

Лекция 15.а

JavaFX свойства, Binding, Обработка на събития

- 15a.1 Обзор на лекцията
- 15a.2 Въведение в JavaFX properties
 - 15a.2a Creating JavaFX properties
 - 15a.2b Binding JavaFX properties
- 15a.3 Свързване на GUI свойства
 - 15a.3a Обработка на събитието Change
- 15a.4 Методи за обработка на събитието Mouse
- 15a.5 Изпълнение на проекта
 - 15a.5.a Изграждане на графичния интерфейс
 - 15a.5.b Редактиране на класа на приложението
 - 15a.5.c Редактиране на класа на Контролера
 - 15a.5.d Рисуване в графичния контекст на Контейнер
- 15a.6 Рисуване на свободна линия
- 15a.7 Избиране на цвят и дебелина на линията
- 15a.8 Редактиране на нарисувана графика
- 15a.9 Обработка на събития
- 15a.10 Верижно разпределение на обработката на събитията
- 15a.11 Методи за обработка на KeyEvent

Литература:

15a.1 Обзор на лекцията

В предишните лекции (Лекция 2, 4 и 9) усвоихме основите на създаване на GUI с JavaFX. Тук ще научите:

- Да създавате връзки между свойства на обекти и да създавате свързани възли в JavaFX**
- да работите с други често използвани компоненти в JavaFX (например, `TitledPane`, `Menu`, `MenuItem` и `RadioButton`),**
- да обработвате събития, породени от движение на мишката и**
- да рисувате в графичния контекст на JavaFX**
- да обработвате събития, породени от клавиатурата**

15a.1a Преговор

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.

15a.1b 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>
PictureBox	<i>pic</i>
TextBox	<i>txt</i>

15a.1c Оразмеряване и позициониране на възлите

По подразбиране Възлите не се оразмеряват в конкретни размери за дължина и ширина. В противен случай, моделираният GUI се разваля при промяна на съдържанието или при промяна в размерите на графичния прозорец.

В допълнение към свойствата `width`, `height` Възлите имат също свойства `prewidth`, `preheight`, `minwidth`, `minheight`, `maxwidth` и `maxheight`. Така е възможно да се зададат интервали, в които да се изменят дължината и ширината, при които се запазва качеството на изображения графичен модел. Размерите се задават в pt (1/72 inch). Предпочитаните размери са тези, които ще се използват в общия случай

15a.1c Оразмеряване и позициониране на възлите

Възлите се разполагат относително базовия им Възел в дървото от възли на Сцената. Всеки Контейнер с подреждане (Layout pane) се грижи за позициониране съдържащите се в него възли, които могат да са също Layout pane, контроли или друго съдържание. При това се отчитат зададените предпочитани, минимални и максимални размери на съдържащите се в контейнера възли.

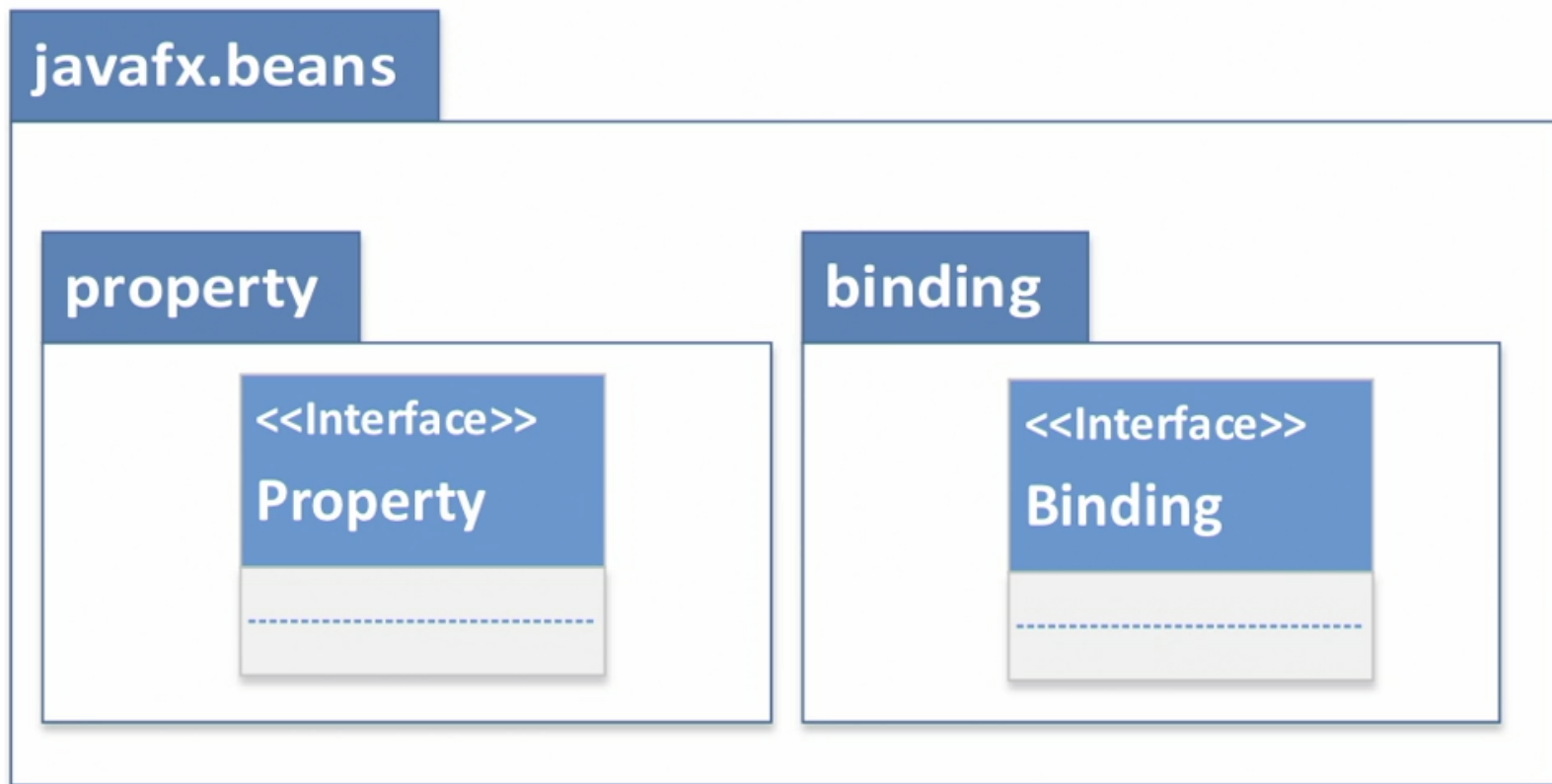
15a.2 Въведение в JavaFX properties

Да разгледаме:

```
public static void main(String[] args) {  
    double weight = 80; // [kg]  
    double height = 1.70; // m  
    double bmi = (weight) / (height * height);  
    System.out.printf("BMI: %.2f%n", bmi); // BMI: 27.68  
    weight++;  
    System.out.printf("BMI: %.2f%n", bmi); // BMI: 27.68  
}
```

Дали не може при промяна на weight или height автоматично да се промени резултата на bmi?

15a.2 Въведение в JavaFX properties



Библиотека JavaFX.beans предоставя богат набор от класове, които ни позволяват да създадем по-динамични програми с минимални усилия. За целта се използват специални типове данни, наречени JavaFX properties и специален механизъм за изграждане на връзки (Bindings) помежду им

15a.2 Въведение в JavaFX properties

```
public class BindingsBMI {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        DoubleProperty height = new SimpleDoubleProperty();
        DoubleProperty weight = new SimpleDoubleProperty();
        //bmi = (weight) / (height * height)
        NumberBinding bmi = Bindings.divide(weight,
                                             Bindings.multiply(height, height));

        for (int i = 0; i < 2; i++) {
            System.out.print("Enter weight in [kg]: ");
            weight.set(input.nextDouble());
            System.out.print("Enter height in [m]: ");
            height.set(input.nextDouble());
            System.out.printf("BMI = %.2f%n", bmi.doubleValue());
            System.out.println("*****");
        }
    }
}
```

```
Enter weight in [kg]: 44
Enter height in [m]: 1.7
BMI = 15.22
*****
Enter weight in [kg]: 55
Enter height in [m]: 1.5
BMI = 24.44
*****
```

15a.2 Въведение в JavaFX properties

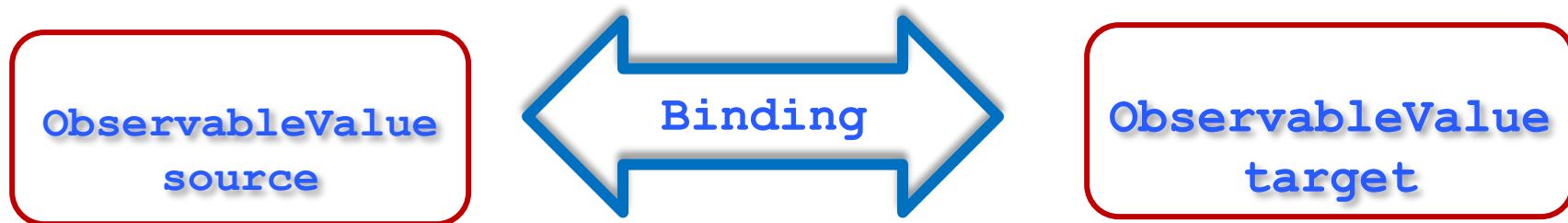
JavaFX въвежда нова концепция, наречена свързване на JavaFX свойства (*binding property*)

Реализацията ѝ позволява промени в стойностите на свойства на зададен обект (*target object*) да се обвържат с промени в стойностите на свойства на друг обект (*source object*), който е **източник** на **такива промени**. Когато свойство на източника на промени приеме нова стойност, същата стойност се присвоява автоматично на привързаното към него свойство на зададения обект. Накратко, зададения обект ще наричаме **привързан обект** (*binding object*) или привързано свойство (*binding property*).

15a.2 Въведение в JavaFX properties

Binding

Unidirectional or bidirectional



15a.2 Въведение в JavaFX properties

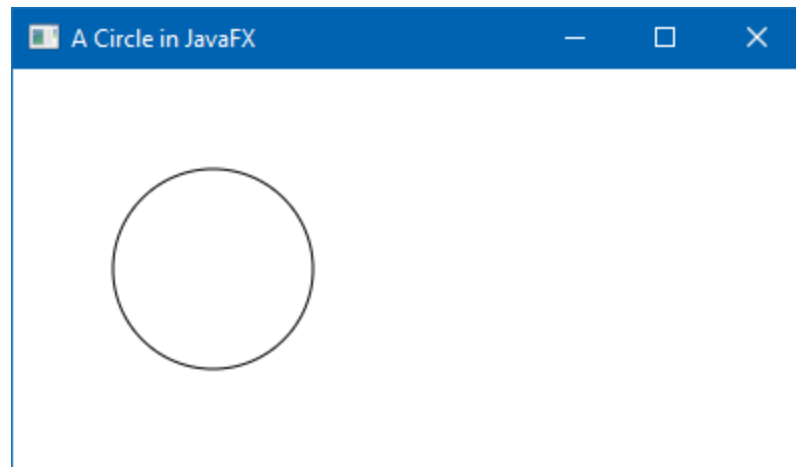
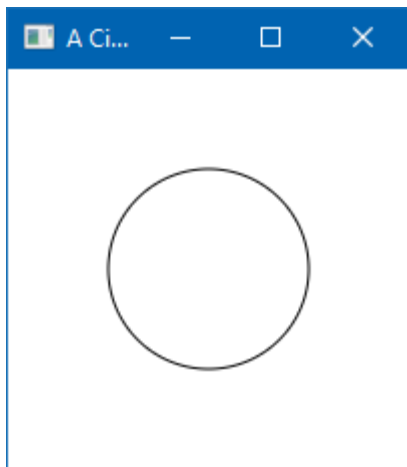
В конкретния случай координатите `centerX` и `centerY` на центъра на окръжността. При моделиране на `class Circle` бихме интерпретирали тези координати като свойства на този клас със съответни `get` и `set` методи. В JavaFX, обаче, сме заинтересувани да **ИЗПОЛЗВАМЕ** класове, които съдържат предварително зададени свойства вместо да имплементираме свойства в наши потребителски дефинирани класове.

За целта ще използваме нова конвенция за именуване на свойства, прилагаща т. нар. **JavaFX *property pattern*** (*шаблон за прилагане на свойства*)

15a.2 Въведение в JavaFX properties

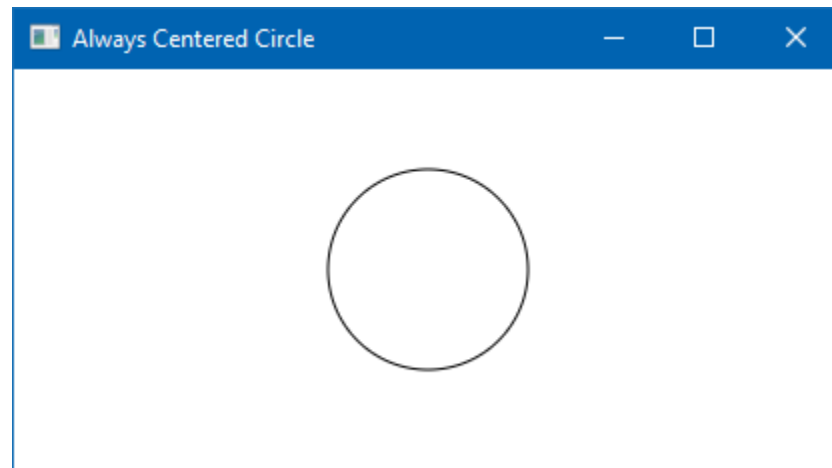
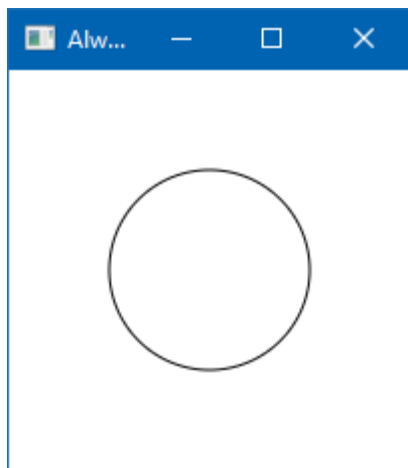
Пример.

Да нарисуваме окръжност в JavaFX и да я поставим центъра ѝ в средата на графичния прозорец с предварително зададени размери 200 x 200 (виж проект `CenteredCircle`, приложен към лекцията). При промяна в размерите на прозореца окръжността вече не е центрирана



15a.2 Въведение в JavaFX properties

Ако желаем окръжността да остава винаги в центъра на графичния прозорец трябва да обвържем стойностите на координатите на центъра ѝ `centerX` и `centerY` с промените в дължината и височината на графичния прозорец.



15a.2 Въведение в JavaFX properties

JavaFX properties са аналогични на **wrapper** класове със следните две допълнителни характеристики:

- ✓ Могат да се свързват с други JavaFX свойства
- ✓ Могат да се прикачват към обекти за обработка на събития

Придават възможност за динамично реагиране при промяна на стойностите на други JavaFX свойства и при възникване на събития.

Дефинирани в package `javafx.beans`, което съдържа пакет `javafx.beans.property`, където са специализирани свойства като `BooleanProperty`, `DoubleProperty`.

Тези свойства „опаковат“ стойност от съответния основен тип, например, `Boolean` или `Double`. Тази стойност е текущата стойност на JavaFX свойството.

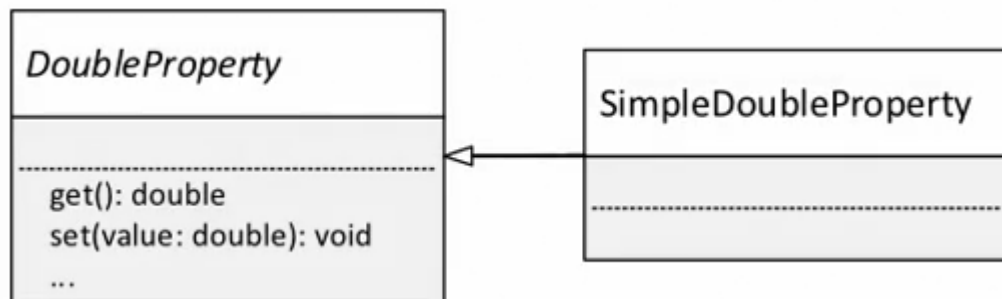
15a.2 Въведение в JavaFX properties

Правило:

Ако трябва да използвате тип на свойство, който опакова тип, примерно, `Double`, използвайте

`DoubleProperty height = new SimpleDoubleProperty();`

DoubleProperty е абстрактен тип, а е конкретната му имплементация **SimpleDoubleProperty**



15a.2 Въведение в JavaFX properties

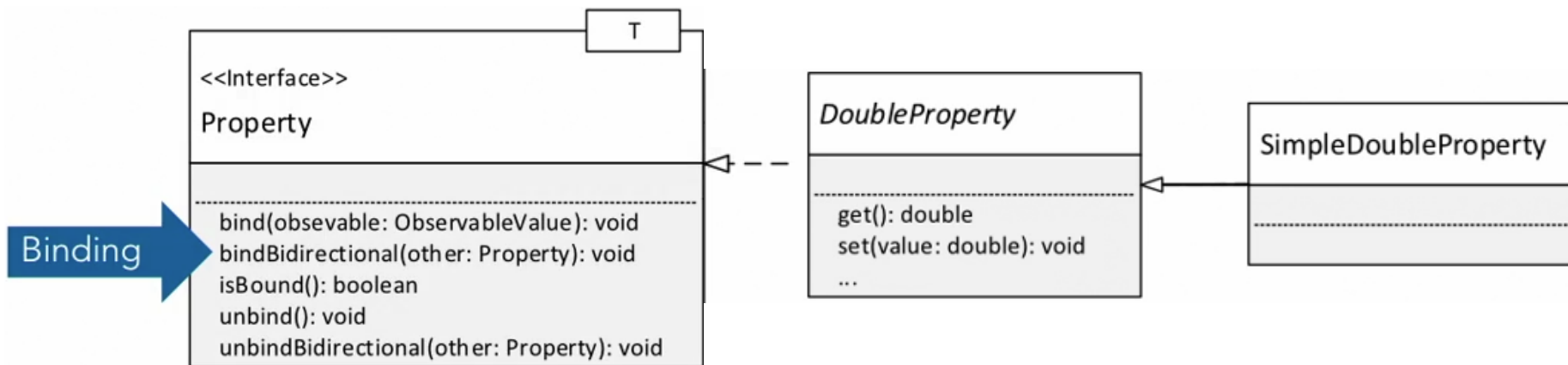
На този пример променливите от тип `DoubleProperty` реферират обекти от тип `SimpleDoubleProperty`.

Това е така, защото `DoubleProperty`, `FloatProperty`, `LongProperty`, `IntegerProperty`, и `BooleanProperty` са **абстрактни** класове. Съответните им конкретни класове `SimpleDoubleProperty`, `SimpleFloatProperty`, `SimpleLongProperty`, `SimpleIntegerProperty`, и `SimpleBooleanProperty` се използват за създаване на обекти от тези JavaFX свойства.

