

Prezentačná vrstva, MVC v JavaFx

UINF/PAZ1c
2.epizóda



JavaFx™

Klasické textové používateľské rozhranie

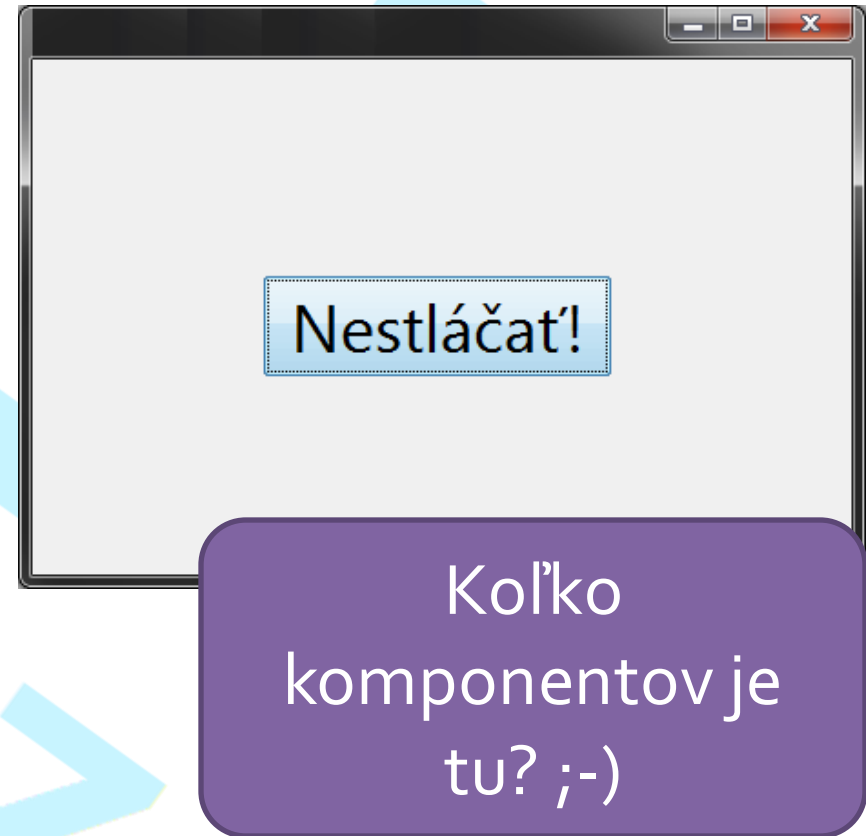
- údaje zadávané do programu postupne

```
Enter username: darthvader
New UNIX password: ****
Retype new UNIX password: ****
passwd: all authentication tokens updated successfully.
```

- program si **aktívne** pýta údaje
- možnosť zmeny predošlých údajov je často nemožná

Udalosťami riadené programovanie

- údaje zobrazované v **komponentoch**
 - oknách, ovládacích prvkoch, widgetoch...
- komponent **pasívne** čaká na interakciu



https://ais2.upjs.sk/ais/servlets/WebUIServlet?appClassName=ais.gui.vs.rh.VSRH009App&kodAplikacie=VSRH009&vie

RNDr. Peter Gurský, PhD.

Fakulta/Univerzita PF UPJŠ – Prírodovedecká fakulta

▼ Filter

☒ Akademický rok 2017/2018 Obdobie Zimný semester

☐ Od dátumu Do dátumu Akcie

☐ Na deň Deň Pravidelnosť Dni

Osoba Miestnosť Stredisko Predmet Študijný program Podprogram Rozvrhová akcia

 Zobrazíť rozvrh pre RNDr. Peter Gurský, PhD.

