "srvI001", "Тип коллекции" );

            prop.setProperty( "srvI002", "Объекты коллекции" );

            prop.setProperty( "srvI003", "Количество элементов" );

            prop.store(output, null);

            //System.out.println(prop);

        } catch (Exception e) {

            System.out.println("Save properties error : " + e);

        }

        try (OutputStream output = new FileOutputStream("client\_en.properties")) {

            Properties prop = new Properties();

            prop.setProperty("windowTitle", "VEHICLES COLLECTION");

            prop.setProperty("windowTitleUser", "User");

            prop.setProperty("filterLabel", "Filter");

            prop.setProperty("buttonHelp", "HELP");

            prop.setProperty("buttonInfo", "INFO");

            prop.setProperty("buttonAdd", "ADD");

            prop.setProperty("buttonUpdate", "UPDATE");

            prop.setProperty("buttonRemoveById", "REMOVE BY ID");

            prop.setProperty("buttonRemoveByIndex", "REMOVE AT INDEX");

            prop.setProperty("buttonRemoveLower", "REMOVE LOWER");

            prop.setProperty("buttonClear", "CLEAR");

            prop.setProperty("buttonExecuteScript", "EXECUTE SCRIPT");

            prop.setProperty("buttonSort", "SORT");

            prop.setProperty("buttonFilterStartsWithName", "Filter starts with name");

            prop.setProperty("buttonPrintUniqueFuelTypes", "Print unique fuel types");

            prop.setProperty("buttonPrintFieldDescendingType", "Print field descending type");

            prop.setProperty("textHelpHelp", "Display help for available commands.");

            prop.setProperty("textHelpInfo", "Display information about the collection.");

            prop.setProperty("textHelpAdd", "Add a new element to the collection.");

            prop.setProperty("textHelpUpdate", "Update the value of the collection element whose id is equal to the given one.");

            prop.setProperty("textHelpRemoveById", "Remove an element from the collection by its id.");

            prop.setProperty("textHelpRemoveByIndex", "Remove the element at the given position in the collection.");

            prop.setProperty("textHelpRemoveLower", "Remove all elements from the collection that are less than the given value.");

            prop.setProperty("textHelpClear", "Clear collection.");

            prop.setProperty("textHelpExecuteScript", "Execute the script from the specified file. The script contains commands in the same form in which they are entered by the user in interactive mode.");

            prop.setProperty("textHelpSort", "Sort the collection in natural order.");

            prop.setProperty("textHelpFilterStartsWithName", "Display elements whose name field value begins with the given substring.");

            prop.setProperty("textHelpPrintUniqueFuelTypes", "Print the unique values ​​of the fuelType field of all elements in the collection.");

            prop.setProperty("textHelpPrintFieldDescendingType", "Display the values ​​of the type field of all elements in descending order.");

            prop.setProperty( "labelVehicleName", "Name");

            prop.setProperty( "labelVehicleX", "Coordinate X");

            prop.setProperty( "labelVehicleY", "Coordinate Y");

            prop.setProperty( "labelVehicleEnginePower", "Engine power");

            prop.setProperty( "labelVehicleFuelConsumption", "Fuel comsumption");

            prop.setProperty( "labelVehicleType", "Type");

            prop.setProperty( "labelVehicleFuelType", "Fuel type");

            prop.setProperty( "elementTitle", "Element");

            prop.setProperty( "addElementTitle", "Add element");

            prop.setProperty( "updateElementTitle", "Update element");

            prop.setProperty( "errorTitle", "Error");

            prop.setProperty( "authTitle", "Authorization");

            prop.setProperty( "labelLogin", "Login" );

            prop.setProperty( "labelPassword", "Password" );

            prop.setProperty( "buttonEnter", "Enter" );

            prop.setProperty( "textEnterLoginAndPassword", "Enter login and password !" );

            prop.setProperty( "textEnterIdUpdate","Enter the Id of the object to be updated" );

            prop.setProperty( "textEnterIdRemove", "Enter the Id of the object to be deleted");

            prop.setProperty( "textEnterIndexRemove", "Enter the index of the object to be deleted");

            prop.setProperty( "textInvalidIndex", "Incorrect index value");

            prop.setProperty( "textEnterIdRemoveLower", "Enter the id of the object for which the program should delete all objects with a lower enginePower value than it.");

            prop.setProperty( "textIdIsNotFound", "Element not found.");

            prop.setProperty( "textInvalidId", "Wrong id value");

            prop.setProperty( "textEnterFileName", "Enter file name.");

            prop.setProperty( "textCantConnectToServer", "Failed to connect to server.");

            prop.setProperty( "textCantReadFromSocket", "Failed to read response from socket");

            prop.setProperty( "textError", "Error" );

            prop.setProperty( "textAuthError", "Failed to login" );

            prop.setProperty( "textNoUniqueFuelType", "There are no elements with a unique type of fuel in the collection." );

            prop.setProperty( "textRemoved", "The data was successfully deleted." );

            prop.setProperty( "srvR001", "R001: Object added to collection successfully" );

            prop.setProperty( "srvR002", "R002: Failed to add object to collection" );

            prop.setProperty( "srvR003", "R003: The object belongs to another user" );

            prop.setProperty( "srvR004", "R004: Object with this id was not found" );

            prop.setProperty( "srvR005", "R005: Object updated successfully" );

            prop.setProperty( "srvR006", "R006: Error updating object" );

            prop.setProperty( "srvR007", "R007: Object belongs to another user" );

            prop.setProperty( "srvR008", "R008: Object with this id was not found" );

            prop.setProperty( "srvR009", "R009: Object deleted successfully" );

            prop.setProperty( "srvR010", "R010: Error deleting object" );

            prop.setProperty( "srvR011", "R011: Object at this index was not found" );

            prop.setProperty( "srvR012", "R012: The object belongs to another user" );

            prop.setProperty( "srvR013", "R013: Object was successfully removed from the collection" );

            prop.setProperty( "srvR014", "R014: Error deleting object" );

            prop.setProperty( "srvR015", "R015: Object(s) removed from collection successfully" );

            prop.setProperty( "srvR016", "R016: Error deleting objects" );

            prop.setProperty( "srvR017", "R017: Collection sorted by EnginePower" );

            prop.setProperty( "srvR018", "R018: User not registered" );

            prop.setProperty( "srvR019", "R019: Unknown request" );

            prop.setProperty( "srvD001", "D001: Error reading vehicle collection from database" );

            prop.setProperty( "srvD002", "D002: Error reading users collection from database" );

            prop.setProperty( "srvD003", "D003: Error creating Vehicle object" );

            prop.setProperty( "srvD004", "D004: Error creating User object" );

            prop.setProperty( "srvD005", "D005: Error executing command" );

            prop.setProperty( "srvD006", "D006: Failed to get next sequence element for vehicles database" );

            prop.setProperty( "srvD007", "D007: Failed to get next sequence element for database users" );

            prop.setProperty( "srvD008", "D008: Failed to add Vehicle to database" );

            prop.setProperty( "srvD009", "D009: Failed to add User to database" );

            prop.setProperty( "srvD010", "D010: Failed to update Vehicle to database" );

            prop.setProperty( "srvD011", "D011: Failed to remove Vehicle from database" );

            prop.setProperty( "srvD012", "D012: Failed to remove Vehicle from database" );

            prop.setProperty( "srvD013", "D013: Failed to remove this user's vehicles from the database" );

            prop.setProperty( "srvI001", "Collection type" );

            prop.setProperty( "srvI002", "Collection objects" );

            prop.setProperty( "srvI003", "Amount of elements" );

            prop.store(output, null);

            //System.out.println(prop);

        } catch (Exception e) {

            System.out.println("Save properties error : " + e);

        }

        try (OutputStream output = new FileOutputStream("client\_ca.properties")) {

            Properties prop = new Properties();

            prop.setProperty("windowTitle", "RECOLLIDA DE VEHICLES");

            prop.setProperty("windowTitleUser", "Usuari");

            prop.setProperty("filterLabel", "Filtre");

            prop.setProperty("buttonHelp", "AJUDA");

            prop.setProperty("buttonInfo", "INFORMACIÓ");

            prop.setProperty("buttonAdd", "AFEGIR");

            prop.setProperty("buttonUpdate", "ACTUALITZACIÓ");

            prop.setProperty("buttonRemoveById", "ELIMINAR PER ID");

            prop.setProperty("buttonRemoveByIndex", "ELIMINAR A L'ÍNDEX");

            prop.setProperty("buttonRemoveLower", "TREURE INFERIOR");

            prop.setProperty("buttonClear", "CLAR");

            prop.setProperty("buttonExecuteScript", "EXECUTAR EL GUIÓ");

            prop.setProperty("buttonSort", "ORDENAR");

            prop.setProperty("buttonFilterStartsWithName", "El filtre comença amb el nom");

            prop.setProperty("buttonPrintUniqueFuelTypes", "Imprimeix tipus de combustible únics");

            prop.setProperty("buttonPrintFieldDescendingType", "Camp d'impressió de tipus descendent");

            prop.setProperty("textHelpHelp", "Mostra l'ajuda per a les ordres disponibles.");

            prop.setProperty("textHelpInfo", "Mostra informació sobre la col·lecció.");

            prop.setProperty("textHelpAdd", "Afegiu un element nou a la col·lecció.");

            prop.setProperty("textHelpUpdate", "Actualitzeu el valor de l'element de col·lecció l'identificador del qual és igual al donat.");

            prop.setProperty("textHelpRemoveById", "Elimina un element de la col·lecció pel seu identificador.");

            prop.setProperty("textHelpRemoveByIndex", "Traieu l'element a la posició donada a la col·lecció.");

            prop.setProperty("textHelpRemoveLower", "Elimina tots els elements de la col·lecció que siguin inferiors al valor donat.");

            prop.setProperty("textHelpClear", "Col·lecció clara.");

            prop.setProperty("textHelpExecuteScript", "Executeu l'script des del fitxer especificat. L'script conté ordres en la mateixa forma en què les introdueix l'usuari en mode interactiu.");

            prop.setProperty("textHelpSort", "Ordena la col·lecció per ordre natural.");

            prop.setProperty("textHelpFilterStartsWithName", "Mostra els elements el valor del camp del nom comença amb la subcadena donada.");

            prop.setProperty("textHelpPrintUniqueFuelTypes", "Imprimeix els valors únics del camp fuelType de tots els elements de la col·lecció.");

            prop.setProperty("textHelpPrintFieldDescendingType", "Mostra els valors del camp tipus de tots els elements en ordre descendent.");

            prop.setProperty( "labelVehicleName", "Nom");

            prop.setProperty( "labelVehicleX", "Coordinar X");

            prop.setProperty( "labelVehicleY", "Coordinar Y");

            prop.setProperty( "labelVehicleEnginePower", "Potència del motor");

            prop.setProperty( "labelVehicleFuelConsumption", "Consum de combustible");

            prop.setProperty( "labelVehicleType", "Tipus");

            prop.setProperty( "labelVehicleFuelType", "Tipus de combustible");

            prop.setProperty( "elementTitle", "Element");

            prop.setProperty( "addElementTitle", "Afegeix un element");

            prop.setProperty( "updateElementTitle", "Actualitzar l'element");

            prop.setProperty( "errorTitle", "Error");

            prop.setProperty( "authTitle", "Autorització");

            prop.setProperty( "labelLogin", "iniciar Sessió" );

            prop.setProperty( "labelPassword", "Contrasenya" );

            prop.setProperty( "buttonEnter", "Entra" );

            prop.setProperty( "textEnterLoginAndPassword", "Introdueix el login i la contrasenya!" );

            prop.setProperty( "textEnterIdUpdate","Introduïu l'identificador de l'objecte que voleu actualitzar" );

            prop.setProperty( "textEnterIdRemove", "Introduïu l'identificador de l'objecte que voleu suprimir");

            prop.setProperty( "textEnterIndexRemove", "Introduïu l'índex de l'objecte que voleu suprimir");

            prop.setProperty( "textInvalidIndex", "Valor de l'índex incorrecte");

            prop.setProperty( "textEnterIdRemoveLower", "Introduïu l'identificador de l'objecte per al qual el programa hauria d'eliminar tots els objectes amb un valor enginePower més baix que ell.");

            prop.setProperty( "textIdIsNotFound", "Element no trobat.");

            prop.setProperty( "textInvalidId", "Valor d'identificador incorrecte");

            prop.setProperty( "textEnterFileName", "Introduïu el nom del fitxer.");

            prop.setProperty( "textCantConnectToServer", "Ha fallat la connexió amb el servidor.");

            prop.setProperty( "textCantReadFromSocket", "No s'ha pogut llegir la resposta del sòcol");

            prop.setProperty( "textError", "Error" );

            prop.setProperty( "textAuthError", "No s'ha pogut iniciar la sessió" );

            prop.setProperty( "textNoUniqueFuelType", "No hi ha elements amb un tipus de combustible únic a la col·lecció." );

            prop.setProperty( "textRemoved", "Les dades s'han suprimit correctament." );

            prop.setProperty( "srvR001", "R001: objecte afegit a la col·lecció correctament" );

            prop.setProperty( "srvR002", "R002: no s'ha pogut afegir l'objecte a la col·lecció" );

            prop.setProperty( "srvR003", "R003: l'objecte pertany a un altre usuari" );

            prop.setProperty( "srvR004", "R004: No s'ha trobat l'objecte amb aquest identificador" );

            prop.setProperty( "srvR005", "R005: Objecte actualitzat correctament" );

            prop.setProperty( "srvR006", "R006: error en actualitzar l'objecte" );

            prop.setProperty( "srvR007", "R007: L'objecte pertany a un altre usuari" );

            prop.setProperty( "srvR008", "R008: No s'ha trobat l'objecte amb aquest identificador" );

            prop.setProperty( "srvR009", "R009: Objecte suprimit correctament" );

            prop.setProperty( "srvR010", "R010: error en suprimir l'objecte" );

            prop.setProperty( "srvR011", "R011: No s'ha trobat l'objecte d'aquest índex" );

            prop.setProperty( "srvR012", "R012: l'objecte pertany a un altre usuari" );

            prop.setProperty( "srvR013", "R013: l'objecte s'ha eliminat correctament de la col·lecció" );

            prop.setProperty( "srvR014", "R014: error en suprimir l'objecte" );

            prop.setProperty( "srvR015", "R015: objecte(s) eliminat(s) de la col·lecció correctament" );

            prop.setProperty( "srvR016", "R016: error en suprimir objectes" );

            prop.setProperty( "srvR017", "R017: Col·lecció ordenada per EnginePower" );

            prop.setProperty( "srvR018", "R018: Usuari no registrat" );

            prop.setProperty( "srvR019", "R019: sol·licitud desconeguda" );

            prop.setProperty( "srvD001", "D001: Error en llegir la col·lecció de vehicles de la base de dades" );

            prop.setProperty( "srvD002", "D002: Error en llegir la col·lecció d'usuaris de la base de dades" );

            prop.setProperty( "srvD003", "D003: Error en crear l'objecte Vehicle" );

            prop.setProperty( "srvD004", "D004: Error en crear l'objecte d'usuari" );

            prop.setProperty( "srvD005", "D005: Error en executar l'ordre" );

            prop.setProperty( "srvD006", "D006: no s'ha pogut obtenir el següent element de seqüència per a la base de dades de vehicles" );

            prop.setProperty( "srvD007", "D007: no s'ha pogut obtenir el següent element de seqüència per als usuaris de la base de dades" );

            prop.setProperty( "srvD008", "D008: no s'ha pogut afegir el vehicle a la base de dades" );

            prop.setProperty( "srvD009", "D009: no s'ha pogut afegir l'usuari a la base de dades" );

            prop.setProperty( "srvD010", "D010: no s'ha pogut actualitzar el vehicle a la base de dades" );

            prop.setProperty( "srvD011", "D011: no s'ha pogut eliminar el vehicle de la base de dades" );

            prop.setProperty( "srvD012", "D012: no s'ha pogut eliminar el vehicle de la base de dades" );

            prop.setProperty( "srvD013", "D013: no s'han pogut eliminar els vehicles d'aquest usuari de la base de dades" );

            prop.setProperty( "srvI001", "Tipus de col·lecció" );

            prop.setProperty( "srvI002", "Objectes de col·lecció" );

            prop.setProperty( "srvI003", "Quantitat d'elements" );

            prop.store(output, null);

            //System.out.println(prop);

        } catch (Exception e) {

            System.out.println("Save properties error : " + e);

