Faculty of Electronics and Information Technology
WARSAW UNIVERSITY OF TECHNOLOGY

# Programowanie Aplikacyjne (PAP)

## JavaFx

Julian Myrcha
Institute of Computer Science
26.02.2025

- Pobieramy plik openjfx-15.0.1_linux-x64_bin-sdk.zip ze strony `https://openjfx.io/`

- Pobieramy plik openjfx-15.0.1_linux-x64_bin-sdk.zip
- rozpakowujemy do katalogu: /opt/java/javafx-sdk-15.0.1

- Pobieramy plik openjfx-15.0.1_linux-x64_bin-sdk.zip
- rozpakowujemy do katalogu: /opt/java/javafx-sdk-15.0.1
- w pliku .bashrc ustawiamy export PATH_TO_FX

```
1 export PATH_TO_FX=/opt/java/javafx-sdk-15.0.1/lib
```

- Pobieramy plik openjfx-15.0.1_linux-x64_bin-sdk.zip
- rozpakowujemy do katalogu: /opt/java/javafx-sdk-15.0.1
- w pliku .bashrc ustawiamy export PATH_TO_FX=/opt/java/javafx-sdk-15.0.1/lib
- w pliku .vscode/settings.json dopisujemy klucz java.project.referencedLibraries

```
{
    "java.format.settings.url": "eclipse-formatter.xml",
    "java.project.referencedLibraries": [
        "/opt/java/javafx-sdk-15.0.1/lib/*.jar"
    ]
}
```

- Pobieramy plik openjfx-15.0.1_linux-x64_bin-sdk.zip
- rozpakowujemy do katalogu: /opt/java/javafx-sdk-15.0.1
- w pliku .bashrc ustawiamy export PATH_TO_FX
- w pliku .vscode/settings.json dopisujemy klucz
- w pliku launch.json dopisujemy klucz vmArgs

```
{
    "type": "java",
    "name": "CodeLens (Launch) - SampleFx",
    "request": "launch",
    "vmArgs": "--module-path /opt/java/javafx-sdk-15.0.1/lib
               --add-modules javafx.controls,javafx.fxml",
    "mainClass": "proz.SampleFx",
    "projectName": "SimpleSwing_8edcdf2a"
}
```

- Pobieramy plik openjfx-15.0.1_linux-x64_bin-sdk.zip
- rozpakowujemy do katalogu: /opt/java/javafx-sdk-15.0.1
- w pliku .bashrc ustawiamy export PATH_TO_FX
- w pliku .vscode/settings.json dopisujemy klucz
- w pliku launch.json dopisujemy klucz vmArgs
- ubuntu ma także pakiet openjfx ... (ja chciałem mieć najnowszą wersję)

- https://gluonhq.com/products/scene-builder/

- https://gluonhq.com/products/scene-builder/
- uruchomienie programu ręcznie

```
export JAVA_TOOL_OPTIONS=--enable-preview
/opt/java/jdk-15/bin/java --module-path /opt/java/javafx-sdk-15.0.1/lib \
             --add-modules javafx.controls,javafx.fxml   -jar SimpleSwing.jar
```

- https://gluonhq.com/products/scene-builder/
- uruchomienie programu ręcznie
- pomocniczy skrypt /bin/javafx

```
#!/bin/bash
/opt/java/jdk-15/bin/java --module-path /opt/java/javafx-sdk-15.0.1/lib \
                --add-modules javafx.controls,javafx.fxml --enable-preview  "$@"
```

- Wykorzystanie kart graficznych do wyświetlania treści

```
slider.valueProperty().addListener( (ov, oldVal, newVal) ->
    offsetText.setText("Wartosc: " + slider.getValue()));
```

- Wykorzystanie kart graficznych do wyświetlania treści
- Spójny mechanizm do wyświetlania UI oraz efektów 3D w ramach jednej technologii

```
slider.valueProperty().addListener( (ov, oldVal, newVal) ->
    offsetText.setText("Wartosc: " + slider.getValue()));
```

- Wykorzystanie kart graficznych do wyświetlania treści
- Spójny mechanizm do wyświetlania UI oraz efektów 3D w ramach jednej technologii
- Elementy graficzne są obiektami (a nie raz narysowaną bitmapą) - zmiana koloru linii spowoduje odrysowanie w innym kolorze

```
1 slider.valueProperty().addListener( (ov, oldVal, newVal) ->
2     offsetText.setText("Wartosc: " + slider.getValue()));
```

- Wykorzystanie kart graficznych do wyświetlania treści
- Spójny mechanizm do wyświetlania UI oraz efektów 3D w ramach jednej technologii
- Elementy graficzne są obiektami (a nie raz narysowaną bitmapą) - zmiana koloru linii spowoduje odrysowanie w innym kolorze
- Bogaty zestaw elementów (np. krzywe) oraz transformacje

```
1 slider.valueProperty().addListener( (ov, oldVal, newVal) ->
2     offsetText.setText("Wartosc: " + slider.getValue()));
```

- Wykorzystanie kart graficznych do wyświetlania treści
- Spójny mechanizm do wyświetlania UI oraz efektów 3D w ramach jednej technologii
- Elementy graficzne są obiektami (a nie raz narysowaną bitmapą) - zmiana koloru linii spowoduje odrysowanie w innym kolorze
- Bogaty zestaw elementów (np. krzywe) oraz transformacje
- Mechanizm ułatwiający zachowanie spójnego wyglądu - style (CSS)

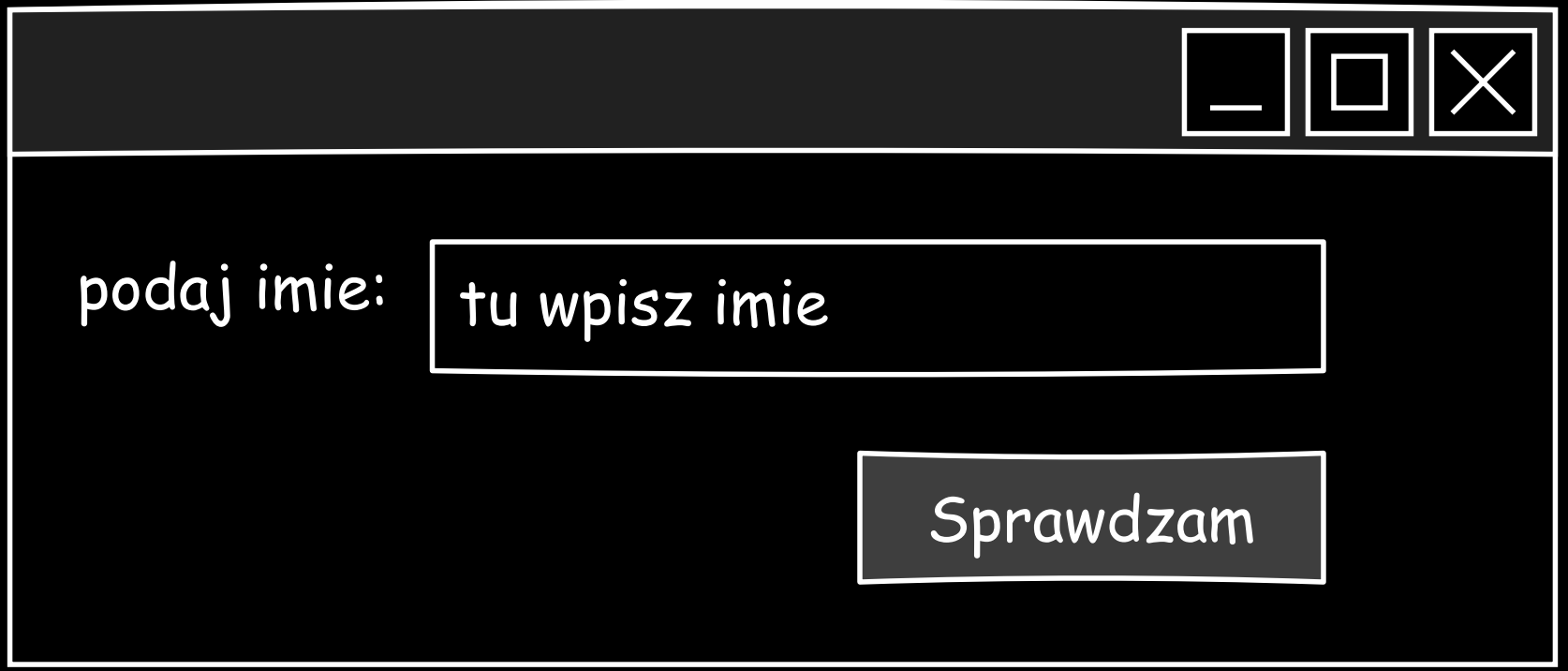
```
1 slider.valueProperty().addListener( (ov, oldVal, newVal) ->
2     offsetText.setText("Wartosc: " + slider.getValue()));
```

- Wykorzystanie kart graficznych do wyświetlania treści
- Spójny mechanizm do wyświetlania UI oraz efektów 3D w ramach jednej technologii
- Elementy graficzne są obiektami (a nie raz narysowaną bitmapą) - zmiana koloru linii spowoduje odrysowanie w innym kolorze
- Bogaty zestaw elementów (np. krzywe) oraz transformacje
- Mechanizm ułatwiający zachowanie spójnego wyglądu - style (CSS)
- Wiele urządzeń - ta sama technologia na desktopie i w aplikacjach mobilnych

```
1 slider.valueProperty().addListener( (ov, oldVal, newVal) ->
2     offsetText.setText("Wartosc: " + slider.getValue()));
```

- Wykorzystanie kart graficznych do wyświetlania treści
- Spójny mechanizm do wyświetlania UI oraz efektów 3D w ramach jednej technologii
- Elementy graficzne są obiektami (a nie raz narysowaną bitmapą) - zmiana koloru linii spowoduje odrysowanie w innym kolorze
- Bogaty zestaw elementów (np. krzywe) oraz transformacje
- Mechanizm ułatwiający zachowanie spójnego wyglądu - style (CSS)
- Wiele urządzeń - ta sama technologia na desktopie i w aplikacjach mobilnych
- Wykorzystanie mechanizmu notyfikacji o zmianach:

```
1 slider.valueProperty().addListener( (ov, oldVal, newVal) ->
2     offsetText.setText("Wartosc: " + slider.getValue()));
```



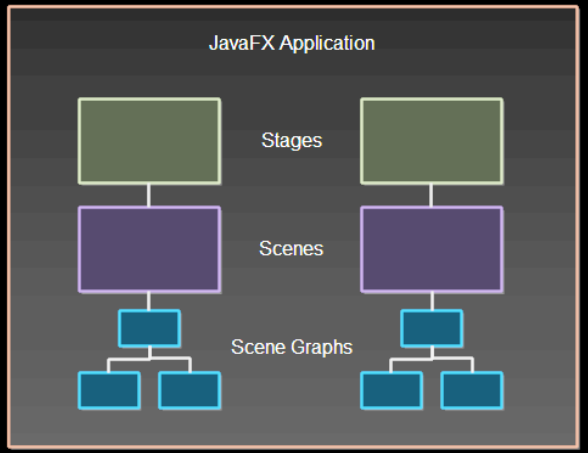

- Okno - stage - nie jest dziedziczone - kontrolki dodajemy do niego
- Kontrolki do których chcemy się odwoływać polami klasy dziedziczącej z Application
- Procedury obsługi zdarzenia metodami tej klasy

- Za pomocą lambda expression dowiązujemy procedurę obsługi przerwania

```
public class Simple extends Application {
    @Override
    public void start(Stage primaryStage) { // okno dostalismy jako parametr!
        btn = new Button("Sprawdzam");
        btn.setMaxWidth(100);
        btn.setOnAction(e->Click());
        text = new TextField("tu wpisz imie");
        Label lbl = new Label("podaj imie:");
        HBox hbox = new HBox(10,lbl,text);
        VBox vbox = new VBox(10,hbox,btn);
        Scene scene = new Scene(vbox,400,400);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Bardzo prosta aplikacja");
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
    public void Click() {
        MessageBox.show(text.getText(), "Click!");
    }
    TextField text;
    Button btn;
}
```

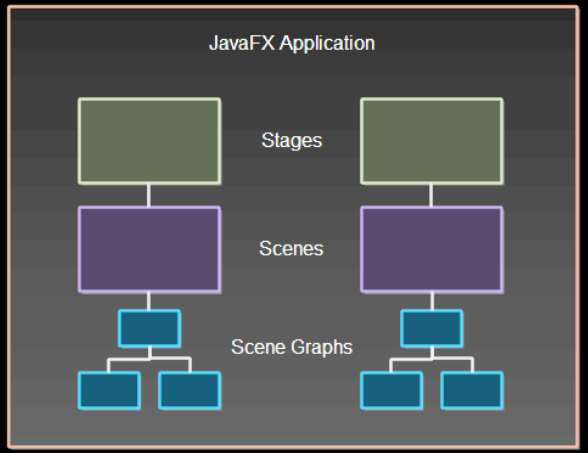
| Właściwość | JavaFx | Swing |
|---|---|---|
| Powiązania właściwości | właściwości obsługują wiązanie, co oznacza, że można nasłuchiwać zmian ich wartości | właściwości nie obsługują bezpośrednio wiązania |
| Deklaratywny układ | JavaFX obsługuje deklaratywne układy za pośrednictwem FXML. | Swing nie ma wbudowanej obsługi deklaratywnego układu. |
| Style | JavaFX obsługuje style oparte na CSS i kodzie | Swing obsługuje tylko stylizację opartą na kodzie. |
| WebView | JavaFX ma WebView, który może renderować nowoczesne strony internetowe | Swing nie ma WebView. |
| Graphics | JavaFX wykorzystuje grafikę wektorową | Swing wykorzystuje grafikę opartą na pikselach. |
| 3D Graphics | JavaFX ma wbudowaną obsługę grafiki 3D | Swing wymaga Java 3D API do grafiki 3D. |
| Concurrency API | JavaFX ma wbudowany interfejs API współbieżności | Swing nie ma wbudowanego interfejsu API współbieżności. |
| Wiek biblioteki | JavaFX jest nowsza | Swing jest starszy. |
| Zawarte w Java SDK | avaFX nie jest uwzględniony w wersji Java 11 i nowszych | Swing jest nadal uwzględniony, ale prawdopodobnie zostanie usunięty pewnego dnia. |