▼ Osoby

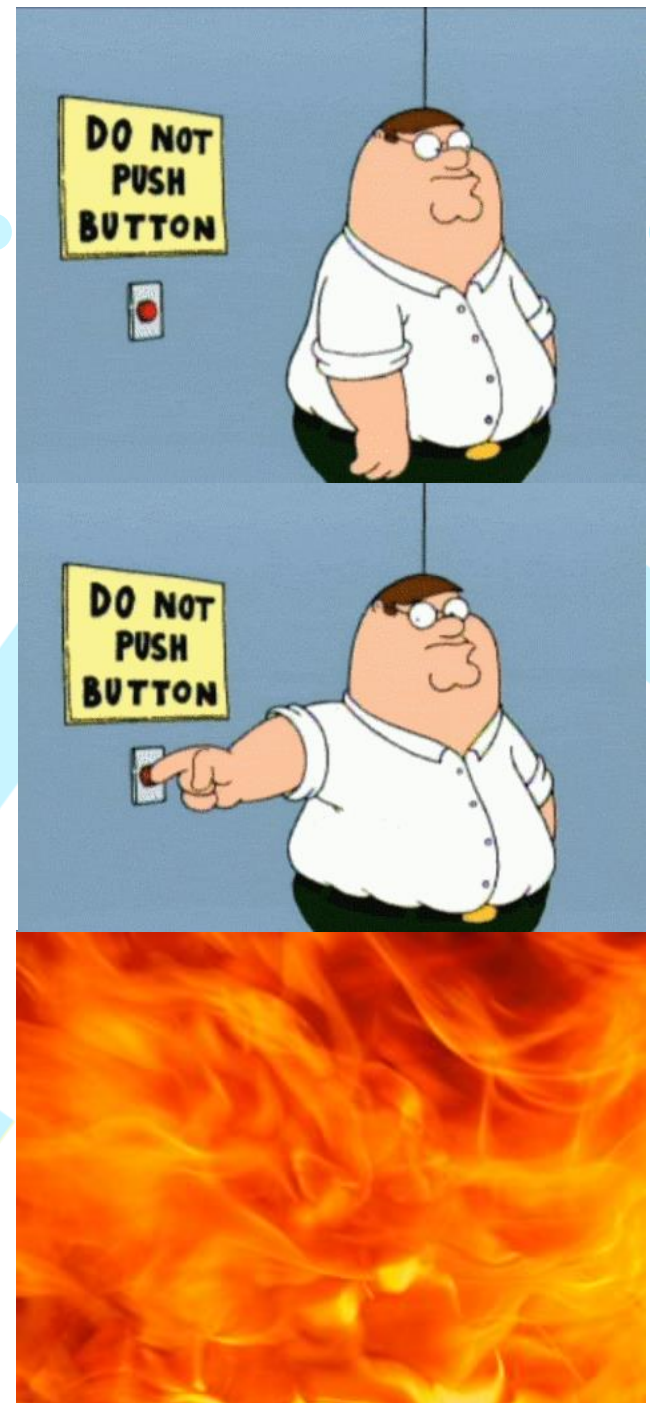
Meno	Priezvisko	Plné meno
Údaje neboli získané.		

/ 0 0

Koľko
komponentov je
tu? ;-)

Interakcia komponentov

- používateľ interakciou s komponentami generuje **udalosti**
- komponent vie **zareagovať** na vhodnú udalosť a vykonať príslušnú **akciu**



Udalosťami riadené programovanie

- naprogramujeme **metódy**, ktoré sa budú volať z komponentov pri spracovávaní udalostí
- rýchle programovanie, "nenáročné", ľahko pochopiteľné

Events		
actionPerformed	<none>	...
ancestorAdded	<none>	...
ancestorMoved	<none>	...
ancestorMoved	<none>	...
ancestorRemoved	<none>	...
ancestorResized	<none>	...
caretPositionChanged	<none>	...
componentAdded	<none>	...
componentHidden	<none>	...
componentMoved	<none>	...
componentRemoved	<none>	...

**EVENT DRIVEN
PROGRAMMING**
SINCE 1995

mouseExited	<none>	...
mouseMoved	<none>	...
mousePressed	<none>	...
mouseReleased	<none>	...
mouseWheelMoved	<none>	...
propertyChange	<none>	...
stateChanged	<none>	...

