**Gymnázium, Praha 6, Arabská 14**
Obor programování

**ROČNÍKOVÝ PROJEKT**

Vladimír Samojlov, 1.E

# Kalkulátor

Duben 2022

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V ..... dne .....  Vladimír Samojlov .....

Název práce: Kalkulačka

Autor: Vladimír Samojlov

Abstrakt: Cílem projektu bylo vytvořit funkční kalkulačku, která bude schopna provádět základní matematické operace, kterými jsou: sčítání, odčítání, násobení, dělení. Dále bude kalkulátor schopen provádět výpočet druhé mocniny i odmocniny.

Tato kalkulačka je schopna počítat se všemi desetinnými čísly. Program pro základní matematické operace postupuje podle jednotlivých kroků. V kalkulačce jsou také uvedeny případy, ve kterých nemá matematická operace smysl.

Klíčová slova: Kalkulačka, Matematické výpočty, Početní operace, Tlačítka, CSS, Java

Title: Calculator

Author: Vladimír Samojlov

Abstract: The goal of this project was to create a functional calculator, that will be able to perform basic mathematical operations, which are: addition, subtraction, multiplication, division. Another options, which the calculator will be able to show is the calculation of the square and the square root.

This calculator is able to count with all decimal numbers. The program for basic mathematical operations follows the individual instructions. The calculator also indicate the cases, in which the mathematical operation does not make sense.

Keywords: Calculator, Mathematical calculations, Computational operations, Buttons, CSS, Java

# Obsah

**1 Úvod**

**2 Tělo kalkulátoru 6**

2.1 Vytváření plochy . . . . . 6

2.1.1 Displej . . . . . 8

2.1.2 Tlačítka . . . . . 8

2.2 Upravení plochy . . . . . 10

2.2.1 Vzhled . . . . . 10

2.2.2 Barva ohraničení tlačítka při stisknutí . . . . . 11

2.2.3 Ikona aplikace . . . . . 11

**3 Matematické číslice, početní operátory a funkce tlačítek 13**

3.1 Desetinná čísla . . . . . 13

3.2 Početní operátory . . . . . 14

3.2.1 Základní početní operátory . . . . . 14

3.2.2 Rozšířené početní operátory . . . . . 16

3.3 Funkce tlačítek . . . . . 18

3.3.1 DEL . . . . . 18

3.3.2 AC . . . . . 18

3.3.3 Desetinná tečka . . . . . 19

3.3.4 Rovnítko . . . . . 19

**4 Zobrazení výsledků 20**

4.1 Vědecká notace čísel . . . . . 20

4.2 Zvětšování a zmenšování čísel . . . . . 21

**Závěr 22**

**Bibliografie 23**

**Seznam obrázků 25**

# 1. Úvod

Tento dokument se zabývá matematickým kalkulátorem, který má za úkol vyřešit zadané početní výpočty nebo operace. Kalkulačka patří mezi jedny z nejfunkčnějších zařízení, které v dnešní době zařazují mezi nejvíce používané nástroje, protože lidem dokáže usnadnit velkou část práce. Můj projekt se zaměřil na kalkulátor, který dokáže vyřešit spousta druhů aritmetických výpočtů.

## Zadání

Kalkulátor má mít níže uvedenou funkcionalitu:

1. Počty se základními i rozšířenými matematickými operacemi.
2. Zpracování výsledku.
3. Funkce jednotlivých tlačítek.

# 2. Tělo kalkulátoru

Tato kapitola se rozděluje na dvě velké podkapitoly, které popisují jak vytvořit a upravit plochu u kalkulačky. První podkapitola zobrazuje a vysvětluje jednotlivé postupy při tvoření plochy, tlačítek a výsledné obrazovky. Druhá podkapitola vyobrazuje především vzhled a tvar, do kterého spadá barevnost a styl plochy kalkulačky.

## 2.1 Vytváření plochy

Plocha u kalkulačky se skládá ze dvou částí, které si nyní podrobněji popíšeme v následujících podkapitolách. Nyní však budu zprvu líčit, jak vytvoříme prostor kalkulačky neboli plochu, kterou následně budu využívat.

