Московский государственный технический университет им. Н.Э. Баумана

Утверждаю:

Галкин В.А

"__"_____2014 г.

Курсовая работа
по курсу «Сетевые технологии»:
«Программа передачи коротких сообщений»

Описание программы
(вид документа)

писчая бумага
(вид носителя)

64
(количество листов)

ИСПОЛНИТЕЛИ:

студенты групп
ИУ5-63 и ИУ5-64
Леонтьев А.В. _____
Корнуков Н.О. _____
Латкин И.И. _____

"__"_____2014 г.

Москва - 2014

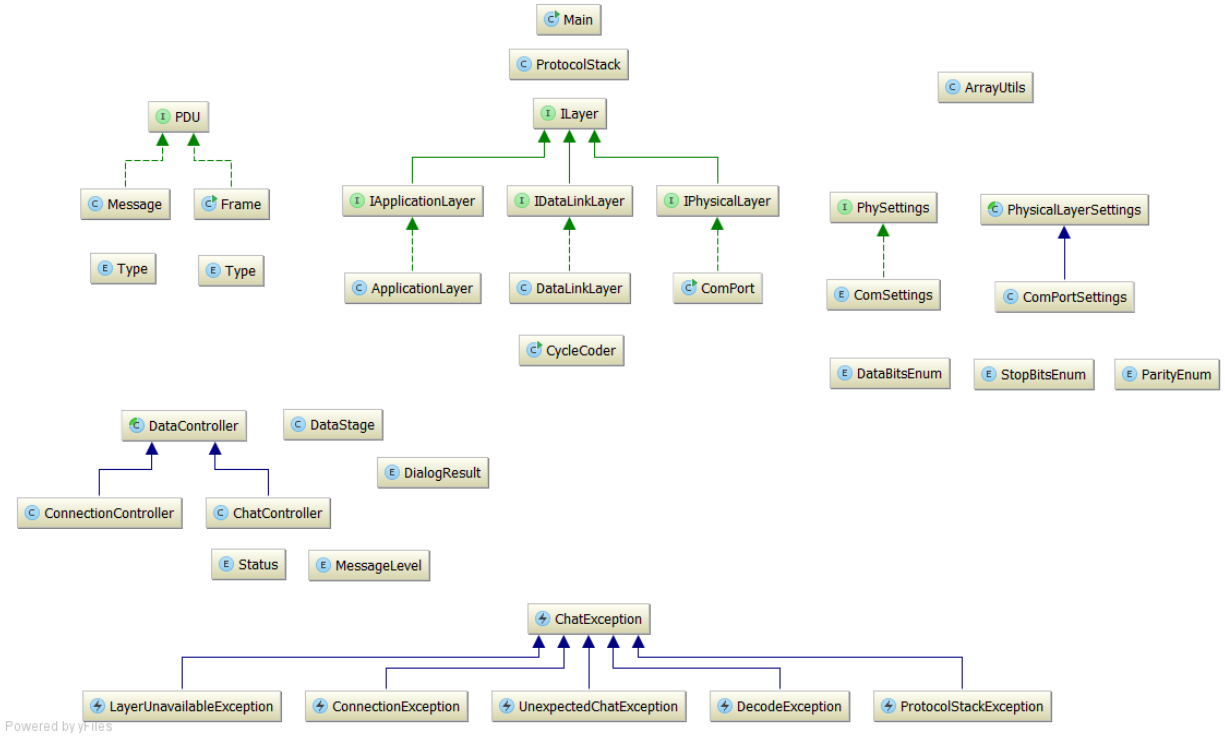
Оглавление

1. Введение

Программный продукт разработан с использованием технологии JavaFX (графический интерфейс пользователя) и RXTX (для работы физического уровня) на языке программирования Java.

Разработка осуществлялась в среде программирования IntelliJ IDEA при помощи распределённой системы контроля версия (git).

2. Схема классов



3. Пакет gui

3.1. Пакет templates

3.1.1. Шаблон chat.fxml

3.1.1.1. Листинг

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.paint.Color?>
<?import javafx.scene.shape.Circle?>
<?import javafx.scene.web.WebView?>

<VBox fx:id="layout" layoutX="0.0" layoutY="0.0" prefHeight="344.0" prefWidth="483.0"
xmlns:fx="http://javafx.com/fxml/1" xmlns="http://javafx.com/javafx/2.2"
fx:controller="gui.ChatController">
```

```
 <children>
  <MenuBar>
   <menus>
    <Menu mnemonicParsing="false" text="Connection">
     <items>
      <MenuItem mnemonicParsing="false" onAction="#onMenuConnect" text="Connect" />
      <MenuItem mnemonicParsing="false" onAction="#onMenuDisconnect"
text="Disconnect" />
     </items>
    </Menu>
    <Menu mnemonicParsing="false" text="Help">
     <items>
      <MenuItem mnemonicParsing="false" onAction="#onMenuAbout" text="About" />
     </items>
    </Menu>
   </menus>
  </MenuBar>
  <SplitPane dividerPositions="0.5036764705882353" focusTraversable="true"
orientation="VERTICAL" prefHeight="200.0" prefWidth="160.0" VBox.vgrow="ALWAYS">
   <items>
    <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="100.0" prefWidth="160.0">
     <children>
      <WebView fx:id="webView" maxHeight="1.7976931348623157E308"
maxWidth="1.7976931348623157E308" prefHeight="133.0" prefWidth="481.0"
AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0"
AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0" />
     </children>
    </AnchorPane>
    <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="100.0" prefWidth="160.0">
     <children>
      <TextArea fx:id="inputField" maxHeight="1.7976931348623157E308"
maxWidth="1.7976931348623157E308" onKeyReleased="#onKeyReleased"
prefHeight="133.0" prefWidth="481.0" wrapText="true" AnchorPane.bottomAnchor="0.0"
AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0" />
     </children>
    </AnchorPane>
   </items>
  </SplitPane>
  <HBox prefHeight="40.0" prefWidth="318.0">
   <children>
    <ToolBar maxHeight="1.7976931348623157E308"
maxWidth="1.7976931348623157E308" prefHeight="40.0" prefWidth="351.0"
HBox.hgrow="ALWAYS">
     <items>
      <Circle fx:id="statusIcon" fill="RED" radius="6.00006103515625" stroke="BLACK"
strokeType="INSIDE" />
      <Label fx:id="statusText" alignment="CENTER_LEFT" text="Not connected"
textAlignment="LEFT">
       <textFill>
        <Color blue="0.388" green="0.388" red="0.388" />
       </textFill>
      </Label>
```

```
        </items>
       </ToolBar>
        <ToolBar maxHeight="1.7976931348623157E308">
          <Circle fx:id="ctsIcon" fill="RED" radius="6.00006103515625" stroke="BLACK"
strokeType="INSIDE" />
          <Label fx:id="ctsText" alignment="CENTER_LEFT" text="CTS"
textAlignment="LEFT">
            <textFill>
              <Color blue="0.388" green="0.388" red="0.388" />
            </textFill>
          </Label>
        </ToolBar>
       <ToolBar maxHeight="1.7976931348623157E308">
        <items>
          <Button fx:id="sendButton" defaultButton="true" disable="true"
mnemonicParsing="false" onAction="#sendClick" text="Send" />
        </items>
       </ToolBar>
     </children>
   </HBox>
  </children>
</VBox>
```

3.1.2. Шаблон connection.fxml

3.1.2.1. Листинг

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.collections.FXCollections?>
<?import javafx.geometry.Insets?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<GridPane fx:id="layout" hgap="10.0" maxHeight="1.7976931348623157E308"
prefHeight="205.0" prefWidth="400.0" vgap="10.0" xmlns:fx="http://javafx.com/fxml/1"
xmlns="http://javafx.com/javafx/2.2" fx:controller="gui.ConnectionController">
  <children>
    <Label text="Nickname:" GridPane.columnIndex="0" GridPane.rowIndex="0"
GridPane.vgrow="SOMETIMES" />
    <TextField fx:id="userName" prefWidth="269.0" text="User" GridPane.columnIndex="1"
GridPane.columnSpan="2147483647" GridPane.rowIndex="0"
GridPane.vgrow="SOMETIMES" />
    <Label text="COM port:" GridPane.columnIndex="0" GridPane.rowIndex="1" />
    <ComboBox fx:id="comPort" maxWidth="1.7976931348623157E308" prefWidth="180.0"
GridPane.columnIndex="1" GridPane.columnSpan="2" GridPane.rowIndex="1">
      <items>
        <FXCollections fx:factory="observableArrayList" />
      </items>
    </ComboBox>
    <Label text="Data bits:" GridPane.columnIndex="0" GridPane.rowIndex="2" />
```

```
    <ComboBox fx:id="dataBits" maxWidth="1.7976931348623157E308" prefWidth="-1.0"
GridPane.columnIndex="1" GridPane.rowIndex="2" />
    <Label text="Parity check:" GridPane.columnIndex="0" GridPane.rowIndex="3" />
    <ComboBox fx:id="parityCheck" maxWidth="1.7976931348623157E308" prefWidth="-1.0"
GridPane.columnIndex="1" GridPane.rowIndex="3" GridPane.vgrow="ALWAYS" />
    <Button alignment="BASELINE_CENTER" defaultButton="true"
maxWidth="1.7976931348623157E308" mnemonicParsing="false" onAction="#onConnect"
text="Connect" GridPane.columnIndex="3" GridPane.halignment="CENTER"
GridPane.hgrow="ALWAYS" GridPane.rowIndex="4" />
    <Label text="Stop bits:" GridPane.columnIndex="2" GridPane.rowIndex="3" />
    <ComboBox fx:id="stopBits" maxWidth="1.7976931348623157E308" prefWidth="-1.0"
GridPane.columnIndex="3" GridPane.rowIndex="3" />
    <Label text="Baud rate:" GridPane.columnIndex="2" GridPane.rowIndex="2" />
    <ComboBox fx:id="baudRate" maxWidth="1.7976931348623157E308" prefWidth="-1.0"
GridPane.columnIndex="3" GridPane.rowIndex="2" />
    <Button fx:id="refresh" maxWidth="1.7976931348623157E308" mnemonicParsing="false"
onAction="#onRefresh" prefWidth="-1.0" text="Refresh" textAlignment="CENTER"
GridPane.columnIndex="3" GridPane.hgrow="ALWAYS" GridPane.rowIndex="1" />
  </children>
  <columnConstraints>
    <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
    <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
    <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
    <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
  </columnConstraints>
  <padding>
    <Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />
  </padding>
  <rowConstraints>
    <RowConstraints minHeight="10.0" vgrow="SOMETIMES" />
    <RowConstraints minHeight="10.0" vgrow="SOMETIMES" />
    <RowConstraints minHeight="10.0" vgrow="SOMETIMES" />
    <RowConstraints minHeight="10.0" vgrow="SOMETIMES" />
  </rowConstraints>
</GridPane>
```

3.2. Класс ChatController

Класс-контроллер для главного окна чата.