- knižnica pre **okienkové aplikácie v Java**
 - Súčasť SDK do Javy 10,
 - od Javy 11 iba ako externý projekt **OpenJFX.io**
 - Podporuje desktop, Android, iPhone
- kreslenie GUI: Scene Builder
- Ďalšie knižnice
 - AWT (pravek),
 - SWT (Eclipse),
 - Swing – naďalej podporovaný, vývoj zastal, kreslenie okienok v NetBeans, používa ho IntelliJ (so steroidami)
 - Compose Multiplatform – najnovší, Kotlin

JavaFx – inštalácia

- <https://openjfx.io/openjfx-docs/#maven>
 - upravujeme pom.xml:

```
<dependencies>
  <dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-controls</artifactId>
    <version>26.0.2</version>
  </dependency>
</dependencies>
<build>
<plugins>
  <plugin>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-maven-plugin</artifactId>
    <version>0.0.8</version>
    <configuration>
      <mainClass>sk.upjs.ics.EntranceApp</mainClass>
    </configuration>
  </plugin>
</plugins>
</build>
```

úplná cesta k triede s
main metódou

JavaFx – návrh okienok

- ručne
 - písanie čistého Java kódu
 - **žiadne bočné textové súbory**
- vizuálne
 - Scene builder poskytuje možnosť tvorby obsahu scény = rozloženie komponentov
 - Vygeneruje FXML súbor
 - **Umožňuje štýlovanie tagov FXML cez CSS**
 - rýchle naštartovanie projektu

Piliere JavaFx

- **javiská a scény**
- **komponenty (prvky scény)** = ovládanie prvky (control), kontajnery (layout), grafy, kreslenie,...
- **vlastnosti a sledovateľné premenné** = dáta pre komponenty
- **udalosti**
 - klik / pohyb myšou / výber položky / ... nad komponentmi

Filozofia komponentov

- Okno programu je javisko = javafx.stage.**Stage**
- Na javisko vkladáme scénu = javafx.scene.**Scene**
- Všetky **komponenty** sú súčasťou nejakej scény
- **Ovládacie prvky** (tlačidlá, zoznamy, ...) by mali bývať v nejakom **kontajneri** (**XxxPane a kamaráti**)
 - Pane, AnchorPane, BorderPane, FlowPane, GridPane,..
 - HBox, VBox, Textflow
 - Accordion, TabPane, ToolBar, ButtonBar,..
- Scéna má **práve jeden koreňový komponent** – nejaký kontajner
- Kontajnery sú ako matriošky
 - kontajner môže byť prvkom nadradeného kontajnera
 - nutnosť pre prispôsobivé okná

Najprv čistá Java bez FXML..

- jedno tlačidlo v kontajneri

```
public class EntranceApp extends Application {
```

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

```
        var button = new Button("stlač ma!");
```

```
        var rootPane = new AnchorPane();
```

```
        rootPane.getChildren().add(button);
```

```
        rootPane.setPrefSize(400, 300);
```

```
        var scene = new Scene(rootPane);
```

```
        stage.setTitle("Hello World");
```

```
        stage.setScene(scene);
```

```
        stage.show();
```

```
    }
```

```
    public static void main(String[] args) {
```

```
        launch(args); }}
```

} Štelujeme
komponenty

Vkladáme
koreňový
kontajner
do scény

Najprv čistá Java bez FXML..

- jedno tlačidlo v kontajneri

```
public class EntranceApp extends Application {  
  
    public void start(Stage stage) throws Exception {
```

spustíme cez

mvn clean javafx:run

alebo v IntelliJ IDEA

Run > Edit Configurations > + > Maven

a do "Run" napísať javafx:run

```
        stage.setScene(scene);
```

```
        stage.show();
```

```
    }
```

