|  |  |
| --- | --- |
| Федеральное агентство железнодорожного транспорта  Омский государственный университет путей сообщения  Кафедра «Автоматика и системы управления»  КОМПОНЕНТ TABLEVIEW  Лабораторная работа № 4  по теме «Программирование» | |
|  | Студент гр. 78-ИС                           Швед Т.Ю.  «    »                  2021 г.  Руководитель –                           Альтман Е. А.  «    »                  2021 г. |
| Омск 2021 | |

# Цель работы

Ознакомиться с реактивными типами данных и POJO объектами, изучить приемы отображения их в компонент TableView. На изучаемом языке программирования Java и изучение применения Java FX для типовых задач. Для этого нужно выполнить следующие написать программу, в которой повторим пример приложения, приведенный в теоретической части, а это:

повторить пример приложения, приведенный в теоретической части;

изменить класс «Expense», добавив в него поле для хранения кате- гории расходов;

переделать основное приложение для работы с новым вариантом класса «Expense»;

добавить в приложение текстовое поле «Итого», в котором бы отображалось общее количество затрат.

## Замечания по текущему документу

Данный документ преследует цель показать создания с реактивными типами данных и POJO объектами, изучить приемы отображения их в компонент TableView на изучаемом языке программирования Java и изучение применения Java FX для типовых задач. Научиться этому является основной целью лабораторной работы и необходимо для выполнения всех последующих лабораторных работ.

## Написание программы, в которой повторим пример приложения, приведенный в теоретической части.

Для начала мы напишем инерфейс, как показано на рисунке 1.

|  |
| --- |
|  |
| Рисунок  – Созданный нами интерфейс |

Данные, с которыми работает пользователь, должны быть реактивными, т.е. не только хранить значения, введенные пользователем, но и уметь сообщать об изменении значения всем объектам, нуждающимся в этой информации.

В JavaFX реактивные типы данных называются «Properties» (*свой- ства*). Они должны реализовывать одни из интерфейсов из иерархии, начина- ющийся с интерфейса «Observable»:

В «JavFX» имеется компонент «TableView», который позволяет выводить коллекцию POJO объектов в виде таблицы. Сформируем POJO класс для хранения затрат, используя реактивные типы данных Что мы получим в итоге, показано на рисунке 2.

|  |
| --- |
|  |
| Рисунок  – POJO класс для хранения затрат |

Далее необходимо, генерировать методы доступа. Методы доступа требуют большое количество кода, поэтому в современных средах разработки их можно автоматически генерировать. В IDE IntelliJ Idea для этого нужно использовать сочетания клавиш Alt- Insert, выбрать пункт Getter and Setter и выбрать поля, как показано на рисунке 3.

|  |
| --- |
|  |
| Рисунок  – Методы доступа |

Далее необходимо переделать приложение для учета затрат. В контроллере главного окна оставим следующие поля, как показано на рисунке 4.

|  |
| --- |
|  |
| Рисунок  – Наш контроллер и его переменные |

В «Scene Builder» вставим таблицу в центр компоновщика, пере- именуем колонки как «Название» и «Цена» и привяжем элементы к полям контроллера. Получим следующий «fxml» файл, как показано на рисунке 5.

|  |
| --- |
|  |
| Рисунок  – Файл «fxml» |

Для корректной работы нужно настроить компонент «TableView». Переделаем метод «init» контроллера, как показано на рисунке 6 и 7.

|  |
| --- |
|  |
| Рисунок  – Запуск тестовой инициализации, т.е. добавляем свои поля |

|  |
| --- |
|  |
| Рисунок  – Запуск инициализации, с обработкой и выводом данных |

А так же необходимо в главном классе, настроить взаимодействие с контролерром и «fxml» файл, как показано на рисунке 8.

|  |
| --- |
|  |
| Рисунок 8 – Запуск программы |

И вот что мы увидим, при запуске программы. Показано на рисунке 7.

|  |
| --- |
|  |
| Рисунок  – Проверка работы |

## Изменение программы

Текст программы набирается в окне файла с расширением «.java» и «.fxml». Исходный код приводится в виде листинга, в приложения А.

2. Выводы по лабораторной работе

Из выше указанного, можно точно сказать, что был достигнут искомый результат. Показан основной процесс создания с реактивными типами данных и POJO объектами, изучить приемы отображения их в компонент TableView на изучаемом языке программирования Java, с применением Java FX .

