

Лекция 4

Въведение в Java FXML.
Създаване на интерактивен потребителски
интерфейс с графичен визуален
редактор.

- 4.1** Въведение
- 4.2** Графични компоненти
- 4.3** Именуване на компоненти от графичния интерфейс
- 4.4** Създаване на интерактивен графичен интерфейс
- 4.5** Създаване на JavaFX интерактивен графичен интерфейс
 - 4.1** Структура на JavaFX прозорец
 - 4.2** Инсталиране на SceneBuilder
 - 4.3** Създаване на JavaFX приложение
 - 4.4** Описание на дървото от възли
 - 4.5** Изграждане на графичния интерфейс
 - 4.6** Въведение в обработка на събития
 - 4.9** Подреждане на компонентите
 - 4.10** Структура на възлите
- 4.6** Приложение на BigDecimal за работа с парични суми
- 4.7** Създаване на менюта в SceneBuilder
- 4.8** Създаване на многократно използваеми FXML графични компоненти

1 Въведение

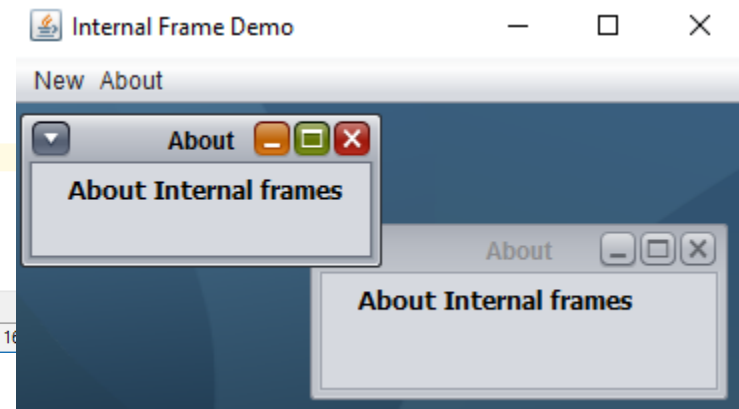
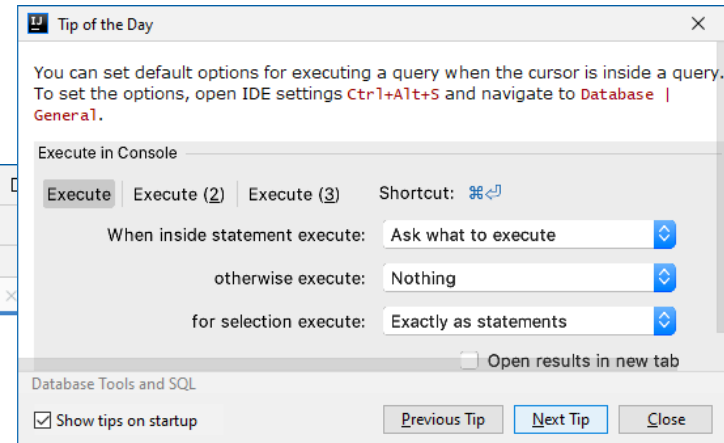
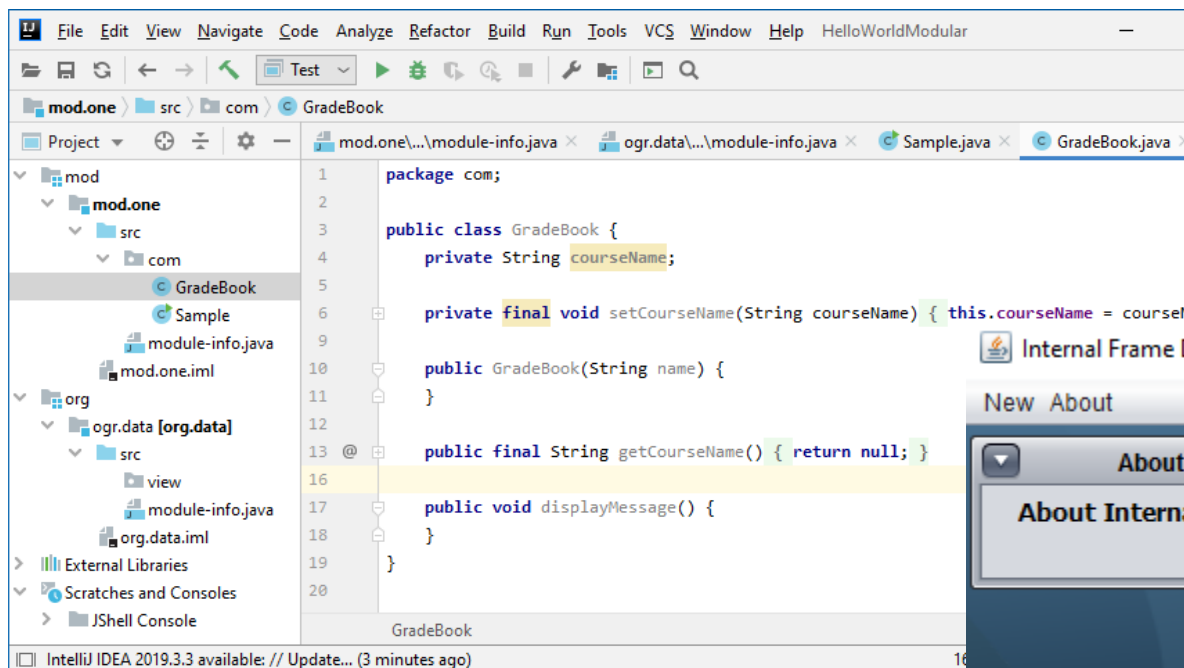
Graphical User Interface (GUI произнася се "GOO-ee")

- Предоставя удобен и интуитивно лесен за разбиране механизъм за взаимодействие на потребителя с приложението**
- Най- често се състои от меню, бутони, етикети и съответни текстови полета за въвеждане на данни**
- Съставен е от типични GUI компоненти- контроли**
- GUI компонентата е обект, който позволява взаимодействие на потребителя с приложението във основа на събития, породени от мишка, клавиатура или друга форма за въвеждане на данни като разпознаване на реч.**

1 Въведение

Видове прозорци

- Модални (диалогови прозорци)
- Прозорец, който не съдържа други прозорци(SDI)
- Прозорец, в който може да се влагат други прозорци (MDI)



2 Графични компоненти

Текстовите полета служат за въвеждане на текст.

Различаваме едноредови и многоредови текстови полета. Текстовите полета могат да се заключват за редактиране на съдържанието ми, когато служат за извеждане на резултат от обработка на данни.

Етикетите служат за изобразяване на текст. Обикновено този текст е неизменен. Най- често етикетите са свързани с текстово поле и изобразяват текст, който пояснява съдържанието на текстовото поле.

Бутоните служат за инициране на дейности за обработка на данни, въведени в графичния интерфейс.

Панелите служат за групиране на графични компоненти или за изобразяване на графична информация

3 Именуване на компоненти от графичния интерфейс

Модифицирана Унгарска нотация

Използва се с цел сорс кода на специализирани приложения (графични, бази данни, мрежово програмиране) да бъде разбираем. Характерно за такива приложения е използването на библиотеки от компоненти

Нотацията използва три буквени префикси за различаване на променливите на стандартните компоненти от потребителски дефинираните променливи.

Модифицирана Унгарска нотация

Control	Prefix
Button	<i>btn</i>
ComboBox	<i>cbo</i>
CheckBox	<i>chk</i>
Label	<i>lbl</i>
ListBox	<i>lst</i>
MainMenu	<i>mnu</i>
RadioButton	<i>rdb</i>
TextArea	<i>txa</i>
TextField	<i>txt</i>

4 Създаване на интерактивен графичен интерфейс

Графичният интерфейс се управлява от събития. Всяко въздействие на потребителя върху компонентите на графичния интерфейс поражда събитие. Обработката на събитието позволява на **потребителя да общува интерактивно** с програмата като ползва графичен потребителски интерфейс.

- Обикновени събития- натискане на бутон на мишката, натискане на клавиш, избиране на елемент на списък, елемент на меню и пр. ,
- Всяко събитие, веднъж породено, води до извикване на метод, наричан метод за обработка на събитието (*event handler*).
- В Java методът за обработка на събитието се дефинира от потребителя на компонентата и се “*пакетира*” в клас, който се дефинира по строго зададени правила.
- За да се изпълни желаната от потребителя обработка на събитието, потребителят на компонента трябва да се „*абонира*“ за съответното събитие на тази компонента.
- В повечето случаи е възможно интегрираната среда за разработка да извърши с помощни средства „пакетирането“ на метода за обработка и „абонирането“ за събитие.

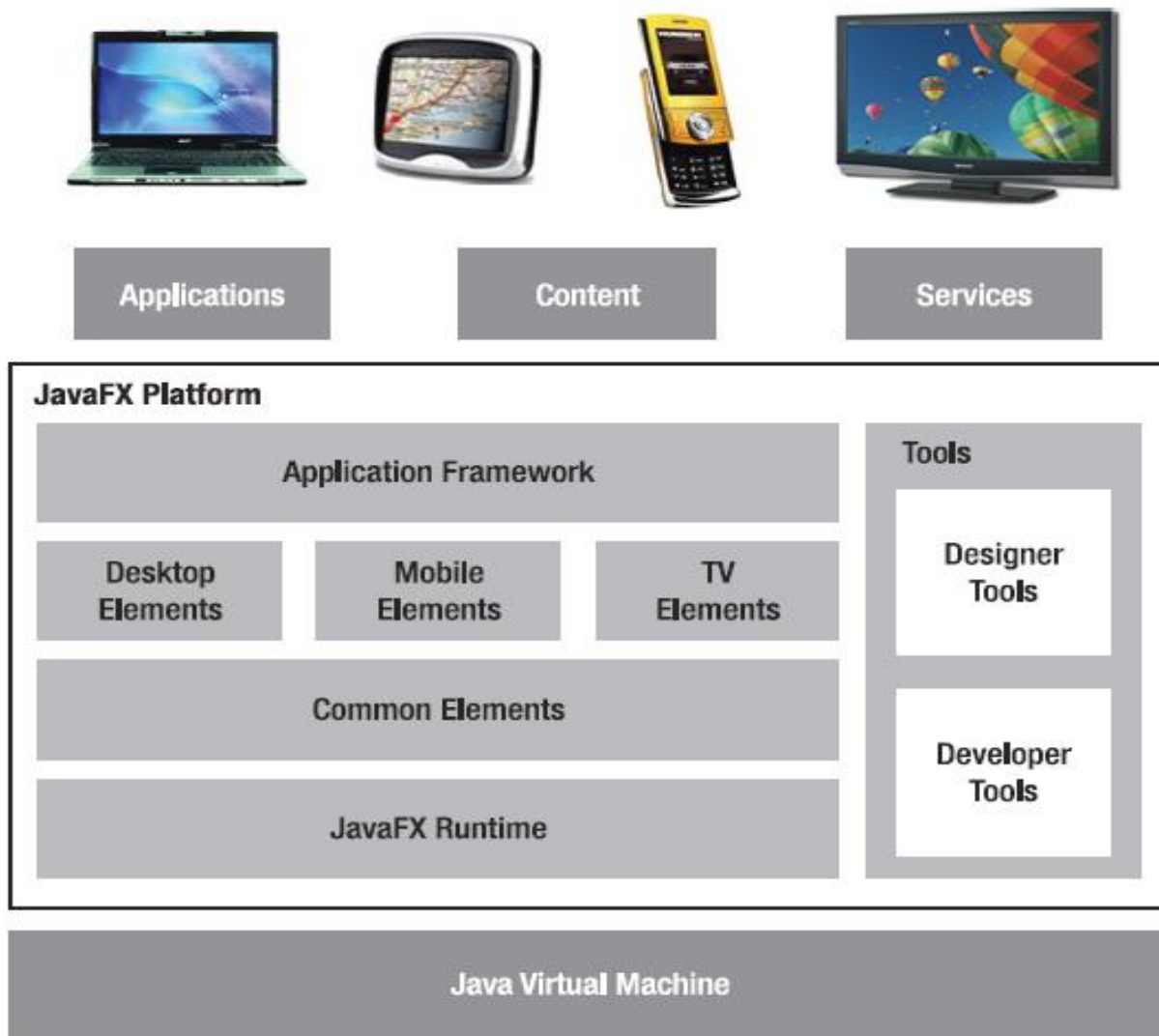
5 Създаване на съвременен интерактивен графичен интерфейс

JavaFX е съвременната технология за създаване на интерактивен потребителски интерфейс с Java с . Принадлежи на група от технологии(Adobe Flex, MS Silverlight и др.), които на основата на единна платформа и приложни програмни средства предоставят на потребителя богато мултимедийно съдържание.

Някои от основните **разлики** на JavaFX по отношение на Swing са:

- Използва **векторна графика**, което позволява запазване на качеството на графичния интерфейс при изобразяването му с различна разделителна способност на различни хардуерни платформи
- **Разделя** описанието на съдържанието на графичния интерфейс от дейностите по обработката на събитията в графичния интерфейс. Съдържанието на графичния интерфейс(**сцена**) се описва с XML-подобен език (FXML), който може да се създава и редактира с текстов редактор. Дейностите по обработка на събитията се програмират на Java(**контролер**)

5 Създаване на съвременен интерактивен графичен интерфейс

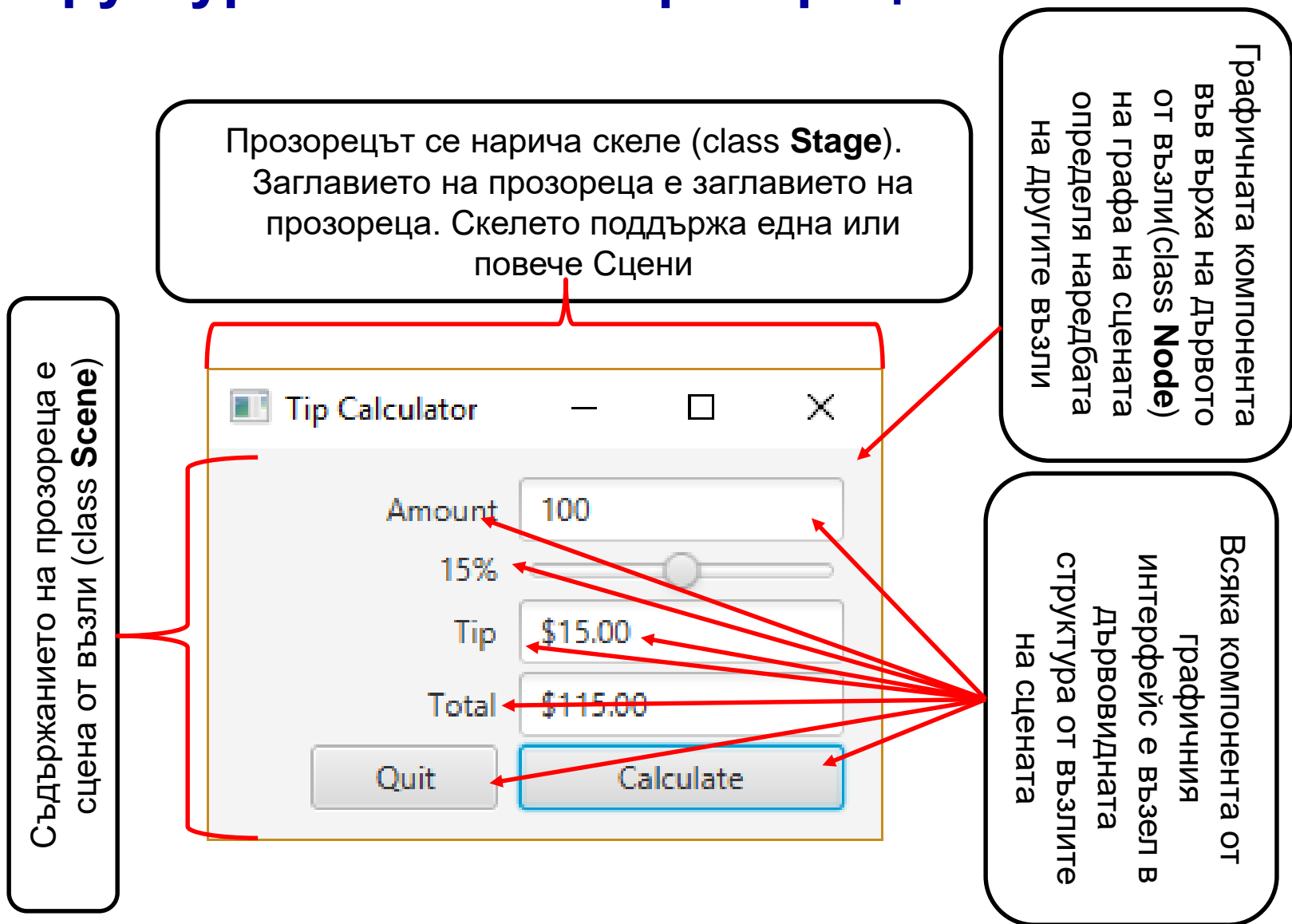


5 Създаване на съвременен интерактивен графичен интерфейс

От това произтичат следните предимства за JavaFX:

- **Улеснено приложение.** JavaFX предоставя една единствена платформа за създаване на GUI, графика и мултимедия
- Предоставя единно описание на графично съдържание на FXML език
- **Предоставя неограничени възможности за редактиране** на графичното съдържание с приложение на CSS
- **Подобрена производителност** при обработка сложни графични сцени(анимация, компютърни игри, видео)

5.1 Структура на JavaFX прозорец



5.1 Структура на JavaFX прозорец

- Прозорецът, в който се изпълнява приложението на JavaFX се разглежда като **рамка на скеле** (аналогично на “театрален подиум”) и е обект от `class Stage`
- В рамката на скелето може да се изобразява една или повече активна **сцена** (аналогична на “театрална сцена”), която дефинира **дървовидна структура** (аналогична на структурата на файловата система), във възлите на която са компонентите на графичния интерфейс. Сцената е обект от `class Scene`.
- Всяка графична компонента на сцената се нарича **Възел** и е обект от `class Node`. С изключение на възела в корена на дървото, всеки възел има точно един базов възел. Всеки възел има набор от **свойства** (текст, цвят, подредба) и **поведение** (методи за трансформации, визуални ефекти), **обработка на стандартни събития**.

