

# Záver semestra

dnes, 11. JFX

dnes, 11:25, projekty

5.5., 12. JFX + 3D (Lukáš)

12.5., Java Reflection Model, záver

na cvičení 13.5. !9:20! Quadterm2

Termíny skúšok sú v AISe, vždy streda 9:00


# JavaFX 3D

- Youtube, 7 videí, Lukáš Gajdošech  
<https://www.youtube.com/playlist?list=PLUtV5iyaCT5GKtStZiVfGb6JbN0cmQ0gJ>
- Zdrojáky (github) alebo <https://drive.google.com/file/d/1KNwE...>
- Trvanie: cca 60 min.
- a je to súčasťou prednášky, aj keď forma je mierne iná
- bude k tomu cvičenie, aj posledná DÚ

# JavaFx



## Dnes bude:

- základné komponenty JavaFx (Node a podtriedy)
- štýly (.css) a Scene Builder (.fxml)
- spracovanie udalostí (myš, klávesnica, ...)
- spôsoby návrhu jednoduchej (pravouhlej) hry
- škálovateľnosť aplikácie 

## Zdroj a literatúra:

- [What Is JavaFX](#)
- [JavaFX 2.0: Introduction by Example](#)
- [Introduction to Java Programming, !!!!Tenth Edition](#)

## Cvičenia:

- jednoduchá pravouhla aplikácia s interakciou:
- maľovátka, euro-kalkulačka,
- logické (pravouhle) hry: pexeso, piškvorky, ...

# JavaFx aplikácia

```
public class Main extends Application {
```

```
    @Override
```

```
    public void start(Stage primaryStage) {
```

```
        try {
```

```
            Button btn = new Button("Press me !");
```

```
            Pane root = new Pane(btn);
```

```
            Scene scene = new Scene(root, 400, 400, Color.ORANGE);
```

```
            scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
```

```
            primaryStage.setScene(scene);
```

```
            primaryStage.show();
```

```
        } catch (Exception e) {
```

```
            e.printStackTrace();
```

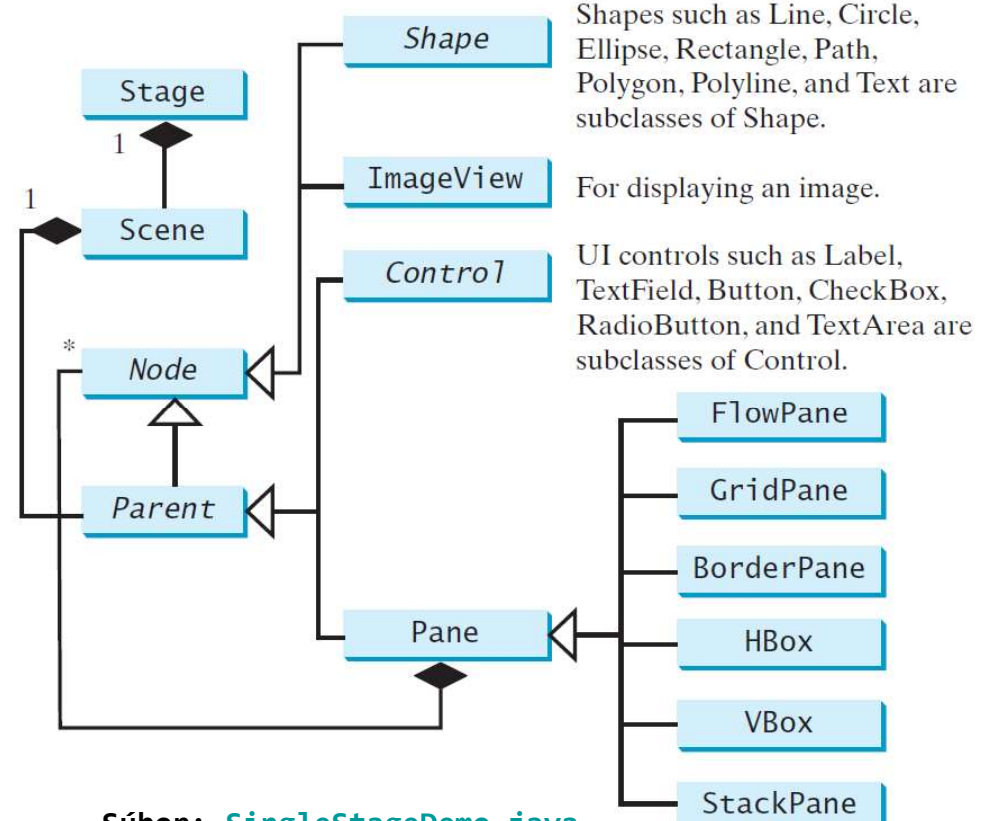
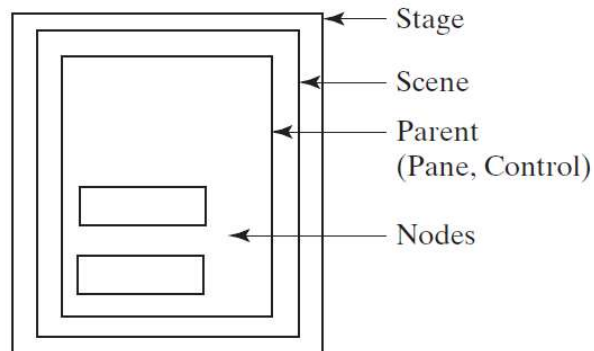
```
        }
```

```
    }
```

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

```
        Launch(args);
```

```
    } }
```



Source: [SingleStageDemo.java](#)

# Node a štýl

**Hint:** Ak zobrazujete komponent, a nevidíte ho, dajte mu **žlté** pozadie. Ak ho nevidíte ani potom, asi má zle rozmery.

Ako ho zafarbiť:

```
Pane root = new Pane(btn);
```

- 1.možnosť

```
root.setStyle("-fx-background-color: #" + "FFFF00");
```

- 2.možnosť (javaafx...Color nemá int getRGB(), na rozdiel od java.awt.Color)

```
Color c = Color.YELLOW;
```

```
root.setStyle("-fx-background-color: #" +  
    Integer.toHexString((int)(256*256*255*c.getRed() +  
                                (255<<8)*c.getGreen() +  
                                (1<<8-1)*c.getBlue() )) );
```

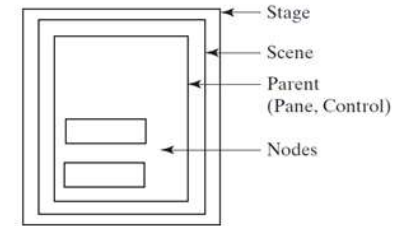
- 3.možnosť

```
root.setBackground(  
    new Background(  
        new BackgroundFill(Color.YELLOW, null, null)  
    ));  
new Color(1,0.60,0.90, 1)
```

Súbor: [SingleStageDemo.java](#)



# Scene



Scéna predstavuje vrchný element stromovej štruktúry elementov typu Node, resp. Parent

Má konštruktory:

- **Scene**(Parent root)  
`new Scene(root);`
- **Scene**(Parent root, double width, double height)  
`new Scene(root, 400, 400);`
- **Scene**(Parent root, double w, double h, Paint fill)  
`new Scene(root, 200, 200, Color.BLUE);`

Parent má deti typu Node, presnejšie poskytuje metódu `ObservableList<Node> getChildren()`

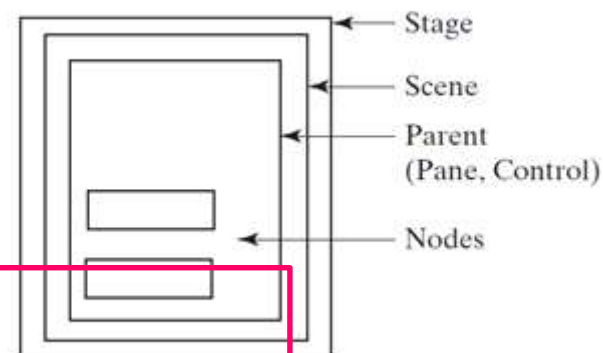
```
root.getChildren().clear()
root.getChildren().add(node)
root.getChildren().addAll(node1, node2, ...)
```

## Parent

- Control,
- Group,
- Region,
  - Axis,
  - Chart,
- Pane
  - BorderPane,
  - FlowPane,
  - GridPane,
  - HBox,
  - StackPane,
  - Vbox,

• ...

# Štruktúra tried



## Node

- [Canvas](#),
- [ImageView](#),
- [Parent](#),
- [MediaView](#),
- [Shape](#)
  - [Circle](#),
  - [Ellipse](#),
  - [Line](#),
  - [Polygon](#),
  - [Polyline](#),
  - [Rectangle](#),
  - [Text](#)
  - ...