        }

        try (OutputStream output = new FileOutputStream("client\_es.properties")) {

            Properties prop = new Properties();

            prop.setProperty("windowTitle", "COLECCIÓN DE VEHÍCULOS");

            prop.setProperty("windowTitleUser", "Usuario");

            prop.setProperty("filterLabel", "Filtrar");

            prop.setProperty("buttonHelp", "AYUDA");

            prop.setProperty("buttonInfo", "INFORMACIÓN");

            prop.setProperty("buttonAdd", "AGREGAR");

            prop.setProperty("buttonUpdate", "ACTUALIZAR");

            prop.setProperty("buttonRemoveById", "ELIMINAR POR ID");

            prop.setProperty("buttonRemoveByIndex", "ELIMINAR EN EL ÍNDICE");

            prop.setProperty("buttonRemoveLower", "QUITAR INFERIOR");

            prop.setProperty("buttonClear", "CLARO");

            prop.setProperty("buttonExecuteScript", "EJECUTAR GUIÓN");

            prop.setProperty("buttonSort", "CLASIFICAR");

            prop.setProperty("buttonFilterStartsWithName", "El filtro comienza con el nombre");

            prop.setProperty("buttonPrintUniqueFuelTypes", "Imprimir tipos de combustible únicos");

            prop.setProperty("buttonPrintFieldDescendingType", "Tipo de campo de impresión descendente");

            prop.setProperty("textHelpHelp", "Mostrar ayuda para los comandos disponibles.");

            prop.setProperty("textHelpInfo", "Mostrar información sobre la colección.");

            prop.setProperty("textHelpAdd", "Añadir un nuevo elemento a la colección.");

            prop.setProperty("textHelpUpdate", "Actualiza el valor del elemento de colección cuyo id es igual al dado.");

            prop.setProperty("textHelpRemoveById", "Elimina un elemento de la colección por su id.");

            prop.setProperty("textHelpRemoveByIndex", "Elimina el elemento en la posición dada en la colección.");

            prop.setProperty("textHelpRemoveLower", "Elimina todos los elementos de la colección que sean menores que el valor dado.");

            prop.setProperty("textHelpClear", "Colección clara.");

            prop.setProperty("textHelpExecuteScript", "Ejecute el script desde el archivo especificado. El script contiene comandos en la misma forma en que los ingresa el usuario en modo interactivo.");

            prop.setProperty("textHelpSort", "Ordena la colección en orden natural.");

            prop.setProperty("textHelpFilterStartsWithName", "Mostrar elementos cuyo valor de campo de nombre comience con la subcadena dada.");

            prop.setProperty("textHelpPrintUniqueFuelTypes", "Imprime los valores únicos del campo fuelType de todos los elementos de la colección.");

            prop.setProperty("textHelpPrintFieldDescendingType", "Muestra los valores del campo tipo de todos los elementos en orden descendente.");

            prop.setProperty( "labelVehicleName", "Nombre");

            prop.setProperty( "labelVehicleX", "Coordinar X");

            prop.setProperty( "labelVehicleY", "Coordinar Y");

            prop.setProperty( "labelVehicleEnginePower", "Potencia del motor");

            prop.setProperty( "labelVehicleFuelConsumption", "Consumo de combustible");

            prop.setProperty( "labelVehicleType", "Tipo");

            prop.setProperty( "labelVehicleFuelType", "Tipo de combustible");

            prop.setProperty( "elementTitle", "Elemento");

            prop.setProperty( "addElementTitle", "Agregar elemento");

            prop.setProperty( "updateElementTitle", "Actualizar elemento");

            prop.setProperty( "errorTitle", "Error");

            prop.setProperty( "authTitle", "Autorización");

            prop.setProperty( "labelLogin", "Acceso" );

            prop.setProperty( "labelPassword", "Contraseña" );

            prop.setProperty( "buttonEnter", "Ingresar" );

            prop.setProperty( "textEnterLoginAndPassword", "¡Ingrese el nombre de usuario y la contraseña!" );

            prop.setProperty( "textEnterIdUpdate","Introduzca el Id del objeto a actualizar" );

            prop.setProperty( "textEnterIdRemove", "Introduzca el Id del objeto a eliminar");

            prop.setProperty( "textEnterIndexRemove", "Introduzca el índice del objeto a eliminar");

            prop.setProperty( "textInvalidIndex", "Valor de índice incorrecto");

            prop.setProperty( "textEnterIdRemoveLower", "Ingrese la identificación del objeto para el cual el programa debe eliminar todos los objetos con un valor de potencia de motor más bajo que él.");

            prop.setProperty( "textIdIsNotFound", "Elemento no encontrado.");

            prop.setProperty( "textInvalidId", "Valor de identificación incorrecto");

            prop.setProperty( "textEnterFileName", "Introduzca el nombre del archivo.");

            prop.setProperty( "textCantConnectToServer", "Error al conectar con el servidor.");

            prop.setProperty( "textCantReadFromSocket", "Error al leer la respuesta del socketа");

            prop.setProperty( "textError", "Error" );

            prop.setProperty( "textAuthError", "Error al iniciar sesión" );

            prop.setProperty( "textNoUniqueFuelType", "No hay elementos con un único tipo de combustible en la colección." );

            prop.setProperty( "textRemoved", "Los datos se eliminaron con éxito." );

            prop.setProperty( "srvR001", "R001: Objeto agregado a la colección con éxito" );

            prop.setProperty( "srvR002", "R002: no se pudo agregar el objeto a la colección" );

            prop.setProperty( "srvR003", "R003: El objeto pertenece a otro usuario" );

            prop.setProperty( "srvR004", "R004: No se encontró el objeto con esta identificación" );

            prop.setProperty( "srvR005", "R005: Objeto actualizado con éxito" );

            prop.setProperty( "srvR006", "R006: Error al actualizar objeto" );

            prop.setProperty( "srvR007", "R007: Objeto pertenece a otro usuario" );

            prop.setProperty( "srvR008", "R008: No se encontró el objeto con esta identificación" );

            prop.setProperty( "srvR009", "R009: Objeto eliminado con éxito" );

            prop.setProperty( "srvR010", "R010: Error al eliminar objeto" );

            prop.setProperty( "srvR011", "R011: No se encontró el objeto en este índice" );

            prop.setProperty( "srvR012", "R012: El objeto pertenece a otro usuario" );

            prop.setProperty( "srvR013", "R013: el objeto se eliminó correctamente de la colección" );

            prop.setProperty( "srvR014", "R014: Error al eliminar el objeto" );

            prop.setProperty( "srvR015", "R015: Objeto(s) eliminado(s) de la colección con éxito" );

            prop.setProperty( "srvR016", "R016: Error al eliminar objetos" );

            prop.setProperty( "srvR017", "R017: Colección ordenada por EnginePower" );

            prop.setProperty( "srvR018", "R018: Usuario no registrado" );

            prop.setProperty( "srvR019", "R019: Solicitud desconocida" );

            prop.setProperty( "srvD001", "D001: Error al leer la colección de vehículos de la base de datos" );

            prop.setProperty( "srvD002", "D002: Error al leer la colección de usuarios de la base de datos" );

            prop.setProperty( "srvD003", "D003: Error al crear el objeto Vehículo" );

            prop.setProperty( "srvD004", "D004: Error al crear el objeto Usuario" );

            prop.setProperty( "srvD005", "D005: Error al ejecutar el comando" );

            prop.setProperty( "srvD006", "D006: Error al obtener el siguiente elemento de secuencia para la base de datos de vehículos" );

            prop.setProperty( "srvD007", "D007: no se pudo obtener el siguiente elemento de secuencia para los usuarios de la base de datos" );

            prop.setProperty( "srvD008", "D008: Error al agregar el vehículo a la base de datos" );

            prop.setProperty( "srvD009", "D009: Error al agregar el usuario a la base de datos" );

            prop.setProperty( "srvD010", "D010: Error al actualizar el vehículo a la base de datos" );

            prop.setProperty( "srvD011", "D011: Error al eliminar el vehículo de la base de datos" );

            prop.setProperty( "srvD012", "D012: Error al eliminar el vehículo de la base de datos" );

            prop.setProperty( "srvD013", "D013: Error al eliminar los vehículos de este usuario de la base de datos" );

            prop.setProperty( "srvI001", "Tipo de colección" );

            prop.setProperty( "srvI002", "Objetos de colección" );

            prop.setProperty( "srvI003", "Cantidad de elementos" );

            prop.store(output, null);

            //System.out.println(prop);

        } catch (Exception e) {

            System.out.println("Save properties error : " + e);

        }

        try (OutputStream output = new FileOutputStream("client\_pt.properties")) {

            Properties prop = new Properties();

            prop.setProperty("windowTitle", "RECOLHA DE VEÍCULOS");

            prop.setProperty("windowTitleUser", "Do utilizador");

            prop.setProperty("filterLabel", "Filtro");

            prop.setProperty("buttonHelp", "AJUDA");

            prop.setProperty("buttonInfo", "INFORMAÇÃO");

            prop.setProperty("buttonAdd", "ADICIONAR");

            prop.setProperty("buttonUpdate", "ATUALIZAR");

            prop.setProperty("buttonRemoveById", "REMOVER POR ID");

            prop.setProperty("buttonRemoveByIndex", "REMOVER NO ÍNDICE");

            prop.setProperty("buttonRemoveLower", "REMOVER INFERIOR");

            prop.setProperty("buttonClear", "CLARO");

            prop.setProperty("buttonExecuteScript", "EXECUTAR SCRIPT");

            prop.setProperty("buttonSort", "ORGANIZAR");

            prop.setProperty("buttonFilterStartsWithName", "O filtro começa com o nome");

            prop.setProperty("buttonPrintUniqueFuelTypes", "Imprimir tipos de combustível exclusivos");

            prop.setProperty("buttonPrintFieldDescendingType", "Tipo descendente do campo de impressão");

            prop.setProperty("textHelpHelp", "Exibir ajuda para os comandos disponíveis.");

            prop.setProperty("textHelpInfo", "Exibir informações sobre a coleção.");

            prop.setProperty("textHelpAdd", "Adicione um novo elemento à coleção.");

            prop.setProperty("textHelpUpdate", "Atualize o valor do elemento da coleção cujo id é igual ao dado.");

            prop.setProperty("textHelpRemoveById", "Remova um elemento da coleção por seu id.");

            prop.setProperty("textHelpRemoveByIndex", "Remova o elemento na posição especificada na coleção.");

            prop.setProperty("textHelpRemoveLower", "Remova todos os elementos da coleção que são menores que o valor fornecido.");

            prop.setProperty("textHelpClear", "Coleção clara.");

            prop.setProperty("textHelpExecuteScript", "Execute o script do arquivo especificado. O script contém comandos na mesma forma em que são inseridos pelo usuário no modo interativo.");

            prop.setProperty("textHelpSort", "Classifique a coleção em ordem natural.");

            prop.setProperty("textHelpFilterStartsWithName", "Elementos de exibição cujo valor do campo de nome começa com a substring fornecida.");

            prop.setProperty("textHelpPrintUniqueFuelTypes", "Imprima os valores únicos do campo fuelType de todos os elementos da coleção.");

            prop.setProperty("textHelpPrintFieldDescendingType", "Exiba os valores do campo de tipo de todos os elementos em ordem decrescente.");

            prop.setProperty( "labelVehicleName", "Nome");

            prop.setProperty( "labelVehicleX", "Coordenada X");

            prop.setProperty( "labelVehicleY", "Coordenada Y");

            prop.setProperty( "labelVehicleEnginePower", "Poder do motor");

            prop.setProperty( "labelVehicleFuelConsumption", "Consumo de combustível");

            prop.setProperty( "labelVehicleType", "Tipo");

            prop.setProperty( "labelVehicleFuelType", "Tipo de combustível");

            prop.setProperty( "elementTitle", "Elemento");

            prop.setProperty( "addElementTitle", "Adicionar elemento");

            prop.setProperty( "updateElementTitle", "Atualizar elemento");

            prop.setProperty( "errorTitle", "Erro");

            prop.setProperty( "authTitle", "Autorização");

            prop.setProperty( "labelUser", "Conecte-se" );

            prop.setProperty( "labelPassword", "Senha" );

            prop.setProperty( "buttonEnter", "Digitar" );

            prop.setProperty( "textEnterLoginAndPassword", "Digite login e senha!" );

            prop.setProperty( "textEnterIdUpdate","Digite o Id do objeto a ser atualizado" );

            prop.setProperty( "textEnterIdRemove", "Digite o Id do objeto a ser deletado");

            prop.setProperty( "textEnterIndexRemove", "Insira o índice do objeto a ser excluído");

            prop.setProperty( "textInvalidIndex", "Valor de índice incorreto");

            prop.setProperty( "textEnterIdRemoveLower", "Digite o id do objeto para o qual o programa deve excluir todos os objetos com um valor enginePower menor do que ele.");

            prop.setProperty( "textIdIsNotFound", "Elemento não encontrado.");

            prop.setProperty( "textInvalidId", "Valor de id errado");

            prop.setProperty( "textEnterFileName", "Digite o nome do arquivo.");

            prop.setProperty( "textCantConnectToServer", "Falha ao conectar ao servidor.");

            prop.setProperty( "textCantReadFromSocket", "Falha ao ler a resposta do soquete");

            prop.setProperty( "textError", "Erro" );

            prop.setProperty( "textAuthError", "Falha ao fazer login" );

            prop.setProperty( "textNoUniqueFuelType", "Não há elementos com um único tipo de combustível na coleção." );

            prop.setProperty( "textRemoved", "Os dados foram excluídos com sucesso." );

            prop.setProperty( "srvR001", "R001: Objeto adicionado à coleção com sucesso" );

            prop.setProperty( "srvR002", "R002: Falha ao adicionar objeto à coleção" );

            prop.setProperty( "srvR003", "R003: O objeto pertence a outro usuário" );

            prop.setProperty( "srvR004", "R004: Objeto com este id não foi encontrado" );

            prop.setProperty( "srvR005", "R005: Objeto atualizado com sucesso" );

            prop.setProperty( "srvR006", "R006: Erro ao atualizar o objeto" );

            prop.setProperty( "srvR007", "R007: Objeto pertence a outro usuário" );

            prop.setProperty( "srvR008", "R008: Objeto com este id não foi encontrado" );

            prop.setProperty( "srvR009", "R009: Objeto deletado com sucesso" );

            prop.setProperty( "srvR010", "R010: Erro ao deletar objeto" );

            prop.setProperty( "srvR011", "R011: O objeto neste índice não foi encontrado" );

            prop.setProperty( "srvR012", "R012: O objeto pertence a outro usuário" );

            prop.setProperty( "srvR013", "R013: O objeto foi removido com sucesso da coleção" );

            prop.setProperty( "srvR014", "R014: Erro ao deletar objeto" );

            prop.setProperty( "srvR015", "R015: Objeto(s) removido(s) da coleção com sucesso" );

            prop.setProperty( "srvR016", "R016: Erro ao excluir objetos" );

            prop.setProperty( "srvR017", "R017: Coleção classificada por EnginePower" );

            prop.setProperty( "srvR018", "R018: Usuário não cadastrado" );

            prop.setProperty( "srvR019", "R019: Pedido desconhecido" );

            prop.setProperty( "srvD001", "D001: Erro ao ler a coleta de veículos do banco de dados" );

            prop.setProperty( "srvD002", "D002: Erro ao ler a coleção de usuários do banco de dados" );

            prop.setProperty( "srvD003", "D003: Erro ao criar o objeto Veículo" );

            prop.setProperty( "srvD004", "D004: Erro ao criar o objeto Usuário" );

            prop.setProperty( "srvD005", "D005: Erro ao executar comando" );

            prop.setProperty( "srvD006", "D006: Falha ao obter o próximo elemento de sequência para o banco de dados de veículos" );

            prop.setProperty( "srvD007", "D007: Falha ao obter o próximo elemento de sequência para usuários do banco de dados" );

            prop.setProperty( "srvD008", "D008: Falha ao adicionar veículo ao banco de dados" );

            prop.setProperty( "srvD009", "D009: Falha ao adicionar usuário ao banco de dados" );

            prop.setProperty( "srvD010", "D010: Falha ao atualizar Veículo para banco de dados" );

            prop.setProperty( "srvD011", "D011: Falha ao remover Veículo do banco de dados" );

            prop.setProperty( "srvD012", "D012: Falha ao remover Veículo do banco de dados" );

            prop.setProperty( "srvD013", "D013: Falha ao remover os veículos deste usuário do banco de dados" );

            prop.setProperty( "srvI001", "Tipo de coleção" );

            prop.setProperty( "srvI002", "Objetos de coleção" );

            prop.setProperty( "srvI003", "Quantidade de elementos" );

            prop.store(output, null);

            System.out.println(prop);        } catch (Exception e) {

            System.out.println("Save properties error : " + e);