**Stage** -Teatr jest zewnętrzną ramką dla aplikacji JavaFX. Teatr(Stage) zazwyczaj odpowiada oknu.



**Stage** -Teatr jest zewnętrzną ramką dla aplikacji JavaFX. Teatr(Stage) zazwyczaj odpowiada oknu.

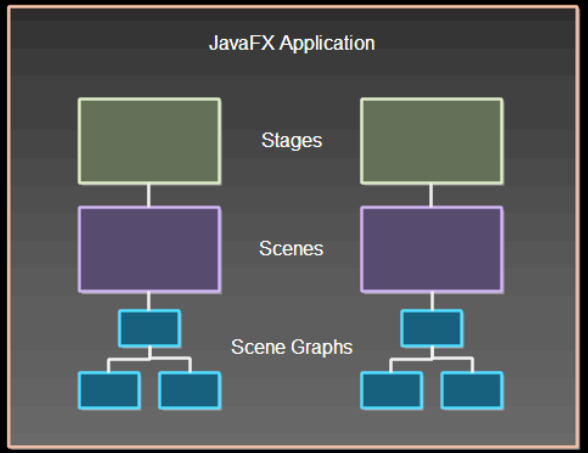
**Scene** -Aby wyświetlić cokolwiek w Stage(Teatrze) aplikacji JavaFX potrzebujemy sceny. Teatr może pokazywać tylko jedną scenę naraz, ale istnieje możliwość wymiany sceny w czasie wykonywania



**Stage** -Teatr jest zewnętrzną ramką dla aplikacji JavaFX. Teatr(Stage) zazwyczaj odpowiada oknu.

**Scene** -Aby wyświetlić cokolwiek w Stage(Teatrze) aplikacji JavaFX potrzebujemy sceny. Teatr może pokazywać tylko jedną scenę naraz, ale istnieje możliwość wymiany sceny w czasie wykonywania

**Scene Graph** -Drzewo (graf) wszystkich kontrolek i układów dołączonych do sceny

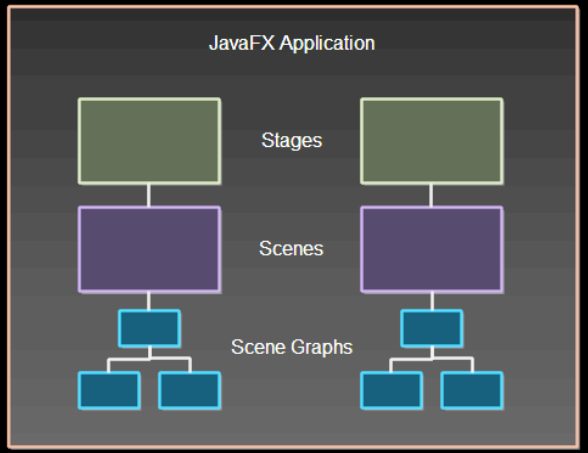


**Stage** -Teatr jest zewnętrzną ramką dla aplikacji JavaFX. Teatr(Stage) zazwyczaj odpowiada oknu.

**Scene** -Aby wyświetlić cokolwiek w Stage(Teatrze) aplikacji JavaFX potrzebujemy sceny. Teatr może pokazywać tylko jedną scenę naraz, ale istnieje możliwość wymiany sceny w czasie wykonywania

**Scene Graph** -Drzewo (graf) wszystkich kontrolek i układów dołączonych do sceny

**Controls** -Komponenty JavaFX takie jak przycisk, przycisk opcji, tabela, drzewo



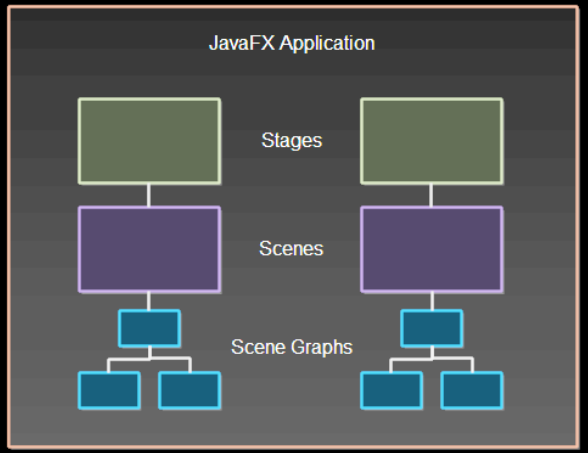
**Stage** -Teatr jest zewnętrzną ramką dla aplikacji JavaFX. Teatr(Stage) zazwyczaj odpowiada oknu.

**Scene** -Aby wyświetlić cokolwiek w Stage(Teatrze) aplikacji JavaFX potrzebujemy sceny. Teatr może pokazywać tylko jedną scenę naraz, ale istnieje możliwość wymiany sceny w czasie wykonywania

**Scene Graph** -Drzewo (graf) wszystkich kontrolek i układów dołączonych do sceny

**Controls** -Komponenty JavaFX takie jak przycisk, przycisk opcji, tabela, drzewo

**Layouts** -układy to komponenty, które zawierają w sobie inne komponenty



- Tworzy obiekt Stage

```
package pap;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class App extends Application {
  @Override
  public void init() {}
  @Override
  public void start(Stage stage) {
    var javaVer = SystemInfo.javaVersion();
    var javafxVer = SystemInfo.javafxVersion();
    var label=new Label("Hello, JavaFX " + javafxVer
          + ", running on Java " + javaVer + ".");
    var scn=new Scene(new StackPane(label),640,480);
    stage.setScene(scn);
    stage.show();
  }
  @Override
  public void close() {}
  public static void main(String[] args) {
    launch();
  }
}
```

- Tworzy obiekt Stage
- Woła jego metodę `init`

```
1  package pap;
2  import javafx.application.Application;
3  import javafx.scene.Scene;
4  import javafx.scene.control.Label;
5  import javafx.scene.layout.StackPane;
6  import javafx.stage.Stage;
7
8  public class App extends Application {
9    @Override
10   public void init() {}
11   @Override
12   public void start(Stage stage) {
13     var javaVer = SystemInfo.javaVersion();
14     var javafxVer = SystemInfo.javafxVersion();
15     var label=new Label("Hello, JavaFX " + javafxVer
16           + ", running on Java " + javaVer + ".");
17     var scn=new Scene(new StackPane(label),640,480);
18     stage.setScene(scn);
19     stage.show();
20   }
21   @Override
22   public void close() {}
23   public static void main(String[] args) {
24     launch();
25   }
26 }
```

- Tworzy obiekt Stage
- Woła jego metodę `init`
- Woła jego metodę `start` i czeka na informację o zamknięciu

```java
package pap;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class App extends Application {
  @Override
  public void init() {}
  @Override
  public void start(Stage stage) {
    var javaVer = SystemInfo.javaVersion();
    var javafxVer = SystemInfo.javafxVersion();
    var label=new Label("Hello, JavaFX " + javafxVer
          + ", running on Java " + javaVer + ".");
    var scn=new Scene(new StackPane(label),640,480);
    stage.setScene(scn);
    stage.show();
  }
  @Override
  public void close() {}
  public static void main(String[] args) {
    launch();
  }
}
```

- Tworzy obiekt Stage
- Woła jego metodę `init`
- Woła jego metodę `start` i czeka na informację o zamknięciu
- Woła metodę `close`

```
1  package pap;
2  import javafx.application.Application;
3  import javafx.scene.Scene;
4  import javafx.scene.control.Label;
5  import javafx.scene.layout.StackPane;
6  import javafx.stage.Stage;
7
8  public class App extends Application {
9    @Override
10   public void init() {}
11   @Override
12   public void start(Stage stage) {
13     var javaVer = SystemInfo.javaVersion();
14     var javafxVer = SystemInfo.javafxVersion();
15     var label=new Label("Hello, JavaFX " + javafxVer
16             + ", running on Java " + javaVer + ".");
17     var scn=new Scene(new StackPane(label),640,480);
18     stage.setScene(scn);
19     stage.show();
20   }
21   @Override
22   public void close() {}
23   public static void main(String[] args) {
24     launch();
25   }
26 }
```

- Tworzy obiekt Stage
- Woła jego metodę `init`
- Woła jego metodę `start` i czeka na informację o zamknięciu
- Woła metodę `close`
- Drzewo budujemy operując na obiektach typu `ObservableList`

```
1  package pap;
2  import javafx.application.Application;
3  import javafx.scene.Scene;
4  import javafx.scene.control.Label;
5  import javafx.scene.layout.StackPane;
6  import javafx.stage.Stage;
7
8  public class App extends Application {
9    @Override
10   public void init() {}
11   @Override
12   public void start(Stage stage) {
13     var javaVer = SystemInfo.javaVersion();
14     var javafxVer = SystemInfo.javafxVersion();
15     var label=new Label("Hello, JavaFX " + javafxVer
16             + ", running on Java " + javaVer + ".");
17     var scn=new Scene(new StackPane(label),640,480);
18     stage.setScene(scn);
19     stage.show();
20   }
21   @Override
22   public void close() {}
23   public static void main(String[] args) {
24     launch();
25   }
26 }
```

```java
package inu;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
public class FxScene extends Application {
    @Override
    public void start(Stage primaryStage) {
        try {
            HBox hbox = new HBox(5);
            Rectangle r1 = new Rectangle(100, 100);
            hbox.getChildren().addAll(r1);
            Scene scene = new Scene(hbox,400,400);
            primaryStage.setScene(scene);
            primaryStage.setTitle("INU 1");
            primaryStage.show();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

**javafx.application** - `Application` zarządza cyklem życia, czyli w niej implementujemy metody odpowiedzialn za tworzenie i wychodzenie

**javafx.stage** - `Stage` - okno(teatr), albo `FileChooser` lub `DirectoryChooser`

**javafx.scene** - `Scene` - scena, `Node` - bazowa klasa dla wszystkich obiektów, które można umieszczać na scenie

**javafx.scene.control** - tutaj są wszystkie kontrolki `Button`, `Label`, `CheckBox`

**javafx.scene.layout** - `HBox`, `VBox`, `BorderPane`, `Pane`, `Region`

**javafx.scene.shape** - `Insets`

**javafx.collections** - `ObservableList` zwaracana przez `getChildren`

# Zamiast importować pojedyńcze klasy można zaimportować całość i mieć z głowy ....

