Main.java

```
package org.example;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;
import java.io.IOException;
import java.net.URL;

public class Main extends Application {
    @Override
    public void start(Stage stage) {
        try {
            FXMLLoader loader = new FXMLLoader(); //Создаем загрузчик
разметки
            URL fxmlUrl = getClass().getResource("/App.fxml"); //Создаем
ссылку на разметку
            loader.setLocation(fxmlUrl);
            Parent root = loader.load(); //Загружаем файл разметки окна
            stage.setScene(new Scene(root)); //Устанавливаем загруженную
разметку на окно
            stage.setResizable(false); //Убираем возможноть менять размер
окна
            stage.setTitle("MemWatcher"); //Ставит заголовок окна
            stage.show(); //Показываем окно
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

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

MemoryBlock.java

```
package org.example.model;

//Класс, содержащий информацию о блоке памяти
public class MemoryBlock {
    public String address;
    public long kbytes;
    public long rss;
    public long dirty;
    public String mode;
    public String mapping;

    public MemoryBlock(String address, long kbytes, long rss, long dirty,
String mode, String mapping) {
```

```
        this.address = address;
        this.kbytes = kbytes;
        this.rss = rss;
        this.dirty = dirty;
        this.mode = mode;
        this.mapping = mapping;
    }

    public String getAddress() {
        return address;
    }

    public long getKbytes() {
        return kbytes;
    }

    public long getRss() {
        return rss;
    }

    public long getDirty() {
        return dirty;
    }

    public String getMode() {
        return mode;
    }

    public String getMapping() {
        return mapping;
    }
}
```

SystemProcessesUtils.java

```
package org.example.model;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.*;

public class SystemProcessesUtils {
    //Метод получения списка всех процессов в системе
    public static Map<Integer, String> getAvailableProcesses() throws
IOException {
        //Вызываем команду для получения информации о процессах в системе
        ProcessBuilder pb = new ProcessBuilder("ps", "-A");
        Map<Integer, String> result = new HashMap<>();
        //Обработка результата
        try (BufferedReader br = new BufferedReader(new
```

```
InputStreamReader(pb.start().getInputStream())) { //Построчно парсим
результат команды
            String line;
            br.readLine(); //Пропускаем строку с заголовком
            while ((line = br.readLine()) != null) {
                String[] proc = line.trim().replaceAll("( )+", "
").split(" ");
                if (proc[3] == null)
                    continue;
                result.put(Integer.valueOf(proc[0]), proc[3]);
            }
        }

        return result;
    }

    //Метод получения используемой процессом памяти
    public static List<MemoryBlock> getProcessInfo(int pid, String
startAddress, String endAddress) throws IOException {
        //Собираем строчку для фильтрации по адресу
        String addressRange =
                (startAddress != null ? startAddress : "") +
                '-' +
                (endAddress != null ? endAddress : "");

        //Вызываем команду для получения информации о использованной
памяти
        ProcessBuilder pb = new ProcessBuilder("pmap", "-A" ,
addressRange, "-q" ,"-x", String.valueOf(pid));
        List<MemoryBlock> result = new ArrayList<>();

        //Обработка результата
        try (BufferedReader br = new BufferedReader(new
InputStreamReader(pb.start().getInputStream()))) {
            String line;
            br.readLine(); //Пропускаем строку с заголовками
            while ((line = br.readLine()) != null) { //Построчно парсим
результат команды
                String[] proc = line.trim().replaceAll("( )+", "
").split(" ");
                String mapping = proc.length > 6 ? "[" + proc[6] + "]" :
proc[5];
                result.add(new MemoryBlock(
                        proc[0],
                        Long.parseLong(proc[1]),
                        Long.parseLong(proc[2]),
                        Long.parseLong(proc[3]),
                        proc[4],
                        mapping
                ));
            }
        }
```