5.1 Структура на JavaFX прозорец

- Възли, които могат да съдържат други възли, се наричат **Контейнери(пана) за подреждане** (`Layout containers`). Примери- `Pane`, `AnchorPane`, `GridPane`, `VBox`, `HBox`
- Възли, които не могат да съдържат други възли, се наричат **Контроли** (`Control`). Примери- `Button`, `Label`, `TextField`, `TextArea`
- Всяка графична компонента се нарича **Възел** и е обект от `class Node`. С изключение на възела в корена на дървовидната структура, всеки възел има точно един базов възел. Всеки възел има набор от **свойства** (текст, цвят, подредба) и **поведение** (методи за трансформации, визуални ефекти), **обработка на стандартни събития**.
- JavaFX дефинира дървото с възли в **текстов файл** с окончание `.fxml` като използва за описанието му **FXML**

5.1 Структура на JavaFX прозорец

Потребителят на графичния интерфейс поражда събития при взаимодействието си с графичните компоненти. Обработката на тези събития позволява да се създаде интерактивност на взаимодействието на потребителя със софтуерното приложение. Методът, който дефинира обработката на събитие, определя последователността от действия, които трябва да се извършат в отговор на това действие на потребителя. Този метод се нарича **Метод за обработка на събитие** (`event handler`) и има специфична дефиниция за всяко конкретно събитие.

- JavaFX дефинира методите за обработка на събития в Java клас, наречен, **Контролер**.

5.1 Структура на JavaFX прозорец

Създаването на GUI с JavaFX е аналогично на създаване на театрална постановка и се състои от следните по-важни стъпки:

- Създаване на обекта на **Скелето , т.е.** графичния прозорец. Това може да е графичен прозорец на десктоп приложение, уеб страница или върху таблет
- Създаване на **Сцена , т.е.** съдържанието на графичния прозорец с „действащи лица“ (**Възлите**), които ще взаимодействат помежду си и с потребителите (“зрителите”). Свойствата на Възлите позволяват да се създаде Сцена с богато визуално (мултимедийно) **съдържание**.
- Създаване на **Възлите**. Това са обекти, производни на `javafx.scene.Node` и включват UI контроли, различни форми, `Text`, картини и средства за възпроизвеждане на мултимедийно съдържание, както и потребителски дефинирани компоненти. Сцената се моделира като насочена **дървовидна структура от възли**.
- Създаване на променливи и класове за представяне на възлите в Сцената.

5.1 Структура на JavaFX прозорец

- ▶ Създаване на методи за обработка на събития като например, кликване с мишка, позволяващи на потребителите да взаимодействат с програмата
- ▶ Създаване на анимации и други мултимедийни ефекти

5.1 Структура на JavaFX прозорец

Подреждането на Възлите на Сцената се осъществява с т.нар.

Контейнери(пана) за подреждане (Layout containers/ panes). Те се представят с класове, по- често използваните от които, са следните:

-AnchorPane class- позволява на Възлите, които се съдържат в тях да се закотвят към четирите страни и центъра (top, bottom, left side, center) на четириъгълна област от Сцената.

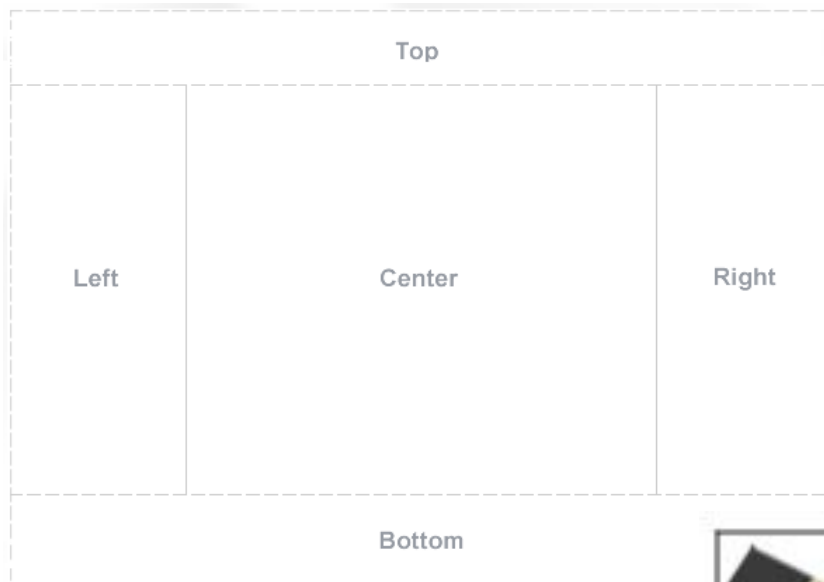
-GridPane class- позволява Възлите да се разполагат в гъвкава мрежа от редове и колони като е възможно възлите да се разпростират в няколко съседни клетки на тази мрежа.



-HBox class- позволява на възлите, съдържащи се в контейнер от този тип да се подреждат последователно в един хоризонтален ред.

-VBox class- позволява на възлите, съдържащи се в контейнер от този тип да се подреждат последователно в един вертикален ред.

5.1 Структура на JavaFX прозорец

Реализирането на GUI с JavaFX изисква да се приложи определен вид подреждане на компонентите от графичния интерфейс като се разполагат в стандартни **Контейнери(пана) за подреждане**



	Sales: Current Year		
	Goods and Services		
			Services 20%
Goods 80%			

5.2 Инсталиране на SceneBuilder

Предназначение на SceneBuilder

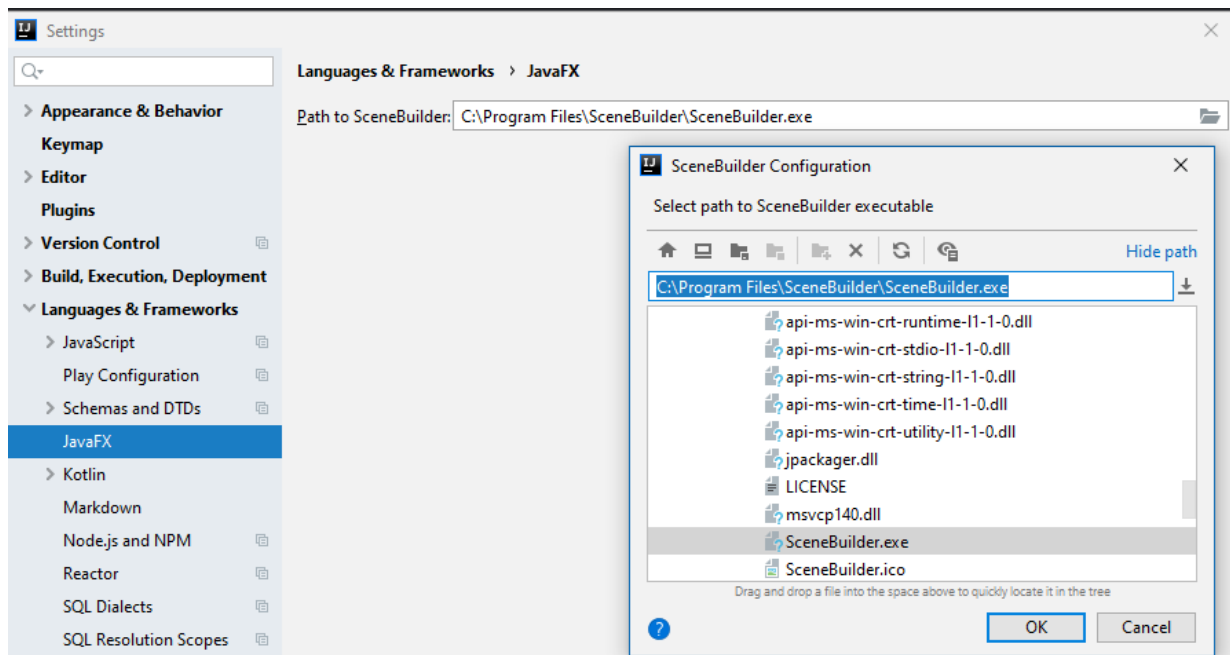
-Графична среда за моделиране на графичния интерфейс и представяне в FXML на скелета на дървото от възли

-Инсталиране

1.Изпълнява се **Windows Installer** (x64 /x86)

2.Конфигурира се в IntelliJ като се използва **Ctrl-ALT-S**

File> Settings ->



5.3 Създаване на JavaFX приложение

Описано е подробно в приложения файл

SetupJFXNonModularProjectWithJDK13IntelliJ.pdf

The screenshot displays the IntelliJ IDEA interface. The 'Projects' tab is active, showing a project named 'Add3JavaFX'. Under 'Source Packages', there is a package 'add3javafx' containing three files: 'Add3JavaFX.java', 'FXMLDocument.fxml', and 'FXMLDocumentController.java'. Below the source packages, there is a 'Libraries' section. A red arrow points from the 'Add3JavaFX.java' file to a callout box. Another red arrow points from the 'FXMLDocumentController.java' file to a callout box. A third red arrow points from the application window preview to a callout box. The application window preview shows a button labeled 'Click Me!' and the text 'Hello World!'.

Изпълним файл на приложението. Създава обектите от графичния интерфейс и ги изобразява (FX main class)

Текстово представяне на структурата на **Сцената** във FXML формат

Контролерът (клас на Java) който съдържа референции към възли и методите за обработка на събития, в които участват тези референции