```
1 import javafx.application.*;
2 import javafx.stage.*;
3 import javafx.scene.*;
4 import javafx.scene.control.*;
5 import javafx.scene.layout.*;
6 import javafx.scene.shape.*;
7 import javafx.collections.*;
```

- Korzeń drzewa węzłów - `Stage`(Teatr)
- Odpowiada oknu (linux, windows) lub całemu ekranowi (mobile) lub kafelkowi
- Do korzenia podłączamy zawartość - `Scene`(Przedstawienie)

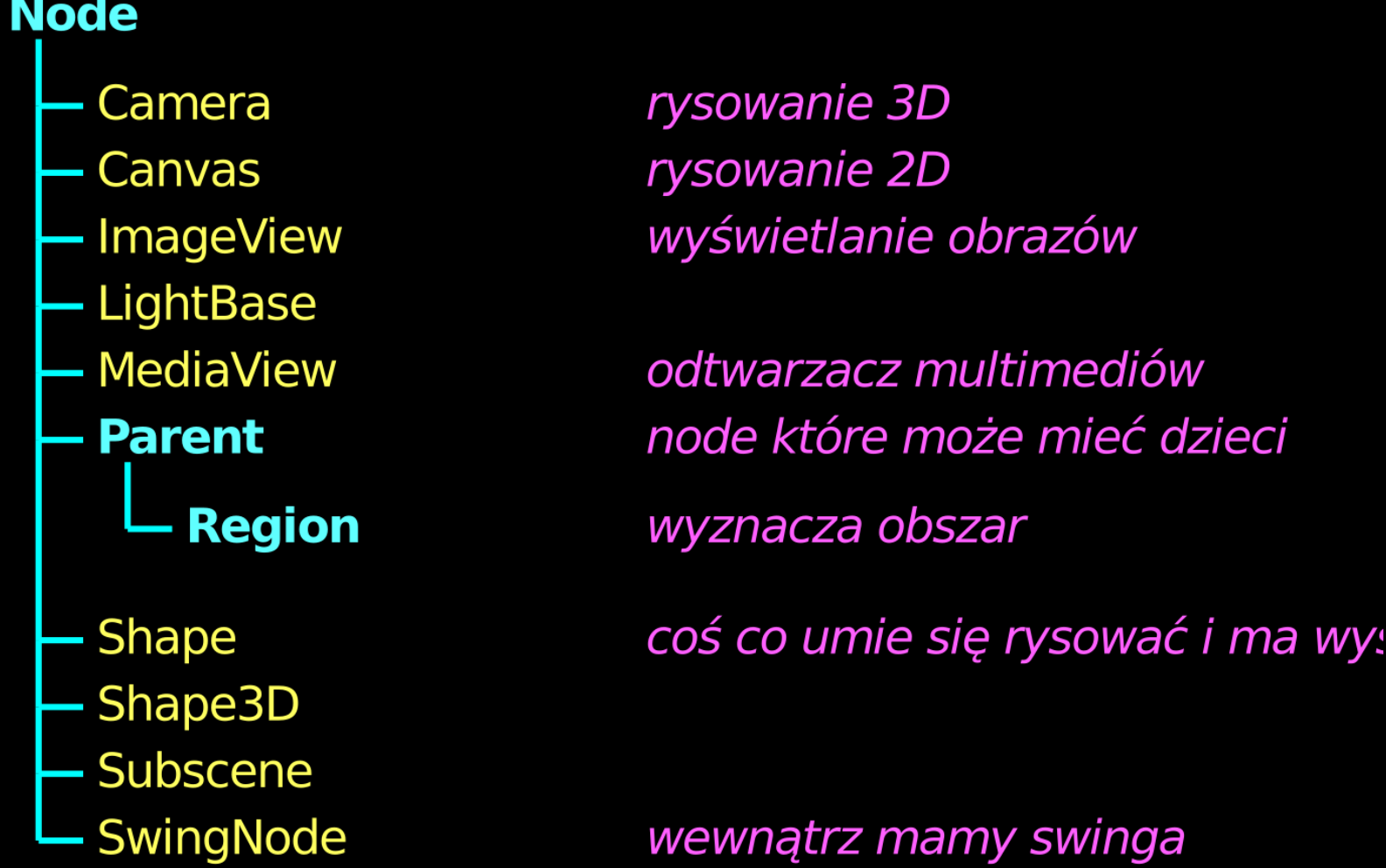
```
void close();
void getMaxHeight(double maxheight);
void setFullScreen(boolean fullscreen);
void setIconified(boolean iconified);
void setMaximized(boolean maximized);
void setMaxHeight(double maxheight);
void setResizable(boolean resizable);
void setScene(Scene scene);
void setTitle(String title);
void show();
void showAndWait();
void toFront()
void toBack()
```

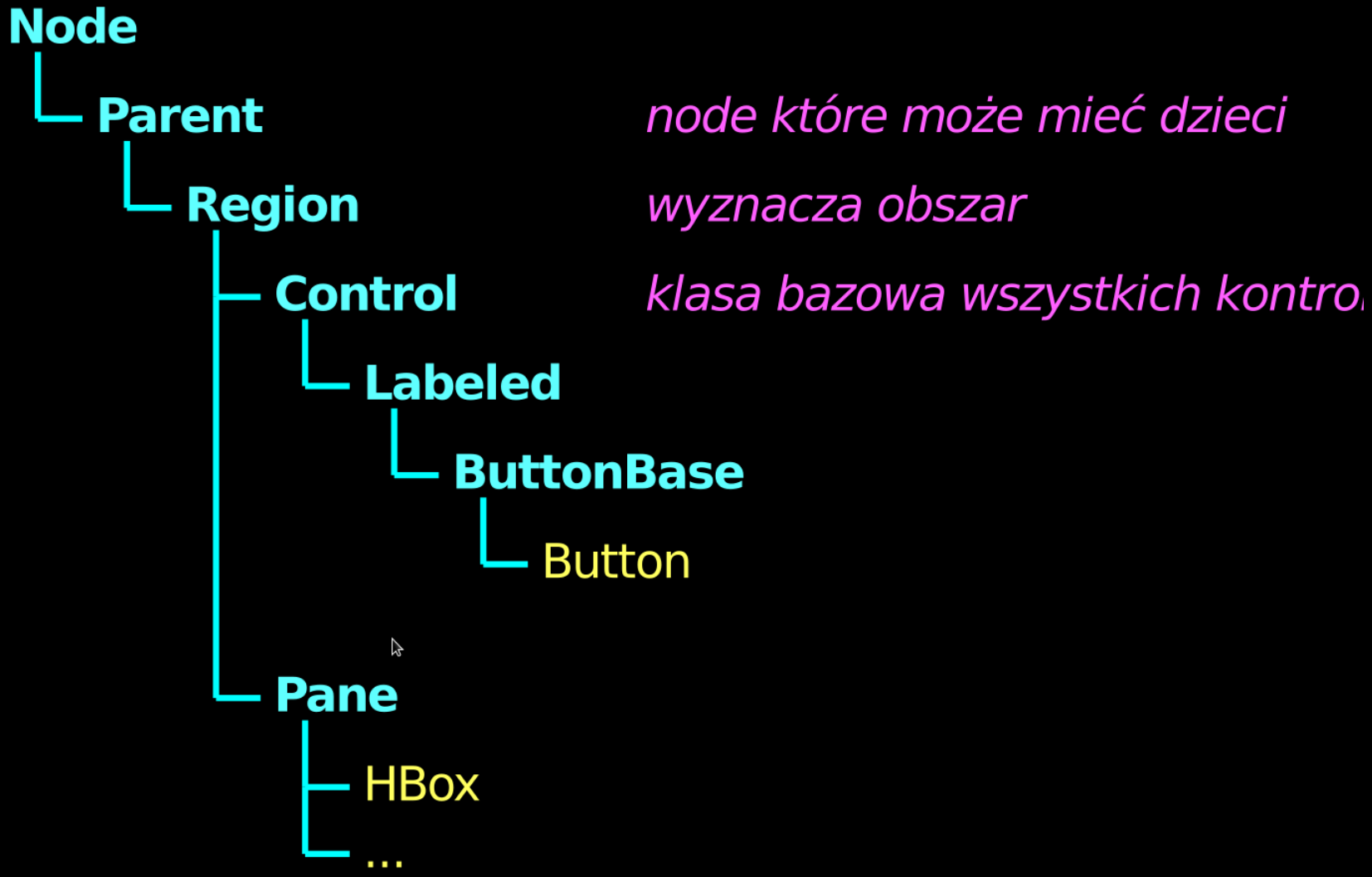
- klasa abstrakcyjna
- metoda `getChildren()` layoutów zwaraca kolekcję obiektów typu `Node`

| Metoda | Opis |
|---|---|
| Parent getParent() | Zwraca rodzica. |
| String getId() | Zwraca ID aktualnego node. |
| void setId(String id) | Ustawia ID. Powinno być unikalne na scenie. |
| Node lookup(String id) | Szuka wsród dzieci wystąpienia ID |
| String getStyle() | zwraca tekst CSS |
| void setStyle(String style) | ustawia CSS |

Przykład

```
1 button.setId("OKBTN");
2 Node node = scene.getRoot().lookup("#OKBTN")
```


Node
Camera
rysowanie 3D
Canvas
rysowanie 2D
ImageView
wyświetlanie obrazów
LightBase
MediaView
odtwarzacz multimediów
Parent
node które może mieć dzieci
Region
wyznacza obszar
Shape
coś co umie się rysować i ma wy
Shape3D
Subscene
SwingNode
wewnątrz mamy swinga


Node
Parent
node które może mieć dzieci
Region
wyznacza obszar
Control
klasa bazowa wszystkich kontro
Labeled
ButtonBase
Button
Pane
HBox
...

- to taki node, który może mieć dzieci

```
HBox hbox = new HBox(30);
Label lblAddress = new Label("Adres zamieszkania:");
TextField txtAddress = new TextField();
hbox.getChildren().addAll(lblAddress, txtAddress);
Scene scene = new Scene(hbox,400,400);
```

- najważniejszą metodą jest `getChildren()` zwracający obiekt klasy `ObservableList`

| Metoda | Opis |
|---|---|
| void add(T element) | dodaje nowy node |
| void add(int index, T element) | dodaje nowy node w podanym miejscu, przesuń pozostałe dalej |
| T set(int index, T element) | dodaje nowy node w podanym miejscu, zwraca poprzedni element, który usuwa z listy |
| T get(int index) | zwraca element z pozycji |
| void addAll(Node nodes...) | dodaje listę nodów |
| void remove(Node node) | kasuje wskazany node |
| void clear() | usuwa wszystkie dzieci |

| Metoda | Opis |
|---|---|
| int size() | liczy dzieci |
| boolean void contains(Object elem) | czy lista zawiera? |
| int indexOf(Object elem) | pozycja gdy jest, -1 gdy nie ma |
| void addListener(ListChange Listener listener) | zdarzenie, gdy zmiana listy |

- Wszystkie kontrolki dziedziczą po `Control`
- Ale większość funkcjonalności mają po jej klasach bazowych ...

| **Metoda** | **Opis** |
|---|---|
| void setTooltip(Tooltip value) | ustawia podpowiedź |
| void setContextmenu(Contextmenu value) | ustawia menu |
| void setSkin(Skin value) | ustawia skórkę |

- Jest to klasa generyczna, zatem musimy podać typ elementu
- Element musi mieć metodę toString()
- Listę można programowo rozwinąć (`show()`) lub zwinąć (`hide()`)

```
ChoiceBox<String> choice = new ChoiceBox<String>();
choice.getItems().addAll("Styczen", "Luty", "Marzec",
        "Kwiecien", "Maj", "Czerwiec", "Lipiec");
hbox.getChildren().add(choice);
```

- Możemy programowo ustawić wartość domyślną (`setValue("Czerwiec")`) podając obiekt, który ma być wybrany
- Możemy programowo pobrać wybraną wartość metodą `getValue()`

Aby być informowanym na bierząco o zmianach wybranego elementu:

- Musimy dostać obiekt klasy `SelectionModel` za pomocą metody `getSelectionModel()`
- Musimy dostać obiect klasy `ReadOnlyObjectProperty` za pomocą metody `selectedItemProperty`
- Musimy dodać słuchacza zmian:

```
1 choice.getSelectionModel()
2       .selectedItemProperty()
3       .addListener( (v, oldValue, newValue)-> lbl.setText(newValue));
```

**observable** - jakiej właściwości dotyczy zmiana

**oldValue** - stara wartość

**newValue** - nowa wartość

- Możliwość ograniczenia liczby widocznych elementów, gdy więcej mamy przewijanie
- Możliwość wprowadzenia tekstu z klawiatury (opcjonalna)
- Woła zdarzenie akcji po zmianie wyboru (łatwiej korzystać)
- Przy obsłudze z klawiatury nie wyświetla pełnej listy, tylko daje poprzedni/następny element (gdy nie jest edytowalny)

```
ComboBox<String> cbox = new ComboBox<String>();
cbox.getItems().addAll("Styczen", "Luty", "Marzec",
        "Kwiecien", "Maj", "Czerwiec", "Lipiec");
cbox.setVisibleRowCount(3);
cbox.setEditable(true);
cbox.setPromptText("wpisz nazwe miesiaca:");
cbox.setOnAction(e->lbl.setText(cbox.getValue()));
```

- Wybieramy tylko z listy
- Lista jest wyświetlona na stałe
- Lista może być przewijana pionowo (typowe) lub poziomo (kto o to prosił?)
- Możemy zaznaczyć kilka elementów (Ctrl+Click) lub (Ctrl+Shift+pierwszy+ostatni) po ustawieniu SelectionMode.MULTIPLE

```
1  package application;
2
3  import javafx.application.Application;
4  import javafx.scene.Scene;
5  import javafx.scene.control.ListView;
6  import javafx.scene.layout.HBox;
7  import javafx.stage.Stage;
8  public class ListBoxExample extends Application {
9      @Override
10     public void start(Stage primaryStage) {
11         try {
12             HBox hbox = new HBox(10);
13             ListView<String> list1 = new ListView<String>();
14             ListView<String> list2 = new ListView<String>();
15             list1.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
16             ObservableList<String> lista = list1.getItems();
17             list2.setItems(lista);
18             lista.addAll("Jablka", "Gruszki", "Sliwki");
19             TextArea text = new TextArea();
20             text.textProperty().bind(list1.getSelectionModel().selectedItemProperty());
21             Button btn = new Button("Dodaj");
22             btn.setOnAction(e->{lista.add("Morele");});
23             hbox.getChildren().addAll(list1, text, list2, btn);
24             Scene scene = new Scene(hbox,400,400);
25             primaryStage.setScene(scene);
26             primaryStage.setTitle("INU 1");
27             primaryStage.show();
28
29         } catch(Exception e) {
```

```
            e.printStackTrace();
        }
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

Reprezentacja danych w postaci tabelarycznej

- - ma wbudowane sortowanie po kolumnach
- - ma wbudowane zmiany rozmiaru kolumn
- - ma wbudowane przenoszenie kolumn