        }

    }

}

## TableViewPanel.java

package client.graphics;

import javax.swing.JPanel;

import javax.swing.JLabel;

import javax.swing.JScrollPane;

import javax.swing.BorderFactory;

import javax.swing.DefaultCellEditor;

import javax.swing.JTable;

import javax.swing.RowFilter;

import javax.swing.JTextField;

import javax.swing.JComboBox;

import javax.swing.table.TableRowSorter;

import javax.swing.event.DocumentEvent;

import javax.swing.event.DocumentListener;

import java.awt.Color;

import client.CommandHandler;

public class TableViewPanel extends JScrollPane {

private JTable table;

private JLabel filterLabel;

public TableViewPanel(VehiclesTableModel model, CommandHandler handler, JPanel filterPanel, CollectionWindow parent) {

setBackground(Color.lightGray);

setBounds(350,35,1325,410);

setBorder(BorderFactory.createLoweredBevelBorder());

table = new JTable(model);

getViewport().add(table);

JComboBox<String> cbType = new JComboBox<String>(VehicleItem.getTypeStrings());

cbType.setEditable(true);

table.getColumnModel().getColumn(7).setCellEditor(new DefaultCellEditor(cbType));

JComboBox<String> cbFuelType = new JComboBox<String>(VehicleItem.getFuelTypeStrings());

cbFuelType.setEditable(true);

table.getColumnModel().getColumn(8).setCellEditor(new DefaultCellEditor(cbFuelType));

TableRowSorter<VehiclesTableModel> sorter = new TableRowSorter<VehiclesTableModel>(model);

table.setRowSorter(sorter);

filterLabel = new JLabel(Resources.getString(Resources.FILTER\_LABEL) + ":");

JTextField filterField = new JTextField(15);

JComboBox<String> filterColumn = new JComboBox<String>(model.getColumnNames());

filterColumn.setSelectedItem("NAME");

filterField.getDocument().addDocumentListener(new DocumentListener() {

@Override

public void insertUpdate(DocumentEvent e) {

update(e);

}

@Override

public void removeUpdate(DocumentEvent e) {

update(e);

}

@Override

public void changedUpdate(DocumentEvent e) {

update(e);

}

private void update(DocumentEvent e) {

String text = filterField.getText();

if (text.trim().length() == 0) {

sorter.setRowFilter(null);

} else {

sorter.setRowFilter(RowFilter.regexFilter("(?i)" + text, filterColumn.getSelectedIndex()));

}

}

});

filterPanel.add(filterLabel);

filterPanel.add(filterField);

filterPanel.add(filterColumn);

}

public void updateLocale() {

filterLabel.setText(Resources.getString(Resources.FILTER\_LABEL) + ":");

}

}

## VehicleItem.java

package client.graphics;

import data.VehicleType;

import data.FuelType;

import data.VehicleSerializable;

public class VehicleItem {

private Integer id;

private String name;

private Float x;

private Long y;

private String creationDate;

private Integer enginePower;

private Long fuelConsumption;

private String type; //!

private String fuelType; //!

private Integer userId;

public VehicleItem(Integer id, String name, Float x, Long y, String creationDate, Integer enginePower, Long fuelConsumption, String type, String fuelType, Integer userId) {

this.id = id;

this.name = name;

this.x = x;

this.y = y;

this.creationDate = creationDate;

this.enginePower = enginePower;

this.fuelConsumption = fuelConsumption;

this.type = type;

this.fuelType = fuelType;

this.userId = userId;

}

public VehicleItem() {

id = 0;

name = "";

x = (float)0;

y = (long)0;

creationDate = "";

enginePower = 0;

fuelConsumption = (long)0;

type = "";

fuelType = "";

userId = 0;

}

public VehicleSerializable createSerializable() {

return new VehicleSerializable(

name,

x,

y,

enginePower,

fuelConsumption,

(type.isEmpty() ? null : VehicleType.valueOf(type)),

(fuelType.isEmpty() ? null : FuelType.valueOf(fuelType)));

}

public String fill(String str) {

String[] s = str.split(",");

if(s.length != 10) {

System.out.println("Invalid item : " + str);

return null;

}

return fill(s[0], s[1], s[2], s[3], s[4], s[5], s[6], s[7], s[8], s[9]);

}

public String fill (String id, String name, String x, String y, String creationDate, String enginePower, String fuelConsumption, String type, String fuelType, String userId) {

String err = "";

if(id != null) {

try {

this.id = Integer.parseInt(id);

} catch (Exception e) {

err += "Ошибка: Идентификатор объекта должен быть целым числом" + "\n";

}

}

if(name != null) {

if(name.isEmpty()) {

err += "Ошибка: Имя не может быть пустым" + "\n";

} else {

this.name = name;

}

}

try {

this.x = Float.parseFloat(x);

} catch (Exception e) {

err += "Ошибка: Координата X должна быть плавающим числом" + "\n";

}

try {

this.y = Long.parseLong(y);

} catch (Exception e) {

err += "Ошибка: Координата Y должна быть плавающим числом" + "\n";

}

if(creationDate != null) {

if(creationDate.isEmpty()) {

err += "Ошибка: Дата создания не может быть пустой" + "\n";

} else {

this.creationDate = creationDate;

}

}

try {

this.enginePower = Integer.parseInt(enginePower);

} catch (Exception e) {

err += "Ошибка: Мощь двигателя должна быть целым числом больше нуля" + "\n";

}

try {

this.fuelConsumption = Long.parseLong(fuelConsumption);

} catch (Exception e) {

err += "Ошибка: Уровень потребления топлива должен быть числом больше нуля" + "\n";

}

try {

this.type = (type == null || type.isEmpty()) ? "" : VehicleType.valueOf(type).toString();

} catch (Exception e) {

err += "Ошибка: Неверный тип транспортного средства" + "\n";

}

try {

this.fuelType = (fuelType == null || fuelType.isEmpty()) ? "" : FuelType.valueOf(fuelType).toString();

} catch (Exception e) {

err += "Ошибка: Неверный тип топлива" + "\n";

}

if(userId != null) {

try {

this.userId = Integer.parseInt(userId);

} catch (Exception e) {

err += "Ошибка: Идентификатор пользователя должен быть целым числом" + "\n";

}

}

if(err.isEmpty()) {

return null;

//return new VehicleItem(v\_id, v\_name, v\_x, v\_y, v\_creationDate, v\_enginePower, v\_fuelConsumption, v\_type, v\_fuelType, v\_userId);

}

return err;

}

public Integer getId () {

return id;

}

public String getName () {

return name;

}

public String getFuelType () {

return fuelType;

}

public String getType () {

return type;

}

public Integer getEnginePower() {

return enginePower;

}

public Long getFuelConsumption() {

return fuelConsumption;

}

public String getCreationDate() {

return creationDate;

}

public Float getX() {

return x;

}

public Long getY() {

return y;

}

public Integer getUserId() {

return userId;

}

public void setUserId(Integer id) {

this.userId = id;

}

public void setId(Integer id) {

this.id = id;

}

public void setName(String name) {

this.name = name;

}

public void setX(Float x) {

this.x = x;

}

public void setY(Long y) {

this.y = y;

}

public void setEnginePower(Integer enginePower) {

this.enginePower = enginePower;

}

public void setFuelConsumption(Long fuelConsumption) {

this.fuelConsumption = fuelConsumption;

}

public void setType(String type) {

this.type = type;

}

public void setFuelType(String fuelType) {

this.fuelType = fuelType;

}

public void setCreationDate(String creationDate) {

this.creationDate = creationDate;

}

public static String[] getTypeStrings() {

return new String[] {"", "PLANE", "SUBMARINE", "BOAT", "BICYCLE", "CHOPPER"};

}

public static String[] getFuelTypeStrings() {

return new String[] {"", "ALCOHOL", "MANPOWER", "ANTIMATTER"};

}

}

## VehiclesTableModel.java

package client.graphics;

import javax.swing.table.DefaultTableModel;

import java.util.ArrayList;

import java.util.Collections;

import client.CommandHandler;

public class VehiclesTableModel extends DefaultTableModel {

private static String[] columnNames = { "ID", "NAME", "X", "Y", "CREATION DATA", "ENGINE POWER", "FUEL CONSUMPTION", "TYPE", "FUEL TYPE", "USER ID" };

private int userId;

private ObjectViewPanel objectView;

private CommandHandler handler;

public VehiclesTableModel () {

super(columnNames, 0);

userId = 0;

objectView = null;

handler = null;

}

public String[] getColumnNames() {

return columnNames;

}

public void setUserId(int userId) {

this.userId = userId;

}

public int getUserId() {

return userId;

}

public void setObjectView(ObjectViewPanel objectView) {

this.objectView = objectView;

}

public void setCommandHandler(CommandHandler handler) {

this.handler = handler;

}

@Override

public Class<?> getColumnClass(int columnIndex) {

switch (columnIndex) {

case 0: return Integer.class;

case 1: return String.class;

case 2: return Float.class;

case 3: return Long.class;

case 4: return String.class;

case 5: return Integer.class;

case 6: return Long.class;

case 7: return String.class;

case 8: return String.class;

case 9: return Integer.class;

default:

System.out.println("Unknown columnIndex " + columnIndex);

return super.getColumnClass(columnIndex);

}

}

@Override

public boolean isCellEditable(int row, int col) {

return (col != 0) && (col != 4) && (col != 9) && ( (Integer) getValueAt(row, 9) == userId);

}

@Override

public void setValueAt(Object aValue, int rowIndex, int columnIndex) {

super.setValueAt(aValue, rowIndex, columnIndex);

VehicleItem item = getObjByIndex(rowIndex);

new Thread(new Runnable() {

@Override

public void run() {

System.out.println("model: run doUpdate thread");

handler.doUpdate(item.getId(), item.createSerializable());

System.out.println("model: stop doUpdate thread");

}

}).start();

}

public void update(ArrayList<VehicleItem> items) {

// первый массив содержит row index текущей модели

ArrayList<Integer> toBeRemoved = new ArrayList<Integer>();

// второй массив содержит индексы элементов новой модели

ArrayList<Integer> toBeAdded = new ArrayList<Integer>();

// находим элементы для удаления

for(int row = 0; row < getRowCount(); row++) {

boolean found = false;

Integer id = (Integer) this.getValueAt(row, 0);

for(VehicleItem v : items) {

if(id.equals(v.getId())) {

found = true;

break;

}

}

if(!found) {

toBeRemoved.add(row);

}

}

// удаляем элементы toBeRemoved

Collections.reverse(toBeRemoved);

for(int row : toBeRemoved) {

System.out.println("REMOVE ROW " + row);

if (objectView != null) {

objectView.remove(getObjByIndex(row));

}

this.removeRow(row);

}

// находим элементы для обновления и потом добавления

for(int idx = 0; idx < items.size(); idx++) {

boolean found = false;

for(int row = 0; row < getRowCount(); row++) {

Integer id = (Integer) this.getValueAt(row, 0);

VehicleItem itemNew = items.get(idx);

if(id.equals(itemNew.getId())) {

found = true;

VehicleItem itemOld = getObjByIndex(row);

if(!itemNew.getName().equals(itemOld.getName())) {

super.setValueAt(itemNew.getName(), row, 1);

System.out.println("UPDATE ROW " + row + " COLUMN 1");

}

if(!itemNew.getX().equals(itemOld.getX())) {

super.setValueAt(itemNew.getX(), row, 2);

System.out.println("UPDATE ROW " + row + " COLUMN 2");

}

if(!itemNew.getY().equals(itemOld.getY())) {

super.setValueAt(itemNew.getY(), row, 3);

System.out.println("UPDATE ROW " + row + " COLUMN 3");

}

if(!itemNew.getCreationDate().equals(itemOld.getCreationDate())) {

super.setValueAt(itemNew.getCreationDate(), row, 4);

System.out.println("UPDATE ROW " + row + " COLUMN 4");

}

if(!itemNew.getEnginePower().equals(itemOld.getEnginePower())) {

super.setValueAt(itemNew.getEnginePower(), row, 5);

System.out.println("UPDATE ROW " + row + " COLUMN 5");

}

if(!itemNew.getFuelConsumption().equals(itemOld.getFuelConsumption())) {

super.setValueAt(itemNew.getFuelConsumption(), row, 6);

System.out.println("UPDATE ROW " + row + " COLUMN 6");

}

if(!itemNew.getType().equals(itemOld.getType())) {

super.setValueAt(itemNew.getType(), row, 7);

System.out.println("UPDATE ROW " + row + " COLUMN 7");

}

if(!itemNew.getFuelType().equals(itemOld.getFuelType())) {

super.setValueAt(itemNew.getFuelType(), row, 8);

System.out.println("UPDATE ROW " + row + " COLUMN 8");

}

if(!itemNew.getUserId().equals(itemOld.getUserId())) {

super.setValueAt(itemNew.getUserId(), row, 9);

System.out.println("UPDATE ROW " + row + " COLUMN 9");

}

}

}

if(!found) {

toBeAdded.add(idx);

}

}

// добавляем элементы toBeAdded

for(int row : toBeAdded) {

System.out.println("ADD ROW");

VehicleItem item = items.get(row);

if (objectView != null) {

objectView.add(item);

}

addRow( new Object[]{

item.getId(),

item.getName(),

item.getX(),

item.getY(),

item.getCreationDate(),

item.getEnginePower(),

item.getFuelConsumption(),

item.getType(),

item.getFuelType(),

item.getUserId()

});

}

}

public VehicleItem getObjById(int id) {

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

int objId = (Integer) getValueAt(i, 0);

if(objId == id) {

return new VehicleItem(

(Integer) objId,

(String) getValueAt(i, 1),

(Float) getValueAt(i, 2),

(Long) getValueAt(i, 3),

(String) getValueAt(i, 4),

(Integer) getValueAt(i, 5),

(Long) getValueAt(i, 6),

(String) getValueAt(i, 7),

(String) getValueAt(i, 8),

(Integer) getValueAt(i, 9)

);

}

}

return null;

}

public VehicleItem getObjByIndex(int idx) {

if(idx < getRowCount()) {

return new VehicleItem(

(Integer) getValueAt(idx, 0),

(String) getValueAt(idx, 1),

(Float) getValueAt(idx, 2),

(Long) getValueAt(idx, 3),

(String) getValueAt(idx, 4),

(Integer) getValueAt(idx, 5),

(Long) getValueAt(idx, 6),

(String) getValueAt(idx, 7),

(String) getValueAt(idx, 8),

(Integer) getValueAt(idx, 9)

);

}

return null;

}

}

## VehicleWindow.java

package client.graphics;

import javax.swing.JComboBox;

import javax.swing.JLabel;

import javax.swing.JTextField;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.SwingConstants;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.GridLayout;

import java.awt.BorderLayout;

import java.awt.Component;

public class VehicleWindow {

private JPanel panel;

private JTextField tfName;

private JTextField tfX;

private JTextField tfY;

private JTextField tfEnginePower;

private JTextField tfFuelConsumption;

private JComboBox<String> cbType;

private JComboBox<String> cbFuelType;

private JTextField tfType;

private JTextField tfFuelType;

private VehicleItem item;

private boolean isAdd;

private boolean isEditable;

public VehicleWindow() {

this.item = new VehicleItem();

isAdd = true;

isEditable = true;

createPanel();

}

public VehicleWindow(VehicleItem item, boolean isEditable) {

this.item = item;

isAdd = false;

this.isEditable = isEditable;

createPanel();

}

public void createPanel() {

ActionListener tfListener = new ActionListener() {

public void actionPerformed(ActionEvent e) {

Component c = (Component)e.getSource();

c.transferFocus();

}

};

tfName = new JTextField(item.getName().toString(), 15);

tfName.addActionListener(tfListener);

if (!isEditable) tfName.setEditable(false);

tfX = new JTextField(item.getX().toString(), 15);

tfX.addActionListener(tfListener);

if (!isEditable) tfX.setEditable(false);

tfY = new JTextField(item.getY().toString(), 15);

tfY.addActionListener(tfListener);

if (!isEditable) tfY.setEditable(false);

tfEnginePower = new JTextField(item.getEnginePower().toString(), 15);

tfEnginePower.addActionListener(tfListener);

if (!isEditable) tfEnginePower.setEditable(false);

tfFuelConsumption = new JTextField(item.getFuelConsumption().toString(), 15);

tfFuelConsumption.addActionListener(tfListener);

if (!isEditable) tfFuelConsumption.setEditable(false);

if(isEditable) {

cbType = new JComboBox<String>(VehicleItem.getTypeStrings());

cbType.setSelectedItem(item.getType() == null ? "" : item.getType().toString());

cbType.setEditable(true);

cbFuelType = new JComboBox<String>(VehicleItem.getFuelTypeStrings());

cbFuelType.setSelectedItem(item.getFuelType() == null ? "" : item.getFuelType().toString());

cbFuelType.setEditable(true);

} else {

tfType = new JTextField(item.getType().toString(), 15);

if (!isEditable) tfType.setEditable(false);

tfFuelType = new JTextField(item.getFuelType().toString(), 15);

if (!isEditable) tfFuelType.setEditable(false);