Изпълнение на проекта по подразбиране

5.4 Описание на дървото от възли

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <?import java.lang.*?>
4 <?import java.util.*?>
5 <?import javafx.scene.*?>
6 <?import javafx.scene.control.*?>
7 <?import javafx.scene.layout.*?>
8
9 <AnchorPane id="AnchorPane" prefHeight="200" prefWidth="320" xmlns:fx="http://
10     fx:controller="add3javafx.FXMLDocumentController">
11     <children>
12         <Button layoutX="126" layoutY="90" text="Click Me!" onAction="#handleButtonAction" fx:id="button" />
13         <Label layoutX="126" layoutY="120" minHeight="16" minWidth="69" fx:id="label" />
14     </children>
15 </AnchorPane>
```

Visual representation of the FXML document:

Click Me!
Hello World!

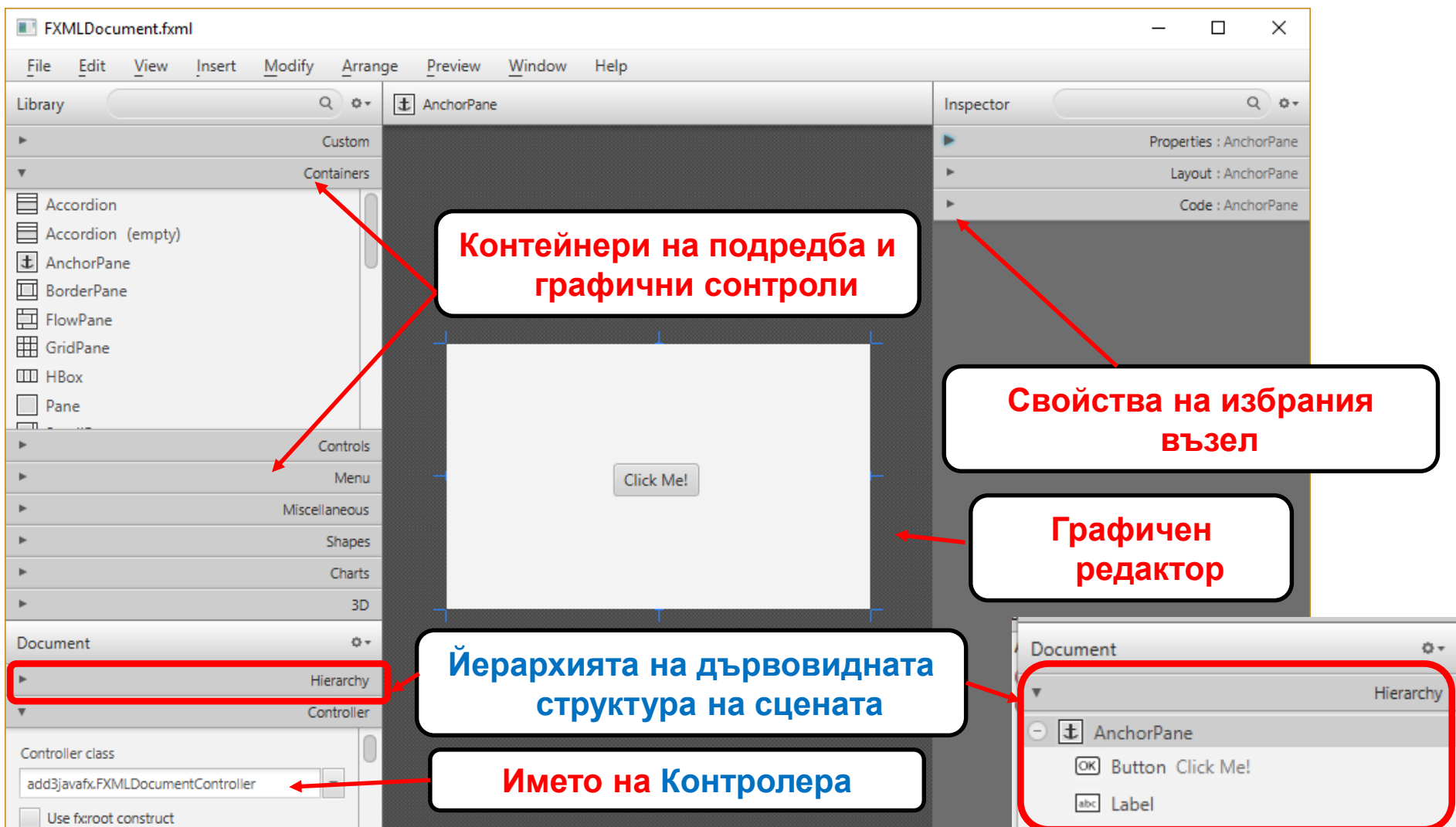
Контейнер на подредба (възел)
в корена на дървовидната
структура на Сцената

Графични Контроли(възли),
вложени в корена на
дървовидната структура на
Сцената

Важно: Този стринг трябва да
съвпада с името на
Контролера на приложението

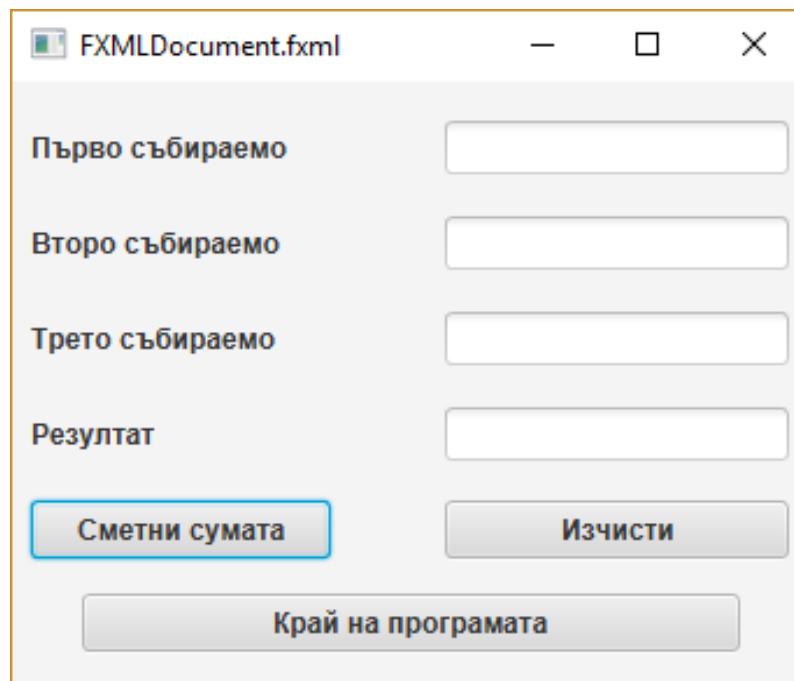
5.5 Изграждане на графичния интерфейс

Клик с десен бутон върху FXML файла и избиране на Open от помощното меню (или двойно кликване с ляв бутон) отваря SceneBuilder



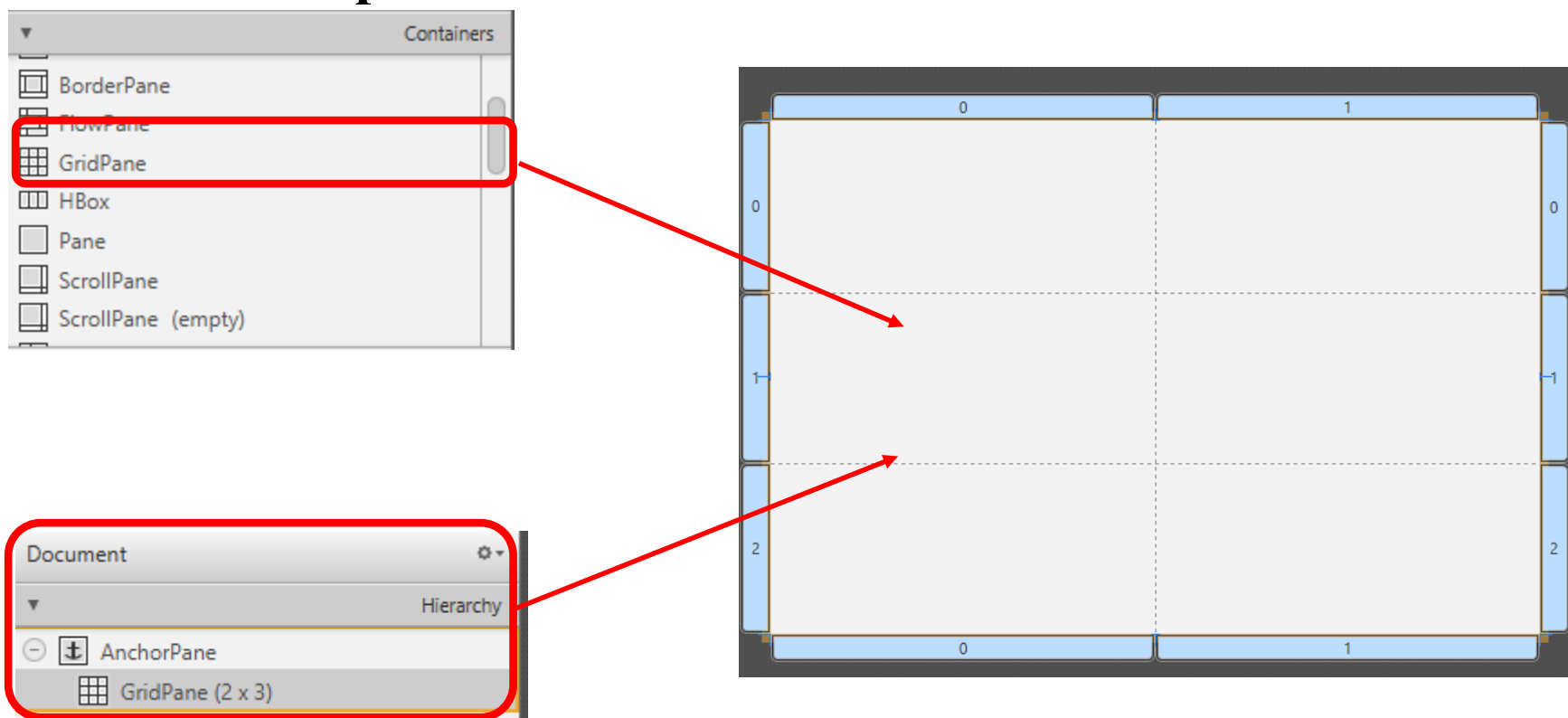
5.5 Изграждане на графичния интерфейс

1. Отваряме Hierarchy и с Ctrl-Click избираме ненужните контроли Button Label в подразбиращия се модел. Натискаме клавиш Del или с Десен бутон и Delete от помощното меню **изтриваме тези контроли**.
2. Целта е да създадем следния модел на графичен интерфейс



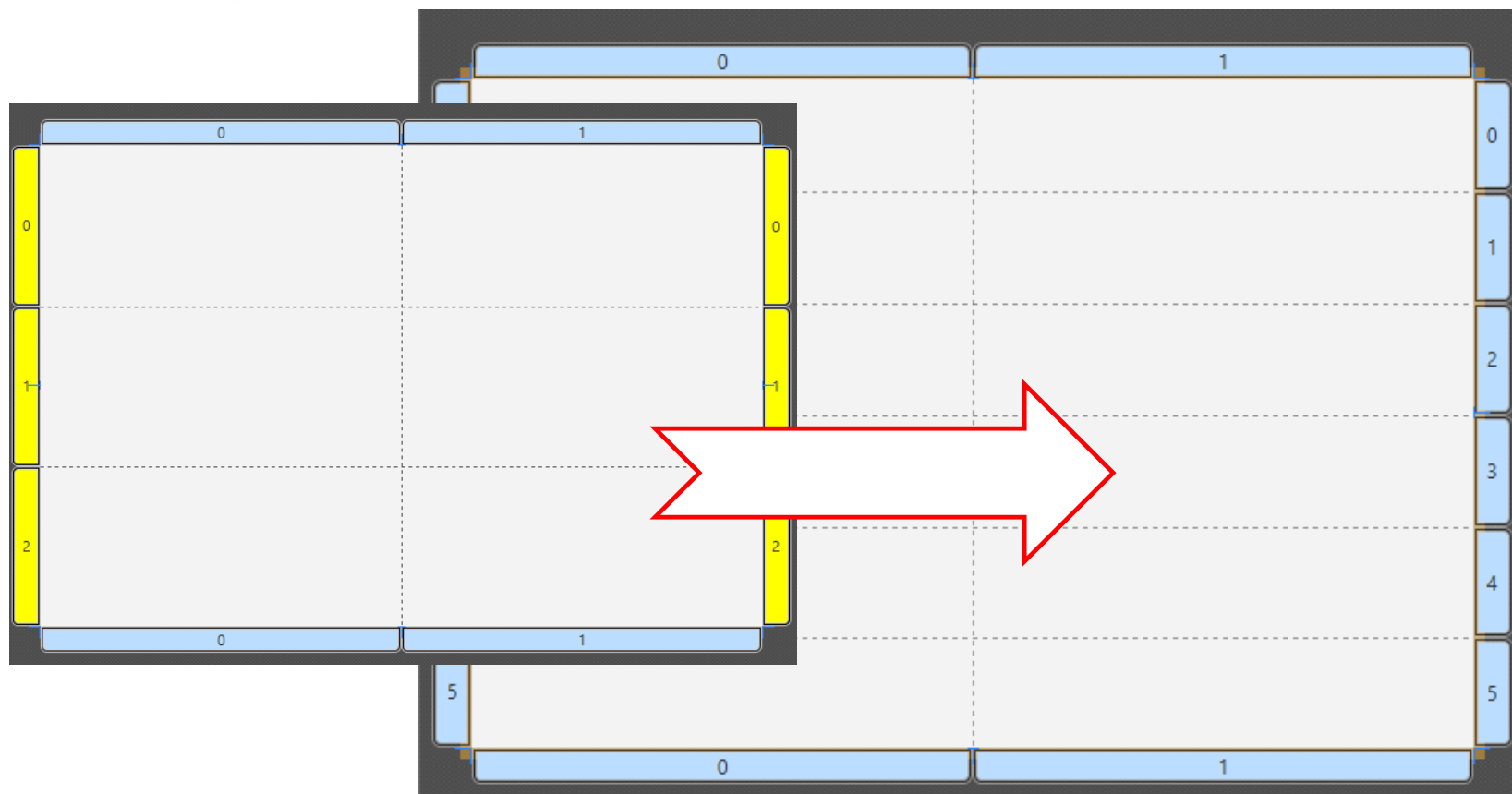
5.5 Изграждане на графичния интерфейс

3. Забелязваме, че изискваният дизайн е изграден от 5 реда и две колони. Затова дърпаме `GridPane` от Контейнерите на подредба в раздела `Containers` върху `AnchorPane`, така че да покрие `AnchorPane`



5.5 Изграждане на графичния интерфейс

4. Добавяме още три реда към грида. За целта избираме последните три негови реда (краищата се оцветяват в жълто). С десен бутон избираме от помощното меню **Add row below**

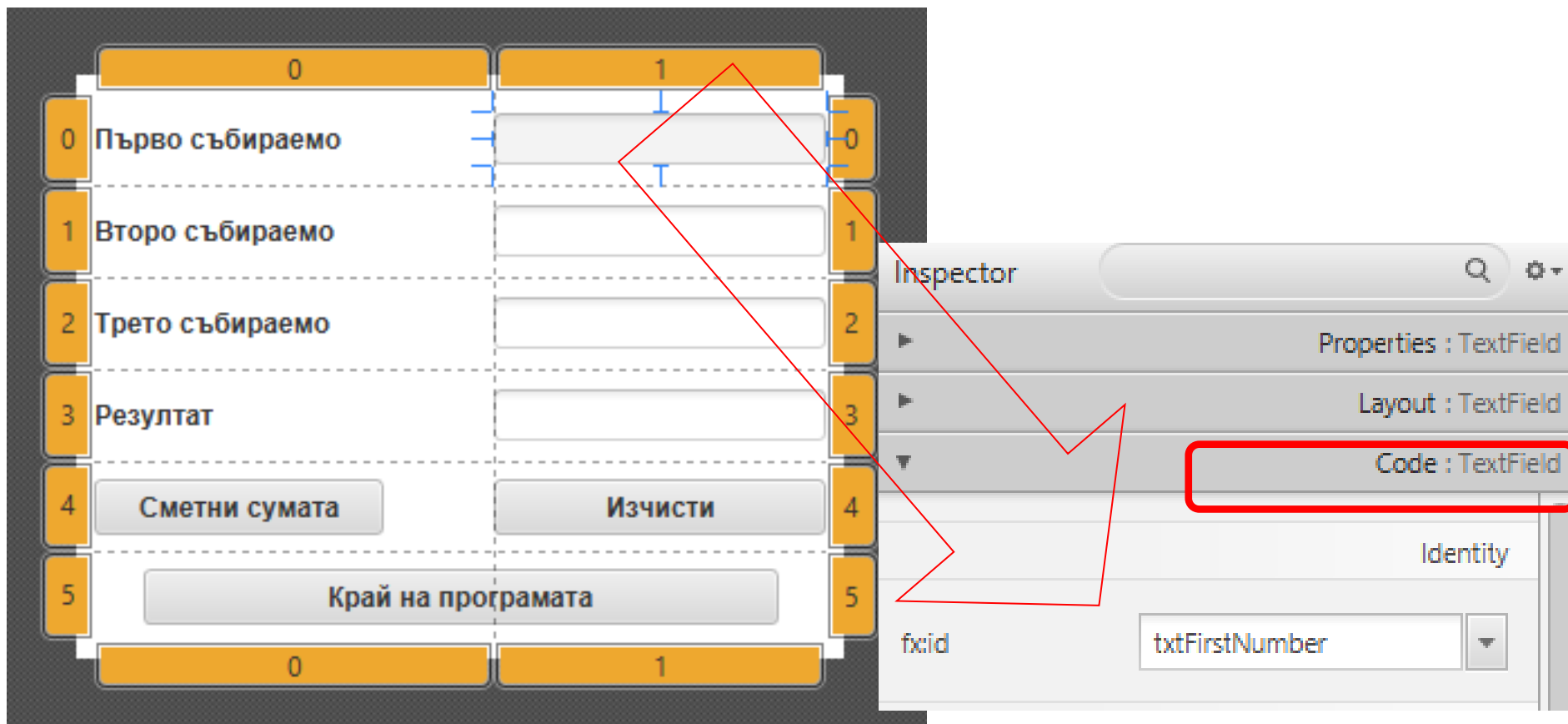


5.5 Изграждане на графичния интерфейс

5. Съобразно изисквания модел на графичния интерфейс дърпаме в съответните клетки **Контроли**- `Label`, `TextField` и `Button`.
6. За всяка от контролите с двойно кликване задаваме необходимия им текст.
7. За всяка от контролите **прилагаме именуване на променлива, с която ще се реферират съобразно Изменената Унгарска нотация.**

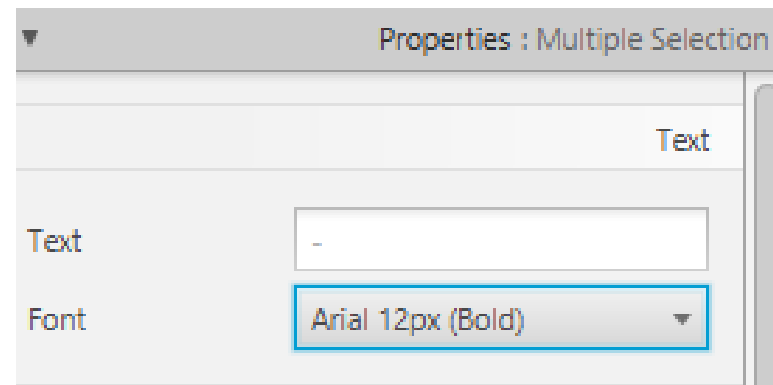
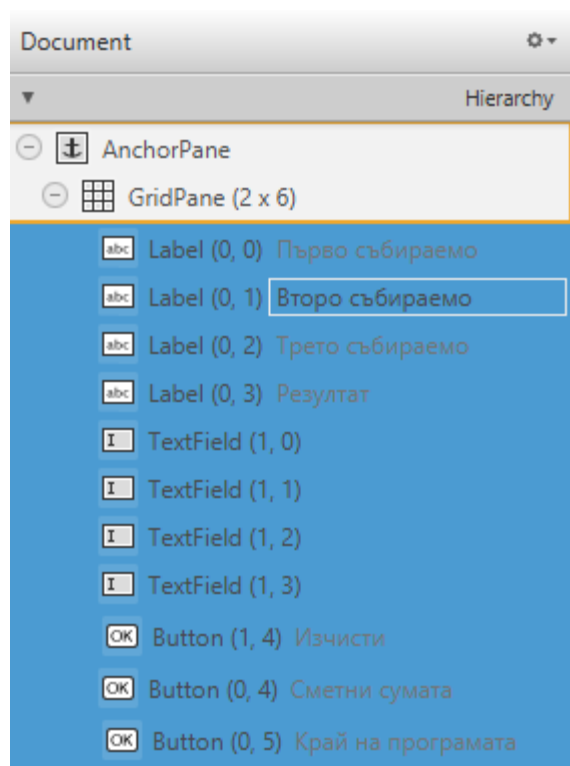
Примерно, при избиране на първото текстово поле, въвеждаме в Групата `Code` от свързаните с него свойства, стойност `txtFirstNumber` за `fx:id`

5.5 Изграждане на графичния интерфейс



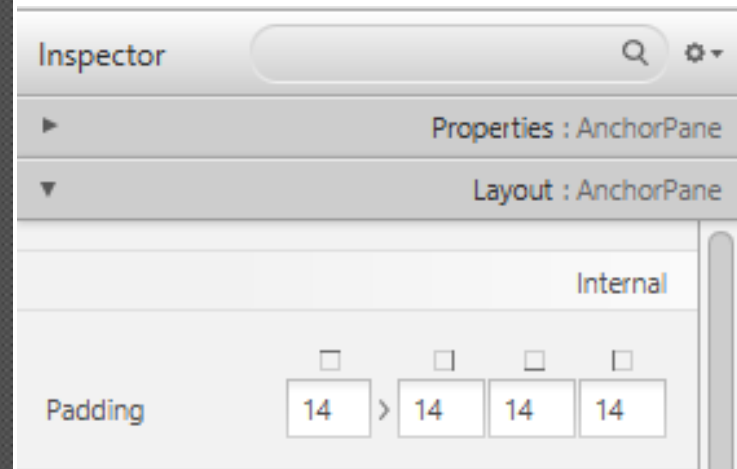
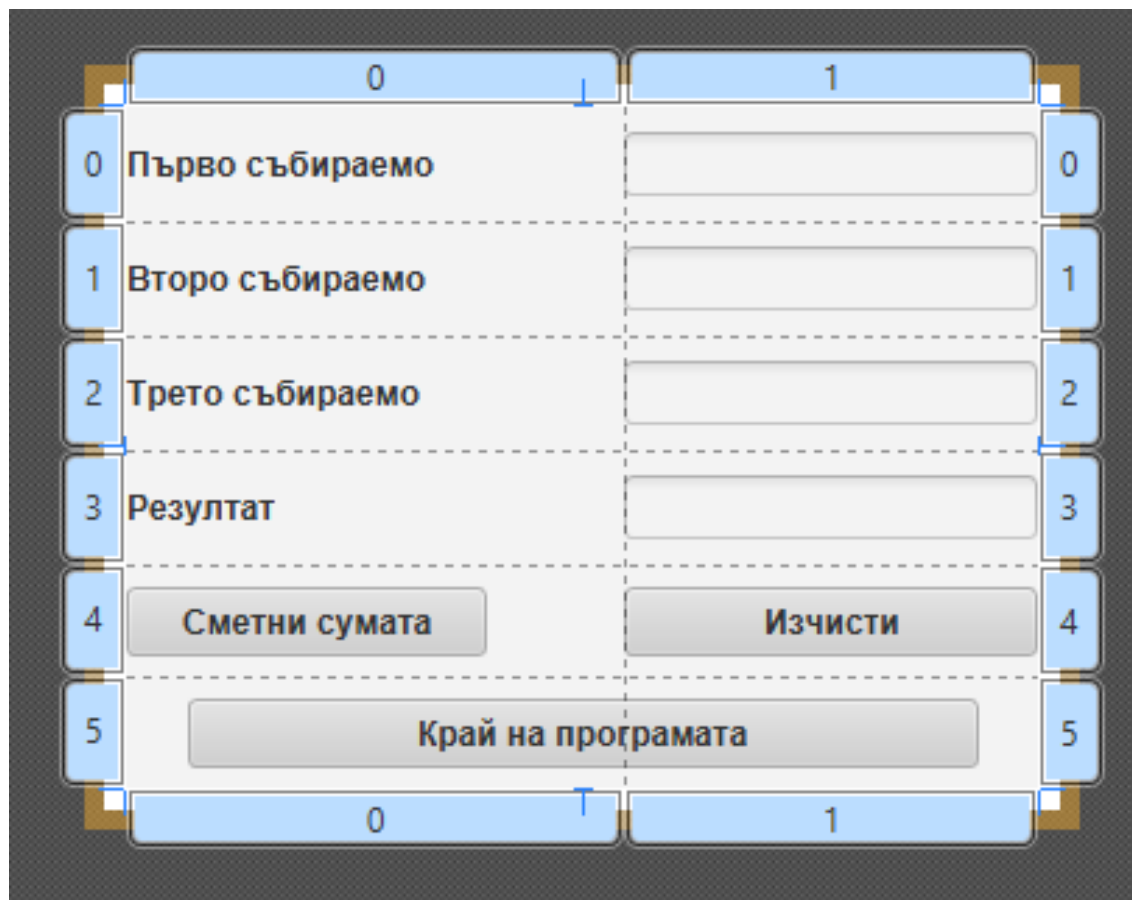
5.5 Изграждане на графичния интерфейс

6. Накрая, с натиснат бутон **Ctrl** избираме последователно (с ляво кликване) контролите от грида, както те са изписани в **Hierarchy**. За така избраната група от контроли задаваме стил и големина на шрифта в **Properties**.



5.5 Изграждане на графичния интерфейс

7. Допълнително, задаваме `Padding 14 px` на `AnchorPane` от рамката на прозореца, подравняване вдясно за текстовите полета и стойностите им по подразбиране.



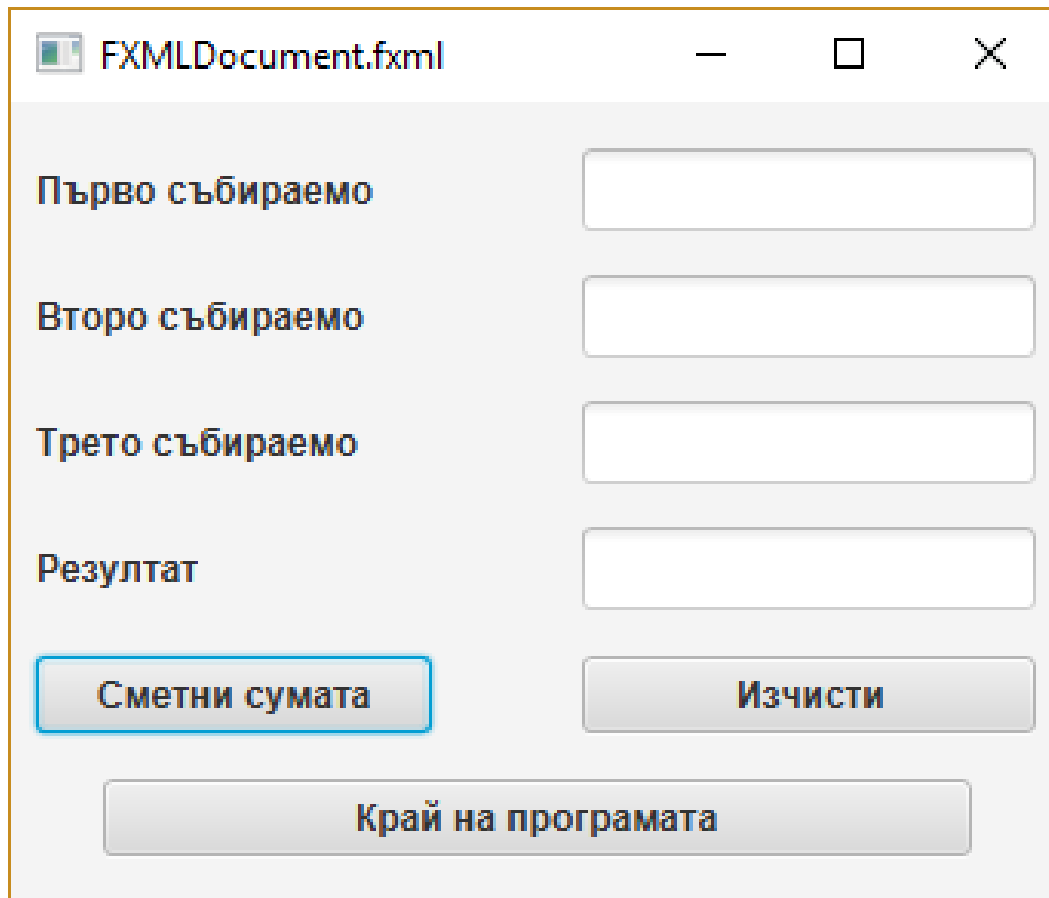
5.5 Изграждане на графичния интерфейс

Важно:

Стойност от 14 px за четирите стойности (TOP, RIGHT, BOTTOM и LEFT) на `Padding` свойството е **препоръчаното отстояние на контрола** от границите на `Scene` в JavaFX.

5.5 Изграждане на графичния интерфейс

8. **Ctrl-P (Preview-> Show Preview in Window) тестваме получения дизайн .**



The screenshot shows a Java Swing window titled "FXMLDocument.fxml". The window has a standard title bar with minimize, maximize, and close buttons. The main content area is light gray and contains four text input fields arranged vertically. The labels for these fields are "Първо събираемо", "Второ събираемо", "Трето събираемо", and "Резултат". Below the input fields, there are three buttons: "Сметни сумата" (highlighted with a blue border), "Изчисти", and "Край на програмата".

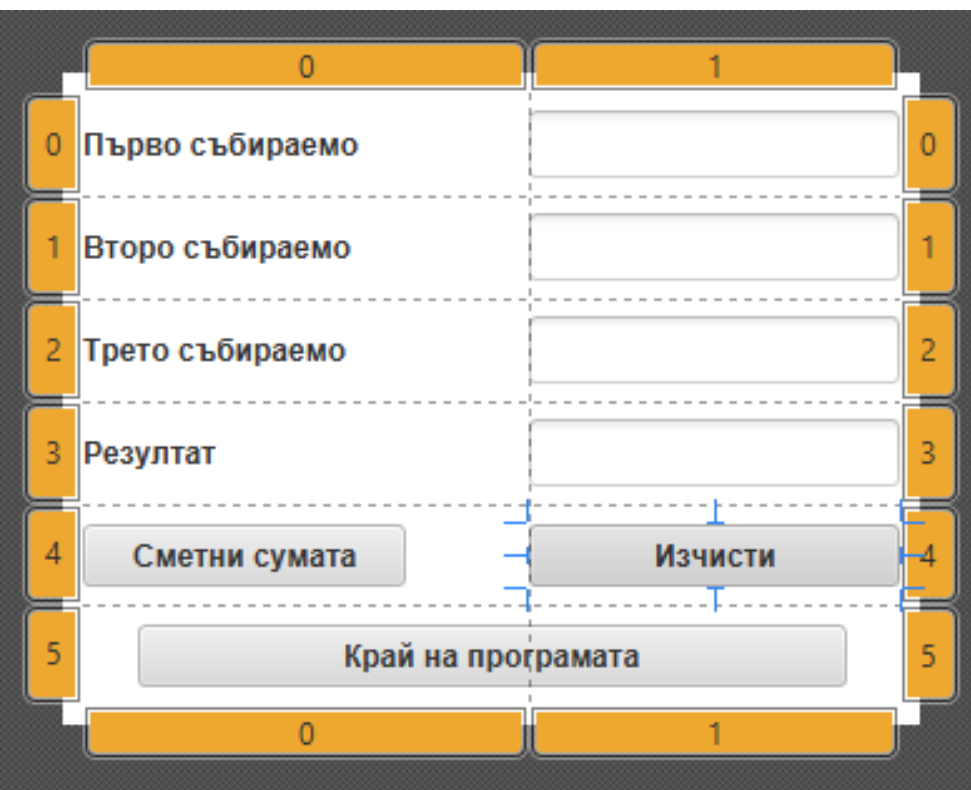
5.5 Изграждане на графичния интерфейс

9. Изпълняваме `File->Save`, с което записваме структурата на дървото от контроли на сцената във `FXML` формат във файла `FXMLDocument.fxml` от проекта.

5.6 Въведение в обработка на събития

1. Тук ще се ограничим обработка на събитието `Action`. Това събитие възниква при **кликване върху бутон, натискане на Return в избрано текстово поле или област, както и при избиране на опция от меню.**
 - ✓ Избираме бутон и полето `OnAction` в групата `Code` от свързаните с него свойства въвеждаме името на метода за обработка на това събитие. Използваме за яснота следния формат за име на този метод
<идентификатор на бутона>OnAction
Наименованието на този метод за бутон с идентификатор `btnClear` ще бъде
btnClearOnAction

5.6 Въведение в обработка на събития



Properties : Button

Layout : Button

Code : Button

Identity

fxid

Main

On Action

#

DragDrop

On Drag Detected

#

On Drag Done

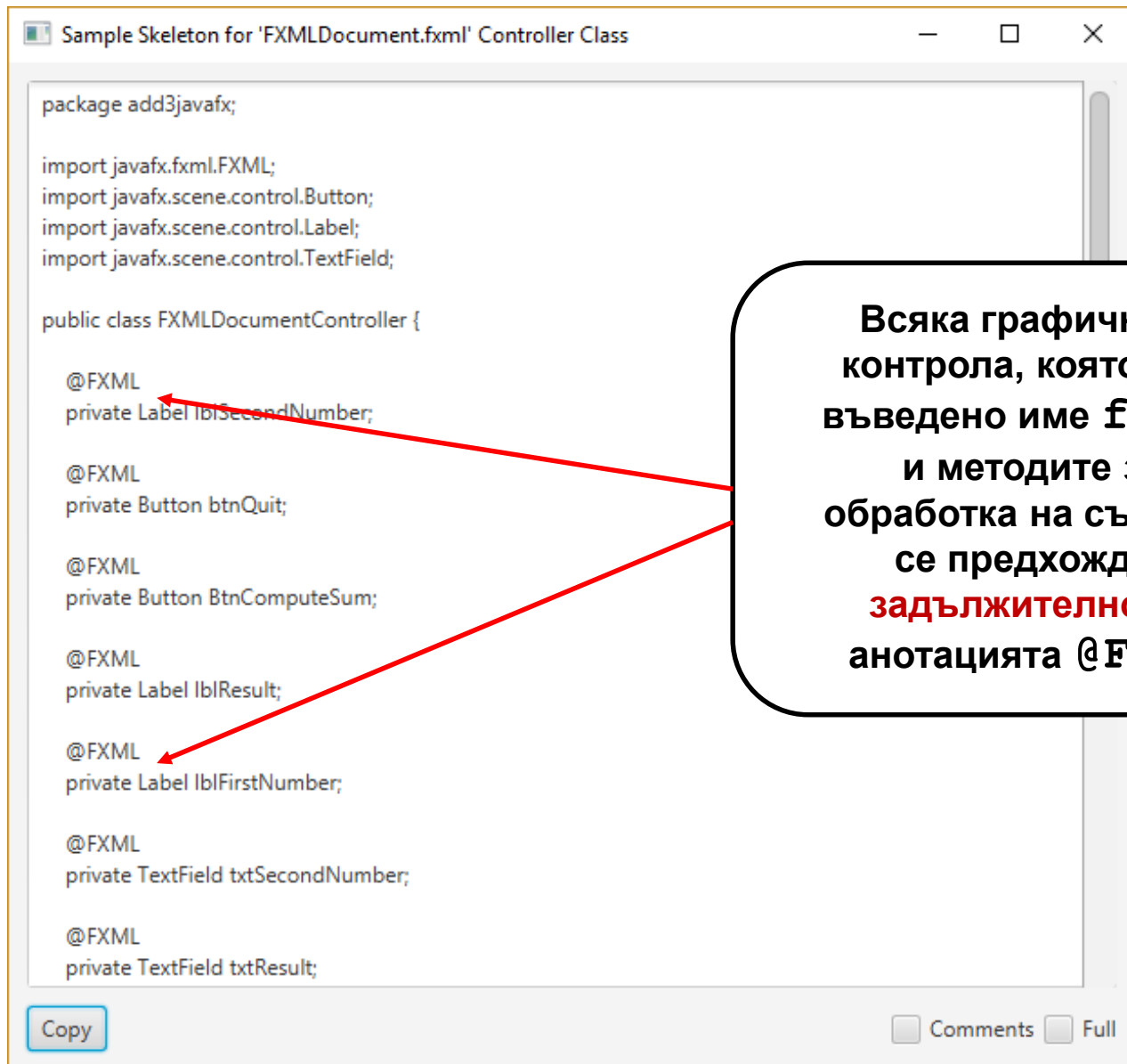
5.6 Въведение в обработка на събития

- ✓ Дефинираме по същия начин методите за обработка на събитието `Action` и за останалите бутони

В резултат се създава скелета на **Контролера** на **Сцената**.

Текстът на този клас е достъпен в SceneBuilder View-> Show Sample Controller Skeleton

5.6 Въведение в обработка на събития



5.6 Въведение в обработка на събития

2. Копираме пълния текст на скелета на Контролера на Сцената (**с опцията Full**) и презаписваме с него текущото съдържание на Контролера на Сцената в проекта на NetBeans (**SceneBuilder** не актуализира автоматично съдържанието на този файл при промени на свойства от Групата Code)
3. С десен бутон върху сорс кода на Контролера на Сцената **в проекта от NetBeans** изберете **Fix Imports**, за да се добавят липсващи пакети на класове на контроли, участващи в Сцената.
4. Компилирайте приложението

5.7 Методи за обработка на събитие

Методите за обработка на събитието `Action` се пишат като се използват `get...()` и `set...()` методите на свойствата на възлите в графичния интерфейс. Това е една добра причина този материал да се използва за затвърждаване на уменията за боравене с тези методи.