Najważniejsze klasy

**TableView** - klasa prezentująca dane

**TableColumn** - klasa definiująca kolumnę

**TableCell** - klasa umożliwiająca opis komórki

```
1 TableView<Student> table = new TableView<Student>();
2 TableColumn<Student,String> firstNameCol = new TableColumn<Student,String>("Imie");
3 TableColumn<Student,String> lastNameCol = new TableColumn<Student,String>("Nazwisko");
4 table.getColumns().addAll(firstNameCol, lastNameCol);
```

ale tablica nie ma danych ... :-)

- klasa jest wewnętrzna, ale statyczna, nie wymaga referencji do zewnętrznej

```
1     public static class Student {
2         private final SimpleStringProperty firstName;
3         private final SimpleStringProperty lastName;
4 
5         private Student(String fName, String lName) {
6             this.firstName = new SimpleStringProperty(fName);
7             this.lastName = new SimpleStringProperty(lName);
8         }
9 
10        public String getFirstName() { return firstName.get(); }
11        public void setFirstName(String fName) {
12            firstName.set(fName);
13        }
14 
15        public String getLastName() {  return lastName.get(); }
16        public void setLastName(String fName) {
17            lastName.set(fName);
18        }
19    }
```

- możemy teraz utworzyć dane

```
final ObservableList<Student> data = FXCollections.observableArrayList(
    new Student("Rowan", "Atkinson"),
    new Student("Jeremy", "Clarkson")
);
```

- i załadować do tabelki

```
    firstNameCol.setMinWidth(100);
    firstNameCol.setCellValueFactory(
            new PropertyValueFactory<Student, String>("firstName"));

    lastNameCol.setMinWidth(100);
    lastNameCol.setCellValueFactory(
            new PropertyValueFactory<Student, String>("lastName"));

    table.setItems(data);
```

Taka niby nudna kontrolka która niewiele robi, ale zajmuje miejsce w drzewie Nodów

- może być pionowy i poziomy
- zajmuje całą szerokość, ale można go skrócić
- można użyć styli aby go narysować

```
final String[] names = new String[]{"March", "April", "May",
    "June", "July", "August"};
final CheckBox[] cbs = new CheckBox[names.length];
final Separator separator = new Separator();
final VBox vbox = new VBox();

for (int i = 0; i < names.length; i++) {
    cbs[i] = new CheckBox(names[i]);
}

separator.setMaxWidth(40);
separator.setAlignment(Pos.CENTER_LEFT);
vbox.getChildren().addAll(cbs);
vbox.setSpacing(5);
vbox.getChildren().add(3, separator);
```

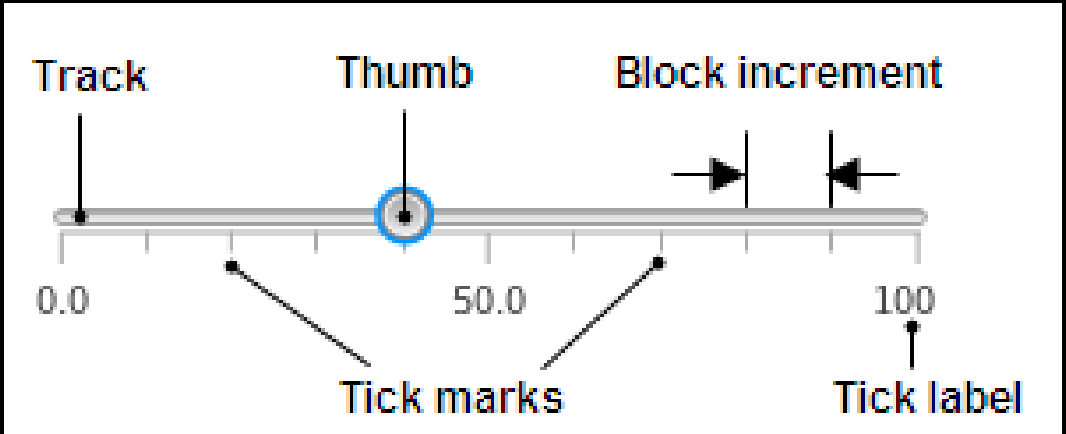
controlStyle.css:

```
.separator{
    -fx-background-color: #e79423;
    -fx-background-radius: 2;
}
```

ale to należy włączyć ...

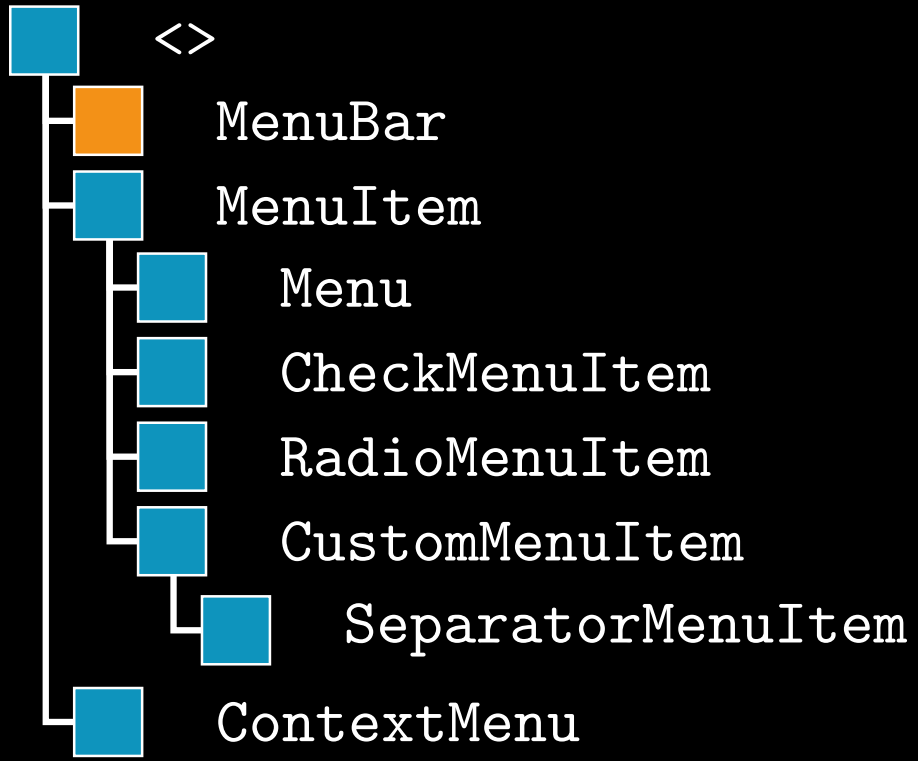
```
scene.getStylesheets().add("separatorsample/controlStyle.css");
```

- służy do wygodnego wprowadzenia wartości z pewnego zakresu




```
Slider slider = new Slider();
slider.setMin(0);
slider.setMax(60);
slider.setValue(1);
slider.setShowTickLabels(true);
slider.setShowTickMarks(true);
slider.setMajorTickUnit(20);
slider.setMinorTickCount(2);
slider.setBlockIncrement(10);
slider.valueProperty().addListener(new ChangeListener<Number>() {
    public void changed(ObservableValue<? extends Number> ov,
        Number old_val, Number new_val) {
            Double val = new_val.doubleValue();
            labelka.setText(String.format("%.2f", new_val));
    }
});
```

- Jest kontrolką
- dodajemy Top kontrolki borderPane



```
    MenuBar menuBar = new MenuBar();
    Menu fileMenu = new Menu("File");
    fileMenu.getItems().add(new MenuItem("Save"));
    fileMenu.setOnAction(e->System.out.println("saved"));
    menuBar.getMenus().add(fileMenu);
    root.setTop(menuBar);
```

## Akceleratory

```
MenuItem clear = new MenuItem("Clear");
    clear.setAccelerator(KeyCombination.keyCombination("Ctrl+X"));
    clear.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent t) {
          ...
        }
});
```

```
CheckMenuItem cmi = new CheckMenuItem(title);
cmi.setSelected(true);
cmi.selectedProperty().addListener(new ChangeListener<Boolean>() {
    public void changed(ObservableValue ov,
    Boolean old_val, Boolean new_val) {
    ...
    }
});
...
```

ukrywanie menu - funkcja `setDisable(True)`

to my musimy je pokazać

```
final ContextMenu cm = new ContextMenu();
MenuItem cmItem = new MenuItem("Copy");
cmItem.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent e) {
        ...
    }
});
cm.getItems().add(cmItem1);

pnl.addEventHandler(MouseEvent.MOUSE_CLICKED,
    new EventHandler<MouseEvent>() {
        @Override public void handle(MouseEvent e) {
            if (e.getButton() == MouseButton.SECONDARY)
                cm.show(pic, e.getScreenX(), e.getScreenY());
        }
});
```

- Służy do wyboru pliku/katalogu

Wybór pojedynczego pliku:

```
FileChooser fileChooser = new FileChooser();
File file = fileChooser.showOpenDialog(stage);
if (file != null) {
```

Wybór listy

```
List<File> list = fileChooser.showOpenMultipleDialog(stage);
if (list != null) {
    for (File file : list) {
```

Zapis pliku

```
File file = fileChooser.showSaveDialog(stage);
if (file != null) {
```

Ustawianie parametrów:

**setTitle** -ustawianie tytułu (pod Linuxem nie działa)

**setInitialDirectory** - katalog, gdzie startujemy

Ustawianie filtrów:

```
fileChooser.getExtensionFilters().addAll(
                new FileChooser.ExtensionFilter("All", "*.*"),
                new FileChooser.ExtensionFilter("csv", "*.csv")
            );
```

Wybór katalogu:

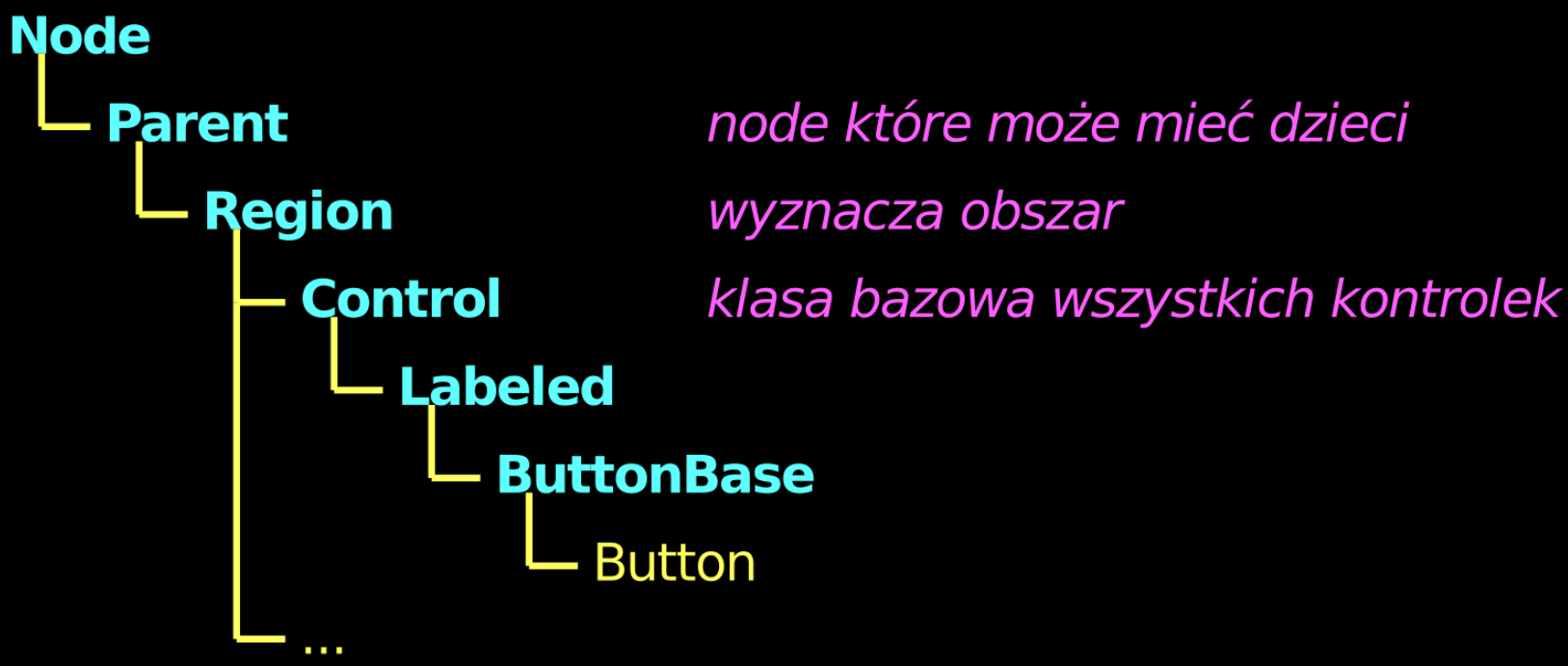
```
DirectoryChooser directoryChooser = new DirectoryChooser();
File selectedDirectory = directoryChooser.showDialog(stage);
if (selectedDirectory != null) {
```

- Służy do tworzenia przycisku

```
HBox hbox = new HBox(3);
Button btn1 = new Button();
Button btn2 = new Button("Accept");
Image img = new Image(getClass().getResourceAsStream("button_ok_16.png"));
Button btn3 = new Button("Accept", new ImageView(img));
hbox.getChildren().addAll(btn1,btn2,btn3);
```

**btn.setGraphicTextGap(double)** - zwiększa odstęp między elementem graficznym a tekstem

- rozszerza `Labelled`:




ma metody:

**setText(String text)** - ustawia tekst

**setGraphic(Node graphic)** - ustawia grafikę

- rozszerza `Node`, czyli możemy wykorzystać efekty z pakietu `javafx.scene.effect`:

```
DropShadow shadow = new DropShadow();
btn3.addEventHandler(MouseEvent.MOUSE_ENTERED, (MouseEvent e) -> {
    ((Node)e.getSource()).setEffect(shadow);
});

btn3.addEventHandler(MouseEvent.MOUSE_EXITED, (MouseEvent e) -> {
    ((Node)e.getSource()).setEffect(null);
});
```

- rozszerza `Node`, czyli możemy wykorzystać efekty z pakietu `javafx.scene.effect`:

```
File f = new File("button-test.css");
scene.getStylesheets().clear();
scene.getStylesheets().add("file:///" + f.getAbsolutePath().replace("\\", "/"));
btn2.getStyleClass().add("button-green");
```

Zmieni wygląd przycisku, jeżeli w pliku `button-test.css` katalogu głównym będzie:

```
.button-green{
    -fx-font: 22 arial;
    -fx-base: #b6e7c9;
}
```

- Jeżeli nie dodamy do grupy to zachowuje się jak checkbox (czyli niezgodnie z intuicją)
- W grupie tylko jeden jest zaznaczony

```
VBox vbox = new VBox(3);
final ToggleGroup group = new ToggleGroup();
RadioButton rb1 = new RadioButton("Shadow");
rb1.setToggleGroup(group);
rb1.setSelected(true);
RadioButton rb2 = new RadioButton("Ok");
rb2.setToggleGroup(group);
RadioButton rb3 = new RadioButton("Cancel");
rb3.setToggleGroup(group);
vbox.getChildren().addAll(rb1,rb2,rb3);
```

- Obsługa zaznaczenia na poziomie grupy:

```
rb1.setToggleGroup(group);
rb1.setSelected(true);
rb1.setUserData("data1");
RadioButton rb2 = new RadioButton("Ok");
rb2.setToggleGroup(group);
rb2.setUserData("data2");
RadioButton rb3 = new RadioButton("Cancel");
rb3.setToggleGroup(group);
rb3.setUserData("data3");
vbox.getChildren().addAll(rb1,rb2,rb3);

group.selectedToggleProperty().addListener(
        (ObservableValue<? extends Toggle> ov, Toggle old_toggle,
        Toggle new_toggle) -> {
            if (group.getSelectedToggle() != null)
                System.out.println(group.getSelectedToggle().getUserData().toString());
    });
```

- Analogiczne do RadioButton
- Istnieje możliwość wyłączenia przycisku

- `setIndeterminate(True)` powoduje że checkbox nie słucha
- `setSelected()` ustawia wartość
- Nie tworzą grup, nasłuchujemy poszczególne czekboxy:

```
cb.selectedProperty().addListener(
        (ObservableValue<? extends Boolean> ov,
            Boolean old_val, Boolean new_val) -> {
                ...
    })
```

- Konwencja nazewnictwa.
- zmiany: `java.beans.PropertyChangeSupport` ale nie do końca wygodnie

```java
public class User {
    private String password;
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

- - właściwości do czytania/pisania (javafx.beans.property.SimpleBooleanProperty)

```
StringProperty password = new SimpleStringProperty("elka");
password.set("weti");
System.out.println("After modification " + password.get() ); // weti
```

- właściwości do czytania (javafx.beans.property.ReadOnlyBooleanWrapper)

```
ReadOnlyStringWrapper userName = new ReadOnlyStringWrapper("jmy");
ReadOnlyStringProperty readOnlyUserName = userName.getReadOnlyProperty();
System.out.println("User: " + password.get() ); // jmy
```

- Metoda zwracająca właściwość - potrzebna do wiązania z innymi właściwościami
- Metody dostępu do wartości właściwości

```
1  public class Book {
2    private StringProperty title
3      = new SimpleStringProperty(this, "title", "Unknown");
4
5    public final StringProperty titleProperty() {
6      return title;
7    }
8    public final String getTitle() {
9      return title.get();
10   }
11   public final void setTitle(String title) {
12     this.title.set(title);
13   }
14 }
```

- dodajemy słuchacza zmian

```
// change listener
salary.addListener((ObservableValue<? extends Number> ov, Number oldVal,
                    Number newVal) -> {
  System.out.println(newVal);
})
```

- gdy chcemy zaznaczyć, że wartość ma być nieustalona

```
// invalidation listener
salary.addListener((Observable o) -> {
  System.out.println("Juz nie pracujesz");
});
```

**wartościowanie leniwe (lazy evaluation)** - argumenty funkcji wyznaczane tylko wtedy gdy potrzebne

**wartościowanie zachłanne (eager evaluation)** - argumenty funkcji wyznaczane zawsze

- Wiązanie dwukierunkowe - obie właściwości muszą być zapisywalne

```
StringProperty fname = new SimpleStringProperty();
StringProperty display = new SimpleStringProperty();
fname.bindBidirectional(display);
display.set("Ala");
fname.set("Kota");
System.out.println(display.get() );
System.out.println(fname.get() );
```

- mamy operacje na właściwościach

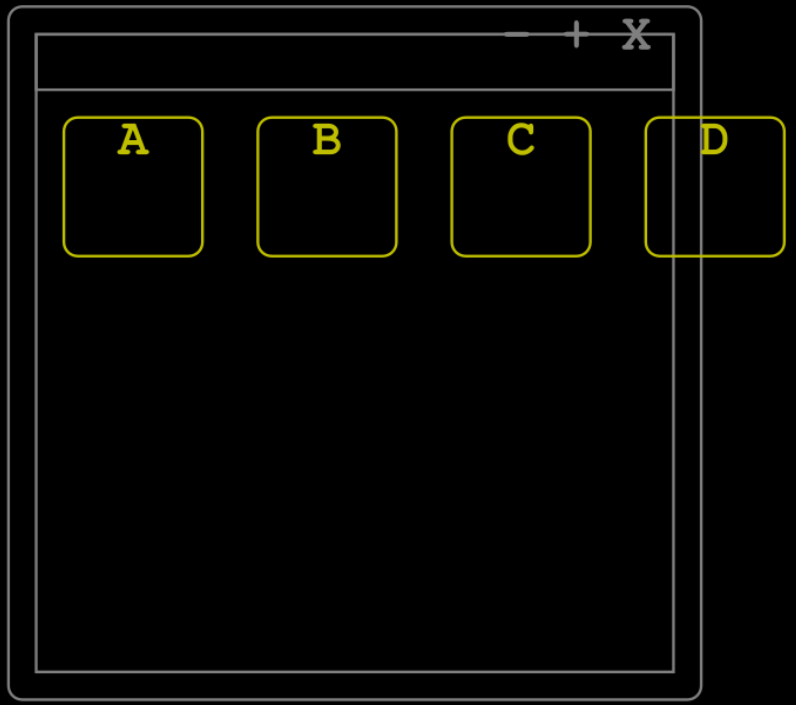
```
1 IntegerProperty width = new SimpleIntegerProperty(10);
2 IntegerProperty height = new SimpleIntegerProperty(10);
3 NumberBinding area = width.multiply(height);
```

Właściwość jest wyliczana leniwie

- Dziedziczymy po klasie, mamy moc języka

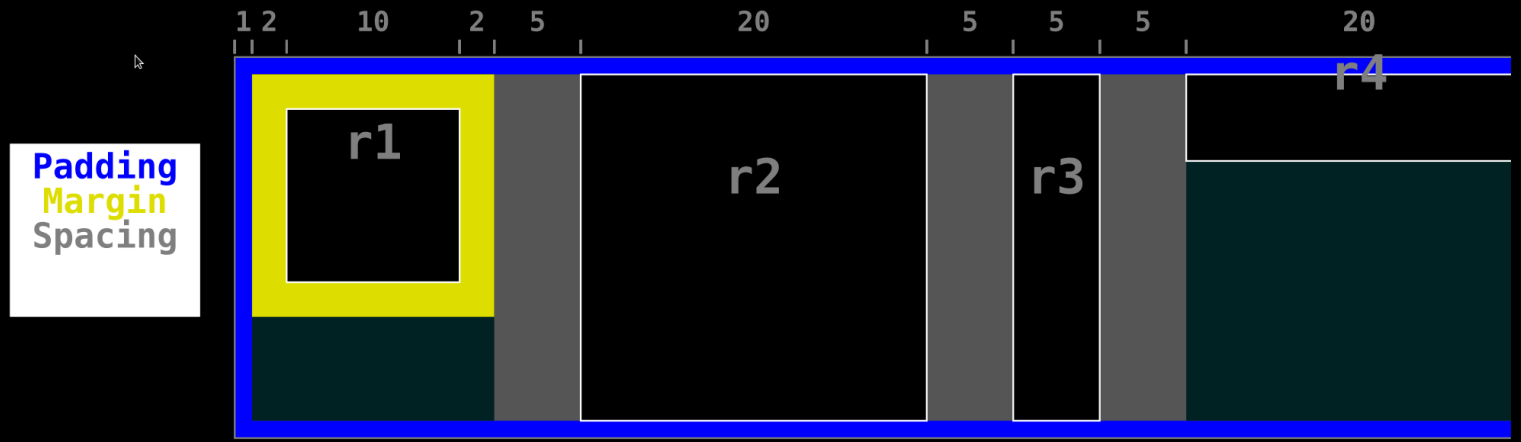
```
1
2  DoubleProperty radius = new SimpleDoubleProperty(2);
3  DoubleBinding volumeOfSphere = new DoubleBinding() {
4    {
5      super.bind(radius); // initial bind
6    }
7
8    @Override
9    protected double computeValue() {
10     return (4 / 3 * Math.PI * Math.pow(radius.get(), 3));
11   }
12 };
```

- Dzieci dodawane z prawej strony
- Rozmiar dzieci jest honorowany
- Ustawiamy `border, padding, margin, spacing, alignment`
  - programowo
  - css
- dla kontrolek które mogą zmieniać rozmiar można to określić:

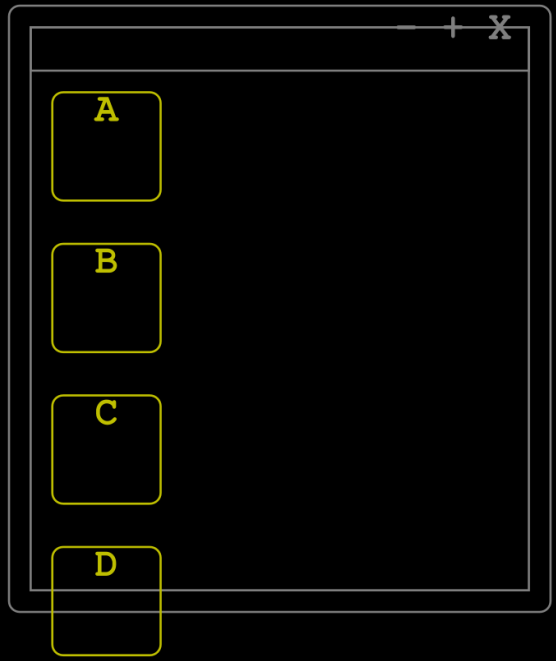



```
1 TextField textField = new TextField();
2 HBox.setHgrow(textField, Priority.ALWAYS);
```

```
HBox hbox = new HBox(5);         // distance between children
hbox.setPadding(new Insets(1)); // distance between row and border

Rectangle r1 = new Rectangle(10, 10);// small
Rectangle r2 = new Rectangle(20, 20);// large
Rectangle r3 = new Rectangle(5, 20); // tall
Rectangle r4 = new Rectangle(20, 5); // short

HBox.setMargin(r1, new Insets(2,2,2,2));//additional margin for a controll
hbox.getChildren().addAll(r1, r2, r3, r4);
```



- Dzieci dodawane od góry
- Układ identyczny do układu HBox



- Wszystkie kontrolki w tym samym miejscu
- Pozwala składać wygląd
- Pozwala pokazywać jedną bez zmiany drzewa kontrolek

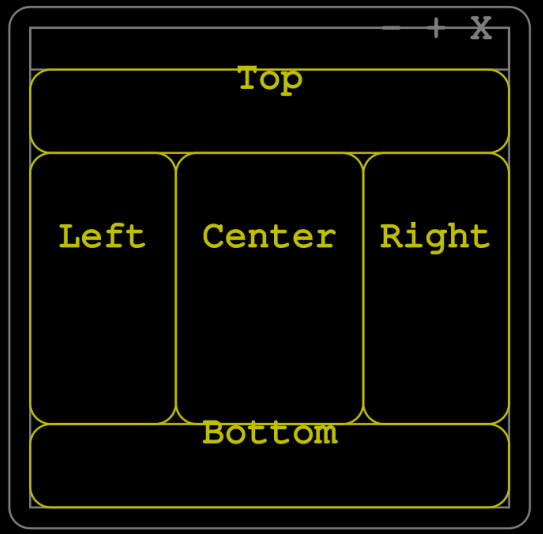
- Wszystkie kontrolki w tym samym miejscu
- Pozwala składać wygląd
- Pozwala pokazywać jedną bez zmiany drzewa kontrolek

- Brzegi dzieci kotwiczymy do brzegów układu, podając odległość
- Zmiany rozmiaru powodują dostosowanie rozmiaru dzieci do nowego rozmiaru