Тези класове придават допълнителна функционалност на съдържащата се в тях стойност по подобие на опаковашите класове `Double`, `Float`, `Long`, `Integer`, и `Boolean` за примитивните типове данни.

15a.2 Въведение в JavaFX properties

DoubleProperty имплементира интерфейс **Observable** с методи **bindBidirectional()**, **unbind()**



DoubleProperty имплементира също интерфейс **ObservableValue**, с методи **addListener()** и **removeListener()** за регистриране на обект **ChangeListener** за обработка на събитията **Change** или **Invalidation** възникващи при промяна на **ObservableValue**, където **ObservableValue** реферира обекта, към чието свойство е свързано **DoubleProperty**

15a.2 Въведение в JavaFX properties

JavaFX свойствата опаковат почти всички данни и много от динамичните структури от данни. Примери:

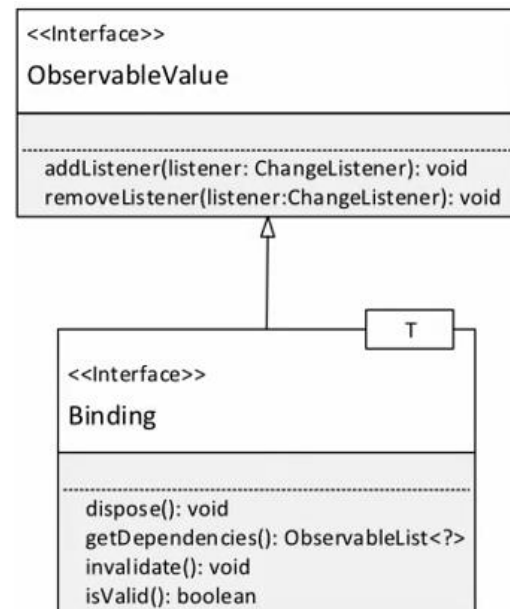
Boolean, Double, Float, Integer, Long, String, Object, List, Map и Set

Декларира се по сходен начин като **DoubleProperty**
StringProperty str = new SimpleStringProperty();
ListProperty lp = new SimpleListProperty();

15a.2 Въведение в JavaFX properties

JavaFX свойствата се свързват със свойства на обекти от тип `ObservableValue`. Типовете на тези свойства като например `BooleanBinding`, `DoubleBinding` имплементират интерфейсите `Binding<T>`, `Observable`, `ObservableValue<T>` и `ObservableBooleanValue` от `javafx.beans.binding`, където `T` е типът на данната опакована в обекта, рефериран с `ObservableValue`.

Докато `Binding<T>` служи за управление на връзката между свойствата, то `ObservableValue` служи за управление на събитията при промяна на стойността на свързаното свойство.



15a.2 Въведение в JavaFX properties

Правило:

Когато свързваното свойство е `String`, то трябва да се използва съответния `String` тип за свързване. Респ., при свързване на числови типове данни се използват `Double` или `Integer binding`

Извод:

Както свързваното, така и свързаното JavaFX свойство поддържат свързване и могат да са източници на събитията `Change` или `Invalidation`

15a.2a Creating JavaFX properties

Опростено създаване на JavaFX свойства (read / write)

1. Декларирайте данна на клас от типа на JavaFX свойството, което ще създавате.

```
private final StringProperty firstName;
```

2. Инициализирайте данната (обикновено едновременно с декларацията) с референция към обект от (**Simple..**) конкретен тип съответен на типа на JavaFX свойството.

```
firstName = new SimpleStringProperty(this, "firstName", "");
```

Тук **this** реферира обекта, съдържащ свойството, **firstName** е името на JavaFX свойството, а последният параметър е подразбиращата се стойност на свойството.

Забележка: При този подход няма възможност за директен достъп до обикновеното Java свойство, опаковано в JavaFX свойството

15a.2a Creating JavaFX properties

3. Създайте Getter метод за свойството „опаковано“ в JavaFX свойството (`public` or `protected`, `final`), следва стила за именуване на Java Getter метод и връща стойността, опакована в JavaFX свойството с метода `get()`

```
public final String getFirstName() {  
    return firstName.get();  
}
```

4. Създайте Setter метод за свойството „опаковано“ в JavaFX свойството (`public` or `protected`, `final`), следва стила за именуване на Java Setter метод и връща стойността, опакована в JavaFX свойството с метода `set()`

```
public final void setFirstName(String value) {  
    firstName.set(value);  
}
```


15a.2a Creating JavaFX properties

5. Създайте **final** Getter метод за самото JavaFX свойство, преобразувано до съответния му абстрактен тип. Името на Getter метода да има суфикс **Property** към името на JavaFX свойството

```
public final StringProperty firstNameProperty() {  
    return firstName;  
}
```

Забележка:

1. При създаване на JavaFX свойство с IntelliSense на IntelliJ трябва да се направи Refactor на името на JavaFX свойството, а също на Getter и Setter методите
2. При създаване на JavaFX свойство с IntelliSense на IntelliJ не се прилагат добрите практики в JavaFX(**виж следващия слайд**)

15a.2a Creating JavaFX properties

`set()/setValue()` и `get()/getValue()` на JavaFX свойство, които имат за цел да уеднаквят достъпа до опаковане стойности от референтен и примитивен тип в случай на JavaFX свойства от типа на `BooleanProperty` or `DoubleProperty`:

Например:

`BooleanProperty`:

`void set(boolean value) // Setter of primitive datatype`

`void setValue(java.lang.Boolean v) // Setter of reference datatype`

`DoubleProperty`:

`void set(double value) // Setter of primitive datatype`

`void setValue(java.lang.Number v) // Setter of primitive datatype`

В тези типове на JavaFX свойства, методите `___Value()` работят с типа съответен на JavaFX свойството, докато директните методи `set()/get()` се използват с примитивния тип. В случая на `Object/String` JavaFX свойства кои от тези методи се използват.

15a.2a Creating JavaFX properties

Добри практики за създаване на JavaFX свойства (read / write)

Създаване на обект от JavaFX свойство при декларацията му по опростения начин отнема повече памет и време за обработка, отколкото при създаване на обикновените свойства в Java. При това, специфичната функционалност на JavaFX свойствата не се използва често. Поради това, създаване на обекти от JavaFX свойство за всяко свойство на клас просто за всеки случай е неоправдано.

Добра практика е обект от JavaFX свойство да се създава единствено, когато се извика Getter метода на това свойство и следователно, само, когато това свойство е действително необходимо.

На следващия слайд показваме как да се дефинира JavaFX свойство надгражда функционалността на обикновените JavaFX свойства и същевременно консумира допълнителен ресурс за допълнителната функционалност само при нужда. С други думи, обединяваме в една дефиниция на обикновено Java свойство и надграждането му в JavaFX свойство .

15a.2a Creating JavaFX properties

1. Декларирайте **private** данна, която ще представлява обикновеното Java свойство, което ще опаковаме в JavaFX свойство.

```
private String _firstName;
```

Разграничаваме обикновеното Java свойство от името на JavaFX свойство като използваме **_** за префикс

2. Декларирайте **без да инициализирате** **private** данна от типа на JavaFX свойството, което ще създавате.

```
private StringProperty firstName;
```

Типът на JavaFX свойството **трябва да е съответен на типа на обикновеното Java свойство** и при това **не е final**.

15a.2a Creating JavaFX properties

3. Създавате **final** Getter за обикновеното Java свойство

```
public final String getFirstName()
```

```
{
```

```
    if (firstName == null)
```

```
        return _firstName;
```

```
    else
```

```
        return firstName.get();
```

```
}
```

Тук се проверява дали JavaFX свойството е инициализирано т.е. дали се използва специфичната за него функционалност.

Ако не е инициализирано, то методът работи като Getter на обикновено Java свойство. Ако JavaFX свойството е инициализирано, то това свойство опакова стойността на обикновеното Java свойство и стойността на **private** данната (**_firstName** в този пример) се прочита с `get()` метода на JavaFX свойството.

15a.2a Creating JavaFX properties

4. Създавате **final** Setter за обикновеното Java свойство

```
public final void setFirstName(String value)
```

```
{  
    if (firstName == null)  
        _firstName = value;  
    else  
        firstName.set(value);  
}
```

Тук отново се проверява дали JavaFX свойството е инициализирано т.е. дали се използва специфичната за него функционалност. Ако не е инициализирано, то методът работи като Setter на обикновено Java свойство. Ако JavaFX свойството е инициализирано, то това свойство опакова стойността на обикновеното Java свойство и стойността на **private** данната (**_firstName** в този пример) се променя със **set()** метода на JavaFX свойството.

15a.2a Creating JavaFX properties

4. Накрая създавате **final** Getter за JavaFX свойството

```
public final StringProperty firstNameProperty()  
{ if (firstName == null)  
    firstName = new  
        SimpleStringProperty( this, "firstName", _firstName);  
    return firstName;  
}
```

Тук отново се проверява дали JavaFX свойството е инициализирано т.е. дали се използва специфичната за него функционалност. Ако не е инициализирано, то едва тогава се създава обект от това свойство като се използва конкретната му имплементация и типът на така създадения обект се преобразува до съответния абстрактен тип на JavaFX свойството при излизане от метода. Забелязва се, че се създава един единствен обект от JavaFX свойството и първоначалната му стойност е текущата стойност на обикновеното Java свойство.

15a.2a Creating JavaFX properties

IntelliJ позволява IntelliJSense генериране на JavaFX свойства. За целта инсталирайте шаблоните за генерирането им, както е описано на сайта на [Gluon](#) JAR файла `intelliј-live-template-settings.zip` с тези настройки е прикачен заедно с материалите за текущата лекция.

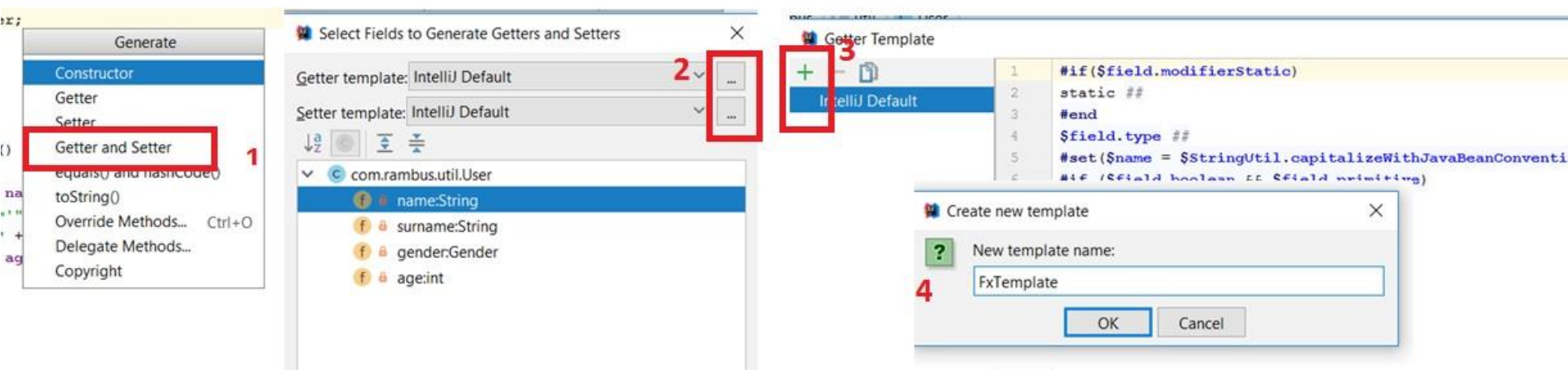
- ✓ Unzip the zip file. This will result in there being a **settings.jar** file as the only file.
- ✓ In IntelliJ, click on the ‘**File->Manage IDE Settings**’ menu
- ✓ Click on ‘**Import Settings...**’
- ✓ Find the settings.jar file on your file system
- ✓ Restart IntelliJ when asked

(създава свойствата опростено, виж следващия слайд)

15a.2a Creating JavaFX properties

Втори начин –позволява да се създаде потребителски дефиниран скрипт:

- Alt + Insert
- Getter-Setter
- Click Getter template -> Click new Template and write your own template script to generate JavaFX property. Save template
- Click Setter template -> Click new Template and write your own template script to generate JavaFX property. Save template



15a.2a Creating JavaFX properties

Трети начин – създаване на шаблон по съществуващ код (виж IntelliJ doc [Create a new template from a fragment of code](#)):

- Маркирайте сорс кода с желаната имплементация на JavaFX свойство и създайте live template (IntelliSense) от нея.
- Изберете **Tools | Save as Live Template** от основното меню. Показва се списък, където новосъздаденият шаблон е добавен към **user** group.
- Задайте кратко название за шаблона(обикновено е от вида fx-propertyType-details), опционално, препоръчително, е да въведете описание на предназначението шаблона и евентуално редактирайте шаблона.
- Потвърдете с **ОК** за запазване на промените.

15a.2b Binding JavaFX properties

Свързването на JavaFX properties се извършва от обект, който позволява:

- да се опише с израз представяне на връзката или зависимостите между свойствата
- стойностите на зависимите свойства да се променят едновременно с промените в стойностите в свойството източник на тези промени.

Поради това този обект е изграден от зависимости на JavaFX properties.

Съществуват два типа на свързване:

- ✓ Свързване на високо ниво
- ✓ Свързване на ниско ниво

15a.2b Binding JavaFX properties

Свързване на високо ниво (удобно за по-прости изрази на връзки между свойства)

- ❑ Fluent API (олекотено представяне)
- ❑ Посредством статични методи на class Bindings

Свързване на ниско ниво (удобно за по-сложни изрази на връзки между свойства, също използва по-малки ресурси от памет)

15a.2b Binding JavaFX properties

Пример за Fluent API

Предоставя на JavaFX типовете свойства методи като **add()** и **multiply()**. Например, **IntegerProperty** има методи **add()** и **multiply()** за извършване на аритметични операции.

Нека да сметнем лицето на квадрат със страна **side**

squareArea = squareSide * squareSide

където стойността на **squareArea** да се мени едновременно с всяка промяна на стойността на **side**

Декларираме съответните **свойства** и **израз на връзката** им.

//define JFX props

IntegerProperty squareArea = **new** SimpleIntegerProperty();

IntegerProperty squareSide = **new** SimpleIntegerProperty();

// define binding squareArea by expression involving squareSide

squareArea.**bind**(squareSide.multiply(**squareSide**));

// squareSide.multiply(squareSide) is the Binding object

15a.2b Binding JavaFX properties

На практика привързан обект (*target*, binding object) или привързаното му свойство (*binding property*) „слуша“ за промени на JavaFX свойство в обекта източник на промени и автоматично променя съответното си свойство при такава промяна. Привързваме JavaFX свойство на обект към JavaFX свойство в източник на промени посредством метода `bind()` по следния начин

```
target.someTargetProperty()  
    .bind(source.someSourceProperty());
```

Методът `bind()` принадлежи на интерфейса `javafx.beans.property.Property`, а JavaFX свойствата са производни на `javafx.beans.property.Property`. Типът на `someSourceProperty()` трябва да имплементира интерфейса `javafx.beans.value.ObservableValue`.

15a.2b Binding JavaFX properties

Обектите от тип **ObservableValue** is „*онаковат*“ стойност и позволяват да се наблюдават промени в тази стойност. JavaFX дефинира стандартни привързващи свойства за примитивните типове и **String**.

За примитивните типове **double**, **float**, **long**, **int**, **boolean** съответните JavaFX свойства са **DoubleProperty**, **FloatProperty**, **LongProperty**, **IntegerProperty**, **BooleanProperty**. За **String** съответното JavaFX свойство е **StringProperty**. Тези свойства имплементират интерфейса **ObservableValue**. И поради това могат да са параметри на метода **bind()**.

15a.2b Binding JavaFX properties

Пример за Fluent API

1. При това използваме методите на Fluent API за представяне на връзката, **а не директни аритметични операции**.
2. Промяната на `squareSide` води до едновременно с нея промяна на стойността на `squareArea`.

Това се нарича **еднопосочно свързване**

Забележка: При еднопосочно свързване може да се менят единствено свойствата, определящи връзката на **зависимост**.

Например, при опит да напишем индиректна зависимост на `squareSide` (като параметър на метода `multiply()`) от друго свойство, примерно, `square2Side`

```
squareSide.bind(square2Side);
```

Получаваме

java.lang.RuntimeException: A bound value cannot be set

15a.2b Binding JavaFX properties

Двупосочно свързване

Промяна в кое да е от свойствата води до едновременно с това промяна в другите свързани свойства.

Например, при опит да напишем индиректна зависимост на `squareSide` от друго свойство, примерно, `square2Side`

```
squareSide.bindBidirectional(square2Side);  
squareSide.set(3);  
square2Side.set(4);  
System.out.println( "Area: " +squareArea.getValue());
```

Получаваме

Area: 16

15a.2b Binding JavaFX properties

Двупосочно свързване

Промяна в кое да е от свойствата води до едновременно с това промяна в другите свързани свойства.

Например, при опит да напишем индиректна зависимост на `squareSide` от друго свойство, примерно, `square2Side`

```
squareSide.bindBidirectional(square2Side);  
squareSide.set(3);  
square2Side.set(4);  
System.out.println( "Area: " +squareArea.getValue());
```

Получаваме

Area: 16

15a.2b Binding JavaFX properties

Свързване посредством статичните методи на клас Bindings

Клас Bindings използва статични методи за създаване на обект на връзката, който после се използва с метод bind() на JavaFX свойство

```
IntegerProperty squareArea = new SimpleIntegerProperty();
```

```
IntegerProperty squareSide = new SimpleIntegerProperty();
```

```
//create a Binding object
```

```
NumberBinding squareBinding =
```

```
Bindings.multiply(squareSide,squareSide);
```

```
Bindings.bindBidirectional(squareSide,square2Side);
```

```
// static method multiply() takes as parameters the operands
```

```
// apply Binding object
```

```
squareArea.bind(squareBinding);
```

Забележка: Сложно е представянето на дълги изрази на връзка

15a.2b Binding JavaFX properties

Свързване посредством статичните методи на клас Bindings

Клас Bindings използва статични методи за създаване на обект на връзката, който после се използва с метод bind() на JavaFX свойство

```
IntegerProperty squareArea = new SimpleIntegerProperty();
```

```
IntegerProperty squareSide = new SimpleIntegerProperty();
```

```
//create a Binding object
```

```
NumberBinding squareBinding =
```

```
    Bindings.multiply(squareSide,squareSide);
```

```
Bindings.bindBidirectional(squareSide,square2Side);
```

```
// static method multiply() takes as parameters the operands
```

```
// apply Binding object
```

```
squareArea.bind(squareBinding);
```

Забележка: Сложно е представянето на дълги изрази на връзка

15a.2b Binding JavaFX properties

Свързване на свойства на ниско ниво

При него се създава променлива, реферираща Bindings обект, от тип произведен на типа на връзката, например, IntegerBinding, StringBinding

Тези типове са абстрактни класове и затова при създаване на обект от произведен на тях тип се съпътства с имплементиране на метода computeValue().