}

panel = new JPanel(new BorderLayout(5,5));

JPanel labels = new JPanel(new GridLayout(0,1,2,2));

labels.add(new JLabel(Resources.getString(Resources.LABEL\_VEHICLE\_NAME) + ":", SwingConstants.TRAILING));

labels.add(new JLabel(Resources.getString(Resources.LABEL\_VEHICLE\_X) + ":", SwingConstants.TRAILING));

labels.add(new JLabel(Resources.getString(Resources.LABEL\_VEHICLE\_Y) + ":", SwingConstants.TRAILING));

labels.add(new JLabel(Resources.getString(Resources.LABEL\_VEHICLE\_ENGINE\_POWER) + ":", SwingConstants.TRAILING));

labels.add(new JLabel(Resources.getString(Resources.LABEL\_VEHICLE\_FUEL\_CONSUMPTION) + ":", SwingConstants.TRAILING));

labels.add(new JLabel(Resources.getString(Resources.LABEL\_VEHICLE\_TYPE) + ":", SwingConstants.TRAILING));

labels.add(new JLabel(Resources.getString(Resources.LABEL\_VEHICLE\_FUEL\_TYPE) + ":", SwingConstants.TRAILING));

panel.add(labels, BorderLayout.LINE\_START);

JPanel controls = new JPanel(new GridLayout(0,1,2,2));

controls.add(tfName);

controls.add(tfX);

controls.add(tfY);

controls.add(tfEnginePower);

controls.add(tfFuelConsumption);

controls.add(isEditable ? cbType : tfType);

controls.add(isEditable ? cbFuelType : tfFuelType);

panel.add(controls, BorderLayout.CENTER);

}

public VehicleItem getVehicleItem() {

return item;

}

public boolean run() {

if(!isEditable) {

JOptionPane.showOptionDialog(null, panel, Resources.getString(Resources.ELEMENT\_TITLE),

JOptionPane.DEFAULT\_OPTION, JOptionPane.INFORMATION\_MESSAGE, null, null, null);

return true;

}

while(true) {

int res = JOptionPane.showOptionDialog(null, panel,

isAdd ? Resources.getString(Resources.ADD\_ELEMENT\_TITLE) : Resources.getString(Resources.UPDATE\_ELEMENT\_TITLE),

JOptionPane.DEFAULT\_OPTION, JOptionPane.INFORMATION\_MESSAGE, null, null, null);

if(res == -1) {

return false;

}

VehicleItem item1 = item;

String err = item1.fill(

null,

tfName.getText(),

tfX.getText(),

tfY.getText(),

null,

tfEnginePower.getText(),

tfFuelConsumption.getText(),

cbType.getSelectedItem().toString(),

cbFuelType.getSelectedItem().toString(),

null

);

if(err == null) {

return true;

} else {

JOptionPane.showMessageDialog(null, err, Resources.getString(Resources.ERROR\_TITLE), JOptionPane.ERROR\_MESSAGE);

}

}

}

}

# package client.resources:

…

# package client:

## Client.java

package client;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.net.SocketException;

import java.net.UnknownHostException;

import java.net.SocketTimeoutException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.io.ByteArrayInputStream;

import java.io.ByteArrayOutputStream;

import communication.Request;

import communication.Response;

import communication.ResponseCode;

import client.graphics.Resources;

public class Client {

private DatagramSocket socket;

private int port;

private InetAddress address;

private String username;

private String password;

private int sendBufferSize = 10000;

private int receiveBufferSize = 10000;

private byte[] recvBuffer = new byte[receiveBufferSize];

public Client(int port, String hostname, String username, String password) throws SocketException, UnknownHostException {

this.username = username;

this.password = password;

socket = new DatagramSocket();

socket.setSoTimeout(1000);

this.port = port;

address = InetAddress.getByName(hostname);

}

public String getUsername() {

return username;

}

public String getPassword() {

return password;

}

public Response doCommand(Request request) {

send(request);

return receive();

}

private Response receive() {

try {

DatagramPacket packet = new DatagramPacket(recvBuffer, recvBuffer.length);

socket.receive(packet);

ObjectInputStream stream = new ObjectInputStream(new ByteArrayInputStream(packet.getData(), packet.getOffset(), packet.getLength()));

Response response = (Response) stream.readObject();

stream.close();

return response;

} catch (SocketTimeoutException e) {

System.out.println("Не удалось подключиться к серверу.");

return new Response(ResponseCode.ERROR, Resources.getString(Resources.TEXT\_CANT\_CONNECT\_TO\_SERVER));

} catch (Exception e) {

System.out.println("Не получилось прочитать ответ из сокета : " + e);

return new Response(ResponseCode.ERROR, Resources.getString(Resources.TEXT\_CANT\_READ\_FROM\_SOCKET) + " : " + e);

}

}

private void send(Request request) {

try {

ByteArrayOutputStream byteStream = new ByteArrayOutputStream(sendBufferSize);

ObjectOutputStream stream = new ObjectOutputStream(byteStream);

stream.writeObject(request);

stream.flush();

byte[] sendBuffer = byteStream.toByteArray();

DatagramPacket packet = new DatagramPacket(sendBuffer, sendBuffer.length, address, port);

socket.send(packet);

stream.close();

}

catch (Exception e) {

System.out.println("Не получилось записать запрос в сокет : " + e);

}

}

}

## CommandHandler.java

package client;

import data.VehicleSerializable;

import commands.\*;

import communication.Response;

import communication.ResponseCode;

import communication.Request;

import communication.RequestType;

import javax.swing.JOptionPane;

import client.graphics.VehicleItem;

import client.graphics.VehiclesTableModel;

import java.util.ArrayList;

import client.graphics.Resources;

public class CommandHandler {

private Client client;

private VehiclesTableModel model;

private CommandReader commandReader;

private Thread updater;

public CommandHandler(Client client, VehiclesTableModel model) {

this.model = model;

this.client = client;

this.commandReader = new CommandReader(null, client, client.getUsername(), client.getPassword());

model.setCommandHandler(this);

updater = new Thread(new Runnable() {

@Override

public void run() {

while (!Thread.currentThread().isInterrupted()) {

doShow();

try {

Thread.sleep(5000);

} catch (InterruptedException e) {

//System.out.println("!!! : " + e);

break;

}

}

System.out.println("Updater завершает работу");

}

});

}

public synchronized void doExecuteScript(String fileName) {

commandReader.runCommand("execute\_script " + fileName);

doShow();

}

public synchronized void doHelp() {

JOptionPane.showMessageDialog(null,

Resources.getString(Resources.BUTTON\_HELP) + " : " + Resources.getString(Resources.TEXT\_HELP\_HELP) + "\n" +

Resources.getString(Resources.BUTTON\_INFO) + " : " + Resources.getString(Resources.TEXT\_HELP\_INFO) + "\n" +

Resources.getString(Resources.BUTTON\_ADD) + " : " + Resources.getString(Resources.TEXT\_HELP\_ADD) + "\n" +

Resources.getString(Resources.BUTTON\_UPDATE) + " : " + Resources.getString(Resources.TEXT\_HELP\_UPDATE) + "\n" +

Resources.getString(Resources.BUTTON\_REMOVE\_BY\_ID) + " : " + Resources.getString(Resources.TEXT\_HELP\_REMOVE\_BY\_ID) + "\n" +

Resources.getString(Resources.BUTTON\_REMOVE\_BY\_INDEX) + " : " + Resources.getString(Resources.TEXT\_HELP\_REMOVE\_BY\_INDEX) + "\n" +

Resources.getString(Resources.BUTTON\_REMOVE\_LOWER) + " : " + Resources.getString(Resources.TEXT\_HELP\_REMOVE\_LOWER) + "\n" +

Resources.getString(Resources.BUTTON\_CLEAR) + " : " + Resources.getString(Resources.TEXT\_HELP\_CLEAR) + "\n" +

Resources.getString(Resources.BUTTON\_EXECUTE\_SCRIPT) + " : " + Resources.getString(Resources.TEXT\_HELP\_EXECUTE\_SCRIPT) + "\n" +

Resources.getString(Resources.BUTTON\_SORT) + " : " + Resources.getString(Resources.TEXT\_HELP\_SORT) + "\n" +

Resources.getString(Resources.BUTTON\_FILTER\_STARTS\_WITH\_NAME) + " : " + Resources.getString(Resources.TEXT\_HELP\_FILTER\_STARTS\_WITH\_NAME) + "\n" +

Resources.getString(Resources.BUTTON\_PRINT\_UNIQUE\_FUEL\_TYPES) + " : " + Resources.getString(Resources.TEXT\_HELP\_PRINT\_UNIQUE\_FUEL\_TYPES) + "\n" +

Resources.getString(Resources.BUTTON\_PRINT\_FIELD\_DESCENDING\_TYPE) + " : " + Resources.getString(Resources.TEXT\_HELP\_PRINT\_FIELD\_DESCENDING\_TYPE));

}

public synchronized void doShow() {

Response response = client.doCommand(new Request(RequestType.SHOW, new ShowCommand(), client.getUsername(), client.getPassword()));

if(response != null) {

if (response.getCode() == ResponseCode.ERROR) {

JOptionPane.showMessageDialog(null,

Resources.getString(Resources.TEXT\_ERROR) + " : " + Resources.getServerText(response.getText()),

Resources.getString(Resources.ERROR\_TITLE), JOptionPane.ERROR\_MESSAGE);

} else {

String str = response.getText();

//System.out.println(str);

String[] ss = str.split("\n");

ArrayList<VehicleItem> items = new ArrayList<VehicleItem>();

for(String s : ss) {

if(!s.isEmpty()) {

VehicleItem item = new VehicleItem();

if (item.fill(s) == null) {

items.add(item);

}

}

}

model.update(items);

}

}

}

public synchronized void doInfo() {

Response response = client.doCommand(new Request(RequestType.INFO, new InfoCommand(), client.getUsername(), client.getPassword()));

if(response != null) {

if (response.getCode() == ResponseCode.ERROR) {

JOptionPane.showMessageDialog(null,

Resources.getString(Resources.TEXT\_ERROR) + " : " + Resources.getServerText(response.getText()),

Resources.getString(Resources.ERROR\_TITLE), JOptionPane.ERROR\_MESSAGE);

} else {

JOptionPane.showMessageDialog(null, Resources.getServerInfo(response.getText()));

}

}

}

public synchronized void doPrintUniqueFuelType() {

Response response = client.doCommand(new Request(RequestType.PRINT\_UNIQUE\_FUEL\_TYPE, new PrintUniqueFuelTypeCommand(), client.getUsername(), client.getPassword()));

if(response != null) {

if (response.getCode() == ResponseCode.ERROR) {

JOptionPane.showMessageDialog(null,

Resources.getString(Resources.TEXT\_ERROR) + " : " + Resources.getServerText(response.getText()),

Resources.getString(Resources.ERROR\_TITLE), JOptionPane.ERROR\_MESSAGE);

} else {

if(response.getText().isEmpty()) {

JOptionPane.showMessageDialog(null, Resources.getString(Resources.TEXT\_NO\_UNIQUE\_FUEL\_TYPE));

} else {

JOptionPane.showMessageDialog(null, response.getText());

}

}

}

}

public synchronized void doAdd(VehicleSerializable item) {

Response response = client.doCommand(new Request(RequestType.ADD, new AddCommand(item), client.getUsername(), client.getPassword()));

if(response != null) {

if (response.getCode() == ResponseCode.ERROR) {

JOptionPane.showMessageDialog(null,

Resources.getString(Resources.TEXT\_ERROR) + " : " + Resources.getServerText(response.getText()),

Resources.getString(Resources.ERROR\_TITLE), JOptionPane.ERROR\_MESSAGE);

return;

}

} else {

return;

}

doShow();

}

public synchronized void doUpdate(int id, VehicleSerializable item) {

Response response = client.doCommand(new Request(RequestType.UPDATE, new UpdateCommand(id, item), client.getUsername(), client.getPassword()));

if(response != null) {

if (response.getCode() == ResponseCode.ERROR) {

JOptionPane.showMessageDialog(null,

Resources.getString(Resources.TEXT\_ERROR) + " : " + Resources.getServerText(response.getText()),

Resources.getString(Resources.ERROR\_TITLE), JOptionPane.ERROR\_MESSAGE);

return;

}

} else {

return;

}

doShow();

}

public synchronized void doRemoveItemById(int id) {

Response response = client.doCommand(new Request(RequestType.REMOVE\_BY\_ID, new RemoveByIdCommand(id), client.getUsername(), client.getPassword()));

if(response != null) {

if (response.getCode() == ResponseCode.ERROR) {

JOptionPane.showMessageDialog(null,

Resources.getString(Resources.TEXT\_ERROR) + " : " + Resources.getServerText(response.getText()),

Resources.getString(Resources.ERROR\_TITLE), JOptionPane.ERROR\_MESSAGE);

return;

}

} else {

return;

}

doShow();

}

public synchronized void doRemoveItemByIndex(int idx) {

Response response = client.doCommand(new Request(RequestType.REMOVE\_AT, new RemoveAtIndexCommand(idx), client.getUsername(), client.getPassword()));

if(response != null) {

if (response.getCode() == ResponseCode.ERROR) {

JOptionPane.showMessageDialog(null,

Resources.getString(Resources.TEXT\_ERROR) + " : " + Resources.getServerText(response.getText()),

Resources.getString(Resources.ERROR\_TITLE), JOptionPane.ERROR\_MESSAGE);

return;

}

} else {

return;

}

doShow();

}

public synchronized void doRemoveLowerItems(VehicleSerializable item) {

Response response = client.doCommand(new Request(RequestType.REMOVE\_LOWER, new RemoveLowerCommand(item), client.getUsername(), client.getPassword()));

if(response != null) {

if (response.getCode() == ResponseCode.ERROR) {

JOptionPane.showMessageDialog(null,

Resources.getString(Resources.TEXT\_ERROR) + " : " + Resources.getServerText(response.getText()),

Resources.getString(Resources.ERROR\_TITLE), JOptionPane.ERROR\_MESSAGE);

return;

}

} else {

return;

}

doShow();

}

public synchronized void doClear() {

Response response = client.doCommand(new Request(RequestType.CLEAR, new ClearCommand(), client.getUsername(), client.getPassword()));

if(response != null) {

if (response.getCode() == ResponseCode.ERROR) {

JOptionPane.showMessageDialog(null,

Resources.getString(Resources.TEXT\_ERROR) + " : " + Resources.getServerText(response.getText()),

Resources.getString(Resources.ERROR\_TITLE), JOptionPane.ERROR\_MESSAGE);

return;

} else {

JOptionPane.showMessageDialog(null, Resources.getString(Resources.TEXT\_REMOVED));

}

} else {

return;

}

doShow();

}

public synchronized boolean doAuthorize() {

Response response = client.doCommand(new Request(RequestType.AUTHORIZE, new AuthorizeCommand(), client.getUsername(), client.getPassword()));

if(response != null) {

if (response.getCode() == ResponseCode.ERROR) {

JOptionPane.showMessageDialog(null,

Resources.getString(Resources.TEXT\_ERROR) + " : " + Resources.getServerText(response.getText()),

Resources.getString(Resources.ERROR\_TITLE), JOptionPane.ERROR\_MESSAGE);

return false;

} else if (response.getCode() == ResponseCode.SUCCESS) {

try {

model.setUserId(Integer.parseInt(response.getText()));

return true;

} catch (Exception e) {}

}

}

JOptionPane.showMessageDialog(null, Resources.getString(Resources.TEXT\_AUTH\_ERROR),

Resources.getString(Resources.ERROR\_TITLE), JOptionPane.ERROR\_MESSAGE);

return false;

}

public void doStart() {

updater.start();

}

public void doStop() {

updater.interrupt();

/\*try {

updater.join();

} catch (Exception e) { }\*/

}

}

## CommandReader.java

package client;

import java.util.Scanner;

import data.VehicleSerializable;

import java.util.ArrayList;

import java.io.File;

import java.io.FileInputStream;

import java.io.InputStream;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import commands.\*;

import communication.Response;

import communication.Request;

import communication.RequestType;

public class CommandReader {

private ArrayList<String> scriptnames;

private Scanner scanner;

private Client client;

private String username;

private String password;

public CommandReader(Scanner scanner, Client client, String username, String password) {

scriptnames = new ArrayList<String>();

this.scanner = scanner;

this.client = client;

this.username = username;

this.password = password;

}

private void communicate(Request request) {

if(request != null) {

try {

Response response = client.doCommand(request);

if(response != null) {

System.out.println("Получили ответ (" + response.getCode() + ") : \n" + response.getText());

}

} catch (Exception e) {

System.out.println(e);

}

}

}

public void run()

{

String command = "";

while(true) {

System.out.println("Введите команду управления коллекцией:");

try {

command = scanner.nextLine();

if(command != null) {

command = command.trim();