```
// Lista przyczepiona do wszystkich rogow
ListView list = new ListView();
AnchorPane.setTopAnchor(list, 10.0);
AnchorPane.setLeftAnchor(list, 10.0);
AnchorPane.setRightAnchor(list, 65.0);
AnchorPane.setBottomAnchor(list, 10.0);
// Przycisk przyczepiony do gornego prawego rogu
Button button = new Button("Add");
AnchorPane.setTopAnchor(button, 10.0);
AnchorPane.setRightAnchor(button, 10.0);
anchorpane.getChildren().addAll(list, button);
```

- W każdym obszarze tylko jedna kontrolka
- domyślne ułożenie:

Top: `Pos.TOP_LEFT`
Bottom: `Pos.BOTTOM_LEFT`
Left: `Pos.TOP_LEFT`
Right: `Pos.TOP_RIGHT`
Center: `Pos.CENTER`




```
1 BorderPane root = new BorderPane();
2 Scene scene = new Scene(root, 400, 250, Color.WHITE);
3 ...
4 root.setCenter(gridpane);
```

- Tyle kolumn i wierszy ile trzeba
- dodajemy kontrolki do wierszy i kolumn

```
GridPane gridpane = new GridPane();
gridpane.setPadding(new Insets(5));
gridpane.setHgap(5);
gridpane.setVgap(5);
ColumnConstraints column1 = new ColumnConstraints(100);
ColumnConstraints column2 = new ColumnConstraints(50, 150, 300);
column2.setHgrow(Priority.ALWAYS);
gridpane.getColumnConstraints().addAll(column1, column2);
...
GridPane.setHalignment(fNameLbl, HPos.RIGHT);
gridpane.add(fNameLbl, 0, 0);
GridPane.setHalignment(lNameLbl, HPos.RIGHT);
gridpane.add(lNameLbl, 0, 1);
// kontrolka, wiersz, kolumna
```

- Chcemy mieć możliwość zmiany wyglądu aplikacji w sposób niezależny od kodu (podział problemu jest zawsze dobry)

```
.root {
-fx-font-size: 26px;
}
.button {
-fx-font-size: 20px;
-fx-max-width:2000px
}
#btn1 {
-fx-font: bold italic 20pt "LucidaBrightDemiBold";
}
```

- Chcemy mieć możliwość zmiany wyglądu aplikacji w sposób niezależny od kodu (podział problemu jest zawsze dobry)
- Pomysł z HTML-a - opis wyglądu za pomocą definiowania styli CSS

```
.root {
-fx-font-size: 26px;
}
.button {
-fx-font-size: 20px;
-fx-max-width:2000px
}
#btn1 {
-fx-font: bold italic 20pt "LucidaBrightDemiBold";
}
```

- Chcemy mieć możliwość zmiany wyglądu aplikacji w sposób niezależny od kodu (podział problemu jest zawsze dobry)
- Pomysł z HTML-a - opis wyglądu za pomocą definiowania styli CSS
- Każdy węzeł drzewa komponentów ma dodaną automatycznie klasę CSS

```
.root {
-fx-font-size: 26px;
}
.button {
-fx-font-size: 20px;
-fx-max-width:2000px
}
#btn1 {
-fx-font: bold italic 20pt "LucidaBrightDemiBold";
}
```

- Chcemy mieć możliwość zmiany wyglądu aplikacji w sposób niezależny od kodu (podział problemu jest zawsze dobry)
- Pomysł z HTML-a - opis wyglądu za pomocą definiowania styli CSS
- Każdy węzeł drzewa komponentów ma dodaną automatycznie klasę CSS
- Można dodawać własne klasy `getStyleClass().add(String styleClass)`

```
.root {
-fx-font-size: 26px;
}
.button {
-fx-font-size: 20px;
-fx-max-width:2000px
}
#btn1 {
-fx-font: bold italic 20pt "LucidaBrightDemiBold";
}
```

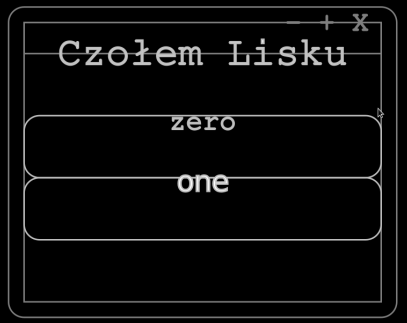
- Chcemy mieć możliwość zmiany wyglądu aplikacji w sposób niezależny od kodu (podział problemu jest zawsze dobry)
- Pomysł z HTML-a - opis wyglądu za pomocą definiowania styli CSS
- Każdy węzeł drzewa komponentów ma dodaną automatycznie klasę CSS
- Można dodawać własne klasy `getStyleClass().add(String styleClass)`
- Każdy węzeł może mieć unikalny id `setId(String id)`

```
.root {
-fx-font-size: 26px;
}
.button {
-fx-font-size: 20px;
-fx-max-width:2000px
}
#btn1 {
-fx-font: bold italic 20pt "LucidaBrightDemiBold";
}
```

- Chcemy mieć możliwość zmiany wyglądu aplikacji w sposób niezależny od kodu (podział problemu jest zawsze dobry)
- Pomysł z HTML-a - opis wyglądu za pomocą definiowania styli CSS
- Każdy węzeł drzewa komponentów ma dodaną automatycznie klasę CSS
- Można dodawać własne klasy `getStyleClass().add(String styleClass)`
- Każdy węzeł może mieć unikalny id `setId(String id)`
- Takie same właściwości obowiązują dzieci (chyba że nadpiszą)

```
.root {
-fx-font-size: 26px;
}
.button {
-fx-font-size: 20px;
-fx-max-width:2000px
}
#btn1 {
-fx-font: bold italic 20pt "LucidaBrightDemiBold";
}
```

# W aplikacji musimy tylko załadować plik CSS




```
scene.getStylesheets().add(getClass().getResource("application.css")
                                     .toExternalForm());
Text lbl = new Text("Hey fox");
VBox vbox = new VBox(10);
Button btn0 = new Button("zero");
Button btn1 = new Button("one");
btn1.setId("btn1");
vbox.getChildren().addAll(btn0,btn1);
root.setCenter(vbox);
root.setTop(lbl);
```

- JavaFx zawiera 2 zestawy domyślne:

- JavaFx zawiera 2 zestawy domyślne:
  - `STYLESHEET_CASPIAN` domyślny w wersji 2.0

```
1 scene.getStylesheets().clear();
2 setUserAgentStylesheet(STYLESHEET_CASPIAN);
```

- JavaFx zawiera 2 zestawy domyślne:
  - `STYLESHEET_CASPIAN` domyślny w wersji 2.0
  - `STYLESHEET_MODENA` domyślny w wersji 8.0

```
1 scene.getStylesheets().clear();
2 setUserAgentStylesheet(STYLESHEET_CASPIAN);
```

- JavaFx zawiera 2 zestawy domyślne:
  - `STYLESHEET_CASPIAN` domyślny w wersji 2.0
  - `STYLESHEET_MODENA` domyślny w wersji 8.0
- przekazanie `Null` ładuje domyślny arkusz

```
1 scene.getStylesheets().clear();
2 setUserAgentStylesheet(STYLESHEET_CASPIAN);
```

- JavaFx zawiera 2 zestawy domyślne:
  - `STYLESHEET_CASPIAN` domyślny w wersji 2.0
  - `STYLESHEET_MODENA` domyślny w wersji 8.0
- przekazanie `Null` ładuje domyślny arkusz

```
1 scene.getStylesheets().clear();
2 setUserAgentStylesheet(STYLESHEET_CASPIAN);
```

- można dograć własne arkusze:

- JavaFx zawiera 2 zestawy domyślne:
  - `STYLESHEET_CASPIAN` domyślny w wersji 2.0
  - `STYLESHEET_MODENA` domyślny w wersji 8.0
- przekazanie `Null` ładuje domyślny arkusz

```
scene.getStylesheets().clear();
setUserAgentStylesheet(STYLESHEET_CASPIAN);
```

- można dograć własne arkusze:
- nie trzeba definiować całości - wystarczy nadpisać zmianami

```
scene.getStylesheets().clear();          // clear all settings
setUserAgentStylesheet(null);            // load default settings
scene.getStylesheets().add(getClass()    // override some settings
     .getResource("ntr.css").toExternalForm());
```

- nazwy właściwości są konwertowane z Wielbłąda do podkreśleń :
  - `btn1.setMaxWidth(2000);` zamieniamy na `-fx-max-width:2000px`

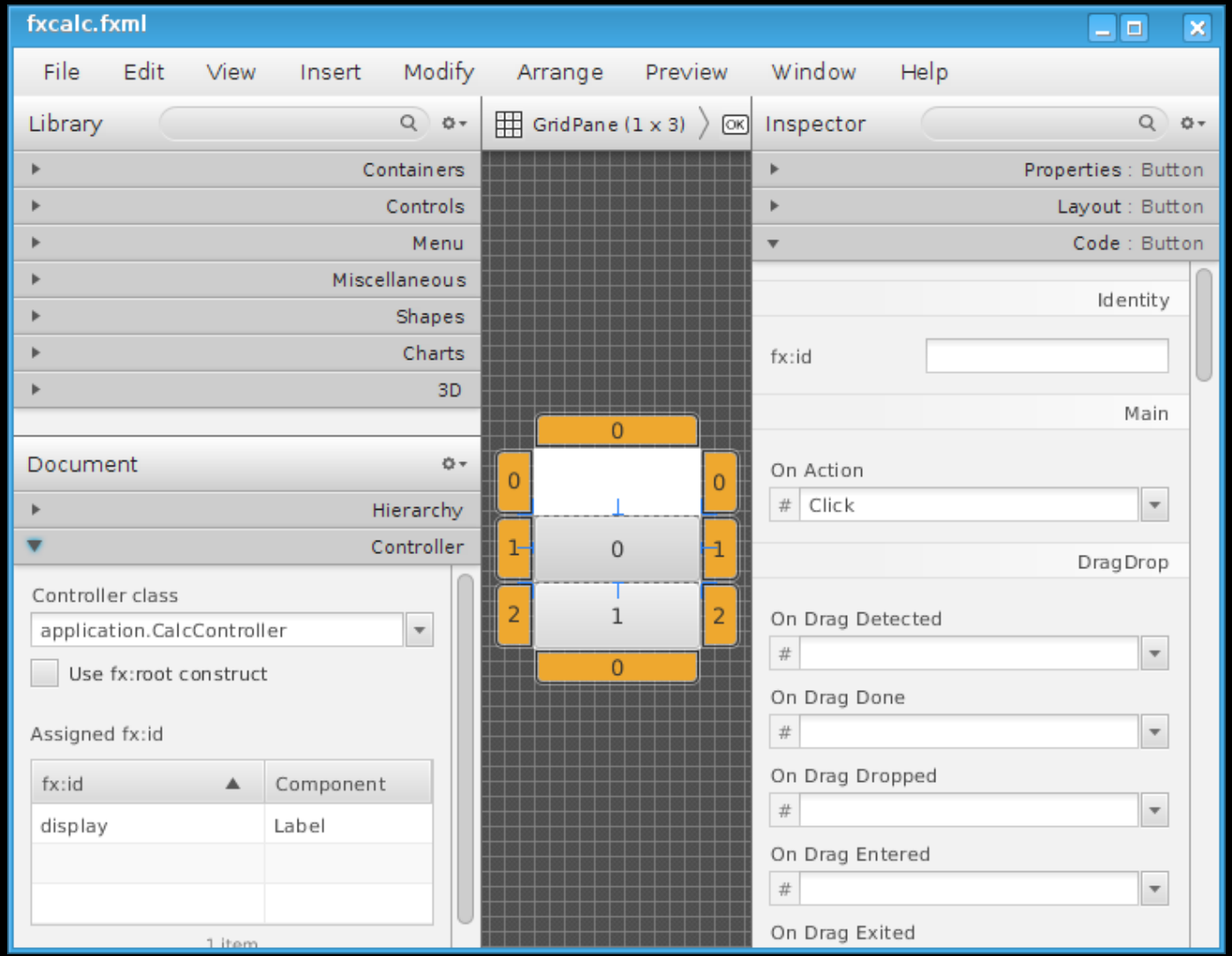
```
/* all buttons, which direct parent is HBox */
.hbox > .button { -fx-text-fill: black; }