```
        return result;
    }
}
```

App.fxml – Разметка окна приложения

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

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.ChoiceBox?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.Menu?>
<?import javafx.scene.control.MenuBar?>
<?import javafx.scene.control.MenuItem?>
<?import javafx.scene.control.Separator?>
<?import javafx.scene.control.TableColumn?>
<?import javafx.scene.control.TableView?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.control.Tooltip?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.VBox?>

<VBox prefHeight="720.0" prefWidth="1026.0"
xmlns="http://javafx.com/javafx/19" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="org.example.controllers.AppController">
  <children>
    <MenuBar VBox.vgrow="NEVER">
      <menus>
        <Menu mnemonicParsing="false" text="File">
          <items>
            <MenuItem mnemonicParsing="false" onAction="#updateProcesses"
text="Update processes" />
                  <MenuItem mnemonicParsing="false" onAction="#exit"
text="Exit" />
          </items>
        </Menu>
            <Menu mnemonicParsing="false" text="Help">
               <items>
                  <MenuItem mnemonicParsing="false"
onAction="#aboutProgram" text="About MemWatcher" />
               </items>
            </Menu>
      </menus>
    </MenuBar>
      <HBox alignment="CENTER_LEFT" VBox.vgrow="NEVER">
         <children>
            <ChoiceBox fx:id="choiceBox" prefHeight="26.0"
prefWidth="264.0">
               <tooltip>
                  <Tooltip text="Choose process to see memory usage" />
               </tooltip></ChoiceBox>
```

```
            <Separator prefHeight="0.0" prefWidth="156.0" visible="false"
HBox.hgrow="ALWAYS" />
            <Label prefHeight="18.0" prefWidth="113.0" text="Minimal
address:" HBox.hgrow="ALWAYS">
               <HBox.margin>
                  <Insets right="5.0" />
               </HBox.margin>
            </Label>
            <TextField fx:id="minAddress" onKeyTyped="#minAddressChanged">
               <tooltip>
                  <Tooltip text="Enter 64-bit minimal address for lookup.
For example &amp;quot;00007fafd66b0000&amp;quot;" wrapText="true" />
               </tooltip></TextField>
            <Separator prefHeight="2.0" prefWidth="35.0" visible="false"
/>
            <Label layoutX="160.0" layoutY="14.0" prefHeight="18.0"
prefWidth="111.0" text="Maximal address:">
               <HBox.margin>
                  <Insets right="5.0" />
               </HBox.margin>
            </Label>
            <TextField fx:id="maxAddress" layoutX="264.0" layoutY="10.0"
onKeyTyped="#maxAddressChanged">
               <tooltip>
                  <Tooltip text="Enter 64-bit maximal address for lookup.
For example &amp;quot;ffffffffff600000&amp;quot;" />
               </tooltip></TextField>
         </children>
         <VBox.margin>
            <Insets top="5.0" />
         </VBox.margin>
         <padding>
            <Insets left="5.0" right="5.0" />
         </padding>
      </HBox>
      <TableView fx:id="table" editable="true" prefHeight="200.0"
prefWidth="200.0" VBox.vgrow="ALWAYS">
          <placeholder>
              <Label text="No such memory blocks" />
          </placeholder>
        <columns>
          <TableColumn fx:id="addressCol" prefWidth="150.0">
               <graphic>
                  <Label text="Address">
                     <tooltip>
                        <Tooltip text="The beginning memory address
allocation" />
                     </tooltip>
                  </Label>
               </graphic></TableColumn>
          <TableColumn fx:id="kbytesCol" minWidth="0.0" prefWidth="85.0">
               <graphic>
                  <Label text="KBytes">
```

```
                    <tooltip>
                        <Tooltip text="Memory allocation in kilobytes" />
                    </tooltip>
                </Label>
            </graphic></TableColumn>
            <TableColumn fx:id="modeCol" prefWidth="140.0">
                <graphic>
                    <Label text="Mode">
                        <tooltip>
                            <Tooltip text="Access mode and privileges" />
                        </tooltip>
                    </Label>
                </graphic></TableColumn>
            <TableColumn fx:id="mappingCol" prefWidth="886.0">
                <graphic>
                    <Label text="Mapping">
                        <tooltip>
                            <Tooltip text="The user-facing name of the
application or library" />
                        </tooltip>
                    </Label>
                </graphic></TableColumn>
        </columns>
         <padding>
            <Insets left="5.0" right="5.0" />
         </padding>
         <VBox.margin>
            <Insets top="5.0" />
         </VBox.margin>
      </TableView>
  </children>
</VBox>
```