}

if (runCommand(command)) {

break;

}

} catch (Exception e) {

System.out.println("Неправильная команда");

}

}

}

/\*\* Читает и исполняет команды управления коллекцией из указанного файла

\* @param fileName имя файла

\* @return true, если была выполнена команда exit, false, если нет

\*/

private boolean execute (String fileName) {

boolean needExit = false;

if(scriptnames.contains(fileName)) {

System.out.println("Рекурсивный вызов скриптов запрещён");

return needExit;

}

File f = new File(fileName);

if (f.exists() && !f.isDirectory()) {

try {

scriptnames.add(fileName);

System.out.println("scriptnames : " + scriptnames.toString());

InputStream fin = new FileInputStream(fileName);

BufferedReader reader = new BufferedReader(new InputStreamReader(fin));

String line;

while ((line = reader.readLine()) != null) {

line = line.trim();

if(!line.isEmpty()) {

System.out.println("########## ВЫПОЛНЯЮ КОМАНДУ : " + line);

needExit = runCommand(line);

if(needExit) {

break;

}

}

}

reader.close();

} catch (Exception e) {

System.out.println(e);

}

scriptnames.remove(fileName);

System.out.println("scriptnames : " + scriptnames.toString());

} else {

System.out.println("Неправильное имя файла: " + fileName);

}

return needExit;

}

/\*\* Выводит список команд управления коллекцией \*/

private void printHelp() {

System.out.println("help : вывести справку по доступным командам");

System.out.println("info : вывести в стандартный поток вывода информацию о коллекции (тип, дата инициализации, количество элементов и т.д.)");

System.out.println("show : вывести в стандартный поток вывода все элементы коллекции в строковом представлении");

System.out.println("add {element} : добавить новый элемент в коллекцию");

System.out.println("update id {element} : обновить значение элемента коллекции, id которого равен заданному");

System.out.println("remove\_by\_id id : удалить элемент из коллекции по его id");

System.out.println("clear : очистить коллекцию");

System.out.println("execute\_script file\_name : считать и исполнить скрипт из указанного файла. В скрипте содержатся команды в таком же виде, в котором их вводит пользователь в интерактивном режиме.");

System.out.println("exit : завершить программу");

System.out.println("remove\_at index : удалить элемент, находящийся в заданной позиции коллекции (index)");

System.out.println("remove\_lower {element} : удалить из коллекции все элементы, меньшие, чем заданный");

System.out.println("sort : отсортировать коллекцию в естественном порядке");

System.out.println("filter\_starts\_with\_name name : вывести элементы, значение поля name которых начинается с заданной подстроки");

System.out.println("print\_unique\_fuel\_type : вывести уникальные значения поля fuelType всех элементов в коллекции");

System.out.println("print\_field\_descending\_type : вывести значения поля type всех элементов в порядке убывания");

}

public boolean runCommand (String command)

{

boolean needExit = false;

if(command == null) {

return needExit;

}

if(command.isEmpty()) {

return needExit;

}

String[] args;

args = command.split(" ");

if(args.length == 0) {

return needExit;

}

switch (args[0]) {

case "exit":

needExit = true;

break;

case "help":

printHelp();

break;

case "info":

communicate(new Request(RequestType.INFO, new InfoCommand(), username, password));

break;

case "show":

communicate(new Request(RequestType.SHOW, new ShowCommand(), username, password));

break;

case "add":

if(args.length == 2) {

try {

VehicleSerializable obj = VehicleSerializable.create(args[1]);

if(obj != null) {

communicate(new Request(RequestType.ADD, new AddCommand(obj), username, password));

}

} catch (Exception e) {

System.out.println("Неправильный формат задания объекта : " + args[1]);

}

} else {

System.out.println("Неверное число аргументов");

}

break;

case "update":

if(args.length == 3) {

try {

int id = Integer.parseInt(args[1]);

VehicleSerializable obj = VehicleSerializable.create(args[2]);

if(obj != null) {

communicate(new Request(RequestType.UPDATE, new UpdateCommand(id, obj), username, password));

}

} catch (Exception e) {

System.out.println("Неправильно введены аргументы : " + e);

}

} else {

System.out.println("Неверное число аргументов.");

}

break;

case "clear":

communicate(new Request(RequestType.CLEAR, new ClearCommand(), username, password));

break;

case "execute\_script":

if(args.length == 2) {

needExit = execute(args[1]);

} else {

System.out.println("Неверное число аргументов.");

}

break;

case "remove\_by\_id":

if(args.length == 2) {

try {

int id = Integer.parseInt(args[1]);

communicate(new Request(RequestType.REMOVE\_BY\_ID, new RemoveByIdCommand(id), username, password));

} catch (Exception e) {

System.out.println("Неправильно введены аргументы : " + e);

}

} else {

System.out.println("Неверное число аргументов.");

}

break;

case "remove\_at":

if(args.length == 2) {

try {

int idx = Integer.parseInt(args[1]);

communicate(new Request(RequestType.REMOVE\_AT, new RemoveAtIndexCommand(idx), username, password));

} catch (Exception e) {

System.out.println("Неправильно введены аргументы : " + e);

}

} else {

System.out.println("Неверное число аргументов.");

}

break;

case "remove\_lower":

if(args.length == 2) {

VehicleSerializable obj = VehicleSerializable.create(args[1]);

if(obj != null) {

communicate(new Request(RequestType.REMOVE\_LOWER, new RemoveLowerCommand(obj), username, password));

}

} else {

System.out.println("Неверное число аргументов.");

}

break;

case "sort":

communicate(new Request(RequestType.SORT, new SortCommand(), username, password));

break;

case "filter\_starts\_with\_name":

if(args.length == 2) {

String str = args[1];

communicate(new Request(RequestType.FILTER\_STARTS\_WITH\_NAME, new FilterStartsWithNameCommand(str), username, password));

} else {

System.out.println("Неверное число аргументов.");

}

break;

case "print\_unique\_fuel\_type":

communicate(new Request(RequestType.PRINT\_UNIQUE\_FUEL\_TYPE, new PrintUniqueFuelTypeCommand(), username, password));

break;

case "print\_field\_descending\_type":

communicate(new Request(RequestType.PRINT\_FIELD\_DESCENDING\_TYPE, new PrintFieldDescendingTypeCommand(), username, password));

break;

default:

System.out.println("Неопознанная команда. Наберите 'help' для справки.");

break;

}

return needExit;

}

}

## Main.java

package client;

import javax.swing.JDialog;

import javax.swing.JFrame;

import client.graphics.CollectionWindow;

import client.graphics.LoginWindow;

import client.graphics.VehiclesTableModel;

import client.graphics.Resources;

public class Main {

private static int port = 2000;

public static void main(String[] args) {

JFrame.setDefaultLookAndFeelDecorated(true);

JDialog.setDefaultLookAndFeelDecorated(true);

Resources.init();

LoginWindow loginWindow = new LoginWindow();

loginWindow.run();

String username = loginWindow.getUsername();

String password = loginWindow.getPassword();

VehiclesTableModel model = new VehiclesTableModel();

try {

System.out.println("Запускаю клиент.");

Client client = new Client(port, "localhost", username, password);

CommandHandler handler = new CommandHandler(client, model);

if(!handler.doAuthorize()) {

System.out.println("Ошибка авторизации");

return;

}

CollectionWindow collectionWindow = new CollectionWindow(model, client, handler);

collectionWindow.run();

} catch (Exception e) {

System.out.println("Аварийное завершение работы клиента : " + e);

}

}

}

# package commands:

## AbstractCommand.java

package commands;

import java.io.Serializable;

public class AbstractCommand implements Serializable{

}

## AddCommand.java

package commands;

import data.VehicleSerializable;

public class AddCommand extends AbstractCommand{

private VehicleSerializable obj;

public AddCommand(VehicleSerializable obj) {

this.obj = obj;

}

public VehicleSerializable getObj() {

return obj;

}

}

## AuthorizeCommand.java

package commands;

public class AuthorizeCommand extends AbstractCommand {

}

## ClearCommand.java

package commands;

public class ClearCommand extends AbstractCommand{

}

## ExitCommand.java

package commands;

public class ExitCommand extends AbstractCommand {

}

## FilterStartsWithNameCommand.java

package commands;

public class FilterStartsWithNameCommand extends AbstractCommand{

private String name;

public FilterStartsWithNameCommand(String name) {

this.name = name;

}

public String getName() {

return name;

}

}

## InfoCommand.java

package commands;

public class InfoCommand extends AbstractCommand{

}

## PrintFieldDescendingTypeCommand.java

package commands;

public class PrintFieldDescendingTypeCommand extends AbstractCommand{

}

## PrintUniqueFuelTypeCommand.java

package commands;

public class PrintUniqueFuelTypeCommand extends AbstractCommand{

}

## RegisterUserCommand.java

package commands;

public class RegisterUserCommand extends AbstractCommand {

    private String username;

    private String password;

    public RegisterUserCommand(String username, String password) {

        this.username = username;

        this.password = password;

    }

    public String getUsername() {

        return username;

    }

    public String getPassword() {

        return password;

    }

}

## RemoveAtIndexCommand.java

package commands;

public class RemoveAtIndexCommand extends AbstractCommand{

private Integer index;

public RemoveAtIndexCommand(Integer index) {

this.index = index;

}

public Integer getIndex() {

return index;

}

}

## RemoveByIdCommand.java

package commands;

public class RemoveByIdCommand extends AbstractCommand{

private Integer id;

public RemoveByIdCommand(Integer id) {

this.id = id;

}

public Integer getId() {

return id;

}

}

## RemoveLowerCommand.java

package commands;

import data.VehicleSerializable;

public class RemoveLowerCommand extends AbstractCommand{

private VehicleSerializable obj;

public RemoveLowerCommand(VehicleSerializable obj) {

this.obj = obj;

}

public VehicleSerializable getObj() {

return obj;

}

}

## ShowCommand.java

package commands;

public class ShowCommand extends AbstractCommand{

}

## SortCommand.java

package commands;

public class SortCommand extends AbstractCommand{

}

## UpdateCommand.java

package commands;

import data.VehicleSerializable;

public class UpdateCommand extends AbstractCommand{

private Integer id;

private VehicleSerializable obj;

public UpdateCommand(Integer id, VehicleSerializable obj) {

this.obj = obj;

this.id = id;

}

public VehicleSerializable getObj() {

return obj;

}

public Integer getId() {

return id;

}

}

# package communication:

## Request.java

package communication;

import java.io.Serializable;

import commands.AbstractCommand;

public class Request implements Serializable{

private RequestType type;

private AbstractCommand command;

private String username;

private String password;

public Request(RequestType type, AbstractCommand command, String username, String password) {

this.type = type;

this.command = command;

this.username = username;

this.password = password;

}

public AbstractCommand getCommand() {

return command;

}

public RequestType getType() {

return type;

}

public String getUsername() {

return username;

}

public String getPassword() {

return password;

}

}

## RequestType.java

package communication;

public enum RequestType{

INFO,

SHOW,

ADD,

UPDATE,

REMOVE\_BY\_ID,

CLEAR,

REMOVE\_AT,

REMOVE\_LOWER,

SORT,

FILTER\_STARTS\_WITH\_NAME,

PRINT\_UNIQUE\_FUEL\_TYPE,

PRINT\_FIELD\_DESCENDING\_TYPE,

REGISTER\_USER,

EXIT,  
 AUTHORIZE

}

## Response.java

package communication;

import java.io.Serializable;

public class Response implements Serializable{

private ResponseCode code;

private String text;

public Response(ResponseCode code, String text) {

this.code = code;

this.text = text;

}

public ResponseCode getCode() {

return code;

}

public String getText() {

return text;

}

}

## ResponseCode.java

package communication;

public enum ResponseCode{

SUCCESS,

ERROR

}

# package data:

## Coordinates.java

package data;

import java.io.Serializable;

public class Coordinates implements Serializable {

private Float x;

private long y;

public Coordinates(Float x, long y)

{

this.x = x;

this.y = y;

}

public Float getX() {

return this.x;

}

public long getY() {

return this.y;

}

}

## CoordinatesDeserializer.java

package data;

import com.google.gson.\*;

import java.lang.reflect.Type;

/\*\*

\* Класс CoordinatesDeserializer

\* @author Буреев Фёдор

\* @version 1.0

\*/

public class CoordinatesDeserializer implements JsonDeserializer<Coordinates> {

/\*\* Десериализует объект класса Coordinates

\* @param json Json-элемент, который десериализуют

\* @param typeOfT Тип объекта, в который прооисходит десериализация

\* @param context Контекст десериализации

\* @return Объект класса Coordinates

\*/

@Override

public Coordinates deserialize(JsonElement json, Type typeOfT, JsonDeserializationContext context) throws JsonParseException {

JsonObject jsonObject = json.getAsJsonObject();

Float x = jsonObject.get("x").getAsFloat();

Long y = jsonObject.get("y").getAsLong();

return new Coordinates(x, y);

}

}

## CoordinatesSerializer.java

package data;

import com.google.gson.\*;

import java.lang.reflect.Type;

/\*\*

\* Класс CoordinatesSerializer

\* @author Буреев Фёдор

\* @version 1.0

\*/

public class CoordinatesSerializer implements JsonSerializer<Coordinates> {

/\*\* Сериализует объект класса Coordinates

\* @param obj объект класса Coordinates, который сериализуют

\* @param type Тип объекта, для которого происходит сериализация

\* @param context Контекст сериализации

\* @return JsonElement сериализованного объекта

\*/

@Override

public JsonElement serialize(Coordinates obj, Type type, JsonSerializationContext context) {

JsonObject result = new JsonObject();

result.addProperty("x", obj.getX());

result.addProperty("y", obj.getY());

return result;

}

}

## FuelType.java

package data;

/\*\*

\* Класс FuelType

\* @author Буреев Фёдор

\* @version 1.0

\*/

public enum FuelType {

/\*\* Спирт \*/

ALCOHOL,

/\*\* Рабочая человеческая сила \*/

MANPOWER,

/\*\* Антиматерия \*/

ANTIMATTER;

}

## LocalDateTimeDeserializer.java

package data;

import java.time.LocalDateTime;

import com.google.gson.\*;

import java.lang.reflect.Type;

/\*\*

\* Класс LocalDateTimeDeserializer

\* @author Буреев Фёдор

\* @version 1.0

\*/

public class LocalDateTimeDeserializer implements JsonDeserializer<LocalDateTime> {

/\*\* Десериализует объект класса LocalDateTime

\* @param json Json-элемент, который десериализуют

\* @param typeOfT Тип объекта, в который прооисходит десериализация

\* @param context Контекст десериализации

\* @return Объект класса LocalDateTime

\*/

@Override

public LocalDateTime deserialize(JsonElement json, Type typeOfT, JsonDeserializationContext context) throws JsonParseException {

return LocalDateTime.parse(json.getAsString());

}

}

## LocalDateTimeSerializer.java

package data;

import java.time.LocalDateTime;

import com.google.gson.\*;

import java.lang.reflect.Type;

/\*\*

\* Класс CoordinatesSerializer

\* @author Буреев Фёдор

\* @version 1.0

\*/

public class LocalDateTimeSerializer implements JsonSerializer<LocalDateTime> {

/\*\* Сериализует объект класса LocalDateTime

\* @param obj объект класса LocalDateTime, который сериализуют

\* @param type Тип объекта, для которого происходит сериализация

\* @param context Контекст сериализации

\* @return JsonElement сериализованного объекта

\*/

@Override

public JsonElement serialize(LocalDateTime obj, Type type, JsonSerializationContext context) {

return new JsonPrimitive(obj.toString());

}

}

## SortByEnginePower.java

package data;

import java.util.\*;

/\*\*

\* Класс SortByEnginePower

\* @version 1.0

\*/

public class SortByEnginePower implements Comparator<Vehicle> {

/\*\*

\* Возвращает результат сравнения мощности двигателей двух объектов класса Vehicle

\* @param a первый объект

\* @param b второй объект

\* @return результат сравнения

\*/

public int compare(Vehicle a, Vehicle b){

return a.getEnginePower() - b.getEnginePower();

}

}

## User.java

package data;

public class User {

    private int id;

    private String username;

    private String password;

    public User(int id, String username, String password) {

        this.id = id;

        this.username = username;

        this.password = password;

    }

    public int getId() {

        return id;

    }

    public String getUsername() {

        return username;

    }

    public String getPassword() {

        return password;

    }

}

## Vehicle.java

package data;

import java.time.LocalDateTime;

import com.google.gson.\*;

public class Vehicle implements Comparable<Vehicle> {

    private int id;

    private String name;

    private Coordinates coordinates;

    private LocalDateTime creationDate;

    private int enginePower;

    private long fuelConsumption;

    private VehicleType type;

    private FuelType fuelType;

    private int userId;

    public static Gson getDeserializer() {

        return new GsonBuilder().