3.2.1. Поля

- @FXML private Button sendButton;
  Кнопка Send
- @FXML private WebView webView;
  Окно сообщений, на базе веб-браузера
- @FXML private TextArea inputField;
  Поле ввода сообщения для передачи
- @FXML private Circle statusIcon;
  Лампочка состояния связи
- @FXML private Label statusText;
  Текст состояния связи

- @FXML private Circle ctsIcon;
  Лампочка, отображающая состояние линии CTS
- public static final String PROGRAM_NAME = "ComChat";
  Имя программы
- public static final String PROGRAM_VERSION = "v0.1 alpha";
  Версия программы

- private ProtocolStack protocolStack;
  Ссылка на стек собранный стек протоколов
- private Status status;
  Состояние связи

- private String localUser = null;
  Имя локального пользователя
- private String remoteUser = null;
  Имя удалённого пользователя
- private boolean isAuthorized = false;
  Факт авторизованности локального пользователя

- private static final String messageSent = "[×] ";
  Префикс отправленного, но не доставленного сообщения
- private static final String messageReceived = "[«] ";
  Префикс принятого сообщения
- private static final String messageAck = "[»] ";
  Префикс доставленного сообщения

- private boolean isConnected = false;
  Факт наличия соединения
- private Integer messageId = 0;
  Абсолютный номер сообщения, используется как id для div'ов сообщений в окне чата
- private boolean isClosing = false;
  Факт того, что приложение находится в процессе завершения своей работы
- private HashMap<Integer, Integer> messageIdToHtmlId = new HashMap<>();
  Отображение id сообщения -> div id строки в окне чата

3.2.2. События

- public void onMenuConnect(ActionEvent event)
  Нажатие Connect в меню
- public void onMenuDisconnect(ActionEvent event)
  Нажатие Disconnect в меню
- public void onMenuAbout(ActionEvent event)
  Нажатие About в меню
- public void onKeyReleased(KeyEvent event)
  Нажатие Enter в поле ввода сообщения
- public void sendClick(ActionEvent event)
  Нажатие кнопки Send

3.2.3. Методы

- public void initWithData(Stage stage, Object data)
  Унаследован от DataController
- private void addUserMessage(String author, String message)
  Добавление пользовательского сообщения в чат
  - author – от чьего имени
  - message – текст сообщения
- private void addSystemMessage(MessageLevel level, String message)
  Добавление системного сообщения в чат
  - level – уровень сообщения
  - message – текст сообщения
- private void send()
  Отправка сообщения из поля ввода
- private void receive(Message message)
  Приём сообщения от прикладного уровня
  - message – принятое сообщение
- private void markMessage(int id)
  Пометка сообщения как доставленного
  - id – номер сообщения прикладного уровня
- private void onConnect()
  Обработка подключения
- private String getHtmlPage()
  Возвращает базовую html-вёрстку окна чата
- private void scrollChatToId(int id)
  Прокрутка чата к id'ному сообщению
  - id – id сообщения в вёрстке (div#id)
- private void updateStatus(boolean connected)
  Смена состояния связи
  - connected – факт наличия соединения
- private void updateCompanionStatus(boolean connected)
  Смена состояния оппонента
  - connected – факт наличия собеседника (факт того, что Auth принят)
- private void updateCTS(boolean CTS)
  Смена состояния CTS
  - CTS – наличие или отсутствие сигнала на линии CTS
- private void onError(Exception e)
  Ошибка от любого уровня
  - e - исключение

3.2.4. Листинг

```
package gui;

import javafx.application.Platform;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
```

```
import javafx.scene.control.TextArea;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.web.WebEngine;
import javafx.scene.web.WebView;
import javafx.stage.Modality;
import javafx.stage.Stage;
import javafx.stage.StageStyle;
import layers.ProtocolStack;
import layers.apl.Message;
import org.controlsfx.dialog.Dialogs;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.Text;
import sun.reflect.generics.reflectiveObjects.NotImplementedException;

import java.io.IOException;
import java.util.HashMap;

public class ChatController extends DataController {
    @FXML private Button sendButton;
    @FXML private WebView webView;
    @FXML private TextArea inputField;
    @FXML private VBox layout;
    @FXML private Circle statusIcon;
    @FXML private Label statusText;
    @FXML private Circle ctsIcon;
    @FXML private Label ctsText;

    public static final String PROGRAM_NAME = "ComChat";
    public static final String PROGRAM_VERSION = "v0.1 alpha";

    private ProtocolStack protocolStack;
    private Status status;

    private String localUser = null;
    private String remoteUser = null;
    private boolean isAuthorized = false;

    private static final String messageSent = "[×] ";
    private static final String messageReceived = "[«] ";
    private static final String messageAck = "[»] ";

    private boolean isConnected = false;

    private Integer messageId = 0;

    private boolean isClosing = false;
```

```
    private HashMap<Integer, Integer> messageIdToHtmlId = new HashMap<>();

    @Override
    public void initWithData(Stage stage, Object data) {
        super.initWithData(stage, data);
        protocolStack = (ProtocolStack) data;

        WebEngine engine = webView.getEngine();
        engine.loadContent(getHtmlPage());

        statusIcon.setFill(Status.NotConnected.toColor());
        statusText.setText(Status.NotConnected.toString());

        protocolStack.getPhy().subscribeConnectionStatusChanged(this::updateStatus);

protocolStack.getPhy().subscribeCompanionConnectedChanged(this::updateCompanionStatus);
        protocolStack.getPhy().subscribeSendingAvailableChanged(this::updateCTS);

        stage.setOnCloseRequest(e -> {
            isClosing = true;
            if (status != Status.NotConnected)
                protocolStack.getApl().disconnect();
        });

        protocolStack.getApl().subscribeToReceive(this::receive);

        Platform.runLater(this::showConnectionDialog);
    }

    private void updateStatus(boolean connected) {
        this.isConnected = connected;
        Status newStatus = Status.fromBoolean(connected);
        if (status == newStatus)
            return;

        Platform.runLater(() -> {
            status = newStatus;
            statusIcon.setFill(status.toColor());
            statusText.setText(status.toString());
            //sendButton.setDisable(!connected);

            if (status == Status.NotConnected && !isClosing) {
                isAuthorized = false;
                protocolStack.getApl().disconnect();
                addSystemMessage(MessageLevel.Info, "Disconnected");
                Dialogs.create()
                        .owner(stage)
                        .title(PROGRAM_NAME)
                        .masthead("Information")
                        .message("Disconnected")
                        .showInformation();
```

```
            }
        });
    }

    private void updateCompanionStatus(boolean connected) {
        if (!this.isConnected)
            return;

        Status newStatus = connected ? Status.Chatting : Status.Connected;

        if (status != newStatus) {
            //if we are connected and connection form closed (overwise Auth will be sent after form
closing)
            if (connected && !isAuthorized && localUser != null) {
                protocolStack.getApl().send(Message.Type.Auth, localUser);
                isAuthorized = true;
            }
            else if (status == Status.Chatting) {
                addSystemMessage(MessageLevel.Info, "Remote user disconnected, waiting for
another companion...");
                isAuthorized = false;
            }

            Platform.runLater(() -> {
                status = newStatus;
                statusIcon.setFill(status.toColor());
                statusText.setText(status.toString());
                sendButton.setDisable(!connected);
            });
        }
    }

    private void updateCTS(boolean CTS) {
        Platform.runLater(() -> {
            if (CTS)
                ctsIcon.setFill(Color.YELLOWGREEN);
            else
                ctsIcon.setFill(Color.RED);
        });
    }

    private void addUserMessage(String author, String message) {
        addUserMessage(author, message, 0);
    }

    private void addUserMessage(String author, String message, Integer id) {
        WebEngine engine = webView.getEngine();
        Document document = engine.getDocument();
        Node body = document.getElementsByTagName("BODY").item(0);
        Element div = document.createElement("div");

        messageIdToHtmlId.put(id, messageId);
```

```
        div.setAttribute("id", messageId.toString());
        div.setAttribute("data-message-id", id.toString());

        Text text = document.createTextNode(message);

        Element mark = document.createElement("b");
        if (id != 0)
            mark.setTextContent(messageSent);
        else
            mark.setTextContent(messageReceived);

        Element b = document.createElement("b");
        b.setTextContent(author + ": ");

        div.appendChild(mark);
        div.appendChild(b);
        div.appendChild(text);

        body.appendChild(div);
        scrollChatToId(messageId++);
    }

    private void addSystemMessage(MessageLevel level, String message) {
        Platform.runLater(() -> {
            WebEngine engine = webView.getEngine();
            Document document = engine.getDocument();

            Node body = document.getElementsByTagName("BODY").item(0);
            Element div = document.createElement("div");
            div.setAttribute("id", messageId.toString());

            div.setAttribute("style", "color: " + level.toHtmlColor());

            Text text = document.createTextNode(message);

            Element b = document.createElement("b");
            b.setTextContent("[" + level.toString() + "]" + " System message: ");

            div.appendChild(b);
            div.appendChild(text);

            body.appendChild(div);
            scrollChatToId(messageId++);
        });
    }

    private void send() {
        String message = inputField.getText();

        int id = protocolStack.getApl().send(Message.Type.Msg, message);

        addUserMessage(localUser, message, id);
```

```
        inputField.clear();
    }

    private void receive(Message message) {
        //JavaFX UI thread synchronization
        Platform.runLater(() -> {
            addSystemMessage(MessageLevel.Debug, "Message received: " +
message.getType().name());

            switch (message.getType()) {
                case Msg:
                    addUserMessage(remoteUser, message.getMsg());
                    protocolStack.getApl().send(Message.Type.Ack,
                            Integer.toString(message.getId()));
                    break;
                case Auth:
                    remoteUser = message.getMsg();
                    addSystemMessage(MessageLevel.Info, "Remote user connected: " +
message.getMsg());
                    protocolStack.getApl().handshakeFinished();
                    //TODO: enable "sendability"
                    break;
                case Ack:
                    int id = Integer.parseInt(message.getMsg());
                    markMessage(id);
                    break;
                default:
                    throw new NotImplementedException();
            }
        });
    }

    private void markMessage(int id) {
        WebEngine engine = webView.getEngine();
        Document document = engine.getDocument();

        Integer htmlId = messageIdToHtmlId.get(id);
        if (htmlId == null)
            return;

        Element div = document.getElementById(htmlId.toString());
        Element mark = (Element) div.getFirstChild();

        mark.setTextContent(messageAck);
    }

    public void sendClick(ActionEvent event) {
        send();
    }

    public void onMenuConnect(ActionEvent event) throws IOException {
        showConnectionDialog();
```

```
    }

    private void showConnectionDialog() {
        try {
            FXMLLoader loader = new
FXMLLoader(Main.class.getResource("/gui/templates/connection.fxml"));

            Parent root = loader.load();

            DataStage connectionStage = new DataStage(loader.getController(), protocolStack);

            connectionStage.setTitle("Connection");
            final double rem = javafx.scene.text.Font.getDefault().getSize() / 13;
            Scene conScene = new Scene(root, 400.0*rem, 205.0*rem);
            connectionStage.setScene(conScene);
            connectionStage.setResizable(false);
            connectionStage.initModality(Modality.WINDOW_MODAL);
            connectionStage.initOwner(layout.getScene().getWindow());
            connectionStage.initStyle(StageStyle.UTILITY);
            connectionStage.showAndWait();

            if (connectionStage.getResult() == DialogResult.OK) {
                localUser = (String) connectionStage.getResultData();
                onConnect();
            } else {
                Dialogs.create()
                        .owner(stage)
                        .title(PROGRAM_NAME)
                        .masthead("Information")
                        .message("Connection cancelled")
                        .showInformation();
            }
        }
        catch (IOException e) {
            Dialogs.create()
                    .owner(stage)
                    .title(PROGRAM_NAME)
                    .masthead("Error")
                    .showException(e);
        }
    }

    private void onConnect() {
        //not sure why runLater needed here, cause we are already in UI thread and webEngine
already loaded
        Platform.runLater(() -> addSystemMessage(MessageLevel.Info, "Successfully
connected"));

        if (!isAuthorized) {
            isAuthorized = true;
            protocolStack.getApl().send(Message.Type.Auth, localUser);
        }
```

```
        protocolStack.getApl().subscribeOnError(this::onError);
    }

    private void onError(Exception e) {
        protocolStack.getApl().disconnect();

        addSystemMessage(MessageLevel.Error, "Connection lost");
        Dialogs.create()
                .owner(stage)
                .title(PROGRAM_NAME)
                .masthead("Error")
                .message("Connection lost")
                .showException(e);
    }

    public void onMenuDisconnect(ActionEvent event) {
        if (status != Status.NotConnected)
            protocolStack.getApl().disconnect();
    }

    private String getHtmlPage() {
        return "<html><head></head><body></body></html>";
    }

    public void onMenuAbout(ActionEvent event) {
        Dialogs.create()
                .owner(stage)
                .title(PROGRAM_NAME)
                .masthead("About")
                .message("BMSTU course project.\nCOM-port based chat for 2
persons.\n\nAuthors:\nLeontiev Aleksey - Application Layer and GUI\nLatkin Igor - Physical
Layer\nKornukov Nikita - Data Link Layer\n\nProject's home:\nhttps://github.com/tech-
team/comchat")
                .showInformation();
    }

    public void onKeyReleased(KeyEvent event) {
        if (event.getCode() == KeyCode.ENTER && !event.isControlDown()) {
            if (status != Status.NotConnected)
                send();
            else
                Dialogs.create()
                        .owner(stage)
                        .title(PROGRAM_NAME)
                        .masthead("Error")
                        .message("You should connect first.\nUse Connection -> Connect.")
                        .showError();
        }
    }

    private void scrollChatToId(int id) {
```

```
        try {
            WebEngine engine = webView.getEngine();
            engine.executeScript("document.getElementById(" + id + ").scrollIntoView()");
        }
        catch (NullPointerException ignored) {}
    }
}
```