## Control

- [ChoiceBox](#),
- [ComboBoxBase](#)
  - [ComboBox](#)
- [Labeled](#)
  - [ButtonBase](#)
    - [Button](#),
    - [CheckBox](#),
    - [ToggleButton](#)
  - [Label](#),
- [ListView](#),
- [TextInputControl](#)
  - [TextArea](#),
  - [TextField](#)

# BorderPane

```
class MyButton extends Button {  
    public MyButton(String text) {  
        super(text);  
        setMaxWidth(Double.MAX_VALUE);  
        setMaxHeight(Double.MAX_VALUE);  
        setStyle("-fx-border-color: blue;  
                 -fx-font: 24px 'Arial'");  
    }  
}
```

```
BorderPane root = new BorderPane();  
root.setTop(new MyButton("Som na vrchu"));  
root.setBottom(new MyButton("I'm in the bottom"));  
root.setRight(new MyButton("Ich bin recht"));  
root.setLeft(new MyButton("Я в левом"));  
root.setCenter(new MyButton("Je suis au milieu"));  
Scene scene = new Scene(root, 600, 400);  
primaryStage.setScene(scene);  
primaryStage.setTitle("BorderPane");  
primaryStage.show();
```





# FlowPane, GridPane

```
FlowPane root = new FlowPane(  
    new MyButton("Som prvý"), new MyButton("Som druhý"), new MyButton("Som tretí"));  
Scene scene = new Scene(root, 300, 400);  
Stage newStage = new Stage();  
newStage.setScene(scene);  
newStage.setTitle("FlowPane");  
newStage.show();
```



```
GridPane root = new GridPane();  
for (int i = 0; i < 5; i++)  
    for (int j = 0; j < 5; j++)  
        root.add(new MyButton(i + "x" + j), i, j);  
root.setHgap(10);  
root.setVgap(10);  
Scene scene = new Scene(root, 400, 400);  
Stage newStage = new Stage();  
newStage.setScene(scene);  
newStage.setTitle("GridPane");  
newStage.show(); }
```



# HBox, VBox, StackPane

```
HBox root = new HBox(  
    new MyButton("Som prvý"), new MyButton("Som druhý"), new MyButton("Som tretí"));
```

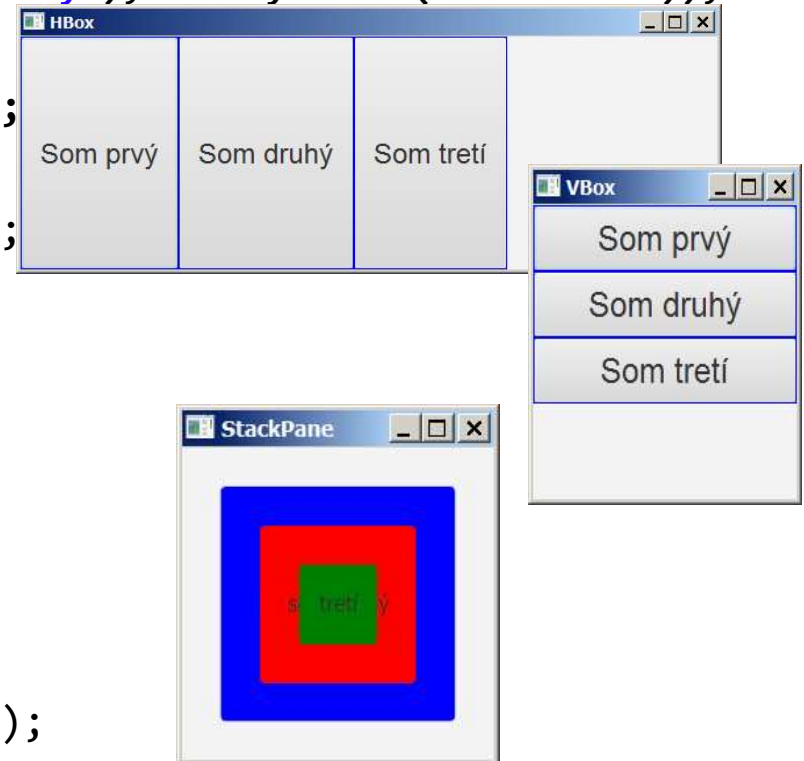
```
VBox root = new VBox(  
    new MyButton("Som prvý"), new MyButton("Som druhý"), new MyButton("Som tretí"));
```

```
Button btn1 = new Button("naozaj som prvý");  
btn1.setPrefSize(150,150);  
btn1.setStyle("-fx-background-color: blue");
```

```
Button btn2 = new Button("som druhý");  
btn2.setPrefSize(100,100);  
btn2.setStyle("-fx-background-color: red");
```

```
Button btn3 = new Button("tretí");  
btn3.setPrefSize(50,50);  
btn3.setStyle("-fx-background-color: green");
```

```
StackPane root = new StackPane(btn1, btn2, btn3);
```



# EventHandler

```
class MyButton extends Button {
    setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event) {
            System.out.println("stlačil si " + text);
        }
    });
    setOnAction(event -> {
        System.out.println("stlačil si " + text);
    });
    setOnMouseClicked(event -> {
        System.out.println("klikol si " + text + ", " +
            event.getX() + ", " + event.getY());
    });
    setOnKeyPressed(event -> {
        System.out.println("stlačil si " + text + ", " +
            event.getCode());
    });
}
```

# Malá kalkulačka

The screenshot shows a window titled "Hypotéka" with a light gray background. It contains five text input fields arranged vertically, each with a label to its left. The labels and their corresponding values are: "Úrok [%]:" with "5.5", "Délka [roky]:" with "20", "Suma:" with "100000", "Mesačně:" with "687,89", and "Spolu:" with "165092,95". Below these fields is a button labeled "Vyhodnot".

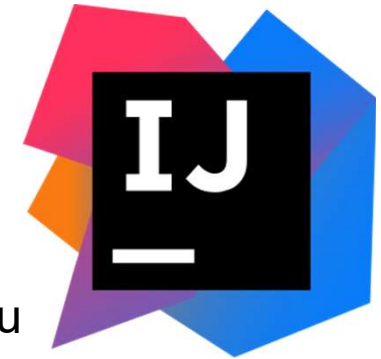
```
public class Hypoteka extends Application {
    TextField tfUrokovaMiera = new TextField(),
               tfPocetRokov = new TextField(),
               tfSuma = new TextField(),
               tfMesacneSplatky = new TextField(),
               tfSpolu = new TextField();
    Button btVypocet = new Button("Vyhodnot");
    GridPane gridPane = new GridPane();
    gridPane.setHgap(5);
    gridPane.setVgap(5);
    gridPane.add(new Label("Úrok [%]:"),0,0);
    gridPane.add(tfUrokovaMiera, 1, 0);
    gridPane.add(new Label("Délka [roky]:"),0,1);
    gridPane.add(tfPocetRokov, 1, 1);
    gridPane.add(new Label("Suma:"),0,2);
    gridPane.add(tfSuma, 1, 2);
    gridPane.add(new Label("Mesačně:"),0,3);
    gridPane.add(tfMesacneSplatky, 1,3);
    gridPane.add(new Label("Spolu:"),0,4);
    gridPane.add(tfSpolu, 1, 4);
    gridPane.add(btVypocet, 1, 5);

    tfUrokovaMiera.setAlignment(Pos.BOTTOM_RIGHT);
    tfPocetRokov.setAlignment(Pos.BOTTOM_RIGHT);
    tfSuma.setAlignment(Pos.BOTTOM_RIGHT);
    tfMesacneSplatky.setAlignment(Pos.BOTTOM_RIGHT);
    tfSpolu.setAlignment(Pos.BOTTOM_RIGHT);
    tfMesacneSplatky.setEditable(false);
    tfSpolu.setEditable(false);

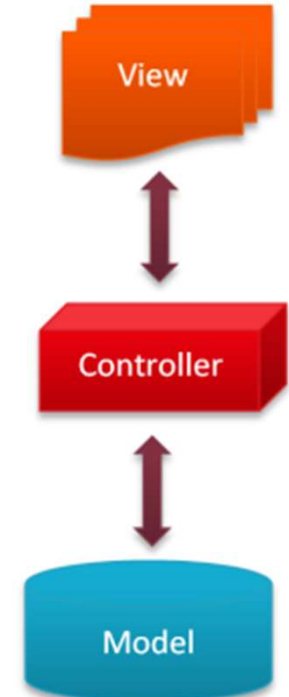
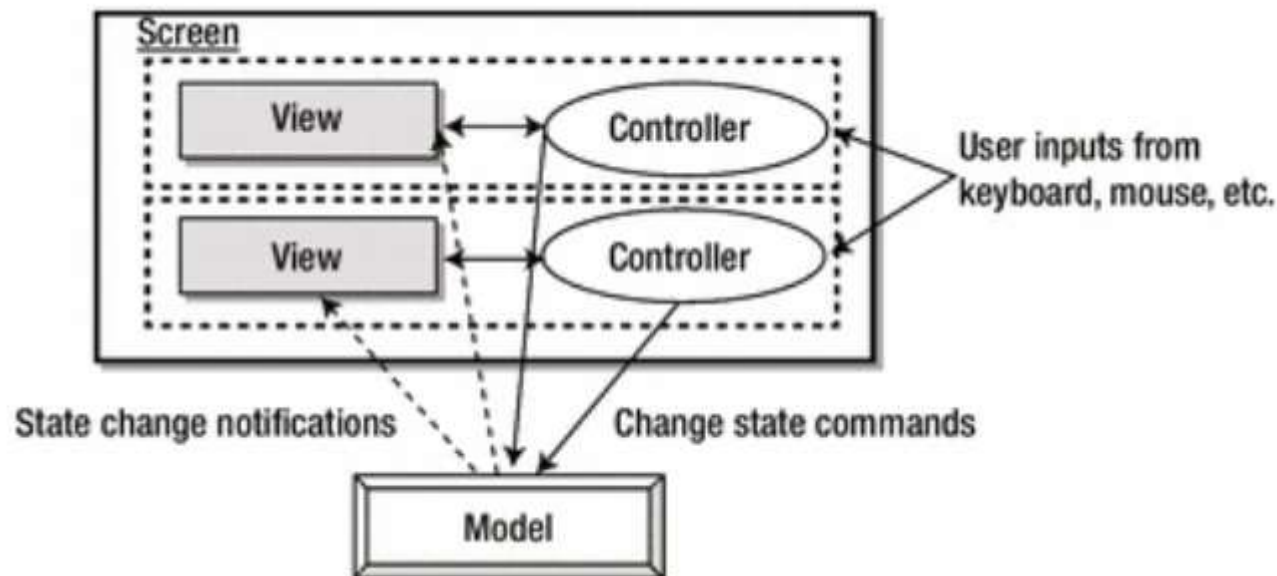
    btVypocet.setOnAction(e -> {
        rocnyUrok = Double.parseDouble(
            tfUrokovaMiera.getText());
        pocetRokov = Integer.parseInt(
            tfPocetRokov.getText());
        suma = Double.parseDouble(
            tfSuma.getText());
        tfMesacneSplatky.setText(
            String.format("%.2f",
                mesacneSplatky()));
        tfSpolu.setText(
            String.format("%.2f",
                getTotalPayment()));
    });
}
```