Při vytváření plochy budeme v JavaFX používat soubor se zkratkou *FXML*. *FXML* představuje jazyk, který slouží k navrhování uživatelských rozhraní neboli lehčeji řečeno formulářů. [1] Pro tento typ souboru budeme využívat JavaFX SceneBuilder, pomocí kterého je možné vytvořit plochu kalkulátoru graficky bez psaní programu. Výsledný výtvor kalkulátoru vytvořený v SceneBuilder bude automaticky přepsán do *FXML* programu.

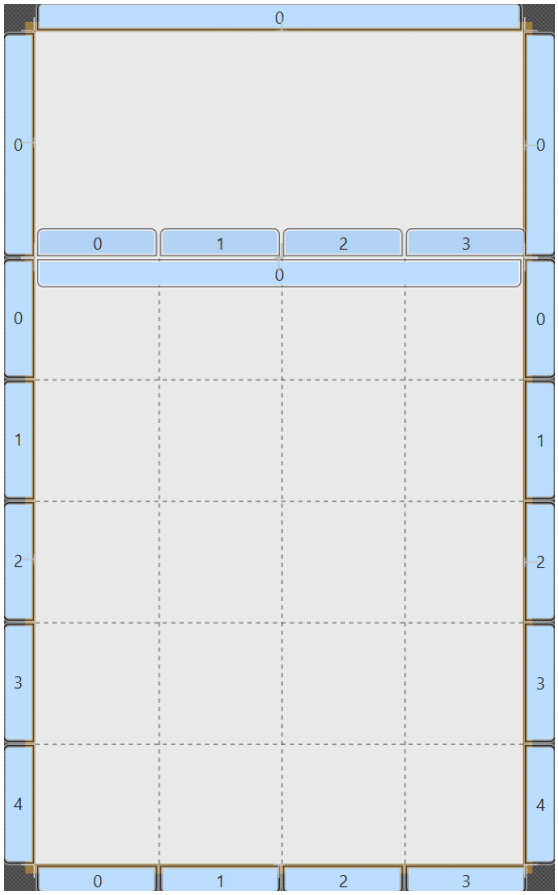
```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.net.*?>
<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.*?>
```

**Výpis kódu 2.1**: Zobrazení základního příkladu FXML souboru [2]

Po otevření Scene Builderu je nejdříve zapotřebí smazat vzor formuláře a hned poté si vytvořit *BorderPane*. Ten využijeme jako plochu, která je rozdělena do pěti samostatných oblastí, přičemž každá oblast může obsahovat dílčí komponentu.

Mezi těchto pět oblasti spadá: *Bottom*, *Center*, *Top*, *Right*, *Left*. Následně si vytvoříme dva komponenty *GridPane*, které pomáhají rozmístit jednotlivé komponenty do mřížky. Pomocí *BorderPane* máme možnost přemístit první *GridPane* do části Center a druhý *GridPane* do části Top.

Následně je zapotřebí upravit velikost délky a šířky *BorderPane* a také i *GridPane*, ve kterém je nutno změnit sloupce a řádky. Vzniknou tedy dvě části schématu, které jsou dále použity v kapitole 2.2.1 a 2.2.2.
Obrázek 2.2 ukazuje jednotlivé části *GridPane* s použitím BorderPane.




**Obrázek 2.2:** Dvě části kalkulátoru vytvořené s pomocí *GridPane* v SceneBuilder. Obrázek zobrazuje rozměry obou částí – Displej (1x1) a Tlačítka (4x5).

U kalkulátoru plochy je podstatné v kódové části SceneBuilder určit název akce *On Action*, na které je vyrobena plocha. Tato akce lze využít k oddělení funkčnosti a stavu od komponenty. Ke každé komponentě *Button* a *TextField* užijeme v souboru *KalkulackaKontroler* anotaci *@FXML*, která označí metody pro použití v jazyce Java.

*KalkulackaKontroler* je *JavaFX* kontrolér, jenž lze dosáhnout pomocí *FXML*. Za pomoci kontroléru lze sestavit v jazyce Java program.

Nyní si jednotlivé části kalkulátoru popíšeme podrobněji. V každé části kalkulačky jsou použity rozdílné komponenty, které umožňují potřebné funkce kalkulátoru v JavaFX.