3.3. Класс ChatController

Класс-контроллер для окна настроек подключения

3.3.1. Поля

- @FXML private TextField userName;
  Поле ввода имени пользователя
- @FXML private ComboBox<String> comPort;
  Список выбора комп-порта
- @FXML private ComboBox<Integer> baudRate;
  Список выбора скорости передачи
- @FXML private ComboBox<String> dataBits;
  Список выбора количества информационных битов
- @FXML private ComboBox<String> stopBits;
  Список выбора количества стоповых битов
- @FXML private ComboBox<String> parityCheck;
  Список выбора способа проверки чётности
- private ProtocolStack protocolStack;
  Ссылка на стек собранный стек протоколов

3.3.2. События

- public void onConnect(ActionEvent event)
  Нажатие на кнопку Connect
- public void onRefresh(ActionEvent event)
  Нажатие на кнопку Refresh (обновления списка ком-портов)

3.3.3. Методы

Все методы унаследованы от DataController.

3.3.4. Листинг

```
package gui;

import javafx.application.Platform;
import javafx.collections.FXCollections;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.ComboBox;
import javafx.scene.control.TextField;
import javafx.stage.Stage;
import layers.ProtocolStack;
import layers.exceptions.ConnectionException;
import layers.phy.ComPort;
```

```
import layers.phy.settings.comport_settings.ComPortSettings;
import layers.phy.settings.comport_settings.DataBitsEnum;
import layers.phy.settings.comport_settings.ParityEnum;
import layers.phy.settings.comport_settings.StopBitsEnum;
import org.controlsfx.dialog.Dialogs;

public class ConnectionController extends DataController {
   @FXML private TextField userName;
   @FXML private ComboBox<String> comPort;
   @FXML private ComboBox<Integer> baudRate;
   @FXML private ComboBox<String> dataBits;
   @FXML private ComboBox<String> stopBits;
   @FXML private ComboBox<String> parityCheck;

   private ProtocolStack protocolStack;

   @Override
   public void initWithData(Stage stage, Object data) {
      super.initWithData(stage, data);
      protocolStack = (ProtocolStack) data;

      comPort.setItems(FXCollections.observableArrayList(ComPort.getAvailablePorts()));

baudRate.setItems(FXCollections.observableArrayList(ComPort.getAvailableBaudRates()));
      dataBits.setItems(FXCollections.observableArrayList(ComPort.getAvailableDataBits()));
      stopBits.setItems(FXCollections.observableArrayList(ComPort.getAvailableStopBits()));
      parityCheck.setItems(FXCollections.observableArrayList(ComPort.getAvailableParity()));

      comPort.setValue(ComPort.getDefaultPort());
      baudRate.setValue(ComPort.getDefaultBaudRate());
      dataBits.setValue(ComPort.getDefaultDataBits());
      stopBits.setValue(ComPort.getDefaultStopBits());
      parityCheck.setValue(ComPort.getDefaultParity());
   }


   public void onConnect(ActionEvent event) {
      if (comPort.getValue().isEmpty())
      {
         Dialogs.create()
               .owner(stage)
               .title(ChatController.PROGRAM_NAME)
               .masthead("Error")
               .message("Please, select COM port.\nIf you do not see any listed, press Refresh
button.")
               .showError();

         return;
      }

      if (userName.getText().isEmpty())
      {
```

```
            Dialogs.create()
                    .owner(stage)
                    .title(ChatController.PROGRAM_NAME)
                    .masthead("Error")
                    .message("Please, input your username.")
                    .showError();

            return;
        }

        try {
            protocolStack.getApl().connect(
                    new ComPortSettings(comPort.getValue(),
                            baudRate.getValue(),
                            DataBitsEnum.fromString(dataBits.getValue()).toDataBits(),
                            StopBitsEnum.fromString(stopBits.getValue()).toStopBits(),
                            ParityEnum.fromString(parityCheck.getValue()).toParity()));

            result = DialogResult.OK;
            resultData = userName.getText();
            this.close();
        }
        catch(ConnectionException e) {
            String message = "Unable to connect with these settings";
            Exception withMessage = new Exception(message, e);

            Platform.runLater(() -> Dialogs.create()
                    .owner(stage)
                    .title("ComChat")
                    .masthead("Error")
                    .message(message) //well, that has no effect for exception dialog unfortunately
                    .showException(withMessage));
        }
    }

    public void onRefresh(ActionEvent event) {
        comPort.setItems(FXCollections.observableArrayList(ComPort.getAvailablePorts(true)));
        comPort.setValue(ComPort.getDefaultPort());
    }
}
```

3.4. Класс DataController

Абстрактный класс-контроллер с возможностью параметризованной инициализации и возврата значения и данных

3.4.1. Поля

- protected Object data;
  Входные данные
- protected Object resultData;
  Выходные данные

- protected DialogResult result = DialogResult.CLOSED;
  Результат работы
- protected Stage stage;
  Stage, на котором происходит отрисовка шаблона

3.4.2. События

- protected void close()
  Закрытие сцены (окна)

3.4.3. Методы

- public void initWithData(Stage stage, Object data)
  Инициализация котроллера
  - stage – сцена, на которой размещены элементы
  - data – произвольный входной объект
- public DialogResult getResult()
  Возврат результата работы окна
- public Object getResultData()
  Возврат результата работы окна (данные)
- public void setStage(Stage stage)
  Установка сцены для контроллера
  - stage – сцена к установке

3.4.4. Листинг

```
package gui;

import javafx.fxml.FXML;
import javafx.stage.Stage;

public abstract class DataController {
   protected Object data;
   protected Object resultData;
   protected DialogResult result = DialogResult.CLOSED;
   protected Stage stage;

   public void initWithData(Stage stage, Object data) {
      this.stage = stage;
      this.data = data;
   }

   public DialogResult getResult() {
      return result;
   }

   public Object getResultData() {
      return resultData;
   }

   public void setStage(Stage stage) {
      this.stage = stage;
   }
```

```
    @FXML
    protected void close() {
        stage.close();
    }
}
```

3.5. Класс DataStage

Абстрактный класс сцены (окна) с возможностью параметризованной инициализации и возврата значения и данных

3.5.1. Поля

- private DataController controller;
  Ссылка на соответствующий сцене котроллер

3.5.2. События

- protected void close()
  Закрытие сцены (окна)

3.5.3. Методы

- public DataStage(DataController controller, Object data)
  Конструктор
  - controller для привязки к сцене
  - data – произвольная входная информация
- public DialogResult getResult()
  Возврат результата работы окна
- public Object getResultData()
  Возврат результата работы окна (данные)
- public DataController getController()
  Возврат контроллера

3.5.4. Листинг

```
package gui;
import javafx.stage.Stage;

public class DataStage extends Stage {
    private DataController controller;

    public DataStage(DataController controller, Object data) {
        this.controller = controller;
        controller.initWithData(this, data);
    }

    public DialogResult getResult() {
        return controller.getResult();
    }

    public Object getResultData() {
        return controller.getResultData();
    }
```

```
    public DataController getController() {
        return controller;
    }
}
```

3.6. Перечисление DialogResult

Перечисление возможных результатов работы окна

3.6.1. Значения

- OK
  Нажата кнопка OK
- CANCEL
  Нажата кнопка CANCEL
- YES
  Нажата кнопка YES
- NO
  Нажата кнопка YES
- CLOSED
  Диалог закрыт
- ERROR
  Произошла ошибка

3.7. Класс Main

Главный класс

3.7.1. События

- public void start(Stage primaryStage)
  Происходит после инициализации JavaFX
  - primaryStage – объект сцены (главное окно)

3.7.2. Методы

- public static void main(String[] args)
  Точка входа в приложение
  - args – массив аргументов командной строки

3.8. Перечисление MessageLevel

Перечисление возможных типов сообщений в чате

3.8.1. Значения

- Error("ERROR", "red")
  Сообщение об ошибке
- Info("INFO", "gray")
  Информационной сообщение (основной тип)
- Debug("DEBUG", "blue")
  Отладочное сообщение

3.8.2. Поля

- String level;
  Строковое описание

- String htmlColor;
  Цвет в html-формате

3.8.2. Методы

- MessageLevel(String level, String htmlColor)
  Конструктор
  - level – уровень сообщения
  - htmlColor – цвет сообщения в html представлении
- public String toString()
  Получение строкового представления
- public String toHtmlColor()
  Получение цветового представления

3.9. Перечисление Status

Перечисление возможных состояний связи

3.9.1. Значения

- NotConnected("Not connected", Color.RED)
  Не подключено
- Connected("Connected. Waiting for companion", Color.YELLOW)
  Подключено
- Chatting("Chatting", Color.YELLOWGREEN)
  Идёт переписка (т.е. подключено и на другом конце есть собеседник)
- Error("Error", Color.RED)
  Произошла ошибка

3.9.2. Поля

- String value;
  Строковое описание
- Color color;
  Цветовое представление

3.9.2. Методы

- private Status(String value, Color color)
  Конструктор
  - value – строковое значение
  - color – цвет статуса
- public String toString()
  Получение строкового представления
- public String toHtmlColor()
  Получение цветового представления
- public static Status fromBoolean(boolean connected)
  Создание статуса по булевому значению (подключен/не подключен)
  - connected – факт наличия соединения

4. Пакет layers

4.1. Интерфейс ILayer

Описывает абстрактную сущность уровня стека протоколов.