Súbor: [Hypoteka.java](#)

# IDE IntelliJ

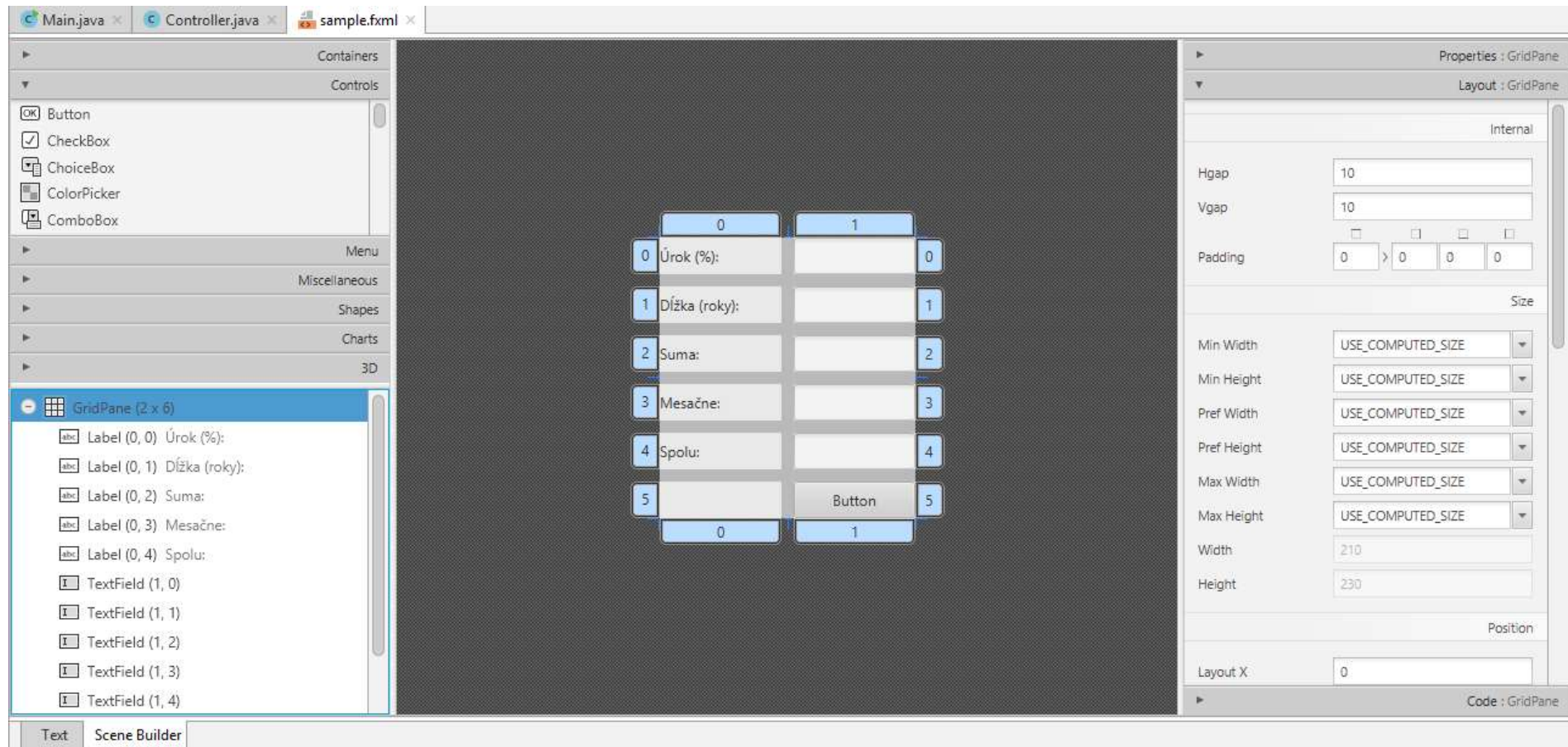


- ďalšia možnosť ako vytvoriť stand-alone/webstart/web aplikáciu
- podporuje tvorbu JavaFX aplikácií (New/JavaFX/JavaFX Application)
- aj FXML aplikácií (New/JavaFX/JavaFX FXML Application)
- podporuje Model-View-Controller (MVC) prostredníctvom FXML
  - View: prezentačná úroveň, vizuálne komponenty (Controls), layouts (Containers), útvary (Shapes), štýly...
  - Controller: logika, väzba medzi View a Modelom,
  - Model: data, reprezentácia



# FXML

## SceneBuilder – IntelliJ





# FXML

SceneBuilder – IntelliJ, Eclipse

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<GridPane alignment="center" hgap="10" vgap="10"
  xmlns="http://javafx.com/javafx/8.0.121"
  xmlns:fx="http://javafx.com/fxml/1"
  fx:controller="sample.Controller">
  <children>
    <Label text="Úrok (%):" />
    <Label text="Dĺžka (roky):" GridPane.rowIndex="1" />
    <Label text="Suma:" GridPane.rowIndex="2" />
    <Label text="Mesačne:" GridPane.rowIndex="3" />
    <Label text="Spolu:" GridPane.rowIndex="4" />
    <TextField fx:id="tfUrokovaMiera" GridPane.columnIndex="1" />
    <TextField fx:id="tfPocetRokov" GridPane.columnIndex="1" GridPane.rowIndex="1" />
    <TextField fx:id="tfSuma" GridPane.columnIndex="1" GridPane.rowIndex="2" />
    <TextField fx:id="tfMesacneSplatky" GridPane.columnIndex="1" GridPane.rowIndex="3" />
    <TextField fx:id="tfSpolu" GridPane.columnIndex="1" GridPane.rowIndex="4" />
    <Button fx:id="btVypocet" onAction="#klikolSiNaVypocitaj" text="Vypočítaj"
      GridPane.columnIndex="1" GridPane.rowIndex="5" />
  </children>
</GridPane>
```

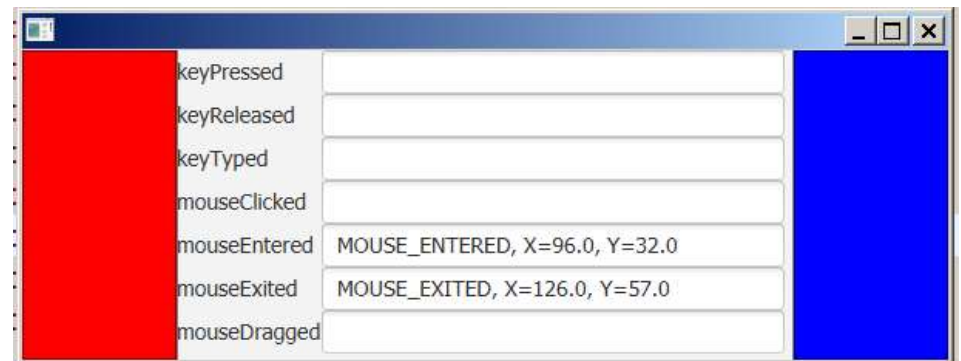
# Controller

```
public class Controller {  
    @FXML  
    private TextField tfUrokovaMiera;  
    @FXML  
    private TextField tfPocetRokov;  
    @FXML  
    private TextField tfSuma;  
    @FXML  
    private TextField tfMesacneSplatky;  
    @FXML  
    private TextField tfSpolu;  
    @FXML  
    private Button btVypocet;  
  
    public void klikolSiNaVypocitaj(ActionEvent event) {  
        double rocnyUrok = Double.parseDouble(tfUrokovaMiera.getText());  
        double pocetRokov = Integer.parseInt(tfPocetRokov.getText());  
        double suma = Double.parseDouble(tfSuma.getText());  
        double mesacnyUrok = rocnyUrok/12/100;  
        double mesacneSplatky =  
            suma*mesacnyUrok/(1-(1/Math.pow(1+mesacnyUrok, pocetRokov*12)));  
        tfMesacneSplatky.setText(String.format("%.2f", mesacneSplatky));  
        double getTotalPayment = mesacneSplatky * pocetRokov * 12;  
        tfSpolu.setText(String.format("%.2f", getTotalPayment));  
    }  
}
```



# MouseEvent, KeyEvent

```
Hashtable<String, Node> h = new Hashtable<String, Node>();
String[] event = { "keyPressed", "keyReleased", "keyTyped",
    "mouseClicked", "mouseEntered", "mouseExited", "mouseDragged"};
SmallPane bluePane = new SmallPane(this, Color.BLUE),
    redPane = new SmallPane(this, Color.RED);
GridPane gp = new GridPane();
for (int i = 0; i < event.length; i++) {
    TextField t = new TextField();
    t.setPrefWidth(300); t.setEditable(false);
    gp.add(new Label(event[i]), 0, i);
    gp.add(t, 1, i);
    h.put(event[i], t);
}
BorderPane bp = new BorderPane();
bp.setCenter(gp);
bp.setRight(bluePane);
bp.setLeft(redPane);
Scene scene = new Scene(bp, 600, 200);
```



# Pokračovanie