            registerTypeAdapter(Vehicle.class, new VehicleDeserializer()).

            registerTypeAdapter(Coordinates.class, new CoordinatesDeserializer()).

            registerTypeAdapter(LocalDateTime.class, new LocalDateTimeDeserializer()).

            create();

    }

    public static Gson getSerializer() {

        return new GsonBuilder().

            registerTypeAdapter(Vehicle.class, new VehicleSerializer()).

            registerTypeAdapter(Coordinates.class, new CoordinatesSerializer()).

            registerTypeAdapter(LocalDateTime.class, new LocalDateTimeSerializer()).

            create();

    }

    /\*\*

     \* Создаёт объект типа Vehicle по строке

     \* @param str Json-строка с параметрами объекта

     \* @return объект класса Vehicle

     \*/

    public static Vehicle create(String str) {

        try {

            Gson gson = Vehicle.getDeserializer();

            Vehicle obj = gson.fromJson(str, Vehicle.class);

            return obj;

        } catch (Exception e) {

            System.out.println("Ошибка чтения объекта : " + e);

            return null;

        }

    }

    /\*\*

     \* Создаёт объект типа Vehicle по указанным параметрам

     \* @param id идентификатор объекта

     \* @param name имя объекта

     \* @param coordinates координаты транспортного средства

     \* @param creationDate время создания объекта

     \* @param enginePower мощь двигателя транспортного средства

     \* @param fuelConsumption потребление топлива транспортного средства

     \* @param type тип транспортного средства

     \* @param fuelType тип топлива, потребляемого транспортным средством

     \*/

    public Vehicle(int id, String name, Coordinates coordinates, LocalDateTime creationDate, int enginePower, long fuelConsumption, VehicleType type, FuelType fuelType, int userId) {

        this.id = id;

        this.name = name;

        this.coordinates = coordinates;

        if(creationDate == null) {

            this.creationDate = LocalDateTime.now();

        } else {

            this.creationDate = creationDate;

        }

        this.enginePower = enginePower;

        this.fuelConsumption = fuelConsumption;

        this.type = type;

        this.fuelType = fuelType;

        this.userId = userId;

    }

    public String print() {

        String result =

            "id: " + id + "\n" +

            "name: " + name + "\n" +

            "coordinates: " + coordinates.getX() + ", " + coordinates.getY() + "\n" +

            "creationDate: " + creationDate + "\n" +

            "enginePower: " + enginePower + "\n" +

            "fuelConsumption: " + fuelConsumption + "\n" +

            "type: " + type + "\n" +

            "fuelType: " + fuelType + "\n";

        return result;

    }

    /\*\*

     \* Возвращает идентификатор объекта

     \* @return идентификатор объекта

     \*/

    public int getId () {

        return id;

    }

    /\*\*

     \* Возвращает имя объекта

     \* @return имя объекта

     \*/

    public String getName () {

        return name;

    }

    /\*\*

     \* Возвращает тип топлива объекта

     \* @return тип топлива

     \*/

    public FuelType getFuelType () {

        return fuelType;

    }

    /\*\*

     \* Возвращает тип транспортного средства

     \* @return тип транспортного средства

     \*/

    public VehicleType getType () {

        return type;

    }

    /\*\*

     \* Возвращает мощность двигателя

     \* @return мощность двигателя

     \*/

    public int getEnginePower() {

        return enginePower;

    }

    /\*\*

     \* Возвращает потребление топлива

     \* @return потребление топлива

     \*/

    public long getFuelConsumption() {

        return fuelConsumption;

    }

    /\*\*

     \* Возвращает время создания объекта

     \* @return время создания объекта

     \*/

    public LocalDateTime getCreationDate() {

        return creationDate;

    }

    /\*\*

     \* Возвращает координаты транспортного средства

     \* @return координаты транспортного средства

     \*/

    public Coordinates getCoordinates() {

        return coordinates;

    }

    public int getUserId() {

        return userId;

    }

    public void setUserId(int id) {

        this.userId = id;

    }

    /\*\*

     \* Устанавливает идентификатор объекта

     \* @param id идентификатор объекта

     \*/

    public void setId(int id) {

        this.id = id;

    }

    /\*\*

     \* Устанавливает имя объекта

     \* @param name имя объекта

     \*/

    public void setName(String name) {

        this.name = name;

    }

    /\*\*

     \* Устанавливает координаты транспортного средства

     \* @param x координата x транспортного средства

     \* @param y координата y транспортного средства

     \*/

    public void setCoordinates(float x, long y) {

        this.coordinates = new Coordinates(x, y);

    }

    /\*\*

     \* Устанавливает мощность двигателя

     \* @param enginePower мощность двигателя

     \*/

    public void setEnginePower(int enginePower) {

        this.enginePower = enginePower;

    }

    /\*\*

     \* Устанавливает потребление топлива

     \* @param fuelConsumption потребление топлива

     \*/

    public void setFuelConsumption(long fuelConsumption) {

        this.fuelConsumption = fuelConsumption;

    }

    /\*\*

     \* Устанавливает тип транспортного средства

     \* @param type тип транспортного средства

     \*/

    public void setType(VehicleType type) {

        this.type = type;

    }

    /\*\*

     \* Устанавливает тип топлива

     \* @param fuelType тип топлива

     \*/

    public void setFuelType(FuelType fuelType) {

        this.fuelType = fuelType;

    }

    /\*\*

     \* Копирует все параметры указанного объекта в этот экземпляр объекта

     \* @param vehicle указанный объект

     \*/

    public void update (Vehicle vehicle) {

        //this.id = vehicle.id; // Оставляем id без изменений.

        this.name = vehicle.name;

        this.coordinates = vehicle.coordinates;

        this.creationDate = vehicle.creationDate;

        this.enginePower = vehicle.enginePower;

        this.fuelConsumption = vehicle.fuelConsumption;

        this.type = vehicle.type;

        this.fuelType = vehicle.fuelType;

    }

    @Override

    public int compareTo(Vehicle vehicle) {

        int id = vehicle.getId();

        if(this.id == id) return 0;

        if (this.id < id) return -1;

        return 1;

    }

}

## VehicleDeserializer.java

package data;

import java.time.LocalDateTime;

import com.google.gson.\*;

import java.lang.reflect.Type;

/\*\*

\* Класс VehicleDeserializer

\* @author Буреев Фёдор

\* @version 1.0

\*/

public class VehicleDeserializer implements JsonDeserializer<Vehicle> {

/\*\* Десериализует объект класса Vehicle

\* @param json Json-элемент, который десериализуют

\* @param typeOfT Тип объекта, в который прооисходит десериализация

\* @param context Контекст десериализации

\* @return Объект класса Vehicle

\*/

@Override

public Vehicle deserialize(JsonElement json, Type typeOfT, JsonDeserializationContext context) throws JsonParseException {

JsonObject jsonObject = json.getAsJsonObject();

JsonElement element;

int id = 0;

element = jsonObject.get("id");

if(element != null) {

id = element.getAsInt();

}

String name = jsonObject.get("name").getAsString();

Coordinates coordinates = context.deserialize(jsonObject.get("coordinates"), Coordinates.class);

LocalDateTime creationDate = null;

element = jsonObject.get("creationDate");

if(element != null) {

creationDate = context.deserialize(element, LocalDateTime.class);

}

int enginePower = jsonObject.get("enginePower").getAsInt();

long fuelConsumption = jsonObject.get("fuelConsumption").getAsLong();

element = jsonObject.get("type");

VehicleType type = null;

if(element != null) {

type = VehicleType.valueOf(element.getAsString());

}

element = jsonObject.get("fuelType");

FuelType fuelType = null;

if(element != null) {

fuelType = FuelType.valueOf(element.getAsString());

}

return new Vehicle(id, name, coordinates, creationDate, enginePower, fuelConsumption, type, fuelType, 0);

}

}

## VehicleSerializable.java

package data;

import com.google.gson.\*;

import java.io.Serializable;

public class VehicleSerializable implements Serializable {

private String name;

private Float x;

private Long y;

private Integer enginePower;

private Long fuelConsumption;

private VehicleType type;

private FuelType fuelType;

public static VehicleSerializable create(String str) {

try {

Gson gson = Vehicle.getDeserializer();

Vehicle obj = gson.fromJson(str, Vehicle.class);

return new VehicleSerializable(obj.getName(), obj.getCoordinates().getX(), obj.getCoordinates().getY(), obj.getEnginePower(), obj.getFuelConsumption(), obj.getType(), obj.getFuelType());

} catch (Exception e) {

System.out.println("Ошибка чтения объекта : " + e);

return null;

}

}

public VehicleSerializable(String name, float x, long y, int enginePower, long fuelConsumption, VehicleType type, FuelType fuelType) {

this.name = name;

this.x = x;

this.y = y;

this.enginePower = enginePower;

this.fuelConsumption = fuelConsumption;

this.type = type;

this.fuelType = fuelType;

}

public Vehicle toVehicle () {

return new Vehicle(0, name, new Coordinates(x, y), null, enginePower, fuelConsumption, type, fuelType, 0);

}

}

## VehicleSerializer.java

package data;

import com.google.gson.\*;

import java.lang.reflect.Type;

/\*\*

\* Класс VehicleSerializer

\* @author Буреев Фёдор

\* @version 1.0

\*/

public class VehicleSerializer implements JsonSerializer<Vehicle> {

/\*\* Сериализует объект класса Vehicle

\* @param obj объект класса Vehicle, который сериализуют

\* @param type Тип объекта, для которого происходит сериализация

\* @param context Контекст сериализации

\* @return JsonElement сериализованного объекта

\*/

@Override

public JsonElement serialize(Vehicle obj, Type type, JsonSerializationContext context) {

JsonObject result = new JsonObject();

result.addProperty("id", obj.getId());

result.addProperty("name", obj.getName());

result.add("coordinates", context.serialize(obj.getCoordinates()));

result.add("creationDate", context.serialize(obj.getCreationDate()));

result.addProperty("enginePower", obj.getEnginePower());

result.addProperty("fuelConsumption", obj.getFuelConsumption());

VehicleType vehicleType = obj.getType();

if(vehicleType != null) {

result.addProperty("type", vehicleType.toString());

}

FuelType fuelType = obj.getFuelType();

if(fuelType != null) {

result.addProperty("fuelType", fuelType.toString());

}

return result;

}

}

## VehicleType.java

package data;

/\*\*

\* Класс VehicleType

\* @author Буреев Фёдор

\* @version 1.0

\*/

public enum VehicleType {

/\*\* Самолёт \*/

PLANE,

/\*\* Подводная лодка \*/

SUBMARINE,

/\*\* Лодка \*/

BOAT,

/\*\* Велосипед \*/

BICYCLE,

/\*\* Вертолёт \*/

CHOPPER;

}

# package server:

## CommandHandler.java

package server;

import communication.Request;

import communication.RequestType;

import communication.Response;

import communication.ResponseCode;

import java.util.concurrent.RecursiveTask;

import java.util.concurrent.ForkJoinPool;

import commands.\*;

public class CommandHandler {

private VehicleCollection vehicles;

private UserManager userManager;

private ForkJoinPool forkJoinPool = ForkJoinPool.commonPool();

private ServerLogger logger;

public CommandHandler(VehicleCollection vehicles, UserManager userManager, ServerLogger logger) {

this.vehicles = vehicles;

this.userManager = userManager;

this.logger = logger;

}

private class CommandHandlerTask extends RecursiveTask<Response> {

private Request request;

public CommandHandlerTask(Request request) {

this.request = request;

}

@Override

protected Response compute() {

return doCommand(request);

}

}

public Response handle(Request request) {

return forkJoinPool.invoke(new CommandHandlerTask(request));

}

private synchronized Response doCommand(Request request) {

RequestType type = request.getType();

logger.info("CommandHandler получил команду : " + type.toString());

int userId = UserManager.UNKNOWN\_USER\_ID;

if(!request.getUsername().isEmpty() || !request.getPassword().isEmpty()) {

userId = userManager.getUserId(request.getUsername(), request.getPassword());

if(userId == UserManager.UNKNOWN\_USER\_ID) {

return new Response(ResponseCode.ERROR, "R018: Пользователь не зарегистрирован");

}

}

switch(type) {

case INFO: {

//InfoCommand cmd = (InfoCommand) request.getCommand();

return new Response(ResponseCode.SUCCESS, vehicles.info());

}

case SHOW: {

//ShowCommand cmd = (ShowCommand) request.getCommand();

return new Response(ResponseCode.SUCCESS, vehicles.show());

}

case ADD: {

AddCommand cmd = (AddCommand) request.getCommand();

Result res = vehicles.add(cmd.getObj().toVehicle(), userId);

return new Response(res.result ? ResponseCode.SUCCESS : ResponseCode.ERROR, res.text);

}

case UPDATE: {

UpdateCommand cmd = (UpdateCommand) request.getCommand();

Result res = vehicles.update(cmd.getId(), cmd.getObj().toVehicle(), userId);

return new Response(res.result ? ResponseCode.SUCCESS : ResponseCode.ERROR, res.text);

}

case REMOVE\_BY\_ID: {

RemoveByIdCommand cmd = (RemoveByIdCommand) request.getCommand();

Result res = vehicles.remove(cmd.getId(), userId);

return new Response(res.result ? ResponseCode.SUCCESS : ResponseCode.ERROR, res.text);

}

case CLEAR: {

//ClearCommand cmd = (ClearCommand) request.getCommand();

vehicles.clear(userId);

return new Response(ResponseCode.SUCCESS, "");

}

case REMOVE\_AT: {

RemoveAtIndexCommand cmd = (RemoveAtIndexCommand) request.getCommand();

Result res = vehicles.remove\_at(cmd.getIndex(), userId);

return new Response(res.result ? ResponseCode.SUCCESS : ResponseCode.ERROR, res.text);

}

case REMOVE\_LOWER: {

RemoveLowerCommand cmd = (RemoveLowerCommand) request.getCommand();

Result res = vehicles.remove\_lower(cmd.getObj().toVehicle(), userId);

return new Response(res.result ? ResponseCode.SUCCESS : ResponseCode.ERROR, res.text);

}

case SORT: {

//SortCommand cmd = (SortCommand) request.getCommand();

vehicles.sort();

return new Response(ResponseCode.SUCCESS, "R017: Коллекция отсортирована по параметру EnginePower.");

}

case FILTER\_STARTS\_WITH\_NAME: {

FilterStartsWithNameCommand cmd = (FilterStartsWithNameCommand) request.getCommand();

return new Response(ResponseCode.SUCCESS, vehicles.filter\_starts\_with\_name(cmd.getName()));

}

case PRINT\_UNIQUE\_FUEL\_TYPE: {

//PrintUniqueFuelTypeCommand cmd = (PrintUniqueFuelTypeCommand) request.getCommand();

return new Response(ResponseCode.SUCCESS, vehicles.print\_unique\_fuel\_type());

}

case PRINT\_FIELD\_DESCENDING\_TYPE: {

//PrintFieldDescendingTypeCommand cmd = (PrintFieldDescendingTypeCommand) request.getCommand();

return new Response(ResponseCode.SUCCESS, vehicles.print\_field\_descending\_type());

}

case REGISTER\_USER: {

RegisterUserCommand cmd = (RegisterUserCommand) request.getCommand();

if(userManager.add(cmd.getUsername(), cmd.getPassword())) {

return new Response(ResponseCode.SUCCESS, "Пользователь зарегистрирован");

} else {

return new Response(ResponseCode.ERROR, "Ошибка регистрации пользователя");

}

}

case EXIT: {

//ExitCommand cmd = (ExitCommand) request.getCommand();

return new Response(ResponseCode.SUCCESS, "Сервер завершает работу");

}

case AUTHORIZE: {

return new Response(ResponseCode.SUCCESS, Integer.toString(userId));

}

}

return new Response(ResponseCode.ERROR, "R019: Неизвестный запрос.");

}

}

## CommunicationManager.java

package server;

import communication.Response;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.TimeUnit;

import java.lang.Thread;

public class CommunicationManager {

private ServerLogger logger;

private CommandHandler handler;

private Receiver receiver;

private Sender sender;

private int delay;

private int task\_num = 0;

private ExecutorService cachedThreadPool = Executors.newCachedThreadPool();

public CommunicationManager(CommandHandler handler, Receiver receiver, Sender sender, ServerLogger logger, int delay) {

this.handler = handler;

this.receiver = receiver;

this.sender = sender;

this.logger = logger;

this.delay = delay;

}

private class CommunicationTask implements Runnable {

public CommunicationTask() {}

@Override

public void run() {

logger.info("CommunicationTask : Начало коммуникации");

RequestInfo info = receiver.receive();

if(info == null) {

logger.warning("CommunicationTask : Выход из коммуникации по ошибке");

return;