Приложение А

(справочное)

Main

**package** sample;  
  
**import** javafx.application.Application;  
**import** javafx.collections.FXCollections;  
**import** javafx.collections.ObservableList;  
**import** javafx.fxml.FXMLLoader;  
**import** javafx.scene.Parent;  
**import** javafx.scene.Scene;  
**import** javafx.stage.Stage;  
  
**public class** Main **extends** Application {  
  
 @Override  
 **public void** start(Stage primaryStage) **throws** Exception{  
  
 FXMLLoader loader = **new** FXMLLoader(); *// создаем загрузку FXML* loader.setLocation(getClass().getResource(**"sample.fxml"**)); *// добавляем нашу FXML* Parent root = loader.load(); *// создаем панель и загружаем наши данные* Controller controller = loader.getController(); *// создаем контроллер* controller.init(); *// запускаем в контроллере метод init* primaryStage.setTitle(**"TableView"**); *// заголовок формы* primaryStage.setScene(**new** Scene(root, 310, 330)); *// размеры формы и наша панель, в форму(окно)* primaryStage.show(); *// показать форму* }  
   
 **public static void** main(String[] args) {  
 *launch*(args);  
 }  
}

sample.fxml

*<?***xml version="1.0" encoding="UTF-8"***?>  
  
<?***import javafx.geometry.Insets***?>  
<?***import javafx.scene.control.Button***?>  
<?***import javafx.scene.control.Label***?>  
<?***import javafx.scene.control.TextField***?>  
<?***import javafx.scene.effect.Blend***?>  
<?***import javafx.scene.layout.BorderPane***?>  
<?***import javafx.scene.layout.HBox***?>  
<?***import javafx.scene.layout.VBox***?>  
<?***import javafx.scene.text.Font***?>  
<?***import javafx.scene.control.TableView***?>  
  
<?***import javafx.scene.control.TableColumn***?>*<**BorderPane fx:id="root" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="200.0" prefWidth="300.0" xmlns="http://javafx.com/javafx/8.0.172" xmlns:fx="http://javafx.com/fxml/1" fx:controller="sample.Controller"**>  
 <**center**>  
 <**TableView fx:id="table" BorderPane.alignment ="CENTER"**>  
 <**columns** >  
 <**TableColumn fx:id ="nameColumn" text="Название"**/>  
 <**TableColumn fx:id ="costColumn" text="Цена"**/>  
 <**TableColumn fx:id ="categoryColumn" text="Категория расходов"**/>  
 </**columns** >  
 </**TableView** >  
 </**center**>  
 <**bottom**>  
 <**Label fx:id ="sum" BorderPane.alignment ="CENTER"**></**Label**>  
 </**bottom**>  
</**BorderPane**>

Controller

**package** sample;  
  
**import** javafx.beans.value.ObservableValue;  
**import** javafx.collections.FXCollections;  
**import** javafx.collections.ObservableList;  
**import** javafx.event.ActionEvent;  
**import** javafx.event.EventHandler;  
**import** javafx.fxml.FXML;  
**import** javafx.scene.control.\*;  
  
**public class** Controller {  
  
 *// -------------------------------------------------------------------- переменные ---------------------------------------------* ObservableList<Expense> **expenseList** = FXCollections.*observableArrayList*();  
  
 @FXML TableView<Expense> **table**;  
 @FXML TableColumn<Expense, String> **nameColumn**;  
 @FXML TableColumn<Expense, Number> **costColumn**;  
 @FXML TableColumn<Expense, String> **categoryColumn**;  
 @FXML Label **sum**;  
  
 *// ----------------------------------------------------------------------- инициализация ---------------------------------------* **public void** init (ObservableList<Expense> expenseList){  
 **table**.setItems(expenseList) ;  
 *// добавление строки в столбик (имя)* **nameColumn**.setCellValueFactory(cellData ->  
 cellData.getValue().nameProperty()) ;  
 *// добавление строки в столбик (цена)* **costColumn**.setCellValueFactory(cellData ->  
 cellData.getValue().costProperty()) ;  
 *// подсчёт затрат* **float** sumCost=0;  
 **for** (Expense e: expenseList){ *// перебор затрат* sumCost += e.**cost**.getValue();  
 }  
 **sum**.setText(**"Итого: "**+sumCost+**" руб."**); *// вывод затрат  
 // добавление строки в столбик (цена)* **categoryColumn**.setCellValueFactory(cellData ->  
 cellData.getValue().categoryOfExpensesProperty()) ;  
 }  
  