```
class SmallPane extends Pane {
    SmallPane(AutoEvent parent, Color color) {
        this.parent = parent;
        this.color = color;
        setPrefWidth(100);
        setFocusTraversable(true);
        setOnKeyPressed(event -> {
            TextField t = (TextField) parent.h.get("keyPressed");
            t.setText(event.getEventType() + ", keyCode="+ event.getCode());
            paint();
            event.consume();
        });
        setOnMouseClicked(event -> {
            TextField t = (TextField) parent.h.get("mouseClicked");
            t.setText(event.getEventType() + ", X="+ event.getX() + ", Y="+ event.getY());
            paint();
            event.consume();
        });
    }
}
```

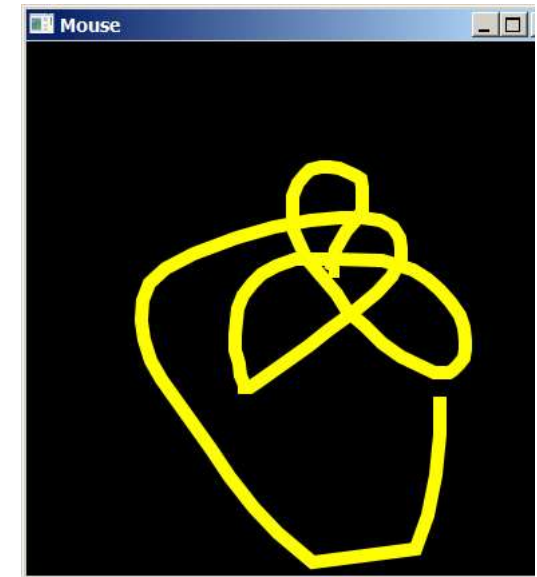
## Event

- [ActionEvent](#),
- [InputEvent](#),
  - [DragEvent](#)
  - [KeyEvent](#),
  - [MouseEvent](#),
  - [TouchEvent](#)
  - ...
- [WindowEvent](#),
- ...

# Polyline, Polygon

```
MousePane p = new MousePane();
Scene scene = new Scene(p, 400, 400, Color.BLACK);
scene.setOnMouseMoved(event -> {
    if (listOfPositions.size() >= 200) {
        listOfPositions.removeElementAt(0);
        listOfPositions.removeElementAt(1);
    }
    listOfPositions.addElement(
        event.getX());
    listOfPositions.addElement(
        event.getY());
    p.paint();
    event.consume();
} );
```

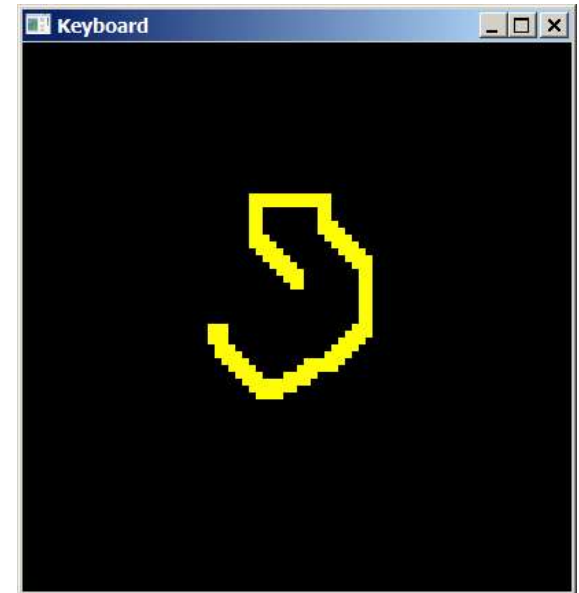
```
class MousePane extends Pane {
    public void paint() {
        getChildren().clear();
        Double[] d =
            listOfPositions.toArray(new Double[]{});
        Polyline p1 = new Polyline();
        p1.setStroke(Color.YELLOW);
        p1.setStrokeWidth(10);
        p1.getPoints().addAll(d);
        getChildren().add(p1);
    }
}
```



Súbor: [MouseDemo.java](#)

# Pomocou šípiek

```
MousePane p = new MousePane();
Scene scene = new Scene(p, 400, 400, Color.BLACK);
scene.setOnKeyPressed(event -> {
    if (listOfPositions.size() >= 200) {
        listOfPositions.removeElementAt(0);
        listOfPositions.removeElementAt(0);
    }
    if (event.getCode() == KeyCode.UP) y -= 5;
    if (event.getCode() == KeyCode.DOWN) y += 5;
    if (event.getCode() == KeyCode.LEFT) x -= 5;
    if (event.getCode() == KeyCode.RIGHT) x += 5;
    listOfPositions.addElement(x);
    listOfPositions.addElement(y);
    p.paint();
    event.consume();
} );
```



# Canvas

```
public void paintCanvas() {
    GraphicsContext gc = getGraphicsContext2D();// kreslenie do canvasu
    gc.clearRect(0, 0, sizeX, sizeY);
    gc.setFill(Color.gray(0.2));
    gc.fillOval(centerX - scale * moloSize, centerY - scale * moloSize,
                scale * 2 * moloSize, scale * 2 * moloSize);
    if (namornik.alive) { // ak sa este neutopil, nakresli obrazok namornika
        gc.drawImage(new Image("namornik.gif"), // namornik.img,
                    namornik.getXPixel(false),
                    namornik.getYPixel(false));
    } else { // ak je utopeny, nakresli vlny zobraz v strede vln pocet krokov
        gc.setStroke(Color.RED);
        gc.strokeText(Integer.toString(namornik.steps),
                    namornik.getXPixel(true) - 8,
                    namornik.getYPixel(true) + 7);
    }
}
```

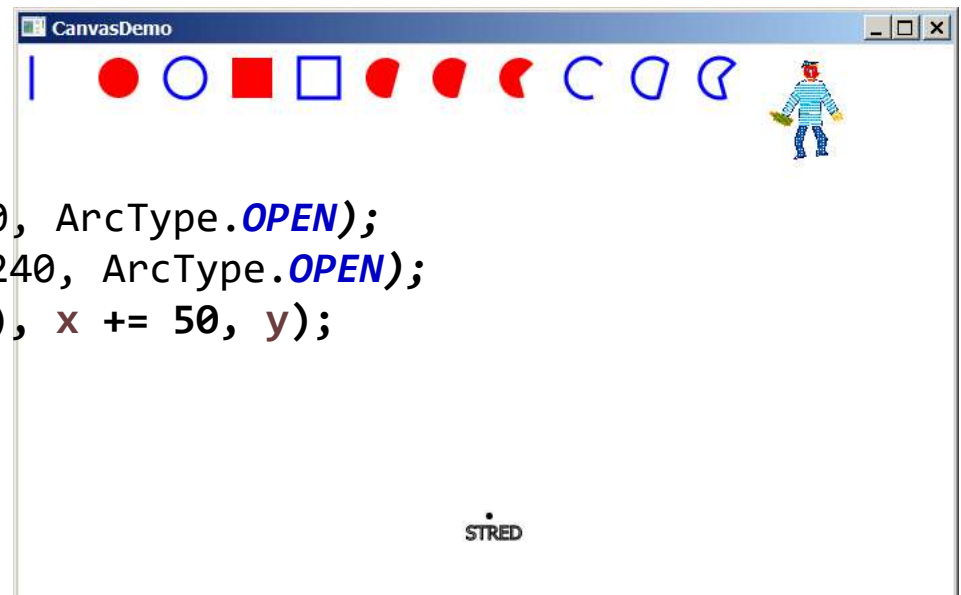


# Kreslenie do Canvas

```
Canvas canvas = new Canvas(700, 700);
GraphicsContext gc = canvas.getGraphicsContext2D();
gc.fillOval(350, 350, 5, 5);
gc.strokeText("STRED", 335, 370);

.gc.setFill(Color.RED);
.gc.setStroke(Color.BLUE);
.gc.setLineWidth(3);
.gc.strokeLine(x, y, x, y + 30);
.gc.fillOval(x += 50, y, 30, 30);
.gc.strokeOval(x += 50, y, 30, 30);
.gc.fillRect(x += 50, y, 30, 30);
.gc.strokeRect(x += 50, y, 30, 30);