}

if((delay > 0) && (task\_num % 2 != 0)) {

logger.info("CommunicationTask : Задержка " + delay + " секунд");

try {

Thread.sleep(delay \* 1000);

} catch (Exception e) {

logger.warning ("CommunicationTask : Ожидание было прервано");

}

}

logger.info("CommunicationTask : Получен запрос : " + info.request.getType());

Response response = handler.handle(info.request);

Thread thread = new Thread(new Runnable() {

@Override

public void run() {

sender.send(response, info.address);

}

});

thread.start();

try {

thread.join();

} catch (Exception e) {

logger.warning("CommunicationTask : прерывание выполнения потока" + e);

}

logger.info("CommunicationTask : Конец коммуникации");

}

}

public void close() {

logger.info("CommunicationManager : Подготовка к завершению работы CommunicationManager");

try {

cachedThreadPool.shutdown();

cachedThreadPool.awaitTermination(10, TimeUnit.SECONDS);

logger.info("CommunicationManager : Работа завершена");

} catch (InterruptedException e) {

logger.warning("CommunicationManager : Ошибка завершения работы CommunicationManager : " + e);

}

}

public void run() {

task\_num++;

cachedThreadPool.submit(new CommunicationTask());

}

}

## DatabaseManager.java

package server;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.PreparedStatement;

import java.sql.Statement;

import java.sql.ResultSet;

import java.util.concurrent.ConcurrentLinkedQueue;

import data.Coordinates;

import data.Vehicle;

import data.VehicleType;

import data.FuelType;

import data.User;

import java.time.LocalDateTime;

public class DatabaseManager {

private Connection connection;

private int port;

private String hostname;

// Tables

private static final String VEHICLES\_TABLE = "vehicles";

private static final String USERS\_TABLE = "users";

private static final String VEHICLE\_SEQUENCE\_FOR\_ID = "vehicle\_sequence";

private static final String USERS\_SEQUENCE\_FOR\_ID = "users\_sequence";

// Table Vehicle fields

private static final String VEHICLE\_ID = "id";

private static final String VEHICLE\_NAME = "name";

private static final String VEHICLE\_COORD\_X = "x";

private static final String VEHICLE\_COORD\_Y = "y";

private static final String VEHICLE\_CREATION\_DATE = "creation\_date";

private static final String VEHICLE\_ENGINE\_POWER = "engine\_power";

private static final String VEHICLE\_FUEL\_CONSUMPTION = "fuel\_consumption";

private static final String VEHICLE\_TYPE = "vehicle\_type";

private static final String VEHICLE\_FUEL\_TYPE = "fuel\_type";

private static final String VEHICLE\_USER\_ID = "user\_id";

// Table users fields

private static final String USER\_ID = "id";

private static final String USER\_NAME = "name";

private static final String USER\_PASSWORD = "password";

// Sequence command

private static final String CREATE\_VEHICLES\_ID\_SEQUENCE = "CREATE SEQUENCE IF NOT EXISTS " + VEHICLE\_SEQUENCE\_FOR\_ID + " START 1 INCREMENT 1";

private static final String CREATE\_USERS\_ID\_SEQUENCE = "CREATE SEQUENCE IF NOT EXISTS " + USERS\_SEQUENCE\_FOR\_ID + " START 1 INCREMENT 1";

// Table commands

private static final String CREATE\_TABLE\_VEHICLES = "CREATE TABLE IF NOT EXISTS " + VEHICLES\_TABLE + " (" +

VEHICLE\_ID + " integer, " +

VEHICLE\_NAME + " text, " +

VEHICLE\_COORD\_X + " real, " +

VEHICLE\_COORD\_Y + " bigint, " +

VEHICLE\_CREATION\_DATE + " text, " +

VEHICLE\_ENGINE\_POWER + " integer, " +

VEHICLE\_FUEL\_CONSUMPTION + " bigint, " +

VEHICLE\_TYPE + " text, " +

VEHICLE\_FUEL\_TYPE + " text, " +

VEHICLE\_USER\_ID + " integer);";

private static final String CREATE\_TABLE\_USERS = "CREATE TABLE IF NOT EXISTS " + USERS\_TABLE + " (" +

USER\_ID + " integer, " +

USER\_NAME + " text, " +

USER\_PASSWORD + " text);";

private static final String SELECT\_TABLE\_VEHICLES = "SELECT \* FROM " + VEHICLES\_TABLE + ";";

private static final String SELECT\_TABLE\_USERS = "SELECT \* FROM " + USERS\_TABLE + ";";

private static final String INSERT\_VEHICLE = "INSERT INTO " +

VEHICLES\_TABLE + " (" +

VEHICLE\_ID + ", " +

VEHICLE\_NAME + ", " +

VEHICLE\_COORD\_X + ", " +

VEHICLE\_COORD\_Y + ", " +

VEHICLE\_CREATION\_DATE + ", " +

VEHICLE\_ENGINE\_POWER + ", " +

VEHICLE\_FUEL\_CONSUMPTION + ", " +

VEHICLE\_TYPE + ", " +

VEHICLE\_FUEL\_TYPE + ", " +

VEHICLE\_USER\_ID + ") VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?);";

private static final String INSERT\_USER = "INSERT INTO " +

USERS\_TABLE + " (" +

USER\_ID + ", " +

USER\_NAME + ", " +

USER\_PASSWORD + ") VALUES (?, ?, ?);";

private static final String UPDATE\_VEHICLE = "UPDATE " +

VEHICLES\_TABLE + " SET " +

VEHICLE\_NAME + " = ?, " +

VEHICLE\_COORD\_X + " = ?, " +

VEHICLE\_COORD\_Y + " = ?, " +

VEHICLE\_CREATION\_DATE + " = ?, " +

VEHICLE\_ENGINE\_POWER + " = ?, " +

VEHICLE\_FUEL\_CONSUMPTION + " = ?, " +

VEHICLE\_TYPE + " = ?, " +

VEHICLE\_FUEL\_TYPE + " = ? WHERE " +

VEHICLE\_ID + " = ?;";

/\*private static final String UPDATE\_USER = "UPDATE " +

USERS\_TABLE + " SET " +

USER\_ID + " = ?," +

USER\_NAME + " = ?," +

USER\_PASSWORD + " ? WHERE " + USER\_ID + " = ?;";

\*/

private static final String DELETE\_VEHICLE = "DELETE FROM " +

VEHICLES\_TABLE + " WHERE " + VEHICLE\_ID + " = ?;";

/\*private static final String DELETE\_USER = "DELETE FROM " +

USERS\_TABLE + " WHERE " + USER\_ID + " = ?;";

\*/

//private static final String CLEAR\_VEHICLES = "TRUNCATE TABLE " + VEHICLES\_TABLE;

private static final String DELETE\_VEHICLES\_WITH\_ID = "DELETE FROM " + VEHICLES\_TABLE + " WHERE " + VEHICLE\_USER\_ID + " = ?;";

private static final String DELETE\_VEHICLE\_IF\_LOWER = "DELETE FROM " +

VEHICLES\_TABLE + " WHERE " + VEHICLE\_ENGINE\_POWER + " < ? AND " + VEHICLE\_USER\_ID + " = ?;";

public DatabaseManager(int port, String hostname) {

this.port = port;

this.hostname = hostname;

}

public void openConnection(String user, String password, String database) throws SQLException, ClassNotFoundException {

Class.forName("org.postgresql.Driver");

connection = DriverManager.getConnection("jdbc:postgresql://" + hostname + ":" + port + "/" + database, user, password);

}

public void closeConnection() {

try {

connection.close();

} catch (Exception e) {

System.out.println("Возникла ошибка во время закрытия соединения с базой данных.");

}

}

public ConcurrentLinkedQueue<Vehicle> getVehicles() throws Exception{

executeCommand(CREATE\_TABLE\_VEHICLES);

executeCommand(CREATE\_VEHICLES\_ID\_SEQUENCE);

try {

ConcurrentLinkedQueue<Vehicle> vehicles = new ConcurrentLinkedQueue<Vehicle>();

Statement statement = connection.createStatement();

ResultSet res = statement.executeQuery(SELECT\_TABLE\_VEHICLES);

while(res.next()) {

Vehicle vehicle = createVehicle(res);

vehicles.add(vehicle);

}

res.close();

statement.close();

return vehicles;

} catch (Exception e) {

throw new Exception("D001: Ошибка чтения коллекции vehicles из базы данных : " + e);

}

}

public ConcurrentLinkedQueue<User> getUsers() throws Exception{

executeCommand(CREATE\_TABLE\_USERS);

executeCommand(CREATE\_USERS\_ID\_SEQUENCE);

try {

ConcurrentLinkedQueue<User> users = new ConcurrentLinkedQueue<User>();

Statement statement = connection.createStatement();

ResultSet res = statement.executeQuery(SELECT\_TABLE\_USERS);

while(res.next()) {

User user = createUser(res);

users.add(user);

}

res.close();

statement.close();

return users;

} catch (Exception e) {

throw new Exception("D002: Ошибка чтения коллекции users из базы данных : " + e);

}

}

private Vehicle createVehicle(ResultSet res) throws Exception{

try {

int id = res.getInt(VEHICLE\_ID);

String name = res.getString(VEHICLE\_NAME);

Float x = res.getFloat(VEHICLE\_COORD\_X);

Long y = res.getLong(VEHICLE\_COORD\_Y);

Coordinates coordinates = new Coordinates(x, y);

LocalDateTime creationDate = LocalDateTime.parse(res.getString(VEHICLE\_CREATION\_DATE));

int enginePower = res.getInt(VEHICLE\_ENGINE\_POWER);

long fuelConsumption = res.getLong(VEHICLE\_FUEL\_CONSUMPTION);

String value;

value = res.getString(VEHICLE\_TYPE);

VehicleType type = (value.isEmpty()) ? null : VehicleType.valueOf(value);

value = res.getString(VEHICLE\_FUEL\_TYPE);

FuelType fuelType = (value.isEmpty()) ? null : FuelType.valueOf(value);

int user\_id = res.getInt(VEHICLE\_USER\_ID);

return new Vehicle(id, name, coordinates, creationDate, enginePower, fuelConsumption, type, fuelType, user\_id);

} catch (Exception e) {

throw new Exception("D003: Ошибка создания объекта Vehicle : " + e);

}

}

private User createUser(ResultSet res) throws Exception{

try {

int id = res.getInt(USER\_ID);

String username = res.getString(USER\_NAME);

String password = res.getString(USER\_PASSWORD);

return new User(id, username, password);

} catch (Exception e) {

throw new Exception("D004: Ошибка создания объекта User : " + e);

}

}

private void executeCommand(String command) throws Exception{

try {

Statement statement = connection.createStatement();

statement.execute(command);

statement.close();

} catch (Exception e) {

throw new Exception("D005: Ошибка выполнения команды : " + command + " " + e);

}

}

public int vehiclesGetNextSequenceNumber() throws Exception{

try{

Statement statement = connection.createStatement();

ResultSet res = statement.executeQuery("SELECT nextval('" + VEHICLE\_SEQUENCE\_FOR\_ID + "');");

res.next();

int value = res.getInt("nextval");

res.close();

statement.close();

return value;

} catch (Exception e) {

throw new Exception("D006: Не удалось получить следующий элемент последовательности для базы данных vehicles : " + e);

}

}

public int usersGetNextSequenceNumber() throws Exception{

try{

Statement statement = connection.createStatement();

ResultSet res = statement.executeQuery("SELECT nextval('" + USERS\_SEQUENCE\_FOR\_ID + "');");

res.next();

int value = res.getInt("nextval");

res.close();

statement.close();

return value;

} catch (Exception e) {

throw new Exception("D007: Не удалось получить следующий элемент последовательности для базы данных users: " + e);

}

}

public void insertVehicle(Vehicle vehicle) throws Exception {

PreparedStatement statement = null;

try {

statement = connection.prepareStatement(INSERT\_VEHICLE);

statement.setInt(1, vehicle.getId());

statement.setString(2, vehicle.getName());

statement.setFloat(3, vehicle.getCoordinates().getX());

statement.setLong(4, vehicle.getCoordinates().getY());

statement.setString(5, vehicle.getCreationDate().toString());

statement.setInt(6, vehicle.getEnginePower());

statement.setLong(7, vehicle.getFuelConsumption());

statement.setString(8, (vehicle.getType() == null) ? "" : vehicle.getType().toString());

statement.setString(9, (vehicle.getFuelType() == null) ? "" : vehicle.getFuelType().toString());

statement.setInt(10, vehicle.getUserId());

//System.out.println(statement);

statement.execute();

} catch (Exception e) {

throw new Exception("D008: Не удалось добавить Vehicle в базу данных : " + e);

}

}

public void insertUser(User user) throws Exception {

PreparedStatement statement = null;

try {

statement = connection.prepareStatement(INSERT\_USER);

statement.setInt(1, user.getId());

statement.setString(2, user.getUsername());

statement.setString(3, user.getPassword());

//System.out.println(statement);

statement.execute();

} catch (Exception e) {

throw new Exception("D009: Не удалось добавить User в базу данных : " + e);

}

}

public void updateVehicle(int id, Vehicle vehicle) throws Exception {

PreparedStatement statement = null;

try {

statement = connection.prepareStatement(UPDATE\_VEHICLE);

statement.setString(1, vehicle.getName());

statement.setFloat(2, vehicle.getCoordinates().getX());

statement.setLong(3, vehicle.getCoordinates().getY());

statement.setString(4, vehicle.getCreationDate().toString());

statement.setInt(5, vehicle.getEnginePower());

statement.setLong(6, vehicle.getFuelConsumption());

statement.setString(7, (vehicle.getType() == null) ? "" : vehicle.getType().toString());

statement.setString(8, (vehicle.getFuelType() == null) ? "" : vehicle.getFuelType().toString());

statement.setInt(9, id);

//System.out.println(statement);

statement.execute();

} catch (Exception e) {

throw new Exception("D010: Не удалось обновить Vehicle в базу данных : " + e);

}

}

public void deleteVehicle(int id) throws Exception {

PreparedStatement statement = null;

try {

statement = connection.prepareStatement(DELETE\_VEHICLE);

statement.setInt(1, id);

statement.execute();

} catch (Exception e) {

throw new Exception("D011: Не удалось удалить Vehicle из базы данных : " + e);

}

}

public void deleteVehiclesIfLower(Vehicle vehicle, int userId) throws Exception {

PreparedStatement statement = null;

try {

statement = connection.prepareStatement(DELETE\_VEHICLE\_IF\_LOWER);

statement.setInt(1, vehicle.getEnginePower());

statement.setInt(2, userId);

statement.execute();

} catch (Exception e) {

throw new Exception("D012: Не удалось удалить Vehicle из базы данных : " + e);

}

}

public void clearVehicles(int userId) throws Exception {

PreparedStatement statement = null;

try {

statement = connection.prepareStatement(DELETE\_VEHICLES\_WITH\_ID);

statement.setInt(1, userId);

statement.execute();

} catch (Exception e) {

throw new Exception("D013: Не удалось удалить vehicles данного пользователя из базы данных : " + e);

}

}

}

## Main.java

package server;

public class Main {

private static int port = 2000;

private static String hostname = "localhost";

private static int pgPort = 5432;

private static String user = "postgres";

private static String password = "1234";

private static String database = "postgres";

private static int delay = 0;

private static final String HELP = "help";

private static final String PG\_HOST = "pg\_host=";

private static final String PG\_PORT = "pg\_port=";

private static final String PG\_USER = "pg\_user=";

private static final String PG\_PASS = "pg\_pass=";

private static final String PG\_DB = "pg\_db=";

private static final String SERVER\_PORT = "srv\_port=";

private static final String DELAY = "delay=";

public static void main(String[] args) {

if((args.length == 1) && (args[0].equals(HELP))) {

System.out.println("Параметры запуска программы : ");

System.out.println(" help вывод справочной информации");

System.out.println(" pg\_host=<hostname> имя хоста, по умолчанию - localhost");

System.out.println(" pg\_port=<port> номер порта, по умолчанию - 5432");

System.out.println(" pg\_user=<user> имя пользователя postgres, по умолчанию - postgres");

System.out.println(" pg\_pass=<password> пароль пользователя postgres, по умолчанию - 1234");

System.out.println(" pg\_db=<databse> имя базы данных postgres, по умолчанию - postgres");

System.out.println(" srv\_port=<port> порт, на котором запущен сервер, по умолчанию - 2000");

System.out.println(" delay=<delay> задержка обработки запроса на указанное кол-во секунд, по умолчанию - 0");

return;

