Лабораторная работа №7

РАЗРАБОТКА JAVA-ПРИЛОЖЕНИЙ (ЧАСТЬ 2)

Цель работы:

Стилизация приложения с помощью CSS. Добавление иконки приложения.

Стиль, используемый в JavaFX по умолчанию хранится в файле **modena.css**. Его можно найти в файле **jfxrt.jar** (**/jre/lib/ext/jfxrt.jar**). Разархивируйте его и найдите **modena.css** в папке **com/sun/javafx/scene/control/skin/modena**. Этот стиль всегда применяется по умолчанию для всех приложений JavaFX. Добавляя пользовательские стили мы переопределяем стили из файла **modena.css**.

1. Подключение пользовательских CSS-стилей

Имя	Цвет	Код	RGB	HSL	Описание
white		#ffffff или #fff	rgb(255,255,255)	hsl(0,0%,100%)	Белый
silver		#c0c0c0	rgb(192,192,192)	hsl(0,0%,75%)	Серый
gray		#808080	rgb(128,128,128)	hsl(0,0%,50%)	Темно-серый
black		#000000 или #000	rgb(0,0,0)	hsl(0,0%,0%)	Черный
maroon		#800000	rgb(128,0,0)	hsl(0,100%,25%)	Темно-красный
red		#ff0000 или #f00	rgb(255,0,0)	hsl(0,100%,50%)	Красный
orange		#ffa500	rgb(255,165,0)	hsl(38.8,100%,50%)	Оранжевый
yellow		#ffff00 или #ff0	rgb(255,255,0)	hsl(60,100%,50%)	Желтый
olive		#808000	rgb(128,128,0)	hsl(60,100%,25%)	Оливковый
lime		#00ff00 или #0f0	rgb(0,255,0)	hsl(120,100%,50%)	Светло-зеленый
green		#008000	rgb(0,128,0)	hsl(120,100%,25%)	Зеленый
aqua		#00ffff или #0ff	rgb(0,255,255)	hsl(180,100%,50%)	Голубой
blue		#0000ff или #00f	rgb(0,0,255)	hsl(240,100%,50%)	Синий
navy		#000080	rgb(0,0,128)	hsl(240,100%,25%)	Темно-синий
teal		#008080	rgb(0,128,128)	hsl(180,100%,25%)	Сине-зеленый
fuchsia		#ff00ff или #f0f	rgb(255,0,255)	hsl(300,100%,50%)	Розовый
purple		#800080	rgb(128,0,128)	hsl(300,100%,25%)	Фиолетовый

- Добавьте файл **DarkTheme.css** в пакет **view**.