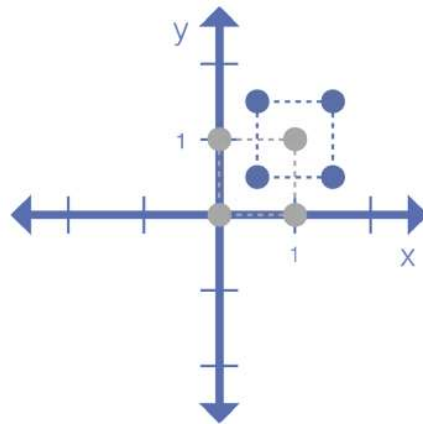
.gc.fillArc(x += 50, y, 30, 30, 45, 240, ArcType.OPEN);
.gc.strokeArc(x += 50, y, 30, 30, 45, 240, ArcType.OPEN);
.gc.drawImage(new Image("namornik.gif"), x += 50, y);
```



# Afinné zobrazenia

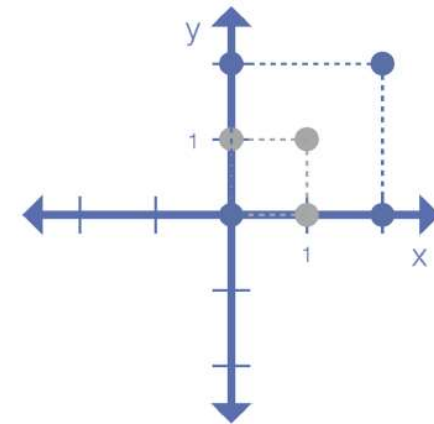
**Translate**

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$



**Scale**

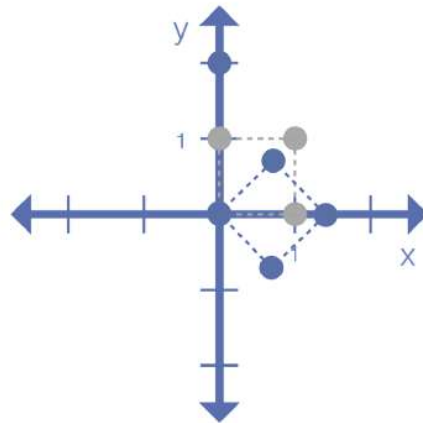
$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



**Rotate**

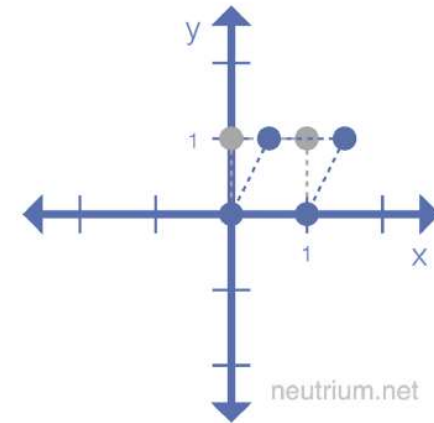
$$\begin{bmatrix} c & s & 0 \\ -s & c & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$c = s = \sin(45^\circ)$$

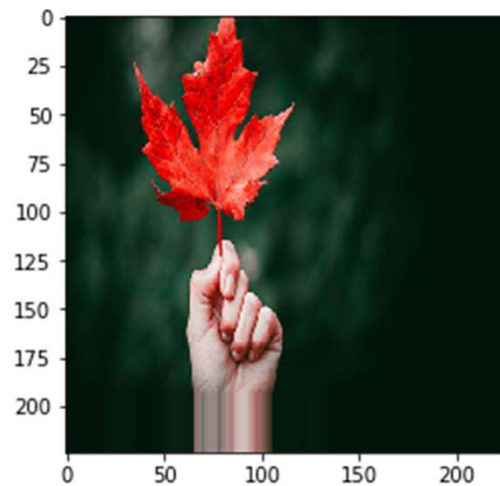
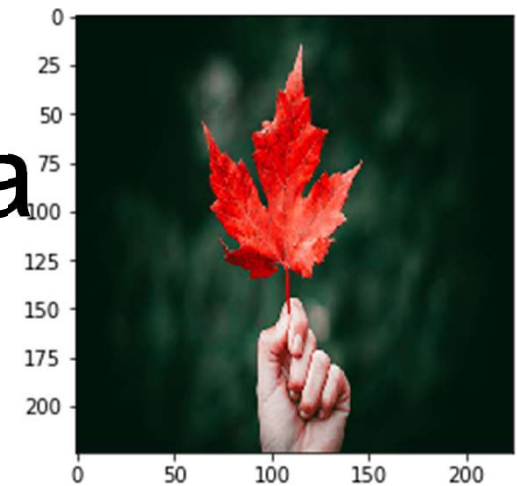


**Shear**

$$\begin{bmatrix} 1 & 0.5 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$



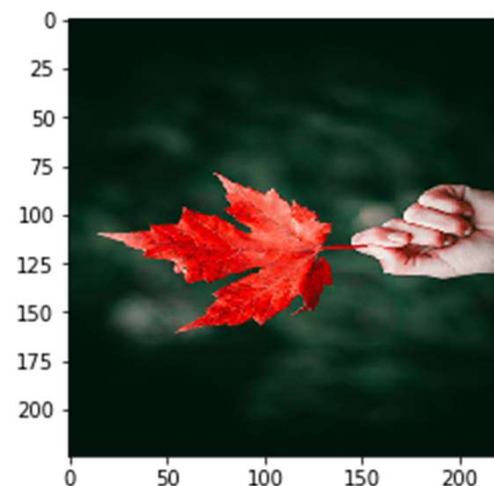
# Afinné zobrazenia



$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

$t_x$  specifies the displacement along the  $x$  axis.

$t_y$  specifies the displacement along the  $y$  axis.

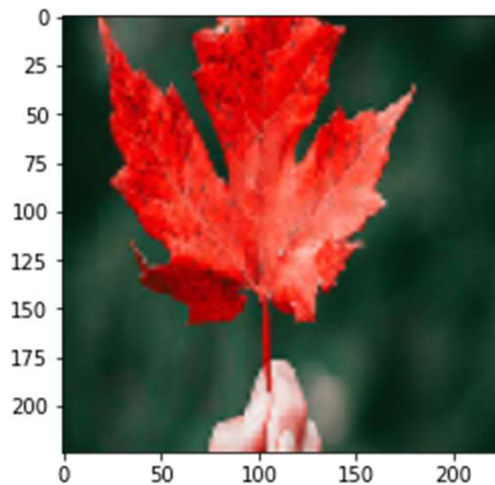
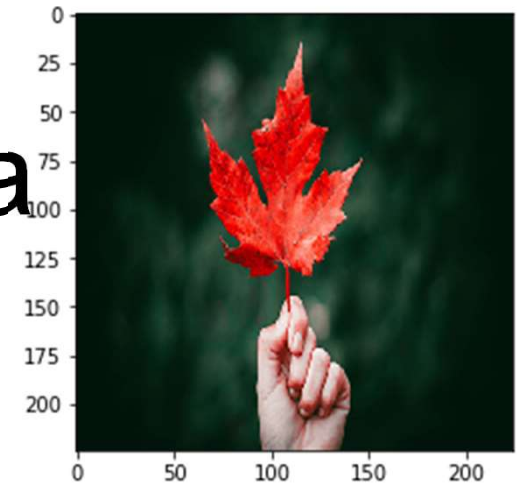


$$\begin{bmatrix} \cos(q) & \sin(q) & 0 \\ -\sin(q) & \cos(q) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$q$  specifies the angle of rotation.

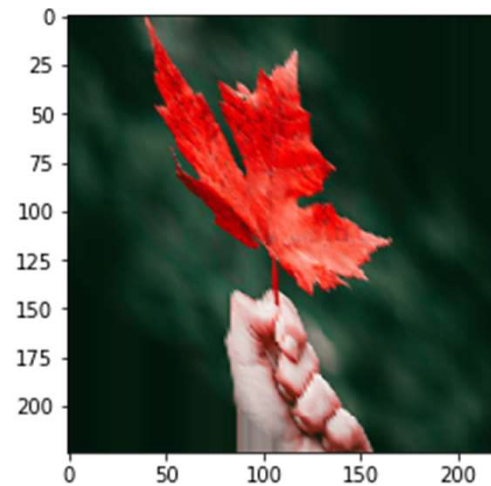


# Afinné zobrazenia



$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$s_x$  specifies the scale factor along the  $x$  axis  
 $s_y$  specifies the scale factor along the  $y$  axis.



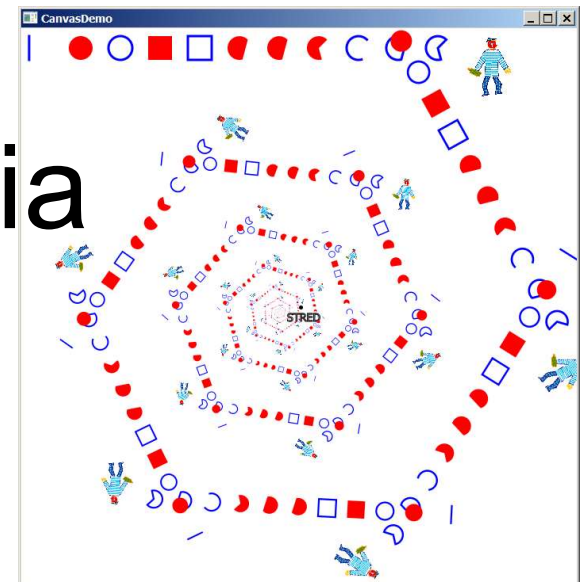
$$\begin{bmatrix} 1 & sh_y & 0 \\ sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$sh_x$  specifies the shear factor along the  $x$  axis  
 $sh_y$  specifies the shear factor along the  $y$  axis.

# Afinné zobrazenia

Z lineárnej algebry:

- otočenie o uhol  $\alpha$  okolo stredu  $x,y$ ,
- posunutie  $dx,dy$
- rovnoľahlosť/natiahnutie  $kx,ky$  podľa stredu  $x,y$



```
Affine af = new Affine();
```

// afinné zobrazenie

```
for (int i = 0; i < 100; i++) {
```

```
    af.append(Affine.scale(0.9, 0.9, 350, 350)); // rovnoľahlosť
```

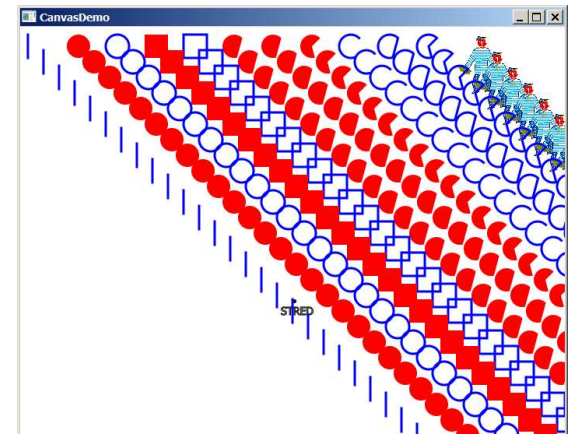
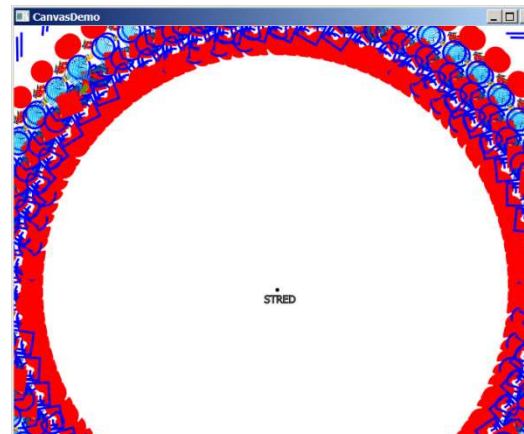
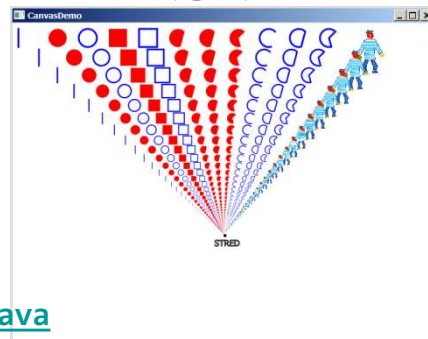
```
    af.append(Affine.rotate(60, 350, 350)); // otočenie
```