}

try {

for(String str : args) {

if(str.startsWith(PG\_HOST)) hostname = str.substring(PG\_HOST.length());

else if(str.startsWith(PG\_PORT)) pgPort = Integer.parseInt(str.substring(PG\_PORT.length()));

else if(str.startsWith(PG\_USER)) user = str.substring(PG\_USER.length());

else if(str.startsWith(PG\_PASS)) password = str.substring(PG\_PASS.length());

else if(str.startsWith(PG\_DB)) database = str.substring(PG\_DB.length());

else if(str.startsWith(SERVER\_PORT)) port = Integer.parseInt(str.substring(SERVER\_PORT.length()));

else if(str.startsWith(DELAY)) delay = Integer.parseInt(str.substring(DELAY.length()));

else {

System.out.println("Неправильно заданы аргументы командной строки");

return;

}

}

} catch (Exception e) {

System.out.println("Неправильно заданы аргументы командной строки");

return;

}

ServerLogger logger = new ServerLogger();

DatabaseManager databaseManager = new DatabaseManager(pgPort, hostname);

try {

databaseManager.openConnection(user, password, database);

System.out.println("Подключение к базе данных установлено.");

} catch (Exception e) {

System.out.println("Не удалось подключиться к базе данных." + e);

return;

}

UserManager userManager = new UserManager(databaseManager);

if(userManager.loadUsers()) {

System.out.println("Успешно загрузили коллекцию пользователей из базы данных.");

} else {

System.out.println("Ошибка загрузки коллекции пользователей из базы данных.");

databaseManager.closeConnection();

return;

}

VehicleCollection vehicles = new VehicleCollection(databaseManager);

if(vehicles.loadCollection()) {

System.out.println("Успешно загрузили коллекцию vehicles из базы данных.");

} else {

System.out.println("Ошибка загрузки коллекции vehicles из базы данных.");

databaseManager.closeConnection();

return;

}

CommandHandler handler = new CommandHandler(vehicles, userManager, logger);

try {

System.out.println("Запускаю сервер.");

Server server = new Server(port, handler, logger, delay);

server.run();

} catch (Exception e) {

System.out.println("Аварийное завершение работы сервера : " + e);

}

databaseManager.closeConnection();

}

}

## Receiver.java

package server;

import java.net.InetSocketAddress;

import java.net.SocketAddress;

import java.io.IOException;

import java.nio.ByteBuffer;

import java.nio.channels.DatagramChannel;

import java.nio.channels.SelectionKey;

import java.nio.channels.Selector;

import java.io.ObjectInputStream;

import java.io.ByteArrayInputStream;

import communication.Request;

import communication.RequestType;

public class Receiver {

private DatagramChannel receiveChannel;

private Selector readSelector;

private SelectionKey readChannelKey;

private int receiveBufferSize = 10000;

private ByteBuffer buffer;

private Server server;

private ServerLogger logger;

public Receiver(int port, ServerLogger logger, Server server) throws IOException {

buffer = ByteBuffer.allocate(receiveBufferSize);

readSelector = Selector.open();

receiveChannel = DatagramChannel.open();

receiveChannel.bind(new InetSocketAddress("localhost", port));

receiveChannel.configureBlocking(false);

readChannelKey = receiveChannel.register(readSelector, SelectionKey.OP\_READ);

this.logger = logger;

this.server = server;

}

public void close() {

try {

receiveChannel.close();

} catch (Exception e) {

logger.warning("Receiver : Ошибка закрытия канала приёма : " + e);

}

}

public boolean waitForConnection() {

logger.info("Receiver : Жду");

try {

if(readSelector.select() > 0) {

return true;

} else {

logger.warning("Receiver : Выход из селектора");

return false;

}

} catch (Exception e) {

logger.warning("Receiver : Не получилось дождаться запрос из канала : " + e);

}

return false;

}

public synchronized RequestInfo receive() {

try {

if (readSelector.selectedKeys() != null) {

readSelector.selectedKeys().remove(readChannelKey);

}

SocketAddress address = receiveChannel.receive(buffer);

Request request = (Request) (new ObjectInputStream(new ByteArrayInputStream(buffer.array())).readObject());

buffer.clear();

if(request.getType() == RequestType.EXIT) {

server.setNeedExit();

logger.info("Receiver : Получен запрос на завершение работы сервера");

}

server.notifyServer();

return new RequestInfo(address, request);

} catch (Exception e) {

logger.warning("Receiver : Не получилось прочитать запрос из канала : " + e);

}

server.notifyServer();

return null;

}

}

## RequestInfo.java

package server;

import communication.Request;

import java.net.SocketAddress;

public class RequestInfo {

SocketAddress address;

Request request;

public RequestInfo(SocketAddress address, Request request) {

this.address = address;

this.request = request;

}

}

## Result.java

package server;

public class Result {

public boolean result;

public String text;

public Result(boolean result, String text) {

this.result = result;

this.text = text;

}

}

## Sender.java

package server;

import java.net.SocketAddress;

import java.io.ObjectOutputStream;

import java.io.ByteArrayOutputStream;

import java.io.IOException;

import communication.Response;

import java.nio.ByteBuffer;

import java.nio.channels.DatagramChannel;

import java.nio.channels.SelectionKey;

import java.nio.channels.Selector;

public class Sender {

private DatagramChannel sendChannel;

private Selector writeSelector;

private SelectionKey writeChannelKey;

private int sendBufferSize = 10000;

private ServerLogger logger;

public Sender (ServerLogger logger) throws IOException {

writeSelector = Selector.open();

sendChannel = DatagramChannel.open();

sendChannel.configureBlocking(false);

writeChannelKey = sendChannel.register(writeSelector, SelectionKey.OP\_WRITE);

this.logger = logger;

}

public void close() {

try {

sendChannel.close();

} catch (Exception e) {

logger.warning("Sender : Ошибка закрытия канала передачи : " + e);

}

}

public synchronized void send(Response response, SocketAddress address) {

logger.info("Sender : Посылаю ответ.");

try {

if (writeSelector.select() > 0) {

ByteArrayOutputStream byteStream = new ByteArrayOutputStream(sendBufferSize);

ObjectOutputStream stream = new ObjectOutputStream(byteStream);

stream.writeObject(response);

stream.flush();

if (writeSelector.selectedKeys() != null) {

writeSelector.selectedKeys().remove(writeChannelKey);

}

sendChannel.send(ByteBuffer.wrap(byteStream.toByteArray()), address);

logger.info("Sender : Отправлен ответ");

stream.close();

return;

}

logger.warning("Sender : Выход из селектора для записи.");

}

catch (Exception e) {

logger.warning("Sender : Не получилось записать ответ в сокет : " + e);

}

}

}

## Server.java

package server;

import java.io.IOException;

public class Server {

private ServerLogger logger;

private Receiver receiver;

private Sender sender;

private CommunicationManager communicationManager;

private volatile boolean needExit = false;

public Server(int port, CommandHandler handler, ServerLogger logger, int delay) throws IOException {

receiver = new Receiver(port, logger, this);

sender = new Sender(logger);

communicationManager = new CommunicationManager(handler, receiver, sender, logger, delay);

this.logger = logger;

}

public void setNeedExit() {

needExit = true;

}

public synchronized void notifyServer() {

this.notify();

}

public synchronized void run() {

logger.info("Server : Начало работы сервера");

while (!needExit) {

if(receiver.waitForConnection()) {

communicationManager.run();

try {

this.wait();

} catch (Exception e) {

logger.warning("Server : Ожидание коммуникации завершилось ошибкой : " + e);

}

}

}

logger.info("Server : Подготовка к завершению работы сервера");

communicationManager.close();

receiver.close();

sender.close();

logger.info("Server : Работа сервера завершена.");

}

}

## ServerLogger.java

package server;

import java.util.logging.FileHandler;

import java.util.logging.Logger;

import java.util.logging.SimpleFormatter;

import java.nio.charset.StandardCharsets;

public class ServerLogger {

Logger logger;

public ServerLogger () {

logger = Logger.getLogger("server");

try {

logger.setUseParentHandlers(false); // Отключает дефолтное логирование в консоль

FileHandler fh = new FileHandler("log.txt", true); //true обеспечивает добавление к файлу, а не затирание файла

logger.addHandler(fh);

SimpleFormatter formatter = new SimpleFormatter();

fh.setFormatter(formatter);

fh.setEncoding(StandardCharsets.UTF\_8.name());

} catch (Exception e) {

System.out.println("Не удалось настроить запись лога в файл.");

}

}

public void info (String str) {

logger.info(Thread.currentThread().getName() + " : " + str);

System.out.println(Thread.currentThread().getName() + " : " + str);

}

public void warning (String str) {

logger.warning(Thread.currentThread().getName() + " : " + str);

System.out.println(Thread.currentThread().getName() + " : " + str);

}

}

## UserManager.java

package server;

import java.util.Optional;

import java.util.concurrent.ConcurrentLinkedQueue;

import data.User;

import java.security.MessageDigest;

public class UserManager {

private ConcurrentLinkedQueue<User> users;

private DatabaseManager databaseManager;

public static int UNKNOWN\_USER\_ID = 0;

public UserManager (DatabaseManager databaseManager) {

users = new ConcurrentLinkedQueue<>();

this.databaseManager = databaseManager;

}

private String hashPassword(String password) {

try {

MessageDigest sha1 = MessageDigest.getInstance("SHA-1");

byte[] bytes = sha1.digest(password.getBytes());

String result = "";

for (byte b : bytes) {

result += String.format("%02x", b);

}

//System.out.println(password + " -> " + result);

return result;

} catch(Exception e) {

System.out.println("Ошибка хэширования пароля : " + e);

}

return password;

}

public int getUserId(String username, String password) {

String hash = hashPassword(password);

Optional<User> result = users.stream()

.filter(user -> (user.getUsername().equals(username) && user.getPassword().equals(hash)))

.findFirst();

if(result.isPresent()) {

return result.get().getId();

} else {

return UNKNOWN\_USER\_ID;

}

}

public boolean add (String username, String password) {

try {

int userId = getUserId(username, password);

if(userId == UNKNOWN\_USER\_ID) {

String hash = hashPassword(password);

userId = databaseManager.usersGetNextSequenceNumber();

User user = new User(userId, username, hash);

databaseManager.insertUser(user);

users.add(user);

} else {

System.out.println("Пользователь " + username + " уже существует");

}

return true;

} catch (Exception e) {

System.out.println("Ошибка добавления пользователя : " + e);

return false;

}

}

public boolean loadUsers() {

try {

users = databaseManager.getUsers();

} catch (Exception e) {

System.out.println(e);

return false;

}

return true;

}

}

## VehicleCollection.java

package server;

import java.util.LinkedList;

import java.util.stream.Collectors;

import java.util.Collections;

import java.util.Arrays;

import java.util.Comparator;

import java.util.Optional;

import java.util.concurrent.ConcurrentLinkedQueue;

import data.FuelType;

import data.SortByEnginePower;

import data.Vehicle;

public class VehicleCollection {

private ConcurrentLinkedQueue<Vehicle> vehicles;

private DatabaseManager databaseManager;

public VehicleCollection (DatabaseManager databaseManager) {

vehicles = new ConcurrentLinkedQueue<>();

this.databaseManager = databaseManager;

}

public String info () {

return "I001: Тип коллекции: ConcurrentLinkedQueue \n" + "Объекты коллекции: Vehicle \n" + "Количество элементов: " + vehicles.size() + "\n";

}

public String show () {

return vehicles.stream()

.map(vehicle -> vehicle.toString())

.collect(Collectors.joining("\n"));

}

public Result add (Vehicle element, int userId) {

try {

int value = databaseManager.vehiclesGetNextSequenceNumber();

element.setId(value);

element.setUserId(userId);

databaseManager.insertVehicle(element);

vehicles.add(element);

return new Result(true, "R001: Объект успешно добавлен в коллекцию.");

} catch (Exception e) {

return new Result(false, "R002: Не получилось добавить объект в коллекцию.");

}

}

public Result update (int id, Vehicle element, int userId) {

Optional<Vehicle> result = vehicles.stream()

.filter(vehicle -> (vehicle.getId() == id))

.findFirst();

if(result.isPresent()) {

if(result.get().getUserId() != userId) {

return new Result(false, "R003: Объект принадлежит другому пользователю");

}

} else {

return new Result(false, "R004: Объект с этим id не был найден");

}

try {

databaseManager.updateVehicle(id, element);

result.get().update(element);

return new Result(true, "R005: Объект успешно обновлён");

} catch (Exception E) {

return new Result(false, "R006: Ошибка обновления объекта");

}

}

public void clear (int userId) {

try {

databaseManager.clearVehicles(userId);

vehicles.removeIf( vehicle -> (vehicle.getUserId() == userId));

} catch (Exception E) {

System.out.println("Не получилось очистить таблицу Vehicles в базе данных");

}

}

public Result remove (int id, int userId) {

Optional<Vehicle> result = vehicles.stream()

.filter(vehicle -> (vehicle.getId() == id))

.findFirst();

if(result.isPresent()) {

if(result.get().getUserId() != userId) {

return new Result(false, "R007: Объект принадлежит другому пользователю");

}

} else {

return new Result(true, "R008: Объект с этим id не был найден");

}

try {

databaseManager.deleteVehicle(id);

vehicles.removeIf( vehicle -> (vehicle.getId() == id) );

return new Result(true, "R009: Объект успешно удалён");

} catch (Exception E) {

return new Result(false, "R010: Ошибка удаления объекта");

}

}

public Result remove\_at (int index, int userId) {

int count = 0;

Vehicle vehicle = null;

for(Vehicle v : vehicles) {

if(count != index) {

count++;

} else {

vehicle = v;

break;

}

}

if(vehicle == null) return new Result(false, "R011: Объект по этому индексу не был найден : " + index);

if(vehicle.getUserId() != userId) {

return new Result(false, "R012: Объект принадлежит другому пользователю");

}

try {

databaseManager.deleteVehicle(vehicle.getId());

vehicles.remove(vehicle);

return new Result(true, "R013: Объект успешно удалён из коллекции.");

} catch (Exception E) {

return new Result(false, "R014: Ошибка удаления объекта");

}

}

public Result remove\_lower (Vehicle element, int userId) {

try {

databaseManager.deleteVehiclesIfLower(element, userId);

vehicles.removeIf( vehicle -> ((vehicle.getEnginePower() < element.getEnginePower()) && (vehicle.getUserId() == userId)));

return new Result(true, "R015: Объект(ы) успешно удалены из коллекции");

} catch (Exception E) {

return new Result(false, "R016: Ошибка удаления объектов");

}

}

public void sort () {

LinkedList<Vehicle> vehicleList = new LinkedList<Vehicle>(vehicles);

Collections.sort(vehicleList, new SortByEnginePower());

vehicles = new ConcurrentLinkedQueue<Vehicle>(vehicleList);

}

public String filter\_starts\_with\_name (String name) {

return vehicles.stream()

.filter(vehicle -> vehicle.getName().startsWith(name))

.map(vehicle -> vehicle.print())

.collect(Collectors.joining("\n"));

}

public String print\_unique\_fuel\_type () {

return Arrays.stream(FuelType.values())

.filter( val -> vehicles.stream().filter(vehicle -> vehicle.getFuelType() == val).count() == 1 )

.map( val -> val.toString())

.collect(Collectors.joining("\n"));

}

public String print\_field\_descending\_type () {

return vehicles.stream()

.map(vehicle -> vehicle.getType())

.filter(val -> val != null)

.sorted(Comparator.reverseOrder())

.map( val -> val.toString())

.collect(Collectors.joining("\n"));

}

public boolean loadCollection() {

try {

vehicles = databaseManager.getVehicles();

} catch (Exception e) {

System.out.println(e);

return false;

}

return true;

}

}

**4. Выводы по работе**

Я разобрался с устройством графических библиотек javax.swing и java.awt, разобрался, как расставлять по окнам компоненты этих библиотек, каких они бывают типов и как с ними работать. Также я научился работать с локалью и файлами свойств для локализации на несколько языков.

Приложение сервер значительно изменено. Теперь, при запуске этого приложения, показывается окно авторизации, в которое можно ввести логин и паролю. В случае корректного ввода появляется следующее окно - окно коллекции, разделённое на три части: панель кнопок (слева) со всеми стандартными операциями из предыдущих лабораторных, таблица (посередине сверху) , выполненная компонентом JTable и содержащая текущую информацию из базы данных (и обновляемая каждую секунду), и графическая область, в которой на координатной сетке изображен размер объектов в виде шариков и их положение. Объекты от разных пользователей нарисованы разными цветами.

Строки таблицы можно фильтровать/сортировать по любой из колонок.

В графической части при создании/удалении объекта этот объект постепенно появляется/исчезает в течение одной секунды. Анимация выполнена при помощи изменения значения размера объекта.

Чуть правее от JTextField, в который можно ввести строку, по которой мы будем фильтровать таблицу, находится JComboBox, в котором можно выбрать один из четырёх языков (каталанский, испанский, португальский, русский). При выборе другого языка все компоненты приложения соответственно обновляются.