За целта се създава обект от тип произведен на типа на връзката като се използва:

- **анонимен клас от типа на връзката**
- **статични методи Bindings.create_____Binding() , където подчертаното замества типа на връзката, Object, Integer, String**
- **Резултат от изпълнение на метод от FluentAPI**

15a.2b Binding JavaFX properties

Свързване на свойства на ниско ниво с анонимен клас

```
IntegerProperty squareSide = new SimpleIntegerProperty();  
// create a Binding type  
IntegerBinding squareArea = new IntegerBinding() {  
    { // Initialiser block of the anonymous class  
        // a list of dependencies (one or more)  
        super.bind(squareSide);  
    }  
    @Override  
    protected int computeValue() {  
        // easy to write complex binding expressions  
        return squareSide.get() * squareSide.get();  
    }  
};
```

15a.2b Binding JavaFX properties

Свързване на свойства на ниско ниво с анонимен клас

```
public class Rectangle {  
    public static void main(String[] args) {  
        DoubleProperty width = new SimpleDoubleProperty(2);  
        DoubleProperty height = new SimpleDoubleProperty(2);  
        DoubleBinding area = new DoubleBinding() {  
            { // binding to multiple properties in case of Invalidation event  
                // It is generally not necessary for you to check if the binding is invalid  
                super.bind(width, height); // initial bind to width and height  
            }  
            @Override  
            protected double computeValue() {  
                return width.get() * height.get();  
            }  
        };  
        System.out.println(area.get());  
    }  
}
```

15a.2b Binding JavaFX properties

Други способи за свързване на свойства на ниско ниво

```
IntegerProperty squareSide = new SimpleIntegerProperty();  
// allows to specify computeValue() with Lambdas  
IntegerBinding squarePerimeter =  
    Bindings.createIntegerBinding(() -> 4 * squareSide.get());  
  
// create a Binding with methods of JavaFX properties  
DoubleBinding square1Diagonal =  
    squareSide.multiply(Math.sqrt(2));  
// create a Binding with static methods of class Bindings  
IntegerBinding squarePerimeter =  
    Bindings.createIntegerBinding(() -> 4 * squareSide.get());  
// Bindings methods for arithmetic ops produce NumberBinding  
NumberBinding square2Diagonal =  
    Bindings.multiply(squareSide, Math.sqrt(2));
```


15a.2b Binding JavaFX properties

Други способи за свързване на свойства на ниско ниво

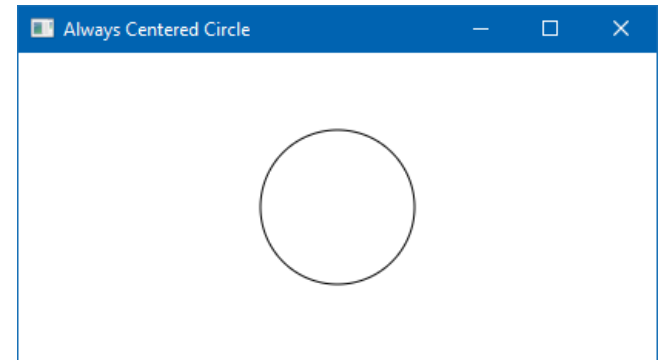
```
public static void main(String[] args) {
    LowLevelBindingExample llbe = new LowLevelBindingExample();
    llbe.squareSide.set(5);
    System.out.println("squareArea: " + llbe.squareArea.get());
    System.out.println("squareDiagonal: " + llbe.square1Diagonal.get());
    System.out.println("squareDiagonal: " + //NumberBinding
        llbe.square2Diagonal.getValue());
    System.out.println("squarePerimeter: " + llbe.squarePerimeter.get());
    llbe.squareSide.set(2);
    System.out.println("squareArea: " + llbe.squareArea.get());
    System.out.println("squareDiagonal: " + llbe.square1Diagonal.get());
    System.out.println("squareDiagonal: " + //NumberBinding
        llbe.square2Diagonal.getValue());
    System.out.println("squarePerimeter: " + llbe.squarePerimeter.get());
}
```

15a.3 Свързване на GUI свойства

```
@Override
public void start(Stage primaryStage) {
    // Create a pane to hold the circle
    Pane pane = new Pane();

    // Create a circle and set its properties
    Circle circle = new Circle();
    circle.centerXProperty().bind(pane.widthProperty().divide(2));
    circle.centerYProperty().bind(pane.heightProperty().divide(2));
    circle.setRadius(50);
    circle.setStroke(Color.BLACK);
    circle.setFill(Color.WHITE);
    pane.getChildren().add(circle); // Add circle to the pane

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 200, 200);
    primaryStage.setTitle("Always Centered Circle"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
}
```



15a.3 Свързване на свойства

На този пример е показано как да привържем JavaFX свойствата `centerX` и `centerY` на `class Circle` съответно към JavaFX свойствата `widthProperty()` и `heightProperty()` на `class Pane` в JavaFX. Забележете, че `circle.centerXProperty()` връща `centerX`, а `pane.widthProperty()` връща `width`, чиито тип е `DoubleProperty`.

JavaFX свойствата за числово свързване като `DoubleProperty` и `IntegerProperty` притежават методи за **събиране**, **изваждане**, **умножение** и **делене** на опакованата от тях стойност като връщат резултата, **опакован в същото JavaFX свойство**. По този начин, `pane.widthProperty().divide(2)` връща ново JavaFX свойство от тип `ObservableValue`, което има стойност половината от текущата стойност на свойството `width`.

15a.3 Свързване на свойства

Изразът

```
circle.centerXProperty().bind(pane.widthProperty().divide(2));
```

може да се запише накратко и като

```
centerX.bind(width.divide(2));
```

Понеже `centerX` е привързано към свойството `width.divide(2)`, тогава при промяна в стойността на `width`, стойността на `centerX` автоматично ще се промени на `width / 2`.

Забележете, че JavaFX свойствата, отговарящи на примитивни типове данни не могат да участват като операнди в аритметични оператори, а трябва да се ползват методи като `divide()`, `multiple()`, `add()`, `subtract()`.

15a.3 Свързване на свойства

В следващия пример е демонстрирано привързване променливата **d1** към променливата **d2**, където двете променливи са от тип **DoubleProperty**, а също и привързване на сумата **sum** на две числови свойства **d1** и **d3** към промени в стойностите им.

Вижда се, че след промяна на **d2** и **d3** се променят съответно **d1** и **sum**.

Видяхме вече, че при **двустранно привързване** всяка промяна в стойностите на едното от двете свойства води до промяна в стойността на другото свойство.

В този случай за привързване на свойствата се ползва метода **bindBidirectional()**.

15a.3 Свързване на свойства

```
public class BindTwo {

    public static void main(String[] args) {
        DoubleProperty d1 = new SimpleDoubleProperty(1);
        DoubleProperty d2 = new SimpleDoubleProperty(2);
        DoubleProperty d3 = new SimpleDoubleProperty(3);
        NumberBinding sum = d1.add(d3);
        // Option 2
        // NumberBinding sum = Bindings.add(d1, d3);
        // bind d1 to the value of d2
        d1.bind(d2);
        // show current values, d1 becomes equal to d2
        System.out.printf("d1: %.2f, d2: %.2f, d3: %.2f, sum(d1, d3): %.2f\n",
            d1.getValue(), d2.getValue(), d3.getValue(), sum.getValue());
        // change source properties
        d2.setValue(5);
        d3.set(4);
        // observe changes in the targets
        System.out.printf("d1: %.2f, d2: %.2f, d3: %.2f, sum(d1, d3): %.2f\n",
            d1.getValue(), d2.getValue(), d3.getValue(), sum.getValue());
    }
}
```

run-single:

d1: 2.00, d2: 2.00, d3: 3.00, sum(d1, d3): 5.00

d1: 5.00, d2: 5.00, d3: 4.00, sum(d1, d3): 9.00

BUILD SUCCESSFUL (total time: 0 seconds)

15a.3a Обработка на събитието `Change`

На практика често се използва това, че JavaFX свойствата имплементират интерфейса `ObservableValue`. В такива случаи промяната в дадено свойство се използва за извършване на промени в други свойства или обработка на данни.

Това става като към свойството, чиято промяна следим регистрираме метод за обработка на събитието `Change`.

Това е методът `changed()` и принадлежи на функционалния интерфейс `ChangeListener`. На следващия пример е демонстрирано как се използва този метод за извеждане на съобщение всеки път когато `totalAmount` промени стойността си.

15a.3a Обработка на събитието Change

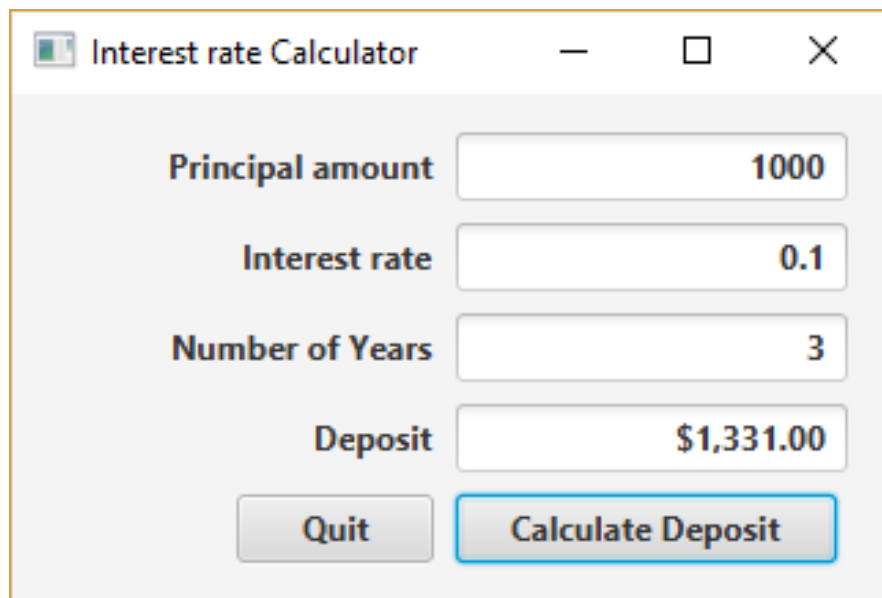
```
public class Invoice {
    private final DoubleProperty totalAmount = new SimpleDoubleProperty();
    public final double getTotalAmount() {
        return totalAmount.get();
    }
    public final void setTotalAmount(double value) {
        totalAmount.set(value);
    }
    public final DoubleProperty totalAmountProperty() {
        return totalAmount;
    }
}
```

```
public static void main(String[] args) {
    Invoice invoice = new Invoice();
    invoice.totalAmountProperty().addListener(new ChangeListener() {
        @Override
        public void changed(ObservableValue o, Object oldVal, Object newVal)
        {
            System.out.println("The invoice total ammount has changed!");
        }
    });
    // optionally write
    // invoice.totalAmountProperty().addListener(
    //     (o, oldVal, newVal) ->
    //     System.out.println("The invoice total ammount has changed!"));
    invoice.setTotalAmount(1000.00);
}
```

```
run-single:
The invoice total ammount has changed!
BUILD SUCCESSFUL (total time: 0 seconds)
```


15a.3a.1 Пример

Тук ще модифицираме приложението за пресмятане на лихви от депозит от уводната лекция в JavaFX където промяната на лихвата да става със **Slider** вместо лихвата да се въвежда като число. При това искаме текущата стойност, избрана със **Slider** –а да се извежда в текстов вид в съответния му етикет.(виж проект *InterestRateCalculatorSlider*)



Interest rate Calculator

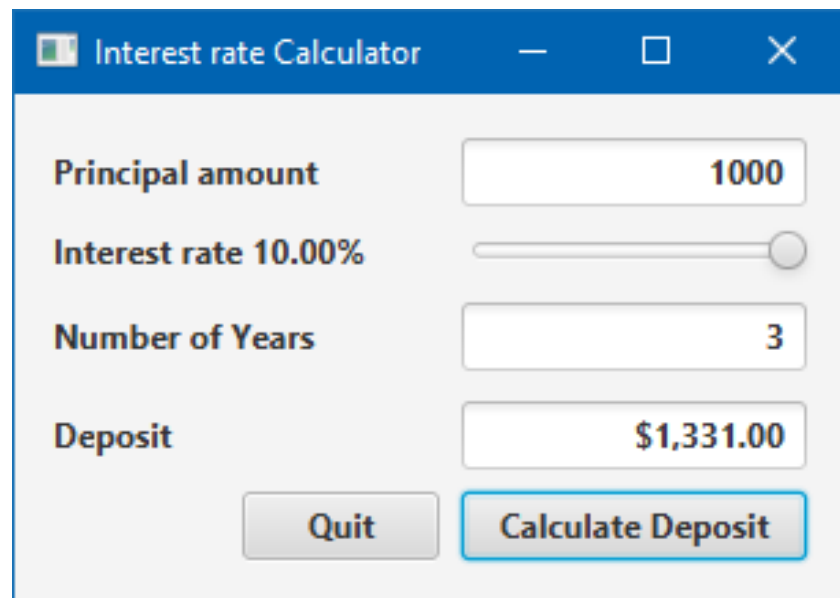
Principal amount 1000

Interest rate 0.1

Number of Years 3

Deposit \$1,331.00

Quit Calculate Deposit



Interest rate Calculator

Principal amount 1000

Interest rate 10.00%

Number of Years 3

Deposit \$1,331.00

Quit Calculate Deposit

15a.3a.1 Пример

Заменяме текстовото поле за въвеждане на лихвата със **Slider** контрол `sldInterestRate` и задаваме подразбиращ се интервал за изменение $[0, 100]$ с инкрементиране през 10 деления и подразбираща се начална стойност от 20 (2.00% лихва). Задаваме също начална стойност „Interest rate 2.00%“ на съответния на Slider-а етикет.

The screenshot shows a window titled "FXMLDocument.fxml" with a light gray background. It contains four input fields and two buttons. The first field is labeled "Principal amount" and is empty. The second field is labeled "Interest rate 2.00%" and features a slider control with a gray knob positioned at the 20 mark. The third field is labeled "Number of Years" and contains the placeholder text "Enter years". The fourth field is labeled "Deposit" and is empty. At the bottom, there are two buttons: "Quit" and "Calculate Deposit".

15a.3a.1 Пример

Редактираме сорс кода в FXMLDocumentController.java.

```
public class FXMLDocumentController {
    // formatters for CURRENCY and percentages
    private static final NumberFormat CURRENCY      = NumberFormat.getCurrencyInstance();
    private static final NumberFormat percent       = NumberFormat.getPercentInstance();

    private BigDecimal interestRate = new BigDecimal(0.02); // 2% default

    @FXML
    private Label lblInterestRate;
    @FXML
    private TextField txtPrincipalAmount;
    @FXML
    private Slider sldInterestRate;
    @FXML
    private TextField txtTotalDeposit;
    @FXML
    private TextField txtNumberYears;
    @FXML
    void btnQuitClicked(ActionEvent event) {
        System.exit(0);
    }
}
```

15a.3a.1 Пример

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

15a.3.1 Пример

@FXML

```
public void initialize(URL url, ResourceBundle rb) {
    CURRENCY.setRoundingMode(RoundingMode.HALF_UP);
    // 0-4 rounds down, 5-9 rounds up
    // Add a listener for changes to tipPercentageSlider's value
    // Option 1 use Lambda statements
    sldInterestRate
        .valueProperty()
        .addListener( (ov, oldValue, newValue) ->
            {
                interestRate = BigDecimal.valueOf(newValue.intValue()/1000.);
                lblInterestRate.setText(String.format("Interest rate %.2f%%",
                    interestRate.multiply(BigDecimal.valueOf(100.))));
            }
        );
    // Option 2 use anonymous class
    // sldInterestRate.valueProperty().addListener(
    //     new ChangeListener<Number>()
    //     {
    //         @Override
    //         public void changed(ObservableValue<? extends Number> ov,
    //             Number oldValue, Number newValue)
    //         {
    //             interestRate = BigDecimal.valueOf(newValue.intValue() / 100.0);
    //             lblAmount.setText("Interest rate" + percent.format(interestRate));
    //         }
    //     }
    // );
}
```

15a.3.1 Пример

Interest rate Calculator

Principal amount

Interest rate 2.00%

Number of Years

Deposit

Interest rate Calculator

Principal amount

Interest rate 5.50%

Number of Years

Deposit

Interest rate Calculator

Principal amount

Interest rate 1.60%

Number of Years

Deposit

15a.4 Рисуване в графичния контекст

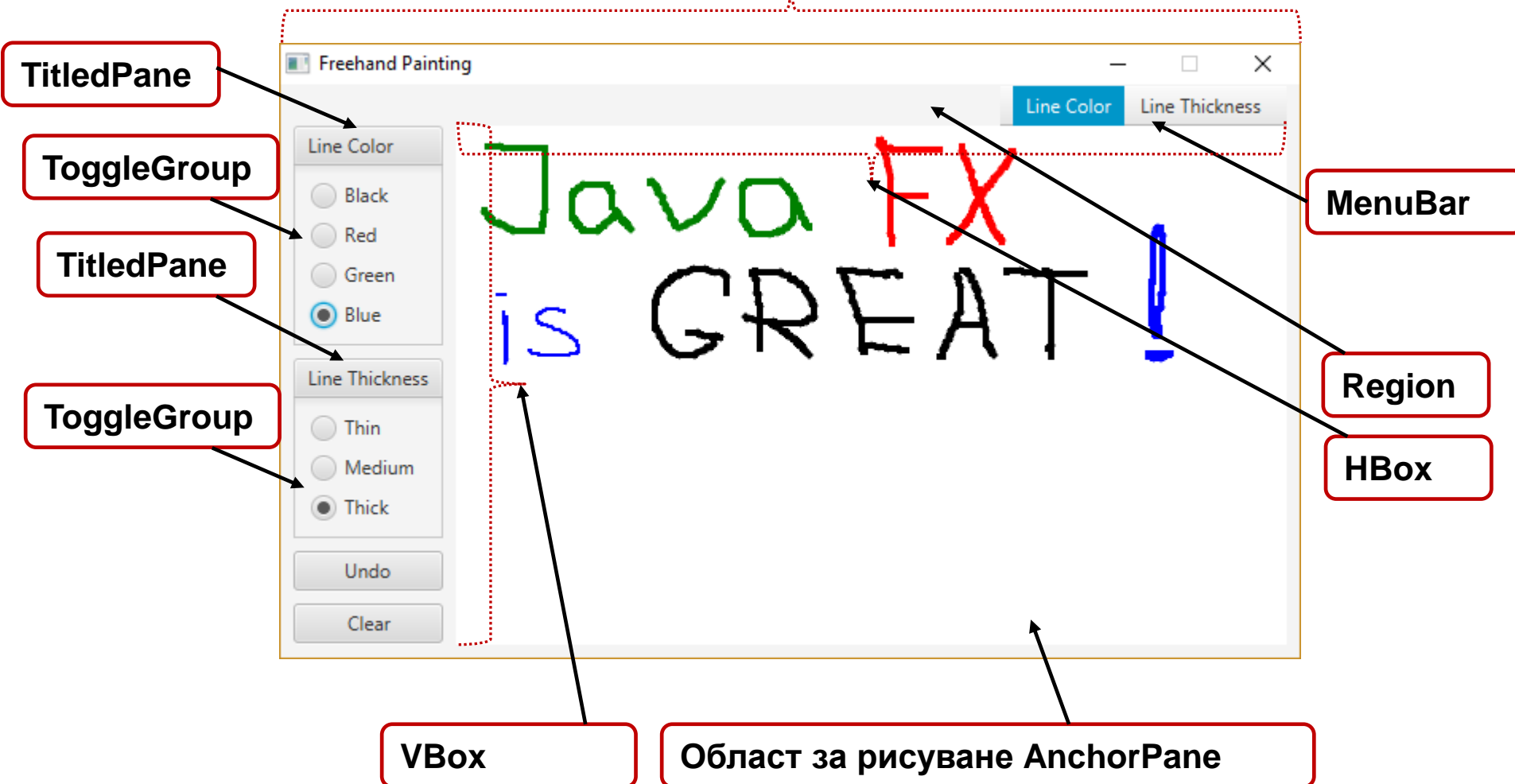
Тук ще създадем приложение Freehand Painting, което позволява да се рисува като се отследява върха на курсора на мишката в графичния контекст на AnchorPane.