```
@FXML
```

```
void btnQuitOnAction(ActionEvent event) {  
    System.exit(0); // Terminate the JavaFX application  
}
```

5.7 Методи за обработка на събитие

@FXML

```
void btnClearOnAction(ActionEvent event) {  
    txtFirstNumber.setText("0");  
    txtSecondNumber.setText("0");  
    txtThirdNumber.setText("0");  
    txtResult.setText("0");  
}
```

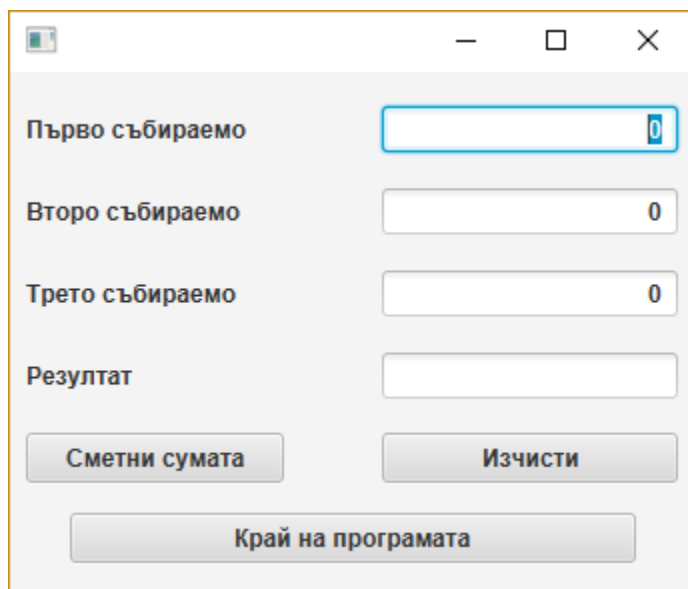
@FXML