### 2.1.1 Displej

Displej kalkulátoru slouží k zobrazení zadaných čísel uživatelem a výsledného početního výsledku. K vytvoření elektronické obrazovky kalkulátoru existuje několik prvků, které je možné v JavaFX použít. Pro tento kalkulátor použijeme textovou komponentu *TextField*. TextField implementuje ovládací prvek uživatelského rozhraní, který přijímá a zobrazuje textový vstup. Slouží především k zadávání textu, přičemž v tomto případě je uplatňován k zobrazení číslic. [3]

Textové pole aplikujeme do vrchní části „*Top*“ v rozvržení *BorderPane*, ve kterém jsou zobrazovány čísla vybrané nebo výsledky výpočtů provedené použivatelem. Pro znázornění psaných symbolů si zvolíme zarovnání *BASELINE_LEFT*, které zarovná text v *TextField* vlevo dolů. V kódové části u textové komponenty *TextField* je důležité pojmenovat *fx: id*, do kterého napíšeme název pojmenování displeje, například *VysledekVypoctu*.

V textové oblasti se vyskytuje spousta druhů rodin, velikostí a stylu znaků, které je možností si vyzkoušet. Pro přijatelnost si zvolíme ideální druh stylu a velikosti písma – *Family:* System, *Style:* Bold, *Size:* 35px.

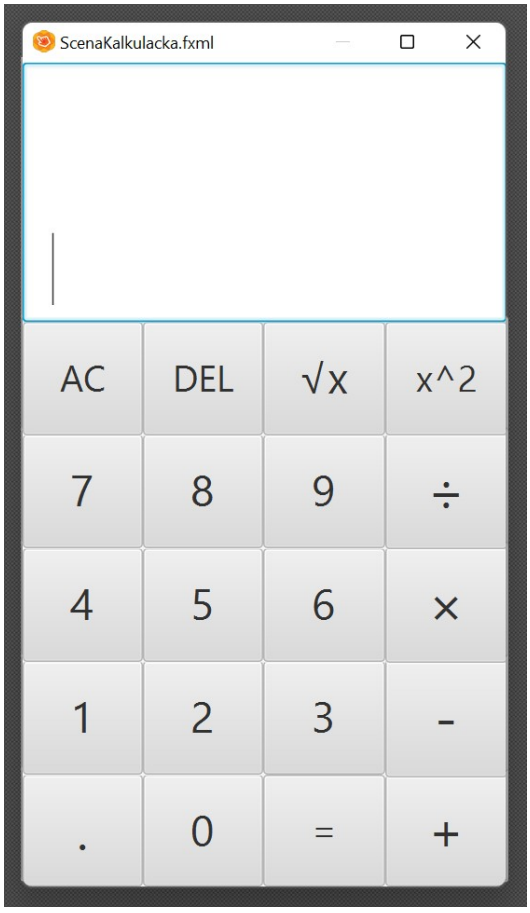
### 2.1.2 Tlačítka

Tlačítka patří mezi další nejdůležitější prvky kalkulačky. Ve svém výkladu popisuje tlačítko *JavaFX* Jakob Jenkov takto: „*A JavaFX Button control enables a JavaFX application to have some action executed when the application user clicks the button. The JavaFX Button control is represented by the class javafx. scene. control. Button.*“ [4]

V překladu do českého jazyka by se dalo jednoduše říct, že za element *Button* neboli tlačítko se dá považovat ovládající prvek nebo komponentu, která umožňuje provést akci. V kalkulátoru jsou tlačítka použita jako matematické číslice nebo početní operátory. Pro využití každé klávesy na prostřední ploše je nutné zavést potřebný počet tlačítek, tak aby na každém místu *GridPane* nezůstalo jediné volné místo v oblasti kláves.

Zprvu je velice důležité vytvořit správný počet kláves kalkulátoru a poté jednotlivé vytvořené čidla přemístit do jednotných prázdných ploch, které nám předem byly poskytnuty komponentou Grid*Pane*. Následně každé čidlo zvětšíme na maximální velikost délky a šířky předepsaného čtverce, který je zobrazen přerušovanou linií.

V podobné situaci jako v podkapitole 2.1.1, je u každé částice *Button* je nezbytné v kódové části pojmenovat *fx: id*, do kterého napíšeme název početní operace, například sčítání. Zbývá tedy tlačítka správně pojmenovat.