Приложението ще позволява да се избира цвят и дебелина на рисуваната линия посредством меню и групи от радио бутони.

Първоначално ще изградим модела на GUI с помощта на SceneBuilder, а впоследствие ще напишем методите за обработка на събитията.

15a.4 Рисуване в графичния контекст

GridPane 2 реда x 2 колони



15a.4 Методи за обработка на събитието Mouse

onMouseClicked	Изпълнява се за даден Възел при натискане и отпускане на бутон на неподвижна мишка, когато мишката е в границите на възела
onMouseDragEntered	Изпълнява се за даден Възел при влачене мишка, когато мишката навлезе в границите на възела
onMouseDragExited	Изпълнява се за даден Възел при влачене мишка, когато мишката напусне границите на възела
onMouseDragged	Изпълнява се за даден Възел при влачене мишка, когато влаченето започне в границите на възела. Изпълнението на метода продължава, докато не спре влаченето
onMouseDragOver	Изпълнява се за даден Възел при влачене мишка, когато влаченето е започнало в границите на друг възел и продължава в този възел
onMouseDragReleased	Изпълнява се за даден Възел , когато спира влаченето, започнало в границите на този възел
onMouseEntered	Изпълнява се за даден Възел , когато мишката навлезе в границите на този възел
onMouseExited	Изпълнява се за даден Възел , когато мишката напусне границите на този възел
onMouseMoved	Изпълнява се за даден Възел , когато мишката навлезе без натиснат бутон в границите на този възел
onMousePressed	Изпълнява се за даден Възел , когато се натисне бутон на мишката в границите на този възел
onMouseReleased	Изпълнява се за даден Възел , когато се отпусне бутон на мишката в границите на този възел

15a.4 Методи за обработка на събитието Mouse

Всеки от тези методи има единствен параметър, `MouseEvent`. Този параметър представлява обектът на събитието `Mouse`, който се използва за обработка на това събитие. Важни свойства на този обект са:

- `getX()` - връща `double`, `x` координатата на върха на курсора на мишката
- `getY()` - връща `double`, `y` координатата на върха на курсора на мишката
- `getButton()` - връща `MouseButton`, за разпознаване кой бутон на мишката е натиснат
- `getClickCount()` - връща `int`, брой пъти е кликнат бутон на мишката

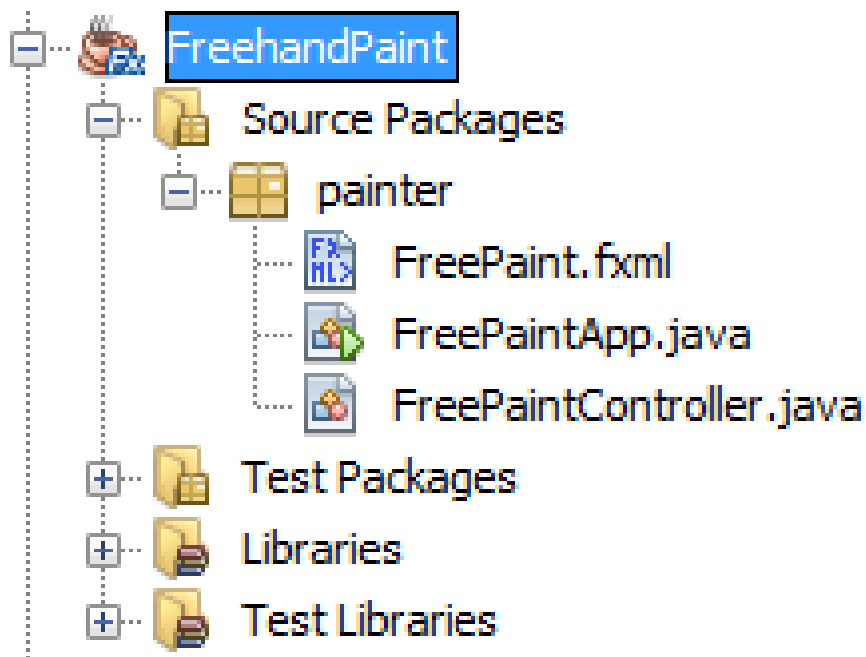
15a.4 Методи за обработка на събитието Mouse

```
public void handle(MouseEvent event) {  
    MouseButton button = event.getButton();  
    if(button == MouseButton.PRIMARY){  
        System.out.println("PRIMARY button clicked on button");  
    }else if(button == MouseButton.SECONDARY){  
        System.out.println("SECONDARY button clicked on button");  
    }else if(button == MouseButton.MIDDLE){  
        System.out.println("MIDDLE button clicked on button");  
    }  
}
```

15a.5 Изпълнение на проекта

Следваме стъпките за създаване на JavaFX проект с NetBeans, описани в предишната лекция.

За удобство преименуваме файловете на Сцената, Контролера и Приложението по следния начин



15a.5 Изпълнение на проекта

При това е **важно да се редактира** елемента `fx:controller` в корена на дървото от възли във FXML файла на Сцената, така че **стойността му да съвпада с наименованието на файла на Контролера** (натиска се десен бутон върху FXML-файла и се избира `Edit` от помощното меню)

```
<GridPane hgap="8.0" maxHeight="-Infinity" maxWidth="-Infinity"
:
fx:controller="painter.FreePaintController">
```

15a.5a Изграждане на графичния интерфейс

Избор на Контейнер за подреждане на възлите

Зададеният модел на графичния интерфейс има два реда и две колони.

Във втората колона на първия ред е разположено менюто, а вторият ред съдържа бутоните (първата колона) и графичната област за рисуване (втората колона).

Поради това заменяме графичния интерфейс, зададен по подразбиране с `GridPane` с два реда и две колони. Задаваме предпочитани дължина 640 и ширина 360 на `GridPane`, както и 8 px за `Vgap`, `Hgap` и `Padding` отстояние от границите на прозореца

15а.5а Изграждане на графичния интерфейс

Inspector

Properties : GridPane

Layout : GridPane

Internal

Hgap: 8

Vgap: 8

Padding: 8 > 8 8 8

Size

Min Width: USE_PREF_SIZE

Min Height: USE_PREF_SIZE

Pref Width: 640

Pref Height: 360

Max Width: USE_PREF_SIZE

Max Height: USE_PREF_SIZE

Width: 640

Height: 360

15a.5a Изграждане на графичния интерфейс

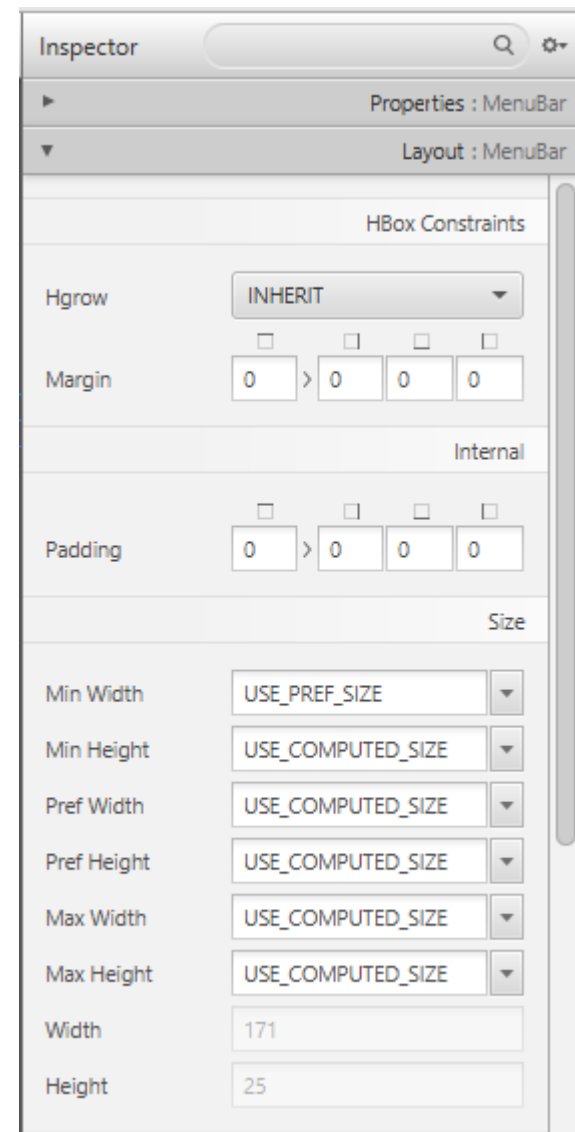
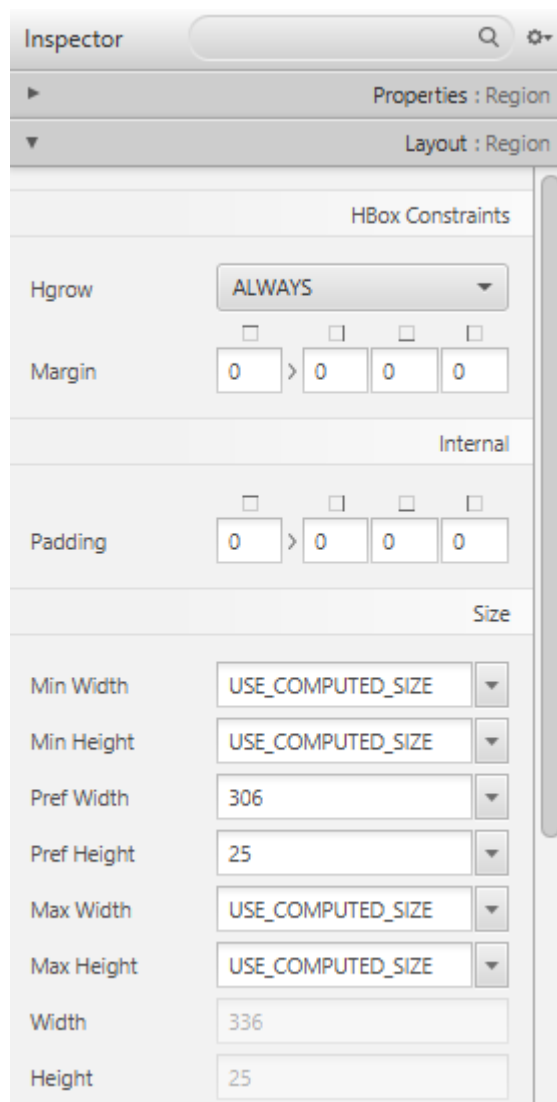
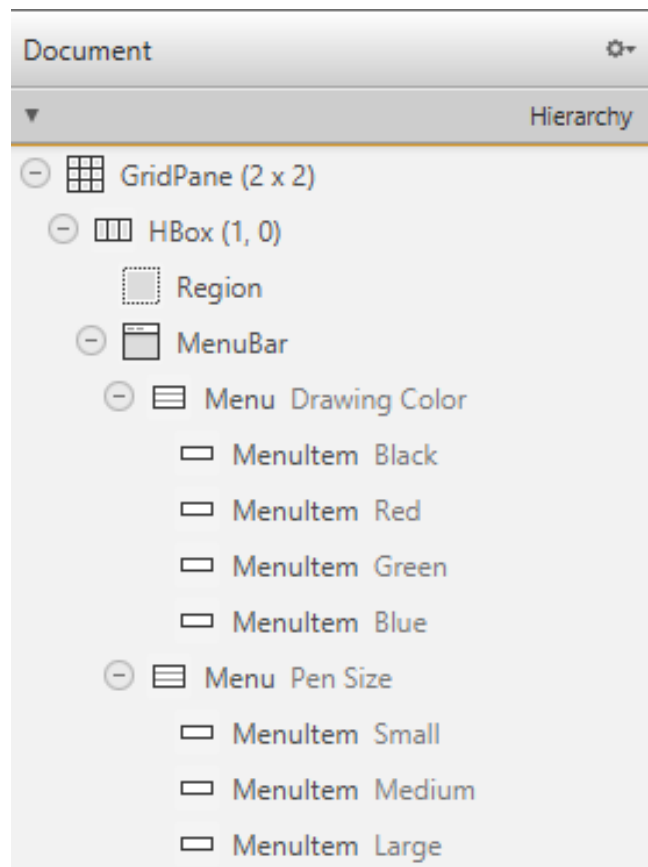
Графично съдържание в първия ред на GridPane

Добавяме HBox във втората клетка на първия ред.

В този HBox добавяме MenuBar от Раздела Controls на SceneBuilder. Добавяме също Region контрола пред MenuBar, за да бъде менюто винаги в дясната част на прозореца независимо от промяната в ширината му (HGrow = ALWAYS). Преименуваме елементите на менюто и добавяме съответен брой MenuItem към тях.

Задаваме fx:id идентификатори в съответствие с Изменената Унгарска нотация

15a.5a Изграждане на графичния интерфейс

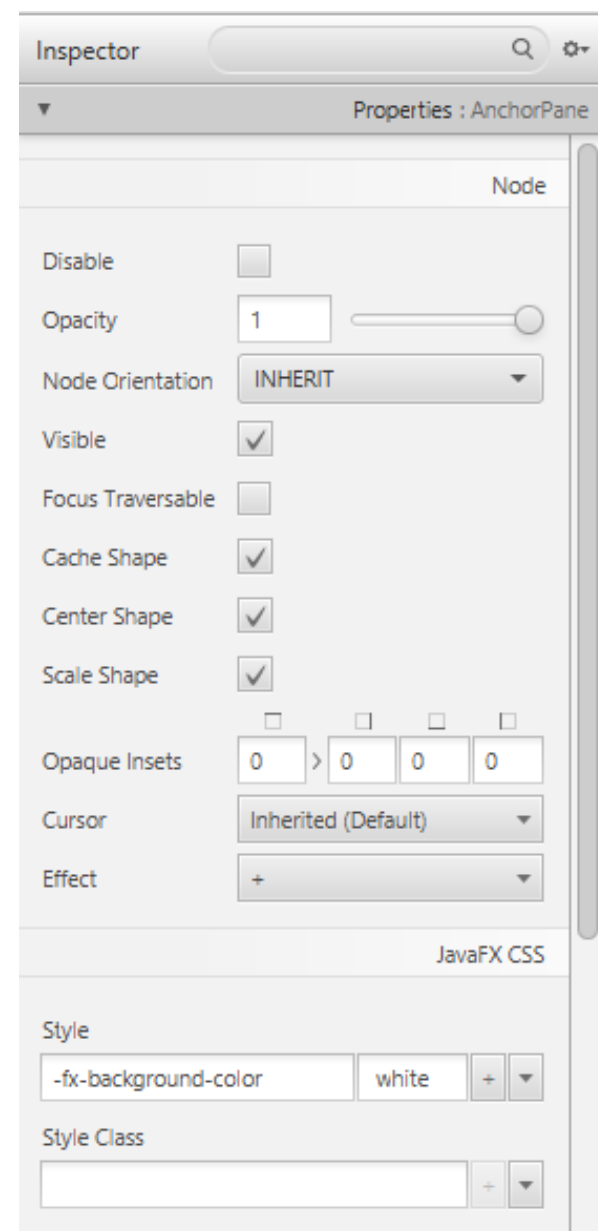
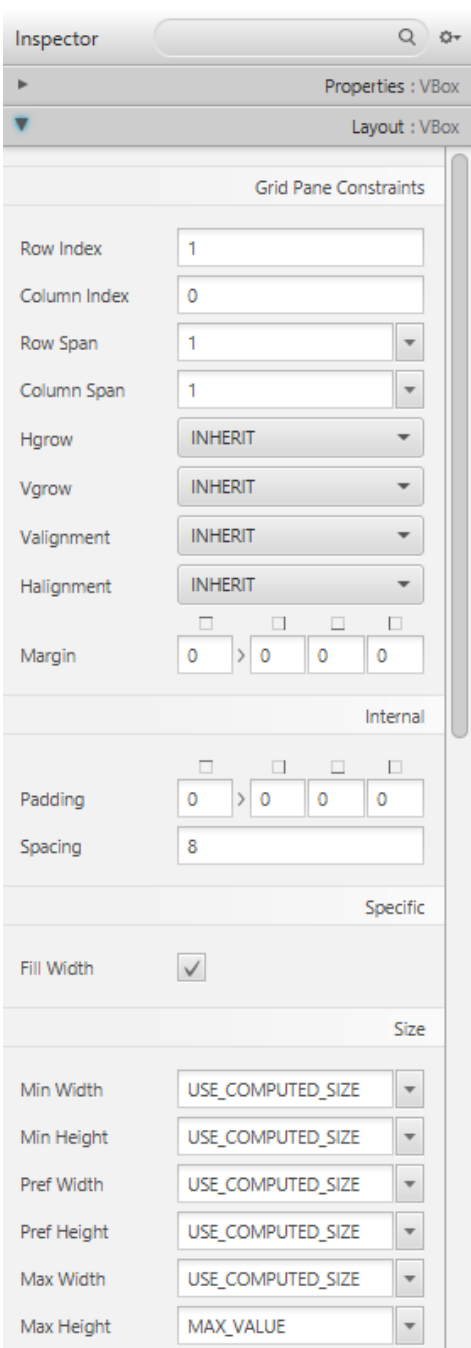


15a.5a Изграждане на графичния интерфейс

Графично съдържание на втория ред на GridPane

На втория ред от GridPane добавяме съответно VBox (област за групите бутони) и AnchorPane (област за рисуване). Задаваме MAX_VALUE за maxHeight свойството на VBox-а, за да запълни цялата височина на клетката в GridPane. Задаваме също 8 px за Spacing свойството на VBox, за да осигурим пространство между групите бутони, които ще добавим там.