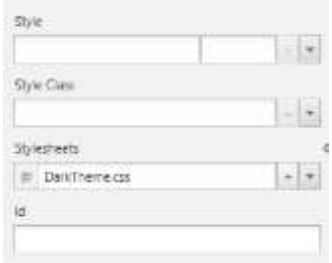
```
.background {
    -fx-background-color: #1d1d1d; //цвет фона элемента (в 16-ричной системе)
}
//стиль для компонента label
.label {
    -fx-font-size: 11pt;  // размер шрифта
    -fx-font-family: "Segoe UI Semibold";  //имя шрифта
    -fx-text-fill: white;  //цвет фонта
    -fx-opacity: 0.6; //уровень прозрачности элемента 0.0-1.0
}

.label-bright {
    -fx-font-size: 11pt;
    -fx-font-family: "Segoe UI Semibold";
    -fx-text-fill: white;
    -fx-opacity: 1;
}

.label-header {
    -fx-font-size: 32pt;
```

```
    -fx-font-family: "Segoe UI Light";
    -fx-text-fill: white;
    -fx-opacity: 1;
}
//стиль для компонента tableview
.table-view {
    -fx-base: #1d1d1d; //основной цвет
    -fx-control-inner-background: #1d1d1d; //цвет внутренней части слайдера
    -fx-background-color: #1d1d1d;
    -fx-table-cell-border-color: transparent; //цвет рамки таблицы: прозрачный
    -fx-table-header-border-color: transparent; //цвет заголовка таблицы: прозрачный
    -fx-padding: 5; //отступ для всего компонента
}

.table-view .column-header-background { //заголовок столбца
    -fx-background-color: transparent;
}

.table-view .column-header, .table-view .filler {
    -fx-size: 35;
    -fx-border-width: 0 0 1 0;
    -fx-background-color: transparent;
    -fx-border-color: //поочередно устанавливается цвет верхней, правой, нижней и левой
границы
        transparent
        transparent
        derive(-fx-base, 80%)
        transparent;
    -fx-border-insets: 0 10 1 0; // вставка границ
}

.table-view .column-header .label {
    -fx-font-size: 12pt;
    -fx-font-family: "Segoe UI Light";
    -fx-text-fill: white;
    -fx-alignment: center-left;  //выравнивание
    -fx-opacity: 1;
}

.table-view:focused .table-row-cell:filled:focused:selected {
    -fx-background-color: -fx-focus-color; //Псевдокласс :focus определяет стиль для
элемента получающего фокус.
    //Например, им может быть текстовое поле формы, в которое устанавливается курсор.
}
//стиль для компонента split-pane
.split-pane:horizontal > .split-pane-divider {
    -fx-border-color: transparent #1d1d1d transparent #1d1d1d;
    -fx-background-color: transparent, derive(#1d1d1d,20%);
}

.split-pane {
    -fx-padding: 1 0 0 0;
}

.menu-bar {
    -fx-background-color: derive(#1d1d1d,20%);
}

.context-menu {
    -fx-background-color: derive(#1d1d1d,50%);
}

.menu-bar .label {
    -fx-font-size: 14pt;
    -fx-font-family: "Segoe UI Light";
    -fx-text-fill: white;
    -fx-opacity: 0.9;
}
```

```
.menu .left-container {
    -fx-background-color: black;
}

.text-field {
    -fx-font-size: 12pt;
    -fx-font-family: "Segoe UI Semibold";
}

/*
 * Push Button в стиле Metro
 * Автор: Pedro Duque Vieira
 * http://pixelduke.wordpress.com/2012/10/23/jmetro-windows-8-controls-on-java/
 */
.button {
    -fx-padding: 5 22 5 22;
    -fx-border-color: #e2e2e2;
    -fx-border-width: 2;
    -fx-background-radius: 0;
    -fx-background-color: #1d1d1d;
    -fx-font-family: "Segoe UI", Helvetica, Arial, sans-serif;
    -fx-font-size: 11pt;
    -fx-text-fill: #d8d8d8;
    -fx-background-insets: 0 0 0 0, 0, 1, 2;
}

.button:hover {  //ожидание
    -fx-background-color: #3a3a3a;
}
//для нажатой кнопки
.button:pressed, .button:default:hover:pressed {
  -fx-background-color: white;
  -fx-text-fill: #1d1d1d;
}
//при наведении на кнопку
.button:focused {
    -fx-border-color: white, white;
    -fx-border-width: 1, 1;
    -fx-border-style: solid, segments(1, 1);
    -fx-border-radius: 0, 0;
    -fx-border-insets: 1 1 1 1, 0;
}
//для неактиной кнопки
.button:disabled, .button:default:disabled {
    -fx-opacity: 0.4;
    -fx-background-color: #1d1d1d;
    -fx-text-fill: white;
}

.button:default {
    -fx-background-color: -fx-focus-color;
    -fx-text-fill: #ffffff;
}

.button:default:hover {
    -fx-background-color: derive(-fx-focus-color,30%);
}
```

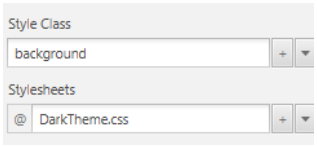
Теперь надо подключить эти стили к нашей сцене. Мы можем сделать это программно, в коде Java, но в этом уроке, чтобы добавить стили в наши fxml-файлы, мы будем использовать **Scene Builder:**

- Подключаем таблицы стилей к файлу **RootLayout.fxml**. Открываем этот файл в Scene Builder, во вкладке **Hierarchy** выберите корневой



контейнер **BorderPane**, перейдите на вкладку **Properties** и укажите файл **DarkTheme.css в** роли таблиц стилей.

- Подключаем таблицы стилей к файлу **PersonEditDialog.fxml**. Во вкладке **Hierarchy** выберите корневой контейнер **AnchorPane**, перейдите на вкладку **Properties** и укажите файл **DarkTheme.css** в роли таблиц стилей. Фон всё ещё белый, поэтому укажите для корневого компонента **AnchorPane** в классе стиля значение **background**.