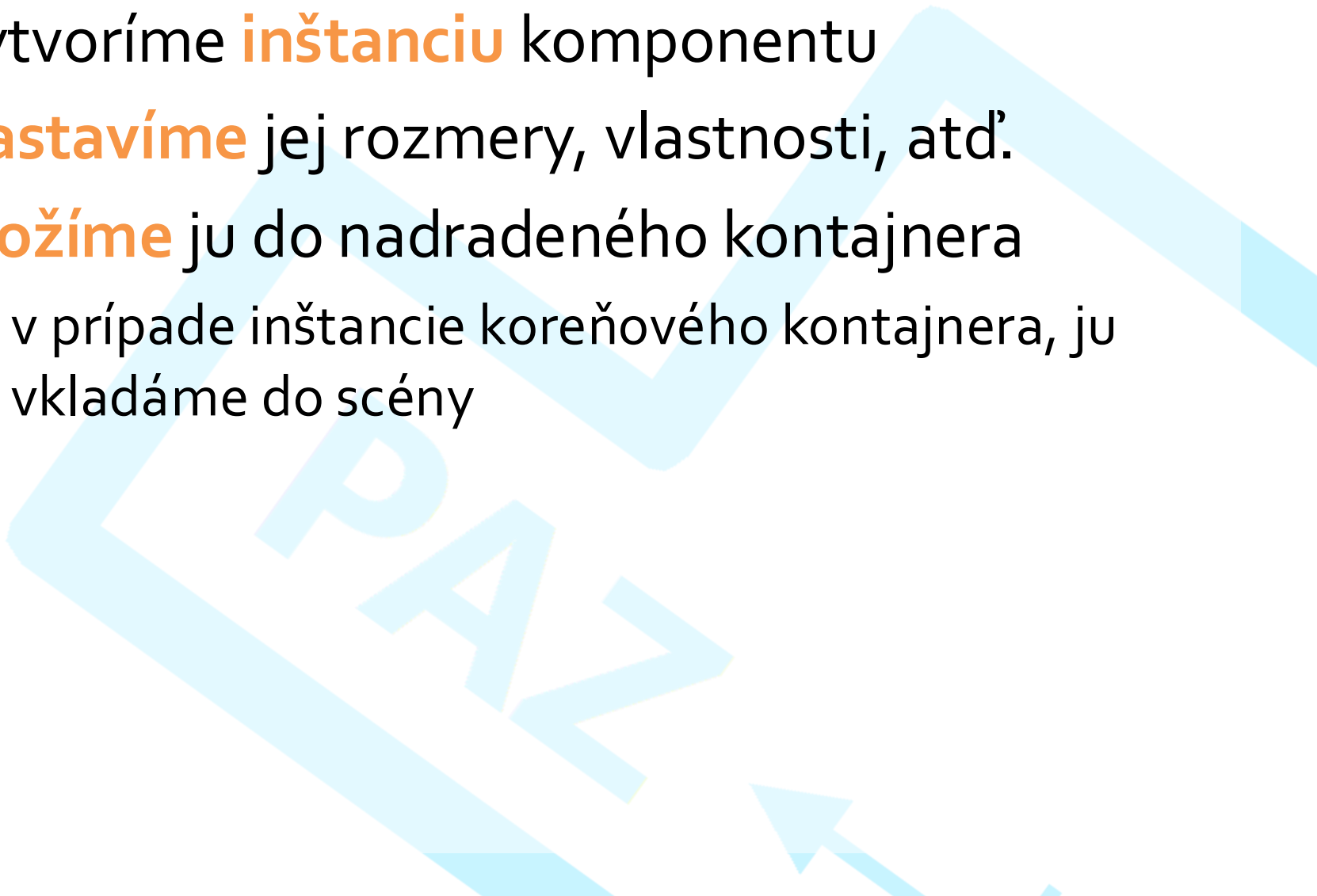
```
    public static void main(String[] args) {
```

```
        launch(args); }}
```

kontajner
do scény

Jednotný postup pre komponenty



- vytvoríme **inštanciu** komponentu
 - **nastavíme** jej rozmery, vlastnosti, atď.
 - **vložíme** ju do nadradeného kontajnera
 - v prípade inštancie koreňového kontajnera, ju vkladáme do scény
- 