Задаваме цвят за фон на AnchorPane като изписваме -fx-background-color в свойството style в Раздела Properties и избираме white за цвят



15a.5a Изграждане на графичния интерфейс

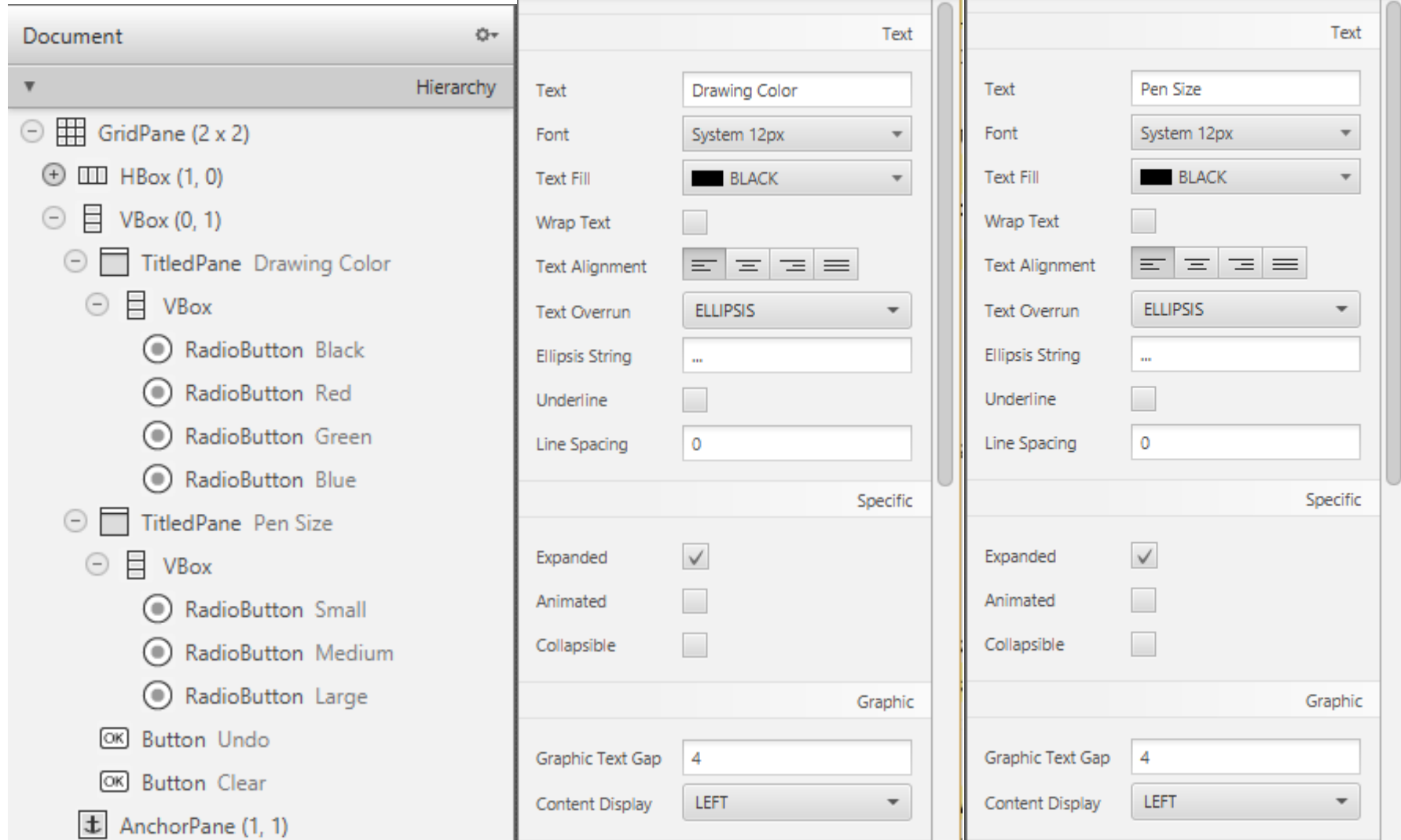
Групи от бутони

Добавяме две `TitledPane` (`Drawing Color`, `Pen Size`) и два `Button` (`Undo`, `Clear`) към `Vbox`-а.

Изключваме свойството `Collapsible` на двете `TitledPane`. Заменяме стандартния `AnchorPane` на всяко `TitledPane` с `Vbox`, в който ще добавим `RadioButton`-и. Задаваме `Spacing = 8` за тези два `Vbox`-а, за да има разстояние между `RadioButton`-ите. Добавяме и самите `RadioButton`-и в съответните `Vbox`-а.

Задаваме `fx:id` идентификатори в съответствие с Изменената Унгарска нотация.

15a.5a Изграждане на графичния интерфейс



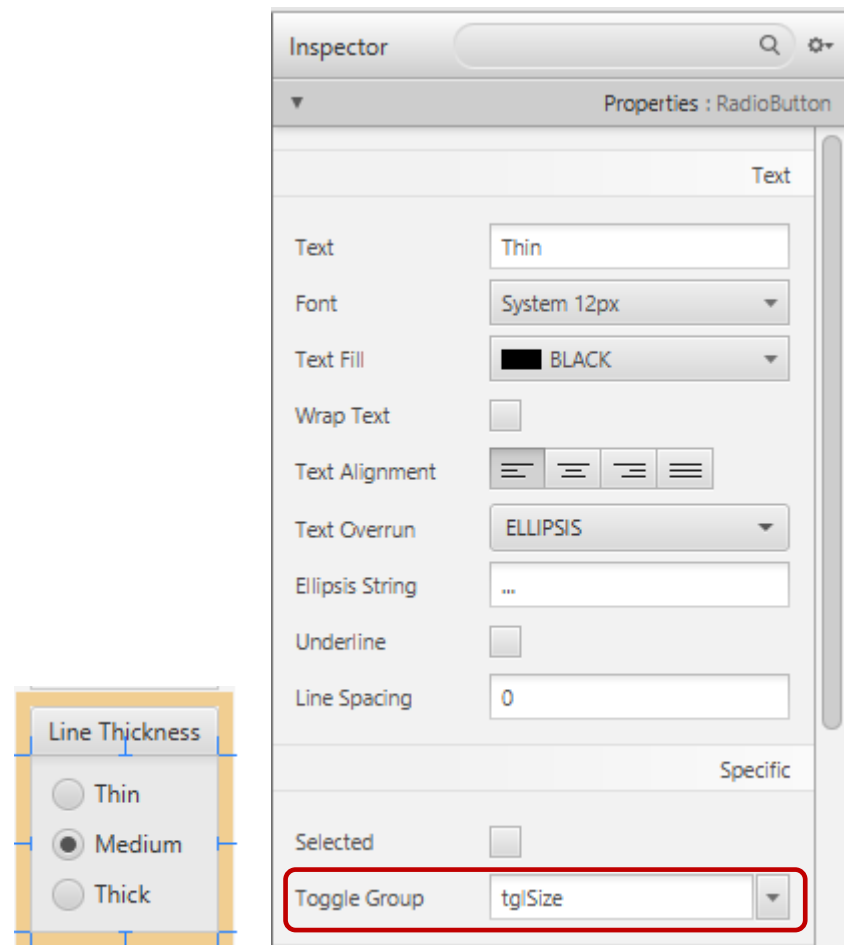
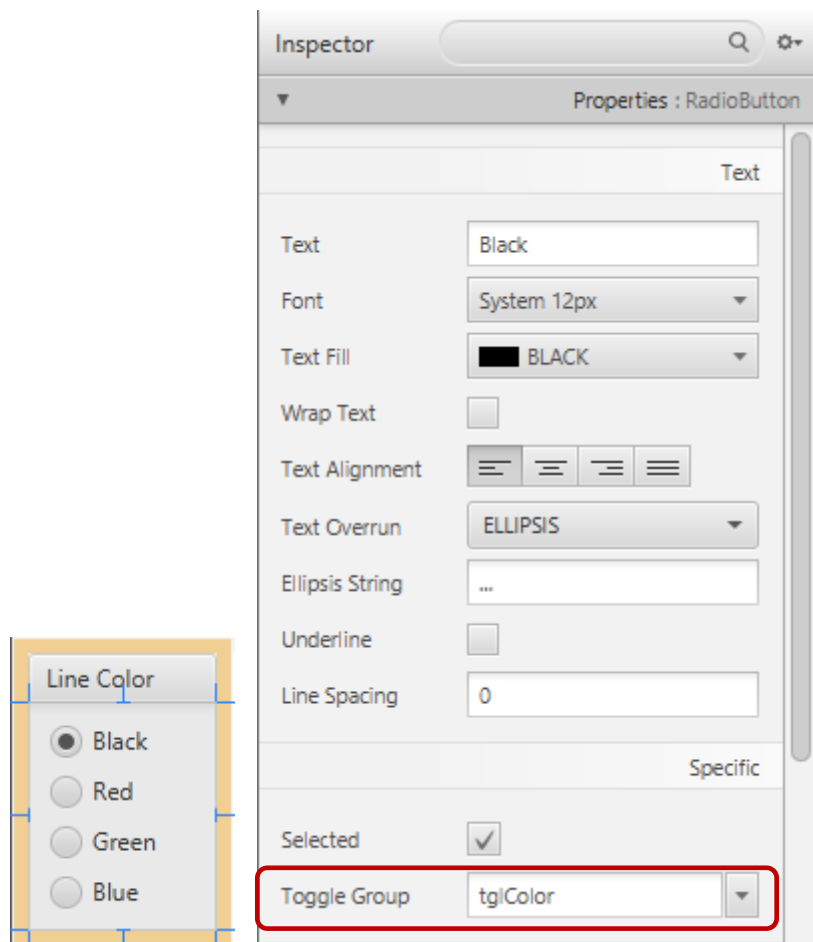
15a.5a Изграждане на графичния интерфейс

Създаване на ToggleGroup-и за бутоните

Накрая групираме `RadioButtons` в `ToggleGroup` като за всеки бутон изписваме името на групата, към която принадлежи в свойството му `ToggleGroup`. При създаването на дървото с Възли, обект от `ToggleGroup` ще бъде създаден и рефериран със съответното име в програмата.

Именуваме тези групи в съответствие с Изменената Унгарска Нотация съответно като `tglColor` и `tglSize`

15a.5a Изграждане на графичния интерфейс



15a.5a Изграждане на графичния интерфейс

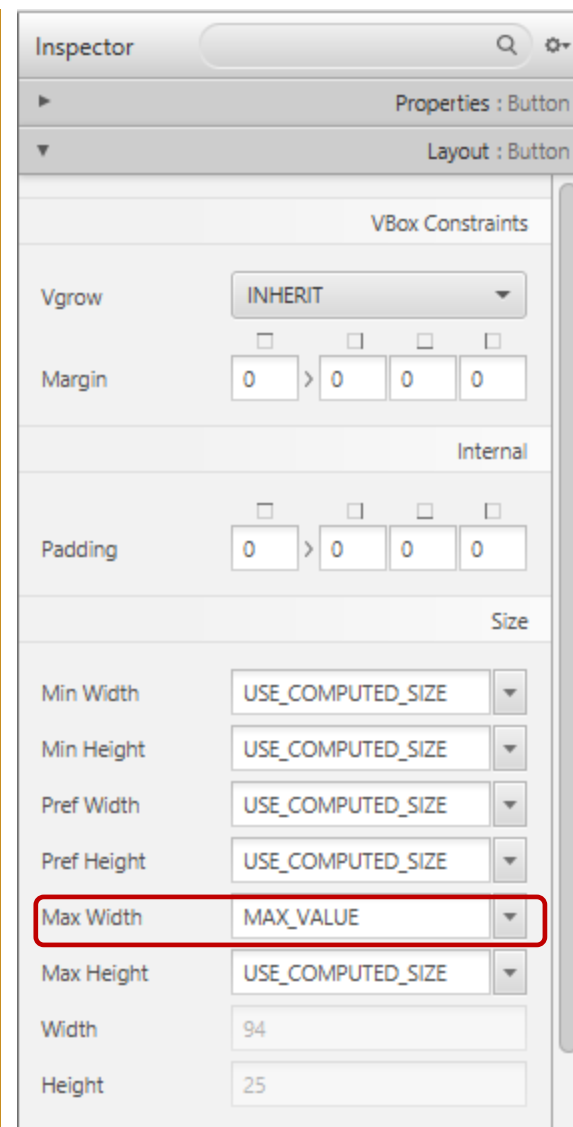
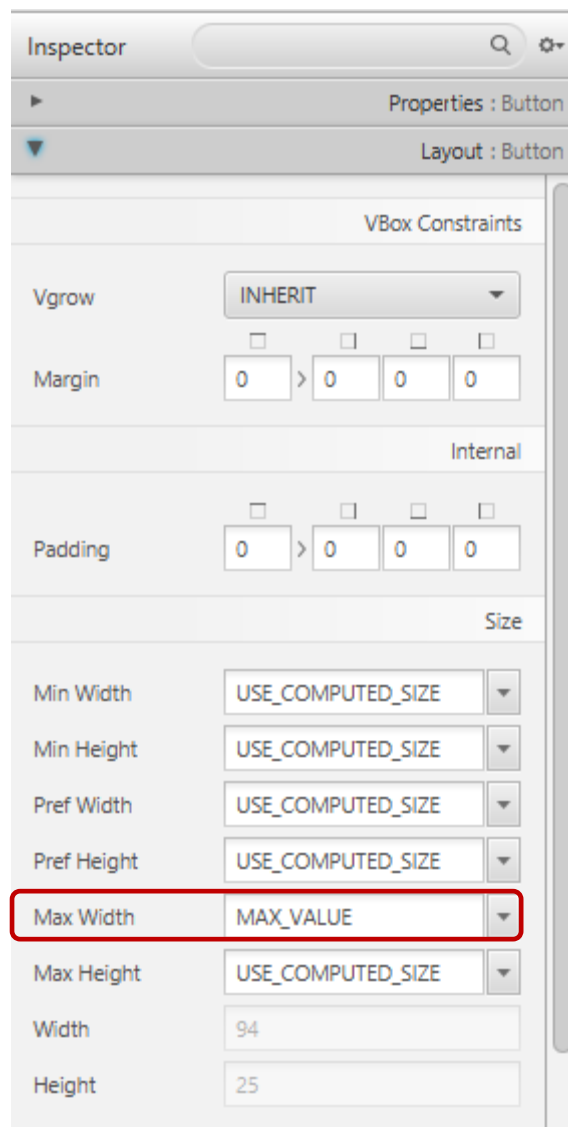
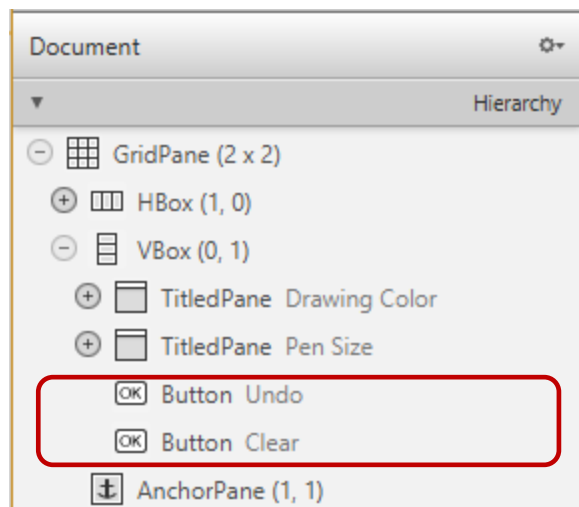
Създаване на Undo и Clear за бутони

Добавяме последователно два бутона Button в края на VBox-а след TitledPane за Pen Size.

Задаваме свойството им `Max width= MAX_VALUE`, за да заемат цялата ширина на VBox-а .

Именуваме тези групи в съответствие с Изменената Унгарска Нотация съответно като `btnUndo` и `btnClear`

15a.5a Изграждане на графичния интерфейс



15a.5a Изграждане на графичния интерфейс

Фиксиране на ширината на първата колона

Ширината на първата колона от GridPane се фиксира, след като са добавени в нея всички контроли. За целта инициализираме

```
Max width= USE_PREF_SIZE,
```

```
Min width =USE_COMPUTED_SIZE
```

```
Pref width=USE_COMPUTED_SIZE
```

Това позволява на тази колона да остане с фиксирана ширина, а да се мени единствено ширината на втората колона при промяна на ширината на графичния прозорец.

15a.5b Редактиране на класа на приложението

Класът на приложението `FreePaintApp` се създава по подразбиране от `NetBeans`. Той е производен на клас `Application`.

Една възможна редакция в него е да се добави **заглавие** на графичния прозорец

```
// Set the Window title  
stage.setTitle("Freehand Painting");
```

или **да се забрани промяната в размерите** на прозореца и размерът на прозореца да е като на сцената

```
// Set Window not resizable  
stage.sizeToScene();  
stage.resizableProperty().setValue(Boolean.FALSE);
```

15a.5c Редактиране на класа на Контролера

Класът на Контролера

`FreePaintController.java` съдържа сорс код на Java за обработка на събития в графичния интерфейс, описани с `FXML` във `FreePaint.fxml`.

Освен декларации на ресурсите, използвани в графичния интерфейс, **този клас може да съдържа допълнителни данни и методи**, които позволяват да се реализира интерактивност в общуването с потребителите на този интерфейс.

Характерно е използването на **анотацията `@FXML` съвместно с всички ресурси и методи, описани с `FXML` във `FreePaint.fxml`**

```

@FXML    private ResourceBundle resources;
@FXML    private URL location;

@FXML    private MenuItem mnuBlack;
@FXML    private MenuItem mnuRed;
@FXML    private MenuItem mnuGreen;
@FXML    private MenuItem mnuBlue;
@FXML    private MenuItem mnuSmall;
@FXML    private MenuItem mnuMedium;
@FXML    private MenuItem mnuLarge;

@FXML    private RadioButton rbtBlack;
@FXML    private ToggleGroup tglColor;
@FXML    private RadioButton rbtRed;
@FXML    private RadioButton rbtGreen;
@FXML    private RadioButton rbtBlue;
@FXML    private RadioButton rbtSmall;
@FXML    private ToggleGroup tglSize;
@FXML    private RadioButton rbtMedium;
@FXML    private RadioButton rbtLarge;

@FXML    private Button btnUndo;
@FXML    private Button btnClear;
@FXML    private AnchorPane drawingAreaAnchorPane;

@FXML
private void clearButtonPressed(ActionEvent event) {
    // clear the drawing area on AnchorPane
    drawingAreaAnchorPane.getChildren().clear();
}

```

15a.5c Редактиране на класа на Контролера

Като пример за допълнителни данни служат константите за задаване на дебелината на линията, с която се рисува, координатите на началото на текущо рисуваната линия, както и данни за текущо избраните стойности на цвят и дебелина на линия .

```
public class FreePaintController {  
    // Sample line thickness  
    private static final int THIN_LINE = 1;  
    private static final int MEDIUM_LINE = 2;  
    private static final int THICK_LINE = 4;  
    private double startingPointX, startingPointY,  
        endPointX, endPointY;  
    // instance variables for managing the FreePaint drawing  
    private int lineStroke; // line thickness  
    private Paint lineColor; // line color
```

15a.5c Редактиране на класа на Контролера

Всеки JavaFX възел (Node) има свойство `userData`, които позволява на възела да се свърже с потребителски дефинирани свойства от произволен тип (Object). Например, по този начин елементите от менюто или радио бутоните могат да предоставят данни за настройката на цвят и дебелина на линията, която е свързана с тях. Настройките за цвят ще задаваме с константи от тип `Color`, а за дебелината на линията на рисуване ще използваме набор от три статични константи `THIN_LINE`, `MEDIUM_LINE`, `THICK_LINE`

15a.5c Редактиране на класа на Контролера

Методът `initialize()` на Контролера служи за инициализиране на данни преди да се изобрази Сцената.