4.1.1. Методы

- ILayer getUpperLayer();
  Возвращает вышележащий уровень стека
- ILayer getLowerLayer();
  Возвращает нижележащий уровень стека
- void setUpperLayer(ILayer layer);
  Устанавливает вышележащий уровень стека
- void setLowerLayer(ILayer layer);
  Устанавливает нижележащий уровень стека
- void connect(PhysicalLayerSettings settings);
  Выполняет подключение
- void disconnect();
  Выполняет отключение
- boolean isConnected();
  Возвращает факт наличия подключения
- void notifyOnError(Exception e);
  Вызывается нижними уровнями для оповещение об ошибке

4.1.2. Листинг

```
package layers;

import layers.exceptions.ConnectionException;
import layers.phy.settings.PhysicalLayerSettings;

public interface ILayer {
   ILayer getUpperLayer();
   ILayer getLowerLayer();
   void setUpperLayer(ILayer layer);
   void setLowerLayer(ILayer layer);

   void connect(PhysicalLayerSettings settings) throws ConnectionException;
   void disconnect();
   boolean isConnected();

   void notifyOnError(Exception e);
}
```

4.2. Интерфейс PDU

4.2.1. Методы

- byte[] serialize();
  Выполняет преобразование текущего объекта в массив байт
- static PDU deserialize(byte[] data);
  Создаёт объект PDU из массива байт *data*

4.2.2. Листинг

```
package layers;

public interface PDU  {
   byte[] serialize();
   static PDU deserialize(byte[] data) {
      return null;
   }
}
```

4.3. Класс ProtocolStack

Реализует понятие стека протоколов. На вход ему передаются три произвольных класса-наследника ILayer, затем все они связываются через setUpperLayer() и setLowerLayer() в единую систему.

4.3.1. Поля

- IApplicationLayer apl;
  Объект прикладного уровня
- IDataLinkLayer dll;
  Объект канального уровня
- IPhysicalLayer phy;
  Объект физического уровня

4.3.2. Методы

- public ProtocolStack(Class apl, Class dll, Class phy)
  Конструктор
  - *apl*, *dll*, *phy* – классы, из объектов которых будет собран стек
- public IApplicationLayer getApl()
  Возвращает объект прикладного уровня
- public IDataLinkLayer getDll()
  Возвращает объект канального уровня
- public IPhysicalLayer getPhy()
  Возвращает объект физического уровня

4.3.3. Листинг

```
package layers;

import layers.apl.IApplicationLayer;
import layers.dll.IDataLinkLayer;
import layers.exceptions.ProtocolStackException;
import layers.phy.IPhysicalLayer;

public class ProtocolStack {
   IApplicationLayer apl;
   IDataLinkLayer dll;
   IPhysicalLayer phy;

   public ProtocolStack(Class apl, Class dll, Class phy) throws ProtocolStackException {
      try {
         this.apl = (IApplicationLayer) apl.newInstance();
```

```
            this.dll = (IDataLinkLayer) dll.newInstance();
            this.phy = (IPhysicalLayer) phy.newInstance();

            this.apl.setLowerLayer(this.dll);

            this.dll.setUpperLayer(this.apl);
            this.dll.setLowerLayer(this.phy);

            this.phy.setUpperLayer(this.dll);

        }
        catch (Exception e) {
            throw new ProtocolStackException(e);
        }
    }

    public IApplicationLayer getApl() {
        return apl;
    }

    public IDataLinkLayer getDll() {
        return dll;
    }

    public IPhysicalLayer getPhy() {
        return phy;
    }
}
```

4.4. Пакет apl

4.4.1. Интерфейс IApplicationLayer

Абстракция прикладного уровня

4.4.1.1. Методы

- void connect(PhysicalLayerSettings settings);
  Соединиться
  - settings – настройки для передачи на физический уровень
- void disconnect();
  Отсоединиться
- int send(Message.Type type, String msg);
  Послать сообщение
  - type – тип сообщения
  - msg – текст сообщения
- void receive(byte[] data);
  Принять сообщение
  - data – сериализованное сообщения
- void subscribeToReceive(final Consumer<Message> receiver);
  Подписаться на получение сообщений
  - receiver - обработчик

- void handshakeFinished();
  Завершить процедуру авторизации

4.4.1.2. Листинг

```
package layers.apl;

import layers.ILayer;
import layers.exceptions.ConnectionException;
import layers.phy.settings.PhysicalLayerSettings;

import java.util.function.Consumer;

public interface IApplicationLayer extends ILayer {
    void connect(PhysicalLayerSettings settings) throws ConnectionException;
    void disconnect();
    int send(Message.Type type, String msg);
    void receive(byte[] data);
    void subscribeToReceive(final Consumer<Message> receiver);

    void handshakeFinished();
}
```

4.4.2. Класс ApplicationLayer

Реализация прикладного уровня

4.4.2.1. Поля

- private IDataLinkLayer dll
  Ссылка на канальный уровень
- private List<Consumer<Message>> receivers
  Список подписок на приём сообщений
- private int messageId
  текущий id переданного сообщения
- private List<Consumer<Exception>> onErrorListeners
  Список подписок на ошибки

4.4.2.1. Методы

Все методы являются реализациями методом интерфейсов IApplicationLayer и ILayer.

4.4.2.2. Листинг

```
package layers.apl;

import layers.ILayer;
import layers.dll.IDataLinkLayer;
import layers.exceptions.ConnectionException;
import layers.phy.settings.PhysicalLayerSettings;

import java.util.LinkedList;
import java.util.List;
import java.util.function.Consumer;
```

```
public class ApplicationLayer implements IApplicationLayer {
    private IDataLinkLayer dll;
    private List<Consumer<Message>> receivers = new LinkedList<>();
    private int messageId = 0;

    private List<Consumer<Exception>> onErrorListeners = new LinkedList<>();

    @Override
    public ILayer getUpperLayer() {
        return null;
    }

    @Override
    public IDataLinkLayer getLowerLayer() {
        return dll;
    }

    @Override
    public void setUpperLayer(ILayer layer) {
    }

    @Override
    public void setLowerLayer(ILayer layer) {
        dll = (IDataLinkLayer) layer;
    }

    @Override
    public void subscribeOnError(Consumer<Exception> listener) {
        onErrorListeners.add(listener);
    }

    private void notifyOnError(Exception e) {
        onErrorListeners.forEach(listener -> listener.accept(e));
    }

    @Override
    public void connect(PhysicalLayerSettings settings) throws ConnectionException {
        getLowerLayer().connect(settings);
    }

    @Override
    public void disconnect() {
        getLowerLayer().disconnect();
        messageId = 1;
    }

    @Override
    public int send(Message.Type type, String msg) {
        System.out.println("Sent: " + msg);

        if (type == Message.Type.Msg)
            ++messageId;
```

```
        dll.send(new Message(messageId, type, msg).serialize());

        return messageId;
    }

    @Override
    public void receive(byte[] data) {
        Message message = Message.deserialize(data);
        System.out.println("Received: " + message.getMsg());
        receivers.forEach(receiver -> receiver.accept(message));
    }

    @Override
    public void subscribeToReceive(final Consumer<Message> receiver) {
        receivers.add(receiver);
    }

    @Override
    public void handshakeFinished() {
        getLowerLayer().handshakeFinished();
    }
}
```

4.4.3. Класс Message

PDU прикладного уровня, сообщение.

4.4.3.1. Перечисление Type

4.4.3.1.1. Значения

- Auth
  Сообщение об авторизации
- Msg
  Пользовательское сообщение
- Ack
  Подтверждение о доставке пользовательского сообщения

4.4.3.1.2. Методы

- public static Type fromInteger(int x)
  Создание типа по соответствующему ему числу (при десериализации)
  - x – тип в числовом представлении (результат ordinal())

4.4.3.2. Поля

- private int id;
  Последовательный уникальный идентификатор сообщения
- private Type type;
  Тип сообщения
- private String msg;
  Текст сообщения

4.4.3.3. Методы

- public Message(int id, Type type, String msg)
  Конструктор
  - *id* – номер сообщения (равен нулю для всех типов сообщений, кроме Msg)
  - *type* – тип сообщения
- public int getId()
  Возвращает id сообщения
- public Type getType()
  Возвращает тип сообщения
- public String getMsg()
  Возвращает текст сообщения
- public byte[] serialize()
  Сериализация сообщения в массив байт
- public static Message deserialize(byte[] data)
  Десериализация сообщения из массива байт
  - *data* – сериализованное сообщение

4.4.3.4. Листинг

```
package layers.apl;

import layers.PDU;
import util.ArrayUtils;

import java.io.UnsupportedEncodingException;
import java.nio.ByteBuffer;
import java.util.Arrays;

public class Message implements PDU {

   public enum Type {
      Auth, Msg, Ack;

      public static Type fromInteger(int x) throws Exception {
         int max = Type.values().length;
         if (x < 0 || x >= max)
            throw new Exception("bad range! x was: " + x);

         return Type.values()[x];
      }
   }

   private int id;
   private Type type;
   private String msg;

   public Message(int id, Type type, String msg) {
      this.id = id;
      this.type = type;
      this.msg = msg;
   }
```

```
    public int getId() {
        return id;
    }

    public Type getType() {
        return type;
    }

    public String getMsg() {
        return msg;
    }

    @Override
    public byte[] serialize() {
        byte typeByte = (byte) type.ordinal();
        byte[] idBytes = ByteBuffer.allocate(4).putInt(this.id).array();
        byte[] infoBytes = ArrayUtils.concatenate(typeByte, idBytes);

        byte[] msgBytes = new byte[0];
        try {
            msgBytes = msg.getBytes("UTF-8");
        } catch (UnsupportedEncodingException ignored) {
            System.err.println(ignored.getMessage());
        }
        return ArrayUtils.concatenate(infoBytes, msgBytes);
    }

    public static Message deserialize(byte[] data)  {
        byte typeByte = data[0];
        byte[] idBytes = Arrays.copyOfRange(data, 1, 5);
        byte[] msgBytes = Arrays.copyOfRange(data, 5, data.length);

        Type type = null;
        try {
            type = Type.fromInteger((int) typeByte);
        } catch (Exception e) {
            e.printStackTrace(); // TODO: review
        }

        int id = ByteBuffer.wrap(idBytes).getInt();

        String msg = null;
        try {
            msg = new String(msgBytes, "UTF-8");
        } catch (UnsupportedEncodingException ignored) { }

        return new Message(id, type, msg);
    }
}
```

4.5. Пакет dll

4.5.1. Интерфейс IDataLinkLayer

Описывает абстрактную сущность, реализующую канальный уровень

4.5.1.1. Методы

- void send(byte[] msg);
  Вызывается с верхнего уровня при передачи сериализованного сообщения *msg*
- void receive(byte[] data);
  Вызывается с нижнего уровня для приёма сериализованного кадра *data*
- void handshakeFinished();
  Вызывается с верхнего уровня для указания на то, что второй узел подключен

4.5.1.2. Листинг

```
package layers.dll;

import layers.ILayer;

public interface IDataLinkLayer extends ILayer {
   void send(byte[] msg);
   void receive(byte[] data);

   void handshakeFinished();
}
```