Выберите кнопку **«OK»** и отметьте свойство **Default Button** в вкладке **Properties**. В результате изменится цвет кнопки и она будет использоваться по умолчанию когда пользователь, находясь в окне, будет нажимать клавишу **Enter**.

- Подключаем таблицы стилей к файлу **PersonOverview.fxml.** Во вкладке **Hierarchy** выберите корневой контейнер **AnchorPane**, перейдите на вкладку **Properties** и укажите файл **DarkTheme.css** в роли таблиц стилей. Вы сражу должны увидеть некоторые изменения: цвет таблицы и кнопок поменялся на чёрный. Все классы стилей **.table-view** и **.button** из файла **modena.css** были применены к таблице и кнопкам. С того момента, как мы переопределили некоторые из стилей в нашем css-файле, новые стили применяются автоматически. Возможно, вам потребуется изменить размер кнопок для того, чтобы отображался весь текст.

Выберите правый компонент **AnchorPane** внутри компонента **SplitPane**. Перейдите на вкладку **Properties** и укажите в классе стиля значение **background**. Теперь фон станет чёрного цвета.

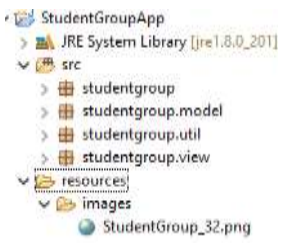
Текстовые метки с другими стилями. Сейчас все текстовые метки (**Label**) с правой стороны имеют одинаковый размер. Для дальнейшей стилизации текстовых меток мы будем использовать уже определённые стили **.label-header** и **label-bright**.

Выберите метку **Person Details** и добавьте в качестве класса стиля значение **label-header**.

Для каждой метки в правой колонке (где отображаются фактические данные о студентах) добавьте в качестве класса стиля значение **label-bright.**

2. Добавление иконки приложения.

Сейчас приложении в панели названия и панели задач используется иконка по умолчанию. Установим свою. Если нет маленькой картинки (32*32), можно скачать https://www.iconfinder.com. Создайте внутри вашего проекта **StudentGroupApp** обычную папку **resources**, а в ней папку **images**. Поместите выбранную вами иконку в папку изображений. Структура папок должна иметь такой вид.



Установим иконку для сцены, в классе **MainApp.java** добавьте следующий код в метод **start(...)**

```
// Устанавливаем иконку приложения.
this.primaryStage.getIcons().add(new Image("file:resources/images/StudentGroup_32.png"));
```

А сверху добавить импорт:

```
import javafx.scene.image.Image;
```

3. Сохранение пользовательских настроек.

Сейчас все изменения нашей базы существует только в памяти и не сохраняются после перезапуска. Благодаря классу **Preferences**, Java позволяет сохранять некоторую

информацию о состоянии приложения. В зависимости от операционной системы, Preferences сохраняются в различных местах (например, в файле реестра Windows).

Мы не можем использовать класс Preferences для сохранения всей базы. Но он позволяет сохранять некоторые простые настройки приложения, например, путь к последнему открытому файлу. Имея эти данные, после перезапуска приложения мы всегда сможем восстанавливать его состояние.

- Следующие два метода обеспечивают сохранение и восстановление настроек нашего приложения. Добавьте их в конец класса **MainApp**:

```
    /**
     * Возвращает preference файла студентов, то есть, последний открытый файл.
     * Этот preference считывается из реестра, специфичного для конкретной
     * операционной системы. Если preference не был найден, то возвращается null.
     *
     * @return
     */
    public File getPersonFilePath() {
        Preferences prefs = Preferences.userNodeForPackage(MainApp.class);
        String filePath = prefs.get("filePath", null);
        if (filePath != null) {
            return new File(filePath);
        } else {
            return null;
        }
    }

    /**
     * Задаёт путь текущему загруженному файлу. Этот путь сохраняется
     * в реестре, специфичном для конкретной операционной системы.
     *
     * @param file - файл или null, чтобы удалить путь
     */
    public void setPersonFilePath(File file) {
        Preferences prefs = Preferences.userNodeForPackage(MainApp.class);
        if (file != null) {
            prefs.put("filePath", file.getPath());

            // Обновление заглавия сцены.
            primaryStage.setTitle("StudentGroupApp - " + file.getName());
        } else {
            prefs.remove("filePath");

            // Обновление заглавия сцены.
            primaryStage.setTitle("StudentGroupApp ");
        }
    }
```