Oživenie komponentov: udalosti

- komponent má **udalosti**, na ktoré vieme zareagovať
- príklad: tlačidlo **Button** podporuje udalosť „stala sa akcia“:
 - o udalosti vytvorí záznam akcie v objekte typu `ActionEvent`
- Udalosť vieme obslúžiť v objekte triedy ktorá implementuje interfejs **EventHandler**

```
interface EventHandler<ActionEvent> {  
    void handle(ActionEvent event);  
}
```

Udalosti v komponentoch

- komponent je oznamovač udalostí, ktoré sa mu stali
- ak niekto klikne na tlačidlo (udalosť **action**)
- tlačidlo pozrie či má poslucháča (**EventHandler**) pre udalosť **action**, ktorého to zaujíma
- tlačidlo oznámi poslucháčovi „*Vážený poslucháč, niekto na mne vyvolal udalosť **action**. Záznam o tejto udalosti je spísaný v objekte typu **ActionEvent***“
 - komponent poslucháčovi (**EventHandler**) spustí metódu `handle` a dodá objekt typu **ActionEvent**

Obsluha udalosti

```
public class SysouthHandler
    implements EventHandler<ActionEvent> {

    public void handle(ActionEvent e) {
        System.out.println("Klik!");
    }
}
```

- Ako docielim, aby poslucháč dostával informácie od gombíka?
- Inštanciu poslucháča **zaregistrujem** na gombíku

Registrácia poslucháča

```
public class EntranceApp extends Application {  
    public void start(Stage stage) throws Exception {  
        ...  
        button.setOnAction(new SysoutHandler());  
        ...  
    }  
}
```

```
public class SysouthHandler  
    implements EventHandler<ActionEvent> {  
    public void handle(ActionEvent e) {  
        System.out.println("klik!");  
    }  
}
```

```
button.setOnAction(new SysouthHandler());
```

ak 1300 komponentov, 1300 tried?

FAIL

The image features a top-down view of brown sugar cubes arranged to form the word "SUGAR" in a large, blocky font. The cubes are light brown and have a rough, crystalline texture. They are laid out on a plain, off-white surface. The word "SUGAR" is centered horizontally and occupies the middle portion of the frame. The cubes are arranged in a way that the letters are clearly legible, with some cubes overlapping to form the strokes of the letters.

anonymné vnútorné triedy!

```
public class EntranceApp extends Application {  
    public void start(Stage stage) throws Exception {  
        ...  
        button.setOnAction(new EventHandler<ActionEvent>() {  
            public void handle(ActionEvent event) {  
                System.out.println("klik!");  
            }  
        });  
        ...  
    }  
}
```

Anonymná vnútorná trieda
implementuje interface a
zároveň vytvoríme jej inštanciu

```
public class EntranceApp extends Application {  
    public void start(Stage stage) throws Exception {  
        ...  
        button.setOnAction((ActionEvent event) -> {  
            System.out.println("klik!");  
        });  
        ...  
    }  
}
```

...alebo cez lambdy

Keď sa nám nechce všetko písať v kóde...

```
public void start(Stage stage) throws Exception {  
    var button = new Button("stlač ma!");  
    button.setOnAction((ActionEvent event) -> {  
        System.out.println("klik!");  
    });  
    var rootPane = new AnchorPane();  
    rootPane.getChildren().add(button);  
    rootPane.setPrefSize(400, 300);  
    var scene = new Scene(rootPane);  
    stage.setTitle("Hello World");  
    stage.setScene(scene);  
    stage.show();  
}
```

Takéto veci kreslíme
v Scene Builderi
... a do Javy to dotiahneme
z FXML súboru

FXML balíček

- Ak chceme použiť FXML, dotiahneme do projektu knižnicu cez maven

```
<dependency>  
  <groupId>org.openjfx</groupId>  
  <artifactId>javafx-fxml</artifactId>  
  <version>26.0.2</version>  
</dependency>
```

Kde uložiť FXML v maven projekte?