4.5.2. Класс DataLinkLayer

4.5.2.1. Поля

- private IApplicationLayer apl;
  Ссылка на верхний уровень
- private IPhysicalLayer phy;
  Ссылка ан нижний уровень
- private Queue<byte[]> framesToSend = new ConcurrentLinkedQueue<>();
  Очередь кадров на отправку
- private Queue<byte[]> systemFramesToSend = new ConcurrentLinkedQueue<>();
  Очередь системных (управляющих кадров) на отправку
- private List<byte[]> receivedChunkMessages = new LinkedList<>();
  В этом списке собирается пришедшее с физического уровня по частям сообщение
- private boolean remoteUserConnected = false;
  Факт наличия собеседника на другом узле
- private AtomicBoolean wasACK = new AtomicBoolean(true);
  Пришло ли подтверждение на последний отправленный кадр
- private AtomicBoolean forceSending = new AtomicBoolean(false);
  Разрешает посылку кадра, даже если подтверждение на предыдущий ещё не получено
- private Thread sendingThread;
  Поток, который отправляет данные из очередей на отправку
- private boolean sendingActive = false;
  Останавливает поток отправки при установке в false

- private static final int SENDING_DELAY = 100;
  Период отправки
- private static final int SENDING_TIMEOUT = 3000;
  Таймаут отправки
- private static final int ACCESSING_PHY_TIMEOUT = 5000;
  Таймаут недоступности физического уровня (CTS)

4.5.2.2. Методы

- private int getSendingCycles()
  Возвращает количество циклов отправки до истечения таймаута
- private int getPhyAccessingCycles()
  Возвращает количество циклов доступа к физическому уровню до истечения таймаута
- private void sendingThreadJob()
  Описывает поток отправки
- private void forceSend()
  Разрешает отправку следующего кадра из очереди, даже в случае, если ACK на предыдущий ещё не пришёл

Остальные методы унаследованы у IDataLinkLayer и ILayer.

4.5.2.3. Листинг

```
package layers.dll;

import layers.ILayer;
import layers.apl.IApplicationLayer;
import layers.exceptions.ConnectionException;
import layers.exceptions.LayerUnavailableException;
import layers.exceptions.UnexpectedChatException;
import layers.phy.IPhysicalLayer;
import layers.phy.settings.PhysicalLayerSettings;
import util.ArrayUtils;

import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;
import java.util.concurrent.ConcurrentLinkedQueue;
import java.util.concurrent.atomic.AtomicBoolean;

public class DataLinkLayer implements IDataLinkLayer {
    private IApplicationLayer apl;
    private IPhysicalLayer phy;

    private Queue<byte[]> framesToSend = new ConcurrentLinkedQueue<>();
    private Queue<byte[]> systemFramesToSend = new ConcurrentLinkedQueue<>();

    private List<byte[]> receivedChunkMessages = new LinkedList<>();
```

```
private boolean remoteUserConnected = false;
private AtomicBoolean wasACK = new AtomicBoolean(true);
private AtomicBoolean forceSending = new AtomicBoolean(false);


private Thread sendingThread;
private boolean sendingActive = false;
private static final int SENDING_DELAY = 100;
private static final int SENDING_TIMEOUT = 3000;
private static final int ACCESSING_PHY_TIMEOUT = 5000;


private int getSendingCycles() {
   return SENDING_TIMEOUT / SENDING_DELAY;
}

private int getPhyAccessingCycles() {
   return ACCESSING_PHY_TIMEOUT / SENDING_DELAY;
}

private void sendingThreadJob() {
   int sendingCycles = getSendingCycles();
   int accessingCycles = getPhyAccessingCycles();

   while (sendingActive) {

      if (systemFramesToSend.isEmpty() && framesToSend.isEmpty()) {
         sendingCycles = getSendingCycles();
         accessingCycles = getPhyAccessingCycles();
      }

      if (!systemFramesToSend.isEmpty()) {

         if (getLowerLayer().readyToSend()) {
            accessingCycles = getPhyAccessingCycles();
            getLowerLayer().send(systemFramesToSend.poll());
         }
         else { // phy is unavailable
            accessingCycles -= 1;
         }

      }
      else {

         if (!framesToSend.isEmpty()) {
            if (getLowerLayer().readyToSend()) {
               accessingCycles = getPhyAccessingCycles();
               if (forceSending.get() || wasACK.get()) { // if we are permitted to send next frame
                  sendingCycles = getSendingCycles();

                  getLowerLayer().send(framesToSend.peek());
                  wasACK.set(false);
```

```
                    forceSending.set(false);
                }
                else {
                    sendingCycles -= 1;
                }

                if (sendingCycles <= 0) {
                    forceSend();
                    sendingCycles = getSendingCycles();
                }
            }
            else { // phy is unavailable
                accessingCycles -= 1;
            }
        }
    }

    if (accessingCycles <= 0 && remoteUserConnected) {
        System.out.println("onError");
        notifyOnError(new LayerUnavailableException("Physical layer was unavailable for "
+ ACCESSING_PHY_TIMEOUT + "ms"));
        systemFramesToSend.clear();
        framesToSend.clear();
    }

    try {
        Thread.sleep(SENDING_DELAY);
    } catch (InterruptedException ignored) {
        sendingActive = false;
    }

  }
}

@Override
public void connect(PhysicalLayerSettings settings) throws ConnectionException {
  getLowerLayer().initMarkBytes(Frame.START_BYTE, Frame.STOP_BYTE);
  getLowerLayer().connect(settings);
  sendingThread = new Thread(this::sendingThreadJob);
  sendingActive = true;
  sendingThread.start();
}

@Override
public void disconnect() {
  sendingActive = false;
  framesToSend.clear();
  systemFramesToSend.clear();
  wasACK.set(true);
  getLowerLayer().disconnect();
}
```

```
    @Override
    public boolean isConnected() {
        return getLowerLayer().isConnected();
    }

    @Override
    public void send(byte[] data) {
        int chunks = (int) Math.ceil((double) data.length / (double) Frame.MAX_MSG_SIZE);
        for (int i = 0; i < chunks; ++i) {
            int minIndex = Frame.MAX_MSG_SIZE * i;
            int maxSize = Math.min(Frame.MAX_MSG_SIZE, data.length - minIndex);
            byte[] chunk = Arrays.copyOfRange(data, minIndex, minIndex + maxSize);
            Frame frameChunk;
            if (i < chunks - 1) {
                frameChunk = Frame.newCHUNKEDFrame(chunk);
            }
            else {
                frameChunk = Frame.newChunkEndFrame(chunk);
            }
            framesToSend.add(frameChunk.serialize());
        }
    }

    @Override
    public void receive(byte[] data) {
        Frame frame = Frame.deserialize(data);
//        System.out.println("isCorrect: " + frame.isCorrect());

        if (frame == null || frame.getType() != Frame.Type.I && !frame.isCorrect())
            return; // throw away this frame

        switch (frame.getType()) {
            case S:
                if (frame.isACK()) {
                    if (framesToSend.isEmpty()) {
                        notifyOnError(new UnexpectedChatException("Frame queue is empty"));
                        return;
                    }
                    framesToSend.poll();
                    wasACK.set(true);
                }
                else if (frame.isRET()) {
                    forceSend();
                }
                break;
            case I:
                if (frame.isCorrect()) {
                    Frame ack = Frame.newACKFrame();
                    systemFramesToSend.add(ack.serialize());


                    receivedChunkMessages.add(frame.getMsg());
```

```
                if (frame.isEND_CHUNKS()) {
                    byte[] resultedMsg = new byte[0];
                    for (byte[] chunk : receivedChunkMessages) {
                        resultedMsg = ArrayUtils.concatenate(resultedMsg, chunk);
                    }
                    receivedChunkMessages.clear();
                    apl.receive(resultedMsg);
                }
            }
            else {
                Frame ret = Frame.newRETFrame();
                systemFramesToSend.add(ret.serialize());
            }
            break;
        }
    }

    private void forceSend() {
        forceSending.set(true);
    }

    @Override
    public void handshakeFinished() {
        remoteUserConnected = true;
    }

    @Override
    public IApplicationLayer getUpperLayer() {
        return apl;
    }

    @Override
    public IPhysicalLayer getLowerLayer() {
        return phy;
    }

    @Override
    public void setUpperLayer(ILayer layer) {
        apl = (IApplicationLayer) layer;
    }

    @Override
    public void setLowerLayer(ILayer layer) {
        phy = (IPhysicalLayer) layer;
        phy.subscribeCcompanionConnectedChanged(status -> remoteUserConnected = status);
    }

    @Override
    public synchronized void notifyOnError(Exception e) {
        getUpperLayer().notifyOnError(e);
    }
}
```

4.5.3. Класс Frame

PDU канального уровня, кадр.

4.5.3.1. Перечисление Type

4.5.3.1.1. Значения

- I
  Информационный кадр
- S
  Управляющий кадр

4.5.3.1.2. Методы

- public static Type fromInteger(int x)
  Создание типа по соответствующему ему числу (при десериализации)
  - x – тип в числовом представлении (результат ordinal())

4.5.3.2. Поля

- public static final int MAX_SIZE = 128;
  Максимальный размер кадра в байтах
- public static final int MAX_MSG_SIZE = MAX_SIZE - 2;
  Максимальный размер области данных в кадре в байтах
- private boolean ACK;
  Бит ACK выставлен
- private boolean RET;
  Бит RET выставлен
- private boolean CHUNKS;
  Бит CHUNKS выставлен, означает то, что сообщение разбито на несколько кадров
- private boolean END_CHUNKS;
  Бит END_CHUNKS выставлен, означает то, что данный кадр является последним в цепочке кадров, передающий одно сообщение по частям

- public static final byte START_BYTE = (byte) 0xFF;
  Стартовый байт
- public static final byte STOP_BYTE = (byte) 0xFF;
  Стоповый байт

- private Type type;
  Тип кадра
- private byte[] msg;
  Сериализованное сообщение

- private boolean correct;
  Факт корректности кадра
- private static CycleCoder cc = new CycleCoder();
  Объект циклического кодировщика

4.5.3.3. Методы

- private Frame(Type type, byte[] msg)
  Конструктор

    - *type* – тип кадра
    - *msg* – сериализованное сообщение
- public static Frame newACKFrame()
  Создаёт управляющий кадр типа ACK
- public static Frame newRETFrame()
  Создаёт управляющий кадр типа RET
- public static Frame newCHUNKEDFrame(byte[] chunk)
  Создаёт новый кадр для передачи части сообщения
    - *chunk* – часть сериализованного сообщения
- public static Frame newChunkEndFrame(byte[] chunk)
  Создаёт кадр, завершающий цепочку передачи одного сообщения по частям
- public Type getType()
  Возвращает тип кадра
- public byte[] getMsg()
  Возвращает сериализованное сообщение в кадре
- public boolean isACK()
  Возвращает значение бита ACK
- public boolean isRET()
  Возвращает значение бита RET
- public boolean isCHUNK()
  Возвращает значение бита CHUNKS
- public boolean isEND_CHUNKS()
  Возвращает значение бита END_CHUNKS
- private void setACK(boolean value)
  Устанавливает бит ACK
- private void setRET(boolean value)
  Устанавливает бит RET
- private void setCHUNKS(boolean value)
  Устанавливает бит CHUNKS
- private void setEND_CHUNKS(boolean value)
  Устанавливает бит END_CHUNKS
- private byte getACK()
  Устанавливает бит ACK в виде байта
- private byte getRET()
  Устанавливает бит RET в виде байта
- private byte getCHUNKS()
  Устанавливает бит CHUNKS в виде байта
- private byte getEND_CHUNKS()
  Устанавливает бит ENC_CHUNKS в виде байта
- public boolean isCorrect()
  Возвращает факт корректности кадра
- private void setCorrect(boolean correct)
  Устанавливает факт корректности кадра

Остальные методы унаследованы от PDU.

