# Лабораторная работа №6

### Задание на лабораторную работу

Разработать графическое приложение, позволяющее работать с табулированными функциями: создавать их, редактировать, сохранять в файлы и считывать из файлов, а также создавать табулированные аналоги функций, загружаемых в программу в виде байт-кода классов.

При разработке окон приложения настоятельно рекомендуется использование визуальных средств и мастеров среды разработки, предназначенных для создания JavaFX приложений.

Использование в качестве имён переменных неинформативных идентификаторов вида label1 (в таком виде их любят создавать среды разработки) запрещено.

Все новые классы этого задания следует создавать в отдельном пакете, название которого выберите самостоятельно, если не указано иное.

Перед тем, как переходить к написанию лабораторной работы, следует скачать и установить JavaFX. Cкачать его можно по ссылке: [gluonhq.com/products/javafx/](https://gluonhq.com/products/javafx/). Для визуального конструирования окон потребуется установленный Scene Builder, скачать его можно здесь: [gluonhq.com/products/scene-builder/](https://gluonhq.com/products/scene-builder/). Настоятельно рекомендуется его использовать, для чего следует выбирать вариант описание окон в FXML – данный файл и редактируется Scene Builder’ом. При работе над графическим Java приложением без FXML файла всю работу по установке элементов окон, добавлении свойств и т.д. придется проделывать вручную из кода вызывающего метода, что с одной стороны требует больше труда, с другой – усложняет дальнейшую поддержку окон.

Как подключить пакеты JavaFX и Scene Builder например, в NetBeans, можно посмотреть здесь: [youtu.be/h\_3AfQhjziw](https://youtu.be/h_3AfQhjziw)

**За реализацию данной работы с помощью JavaEE в браузере можно получить до 5 дополнительных баллов!**

### Задание 1 *(1 балл)*

Описать класс документа с табулированной функцией TabulatedFunctionDoc. Разместите его в новый пакет, имя которому дайте сами.

Объект этого класса должен описывать открытый в программе документ, т.е. содержать ссылку на объект табулированной функции в приватной переменной, текущее имя файла документа, а также булевский флаг, показывающий то, изменялся ли документ с момента последнего сохранения. Значения последних двух полей должны соответствующим образом изменяться в ходе работы методов класса.

Дополнительно реализуйте в классе следующие методы, соответствующие действиям с документом, которые может выполнить пользователь:

* метод newFunction(double leftX, double rightX, int pointsCount) должен заменять объект табулированной функции на новый с указанными параметрами;
* метод tabulateFunction(Function function, double leftX, double rightX, int pointsCount) должен заменять объект табулированной функции на новый, полученный путём табулирования указанной функции с указанными параметрами.
* Метод saveFunctionAs(String filenName) должен сохранять текущую табулированную функцию в формате JSON. Для работы с ним следует подключить библиотеку org.json.simple.JSONObject (для этого ее потребуется скачать, например здесь: <http://www.java2s.com/Code/Jar/j/Downloadjsonsimple11jar.htm>, а пример работы с JSON рассмотрен здесь <https://howtodoinjava.com/java/library/json-simple-read-write-json-examples>). Для подключения библиотеки в NetBeans следует щелкнуть правой клавишей мыши по проекту в дереве проекта, выбрать свойства, далее в разделе Libraries – Classpath нажать “+”, выбрать “Add JAR/Folder” и указать файл библиотеки, извлеченный из архива.
* метод saveFunction() должен сохранять текущую табулированную функцию в текстовый файл с текущим именем в формате JSON;
* метод loadFunction(String fileName) должен загружать из файла с указанным именем табулированную функцию в формате JSON. Для разбора JSON формата предлагается использовать класс org.json.simple.parser.JSONParser (разбор JSON описан здесь: <https://hr-vector.com/java/json-parser> )

При написании этих методов воспользуйтесь методами класса TabulatedFunctions.

Определите в классе два доступных только для чтения булевских свойства modified и fileNameAssigned, показывающих, соответственно, изменялся ли документ с последнего сохранения и назначено ли для документа имя файла.

Чтобы предоставить возможность работы с хранящейся табулированной функцией, а также обеспечить отслеживание изменений, вместо предоставления прямого доступа к объекту функции поступим следующим образом. Пусть класс документа сам реализует интерфейс TabulatedFunction и содержит реализации всех методов этого интерфейса. Но вместо действительной реализации этих методов (такой, какая была в классах ArrayTabulatedFunction и LinkedListTabulatedFunction) в них просто будут вызываться методы объекта табулированной функции, хранящейся в документе. Т.е. объект документа, вместо того, чтобы самому, например, вычислять значение функции, будет заставлять это делать хранящийся в документе объект табулированной функции.