```
    af.append(Affine.translate(20, 20)); // posunutie
```

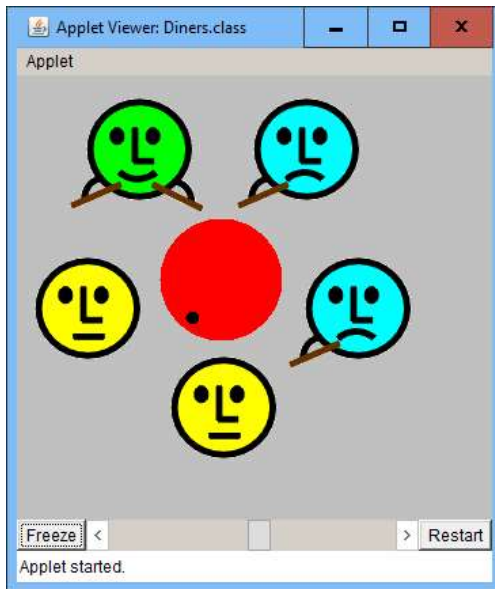
```
    gc.setTransform(af);
```

```
    paintShapes(gc);
```

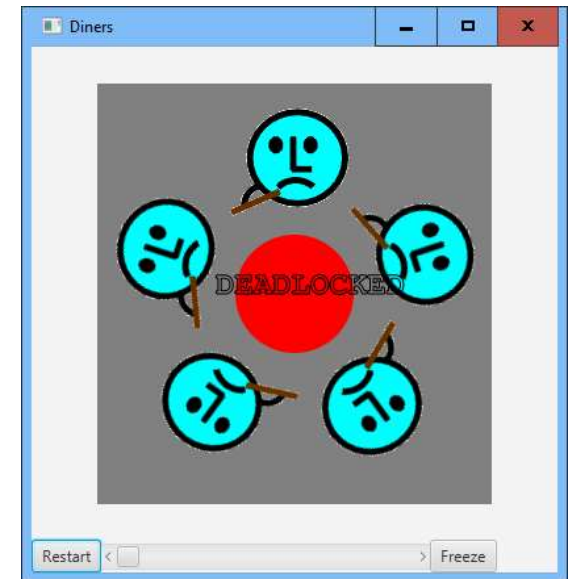
```
}
```



# Večerajúci ešte raz



- niektorí sedia chrbtom k stolu
- ktorá je ľavá a pravá vidlička ?
- synchronizácia cez vidličku nefunguje ?
- čo je Applet ?
- kde je deadlock ?



- otočíme je ich  
`philPlace[i] = new Affine();`
  - ✓ `Affine.rotate(i*360/5, w/2, h/2);`
  - ✓ `Affine.translate(0, -radius);`
  - ✓ `Affine.translate(-philW/2, -philH/2);`
- i-ty filozóf má vidličku  $(i-1)\%5$  a  $i$ 
  - ✓ *boli vymenené ☺*
- ✓ synchronizácia cez vidličku funguje !
- ✓ prerobené do javafx
- ✓ kde je deadlock ?

# JavaFx

## pokračovanie

Už vieme (quadterm2):

- kresliť do Canvas, vložiť Canvas->Pane->Scene->Stage,
- simulovať (Thread+Platform.runLater, Timeline, AnimationTimer) ,
- chytať ActionEvent, KeyEvent a MouseEvent,
- a že uhol dopadu sa rovná uhlu odrazu ☺



Dnes:

- rôzne spôsoby návrhu jednoduchšej (pravouhlej) hry,
- aspekt škálovateľnosti,
- perzistencia,
- príklady ex-skúškových príkladov

Zdroj a literatúra:

- [Introduction to Java Programming, !!!!Tenth Edition](#)

Cvičenia: jednoduché aplikácie s GUI:

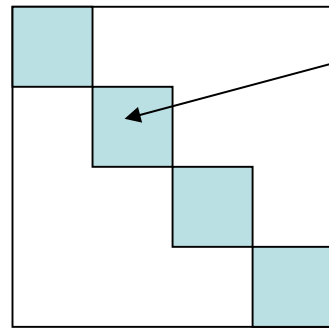
- euro-kalkulačka,
- logické hry: hra15, pexeso, ...

# Hracia plocha

hracia plocha je často šachovnica rôznych rozmerov. Ako ju implementujeme:

1. jeden veľký canvas v Pane-li:

- musíme riešiť transformáciu pixelových súradníc do súradníc hracej plochy:

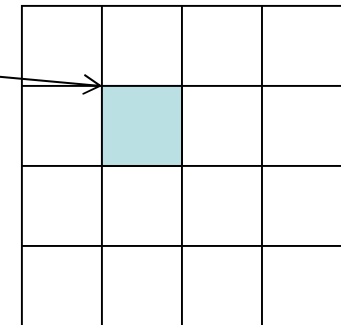


`[event.getX(),event.getY()]->[1,1]`

- a naopak, v metóde `paintMôjCanvas/paintMôjComponent [i,j] -> [pixelX, pixelY]`

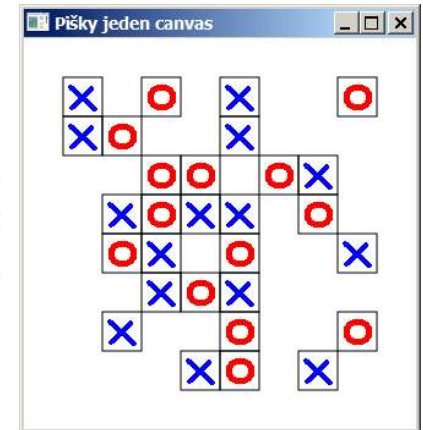
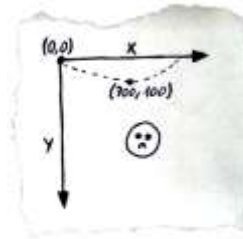
2. grid canvasov/Pane-lov:

- každý canvas/panel má svoje súradnice od `[0,0]`
- každý canvas/panel má svoj mouse event handler
- každý canvas panel má svoju metódu `paint/paintMôjCanvas`
- veľkosť gridu upravíme podľa veľkosti obrázkov,  
resp. veľkosť obrázku upravíme podľa veľkosti panelu



3. grid buttonov/Button-ov, Button môže mať obrázok ako ikonu

# 1. Riešenie Canvas



```
class Piskyground extends Canvas {
    Image imageO = new Image("o.gif");           // čítanie obrázku
    Image imageX = new Image("x.gif");
    double cellSize = 2+Math.max( // 2+ znamená dva pixle pre orámovanie obrázku
        Math.max(imageX.getWidth(), imageO.getWidth()), // zoberieme najväčší
        Math.max(imageX.getHeight(), imageO.getHeight())); // z rozmerov obrázkov

    public Piskyground() {
        setWidth(SIZE * cellSize);                // veľkosť hracej plochy
        setHeight(SIZE * cellSize);
        setOnMouseClicked(event -> {              // mouse event handler pre celú plochu
            int col = getCol(event.getX());         // transformácia z pixlov na riadok
            int row = getRow(event.getY());         // stĺpec
            if (ps.playground[col][row] != 0) return; // logika hry:niekto tam už...
            ps.playground[col][row]=(ps.nextPlayerIsX) ? 1 : -1; // kto je na ťahu
            paintCell(col, row);                   // prekresli len kliknuté políčko
            ps.nextPlayerIsX = !ps.nextPlayerIsX;  // // logika hry:ďalší na ťahu
        });
    }
}
```



# 1. Riešenie Canvas

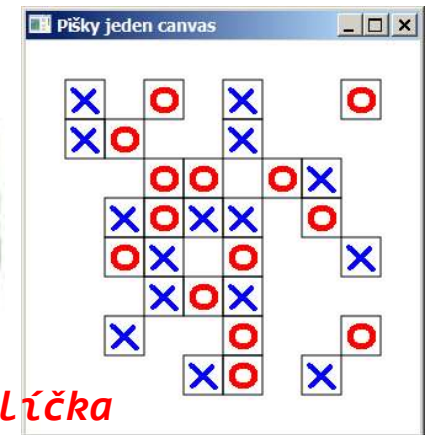
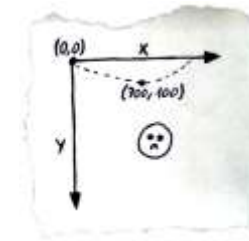
```
class Piskyground extends Canvas {
```