```
void btnComputeSumOnAction(ActionEvent event) {  
    int firstNum = Integer.parseInt(txtFirstNumber.getText());  
    int secondNum = Integer.parseInt(txtSecondNumber.getText());  
    int thirdNum = Integer.parseInt(txtThirdNumber.getText());  
    // b)  
    int sum = firstNum + secondNum + thirdNum;  
    // c)  
    txtResult.setText(String.format("%d", sum));  
}
```


5.7 Изпълнение на програмата

```
public class Add3JavaFX extends Application {  
    @Override  
    public void start(Stage stage) throws Exception {  
        // Прочита дървовидната структура на Сцената  
        Parent root = FXMLLoader.load(getClass().getResource("FXMLDocument.fxml"));  
        // Създава обект от Сцената, инициализира обектите в Сцената  
        Scene scene = new Scene(root);  
  
        stage.setTitle("Add 3 numbers");// Задава заглавие на Постановката  
        stage.setScene(scene);           // Зарежда Сцената в Постановката  
        stage.show();                    // Показва Сцената в графичния прозорец  
    }  
    /**  
     * Начална точка за изпълнение на JavaFX приложение  
     */  
    public static void main(String[] args) { launch(args); }  
}
```

5.7 Методи за обработка на събитие

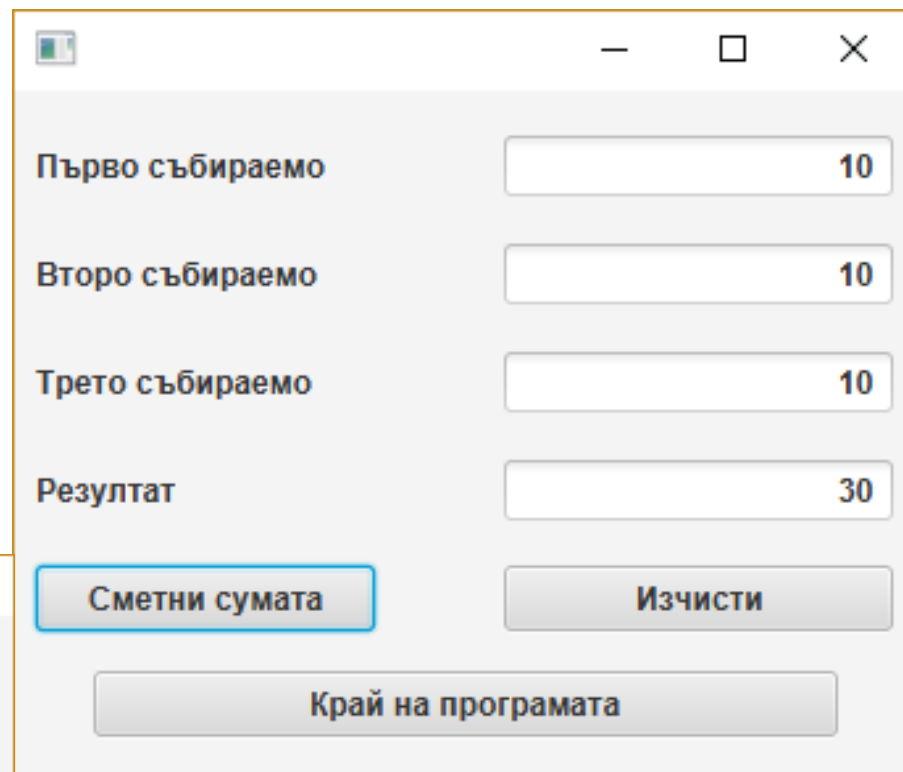


Първо събираемо

Второ събираемо

Трето събираемо

Резултат

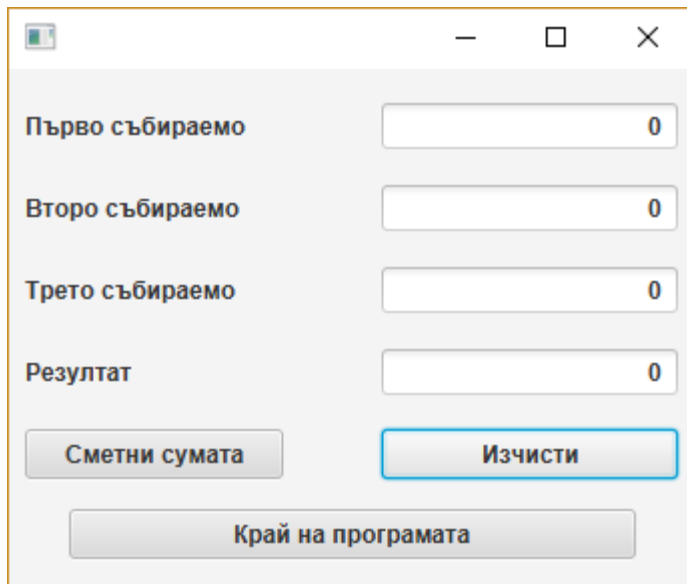


Първо събираемо

Второ събираемо

Трето събираемо

Резултат



Първо събираемо

Второ събираемо

Трето събираемо

Резултат

5.8 Диалогов прозорец

Диалоговите прозорци имат за цел да изведат съобщение или позволяват да се въведат данни в графичен режим. Те спират достъпа на потребителя до останалата част от графичния интерфейс, докато тези прозорци не бъдат затворени. Веднъж, след като бъдат показани, се изчаква отговор от потребителя и след това се обработва полученият отговор.

Различаваме следните основни видове диалогови прозорци:

- Диалогов прозорец за извеждане на съобщение
- Диалогов прозорец за въвеждане на данни

```
// Simple dialog window for displaying messages  
private static Alert alertInfo = new Alert(AlertType.INFORMATION);  
// Dialog window for displaying messages with confirm and reject options  
private static Alert alertConfirm = new Alert(AlertType.CONFIRMATION);  
// Input text dialog window  
private static TextInputDialog dialog = new TextInputDialog();
```

5.8a Диалогов прозорец за съобщения

Диалоговите прозорци за извеждане на съобщения се създават с помощта на компонентата **Alert** в **JavaFX**

Добра практика е тази компонента и нейните свойства да се инициализират в статичен метод, който да се изпълнява всеки път, когато е нужен такъв диалогов прозорец.

```
//ClassName.messageDialog("YOUR INFORMATION HERE", "TITLE BAR MESSAGE", "HEADER MESSAGE");
```

```
// Info message dialog with a header
```

```
public static void messageDialog(String infoMessage,  
                                String titleBar, String headerMessage) {
```

```
    alertInfo.setTitle(titleBar);  
    alertInfo.setHeaderText(headerMessage);  
    alertInfo.setContentText(infoMessage);  
    alertInfo.showAndWait();
```

```
}
```

```
//ClassName.messageDialog("YOUR INFORMATION HERE", "TITLE BAR MESSAGE");
```

```
// Info message dialog without a header
```

```
public static void plainMessageDialog(String infoMessage, String titleBar) {
```

```
    /* By specifying a null headerMessage String, we cause the dialog to  
       not have a header */
```

```
    messageDialog(infoMessage, titleBar, null);
```

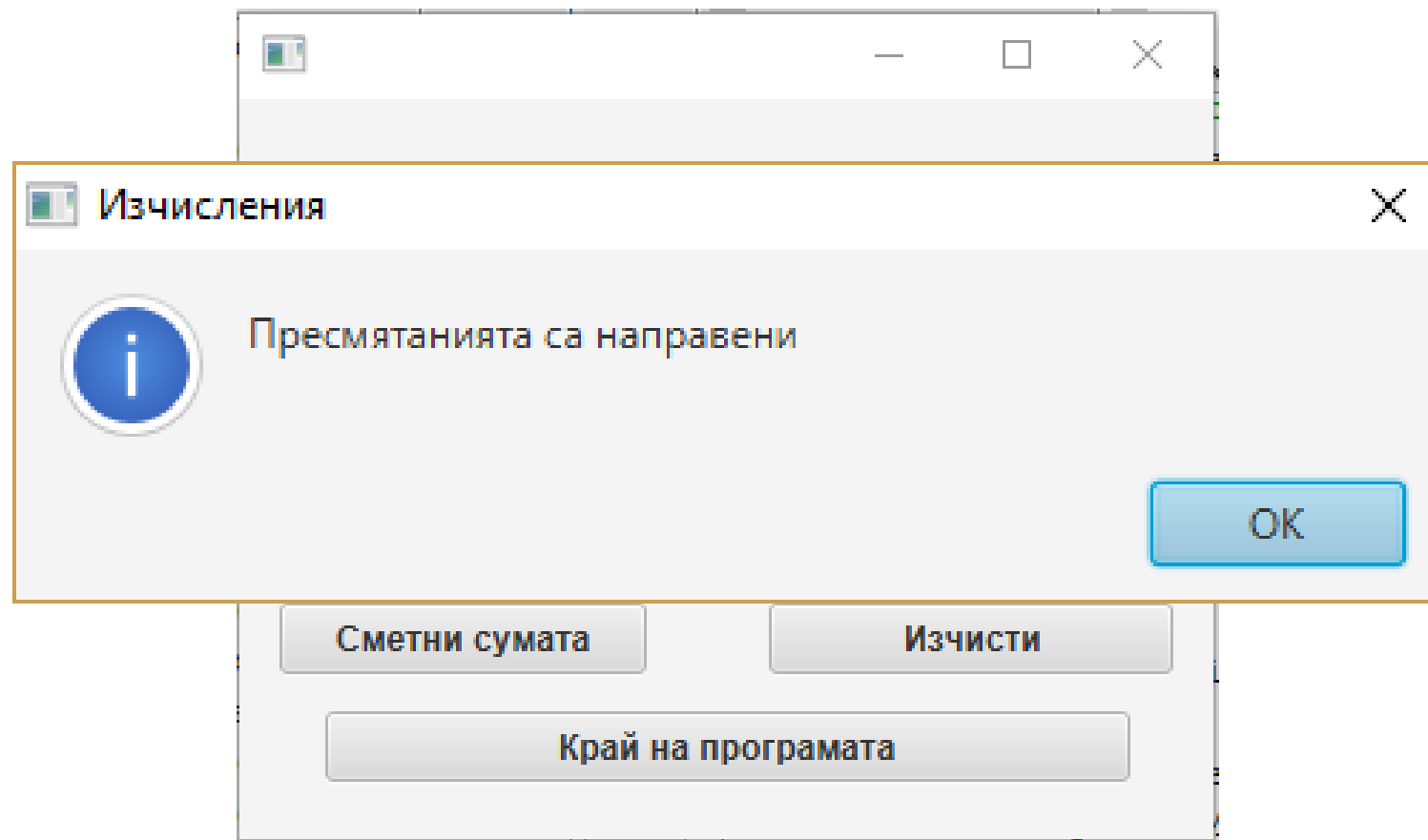
```
}
```

5.8a Диалогов прозорец за съобщения

За илюстрация, променяме предходния проект да извежда диалогови прозорци при натискане на бутоните. Статичните методи на диалоговите прозорци реализираме в Application класа на JavaFX приложението.

```
@FXML
void btnComputeSumClicked(ActionEvent event) {
    int firstNum = Integer.parseInt(txtFirstNumber.getText());
    int secondNum = Integer.parseInt(txtSecondNumber.getText());
    int thirdNum = Integer.parseInt(txtThirdNumber.getText());
    // b)
    int sum = firstNum + secondNum + thirdNum;
    // c)
    txtResult.setText(String.format("%d", sum));
    String infoMessage = "Пресмятанията са направени";
    String titleMessage = "Изчисления";
    Add3JavaFX.showMessageDialog(infoMessage, titleMessage);
}
```

5.8a Диалогов прозорец за съобщения



5.8a Диалогов прозорец за съобщения

Диалоговият прозорец за извеждане на съобщения може и да се управлява в зависимост от бутона избран за отговор от потребителя. При затваряне диалоговият прозорец връща `ButtonType.OK` или `ButtonType.CANCEL` в зависимост от това кой бутон е натиснат.

```
// Info message dialog with a confirmation
public static ButtonType confirmDialog(String infoMessage,
                                       String titleBar, String headerMessage) {
    alertConfirm.setTitle(titleBar);
    alertConfirm.setHeaderText(headerMessage);
    alertConfirm.setContentText(infoMessage);
    //Optional<ButtonType> result = alertConfirm.showAndWait();
    var result = alertConfirm.showAndWait();
    return result.get();
}

//
//      if (result.get() == ButtonType.OK) {
//          // ... user chose OK
//      } else { //result.get() == ButtonType.CANCEL
//          // ... user chose CANCEL or closed the dialog
//      }
//
```

5.8a Диалогов прозорец за съобщения

Ключовата дума

var

Замества изписване на тип на локална данна, когато компилаторът може да се „досети“ за този тип по присвоената стойност на тази данна

- Не може да ползва за означаване на тип на стойност, връщана при изпълнение на метод

// **wrong**!!

```
public var method() { return 1 ;}
```

- Не може да ползва за деклариране на параметър на метод

// **wrong**!!

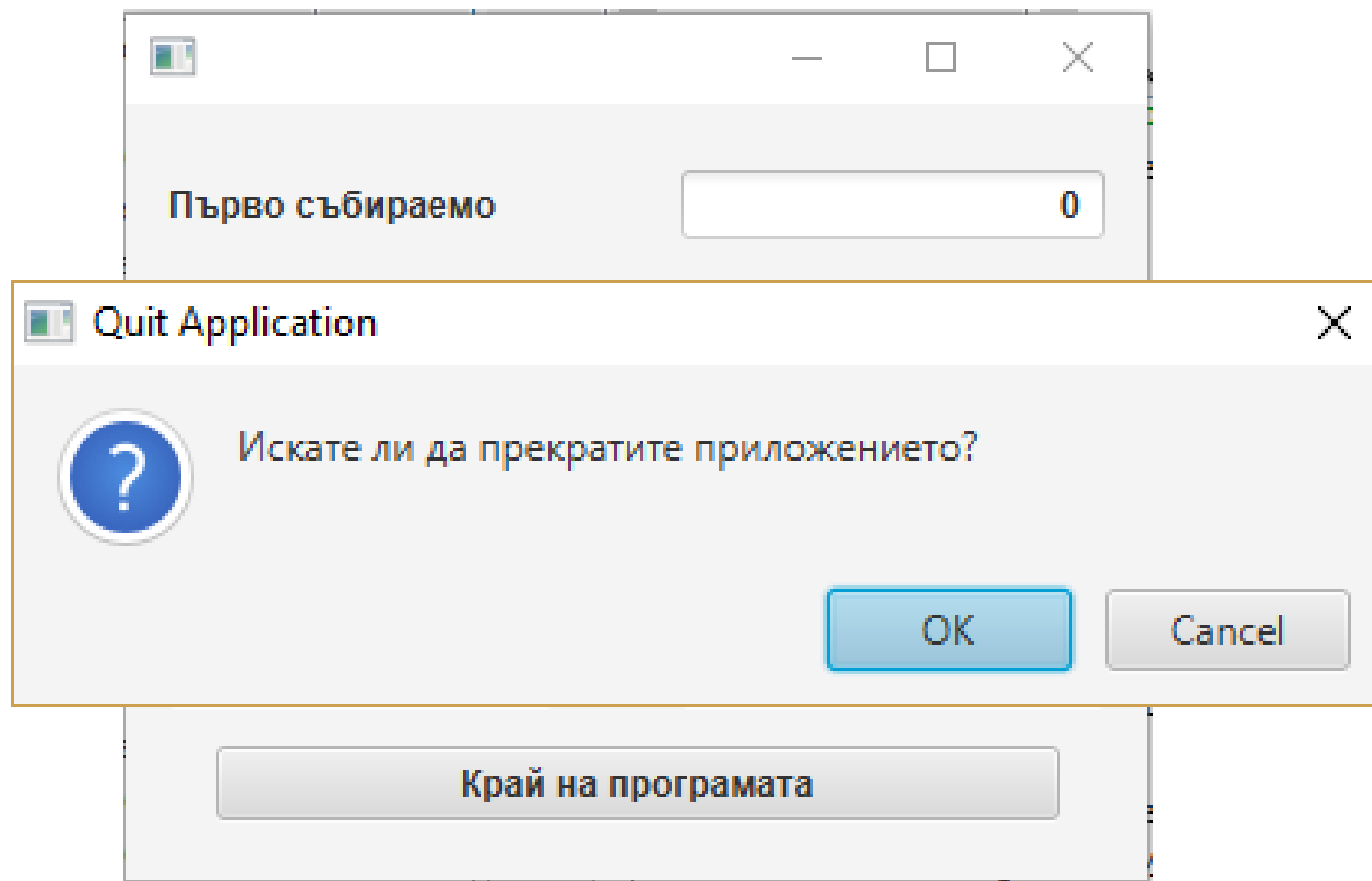
```
public void method(var number) {  
var string; // also wrong }
```


5.8а Диалогов прозорец за съобщения

Проверката на типа на натиснатия бутон позволява да се управлява логиката на изпълнение на програмата.