4.5.3.4. Листинг

```
package layers.dll;

import layers.PDU;
```

```
import layers.exceptions.DecodeException;
import util.ArrayUtils;

import java.util.Arrays;

public class Frame implements PDU {
   public enum Type {
      I, S;

      public static Type fromInteger(int x) throws Exception {
         int max = Type.values().length;
         if (x < 0 || x >= max)
            throw new Exception("bad range! x was: " + x);

         return Type.values()[x];
      }
   }

   public static final int MAX_SIZE = 128;
   public static final int MAX_MSG_SIZE = MAX_SIZE - 2;


   private boolean ACK = false;
   private boolean RET = false;
   private boolean CHUNKS = false;
   private boolean END_CHUNKS = false;

   public static final byte START_BYTE = (byte) 0xFF;
   public static final byte STOP_BYTE = (byte) 0xFF;


   private Type type;
   private byte[] msg;

   private boolean correct;
   private static CycleCoder cc = new CycleCoder();

   private Frame(Type type, byte[] msg) {
      this.type = type;
      this.msg = msg;
   }

   public static Frame newACKFrame() {
      Frame frame = new Frame(Type.S, new byte[0]);
      frame.setACK(true);
      return frame;
   }

   public static Frame newRETFrame() {
      Frame frame = new Frame(Type.S, new byte[0]);
      frame.setRET(true);
      return frame;
```

```
}

public static Frame newCHUNKEDFrame(byte[] chunk) {
   Frame frame = new Frame(Type.I, chunk);
   frame.setCHUNKS(true);
   return frame;
}

public static Frame newChunkEndFrame(byte[] chunk) {
   Frame frame = new Frame(Type.I, chunk);
   frame.setCHUNKS(true);
   frame.setEND_CHUNKS(true);
   return frame;
}


public Type getType() {
   return type;
}

public byte[] getMsg() {
   return msg;
}

public boolean isACK() {
   return ACK;
}

public boolean isRET() {
   return RET;
}

public boolean isCHUNK() {
   return CHUNKS;
}

public boolean isEND_CHUNKS() {
   return END_CHUNKS;
}


private void setACK(boolean value) {
   ACK = value;
}

private void setRET(boolean value) {
   RET = value;
}

private void setCHUNKS(boolean value) {
   CHUNKS = value;
}
```

```
private void setEND_CHUNKS(boolean value) {
   END_CHUNKS = value;
}


private byte getACK() {
   return (byte) (ACK ? 1 : 0);
}

private byte getRET() {
   return (byte) (RET ? 1 : 0);
}

private byte getCHUNKS() {
   return (byte) (CHUNKS ? 1 : 0);
}

private byte getEND_CHUNKS() {
   return (byte) (END_CHUNKS ? 1 : 0);
}

public boolean isCorrect() {
   return correct;
}

private void setCorrect(boolean correct) {
   this.correct = correct;
}

@Override
public byte[] serialize() {
   byte typeByte = (byte) type.ordinal();

   byte supervisorInfoByte = getACK();
   supervisorInfoByte <<= 1;
   supervisorInfoByte += getRET();
   supervisorInfoByte <<= 1;
   supervisorInfoByte += getCHUNKS();
   supervisorInfoByte <<= 1;
   supervisorInfoByte += getEND_CHUNKS();


   byte[] infoBytes = new byte[2];
   infoBytes[0] = typeByte;
   infoBytes[1] = supervisorInfoByte;

   byte[] frame = ArrayUtils.concatenate(infoBytes, msg);

   byte[] encodedFrame = cc.encode(frame);

   byte[] withStart = ArrayUtils.concatenate(START_BYTE, encodedFrame);
```

```
        return ArrayUtils.concatenate(withStart, STOP_BYTE);
    }

    public static Frame deserialize(byte[] data) {
        boolean correct = true;
        try {
            data = cc.decode(data);
        } catch (DecodeException e) {
            correct = false;
            return null;
        }

        byte typeByte = data[0];
        Type type = null;
        try {
            type = Type.fromInteger(typeByte);
        } catch (Exception ignored) {}

        byte supervisorInfoByte = data[1];
        byte mask = 0x01;
        byte END_CHUNKS = (byte) (supervisorInfoByte & mask);
        supervisorInfoByte >>= 1;
        byte CHUNKS = (byte) (supervisorInfoByte & mask);
        supervisorInfoByte >>= 1;
        byte RET = (byte) (supervisorInfoByte & mask);
        supervisorInfoByte >>= 1;
        byte ACK = (byte) (supervisorInfoByte & mask);

        byte[] msg = Arrays.copyOfRange(data, 2, data.length);

        Frame frame = new Frame(type, msg);
        frame.setACK(ACK != 0);
        frame.setRET(RET != 0);
        frame.setCHUNKS(CHUNKS != 0);
        frame.setEND_CHUNKS(END_CHUNKS != 0);
        frame.setCorrect(true);

        return frame;
    }
}
```

4.5.4. Класс CycleCoder

Реализует табличный алгоритм циклического кодирования с обнаружением ошибок. Таблица соответствует коду [7, 4] с полиномом $x^3 + x + 1$.

4.5.4.1. Поля

- private byte[] map; //raw half byte -> coded byte map
  Карта, отображающая незакодированные 4 бита в закодированные 7

4.5.4.2. Методы

- public CycleCoder()
  Конструктор, инициализирует таблицу *map*
- public byte[] encode(byte[] data)
  Кодирование
- public byte[] decode(byte[] data)
  Декодирование
- private byte mapReverse(byte coded)
  Обратный поиск по отображению *map* (value -> key)
- public static void main(String[] args)
  Тестовая функция

4.5.4.3. Листинг

```
package layers.dll;

import layers.exceptions.DecodeException;

import java.util.Arrays;

public class CycleCoder {
   private byte[] map; //raw half byte -> coded byte map

   public CycleCoder() {
      map = new byte[] {0, 11, 22, 29, 39, 44, 49, 58, 69, 78, 83, 88, 98, 105, 116, 127};
   }

   public byte[] encode(byte[] data) {
      byte[] result = new byte[data.length * 2];

      for (int i = 0; i < data.length; ++i) {
         byte b = data[i];

         byte halfByte1 = (byte)(b & (byte) 0b00001111);
         result[2 * i] = map[halfByte1];

         byte halfByte2 = (byte)((b & 0xFF) >> (byte) 4);
         result[2 * i + 1] = map[halfByte2];
      }

      return result;
   }

   public byte[] decode(byte[] data) throws DecodeException {
      byte[] result = new byte[data.length / 2];

      for (int i = 0; i < data.length - 1; i += 2) {
         byte res = 0;

         res |= mapReverse(data[i]);
         res |= mapReverse(data[i + 1]) << 4; //shift half byte left
```

```
            result[i / 2] = res;
        }

        return result;
    }

    private byte mapReverse(byte coded) throws DecodeException {
        for (int i = 0; i < map.length; ++i) {
            byte b = map[i];

            if (b == coded)
                return (byte) i;
        }

        throw new DecodeException();
    }

    public static void main(String[] args) {
        byte[] in = {10, 20, 30, 127, 0, 1};

        CycleCoder coder = new CycleCoder();

        //should output "equal"
        try {
            byte[] out = coder.decode(coder.encode(in));
            if (Arrays.equals(in, out))
                System.out.println("equal");
            else
                System.out.println("not equal");
        }
        catch (DecodeException e) {
            System.out.println("DecodeException");
        }

        //should output "DecodeException"
        try {
            byte[] coded = coder.encode(in);
            coded[0] = 101;

            byte[] out = coder.decode(coded);
            if (Arrays.equals(in, out))
                System.out.println("equal");
            else
                System.out.println("not equal");
        }
        catch (DecodeException e) {
            System.out.println("DecodeException");
        }
    }
}
```

4.6. Пакет phy

4.6.1. Пакет settings

4.6.1.1. Пакет comport_settings

4.6.1.1.1. Класс ComPortSettings

4.6.1.1.1.1. Методы

- public ComPortSettings(String port, int baudRate, int dataBits, int stopBits, int parity)
  Конструктор
  - *port* – системный идентификатор порта
  - *baudRate* – скорость передачи
  - *dataBits* – количество информационных бит
  - *stopBits* – количество стоповых бит
  - *parity* – способ проверки чётности
- public String getPort()
  Возвращает порт
- public int getBaudRate()
  Возвращает скорость передачи
- public int getDataBits()
  Возвращает количество информационных бит
- public int getStopBits()
  Возвращает стоповых бит
- public int getParity()
  Возвращает способ проверки чётности

4.6.1.1.1.2. Листинг

```
package layers.phy.settings.comport_settings;

import layers.phy.settings.PhysicalLayerSettings;

public class ComPortSettings extends PhysicalLayerSettings {

   public ComPortSettings(String port, int baudRate, int dataBits, int stopBits, int parity) {
      settings.put(ComSettings.PORT_NAME, port);
      settings.put(ComSettings.BAUD_RATE, baudRate);
      settings.put(ComSettings.DATA_BITS, dataBits);
      settings.put(ComSettings.STOP_BITS, stopBits);
      settings.put(ComSettings.PARITY, parity);
   }

   public String getPort() {
      return (String) settings.get(ComSettings.PORT_NAME);
   }

   public int getBaudRate() {
      return (int) settings.get(ComSettings.BAUD_RATE);
   }

   public int getDataBits() {
      return (int) settings.get(ComSettings.DATA_BITS);
   }
```

```
    public int getStopBits() {
        return (int) settings.get(ComSettings.STOP_BITS);
    }

    public int getParity() {
        return (int) settings.get(ComSettings.PARITY);
    }
}
```

4.6.1.1.2. Перечисление ComSettings

4.6.1.1.2.1. Значения

- PORT_NAME
  Имя порта
- BAUD_RATE
  Скорость
- DATA_BITS
  Количество информационных бит
- STOP_BITS
  Количество стоповых бит
- PARITY
  Способ проверки чётности

4.6.1.1.2.2. Методы

Все методы являются унаследованными

4.6.1.1.2.3. Листинг

```
package layers.phy.settings.comport_settings;

import layers.phy.settings.PhySettings;

public enum ComSettings implements PhySettings {
    PORT_NAME,
    BAUD_RATE,
    DATA_BITS,
    STOP_BITS,
    PARITY
}
```

4.6.1.1.3. Перечисление DataBitsEnum

4.6.1.1.3.1. Значения

- DataBits5("5 bits", SerialPort.DATABITS_5),
- DataBits6("6 bits", SerialPort.DATABITS_6),
- DataBits7("7 bits", SerialPort.DATABITS_7),
- DataBits8("8 bits", SerialPort.DATABITS_8);

4.6.1.1.3.2. Поля

- private String name;
  Строковое представление
- private int value;
  Числовое представление

4.6.1.1.3.3. Методы

- private DataBitsEnum(String name, int data_bits)
  Конструктор
    - name – строковое представление
    - data_bits – числовое значение
- public String toString()
  Возвращает строковое представление
- public int toDataBits()
  Возвращает числовое представление
- public static DataBitsEnum getDefault()
  Возвращает значение по умолчанию
- public static DataBitsEnum fromString(String str)
  Создаёт объект по строковому представлению