В нашия пример, **там инициализираме подразбиращите се стойности за цвят и дебелина** на линия, както и свойството `userData` на елементите на менюто и радио бутоните.

Свойството `userData` е удобно средство да извлечем конкретен цвят или дебелина на линията за рисуване при избиране на елемент на меню или бутон, свързан с тези характеристики.

@FXML

```
private void initialize() {
```

```
// default values
```

```
    lineStroke = FreePaintController.MEDIUM_LINE;    // default thickness
```

```
    lineColor = Color.BLACK;                          // default color
```

```
// Initialize the userData property
```

```
mnuBlack.setUserData(Color.BLACK);
```

```
rbtBlack.setUserData(Color.BLACK);
```

```
mnuRed.setUserData(Color.RED);
```

```
rbtRed.setUserData(Color.RED);
```

```
mnuGreen.setUserData(Color.GREEN);
```

```
rbtGreen.setUserData(Color.GREEN);
```

```
mnuBlue.setUserData(Color.BLUE);
```

```
rbtBlue.setUserData(Color.BLUE);
```

```
mnuSmall.setUserData(FreePaintController.THIN_LINE);
```

```
rbtSmall.setUserData(FreePaintController.THIN_LINE);
```

```
mnuMedium.setUserData(FreePaintController.MEDIUM_LINE);
```

```
rbtMedium.setUserData(FreePaintController.MEDIUM_LINE);
```

```
mnuLarge.setUserData(FreePaintController.THICK_LINE);
```

```
rbtLarge.setUserData(FreePaintController.THICK_LINE);
```

```
}
```

15a.5c Редактиране на класа на Контролера

Важно е всяка променлива дефинирана с `fx:id` във `FXML` файла на Сцената да се декларира с анотацията `@FXML` във файла на Контролера със същия идентификатор.

15a.5d Рисуване в графичния контекст

AnchorPane е Контейнер за подреждане на възли, който позволява на съдържащите се в него възли да се закотвят на фиксирани разстояния от неговите граници.

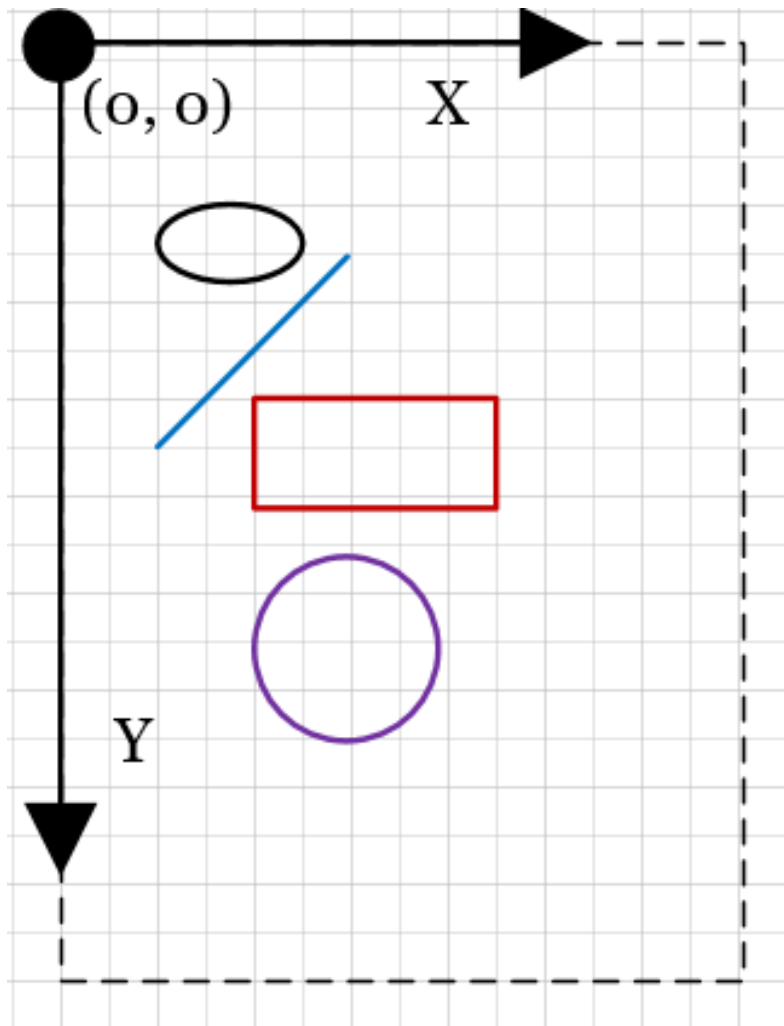
Подобно на другите Контейнери за подреждане , той позволява в него да се добавят обекти на **JavaFX** форми (2D, 3D) от `javafx.scene.shape` на конкретни **X-Y** координати. Тези координати се измерват спрямо координатна система, чието начало е позиционирано в горния ляв ъгъл на **AnchorPane**. Оста **X** е насочена хоризонтално наляво, а оста **Y** е насочена вертикално надолу.

15a.5d Рисуване в графичния контекст

AnchorPane подобно на другите Контейнери за подреждане на възли има методи за определяне на неговите текущи дължина и ширина-

- **getWidth()** връща ширината като **double** стойност.
- **getHeight()** връща височината като **double** стойност.

15a.5d Рисуване в графичния контекст



15a.5d Рисуване в графичния контекст

Рисуването на JavaFX форми се извършва по следния начин:

1. Определят се координатите `X``Y`, изисквани за позициониране на формата в координатната система на Контейнера за подреждане. Тези координати са от тип `double`. Те могат да се зададат експлицитно или като се прочетат от текущото положение на курсора на мишката като се използват свойствата `x` и `y` на обекта на събитието `MouseEvent`. Обектът на събитието `MouseEvent` се подава като параметър на всеки от методите за обработка на събитието `Mouse`

15a.5d Рисуване в графичния контекст

2. Създава се обект от съответната JavaFX форма (**линия, правоъгълник, елипса, кръг, дъга** и пр.) като се подават необходимите параметри за положение и размери на формата като `double` стойности
3. Задават се нужните свойства за цвят (`setStroke(Paint p)`), дебелина (`setStrokeWidth(double w)`), цвят за запълване(`setFill(Paint p)`)
4. Добавя се геометричния обект към списъка с Възли на Контейнера

1	//x,y of starting point, x,y of ending point
2	Line line = new Line(0., 0., 100., 600.);
3	line.setStroke(Color.RED);
4	line.setStrokeWidth(4.);
5	layoutContainer.getChildren().add(line);
6	//x,y of upper left corner, width, height
7	Rectangle rectangle = new Rectangle(15a.,20.,100.,200.);
8	rectangle.setStroke(Color.BLACK);
9	rectangle.setFill(null);
10	rectangle.setStrokeWidth(2.);
11	layoutContainer.getChildren().add(rectangle);
12	//x,y of center, x- radius, y- radius
13	Ellipse ellipse = new Ellipse(100., 200., 80., 40.);
14	ellipse.setStroke(Color.BLACK);
15	ellipse.setFill(null);
16	ellipse.setStrokeWidth(4.);
17	layoutContainer.getChildren().add(ellipse);
18	//x,y of center, radius
19	Circle circle = new Circle(100., 100, 80.);
20	circle.setStroke(Color.BLACK);
21	circle.setFill(Color.BLUE);// Filled circle
22	circle.setStrokeWidth(3.);
23	layoutContainer.getChildren().add(circle);
24	//x,y of starting point
25	Text text = new Text (15a., 20., "This is an opaque text sample");
26	text.setFont(Font.font("Verdana", FontPosture.ITALIC, 15));
27	text.setFill(Color.RED);// by default aplha = 1 (opaque)
28	layoutContainer.getChildren().add(text);
29	Text texts = new Text (15a., 40., "This is a semitransparent text sample");
30	texts.setFont(Font.font("Verdana", FontPosture.ITALIC, 15));
31	texts.setFill(Color.rgb(0,0,0,0.5));//black, semivisible (50% transparent)
32	layoutContainer.getChildren().add(texts);

15a.6 Рисуване на свободна линия

Свободна линия получаваме като свързваме всеки две съседни точки с JavaFX линия при „влачене“ на мишката. За целта се обработва метода `onMouseDragged()`, който от своя страна изпълнява метода `drawingAreaMouseDragged (MouseEvent event)`, написан в Контролера

```
@FXML
private void drawingAreaMouseDragged(MouseEvent event) {
    endPointX = event.getX();
    endPointY = event.getY();
    Line line = new Line(startingPointX, startingPointY, endPointX, endPointY);
    line.setStroke(lineColor);
    line.setStrokeWidth(lineStroke);
    drawingAreaAnchorPane.getChildren().add(line);
    startingPointX= endPointX;
    startingPointY =endPointY;
}
```

15a.6 Рисуване на свободна линия

Координатите `XY` на началната точка (данните `startingPointX`, `startingPointY`) прочитаме от обекта на събитието `MouseEvent` при натискане на бутон на мишката в областта за рисуване (`AnchorPane drawingAreaAnchorPane`)

За целта обработваме метода `onMousePressed()`, който извиква метода `void drawingAreaMousePressed(MouseEvent event)`, написан в Контролера

```
@FXML
void drawingAreaMousePressed(MouseEvent event) {
    startingPointX = event.getX();
    startingPointY = event.getY();
}
```

15a.7 Избиране на цвят и дебелина на линията

Изборът на цвят и дебелина се осъществява с предварително дефинирани стойности посредством радиобутони и елементи от менюто.

За целта се обработва събитието `Action`, което се създава при кликване на бутон или елемент от менюто.

При обработка на това събитие се идентифицира източника на събитието и се **прочита стойността на `userData`, съхранявана в него**. Така прочетената стойност се използва за инициализиране на текущата стойност за цвят или дебелина на линия.

15a.7 Избиране на цвят и дебелина на линията

1	@FXML
2	private void sizeMenuSelected(ActionEvent event) {
3	// user data for each lineStroke RadioButton is the corresponding thickness
4	lineStroke
5	= (int) ((MenuItem) event.getSource()).getUserData();
6	}
7	
8	@FXML
9	private void sizeRadioButtonSelected(ActionEvent event) {
10	// user data for each lineStroke RadioButton is the corresponding thickness
11	lineStroke
12	= (int) tglSize.getSelectedToggle().getUserData();
13	}
1	@FXML
2	private void colorMenuSelected(ActionEvent event) {
3	// user data for each color RadioButton is the corresponding Color
4	lineColor
5	= (Color) ((MenuItem) event.getSource()).getUserData();
6	}
7	
8	@FXML
9	private void colorRadioButtonSelected(ActionEvent event) {
10	// user data for each color RadioButton is the corresponding Color
11	lineColor
12	= (Color) tglColor.getSelectedToggle().getUserData();
13	}

15a.8 Редактиране на нарисувана графика

Редактирането на нарисуваната графика е демонстриране с изтриване на последния графичен обект (**Undo**) и изтриване на всички нарисувани графични форми(**Clear**).

Undo се извършва като се изтрива последния добавен обект в списъка с възли в Контейнера за подреждане (**AnchorPane drawingAreaAnchorPane**)

Clear се извършва като се изтриват всички добавени обекти в списъка с възли в Контейнера за подреждане (**AnchorPane drawingAreaAnchorPane**)

15a.8 Редактиране на нарисувана графика

1	@FXML
2	private void undoButtonPressed(ActionEvent event) {
3	int count = drawingAreaAnchorPane.getChildren().size();
4	// if there are any shapes remove the last one added
5	if (count > 0) {
6	drawingAreaAnchorPane.getChildren().remove(count - 1);
7	}
8	}
9	@FXML
10	private void clearButtonPressed(ActionEvent event) {
11	// clear the drawing area on AnchorPane
12	drawingAreaAnchorPane.getChildren().clear();
13	}

15a.9 Обработка на събития

Събитията се използват за предаване на данни на приложението за действия, извършени от потребителя с цел да се отговори на тези действия. JavaFX предоставя необходимите средства за регистриране на събития, насочване на събитието през верига от Възли към Възела източника на събитието и обработка на събитието по съответен начин.

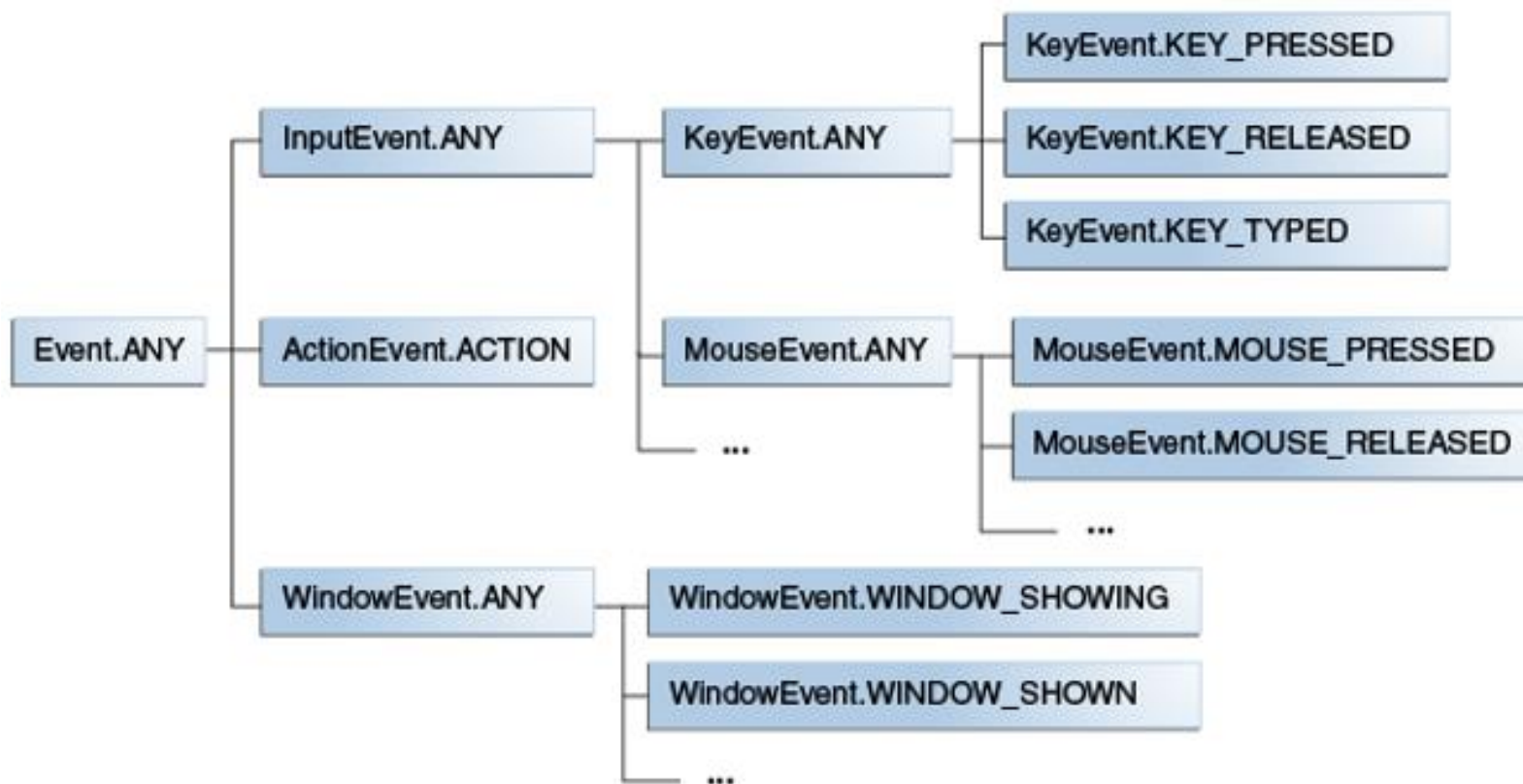
15a.9a Свойства на събитие

Събитията се характеризират със следните свойства:

- **Тип на събитието (EventType)**, което се е случило. Пример, `KeyEvent MOUSE_PRESSED`, `MOUSE_TYPED`, `MOUSE_RELEASED`
- **Източник на събитието (Source)**- Възел, който обработва събитието. Мени се при обработката на събитието
- **Цел (Target)**- Възел от Сцената, към който е приложено действието на потребителя. Не се мени при обработката на събитието. Това е **Възелът**, който е бил на фокус при регистриране на събитието при `Key` или **Възелът**, върху който е бил курсорът на мишката при `MouseEvent`

15a.9 Обработка на събития

Йерархията на наследственост на типове от събития позволява да се филтрират различни типове събития



15а.10 Обработка на събитие Key

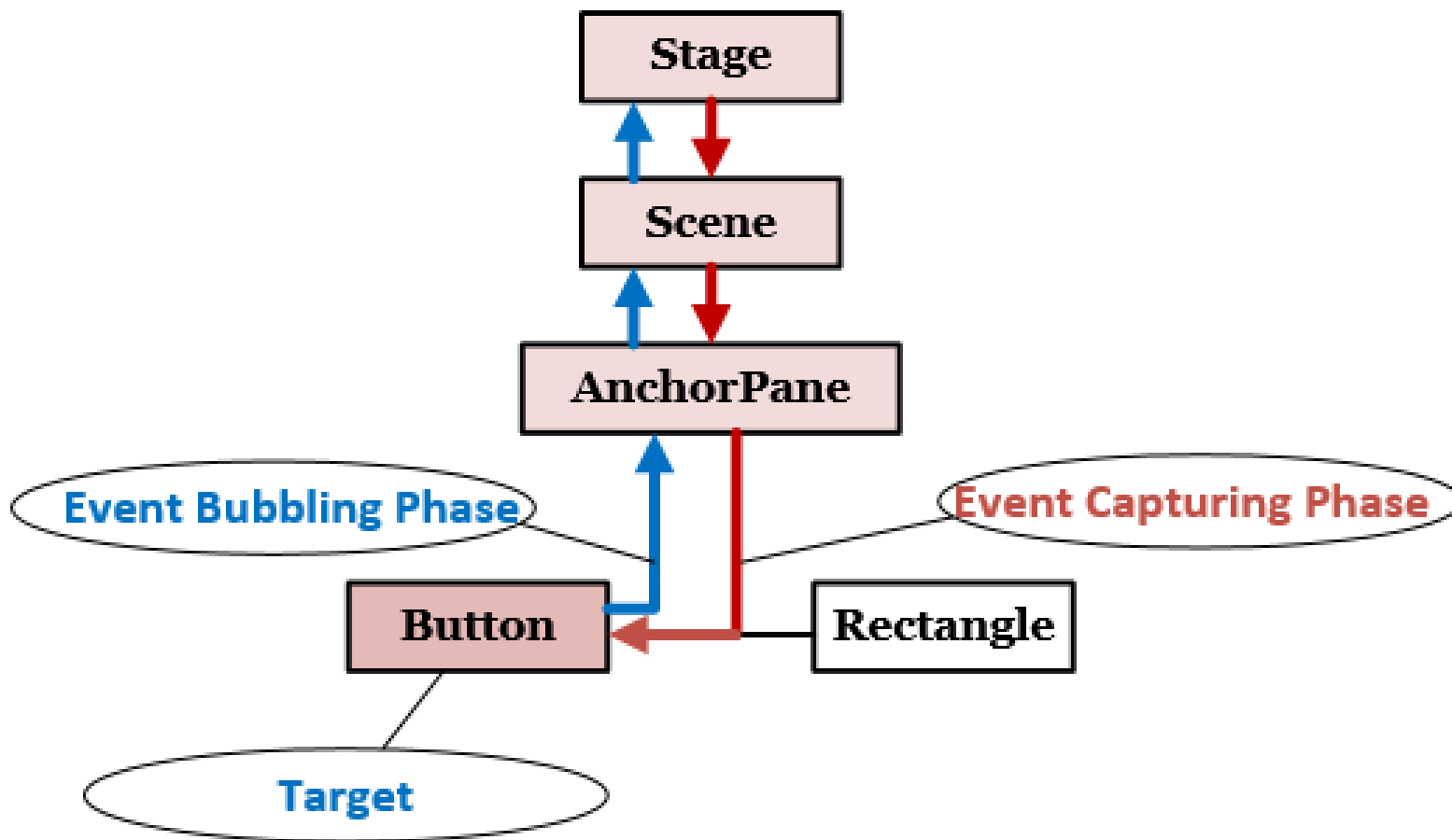
Обработката на събитието Key се извършва в същата последователност, както и на всички други събития.