```
@FXML
void btnQuitClicked(ActionEvent event) {
    String quitApp = "Искате ли да прекратите приложението?";
    String quitHeader ="Quit Application";
    if( ButtonType.OK == Add3JavaFX.confirmDialog(quitApp, quitHeader, null)){
        Platform.exit();
    }
}
```

5.8a Диалогов прозорец за съобщения



5.8b Диалогов прозорец за въвеждане

Диалоговият прозорец за въвеждане на данни се създава с компонентата `TextInputDialog`. Тя има същите свойства за инициализиране, както компонентата `Alert`. По-конкретно, свойството `ContentText` е подканящият текст за въвеждане на данни в тази компонента

```
public static String inputDialog(String infoMessage, String titleBar,  
                                String headerMessage) {  
  
    dialog.setTitle(titleBar);  
    dialog.setHeaderText(headerMessage);  
    dialog.setContentText(infoMessage);  
    // Traditional way to get the response value.  
    // Optional<String> result = dialog.showAndWait();  
    var result = dialog.showAndWait();  
    if (result.isPresent()) {  
        return result.get();  
    }  
    else  
        return null;  
}
```

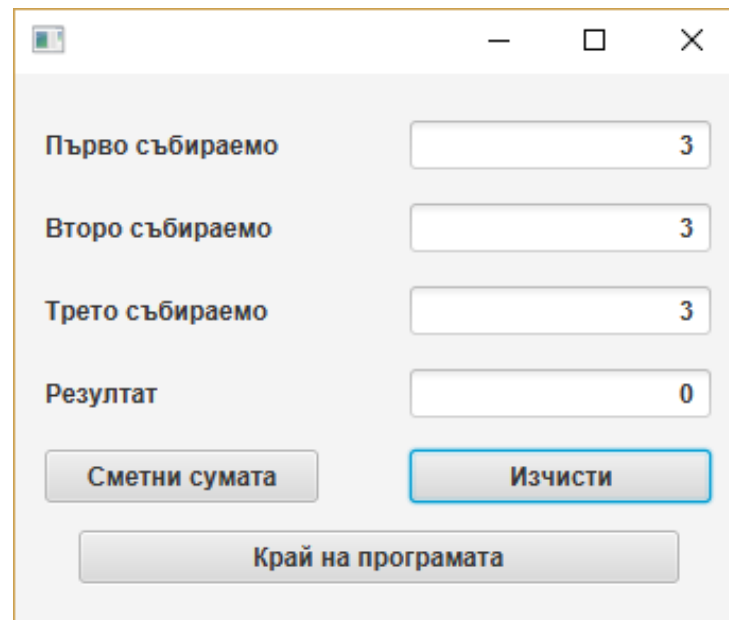
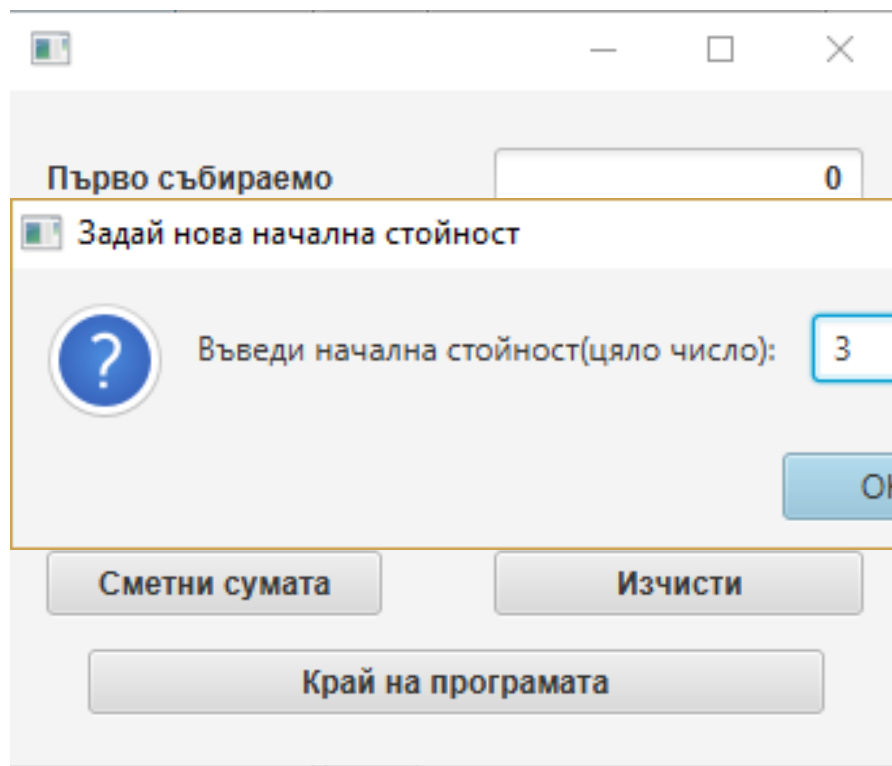
5.8b Диалогов прозорец за въвеждане

Диалоговият прозорец връща `String` или `null` в зависимост от това дали е въведена данна в текстовото поле на диалоговия прозорец.

Забележка: При изпълнение `setText(null)` свойството `text` не се променя.

```
@FXML
void btnClearClicked(ActionEvent event) {
    String infoMessage = "Въведи начална стойност(цяло число): ";
    String titleMessage = "Задай нова начална стойност";
    String input = Add3JavaFX.inputDialog(infoMessage, titleMessage, null);
    txtFirstNumber.setText(input);
    txtSecondNumber.setText(input);
    txtThirdNumber.setText(input);
    txtResult.setText("0");
}
```

5.8b Диалогов прозорец за въвеждане



5.8с Диалогов прозорец за съобщения

Диалоговите прозорци могат да се изпълняват само като част от JavaFX приложение. Когато е нужно такъв прозорец да се отвори в Конзолно приложение е необходимо да се изпълни метод за стартиране на диалоговия прозорец като част от JavaFX. Например, ако статичните методи за създаване на диалоговите прозорци са *написани в class*

LogicComputationTest, то в конзолно приложение тези статични методи трябва да се изпълнят с помощта на метода *PlatformImpl.startup()* по следния начин

```
PlatformImpl.startup(() -> {  
    LogicComputationTest.showMessageDialog("Това е съобщение", "Съобщение");  
});
```

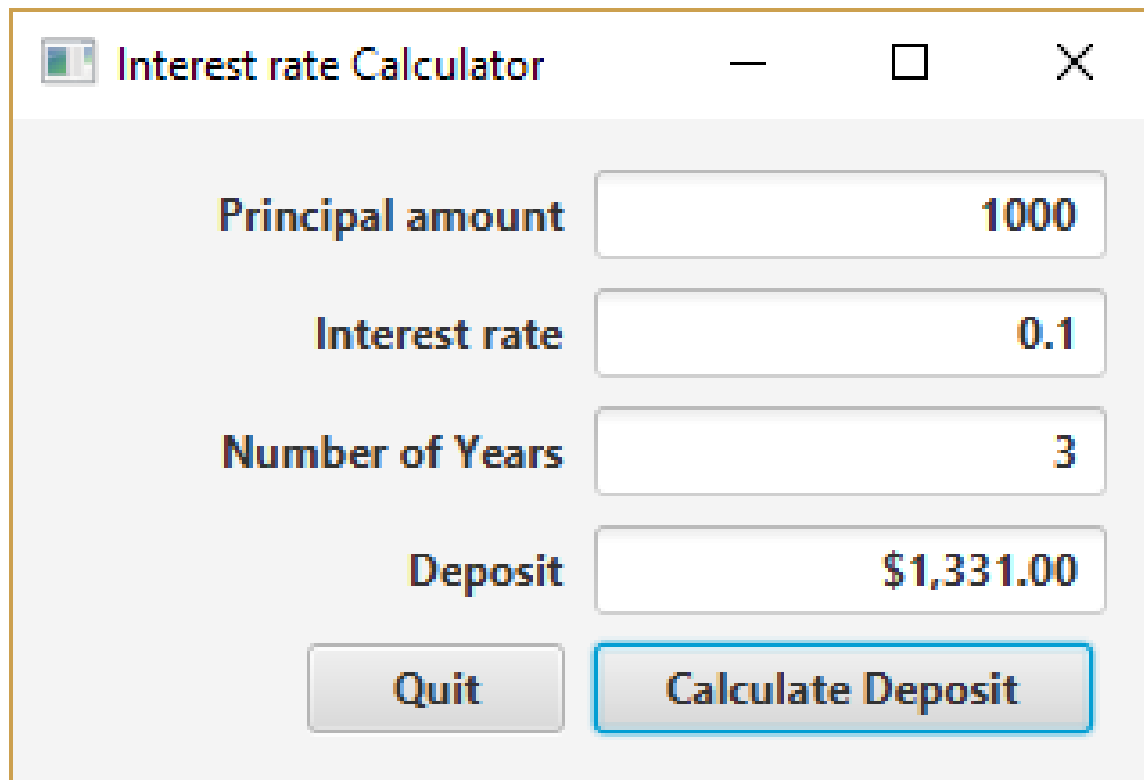
5.8c Диалогов прозорец за съобщения

```
private String input = "";
public void getSomeInput() {
    PlatformImpl.startup(() -> {
        input = LogicComputationTest.inputDialog("Въведете име: ",
                                                    "Въвеждане", null);
    });
    System.out.println(input); // process input
}
```

```
private ButtonType bt;
public void getButtonType() {
    PlatformImpl.startup(() -> {
        bt = LogicComputationTest.confirmDialog("Нов проект?",
                                                "Стартира нов проект", null);
    }); // process input
    System.out.println(bt == ButtonType.OK ? "Нов проект" : "Отказ");
}
```

5.9 Поддреждане на компонентите

Нека да построим JavaFX за пресмятане на лихва със следния графичен интерфейс



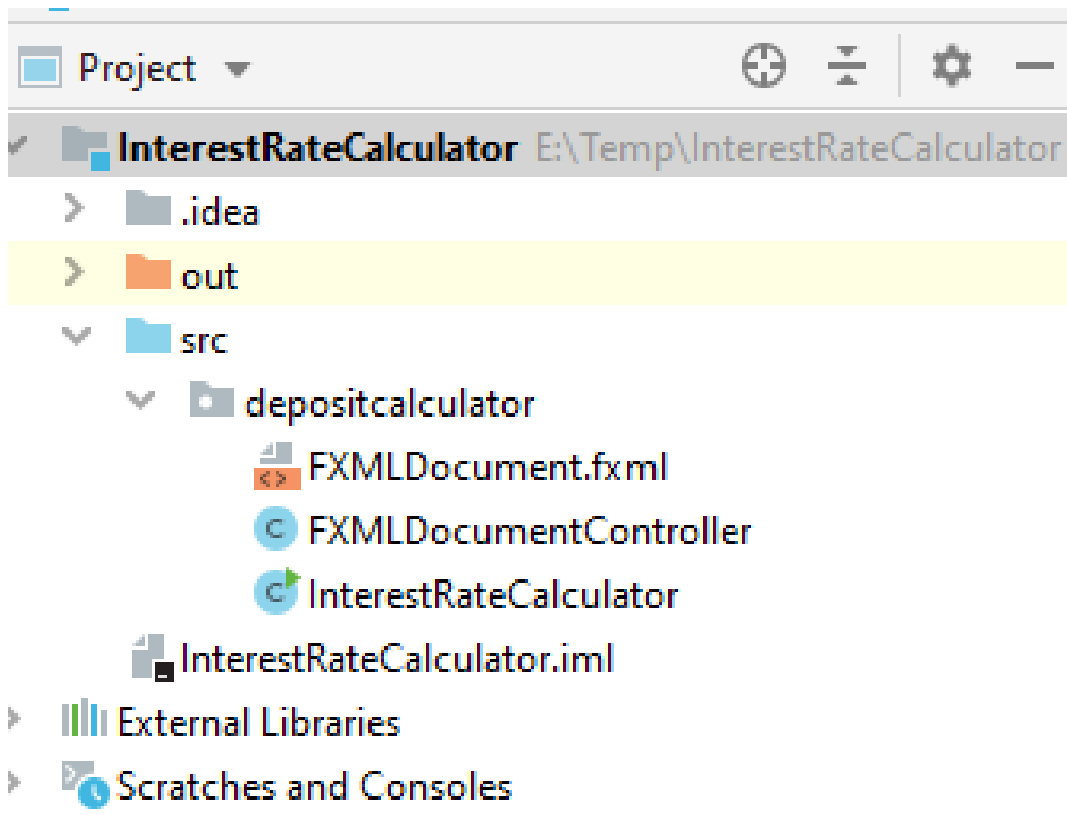
The image shows a JavaFX window titled "Interest rate Calculator". It contains four input fields and two buttons. The first three fields are for input: "Principal amount" (1000), "Interest rate" (0.1), and "Number of Years" (3). The fourth field, labeled "Deposit", shows the calculated result: "\$1,331.00". At the bottom, there are two buttons: "Quit" and "Calculate Deposit".

Field Label	Value
Principal amount	1000
Interest rate	0.1
Number of Years	3
Deposit	\$1,331.00

Buttons: Quit, Calculate Deposit

5.9 Поддреждане на компонентите

Създаваме FXML проект в IntelliJ в указаната в предишния пример последователност.



5.9 Подреждане на компонентите

Класът на приложението е преименуван с Refactor-
> Rename на `InterestRateCalculator.java`

Този клас е произведен на клас `Application` и има статичен метод `launch()`. С този метод започва изпълнението на приложението като създава обект от `InterestRateCalculator` и изпълнява наследения метод `start()` от клас `Application` за този обект. В метода `start()` се създава дървото от възли, което после се добавя към `Scene` (сцената). Така създадената сцена се поставя на `Stage`, чиито обект се реферира с параметър на метода `start()`

5.9 Подреждане на компонентите

Забелязваме, че компонентите в зададеният модел на графичен интерфейс са подредени в редове и колонки.

The image shows a window titled "Interest rate Calculator" with a standard Windows-style title bar (minimize, maximize, close buttons). The window contains a form with the following components arranged in rows and columns:

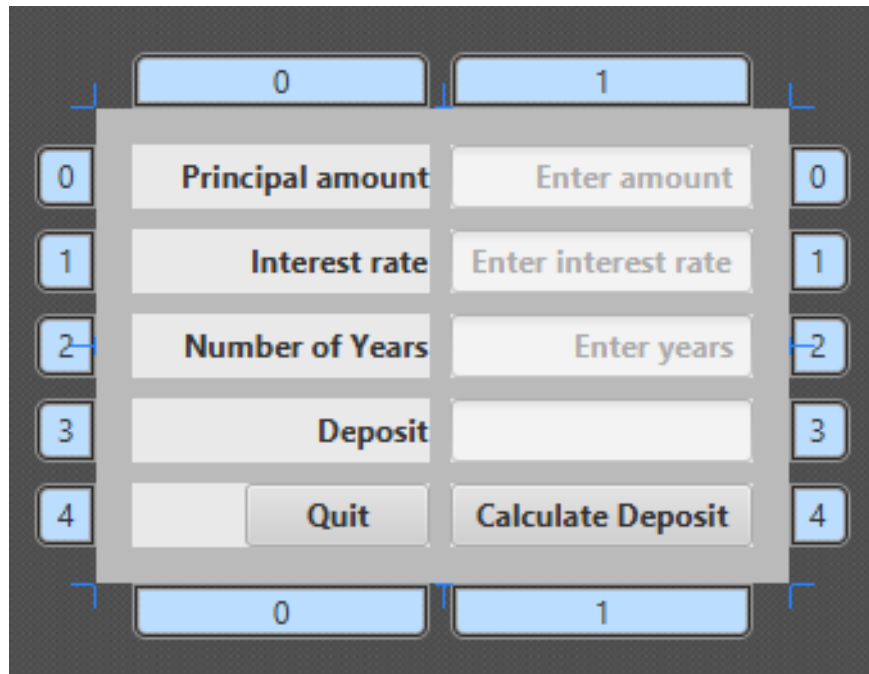
Label	Value
Principal amount	1000
Interest rate	0.1
Number of Years	3
Deposit	\$1,331.00

At the bottom of the form, there are two buttons: "Quit" and "Calculate Deposit".

5.9 Подреждане на компонентите

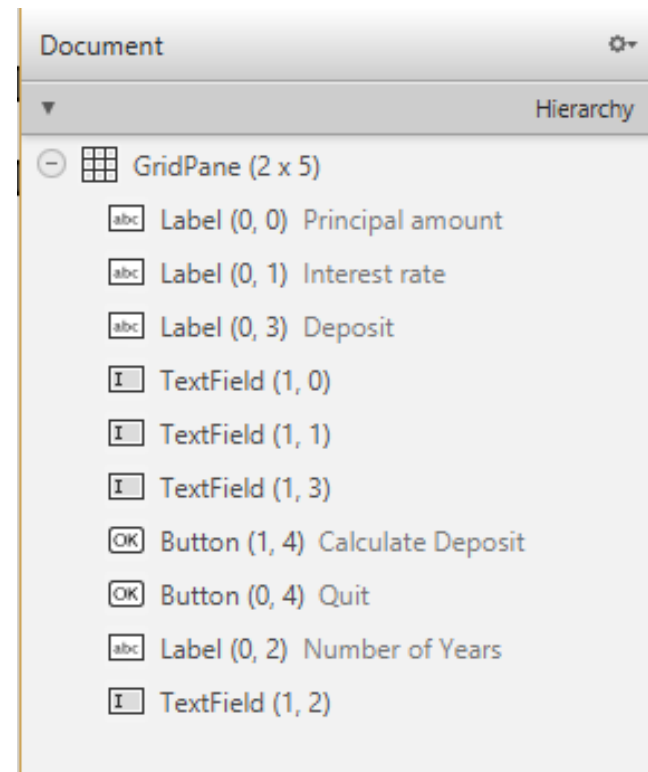
По тази причина ще използваме `GridPane` за разполагане на компонентите в клетки по редове и колони.

Добавяме текстови полета, етикети и бутони, както в предходния пример на приложение на JavaFX.



5.9 Подреждане на компонентите

Имената на всички компоненти са в стила на **Изменената Унгарска нотация**, а разположението на компонентите по редове и колони е показано в Раздела Document (Hierarchy) на SceneBuilder



5.9 Подреждане на компонентите

Отстоянието на съдържанието (content) на Възел от неговите граници се нарича `Padding`. За `GridPane` по стандарт `Padding` е на отстояние 14 px от всички 4 страни.

Отстоянията по хоризонталата и вертикалата на клетките на `GridPane` се определят съответно от `Hgap` и `Vgap` (по стандарт се избират 8 px)

5.9 Подреждане на компонентите

След създаване на графичния модел Scene Builder получаваме скелета на Контролера