/* labels and text boxes */
.label, .text { -fx-font-size: 20px; }
```

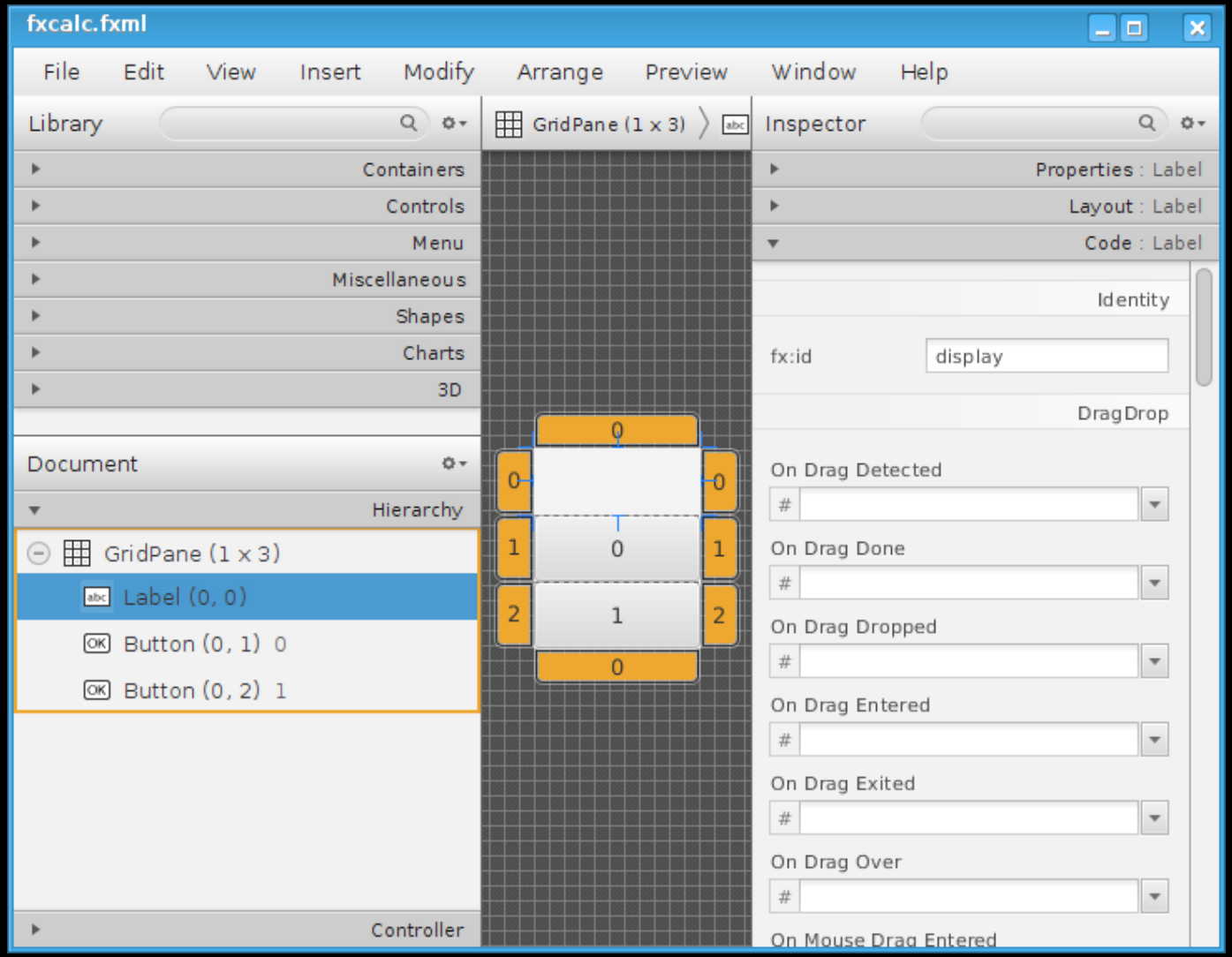
- pseudoselektory - właczają się, gdy warunek jest spełniony

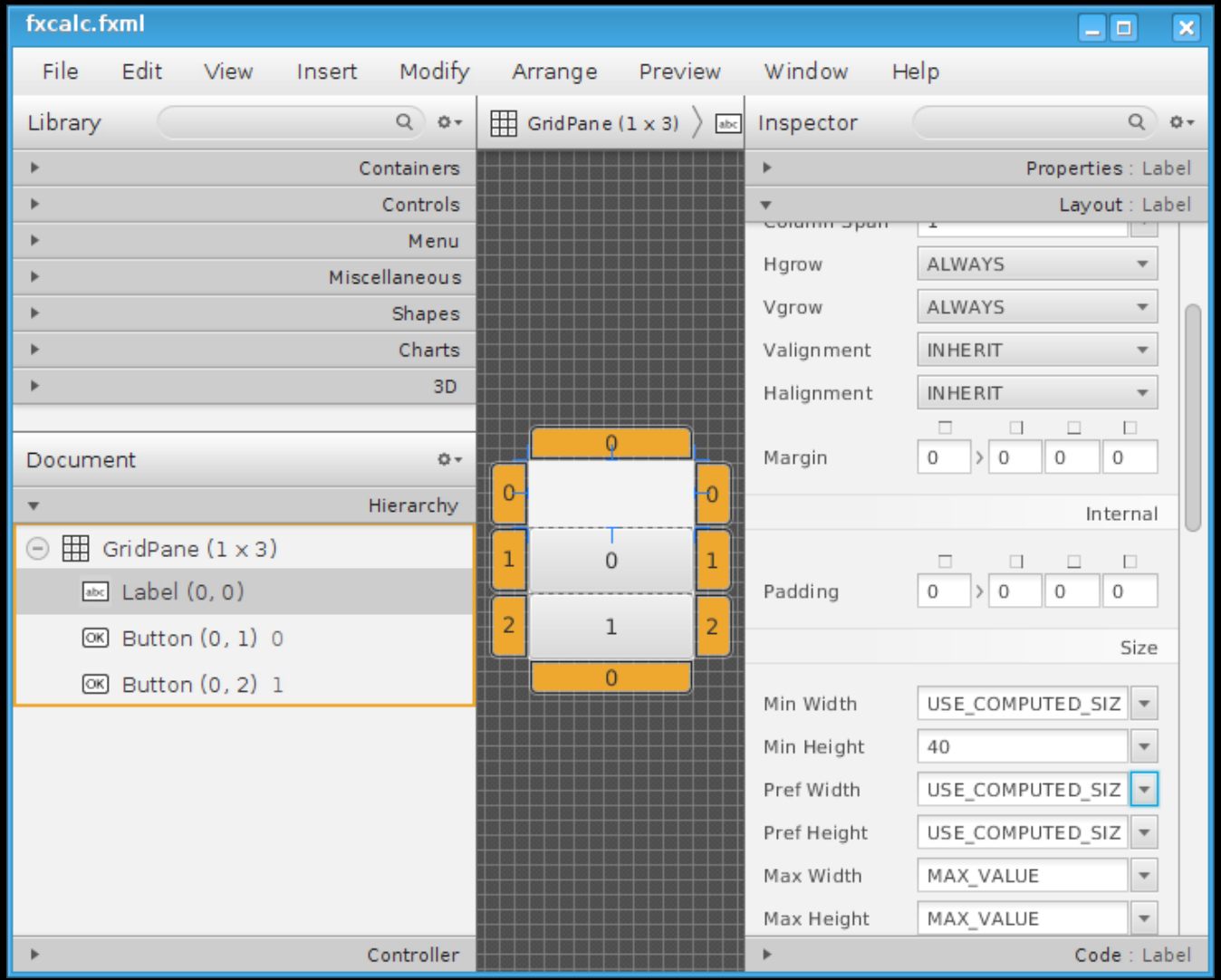
```
.num-button {
-fx-background-color: white,blue,white;
-fx-background-radius: 50%;
-fx-background-insets: 0, 1, 2;
-fx-font-family: "Helvetica";
-fx-text-fill: black;
}
.num-button:hover {
-fx-background-color: black,white,black;
-fx-text-fill: white;
}
```

```
1  package application;
2
3  import javafx.event.ActionEvent;
4  import javafx.fxml.FXML;
5  import javafx.scene.control.Button;
6  import javafx.scene.control.Label;
7
8  public class CalcController {
9    @FXML
10   public void Click(ActionEvent event) {
11     String v = ((Button)event.getSource()).getText();
12     display.setText(display.getText()+v);
13   }
14
15   @FXML
16   private Label display;
17 }
```

```
1  package application;
2
3  import javafx.application.Application;
4
5  import javafx.fxml.FXMLLoader;
6  import javafx.scene.Parent;
7  import javafx.scene.Scene;
8  import javafx.stage.Stage;
9
10 public class FxCalc extends Application {
11
12   @Override
13   public void start(Stage primaryStage) {
14     try {
15       Parent root = FXMLLoader.load(getClass().getResource("fxcalc.fxml"));
16       Scene scene = new Scene(root,400,400);
17       scene.getStylesheets().add(getClass().getResource("fxcalc.css").toExternalForm());
18       primaryStage.setScene(scene);
19       primaryStage.setTitle("Scene Buildered");
20       primaryStage.show();
21     } catch(Exception e) {
22       e.printStackTrace();
23     }
24   }
25
26   public static void main(String[] args) {
27     launch(args);
28   }
29 }
```


fxcalc.fxml
File
Edit
View
Insert
Modify
Arrange
Preview
Window
Help
Library
Containers
Controls
Menu
Miscellaneous
Shapes
Charts
3D
Document
Hierarchy
Controller
Controller class
application.CalcController
Use fx:root construct
Assigned fx:id
fx:id
Component
display
Label
1 item
GridPane (1 x 3)
OK
0
0
0
1
0
1
2
1
2
0
Inspector
Properties : Button
Layout : Button
Code : Button
Identity
fx:id
Main
On Action
#
Click
DragDrop
On Drag Detected
On Drag Done
On Drag Dropped
On Drag Entered
On Drag Exited


fxcalc.fxml
File
Edit
View
Insert
Modify
Arrange
Preview
Window
Help
Library
Containers
Controls
Menu
Miscellaneous
Shapes
Charts
3D
Document
Hierarchy
GridPane (1 x 3)
Label (0, 0)
Button (0, 1) 0
Button (0, 2) 1
Controller
GridPane (1 x 3)
Inspector
Properties : Label
Layout : Label
Code : Label
Identity
fx:id
display
DragDrop
On Drag Detected
On Drag Done
On Drag Dropped
On Drag Entered
On Drag Exited
On Drag Over
On Mouse Drag Entered


fxcalc.fxml
File
Edit
View
Insert
Modify
Arrange
Preview
Window
Help
Library
Containers
Controls
Menu
Miscellaneous
Shapes
Charts
3D
Document
Hierarchy
GridPane (1 x 3)
Label (0, 0)
Button (0, 1) 0
Button (0, 2) 1
Controller
GridPane (1 x 3)
Inspector
Properties : Label
Layout : Label
Hgrow
ALWAYS
Vgrow
ALWAYS
Valignment
INHERIT
Halignment
INHERIT
Margin
Internal
Padding
Size
Min Width
USE_COMPUTED_SIZ
Min Height
40
Pref Width
USE_COMPUTED_SIZ
Pref Height
USE_COMPUTED_SIZ
Max Width
MAX_VALUE
Max Height
MAX_VALUE
Code : Label

- Pozwala na oddzielenie projektu widoku od logiki biznesowej

```
public class FxScene extends Application {
    @Override
    public void start(Stage primaryStage) {
        try {
            Parent root = FXMLLoader.load(getClass()
                    .getResource("fxscene.fxml"));
            Scene scene = new Scene(root,400,400);
            primaryStage.setScene(scene);
            primaryStage.setTitle("Scene Buildered");
            primaryStage.show();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
```

- Pozwala na oddzielenie projektu widoku od logiki biznesowej
- Mogą pracować dwa zespoły

```
public class FxScene extends Application {
    @Override
    public void start(Stage primaryStage) {
        try {
            Parent root = FXMLLoader.load(getClass()
                    .getResource("fxscene.fxml"));
            Scene scene = new Scene(root,400,400);
            primaryStage.setScene(scene);
            primaryStage.setTitle("Scene Buildered");
            primaryStage.show();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
```

- Pozwala na oddzielenie projektu widoku od logiki biznesowej
- Mogą pracować dwa zespoły
- Plik fxml jest zewnętrznym plikiem ładowanym dynamicznie

```
public class FxScene extends Application {
    @Override
    public void start(Stage primaryStage) {
        try {
            Parent root = FXMLLoader.load(getClass()
                    .getResource("fxscene.fxml"));
            Scene scene = new Scene(root,400,400);
            primaryStage.setScene(scene);
            primaryStage.setTitle("Scene Buildered");
            primaryStage.show();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
```

- Pozwala na oddzielenie projektu widoku od logiki biznesowej
- Mogą pracować dwa zespoły
- Plik fxml jest zewnętrznym plikiem ładowanym dynamicznie

```
public class FxScene extends Application {
    @Override
    public void start(Stage primaryStage) {
        try {
            Parent root = FXMLLoader.load(getClass()
                        .getResource("fxscene.fxml"));
            Scene scene = new Scene(root,400,400);
            primaryStage.setScene(scene);
            primaryStage.setTitle("Scene Buildered");
            primaryStage.show();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
```

- Zyskaliśmy wielkie uproszczenie metody start

- Scene Builder 'widzi' tylko pola i metody oznaczone atrybutem `@FXML`

```
    @FXML
    public void Click() {
        System.out.println(edbox.getText());
    }
    @FXML
    private TextField edbox;
    public static void main(String[] args) {
        launch(args);
    }
}
```

- Scene Builder 'widzi' tylko pola i metody oznaczone atrybutem `@FXML`

```
    @FXML
    public void Click() {
        System.out.println(edbox.getText());
    }
    @FXML
    private TextField edbox;
    public static void main(String[] args) {
        launch(args);
    }
}
```

- Kontroler podłączamy zakładce Dokument po prawej stronie

- Scene Builder 'widzi' tylko pola i metody oznaczone atrybutem `@FXML`