А сверху добавить импорт:

```
import java.io.File;
import java.util.prefs.Preferences;
```

Хранение данных в XML. Почему именно XML? Один из наиболее распространённых способов хранения данных, это использование баз данных. В то время, как данные, которые мы должны хранить, являются объектами, базы данных содержат их в виде реляционных данных (например, таблиц). Это называется объектно-реляционное рассогласование импендансов. Для того, чтобы привести наши объектные данные в соответствие с реляционными таблицами, требуется выполнить дополнительную работу. Существуют фреймворки, которые помогают приводить объектные данные в соответствие с реляционной базой данных (Hibernate - один из наиболее популярных), но чтобы начать их использовать, также необходимо проделать дополнительную работу и настройку.

Для нашей простой модели данных намного легче хранить данные в виде XML. Для этого мы будем использовать библиотеку JAXB (Java Architechture for XML Binding). Библиотека JAXB уже включена в JDK. Это значит, что никаких дополнительных библиотек подключать не придётся. JAXB предоставляет две основные функции: способность к маршаллированию объектов Java в XML и обратную демаршализацию из xml-файла в объекты Java. Для того, чтобы с помощью JAXB можно было выполнять подобные преобразования, нам необходимо подготовить нашу модель.

- Подготовка класса-модели для JAXB

Данные, которые мы хотим сохранять, находятся в переменной **personData** класса **MainApp**. JAXB требует, чтобы внешний класс наших данных был отмечен аннотацией **@XmlRootElement** (только класс, поле этой аннотацией пометить нельзя). Типом переменной **personData** является **ObservableList**, а его мы не можем аннотировать. Для того, чтобы разрешить эту ситуацию, необходимо создать *класс-обёртку*, который будет использоваться исключительно для хранения списка студентов, и который мы сможем аннотировать как **@XmlRootElement**.

Создайте в пакете **studentgroup.model** новый класс **PersonListWrapper.**

```
package studentgroup.model;

import java.util.List;

import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

/**
 * Вспомогательный класс для обёртывания списка студентов.
 * Используется для сохранения списка студентов в XML.
 *
 * @author Marco Jakob
 */
@XmlRootElement(name = "persons")    //определяет имя корневого элемента
public class PersonListWrapper {

    private List<Person> persons;

    @XmlElement(name = "person")     //необязательное имя, его можем задать для элемента
    public List<Person> getPersons() {
        return persons;
    }

    public void setPersons(List<Person> persons) {
        this.persons = persons;
    }
}
```

- Чтение и запись данных с помощью JAXB

Сделаем наш класс **MainApp** ответственным за *чтение и запись* данных нашего приложения. Для этого добавьте в конец класса **MainApp.java** два метода:

```
    /**
     * Загружает информацию о студентах из указанного файла.
     * Текущая информация о студентах будет заменена.
     *
     * @param file
     */
    public void loadPersonDataFromFile(File file) {
```

```
        try {
            JAXBContext context = JAXBContext
                    .newInstance(PersonListWrapper.class);
            Unmarshaller um = context.createUnmarshaller();

            // Чтение XML из файла и демаршализация.
            PersonListWrapper wrapper = (PersonListWrapper) um.unmarshal(file);

            personData.clear();
            personData.addAll(wrapper.getPersons());

            // Сохраняем путь к файлу в реестре.
            setPersonFilePath(file);

        } catch (Exception e) { // catches ANY exception
            Alert alert = new Alert(AlertType.ERROR);
            alert.setTitle("Error");
            alert.setHeaderText("Could not load data");
            alert.setContentText("Could not load data from file:\n" + file.getPath());

            alert.showAndWait();
        }
    }
    /**
     * Сохраняет текущую информацию о студентах в указанном файле.
     *
     * @param file
     */
    public void savePersonDataToFile(File file) {
        try {
            JAXBContext context = JAXBContext
                    .newInstance(PersonListWrapper.class);
            Marshaller m = context.createMarshaller();
            m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);

            // Обёртываем наши данные о студентах.
            PersonListWrapper wrapper = new PersonListWrapper();
            wrapper.setPersons(personData);

            // Маршаллируем и сохраняем XML в файл.
            m.marshal(wrapper, file);

            // Сохраняем путь к файлу в реестре.
            setPersonFilePath(file);
        } catch (Exception e) { // catches ANY exception
            Alert alert = new Alert(AlertType.ERROR);
            alert.setTitle("Error");
            alert.setHeaderText("Could not save data");
            alert.setContentText("Could not save data to file:\n" + file.getPath());
            alert.showAndWait();
        }
    }
```