4.6.1.1.3.4. Листинг

```
package layers.phy.settings.comport_settings;

import gnu.io.SerialPort;

public enum DataBitsEnum {
   DataBits5("5 bits", SerialPort.DATABITS_5),
   DataBits6("6 bits", SerialPort.DATABITS_6),
   DataBits7("7 bits", SerialPort.DATABITS_7),
   DataBits8("8 bits", SerialPort.DATABITS_8);

   private String name;
   private int value;

   private DataBitsEnum(String name, int data_bits) {
      this.name = name;
      this.value = data_bits;
   }

   @Override
   public String toString() {
      return name;
   }

   public int toDataBits() {
      return value;
   }

   public static DataBitsEnum getDefault() {
      return DataBits8;
   }

   public static DataBitsEnum fromString(String str) {
      for (DataBitsEnum e : DataBitsEnum.values()) {
         if (e.name.equals(str))
            return e;
      }
      throw new IllegalArgumentException("Data bits not found");
   }
```

}

4.6.1.1.4. Перечисление StopBitsEnum

4.6.1.1.4.4. Значения

- StopBits1("1", SerialPort.STOPBITS_1),
- StopBits1_5("1.5", SerialPort.STOPBITS_1_5),
- StopBits2("2", SerialPort.STOPBITS_2);

4.6.1.1.4.2. Поля

- private String name;
  Строковое представление
- private int value;
  Числовое представление

4.6.1.1.4.3. Методы

- private StopBitsEnum (String name, int stop_bits)
  Конструктор
  - *name* – строковое представление
  - *stop_bits* – числовое значение
- public String toString()
  Возвращает строковое представление
- public int toStopBits()
  Возвращает числовое представление
- public static StopBitsEnum getDefault()
  Возвращает значение по умолчанию
- public static StopBitsEnum fromString(String str)
  Создаёт объект по строковому представлению

4.6.1.1.4.4. Листинг

```
package layers.phy.settings.comport_settings;

import gnu.io.SerialPort;

public enum StopBitsEnum {
    StopBits1("1", SerialPort.STOPBITS_1),
    StopBits1_5("1.5", SerialPort.STOPBITS_1_5),
    StopBits2("2", SerialPort.STOPBITS_2);

    private String name;
    private int value;

    private StopBitsEnum(String name, int stop_bits) {
        this.name = name;
        this.value = stop_bits;
    }

    @Override
    public String toString() {
        return name;
    }

    public int toStopBits() {
        return value;
```

```
    }

    public static StopBitsEnum getDefault() {
        return StopBits1;
    }

    public static StopBitsEnum fromString(String str) {
        for (StopBitsEnum e : StopBitsEnum.values()) {
            if (e.name.equals(str))
                return e;
        }
        throw new IllegalArgumentException("Stop bits not found");
    }
}
```

4.6.1.1.5. Перечисление ParityEnum

4.6.1.1.5.1. Значения

- Even(SerialPort.PARITY_EVEN),
- Mark(SerialPort.PARITY_MARK),
- Odd(SerialPort.PARITY_ODD),
- Space(SerialPort.PARITY_SPACE),
- None(SerialPort.PARITY_NONE);

4.6.1.1.5.2. Поля

- private int parity;
  Числовое представление

4.6.1.1.5.3. Методы

- private ParityEnum(int parity)
  Конструктор
  - *parity* – числовое значение
- public String toString()
  Возвращает строковое представление
- public int toParity ()
  Возвращает числовое представление
- public static ParityEnum getDefault()
  Возвращает значение по умолчанию
- public static ParityEnum fromString(String str)
  Создаёт объект по строковому представлению

4.6.1.1.5.4. Листинг

```
package layers.phy.settings.comport_settings;

import gnu.io.SerialPort;

public enum ParityEnum {
    Even(SerialPort.PARITY_EVEN),
    Mark(SerialPort.PARITY_MARK),
    Odd(SerialPort.PARITY_ODD),
    Space(SerialPort.PARITY_SPACE),
    None(SerialPort.PARITY_NONE);

    private int parity;
```

```
    private ParityEnum(int parity) {
        this.parity = parity;
    }

    public int toParity() {
        return this.parity;
    }

    public static ParityEnum getDefault() {
        return ParityEnum.None;
    }

    public static ParityEnum fromString(String str) {
        return valueOf(str);
    }
}
```

4.6.1.2. Интерфейс PhySettings

4.6.1.2.1. Методы

- String name();
  Преобразует идентификатор перечисления в строку

4.6.1.2.2. Листинг

```
package layers.phy.settings;

public interface PhySettings {
    String name(); // will be implemented by java.lang.Enum class
}
```

4.6.1.3. Класс PhysicalLayerSettings

4.6.1.3.1. Поля

- protected Map<PhySettings, Object> settings
  Отображение (параметр, значение)

4.6.1.3.2. Листинг

```
package layers.phy.settings;

import java.util.HashMap;
import java.util.Map;

public abstract class PhysicalLayerSettings {
    protected Map<PhySettings, Object> settings = new HashMap<>();

}
```

4.6.2. Интерфейс IPhysicalLayer

4.6.2.1. Методы

- boolean readyToSend();
  Возвращает значение RTS
- void send(byte[] data);
  Передаёт массив байт через ком-порт
  - data – массив к отправке
- void subscribeConnectionStatusChanged(Consumer<Boolean> listener);
  Подписка на событие изменение состояния связи
  - listener - обработчик
- void subscribeCompanionConnectedChanged(Consumer<Boolean> listener);
  Подписка на событие подключения/отключения оппонента
  - listener - обработчик
- void subscribeSendingAvailableChanged(Consumer<Boolean> listener);
  Подписка на CTS
  - listener – обработчик

4.6.2.2. Листинг

```
package layers.phy;

import layers.ILayer;

import java.util.function.Consumer;

public interface IPhysicalLayer extends ILayer {
   boolean readyToSend();
   void send(byte[] data);
   void subscribeConnectionStatusChanged(Consumer<Boolean> listener);
   void subscribeCompanionConnectedChanged(Consumer<Boolean> listener);
   void subscribeSendingAvailableChanged(Consumer<Boolean> listener);
}
```

4.6.3. Класс ComPort

4.6.3.1. Поля

- private static Logger LOGGER = Logger.getLogger("PhysicalLayerLogger");
  Объект для вывода отладочных сообщений в консоль
- private static List<String> availablePorts;
  Список доступных комп-портов
- private static final String PORT_NAME = "ChatPort";
  Уникальный строковый идентификатор порта
- private static final int TIME_OUT = 2000;
  Таймаут открытия порта
- private IDataLinkLayer dataLinkLayer;
  Ссылка на верхний уровень
- private SerialPort serialPort;
  Объект библиотеки RXTX, последовательный порт
- private OutputStream outStream;
  Ссылка на выходной поток порта
- private InputStream inStream;
  Ссылка на входной поток порта

- private boolean connected;
  Факт наличия соединения
- private List<Consumer<Boolean>> connectionChangedListeners;
  Список подписок на изменение состояние связи
- private List<Consumer<Boolean>> companionConnectedListeners;
  Список подписок на подключение/отключение оппонента
- private List<Consumer<Boolean>> sendingAvailableChangedListeners;
  Список подписок на изменение CTS

4.6.3.2. Методы

- private void setConnected(boolean status)
  Изменяет значение переменной connected
  - status – значения для установки
- public static List<String> getAvailablePorts(boolean ignoreCache)
  Возвращает список доступных ком-портов
  - ignoreCache – учитывать или нет ранее полученные значения
- public static List<String> getAvailablePorts()
  Возвращает список доступных ком-портов с учётом кэша
- public static List<Integer> getAvailableBaudRates()
  Возвращает список доступных скоростей
- public static List<String> getAvailableDataBits()
  Возвращает список доступных количеств информационных бит
- public static List<String> getAvailableStopBits()
  Возвращает список доступных количеств стоповых бит
- public static List<String> getAvailableParity()
  Возвращает список доступных вариантов проверки чётности
- public static String getDefaultPort()
  Возвращает ком-порт по умолчанию или пустую строку, если их список пуст
- public static int getDefaultBaudRate()
  Возвращает скорость по умолчанию
- public static String getDefaultDataBits()
  Возвращает количество информационных бит по умолчанию
- public static String getDefaultStopBits()
  Возвращает количество стоповых бит по умолчанию
- public static String getDefaultParity()
  Возвращает способ проверки чётности по умолчанию
- private void connect(ComPortSettings settings)
  Выполняет соединение
  - settings – настройки соединения
- public synchronized void send(byte[] data)
  Отправляет массив байт по ком-порту
  - data – массив для передачи
- public boolean readyToSend()
  Возвращает значение RTS
- private void notifyConnectionStatusChanged(boolean status)
  Оповещает верхний уровень об изменении состояния соединения
  - status – наличие соединения
- private void notifyCompanionConnectedChanged(boolean status)
  Оповещает верхний уровень об подключении/отключении оппонента
  - status – наличие оппонента

- private void notifySendingAvailableChanged(boolean status)
  Оповещает верхний уровень об изменении состояния линии CTS
  - status – наличие CTS

Остальные методы являются унаследованными.