AppController.java

```
package org.example.controllers;

import javafx.application.Platform;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.util.StringConverter;
import org.example.model.MemoryBlock;
import org.example.model.SystemProcessesUtils;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
```

```java
import java.util.Optional;

public class AppController {

    //Маппинг свойств из разметки (App.fxml)
    @FXML
    public ChoiceBox<Map.Entry<Integer, String>> choiceBox;

    @FXML
    public TextField minAddress;

    @FXML
    public TextField maxAddress;

    @FXML
    TableView<MemoryBlock> table;

    @FXML
    TableColumn<MemoryBlock, String> addressCol;

    @FXML
    TableColumn<MemoryBlock, String> kbytesCol;

    @FXML
    TableColumn<MemoryBlock, String> modeCol;

    @FXML
    TableColumn<MemoryBlock, String> mappingCol;


    //Инициализация контроллера
    @FXML
    public void initialize() {
        choiceBox //Устанавливаем обработчик выбора процесса из списка
                .getSelectionModel()
                .selectedItemProperty()
                .addListener((observable, oldValue, newValue) -> {
                    if (newValue != null) {
                        updateTable(newValue.getKey());
                        return;
                    }
                    updateTable(choiceBox.getItems().get(0).getKey());
                });
        choiceBox.setConverter(new StringConverter<>() { //Конвертер для
отображения элементов в выпадающем списке
            @Override
            public String toString(Map.Entry<Integer, String> object) {
                if (object == null)
                    return "";
                return object.getValue();
            }
            @Override
            public Map.Entry<Integer, String> fromString(String string) {
```

```
                return choiceBox.getItems().get(0);
            }
        });

        //Обработчики для генерации текста в клетках таблицы
        addressCol.setCellValueFactory(new
PropertyValueFactory<>("address"));
        kbytesCol.setCellValueFactory(new
PropertyValueFactory<>("kbytes"));
        modeCol.setCellValueFactory(new PropertyValueFactory<>("mode"));
        mappingCol.setCellValueFactory(new
PropertyValueFactory<>("mapping"));

        updateProcesses(); //Загружаем таблицу с данными о процессе
    }

    //Маппинг обработчиков из разметки (App.fxml)
    @FXML
    public void minAddressChanged() {
        updateTable(choiceBox.getValue().getKey());
    }

    @FXML
    public void maxAddressChanged() {
        updateTable(choiceBox.getValue().getKey());
    }

    @FXML
    public void updateProcesses() {
        List<Map.Entry<Integer, String>> vals;
        try {
            vals = new
ArrayList<>(SystemProcessesUtils.getAvailableProcesses().entrySet());
//Получаем список процессов системы
            choiceBox.setItems(FXCollections.observableArrayList(vals));
//Устанавливаем список в выпадающее меню
            choiceBox.setValue(vals.get(0)); //Устанавливаем начальное
значение в списке
        } catch (IOException e) {
            showError(e);
            e.printStackTrace();
        }
    }

    @FXML
    public void aboutProgram() {
        new AboutController(table.getScene().getWindow()).show();
//Отображаем окно с информацией о программе
    }

    @FXML
    public void exit() {
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
```

```
        alert.setTitle("Exit Dialog");
        alert.setHeaderText(null);
        alert.setContentText("Do you really want to exit the
MemWatcher?");

        Optional<ButtonType> res = alert.showAndWait();

        if (res.isPresent() && res.get() == ButtonType.OK)
            Platform.exit();
    }

    private void updateTable(int pid) {
        try {
            //Получаем данные об используемой процессом памяти
            List<MemoryBlock> data =
SystemProcessesUtils.getProcessInfo(pid, minAddress.getText(),
maxAddress.getText());
            //устанавливаем полученные данные в таблицу
            table.setItems(FXCollections.observableArrayList(data));
        } catch (IOException e) {
            showError(e);
            e.printStackTrace();
        }
    }

    //Обработка отображния об ошибке
    private void showError(Throwable t) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Error Dialog");
        alert.setHeaderText(null);
        alert.setContentText(t.getMessage());
        System.err.println("error");
        alert.showAndWait();
    }
}
```

About.fxml – Разметка окошка «О приложении»

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

<?import javafx.scene.control.ButtonType?>
<?import javafx.scene.control.DialogPane?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.image.Image?>
<?import javafx.scene.image.ImageView?>
<?import javafx.scene.layout.VBox?>


<DialogPane onMouseClicked="#closeDialog" prefHeight="233.0"
prefWidth="300.0" xmlns="http://javafx.com/javafx/19"
xmlns:fx="http://javafx.com/fxml/1">
```

```
   <content>
      <VBox alignment="TOP_CENTER" maxHeight="-Infinity" maxWidth="-
Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="184.0"
prefWidth="300.0">
         <children>
            <ImageView fitHeight="65.0" fitWidth="73.0"
pickOnBounds="true" preserveRatio="true">
               <image>
                  <Image url="@index.png" />
               </image>
            </ImageView>
            <Label prefHeight="49.0" prefWidth="278.0"
text="&quot;MemWacher&quot; allows you to lookup all using memory on any
linux systems." textAlignment="CENTER" wrapText="true" />
            <Label text=" This program is made by Maxim Varlamov." />
            <Label text="© &quot;MemWatcher&quot; 2023-2023" />
         </children>
      </VBox>
   </content>
   <buttonTypes>
      <ButtonType fx:constant="CANCEL" />
   </buttonTypes>
</DialogPane>
```

```
package org.example.controllers;

import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.control.Dialog;
import javafx.scene.control.DialogPane;
import javafx.stage.Modality;
import javafx.stage.Window;

import java.io.IOException;

public class AboutController  extends Dialog<Void> {
    public AboutController(Window window) {
        FXMLLoader loader = new FXMLLoader(); //Загрузчик разметки
        loader.setLocation(getClass().getResource("/About.fxml"));
        loader.setController(this);

        try {
            DialogPane pane = loader.load();
            initOwner(window); //Устанавливаем родительский компонент
            initModality(Modality.APPLICATION_MODAL); //Устанавливаем
режим диалогового окна
            setResizable(true);
            setTitle("About MemWatcher");
            setDialogPane(pane);
            getDialogPane()
                    .getScene()
```

```
                    .getWindow()
                    .setOnCloseRequest(event -> closeDialog());
//Обрабатываем событие закрытия окна
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    //Обработчик закрытия окна
    @FXML
    public void closeDialog() {
        getDialogPane().getScene().getWindow().hide();
    }
}
```