Таким образом, требуется реализовать все методы из интерфейса TabulatedFunction, а также переопределить методы из класса Object так, чтобы для реального выполнения действия вызывались методы объекта хранящейся в документе табулированной функции. Исключениями будут являться методы equals() и clone() (подумайте, почему это так, и реализуйте методы корректно). В методах, которые меняют функцию, нужно ещё и изменять значение поля, показывающего, что документ изменялся. Подумайте, можно ли всё это сделать путем наследования от одного из класса табулированной функции.

### Задание 2 *(2 балла)*

Сделайте ваше приложение оконным. Для этого добавьте **extends** Application в основной класс программы, в котором запускается main. Добавьте в этот класс метод start следующего содержания:

@Override

**public void** start(Stage stage) **throws** Exception {

FXMLLoader loader=**new** FXMLLoader(getClass().getResource("FXMLMainForm.fxml"));

Parent root = loader.load();

FXMLMainFormController ctrl=loader.getController();

Scene scene = new Scene(root);

stage.setTitle("*Впишите сюда заголовок главного окна*");

stage.setScene(scene);

stage.show();

}

Обратите внимание на объявленную переменную ctrl, которая нигде не используется. Ее использование предвидится далее. А в методе main вставьте вызов launch(args), который и запустит метод start. Для того, чтоб данный код заработал, добавьте к приложению FXML файл с описанием окна FXMLMainForm.fxml. Например, в NetBeans для этого следует щелкнуть по пакету, куда вы хотите добавить новый файл (тот пакет, в котором находится основной класс), выбрать New -> Empty FXML, далее в строке имени файла указать FXMLMainForm.fxml, нажать далее, выбрать флажок Use Java Controller, Create New, затем нажать далее, не устанавливать флажок об использовании CSS и нажать готово. Дважды щелкните по файлу FXMLMainForm.fxml, должен открыться Scene Builder с пустым окном.

Теперь реализуйте дизайн основного окна программы. Пример такого окна показан на рисунке 1.

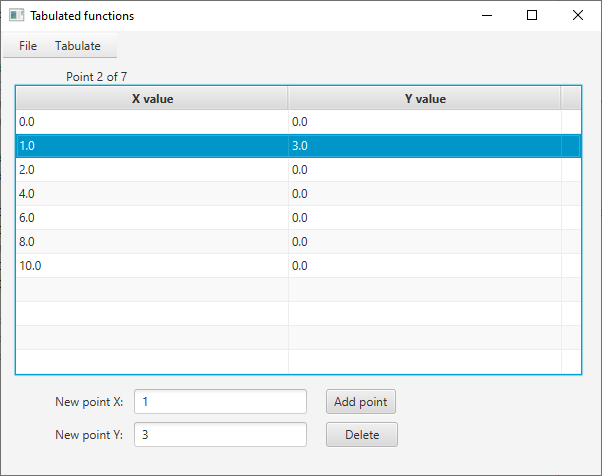


Рисунок 1 – Примерный вид основного окна программы

Для этого откройте палитру элементов слева и накидайте на форму требуемые элементы управления: Label, TableView, TextField, Button. Обратите внимание, что в правой части экрана Scene Builder отображает свойства выбранного элемента. В разделе Code поле fx:id представляет собой имя элемента, которое затем можно использовать в контроллере окна, а в поля “On …” можно вписать различные обработчики событий. Текстовые поля для ввода значений X и Y назовите edX, edY. Если всё сделано правильно, то после запуска приложения будет открываться основное окно, по закрытию которого будет закрываться ваше приложение. Для того, чтобы добавить к кнопке Add point метод, запускаемый по её нажатию, введите в Scene Builder для требуемой кнопки в поле “On Action” название метода btNewClick. Затем внесите его в контроллер окна:

@FXML

**private void** btNewClick(ActionEvent av) {

}

Для того, чтобы убедиться, что данный метод работает, вставьте в него одну простую строку:

edY.setText(edX.getText());

Чтобы это заработало, текстовые поля edX и edY должны быть описаны в контроллере:

@FXML

**private** TextField edX;

Аналогичным образом в контроллер окна добавляются и другие элементы управления в главном окне, к которым требуется обращаться. В данном примере при нажатии на кнопку Add point текст из поля edX будет копироваться в строку edY.