```
@FXML
private ResourceBundle resources;

@FXML
private URL location;

@FXML
private Label lblAmount;

@FXML
private Label lblTip;

@FXML
private Label lblTotal;

@FXML
private TextField txtPrincipalAmount;

@FXML
private TextField txtInterestRate;

@FXML
private TextField txtTotalDeposit;

@FXML
private Button btnCalculate;

@FXML
private Button btnQuit;

@FXML
private TextField txtNumberYears;
```

6 BigDecimal за работа с парични суми

Като стандарт в Java за пресмятания с парични суми се утвърди да бъде `class BigDecimal`.

Конструкторът му позволява създаване на обект от всички числови типове, а също и от тип `String`.

```
BigDecimal bdAmount = new BigDecimal(1);
```

```
BigDecimal bdPercent = new BigDecimal("0.2");
```

- Събиране `bdAmount = bdAmount.add(bdPercent)`
- Изваждане `bdAmount =
bdAmount.subtract(bdPercent)`
- Умножение `bdAmount =
bdAmount.multiply(bdPercent)`
- Делене `bdAmount =
bdAmount.divide(bdPercent)`

7 Приложение на BigDecimal и NumberFormat

```
private static final NumberFormat CURRENCY = NumberFormat.getCurrencyInstance();

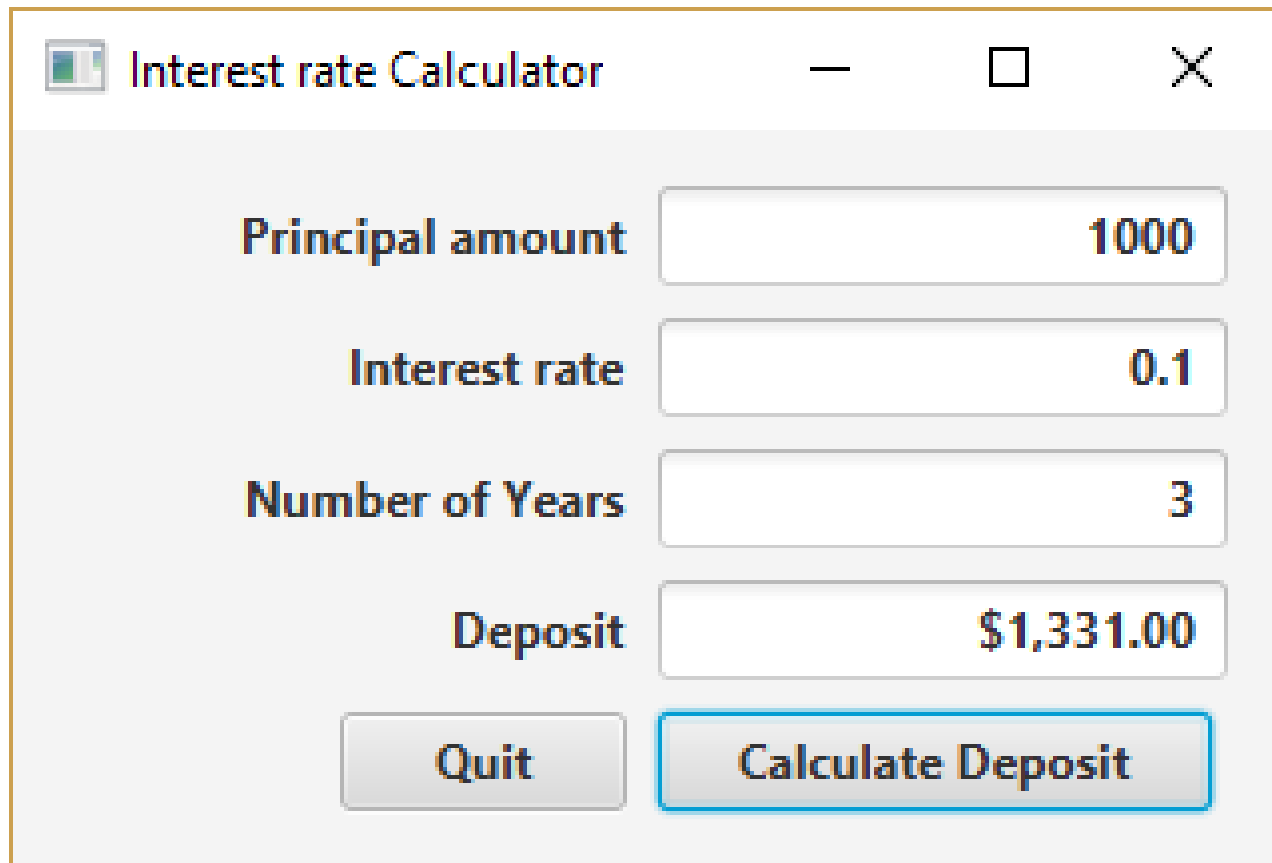
@FXML
private TextField txtNumberYears;

@FXML
void btnQuitOnAction(ActionEvent event) { System.exit(0); }

@FXML
void btnCalculateOnAction(ActionEvent event) {
    try {
        BigDecimal principalAmount = new BigDecimal(txtPrincipalAmount.getText());
        int years = Integer.parseInt(txtNumberYears.getText());
        BigDecimal interestRate = new BigDecimal(txtInterestRate.getText());
        BigDecimal term = new BigDecimal(1);
        interestRate = interestRate.add( new BigDecimal(1.));
        for (int i = 0; i < years; i++) {
            term = term.multiply(interestRate);
        }
        BigDecimal total = principalAmount.multiply(term);
        txtTotalDeposit.setText(CURRENCY.format(total));
    } catch (NumberFormatException ex) {
        txtPrincipalAmount.setText("Enter amount");
        txtPrincipalAmount.selectAll();
        txtPrincipalAmount.requestFocus();
    }
}