 *// ----------------------------------------- пустая инициализация (если пользователь ничего не передал)* **public void** init() {  
 **this**.testInit();  
 **this**.init(**expenseList**);  
 }  
  
 *// ----------------------------------------- тестовая инициализация* **public void** testInit() {  
 **expenseList**.add(**new** Expense ( **"Хлеб"**, **new** Float(30), **"Еда"**));  
 **expenseList**.add ( **new** Expense ( **"Вода"**, **new** Float(35 ), **"Еда"**));  
 **expenseList**.add ( **new** Expense ( **"Проезд"**, **new** Float(30 ), **"Личное"**));  
 **expenseList**.add ( **new** Expense ( **"Верёвка"**, **new** Float(150 ), **"Хозяйство"**));  
 **expenseList**.add ( **new** Expense ( **"Книга"**, **new** Float(340 ), **"Личное"**));  
 **expenseList**.add ( **new** Expense ( **"Мыло"**, **new** Float(120 ), **"Хозяйство"**));  
 **expenseList**.add ( **new** Expense ( **"Масло"**, **new** Float(35 ), **"Еда"**));  
 **expenseList**.add ( **new** Expense ( **"Сыр"**, **new** Float(220 ), **"Еда"**));  
 **expenseList**.add ( **new** Expense ( **"Табурет"**, **new** Float(1500 ), **"Хозяйство"**));  
 *//ObservableValue<Number> x; // данная строчка не имеет смысла, но может пригодится если переделать на возвращаемую функцию. Но у нас void =D* }  
}

Observable

**package** sample;  
  
**import** javafx.beans.InvalidationListener;  
  
**public interface** Observable {  
 **void** addListener(InvalidationListener listener);  
 **void** removeListener(InvalidationListener listener);  
}

Expense

**package** sample;  
  
**import** javafx.beans.property.FloatProperty;  
**import** javafx.beans.property.SimpleFloatProperty;  
**import** javafx.beans.property.SimpleStringProperty;  
**import** javafx.beans.property.StringProperty;  
**import** javafx.beans.value.ObservableValue;  
  
**public class** Expense {  
 *//--------------------------------------------------------------------------------- переменные ------------------------------------------------------------* StringProperty **name** = **new** SimpleStringProperty(**""**); *// имя* FloatProperty **cost** = **new** SimpleFloatProperty( 0 ) ; *// цена* StringProperty **categoryOfExpenses** = **new** SimpleStringProperty(**""**); *// категория расходов  
  
 // --------------------------------------------------------------------------------- обработка событый -------------------------------------------------------  
  
 //----------------------------- ИМЯ* **public** String getName () { *// вернуть имя (ещё не использовали, но вдруг пригодиться)* **return name**.get();  
 }  
 **public** StringProperty nameProperty () { *// вернуть значение-строку с именем* **return name**;  
 }  
 **public void** setName (String name) { *// записать имя (ещё не использовали, но вдруг пригодиться)* **this**.**name**.set(name);  
 }  
 *//------------------------------ ЦЕНА* **public** FloatProperty costProperty () { *// вернуть значение-дробное с ценой* **return cost**;  
 }  
 *//----------------------------- КАТЕГОРИЯ* **public** String getСategoryOfExpenses () { *// вернуть категорию (ещё не использовали, но вдруг пригодиться)* **return categoryOfExpenses**.get();  
 }  
 **public** StringProperty categoryOfExpensesProperty () { *// вернуть значение-строку с категорией* **return categoryOfExpenses**;  
 }  
 **public void** setСategoryOfExpenses (String name) { *// записать категорию (ещё не использовали, но вдруг пригодиться)* **this**.**categoryOfExpenses**.set(name);  
 }  
  
  
  
 *//-------------------------------------------------------------------------------------конструтор -----------------------------------------------* **public** Expense ( String name , Float cost, String categoryOfExpenses) {  
 **this**.**name** = **new** SimpleStringProperty(name); *// добавить имя в класс* **this**.**cost** = **new** SimpleFloatProperty(cost); *// добавить цену в класс* **this**.**categoryOfExpenses** = **new** SimpleStringProperty(categoryOfExpenses); *// добавить цену в класс* }  
  
}