А сверху добавить импорт:

```
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Marshaller;
import javax.xml.bind.Unmarshaller;
import studentgroup.model.PersonListWrapper;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
```

Маршаллинг и демаршализация готовы. Теперь для того, чтобы использовать новый функционал, давайте создадим пункты меню для сохранения и загрузки.

- Обработка действий меню.

В приложении **Scene Builder** откройте файл **RootLayout.fxml** и перенесите необходимое количество пунктов меню (**MenuItem**) из вкладки **Library** на вкладку **Hierarchy**. Создайте следующие пункты меню: **New**, **Open…**, **Save**, **Save as…** и **Exit**. Установите на пункты меню горячие клавиши используя свойство **Accelerator** во вкладке **Properties**.

- Класс RootLayoutController

Для обработки поведения меню необходим ещё один класс-контроллер. В пакете **studentgroup.view** создайте класс **RootLayoutControlle**r.

```
package studentgroup.view;

import java.io.File;

import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.stage.FileChooser;
import studentgroup.MainApp;

/**
 * Контроллер для корневого макета. Корневой макет предоставляет базовый
 * макет приложения, содержащий строку меню и место, где будут размещены
 * остальные элементы JavaFX.
 *
 * @author Marco Jakob
 */
public class RootLayoutController {

    // Ссылка на главное приложение
    private MainApp mainApp;

    /**
     * Вызывается главным приложением, чтобы оставить ссылку на самого себя.
     *
     * @param mainApp
     */
    public void setMainApp(MainApp mainApp) {
        this.mainApp = mainApp;
    }
    /**
     * Создаёт пустую базу
     */
    @FXML
    private void handleNew() {
        mainApp.getPersonData().clear();
        mainApp.setPersonFilePath(null);
    }
    /**
     * Открывает FileChooser, чтобы пользователь имел возможность
     * выбрать базу для загрузки
     */
    @FXML
    private void handleOpen() {
        FileChooser fileChooser = new FileChooser();

        // Задаём фильтр расширений
        FileChooser.ExtensionFilter extFilter = new FileChooser.ExtensionFilter(
                "XML files (*.xml)", "*.xml");
        fileChooser.getExtensionFilters().add(extFilter);

        // Показываем диалог загрузки файла
        File file = fileChooser.showOpenDialog(mainApp.getPrimaryStage());
```

```
        if (file != null) {
            mainApp.loadPersonDataFromFile(file);
        }
    }
    /**
     * Сохраняет файл в файл студентов, который в настоящее время открыт.
     * Если файл не открыт, то отображается диалог "save as".
     */
    @FXML
    private void handleSave() {
        File personFile = mainApp.getPersonFilePath();
        if (personFile != null) {
            mainApp.savePersonDataToFile(personFile);
        } else {
            handleSaveAs();
        }
    }
    /**
     * Открывает FileChooser, чтобы пользователь имел возможность
     * выбрать файл, куда будут сохранены данные
     */
    @FXML
    private void handleSaveAs() {
        FileChooser fileChooser = new FileChooser();

        // Задаём фильтр расширений
        FileChooser.ExtensionFilter extFilter = new FileChooser.ExtensionFilter(
                "XML files (*.xml)", "*.xml");
        fileChooser.getExtensionFilters().add(extFilter);

        // Показываем диалог сохранения файла
        File file = fileChooser.showSaveDialog(mainApp.getPrimaryStage());

        if (file != null) {
            // Make sure it has the correct extension
            if (!file.getPath().endsWith(".xml")) {
                file = new File(file.getPath() + ".xml");
            }
            mainApp.savePersonDataToFile(file);
        }
    }
    /**
     * Открывает диалоговое окно about.
     */
    @FXML
    private void handleAbout() {
        Alert alert = new Alert(AlertType.INFORMATION);
        alert.setTitle("StudentGroupApp");
        alert.setHeaderText("About");
        alert.setContentText("Some text");

        alert.showAndWait();
    }
    /**
     * Закрывает приложение
     */
    @FXML
    private void handleExit() {
        System.exit(0);
    }
}
```