Добавьте к основному классу программы поле **public static** TabulatedFunctionDoc tabFDoc, который будет являться документом табулированной функции, с которым будут работать окна. Сделайте так, чтобы при запуске программы он проходил инициализацию и в нем создавалась табулированная функция по умолчанию с помощью конструктора её класса.

Для отображения данных в таблице TableView следует создать класс, объекты которого будут описывать отображаемые в таблице строки. В данном случае нужны два значения чисел с плавающей точкой – такой, как класс FunctionPoint. Однако, используемый в нём тип double не подходит, так как является простым типом. Для того, чтобы настроить отображение данных в таблице через фабрику, необходимо использовать объектный тип Double. Создайте новый класс FunctionPointT, который будет содержать в себе значения x и y, а сеттеры и геттеры класса будут работать с типом Double, причем имена геттеров будут формироваться строго в соответствии с именем переменных (например, для переменной myVariable геттер должен иметь имя getMyVariable). Теперь в соответствии с каждой строкой в таблице можно будет поставить в соответствие один экземпляр класса FunctionPointT. Укажите тип данных для таблицы с помощью дженерика при её объявлении в контроллере:

@FXML

**private** TableView<FunctionPointT> table;

где table – ваше имя таблицы fx:id. Используя свойства table.getColumns() и класс TableColumn создайте нужные колонки в таблице в методе initialize контроллера, настройте фабрики для отображения.

Добавьте в класс контроллера метод redraw() без параметров, сделайте его доступным из других классов. В нем напишите код для обновления данных в таблице из поля класса tabFDoc. Добавьте методы для кнопок Add point, Delete point, убедитесь, что они корректно работают, вызывая после них метод redraw().

Поскольку табулированная функция, находящаяся в статичной переменной, может быть изменена откуда угодно из программы, нужно корректно следить за тем, когда она в действительности изменилась, и в этом случае выполнять перерисовку таблицы. Для этого удобно воспользоваться паттерном проектирования «издатель-подписчик», отдав на откуп перерисовку таблицы самому документу. То есть надо зарегистрировать в документе табулированной функции контроллер окна с таблицей, которая ответственна за отображение данных документа. Тогда при любом изменении табулированной функции tabFDoc сам сможет вызывать перерисовку таблицы, и данные в ней будут всегда актуальны. Для вызова перерисовки следует воспользоваться паттерном проектирования «посредник», создав внутри класса TabulatedFunctionDoc метод CallRedraw(), из него и будет вызываться метод redraw() в контроллере, если таковой зарегистрирован. А для регистрации окна следует добавить в класс TabulatedFunctionDoc метод registerRedrawFunctionController, принимающий в качестве аргумента экземпляр контроллера. Подумайте над типом объекта, передаваемого в функцию registerRedrawFunctionController, и на что он влияет. Нужно ли объявлять интерфейс, который реализует класс контроллера? Что бы изменилось, если tabFDoc был бы не статичный? Затем добавьте внутри класса документа табулированной функции вызов CallRedraw() во все методы, которые приводят к изменению функции в том случае, если они действительно её меняют, в том числе и в сам метод регистрации контроллера. В случае же ошибочного вызова функций, если табулированная функция не изменилась, вызов CallRedraw() не должен быть осуществлен. После этого следует убрать все ссылки на функцию redraw() внутри контроллера окна – табулированная функция должна оставаться актуальной при её изменении. Надпись label о том, сколько всего точек в табулированной функции, а так же показывающая номер текущей выбранной пользователем строки, должна меняться как из функции redraw(), так и из событий при отпускании кнопки мыши на таблице и при отпускании клавиши на ней. Для этого напишите соответствующие обработчики событий.

### Задание 3 *(2 балла)*

Добавьте меню в основное окно программы со следующими пунктами:

* элементы основного меню окна (компонент типа MenuBar) для работы с файлами табулированных функций и для операции табуляции;
* элементы первого меню (компоненты типа Menu) для создания нового документа (MenuItem), открытия документа из файла, сохранения документа, сохранения документа под новым именем, выхода из программы;
* элемент второго меню (компоненты типа Menu) для загрузки и табулирования функции;

При выборе пункта «создания нового документа» должно появляться диалоговое окно, в котором будет предложено выбрать границы новой табулированной функции и количества точек, после чего существующая функция должна быть заменена новой. Для этого реализуйте класс вспомогательного окна, в котором вводятся параметры табулирования функции. Пример такого окна показан на рисунке 2. Для создания класса аналогично воспользуйтесь FXML файлом для позиционирования элементов управления и Scene Builder для его редактирования. При этом должен использоваться новый класс-контроллер для этого окна.