```
    @FXML
    public void Click() {
        System.out.println(edbox.getText());
    }
    @FXML
    private TextField edbox;
    public static void main(String[] args) {
        launch(args);
    }
}
```

- Kontroler podłączamy zakładce Dokument po prawej stronie
- Po podłączeniu kontrolera przy definiowaniu obsługi zdarzeń mamy listę metod z atrybutem FXML

- Scene Builder 'widzi' tylko pola i metody oznaczone atrybutem `@FXML`

```
    @FXML
    public void Click() {
        System.out.println(edbox.getText());
    }
    @FXML
    private TextField edbox;
    public static void main(String[] args) {
        launch(args);
    }
}
```

- Kontroler podłączamy zakładce Dokument po prawej stronie
- Po podłączeniu kontrolera przy definiowaniu obsługi zdarzeń mamy listę metod z atrybutem FXML
- Analogicznie musimy podpiąć pozostałe kontrolki - zakładka code, identity, fx:id

```
<AnchorPane xmlns="http://javafx.com/javafx/8"
            xmlns:fx="http://javafx.com/fxml/1"
            fx:controller="application.FxScene">
   <children>
      <VBox prefHeight="200.0" prefWidth="100.0"
            AnchorPane.bottomAnchor="0.0"
            AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0"
            AnchorPane.topAnchor="0.0">
         <children>
            <Button maxWidth="1.7976931348623157E308"
                    mnemonicParsing="false"
                    onAction="#Click" text="Zero" />
            <TextField fx:id="edbox" promptText="wpisz" />
         </children>
      </VBox>
   </children>
</AnchorPane>
```

- Zyskaliśmy wielkie uproszczenie metody start
- Czy jest to jednak bardziej czytelne?

- Nie jest konieczne, aby kontroler znajdował się w tej samej klasie, która jest odpowiedzialna za aplikację

# Wywołanie drugiego okna

```
try {
FXMLLoader fxmlLoader = new FXMLLoader(getClass()
                              .getResource("Second.fxml"));
  Parent root1 = (Parent) fxmlLoader.load();
  Stage stage = new Stage();
  stage.setScene(new Scene(root1));
  stage.showAndWait();
} catch(Exception e) {
  e.printStackTrace();
}
```

- W ekploratorze pakietów wybieramy Export

```
java --module-path /opt/java/javafx-sdk-17.0.1/lib --add-modules javafx.controls,javafx.fxml --enable-preview  "
javafx -cp sample2-1.0-SNAPSHOT.jar edu.pw.elka.App
export PATH_TO_FX=path/to/javafx-sdk-17/lib
javac --module-path
java --module-path
```

- W ekploratorze pakietów wybieramy Export
- W liście opcji wybieramy Java a potem Jar lub Runnable Jar

```
java --module-path /opt/java/javafx-sdk-17.0.1/lib --add-modules javafx.controls,javafx.fxml --enable-preview  "
javafx -cp sample2-1.0-SNAPSHOT.jar edu.pw.elka.App
export PATH_TO_FX=path/to/javafx-sdk-17/lib
javac --module-path
java --module-path
```

- W ekploratorze pakietów wybieramy Export
- W liście opcji wybieramy Java a potem Jar lub Runnable Jar
- utworzony plik wywołujemy za pomocą polecenia `java -jar plik.jar`

```
java --module-path /opt/java/javafx-sdk-17.0.1/lib --add-modules javafx.controls,javafx.fxml --enable-preview  "
javafx -cp sample2-1.0-SNAPSHOT.jar edu.pw.elka.App
export PATH_TO_FX=path/to/javafx-sdk-17/lib
javac --module-path
java --module-path
```

- graf przedstawiający scenę nie jest odporny na modyfikację z innych wątków!
- najlepiej korzystać z pakietu `javafx.concurrent`
- ma interfejs `Worker` i jego dwie implementacje:

  **Task** - obserwowalna implementacja klasy `java.util.concurrent.FutureTask`

  **Service** -

  **WorkerStateEvent** - event który jest wysyłany gy zmienia się stan
- Uruchomienie `Task`:

```
1 Thread th = new Thread(task);
2 th.setDaemon(true);
3 th.start();
```

- lub:

```
1 ExecutorService.submit(task);
```

```
import javafx.concurrent.Task;

Task<Integer> task = new Task<Integer>() {
    @Override protected Integer call() throws Exception {
        int iterations;
        for (iterations = 0; iterations < 100000; iterations++) {
            if (isCancelled()) {
                break;
            }
            System.out.println("Iteration " + iterations);
        }
        return iterations;
    }
};
```

```
import javafx.concurrent.Task;

Task task = new Task<Void>() {
    @Override public Void call() {
        static final int max = 1000000;
        for (int i=1; i<=max; i++) {
            if (isCancelled()) {
               break;
            }
            updateProgress(i, max);
        }
        return null;
    }
};
ProgressBar bar = new ProgressBar();
bar.progressProperty().bind(task.progressProperty());
new Thread(task).start();
```

```
1  public void start(Stage primaryStage) {
2    Button btn = new Button("Sprawdzam");                    // (1)
3    btn.setMaxWidth(100);
4    TextField text = new TextField("tu wpisz imie");         // (1)
5    Label lbl = new Label("podaj imie:");
6    HBox hbox = new HBox(10,lbl,text,btn);
7    Scene scene = new Scene(hbox,400,200);
8    btn.setOnAction(e->System.out.println(text.getText())); //(2)
9    primaryStage.setScene(scene);
10   primaryStage.setTitle("Bardzo prosta aplikacja");
11   primaryStage.show();
12 }
```

- Kontrolki mogą być deklarowane w funkcji (1)
- I wykorzystywane w obsłudze zdarzeń (2)

- możemy zaaplikować następujące transformacje do grupy nodów:

- możemy zaaplikować następujące transformacje do grupy nodów:
  - translacja (`Translate`)

- możemy zaaplikować następujące transformacje do grupy nodów:
  - translacja (`Translate`)
  - rotacja (`Rotation`)

- możemy zaaplikować następujące transformacje do grupy nodów:
  - translacja (`Translate`)
  - rotacja (`Rotation`)
  - skalowanie (`Scaling`)

- możemy zaaplikować następujące transformacje do grupy nodów:
  - translacja (`Translate`)
  - rotacja (`Rotation`)
  - skalowanie (`Scaling`)
  - przekoszenie (`Shearing`) `Shearing`)

- możemy zaaplikować następujące transformacje do grupy nodów:
  - translacja (`Translate`)
  - rotacja (`Rotation`)
  - skalowanie (`Scaling`)
  - przekoszenie (`Shearing`) `Shearing`)
- transformacje możemy wykonywać w 3 współrzędnych (zatem są to transformacje 3D)

- możemy zaaplikować następujące transformacje do grupy nodów:
  - translacja (`Translate`)
  - rotacja (`Rotation`)
  - skalowanie (`Scaling`)
  - przekoszenie (`Shearing`) `Shearing`)
- transformacje możemy wykonywać w 3 współrzędnych (zatem są to transformacje 3D)
- transformacje 'składamy' w kolejności od lewej do prawej

```
Rectangle cube = new Rectangle(50, 50);
cube.setTranslateX(50);
cube.setTranslateY(50);

cube.setRotate(45); // rotation to the center of the object

g.getChildren().add(t);
g.getChildren().add(c);
g.getChildren().add(cube);

t.setTranslateX(0);

Rotate r = new Rotate(45,50,50); // rotation to the given point
cube.getTransforms().addAll(r);
```

**Blend** - nakładanie

**Bloom** - rozświetlenie

**Blur** - rozmazanie

**MotionBlur** - rozmazanie ruchem

**GausianBlur** - rozmazanie

**Drop Shadow Effect** - cienie na scianie

**Inner Shadow Effect** - cienie we wgłębieniu

**Reflection Effect** - odbicie na posadce

**Ligtning** - oświetlenie z zadanego kierunku

- Nakładanie (SRC_ATOP,MULTIPLY,SRC_OVER)

```
static Node blendMode() {
        Rectangle r = new Rectangle();
        r.setX(590);
        r.setY(50);
        r.setWidth(50);
        r.setHeight(50);
        r.setFill(Color.BLUE);

        Circle c = new Circle();
        c.setFill(Color.RED);
        c.setCenterX(590);
        c.setCenterY(50);
        c.setRadius(25);
        c.setBlendMode(BlendMode.SRC_ATOP);

        Group g = new Group();
        g.setBlendMode(BlendMode.SRC_OVER);
        g.getChildren().add(r);
        g.getChildren().add(c);
        return g;
    }
```

- Wszystko powyżej tresholdu (0..1) świeci

```
static Node bloom() {
        Group g = new Group();

        Rectangle r = new Rectangle();
        r.setX(10);
        r.setY(10);
        r.setWidth(160);
        r.setHeight(80);
        r.setFill(Color.DARKBLUE);

        Text t = new Text();
        t.setText("Bloom!");
        t.setFill(Color.YELLOW);
        t.setFont(Font.font("null", FontWeight.BOLD, 36));
        t.setX(25);
        t.setY(65);
        g.setCache(true);
        //g.setEffect(new Bloom());
        Bloom bloom = new Bloom();
        bloom.setThreshold(1.0);
        g.setEffect(bloom);
        g.getChildren().add(r);
        g.getChildren().add(t);
        g.setTranslateX(350);
        return g;
    }
```

- Rozmazujemy

```
static Node boxBlur() {
        Text t = new Text();
        t.setText("Blurry Text!");
        t.setFill(Color.RED);
        t.setFont(Font.font("null", FontWeight.BOLD, 36));
        t.setX(10);
        t.setY(40);

        BoxBlur bb = new BoxBlur();
        bb.setWidth(5);
        bb.setHeight(5);
        bb.setIterations(3);

        t.setEffect(bb);
        t.setTranslateX(300);
        t.setTranslateY(100);

        return t;
    }
```

- Rozmazujemy

```
static Node motionBlur() {
        Text t = new Text();
        t.setX(20.0f);
        t.setY(80.0f);
        t.setText("Motion Blur");
        t.setFill(Color.RED);
        t.setFont(Font.font("null", FontWeight.BOLD, 60));

        MotionBlur mb = new MotionBlur();
        mb.setRadius(15.0f);
        mb.setAngle(45.0f);

        t.setEffect(mb);

        t.setTranslateX(300);
        t.setTranslateY(150);

        return t;
    }
```

- Rozmazujemy

```
static Node gaussianBlur() {
        Text t2 = new Text();
        t2.setX(10.0f);
        t2.setY(140.0f);
        t2.setCache(true);
        t2.setText("Gaussian Blur");
        t2.setFill(Color.RED);
        t2.setFont(Font.font("null", FontWeight.BOLD, 36));
        t2.setEffect(new GaussianBlur());
        return t2;
    }
```

```
static Node dropShadow() {
        Group g = new Group();

        DropShadow ds = new DropShadow();
        ds.setOffsetY(3.0);
        ds.setOffsetX(3.0);
        ds.setColor(Color.GRAY);

        Text t = new Text();
        t.setEffect(ds);
        t.setCache(true);
        t.setX(20.0f);
        t.setY(70.0f);
        t.setFill(Color.RED);
        t.setText("JavaFX drop shadow effect");
        t.setFont(Font.font("null", FontWeight.BOLD, 32));
        DropShadow ds1 = new DropShadow();
        ds1.setOffsetY(4.0f);
        ds1.setOffsetX(4.0f);
        ds1.setColor(Color.CORAL);

        Circle c = new Circle();
        c.setEffect(ds1);
        c.setCenterX(50.0f);
        c.setCenterY(325.0f);
        c.setRadius(30.0f);
        c.setFill(Color.RED);
        c.setCache(true);

        g.getChildren().add(t);
        g.getChildren().add(c);
        return g;
```

```
static Node innerShadow() {
        InnerShadow is = new InnerShadow();
        is.setOffsetX(2.0f);
        is.setOffsetY(2.0f);

        Text t = new Text();
        t.setEffect(is);
        t.setX(20);
        t.setY(100);
        t.setText("Inner Shadow");
        t.setFill(Color.RED);
        t.setFont(Font.font("null", FontWeight.BOLD, 80));

        t.setTranslateX(300);
        t.setTranslateY(300);

        return t;
    }
```

# Reflection Effect - odbicie na posadce

```
static Node reflection() {
        Text t = new Text();
        t.setX(10.0f);
        t.setY(50.0f);
        t.setCache(true);
        t.setText("Reflection in JavaFX...");
        t.setFill(Color.RED);
        t.setFont(Font.font("null", FontWeight.BOLD, 30));

        Reflection r = new Reflection();
        r.setFraction(0.9);

        t.setEffect(r);

        t.setTranslateY(400);
        return t;
    }
```

```
static Node lighting() {
        Distant light = new Distant();
        light.setAzimuth(-135.0f);

        Lighting l = new Lighting();
        l.setLight(light);
        l.setSurfaceScale(5.0f);

        Text t = new Text();
        t.setText("JavaFX"+"\n"+"Lighting!");
        t.setFill(Color.RED);
        t.setFont(Font.font("null", FontWeight.BOLD, 70));
        t.setX(10.0f);
        t.setY(10.0f);
        t.setTextOrigin(VPos.TOP);

        t.setEffect(l);

        t.setTranslateX(0);
        t.setTranslateY(320);

        return t;
    }
```