Обратите внимание на методы в классе **RootLayoutController**, которые используют компонент **FileChooser**. Сперва мы создаём новый экземпляр класса **FileChooser**. Потом применяем фильтр расширения - при выборе файлов будут показываться только те, которые имеют расширение .xml. Ну и наконец, мы отображаем

данный компонент выше **PrimaryStage**. Если пользователь закрывает диалог выбора файлов ничего не выбрав, то возвращается **null**. В противном случае мы берём выбранный файл и передаём его в методы **loadPersonDataFromFile(...)** или **savePersonDataToFile(...),** которые находятся в классе **MainApp**.

- Связывание fxml-представления с классом-контроллером

В приложении **Scene Builder** откройте файл **RootLayout.fxml**. Во вкладке **Controller** в качестве класса-контроллера выберите значение **RootLayoutController**. Перейдите на вкладку **Hierarchy** и выберите пункт меню. Во вкладке **Code** в качестве значений свойства **On Action** вы можете увидеть все доступные методы выбранного класса-контроллера. Выберите метод, соответствующий данному пункту меню. Повторите предыдущий шаг для каждого пункта меню. Закройте приложение Scene Builder и обновите проект (нажмите **Refresh (F5**) на корневой папке вашего проекта).

- Связывание главного класса с классом RootLayoutController

В некоторых местах кода классу **RootLayoutController** требуется ссылка на класс **MainApp**. Эту ссылку мы ещё пока не передали. Откройте класс **MainApp** и замените метод **initRootLayout()** следующим кодом:

```
    /**
     * Инициализирует корневой макет и пытается загрузить последний открытый
     * файл с данными о студентах.
     */
    public void initRootLayout() {
        try {
            // Загружаем корневой макет из fxml файла.
            FXMLLoader loader = new FXMLLoader();
            loader.setLocation(MainApp.class
                    .getResource("view/RootLayout.fxml"));
            rootLayout = (BorderPane) loader.load();

            // Отображаем сцену, содержащую корневой макет.
            Scene scene = new Scene(rootLayout);
            primaryStage.setScene(scene);

            // Даём контроллеру доступ к главному приложению.
            RootLayoutController controller = loader.getController();
            controller.setMainApp(this);

            primaryStage.show();
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Пытается загрузить последний открытый файл с данными о студентах.
        File file = getPersonFilePath();
        if (file != null) {
            loadPersonDataFromFile(file);
        }
    }
```

Обратите внимание на два изменения: на строки, дающие доступ контроллеру к главному классу приложения и на три последних строки для загрузки последнего открытого файла с записями.

- Чтобы определить процесс преобразования типа данных как день рождения, мы должны предоставить собственный класс **LocalDateAdapter** внутри пакета **studentgroup.util**.

```
package studentgroup.util;

import java.time.LocalDate;

import javax.xml.bind.annotation.adapters.XmlAdapter;

/**
 * Адаптер (для JAXB) для преобразования между типом LocalDate и строковым
 * представлением даты в стандарте ISO 8601, например как '2019-12-22'.
  */
public class LocalDateAdapter extends XmlAdapter<String, LocalDate> {

    @Override
    public LocalDate unmarshal(String v) throws Exception {
        return LocalDate.parse(v);
    }

    @Override
    public String marshal(LocalDate v) throws Exception {
        return v.toString();
    }
}
```

Потом откройте класс **Person.java** и аннотируйте метод **getBirthday():**

```
@XmlJavaTypeAdapter(LocalDateAdapter.class)
    public LocalDate getBirthday() {
        return birthday.get();
    }
```

А сверху добавить импорт:

```
import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter;
import studentgroup.util.LocalDateAdapter;
```