```
...
```

```
public void paintCell(int col, int row) { // kreslenie políčka
    double px = getPixelX(col); // transformácia row, col
    double py = getPixelY(row); // na pixlové súradnice px, py
    GraphicsContext gc = getGraphicsContext2D(); // do gc kreslíme
    gc.strokeRect(px, py, cellSize, cellSize); // kresli rámček šírky 1px
    if (ps.playground[col][row] == 1) gc.drawImage(imageX, px + 1, py + 1);
    else
        if (ps.playground[col][row] == -1) gc.drawImage(image0, px + 1, py + 1);
}
```

Napriek tomu, že transformácie row, col do pixelových súradníc sú často jednoduché lineárne transformácie (\* / niečo, +- niečo), doprajte si tú abstrakciu a vytiahnite ich do extra metód !!!

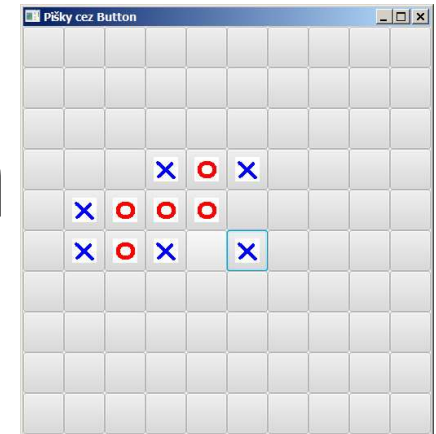
```
private int getRow/Col(double pixel) {
    return (int)(pixel/cellSize);
}
private double getPixelX/Y(int i) {
    return i*cellSize;
}
```



## 2. Riešenie GridPane/Button

Aby ste vedeli uložiť a načítať konfiguráciu hry, reprezentujte ju extra triedou, ktorá je serializovateľná

```
class PiskyState implements Serializable {  
    public int[][] playground = new int[SIZE][SIZE];  
    public boolean nextPlayerIsX = false; // na ťahu  
    public long elapsedTime = 0;          // čas...  
                                           // ďalšie veci, čo predstavujú konfiguráciu  
}  
  
class Piskyground extends GridPane {  
    public Piskyground() {  
        for (int i = 0; i < SIZE; i++)  
            for (int j = 0; j < SIZE; j++)  
                add(new PiskyCell(i, j), i, j);  
                                           // pridaj všetky políčka  
    }  
}
```



Výhody:

- nepotrebujeme transformácie pixel<->cell,
- nikdy si nepomýlite riadok, stĺpec, lebo každé políčko má svoj lokálny event-handler,
- pomerne ľahké riešenie, ak to grafika úlohy dovoľí

Súbor: [PiskvorkyGridButton.java](#)

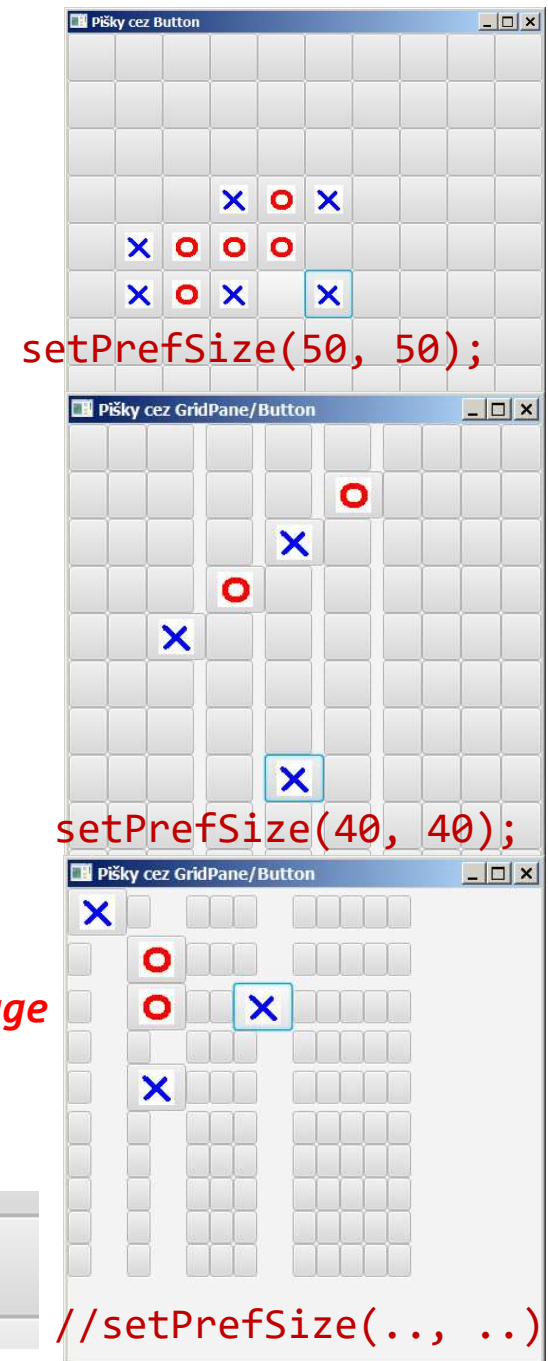


## 2. Riešenie GridPane/Button

```
class PiskyCell extends Button {
    int i, j;      // políčko si pamätá svoje súradnice
    public PiskyCell(int i, int j) {
        this.i = i; this.j = j;      // odtiaľto ...
        setPrefSize(50, 50); // vyexperimentovaná veľkosť
        setOnAction(event -> {
            if (ps.playground[i][j] != 0) return;
            if (ps.nextPlayerIsX) {
                ps.playground[i][j] = 1; // button.setGraphic
                setGraphic(new ImageView(new Image("x.gif")));
            } else {
                ps.playground[i][j] = -1; // ImageView, nie Image
                setGraphic(new ImageView(new Image("o.gif")));
            }
            ps.nextPlayerIsX = !ps.nextPlayerIsX;
        });
    }
}
```

Nevýhody:

- renderovanie gridu nemáte úplne pod kontrolou
- nevieme sa zbaviť škaredého lemu okolo obrázka

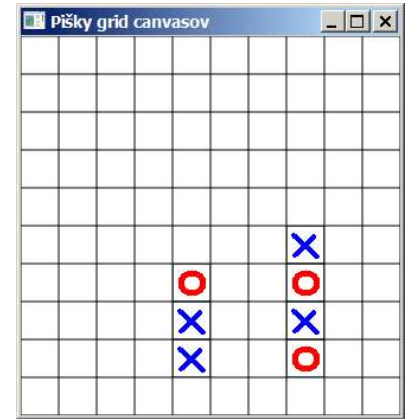


Súbor: [PiskvorkyGridButton.java](#)

# 3. Riešenie Grid/Canvas

```
class PiskyCell extends Canvas {
    int i, j; // rovnako, políčko si pamätá svoje súradnice
    Image imageO = new Image("o.gif");
    Image imageX = new Image("x.gif");
    double cellSize = 2 + // veľkosť bunky aj s orámovaním
        Math.max(Math.max(imageX.getWidth(), imageO.getWidth()),
            Math.max(imageX.getHeight(), imageO.getHeight()));

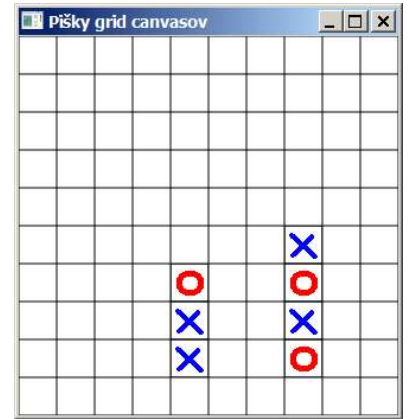
    public PiskyCell(int i, int j) {
        this.i = i; this.j = j;
        setWidth(cellSize); setHeight(cellSize); // nastav veľkosť bunky
        setOnMouseClicked(event -> {
            if (ps.playground[i][j] != 0) return; // „logika“ hry
            ps.playground[i][j] = (ps.nextPlayerIsX)?1:-1;
            paintCell(); // treba ju prekresliť po zmene stavu
            ps.nextPlayerIsX = !ps.nextPlayerIsX;
        });
    }
}
```



# 3. Riešenie Grid/Canvas