4.6.3.3. Листинг

```
package layers.phy;

import gnu.io.*;
import layers.ILayer;
import layers.ProtocolStack;
import layers.apl.ApplicationLayer;
import layers.dll.DataLinkLayer;
import layers.dll.IDataLinkLayer;
import layers.exceptions.ConnectionException;
import layers.phy.settings.PhysicalLayerSettings;
import layers.phy.settings.comport_settings.ComPortSettings;
import layers.phy.settings.comport_settings.DataBitsEnum;
import layers.phy.settings.comport_settings.ParityEnum;
import layers.phy.settings.comport_settings.StopBitsEnum;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.nio.ByteBuffer;
import java.util.Enumeration;
import java.util.LinkedList;
import java.util.List;
import java.util.TooManyListenersException;
import java.util.function.Consumer;
import java.util.logging.Logger;

import static java.util.Arrays.asList;
import static java.util.Arrays.stream;

public class ComPort implements IPhysicalLayer, SerialPortEventListener {
    private static Logger LOGGER = Logger.getLogger("PhysicalLayerLogger");
    private static List<String> availablePorts;
    private static final String PORT_NAME = "ChatPort";
    private static final int TIME_OUT = 2000;

    private IDataLinkLayer dataLinkLayer;

    private SerialPort serialPort;
    private OutputStream outStream;
    private InputStream inStream;
    private boolean connected = false;

    private List<Consumer<Boolean>> connectionChangedListeners = new LinkedList<>();
    private List<Consumer<Boolean>> companionConnectedListeners = new LinkedList<>();
```

```
    private List<Consumer<Boolean>> sendingAvailableChangedListeners = new
LinkedList<>();

    @Override
    public boolean isConnected() {
        return connected;
    }

    private void setConnected(boolean status) {
        this.connected = status;
        notifyConnectionStatusChanged(status);
    }

    public static List<String> getAvailablePorts(boolean ignoreCache) {
        if (!ignoreCache && availablePorts != null)
            return availablePorts;

        availablePorts = new LinkedList<>();

        Enumeration portEnum = CommPortIdentifier.getPortIdentifiers();

        while (portEnum.hasMoreElements()) {
            CommPortIdentifier port = (CommPortIdentifier) portEnum.nextElement();

            if(port.getPortType() == CommPortIdentifier.PORT_SERIAL)
                availablePorts.add(port.getName());
        }

        return availablePorts;
    }

    public static List<String> getAvailablePorts() {
        return getAvailablePorts(false);
    }

    public static List<Integer> getAvailableBaudRates() {
        return asList(300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600,
115200);
    }

    public static List<String> getAvailableDataBits() {
        List<String> names = new LinkedList<>();
        stream(DataBitsEnum.values()).forEach(e -> names.add(e.toString()));
        return names;
    }

    public static List<String> getAvailableStopBits() {
        List<String> names = new LinkedList<>();
        stream(StopBitsEnum.values()).forEach(e -> names.add(e.toString()));
        return names;
    }
```

```
    public static List<String> getAvailableParity() {
        List<String> names = new LinkedList<>();
        stream(ParityEnum.values()).forEach(e -> names.add(e.toString()));
        return names;
    }

    public static String getDefaultPort() {
        String port = "";
        List<String> ports = getAvailablePorts();
        if (!ports.isEmpty())
            port = ports.get(0);

        return port;
    }

    public static int getDefaultBaudRate() {
        return 9600;
    }

    public static String getDefaultDataBits() {
        return DataBitsEnum.getDefault().toString();
    }

    public static String getDefaultStopBits() {
        return StopBitsEnum.getDefault().toString();
    }

    public static String getDefaultParity() {
        return ParityEnum.getDefault().toString();
    }

    @Override
    public void connect(PhysicalLayerSettings settings) throws ConnectionException {
        try {
            connect((ComPortSettings) settings);
        } catch (NoSuchPortException | UnsupportedCommOperationException |
PortInUseException e) {
            throw new ConnectionException(e);
        }
    }

    private void connect(ComPortSettings settings) throws NoSuchPortException,
UnsupportedCommOperationException, PortInUseException {
        String port = settings.getPort();

        LOGGER.info("Connecting to port " + port);
        CommPortIdentifier portId = CommPortIdentifier.getPortIdentifier(port);

        try {
            serialPort = (SerialPort) portId.open(PORT_NAME, TIME_OUT);
            LOGGER.info("Port " + port + " opened successfully");
```

```
            serialPort.setSerialPortParams(settings.getBaudRate(), settings.getDataBits(),
settings.getStopBits(), settings.getParity());
            serialPort.setFlowControlMode(SerialPort.FLOWCONTROL_NONE);

            outStream = serialPort.getOutputStream();
            inStream = serialPort.getInputStream();

        } catch (PortInUseException e) {
            LOGGER.severe("Port " + port + " is already in use");
            throw e;
        } catch (UnsupportedCommOperationException e) {
            LOGGER.severe("Unsupported com port params");
            disconnect();
            throw e;
        } catch (IOException e) {
            LOGGER.severe("Error while opening streams for serial port");
            disconnect();
        }

        try {
            serialPort.addEventListener(this);
            serialPort.notifyOnDataAvailable(true);
            serialPort.notifyOnCTS(true);
            serialPort.notifyOnDSR(true);
        } catch (TooManyListenersException e) {
            LOGGER.severe("Too many listeners");
            disconnect();
        }

        serialPort.setRTS(true);
        serialPort.setDTR(true);
        setConnected(true);
        notifyCompanionConnectedChanged(serialPort.isDSR());
    }

    @Override
    public synchronized void disconnect() {
        if (serialPort != null) {
            try {
                outStream.close();
                inStream.close();
            } catch (IOException ignored) {
            }

            serialPort.setRTS(false);
            serialPort.setDTR(false);

            serialPort.close();

            LOGGER.info("Port " + serialPort.getName() + " closed");
            outStream = null;
            inStream = null;
```

```
            serialPort = null;
        }
        else {
            LOGGER.info("Port is not opened");
        }

        notifySendingAvailableChanged(false);
        //notifyCcompanionConnectedChanged(false); //do not notify me, when i press disconnect
by myself
        setConnected(false);
    }

    @Override
    public synchronized void send(byte[] data) {
//        System.out.println("ready? - " + readyToSend());

        serialPort.setRTS(false);
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        try {
            outStream.write(data);
        } catch (IOException e) {
            LOGGER.warning("Exception occurred: " + e.getMessage());
            notifyOnError(e);
            return;
        }
        try {
            outStream.flush();
        } catch (IOException ignored) { }
        serialPort.setRTS(true);
    }

    @Override
    public boolean readyToSend() {
        return serialPort.isDSR() && serialPort.isCTS();
    }

    @Override
    public IDataLinkLayer getUpperLayer() {
        return dataLinkLayer;
    }

    @Override
    public ILayer getLowerLayer() {
        return null;
    }

    @Override
    public void setUpperLayer(ILayer layer) {
```

```
    dataLinkLayer = (IDataLinkLayer) layer;
}

@Override
public void setLowerLayer(ILayer layer) {

}

@Override
public void subscribeConnectionStatusChanged(Consumer<Boolean> listener) {
    connectionChangedListeners.add(listener);
}

@Override
public void subscribeCompanionConnectedChanged(Consumer<Boolean> listener) {
    companionConnectedListeners.add(listener);
}

@Override
public void subscribeSendingAvailableChanged(Consumer<Boolean> listener) {
    sendingAvailableChangedListeners.add(listener);
}

private void notifyConnectionStatusChanged(boolean status) {
    connectionChangedListeners.forEach(listener -> listener.accept(status));
}

private void notifyCompanionConnectedChanged(boolean status) {
    companionConnectedListeners.forEach(listener -> listener.accept(status));
}

private void notifySendingAvailableChanged(boolean status) {
    sendingAvailableChangedListeners.forEach(listener -> listener.accept(status));
}

public void notifyOnError(Exception e) {
    onErrorListeners.forEach(listener -> listener.accept(e));
}


@Override
public synchronized void serialEvent(SerialPortEvent event) {
    switch (event.getEventType()) {
        case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
            System.out.println("OUTPUT_BUFFER_EMPTY");
            break;

        case SerialPortEvent.DATA_AVAILABLE:
            dataAvailable(event);
            break;

        case SerialPortEvent.BI:
```

```
            System.out.println("BI");
            break;

        case SerialPortEvent.CD:
            System.out.println("CD");
            break;

        case SerialPortEvent.CTS:
            System.out.println("CTS = [" + serialPort.isCTS() + "]");
            notifySendingAvailableChanged(serialPort.isCTS());
            break;

        case SerialPortEvent.DSR:
            System.out.println("DSR");
//              if (!serialPort.isDSR()) {
//                  setConnected(false);
//              }
            notifyCcompanionConnectedChanged(serialPort.isDSR());
            break;

        case SerialPortEvent.FE:
            System.out.println("FE");
            break;

        case SerialPortEvent.OE:
            System.out.println("OE");
            break;

        case SerialPortEvent.PE:
            System.out.println("PE");
            break;
        case SerialPortEvent.RI:
            System.out.println("RI");
            break;
      }
    }

    public void dataAvailable(SerialPortEvent event) {
      // reading size of the data
      try {
        byte[] dataSize = new byte[2];
        for (int i = 0; i < dataSize.length; ++i) {
          dataSize[i] = (byte) inStream.read();
        }

        // reading the whole data
        int size = (int) ByteBuffer.wrap(dataSize).getShort();
        byte[] data = new byte[size];
        for (int i = 0; i < size; ++i) {
          data[i] = (byte) inStream.read();
        }
```

```
            getUpperLayer().receive(data);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

4.7. Пакет exceptions

4.7.4. Класс ChatException

Корневое в иерархии приложения исключение, от него наследуются все остальные.

4.7.4.1. Методы

- public ChatException(String message)
  Коструктор
  - message – текстовое представление исключения
- public ChatException(String message, Throwable cause)
  Конструктор
  - message – текстовое представление исключения
  - cause – причина исключения
- public ChatException(Throwable cause)
  Конструктор
  - cause – причина исключения

4.7.4.2. Листинг

```
package layers.exceptions;

public class ChatException extends Exception {
    public ChatException(String message) {
        super(message);
    }

    public ChatException(String message, Throwable cause) {
        super(message, cause);
    }

    public ChatException(Throwable cause) {
        super(cause);
    }

}
```

4.7.2. Класс ConnectionException

4.7.2.1. Методы

Все методы полностью аналогичны методам предка

4.7.2.2. Листинг

```
package layers.exceptions;
```

```
public class ConnectionException extends ChatException {
    public ConnectionException(String message) {
        super(message);
    }

    public ConnectionException(String message, Throwable cause) {
        super(message, cause);
    }

    public ConnectionException(Throwable cause) {
        super(cause);
    }
}
```

4.7.3. Класс LayerUnavailableException

4.7.3.1. Методы

Все методы полностью аналогичны методам предка

4.7.3.2. Листинг

```
package layers.exceptions;

public class LayerUnavailableException extends ChatException {
    public LayerUnavailableException(String message) {
        super(message);
    }

    public LayerUnavailableException(String message, Throwable cause) {
        super(message, cause);
    }

    public LayerUnavailableException(Throwable cause) {
        super(cause);
    }
}
```

4.7.4. Класс ProtocolStackException

4.7.4.1. Методы

Все методы полностью аналогичны методам предка

4.7.4.2. Листинг

```
package layers.exceptions;

public class ProtocolStackException extends ChatException {
    public ProtocolStackException(Exception cause) {
        super(cause);
    }
}
```

4.7.5. Класс UnexpectedChatException

4.7.5.1. Методы

Все методы полностью аналогичны методам предка

4.7.5.2. Листинг

```
package layers.exceptions;

public class UnexpectedChatException extends ChatException {
   public UnexpectedChatException(String message) {
      super(message);
   }

   public UnexpectedChatException(String message, Throwable cause) {
      super(message, cause);
   }

   public UnexpectedChatException(Throwable cause) {
      super(cause);
   }
}
```

4.7.6. Класс DecodeException

4.7.6.1. Методы

Все методы полностью аналогичны методам предка

4.7.6.2. Листинг

```
package layers.exceptions;

public class DecodeException extends ChatException {
   public DecodeException() {
      super();
   }
}
```

5. Пакет util

5.1. Класс ArrayUtils

5.1.1. Методы

- public static byte[] concatenate(byte[] A, byte[] B)
  Объединение двух массивов байт
  - A, B – объединяемые операнды
- public static byte[] concatenate(byte a, byte[] B)
  Объединение байта и массива байт
  - a, B – объединяемые операнды

- public static byte[] concatenate(byte[] A, byte b)
  Объединение массива и байта
  - A, b – объединяемые операнды

5.1.2. Листинг

```
package util;

import java.lang.reflect.Array;

public class ArrayUtils {
   public static byte[] concatenate(byte[] A, byte[] B) {
      int aLen = A.length;
      int bLen = B.length;

      byte[] C = (byte[]) Array.newInstance(A.getClass().getComponentType(), aLen + bLen);
      System.arraycopy(A, 0, C, 0, aLen);
      System.arraycopy(B, 0, C, aLen, bLen);

      return C;
   }

   public static byte[] concatenate(byte a, byte[] B) {
      byte[] A = new byte[1];
      A[0] = a;
      return concatenate(A, B);
   }

   public static byte[] concatenate(byte[] A, byte b) {
      byte[] B = new byte[1];
      B[0] = b;
      return concatenate(A, B);
   }
}
```