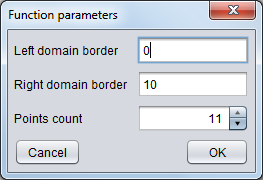


Рисунок 2 – Примерный вид окна параметров табулирования функции

Окно должно быть модальным, т.е. оно должно блокировать работу пользователя с родительским приложением до тех пор, пока пользователь его не закроет. Для этого следует к алгоритму показа окна добавить следующий код:

stage.initModality(Modality.APPLICATION\_MODAL);

stage.initOwner(primaryStage);

stage.showAndWait();

В данном коде имеется ввиду, что stage – это stage нового окна на рис. 2, а primaryStage – это stage родительского окна. Чтобы контроллер родителя «знал» свой stage, следует воспользоваться контроллером ctrl, полученном в методе start, и с помощью него передать в объект класса контроллера основного окна его stage. Для этого следует добавить в контроллер необходимые поля и методы. Ссылаться на данные методы можно будет в виде ctrl.вашНовыйМетод().

Для простоты данное окно не должно позволять изменять свой размер:

stage.setResizable(false);

Расположите на форме следующие компоненты:

* два поля редактирования для левой и правой границы области определения табулированной функции (компоненты типа TextField);
* изменяемое числовое поле для редактирования количества точек табулированной функции (компонент типа Spinner);
* текстовые пояснения для редактируемых полей (компоненты типа Label);
* кнопки отмены действия и подтверждения действия (компоненты типа Button).

Для текстовых полей редактирования задайте значения по умолчанию.

Поведение изменяемого числового поля Spinner определяется фабрикой, установленной для его значений. Объявите его в контроллере окна с использованием дженерика <Integer>:

@FXML

**private** Spinner<Integer> spinner;

Далее при инициализации окна установите ему фабрику целых чисел и задайте пределы изменения его значений. Установите в нем возможность задавать значения с клавиатуры.

Для удобства дальнейшей работы определите в классе две целочисленных «константы» OK и CANCEL с различными значениями. Также объявите целочисленное поле, в котором будет храниться статус окна после его закрытия, т.е. то, какую из кнопок нажал пользователь.

Напишите обработчик события нажатия на кнопку подтверждения операции, выполняющий следующие действия:

* проверка корректности введённых числовых значений,
* сокрытие окна,
* присвоение полю статуса окна значения OK.

Введённое в текстовое поле значение может быть получено с помощью метода String getText(). Для преобразования к типу double будет удобно воспользоваться методом Double.parseDouble(), выбрасывающим исключение NumberFormatException в том случае, если строка содержит не число, или Double.valueOf(), если нужен фомат Double. Если пользователь ввёл некорректные данные, нужно вывести ему сообщение об ошибке. Это разумно сделать во всплывающем диалоговом окне, которое можно вывести с помощью класса Alert. Для этого создайте новый alert, укажите его тип, установите его заголовок, сообщение, и вызовите его метод ShowAndWait(). Все необходимые для этого методы изучите самостоятельно.

Для сокрытия окна необходимо для stage окна дать команду hide(), для закрытия – close(). Как получить stage окна в его контроллере, и как обращаться к его методам, Вы уже знаете.

Напишите обработчик события нажатия на кнопку отмены операции, выполняющий следующие действия:

* сокрытие окна,
* присвоение полю статуса окна значения CANCEL.

Поскольку кроме кнопки отмены операции пользователь может просто нажать на закрывающую окно кнопку, требуется предусмотреть этот случай. Для этого можно установить обработчик на событие закрытия с помощью метода setOnCloseRequest для stage, или спроектировать программу определенным образом, чтобы было не важно, как именно было закрыто окно.

Добавьте в класс метод showDialog(), который и будет использоваться для вывода окна диалога на экран. Метод должен делать окно видимым и возвращать статус окна после завершения работы с ним. Подумайте, следует ли создавать новое окно каждый раз при обращении к showDialog(), или можно показать уже созданный экземпляр. Подумайте, что следует для этого добавить в код, чтобы экземпляр окна был всегда один. Подумайте, какие плюсы Вы получите в этом случае. Реализуйте наиболее практичный вариант.

Для того, чтобы получить введенные пользователем значения, добавьте методы в контроллер нового окна getLeftDomainBorder(), getRightDomainBorder() и getPointsCount(), возвращающие, соответственно, введённые величины.