@Override
public void initialize(URL url, ResourceBundle rb) { CURRENCY.setRoundingMode(RoundingMode.HALF_UP); }
```

6 BigDecimal за работа с парични суми



A screenshot of a Java Swing window titled "Interest rate Calculator". The window has a standard title bar with a minimize button, a maximize button (disabled), and a close button. The main content area is light gray and contains four input fields with labels to their left: "Principal amount" with value "1000", "Interest rate" with value "0.1", "Number of Years" with value "3", and "Deposit" with value "\$1,331.00". At the bottom, there are two buttons: "Quit" and "Calculate Deposit". The "Calculate Deposit" button is highlighted with a blue border.

Field	Value
Principal amount	1000
Interest rate	0.1
Number of Years	3
Deposit	\$1,331.00

Buttons: Quit, Calculate Deposit

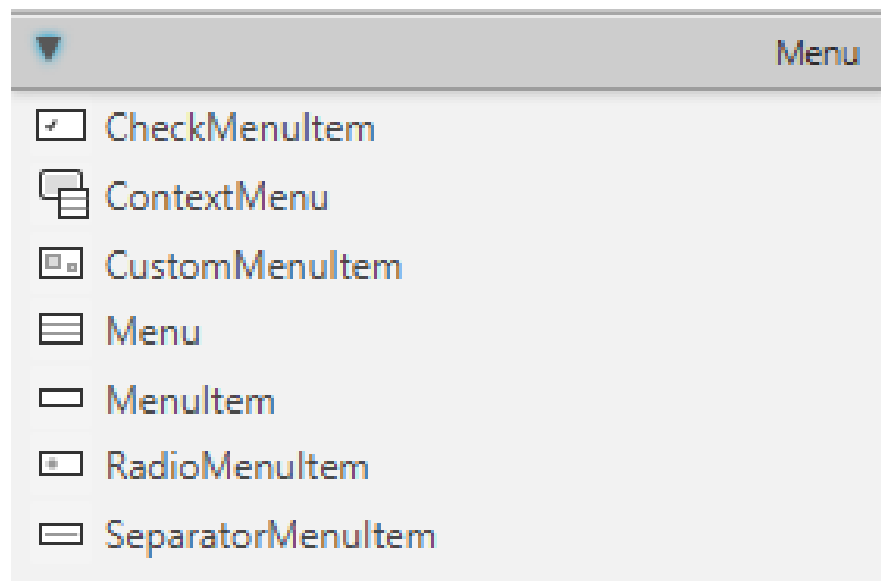
8 Добавяне на меню

В JavaFX API могат да се създават менюта посредством следните класове.

MenuBar – служи като **контейнер** за цялото меню

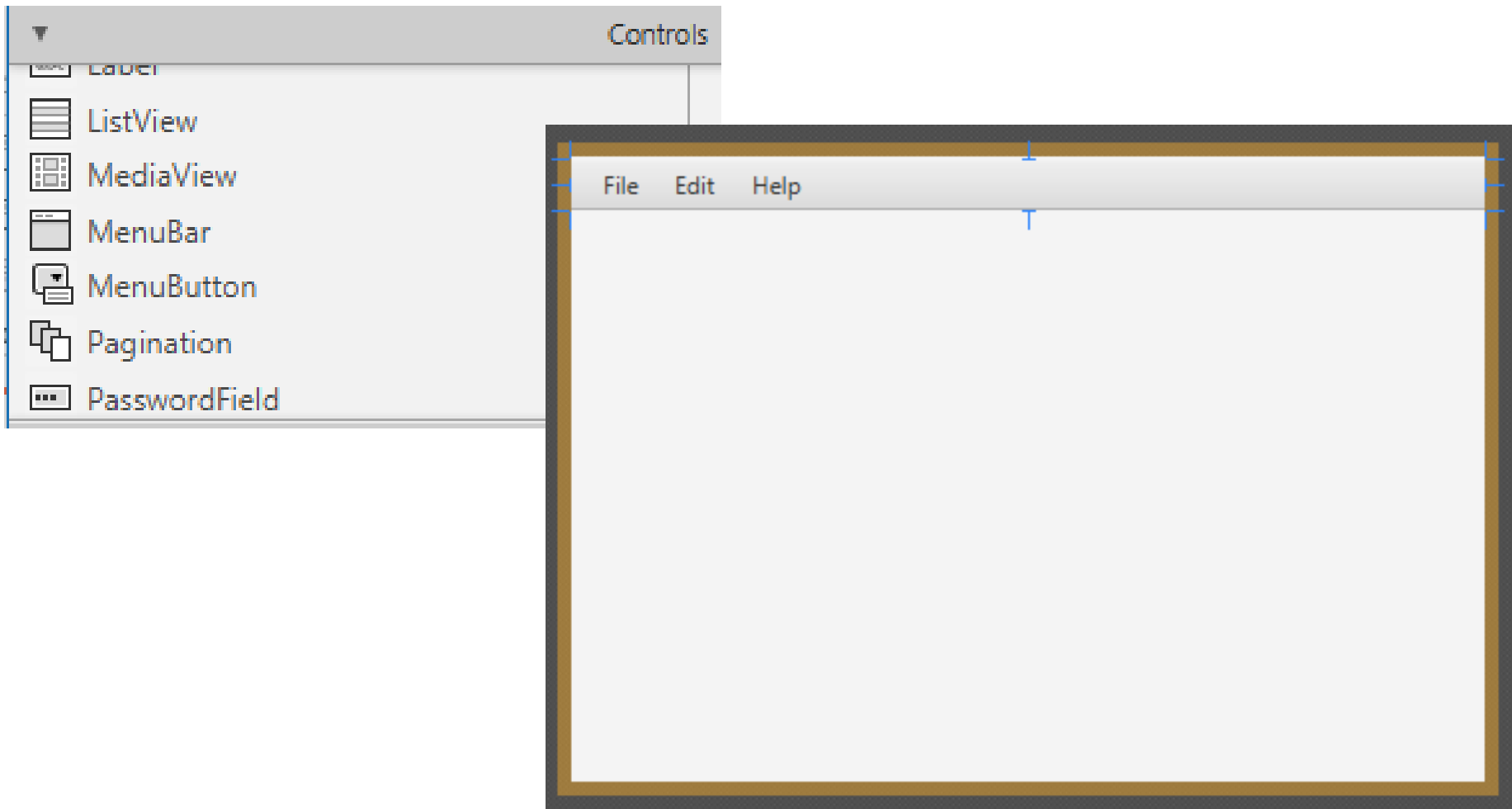
В **MenuBar** се съдържат

- **Menu** – служат за **контейнер** на **MenuItem**
- **MenuItem** служат за **обработка на събитие** и могат да бъдат
 - **CheckMenuItem**
 - **RadioMenuItem**
 - **CustomMenuItem**
 - **SeparatorMenuItem**



8 Добавяне на меню

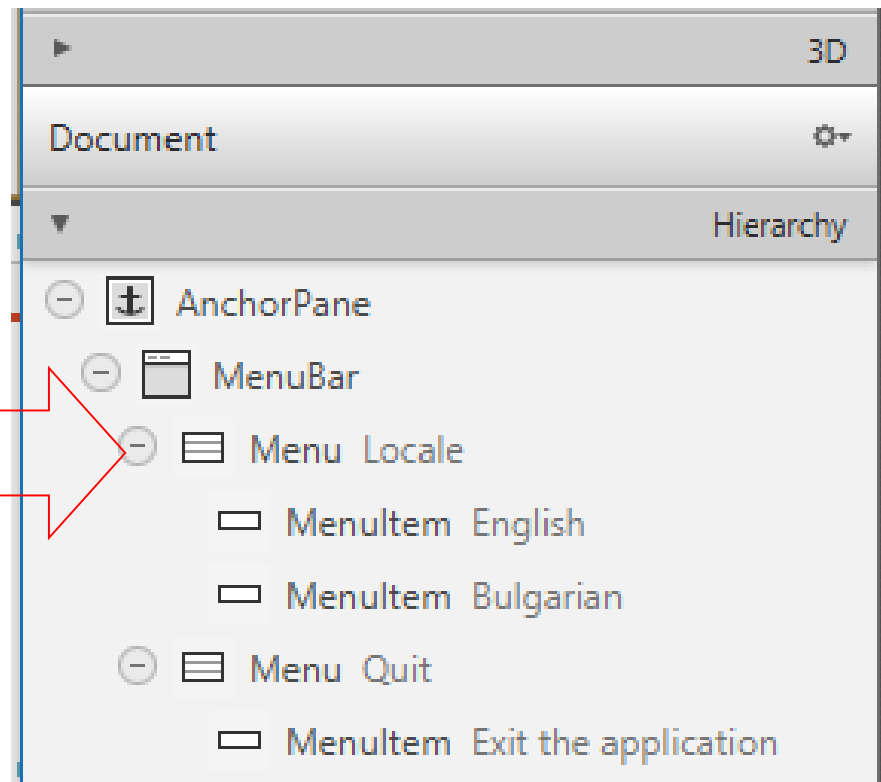
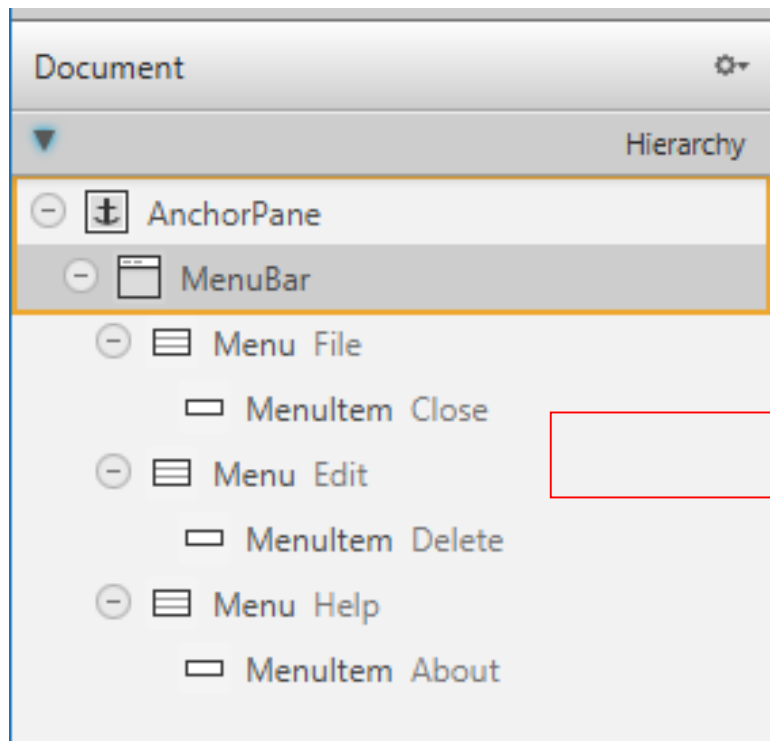
1. Първо добавяме **MenuBar** от списъка Controls



8 Добавяне на меню

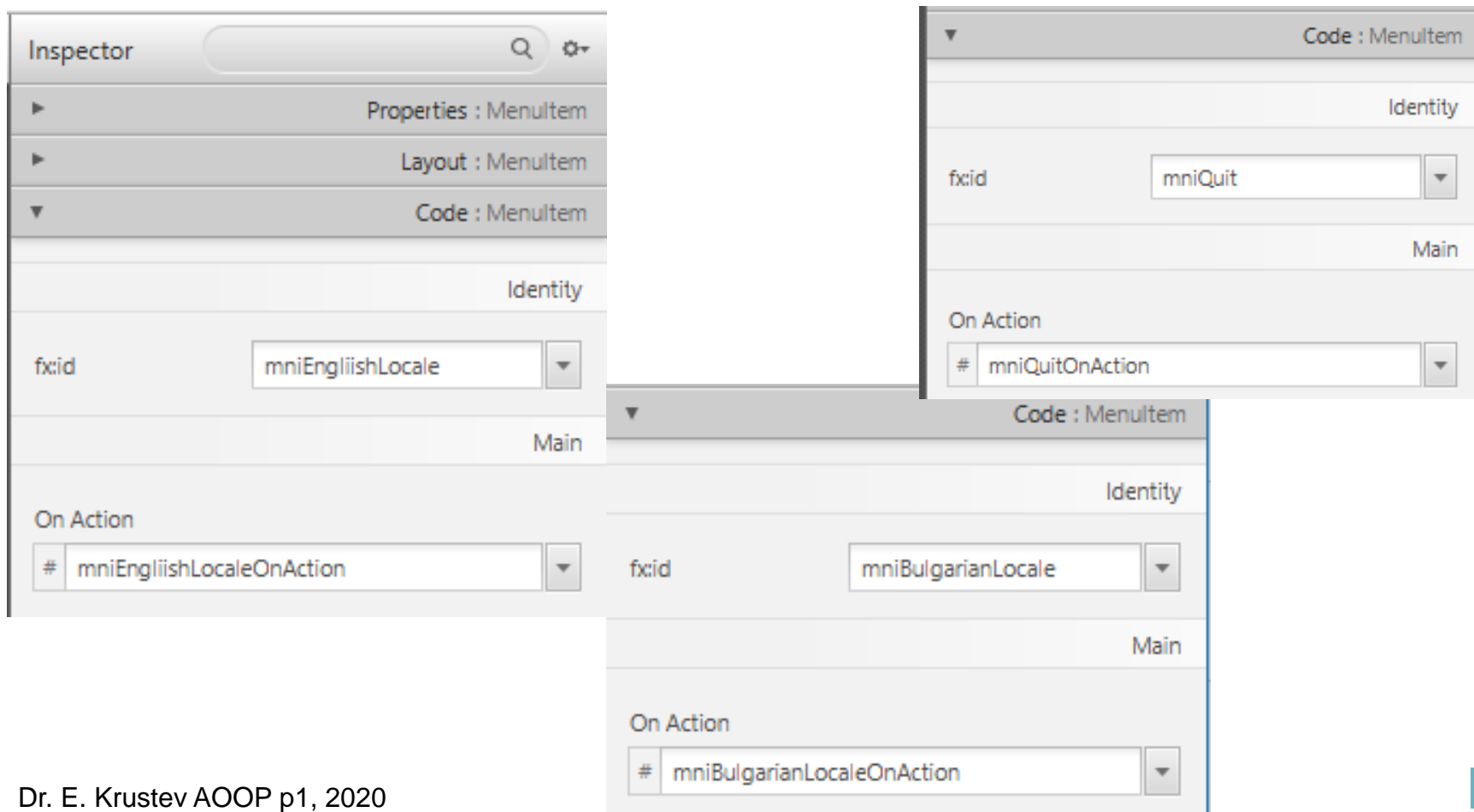
MenuBar , добавен в **SceneBuilder**, съдържа три подразбиращи се **Menu** обекта с по един **MenuItem**.

2. Редактираме тези подразбиращи се обекти, според изискванията на конкретния проект



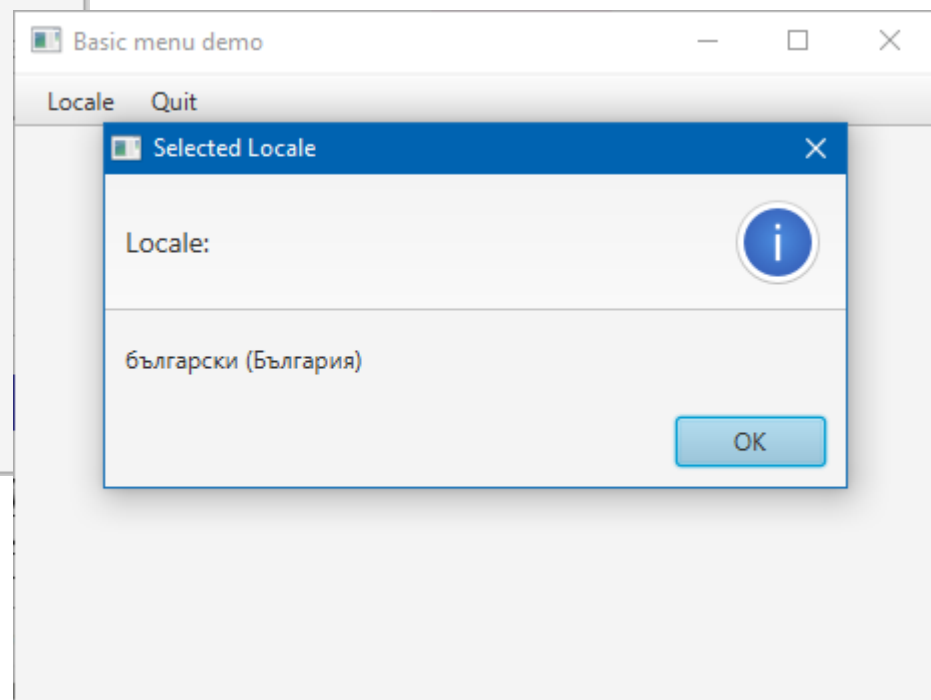
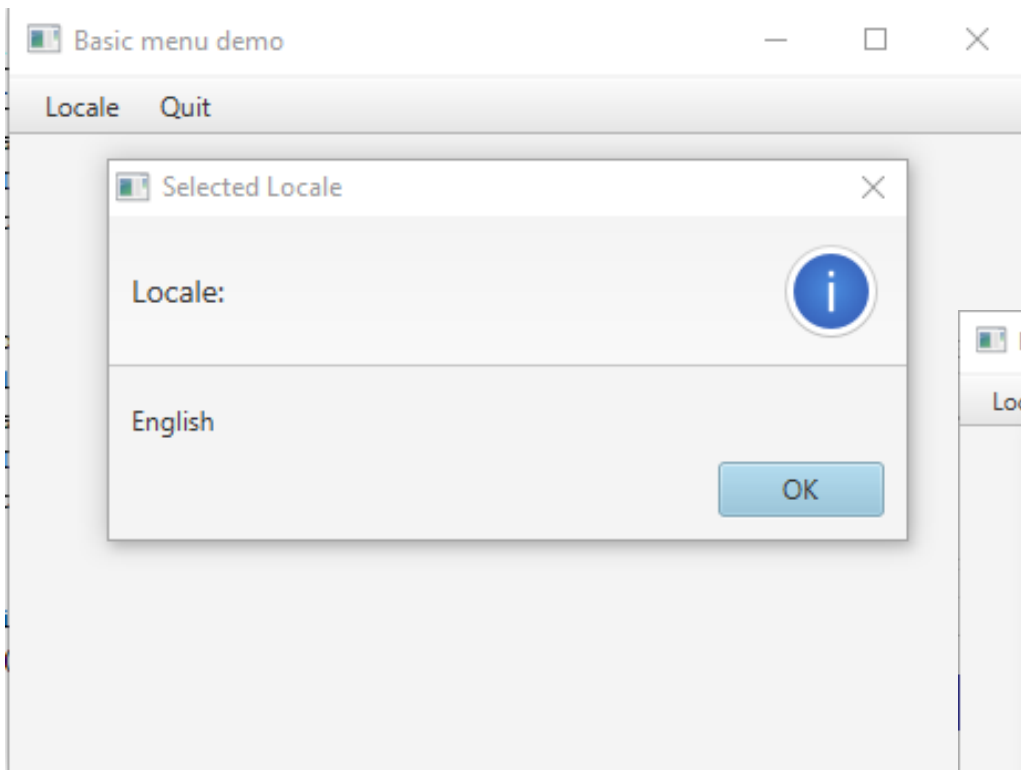
8 Добавяне на меню

3. Задаваме **смислени идентификатори** за **MenuItem** и **методите за обработка** на събитието **OnAction** в съответствие с Модифицираната унгарска нотация.



8 Добавяне на меню

4. Дефинираме в Контролера методите за обработка на събитието **Action**



@FXML

```
void mniBulgarianLocaleOnAction(ActionEvent event) {  
    locale = new Locale("bg", "BG");  
    Locale.setDefault(locale);  
    mb.setContentText(locale.getDisplayName(locale));  
    mb.showAndWait();  
}
```

@FXML

```
void mniEnglishLocaleOnAction(ActionEvent event) {  
    locale = Locale.ENGLISH;  
    Locale.setDefault(locale);  
    mb.setContentText(locale.getDisplayName(locale));  
    mb.showAndWait();  
}
```

@FXML

```
void mniQuitOnAction(ActionEvent event) {  
    Platform.exit();  
}
```

@FXML

```
void initialize() {  
    locale = Locale.ENGLISH;  
    mb = new Alert(Alert.AlertType.INFORMATION);  
    mb.setTitle("Selected Locale");  
    mb.setHeaderText("Locale:");  
    mb.setContentText(locale.getDisplayName(locale));  
}
```


8 Добавяне на меню

Стандартна опция е елементите на менюто да са достъпни с комбинация от клавиш **ALT** и буква от дума.

В JavaFX потребителят натиска клавиша ALT и буква предхождана от **подчертаване_ за активиране** на елемент от меню и последващо използване на стрелките на клавиатурата за достъп до други елементи на менюто. Освен подчертаване в Scene Builder трябва да се избере опцията Mnemonic Parsing

8 Добавяне на меню

The screenshot displays a JavaFX application window titled "Basic menu demo". The window has a menu bar with two items: "Locale" and "Quit". The "Locale" menu is currently open, showing a list of options: "English" and "Bulgarian".

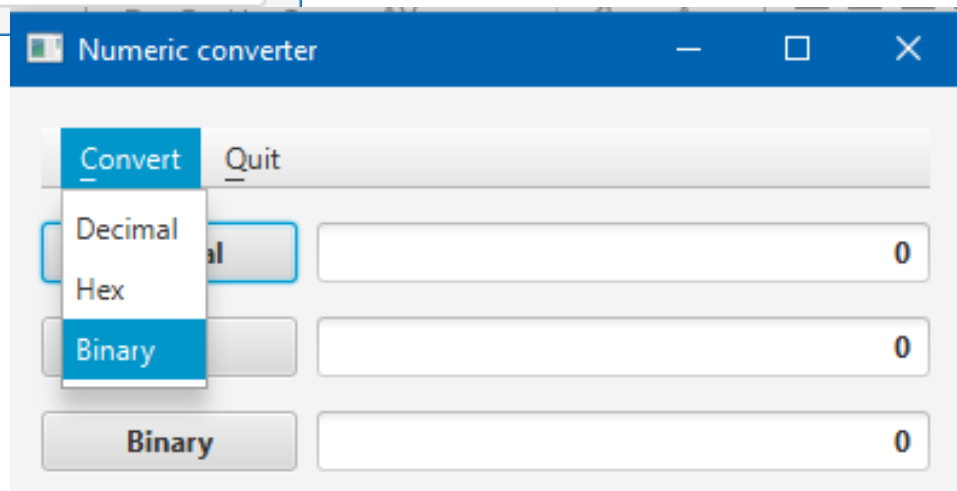
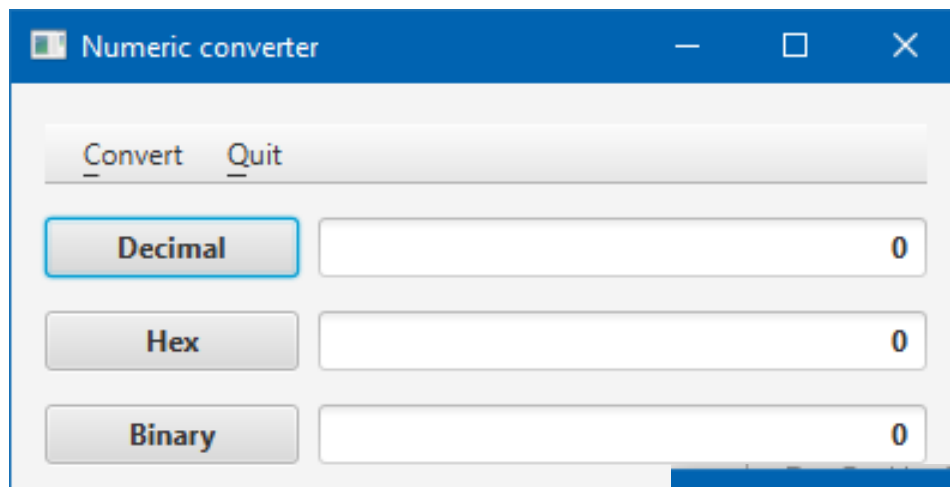
On the right side of the window, there is an "Inspector" panel. The "Properties : Menu" section is active, showing the following properties for the selected menu item:

- Text:** _Quit
- Specific:**
- Accelerator:** none
- Node:**
- Disable:** ☐
- Visible:** ☒
- JavaFX CSS:**
- Style:** [] [] + ▾
- Style Class:** [] + ▾
- Id:** []
- Extras:**
- Mnemonic Parsing:** ☒

Задачи

Задача 1

Напишете JavaFX приложение за преобразуване на числа в десетична, двоична и шестнайсетична бройна система. Нека приложението да има меню с опции, които възпроизвеждат действията на съответните бутони



Задачи

Задача 2

Напишете JavaFX приложение, която представя калкулатор.
Демонстрирайте приложението на графичната компонента в JavaFX
FXML приложение