Подытожим. Приложение запускается через метод **main(...)** класса **MainApp**. Вызывается конструктор **public MainApp()** и добавляются некоторые тестовые данные. Дальше в классе **MainApp** запускается метод **start(...),** который вызывает метод **initRootLayout()** для инициализации корневого макета из файла **RootLayout.fxml**. Файл fxml уже знает, какой контроллер следует использовать и связывает представление с **RootLayoutController'**ом. Класс **MainApp** из fxml-загрузчика получает ссылку на **RootLayoutController** и передаёт этому контроллеру ссылку на самого себя. Имея эту ссылку, контроллер может обращаться к публичным методам класса **MainApp**. В конце метода **initRootLayout** из настроек **Preferences** получаем путь к последнему открытому файлу базы. Если этот файл в настройках описан - загружаем из него данные. Эта процедура перезапишет тестовые данные, которые мы добавляли в конструкторе.

ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ

1. Создать свой стиль, учитывая размеры и стиль выбранного шрифта поправить размеры объектов формы.
2. Добавить иконку в окно добавления и редактирования студентов.

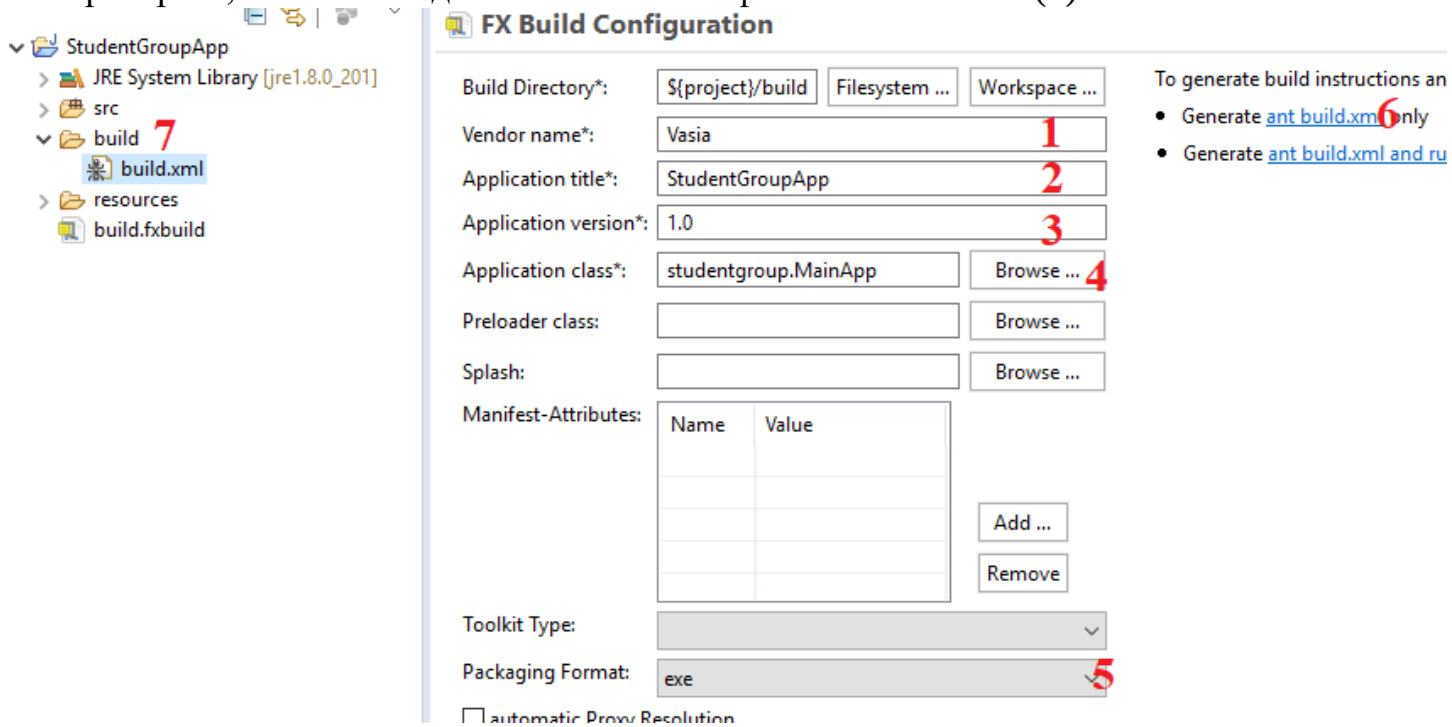
3. **Бонусное задание**. Развернуть свое готовое и протестированное приложение. Развёртывание - это процесс упаковки и доставки программного обеспечения