1. При **натискане на клавиш** се определя **Възела-Цел на събитието (Target)**, това е **Възелът**, който е на фокус в този момент.
2. **Определя се пътя** от Stage до Target (Event Capturing Phase). На тази фаза се изпълняват *филтри* евентуално дефинирани по този път.
3. **Възелът-Цел на събитието (Target)** създава обекта на събитието (KeyEvent) и извършва необходимата обработка на събитието като използва данните капсулирани в обекта на събитието KeyEvent

15a.10 Обработка на събитието Key

4. Обработката на събитието (Key) продължава по обратния път на веригата от Възли, създадена на **Стъпка 2**. В случай, че Възел по този път от Target към Stage има регистриран метод за обработка на този тип събитие, то този метод обработва събитието. Когато този метод приключи изпълнение, събитието продължава пътя си нагоре по тази верига до следващия Възел. Ако няма друг метод за обработка на това събитие, регистриран в следващите Възли, то Възела в корена на Дървото от възли на Сцената получава това събитие и обработката на събитието приключва.

15a.9 Верижно разпределение на обработката на събитията



15a.10 Обработка на събитието Key

Възможността обработката на едно събитие да **„изплува“** и да се извърши от Възел, стоящ по-нависоко в дървовидната структура на Сцената, позволява по-ефективна обработка на събитията. Например, Контейнерът на подреждането може да обработва събитията, възникнали в съдържащите се в него Възли, вместо тази обработка да се извършва поотделно във всеки такъв Възел.

Всяко събитие може да се **„консумира“** от метод за обработка на събитието при предаването му по веригата от Възли от Target към Stage като се извика метода **consume ()** на обекта на събитието (KeyEvent). Това спира по-нататъшната обработка на събитието.

15a.11 Методи за обработка на KeyEvent

Всяко натискане на клавиш поражда събитието **Key**. Различаваме следните типове на това събитие:

KeyEvent.KEY_PRESSED- събитие от този тип настъпва при натискане на произволен клавиш

KeyEvent.KEY_RELEASED събитие от този тип настъпва при отпускане на произволен клавиш

KeyEvent.KEY_TYPED-събитие от този тип позволява да се прочете печатаем символ от клавиатурата. Настъпва след **KEY_PRESSED** и преди **KEY_RELEASED**.

15a.11 Методи за обработка на KeyEvent

Забележка

Възлите трябва да са с активирано (**true**) свойство **isFocusTraversable()**, за да обработват тези типове събития

15a.11 Методи за обработка на KeyEvent

Методите за обработка на тези типове събития имат единствен параметър от тип `KeyEvent` (обекта на събитието `Key`)

Обектът на събитието има следните важни свойства:

- `character` - прочита се с метода `getCharacter()`, който връща `String` на Unicode символ
- `text` - прочита се с метода `getText()`, връща `String` с наименование на управляващи клавиши или печатаеми символи "HOME", "F1" или "A"
- `code` - прочита се с метода `getCode()`, който връща `KeyCode`, където за прочитане на текста на `KeyCode` се използва метода `getName()`

15a.11 Методи за обработка на KeyEvent

Обработката на различните типове събития се прилага в зависимост от конкретните изисквания:

Събитие от тип `KeyEvent.KEY_TYPED` се обработва, когато има за цел да се прочете символ на клавиш, свързан с печатаем символ. В този случай се използва метода `getCharacter()` на обекта на събитието `KeyEvent`. В `SceneBuilder` името на метода за обработка на това събитие се задава в Раздела `Code` като част от описание на свойствата на избран Възел. Това може да стане и директно като се използва свойството `setOnKeyTyped()` на избрания Възел

15a.11 Методи за обработка на KeyEvent

Събитията от тип `KeyEvent.KEY_PRESSED` и `KeyEvent.KEY_RELEASED` се обработват, когато има за цел да се прочете произволен символ на клавиш, предимно управляващи и функционални клавиши. В този случай се използват методите `getText()` и `getCode()` на обекта на събитието `KeyEvent`. В `SceneBuilder` името на метода за обработка на тези събития се задава в Раздела `Code` като част от описание на свойствата на избран Възел. Това може да стане и директно като се използва свойството `setOnKeyPressed()` или `setOnKeyReleased()` на избрания Възел

15a.11 Методи за обработка на KeyEvent

Забележка

Метод `getText()` връща празен `String` при събитие от тип `KeyEvent.KEY_TYPED`

Метод `getCode()` връща `KeyCode.UNDEFINED` при събитие от тип `KeyEvent.KEY_TYPED`

Следователно тези два метода е безпредметно да се използват със събитие от тип `KeyEvent.KEY_TYPED`

15a.11 Методи за обработка на KeyEvent

Забележка

Метод `getCode()` се използва за филтриране на най- често използваните функционални и навигационни клавиши. За удобство се използват следните именувани константи на `KeyCode`, а именно, `KeyCode.ENTER`, `KeyCode.ESCAPE`, `KeyCode.F1`, `KeyCode.SPACE`, клавишите за компютърни игри, стрелките за преместване в хоризонтално и вертикално положение и много други.

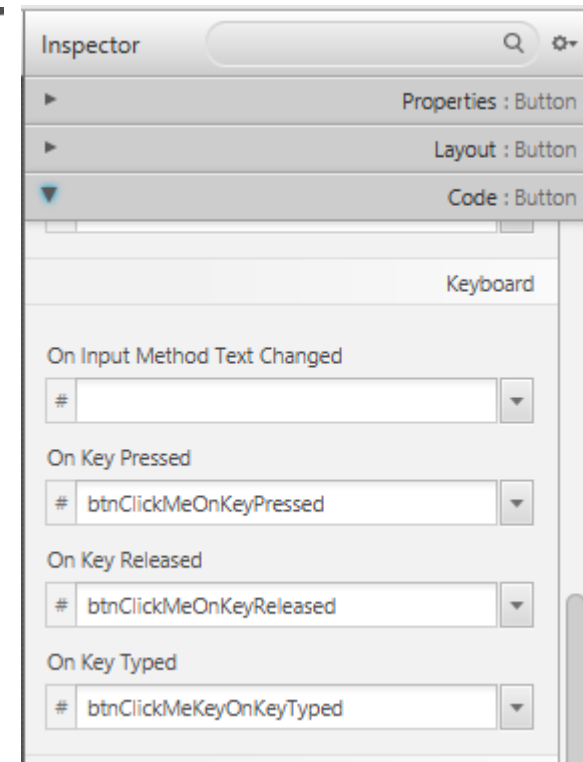
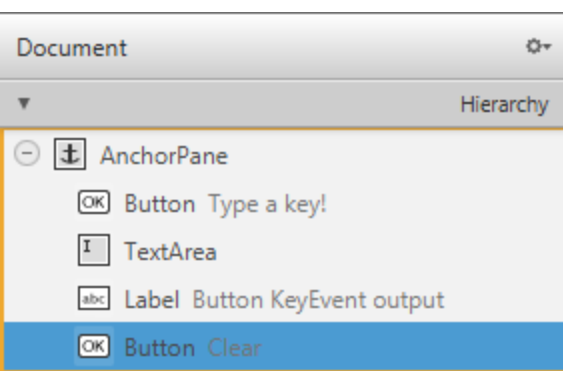
15a.11 Методи за обработка на KeyEvent

Редът на изпълнение на методите за обработка на трите основни типове събития на събитието `Key` е следният(в общия случай):

1. Методът за обработка на събитието `KeyEvent.KEY_PRESSED`
2. Методът за обработка на събитието `KeyEvent.KEY_TYPED`
3. Методът за обработка на събитието `KeyEvent.KEY_RELEASED`

Следното приложение (`JavaFXKeyEvenSimpleDemo`) демонстрира тези особености

15a.11 Методи за обработка на KeyEvent



15a.11 Методи за обработка на KeyEvent

@FXML

```
void btnClearClicked(ActionEvent event) {  
    txaKeyEventOutput.clear();  
    cntBtnKeyEvent = 1;  
}
```

@FXML

```
void btnClickMeOnKeyPressed(KeyEvent event) {  
    String outputText = String.format("%n%d. Key pressed- %s",  
        cntBtnKeyEvent++, event.getCode().getName());  
    txaKeyEventOutput.appendText(outputText);  
    txaKeyEventOutput.appendText(keyEventPressReleaseDetails(event));  
}
```

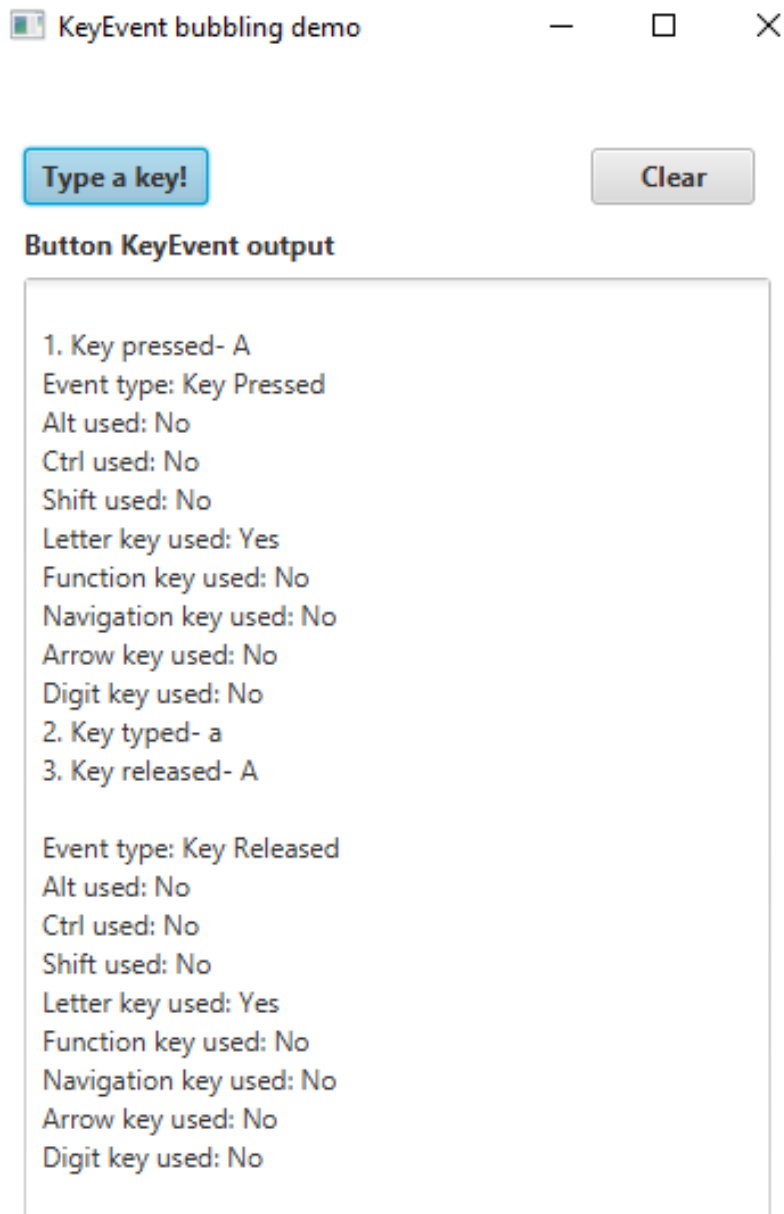
@FXML

```
void btnClickMeOnKeyReleased(KeyEvent event) {  
    String outputText = String.format("%n%d. Key released- %s%n",  
        cntBtnKeyEvent++, event.getCode().getName());  
    txaKeyEventOutput.appendText(outputText);  
    txaKeyEventOutput.appendText(keyEventPressReleaseDetails(event));  
}
```

15a.11 Методи за обработка на KeyEvent

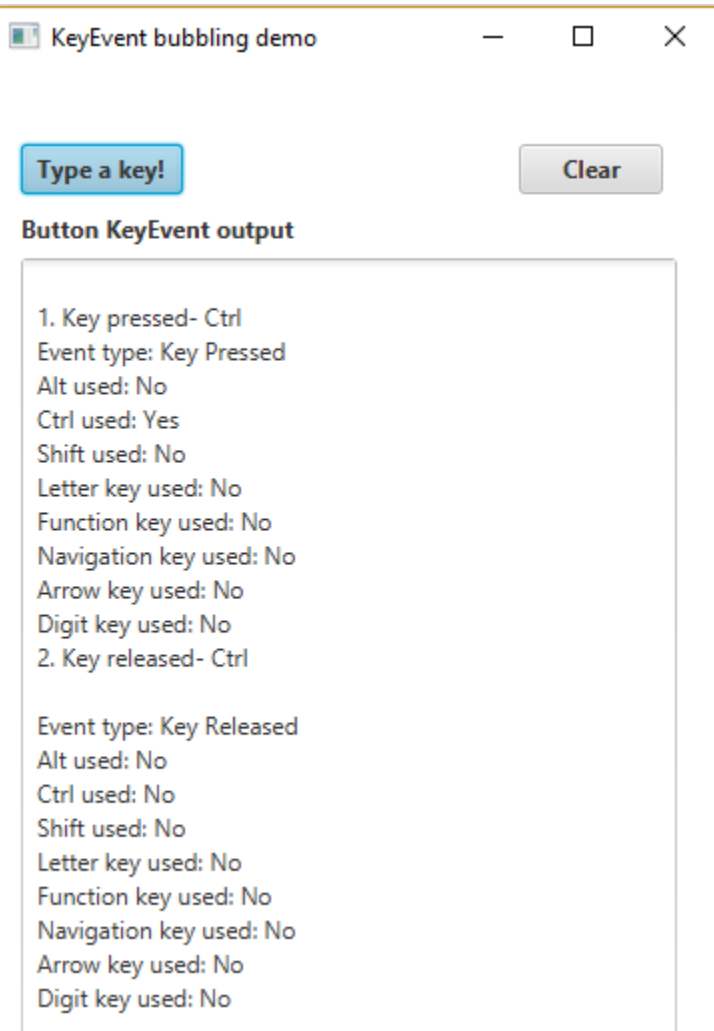
```
private String keyEventPressReleaseDetails(KeyEvent event) {  
    String details = "\nEvent type: ";  
    details += event.getEventType() == KeyEvent.KEY_PRESSED ? "Key Pressed"  
        : event.getEventType() == KeyEvent.KEY_RELEASED ? "Key Released"  
        : event.getEventType() == KeyEvent.KEY_TYPED ? "Key Typed"  
        : "Undefined";  
  
    details += String.format("\nAlt used: %s", (event.isAltDown() ? "Yes" : "No"));  
    details += String.format("\nCtrl used: %s", (event.isControlDown() ? "Yes" : "No"));  
    details += String.format("\nShift used: %s", (event.isShiftDown() ? "Yes" : "No"));  
    details += String.format("\nLetter key used: %s",  
        (event.getCode().isLetterKey() ? "Yes" : "No"));  
    details += String.format("\nFunction key used: %s",  
        (event.getCode().isFunctionKey() ? "Yes" : "No"));  
    details += String.format("\nNavigation key used: %s",  
        (event.getCode().isNavigationKey() ? "Yes" : "No"));  
    details += String.format("\nArrow key used: %s",  
        (event.getCode().isArrowKey() ? "Yes" : "No"));  
    details += String.format("\nDigit key used: %s",  
        (event.getCode().isDigitKey() ? "Yes" : "No"));  
    return details;  
}
```


15a.11 Методи за обработка на KeyEvent



При натискане на печатаем, ESC, BACK_SPACE, ENTER, RETURN, PLUS, MINUS клавиш събитията се изпълняват в указания ред

15a.11 Методи за обработка на KeyEvent



При натискане на `Ctrl`, `Shift`, `Alt`, `F1`, ..., `F12`, `CAPSLOCK`, `INS`, `DEL`, `HOME`, `END`, `PgDn`, `PgUp`, `NUM_LOCK` клавиш, както и при стрелките от цифровата клавиатура, събитието `KeyEvent.KEY_TYPED` не се изпълнява

15a.11 Методи за обработка на KeyEvent

KeyEvent bubbling demo

Type a key!

Clear

Button KeyEvent output

```
1. Key pressed- Tab
Event type: Key Pressed
Alt used: No
Ctrl used: No
Shift used: No
Letter key used: No
Function key used: No
Navigation key used: No
Arrow key used: No
Digit key used: No
```

При натискане на навигационен клавиш TAB или стрелките за навигация(извън цифровата клавиатура), се изпълнява единствено събитието `KeyEvent.KEY_PRESSED`

15a.11 Методи за обработка на KeyEvent

Следващият пример (`JavaFXKeyEventDemo`) показва как се реализира „изплуването“ (`bubbling`) при обработка на събития. За целта в `AnchorPane` са поставени два `FocusTraversable` Възела- `Button` и `Rectangle`. Последният не е `FocusTraversable` по подразбиране и е нужно това негово свойство да се активира при създаването му със `SceneBuilder`.