```
class PiskyCell extends Canvas {  
    public void paintCell() { // prekreslenie políčka  
        GraphicsContext gc = getGraphicsContext2D();  
        gc.strokeRect(0, 0, getWidth(), getHeight()); // rámček  
        if (ps.playground[i][j] == 1) gc.drawImage(imageX, 1, 1); // obrázok x,o  
        else if (ps.playground[i][j] == -1) gc.drawImage(imageO, 1,1);  
    } }  

```

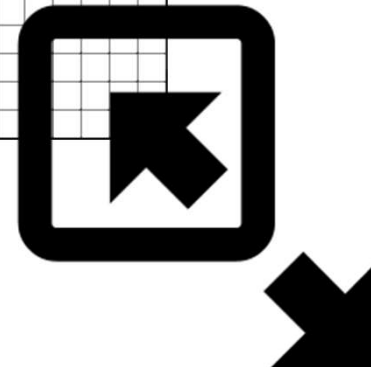
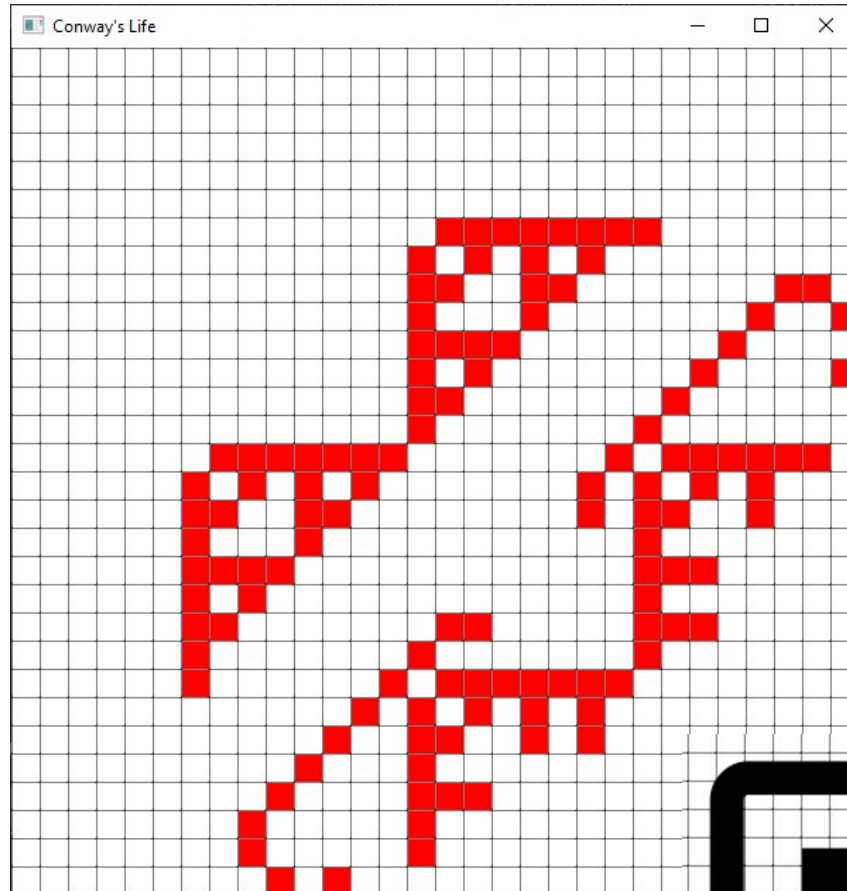
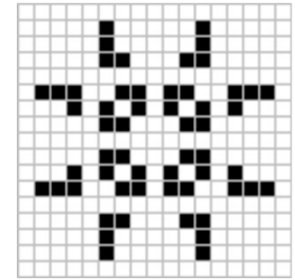


Vo všetkých troch riešeniach sme použili vnorené triedy

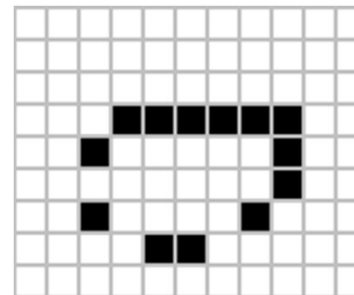
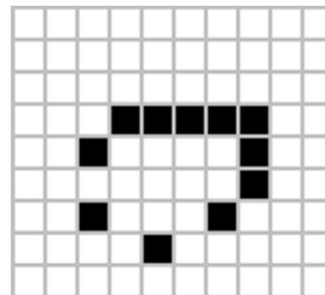
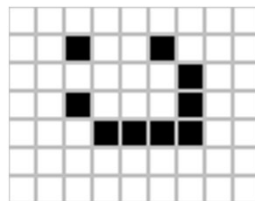
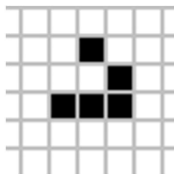
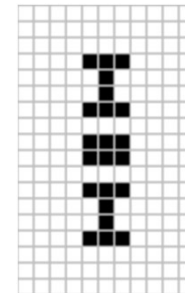
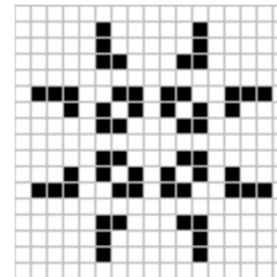
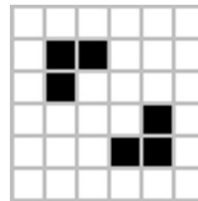
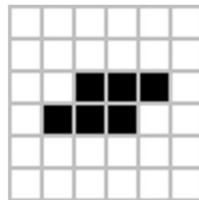
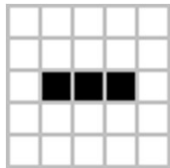
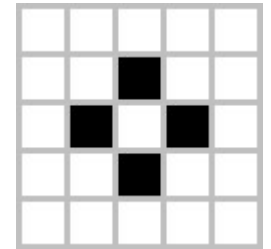
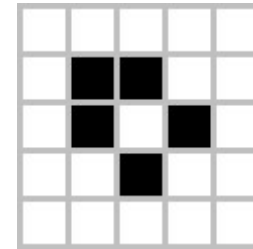
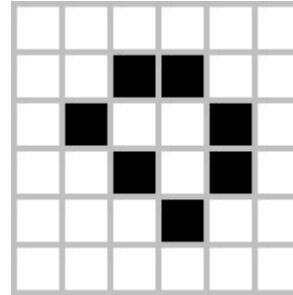
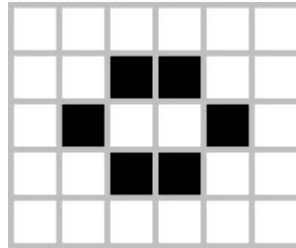
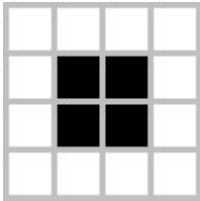
```
public class PiskvorkyGridCanvas extends Application {  
    PiskyState ps = new PiskyState(); // konfigurácia hry, vidia všetci  
    public void start(Stage primaryStage) { ... }  
    public static void main(String[] args) { ... }  
    class Piskyground extends GridPane { ... } // vytvorí konfiguráciu  
    class PiskyCell extends Canvas { ... } // kreslí z konfigurácie  
    public PiskyCell(int i, int j) { ... }  
}  
// akurát PiskyState nie je vnorená, prečo ?  
public class PiskyState implements Serializable { ... }
```

Súbor: [PiskvorkyGridCanvas.java](#)

# Conway's Life



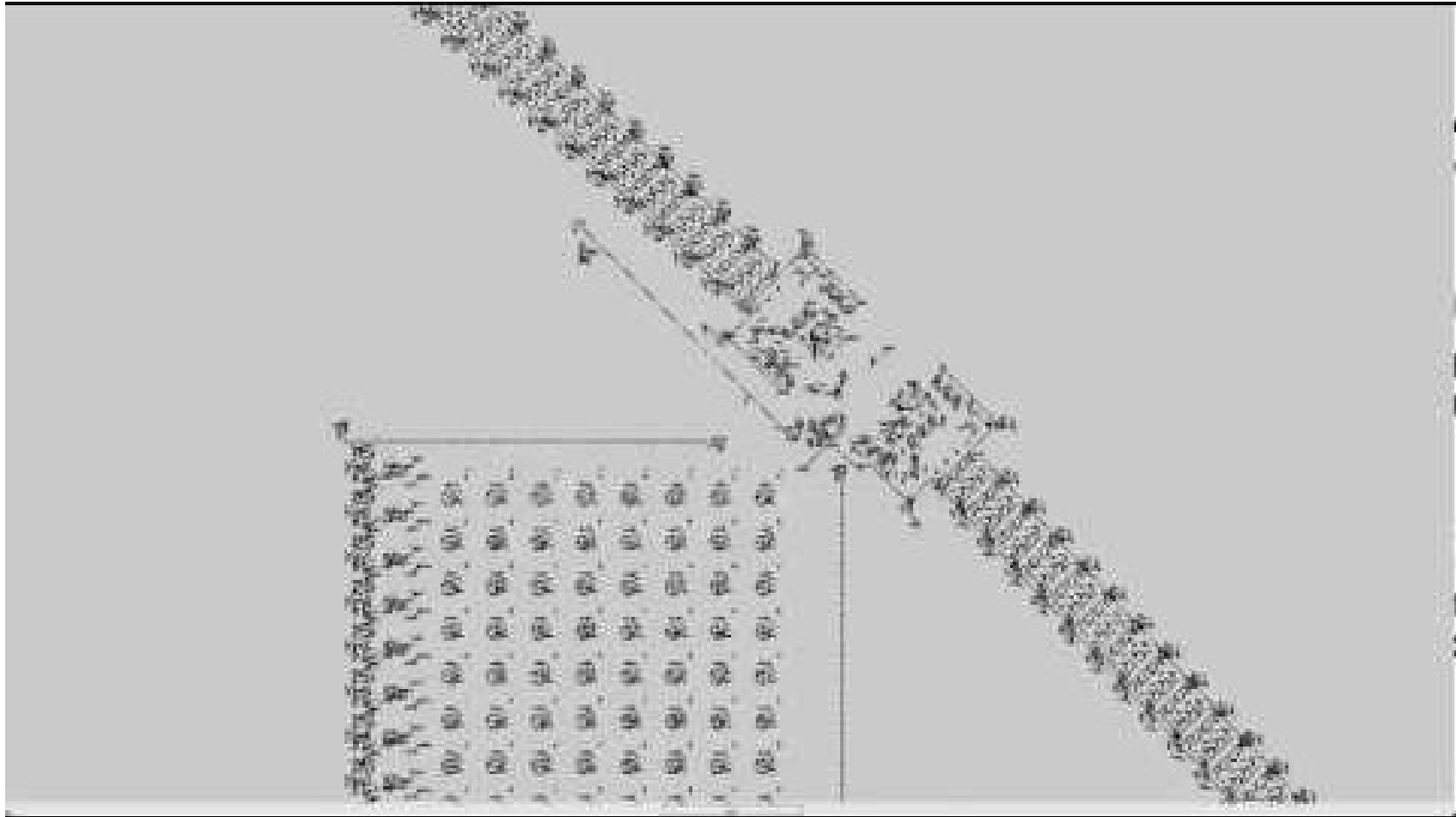
# Stills



[https://en.wikipedia.org/wiki/Conway's\\_Game\\_of\\_Life](https://en.wikipedia.org/wiki/Conway's_Game_of_Life)

# Life is Turing Complete

Paul Rendell



<https://www.ics.uci.edu/~welling/teaching/271fall09/Turing-Machine-Life.pdf>