- Java zdrojáky sú v `src/main/java/[balíček]`
 - Po skompilovaní odchádzajú .class súbory do adresára `target/classes/[balíček]`
- Nekompilované súbory (teda aj FXML, CSS,...) dávame do `src/main/resources/[balíček]`
 - Balíček rovnaký ako ten, v ktorom budú triedy, ktoré s daným FXML pracujú
 - Po vytvorení class súborov sa ešte všetko zo `src/main/resources/` **skopíruje** do `target/classes/`
 - Java virtuálny stroj (JVM) vidí .class súbory v rovnakom adresári ako .fxml súbory

Natahujeme FXML do javy

```
public void start(Stage stage) throws Exception {  
    Parent rootPane = FXMLLoader.load(  
        getClass().getResource("HelloWorld.fxml"));  
    // button.setOnAction((ActionEvent event) -> {  
    //     System.out.println("klik!");  
    // });  
    var scene = new Scene(rootPane);  
    stage.setTitle("Hello World");  
    stage.setScene(scene);  
    stage.show();  
}
```

Stratili sme
referenciu
na tlačidlo

Kontrolér - objekt čo má referencie na komponenty

- Komponenty musíme pomenovať – priradiť im jedinečné ***fx:id*** (v SB v sekcii code), ktoré bude v jave predstavovať názov premennej
 - Stane sa to, že element komponentu v FXML súbore obohatíme o atribút ***fx:id="názovPremennej"***
- Tiež vieme nastaviť názov metódy, ktorá sa zavolá pri stlačení tlačítla, vid' *On Action*
- Scene Builder nám vie vygenerovať kostru kontroléra
 - View > Show Sample Controller Skeleton

Prepojíme kontrolér s FXML

The screenshot shows the GlueXUI IDE interface. The main workspace displays an FXML document with a single button labeled "Vypísať Klik". The left sidebar contains a "Library" panel with various UI components like Containers, Controls (Button, CheckBox), GlueXUI, Menu, Miscellaneous, Shapes, Charts, and 3D. Below the library is a "Document" panel showing the "Hierarchy" and "Controller" sections. The "Controller" section shows the "Controller class" set to "sk.gursky.paz1c.HelloWorldController". A blue callout bubble points to this text with the text "Dopíšeme plný názov triedy". The right sidebar contains an "Inspector" panel showing the "Properties", "Layout", and "Code" for the selected "AnchorPane".

Untitled

File Edit View Insert Modify Arrange Preview Window Help

Library

Containers

Controls

Button

CheckBox

GlueXUI

Menu

Miscellaneous

Shapes

Charts

3D

Document

Hierarchy

Controller

Controller class

sk.gursky.paz1c.HelloWorldController

Use fx:root construct

Assigned fx:id

fx:id	Component
vypisatKlikButton	Button

AnchorPane

Inspector

Properties : AnchorPane

Layout : AnchorPane

Code : AnchorPane

Identity

fx:id

DragDrop

On Drag Detected

On Drag Done

On Drag Exited

On Drag Over

Dopíšeme plný názov triedy

V kontroléri máme inject-nuté referencie na komponenty

```
public class HelloWorldController
```

```
@FXML
```

```
private Button stlacMaBtn;
```

```
@FXML
```

```
void stlacMaOnClick(ActionEvent event) {  
    System.out.println("KLIKOL SI");  
}
```

```
}
```

Inštanciu tlačidla vyrobí aplikácia, našteluje ju podľa FXML súboru a referenciu uloží tu

Táto metóda sa zavolá pri stlačení tlačidla. Všímnite si ju v FXML súbore

(Voliteľne) Keď si chceme kontrolér vyrobiť sami, prepojíme ho s komponentmi v kóde:

```
public void start(Stage stage) throws Exception {  
    var mainController = new HelloWorldController();  
    var fxmlLoader = new FXMLLoader(  
        getClass().getResource("HelloWorld.fxml"));  
    fxmlLoader.setController(mainController);  
    Parent rootPane = fxmlLoader.load();  
  
    var scene = new Scene(rootPane);  
    stage.setTitle("Hello World");  
    stage.setScene(scene);  
    stage.show();  
}
```

Otázky?