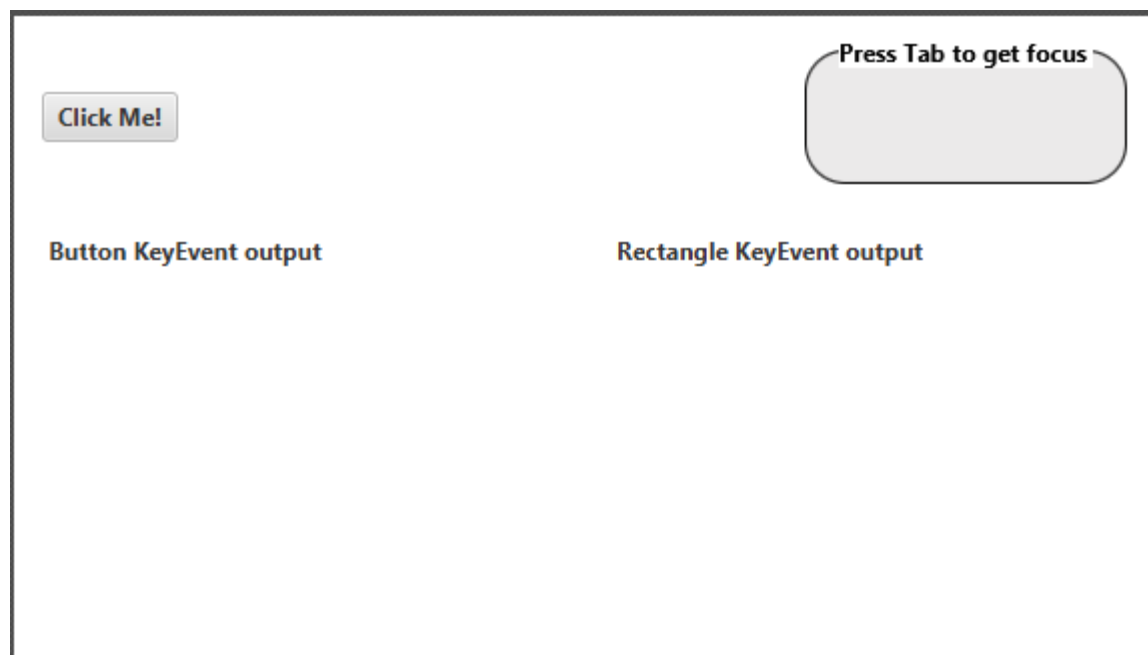
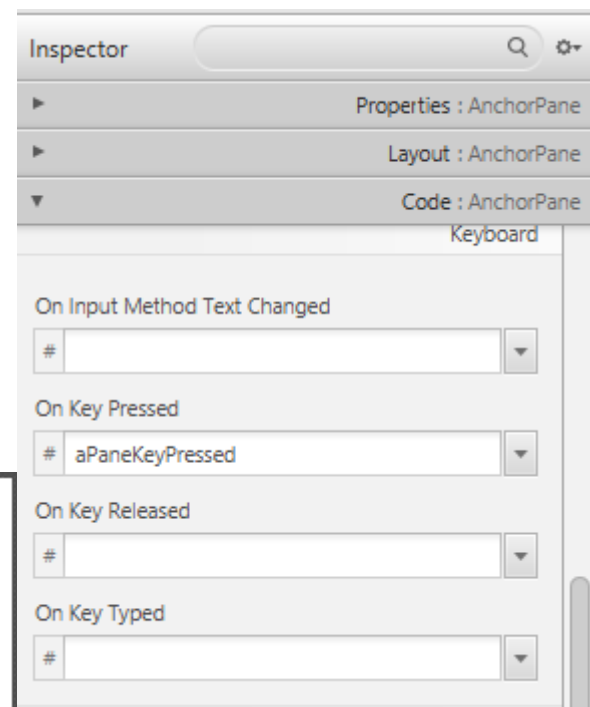
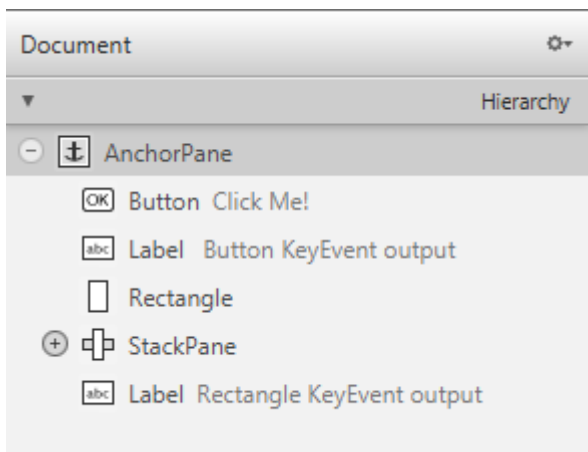
Във всяка от тези две компоненти `Button` и `Rectangle`, както и съдържащия ги Контейнер за подреждане `AnchorPane` е дефиниран метод за обработка на събитие от тип `KeyEvent.KEY_PRESSED`.

15a.11 Методи за обработка на KeyEvent

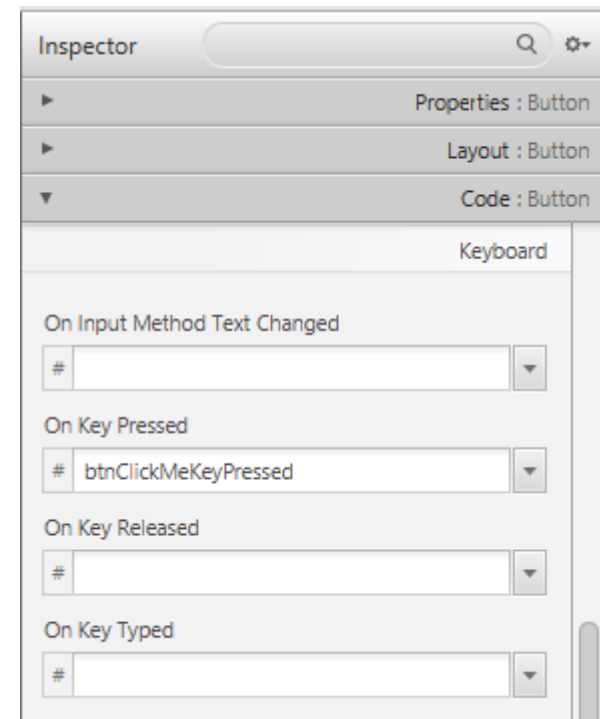
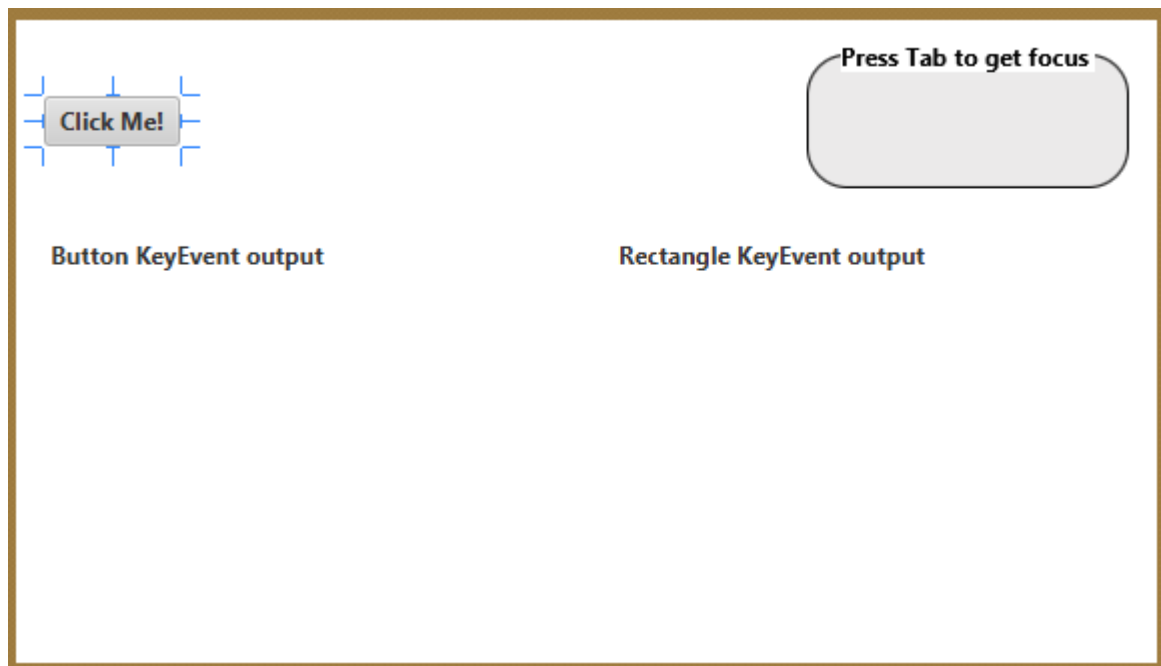
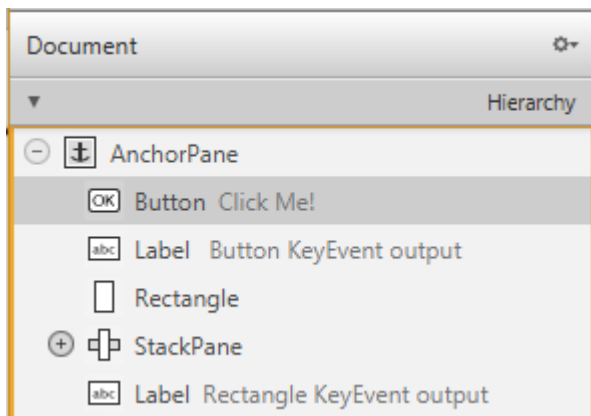
Изпълнението показва как обработката на това събитие започва с `Target`-а, който има фокус, и впоследствие „изплува“ в йерархията от обхващащи го Контейнери на подреждане и се обработката продължава с методите им за обработка на същия тип събитие

В случай, че `Button` има фокус при натискане на клавиш, то първо този `Control` обработва събитието от тип `KeyEvent.KEY_PRESSED`. След приключване на изпълнението на този метод, събитието „изплува“ нагоре в непосредствено съдържащия бутон Контейнер за подреждане `AnchorPane`. Понеже в този Контейнер има регистриран метод за обработка на същото това събитие, то този метод се изпълнява. След приключване на това изпълнение, събитието „изплува“ нагоре и стига до `Stage`, където завършва и цялата обработка на това събитие.

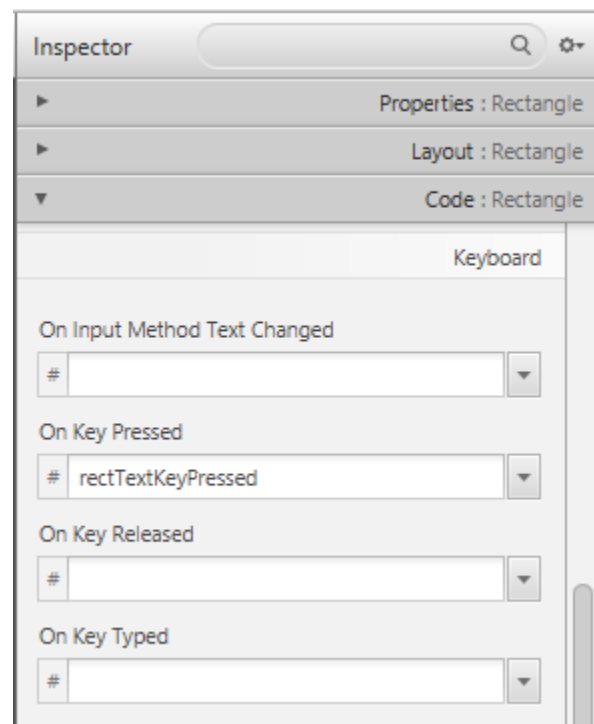
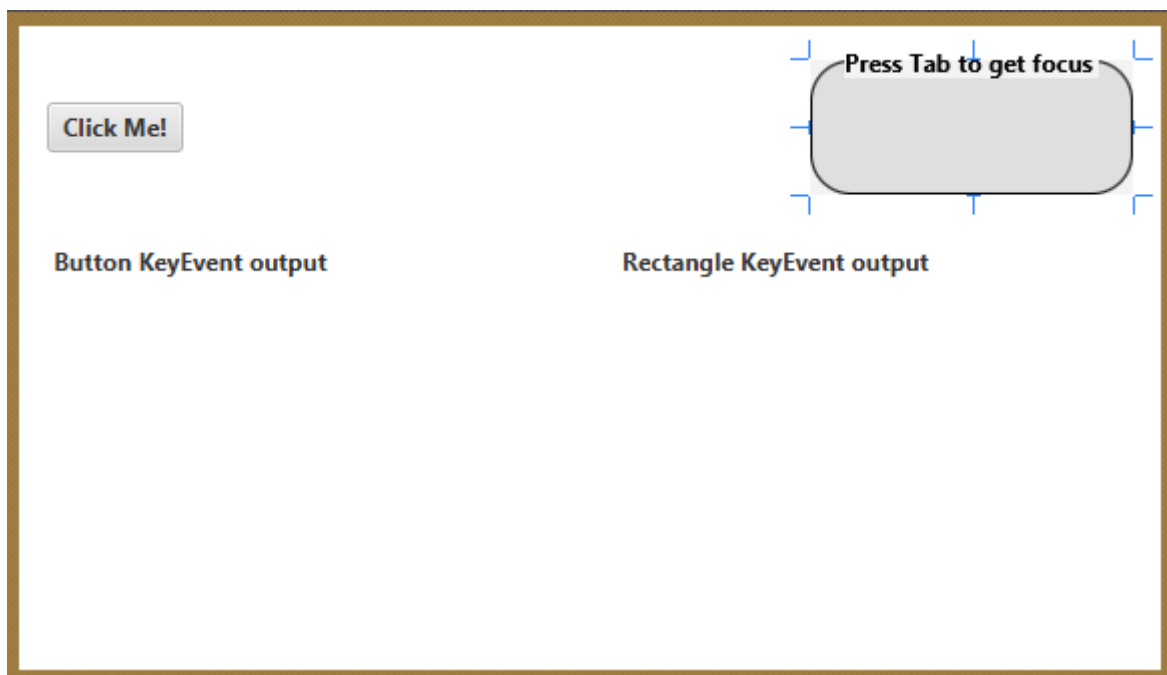
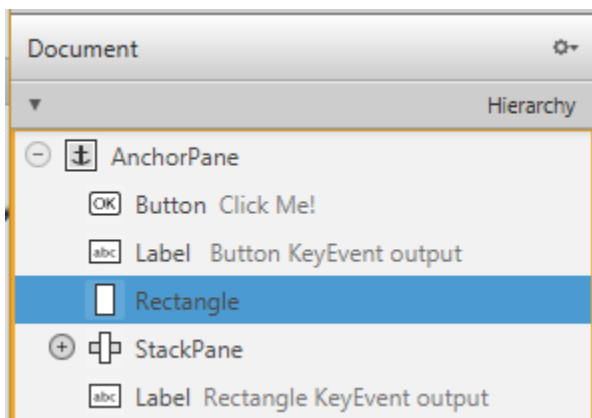
15a.11 Методи за обработка на KeyEvent



15a.11 Методи за обработка на KeyEvent



15a.11 Методи за обработка на KeyEvent



15a.11 Методи за обработка на KeyEvent

```
public class FXMLDocumentController {

    private int cntBtnKeyEvent, //sequence of KeyEvents on Button
              cntRectangleKeyEvent; //sequence of KeyEvents on Rectangle

    @FXML
    void initialize() {
        cntBtnKeyEvent = cntRectangleKeyEvent = 1;
        assert btnClickMe != null : "fx:id=\"btnClickMe\" was not injected: check your FXML file 'FXMLDocument.fxml'.";
        assert lblOutputText != null : "fx:id=\"lblOutputText\" was not injected: check your FXML file 'FXMLDocument.fxml'.";
        assert rectText != null : "fx:id=\"rectText\" was not injected: check your FXML file 'FXMLDocument.fxml'.";
    }

    @FXML
    private ResourceBundle resources;

    @FXML
    private URL location;

    @FXML
    private Button btnClickMe;

    @FXML //output KeyEvents on Button focus
    private Label lblOutputText;

    @FXML //output KeyEvents on Rectangle focus
    private Label lblOutputRectangle;

    @FXML // Rectangle with Focus Traversable activated
    private Rectangle rectText;
}
```

15a.11 Методи за обработка на KeyEvent

```
@FXML
void btnClickMeKeyPressed(KeyEvent event) {
    String outputText = String.format("%s%n%d. Key pressed- %s",
        lblOutputText.getText(), cntBtnKeyEvent++,
        event.getCode().getName());
    lblOutputText.setText(outputText);
}

@FXML
void rectTextKeyPressed(KeyEvent event) {
    String outputText = String.format("%s%n%d. Key pressed- %s",
        lblOutputRectangle.getText(), cntRectangleKeyEvent++,
        event.getCode().getName());
    lblOutputRectangle.setText(outputText);
}

@FXML
void aPaneKeyPressed(KeyEvent event) {
    String outputText;

    if (event.getTarget() == btnClickMe) {
        outputText = String.format("%s%n%d. AnchorPane: Key pressed- %s",
            lblOutputText.getText(), cntBtnKeyEvent++,
            event.getCode().getName());
        lblOutputText.setText(outputText);
    } else {
        outputText = String.format("%s%n%d. AnchorPane: Key pressed- %s",
            lblOutputRectangle.getText(), cntRectangleKeyEvent++,
            event.getCode().getName());
        lblOutputRectangle.setText(outputText);
    }
}
```

15a.11 Методи за обработка на KeyEvent

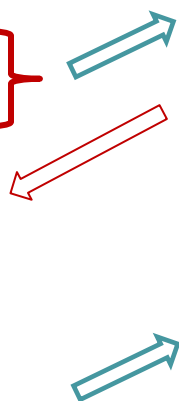


Button KeyEvent output

1. Key pressed- E
2. AnchorPane: Key pressed- E
3. Key pressed- Tab
4. AnchorPane: Key pressed- Tab
5. Key pressed- W
6. AnchorPane: Key pressed- W
7. Key pressed- S
8. AnchorPane: Key pressed- S
9. Key pressed- Tab
10. AnchorPane: Key pressed- Tab

Rectangle KeyEvent output

1. Key pressed- T
2. AnchorPane: Key pressed- T
3. Key pressed- Tab
4. AnchorPane: Key pressed- Tab
5. Key pressed- B
6. AnchorPane: Key pressed- B
7. Key pressed- B
8. AnchorPane: Key pressed- B



Задачи

1. Напишете *JavaFX* приложение, което изобразява *AnchorPane* в средата, на който е изписан символа 'А' (Нека при натиснат на бутон на мишката върху символа 'А' този символ да се мести заедно с курсора на мишката върху панела до отпускане на бутона на мишката).
2. Напишете *JavaFX* приложение , което е вариант на *задача 1*, при който символа се мести нагоре, надолу, наляво и надясно в панела при използване на съответните стрелки от клавиатурата. Като упътване използвайте, че методът *getCode()* на *KeyEvent* връща *KeyCode.KP_DOWN*, *KeyCode.KP_UP*, *KeyCode.KP_LEFT*, *KeyCode.KP_RIGHT* целочислени константи при натискане на стрелките нагоре, надолу, наляво и надясно(<https://docs.oracle.com/javafx/2/api/javafx/scene/input/KeyCode.html>) .