пользователю. Это иллюстрирует кроссплатформенность Java, состоящее в том, что пользователь сможет запуститься ваше приложение на любом устройстве, где установлена виртуальная машина Java (JVM). Применим “нативную упаковку” (Native Packaging) (другое название - “Автономная упаковка приложения” (Self-Containde Application Package)). Нативный Пакет - это пакет, который вместе с приложением Java содержит и среду выполнения Java для конкретной платформы. Цель - создание автономно-упакованного приложения, которое будет размещаться на компьютере пользователя в одной директории.

- Редактируем файл **build.fxbuild**

Файл **build.fxbuild** используется плагином e(fx)clipse для генерации файла, который в свою очередь будет использоваться инструментом сборки **Ant**. (Если у вас нет файла **build.fxbuild**, то создайте в **Eclipse** новый проект **JavaFX** и скопируйте оттуда сгенерированный файл).

- Откройте файл **build.fxbuild** из корневой папки вашего проекта.
- Заполните все поля, помеченные звёздочками (1-4)
- Выберите формат упаковки: для **Windows – exe**. (5)
- Справа нажмите на ссылку **Generate ant build.xml only**. (6)
- Проверьте, была ли создана папка **build** с файлом **build.xml**. (7)



- Добавляем в установщик иконки.

Создайте в папке **build** подкаталог **build/package/windows.** Положите туда иконки со следующим названием **НазваниеВашегоПриложения.ico, НазваниеВашегоПриложения-setup-icon.bmp**

- Добавляем ресурсы.

Папка **resources** не будет скопирована автоматически. Добавить её в папку **build**. Создайте в папке **build** подкаталог **dist**. Скопируйте папку **resources** (с иконками приложения, а не установщика) в **build/dist**

- Редактируем **build.xml** и включаем в него иконки.

Плагин e(fx)clipse сгенерировал файл **build/build.xml**, который готов для выполнения Ant‘ом. Но пока наши иконки и изображения ресурсов работать не будут. Пока (?) плагину e(fx)clipse нельзя указать, что нам необходимо включить дополнительные ресурсы, например папку **resource** и иконки, которые мы добавили ранее. Нам необходимо сделать это вручную путём редактирования **build.xml.**

- Откройте **build.xml** и найдите тег **<path id="fxant">.** Добавьте одну строку для **${basedir}** (это сделает наши иконки доступными):

```
<file name="${basedir}"/>
```

- Ниже найдите блок кода fx:resources id="appRes". Добавьте одну строку для наших ресурсов resources

```
<fx:fileset dir="dist" includes="resources/**"/>
```

- Добавьте номер версии в конце fx:application id="fxApplication"

```
version="1.0"
```

- **EXE**-установщик под **Windows**

С помощью утилиты Inno Setup можно создать установщик для Windows в виде единого .**exe**-файла. Созданный .**exe**-установщик будет устанавливать программу на уровне пользователя (не потребуются права администратора). Также он создаст ярлык (в меню или на рабочем столе).

- Скачайте **Inno Setup** версии 5 или выше. **Ant**-скрипт будет использоваться для автоматической генерации установщика.
- Укажите системе Windows путь к **Inno Setup** (например, *C:\Program Files (x86)\Inno Setup 5*). Для этого добавьте **Inno Setup** в переменную **Path** в переменных окружения системы (*Панель управления ->Система и безопасность ->Система -> Дополнительные параметры системы -> Переменные среды -> Системные переменные -> Path-> Изменить -> Создать*)
- Перезапустите приложение Eclipse.
- запустим файл build.xml с помощью Ant: кликните ПКМ на файле build.xml | Run As | Ant Build.
- Если всё прошло удачно, вы должны найти нативные сборки в папке build/deploy/bundles. Файл StudentGroupApp-1.0.exe может использоваться как единственный файл для установки приложения. Этот установщик скопирует нашу сборку в папку C:/Users/[yourname]/AppData/Local/StudentGroupApp.