**Obrázek 2.3:**

Vytvoření výsledné plochy kalkulačky v aplikaci SceneBuilder. Plocha zahrnuje vytvořený Displej (*TextField*) a Tlačítka (*Button*), které jsou vytvořeny pomocí komponentů uvedené v závorce. Jednotlivá čidla jsou pojmenována a změněna.

U každé klávesy je v tabulce nutné stanovit znak, mezi které patří matematická čísla – 0 až 9, aritmetické operátory – sčítání, odčítání, násobení, dělení, umocňování i odmocňování na druhou exponentu nebo také jiné funkce kalkulátoru – desetinná tečka, AC, DEL, rovnítko. Jednotlivé symboly v předkreslené tabulce stanovíme tak, aby každé tlačítko neobsahovalo stejný symbol.

Obrázek 2.3 znázorňuje vytvořenou plochu kalkulátoru – Displej s tlačítky, která vyobrazuje komponenty *TextField* a *Button*.

## 2.2 Upravení plochy

Pro upravení plochy kalkulátory budeme využívat kaskádové styly se zkratkou CSS a zároveň SceneBuilder. Kaskádové styly jsou designový jazyk, který slouží ke zlepšení vzhledu a dojmu z aplikace. Pomocí CSS lze ovládat zbarvení textu, styl fontů, mezery mezi odstavci, velikost sloupců a rozložení. S výjimkou toho je možností řídit obrázky na pozadí nebo použité barvy, návrhy rozvržení, variace zobrazení a další řadu efektů pro různá zařízení.

Kaskádové styly lze ovládat i za pomoci SceneBuilder, který jednotlivé zlepšení uplatní do svého souboru FXML nebo je možno CSS použít v samotném souboru Java. [5] [6]

### 2.2.1 Vzhled

V JavaFX existuje spousta pravidel stylu, pomocí nichž je možné si zvolit stylování. Jak výše bylo uvedeno, pro stylování budeme používat *CSS*, ve kterém je zapotřebí napsat program pro a upravení barvy plochy a stylu fontu kalkulačky.

Pro změnu vzhledu je nutnost zajistit program, který dokáže vytvořit barevný přechod nebo jednobarevnou výplň v prostoru kalkulátoru. V případě použití kaskádových stylů si zvolíme různobarevný lineární přechod. Pro tento případ je nejlepší volbou si zvolit funkci *linear-gradient*, která předává různé parametry mající vliv na výchozí podobu celého přechodu.

U této funkce lze zvolit také směr barevného přechodu. K této funkci přidáme vlastnost barvy *-fx-background-color*, která zobrazí lineární přechod dvou barev. [7]

Lineární gradient lze použít u jednotlivých tlačítek v SceneBuilder. Při použití jednobarevných čidel není nezbytné použít CSS, jelikož v SceneBuilder je možné si změnit barvu tlačítka, popřípadě i barvu a font textu.
Na obrázku 2.4 je znázorněn zápis pro lineární přechod dvou různých barev.
Na obrázku 2.5 je použit *linear-gradient*.

### 2.2.2 Barva ohraničení tlačítka při stisknutí

V této podpodkapitole platí stejný princip jako v předchozí podpodkapitole 2.2.1. Je zapotřebí využít kaskádové styly k tomu, aby při stisknutí tlačítka, došlo ke změně barvy ohraničení čidla kalkulačky. Pro tuto činnost využijeme efekt *pressed*, po které uvedeme barvu a šířku ohraničení.

Díky efektu *pressed* stačí uvést pouze dvě vlastnosti: *-fx-border-color* a *-fx-border-width*.
Na obrázku 2.4 je znázorněn zápis ohraničení tlačítka při stisknutí.
Na obrázku 2.5 je ukázána barva ohraničení při stisknutí tlačítka.

```
.LinearniGradient{
    -fx-background-color: linear-gradient(to top right, #FF6699, #FFE066);
}
.button:pressed{
    -fx-border-color: #FFE066;
    -fx-border-width: 4;
}
```

**Obrázek 2.4:** Zápis pro lineární přechod dvou různých barev a ohraničení tlačítka při stisknutí v CSS.

### 2.2.3 Ikona aplikace

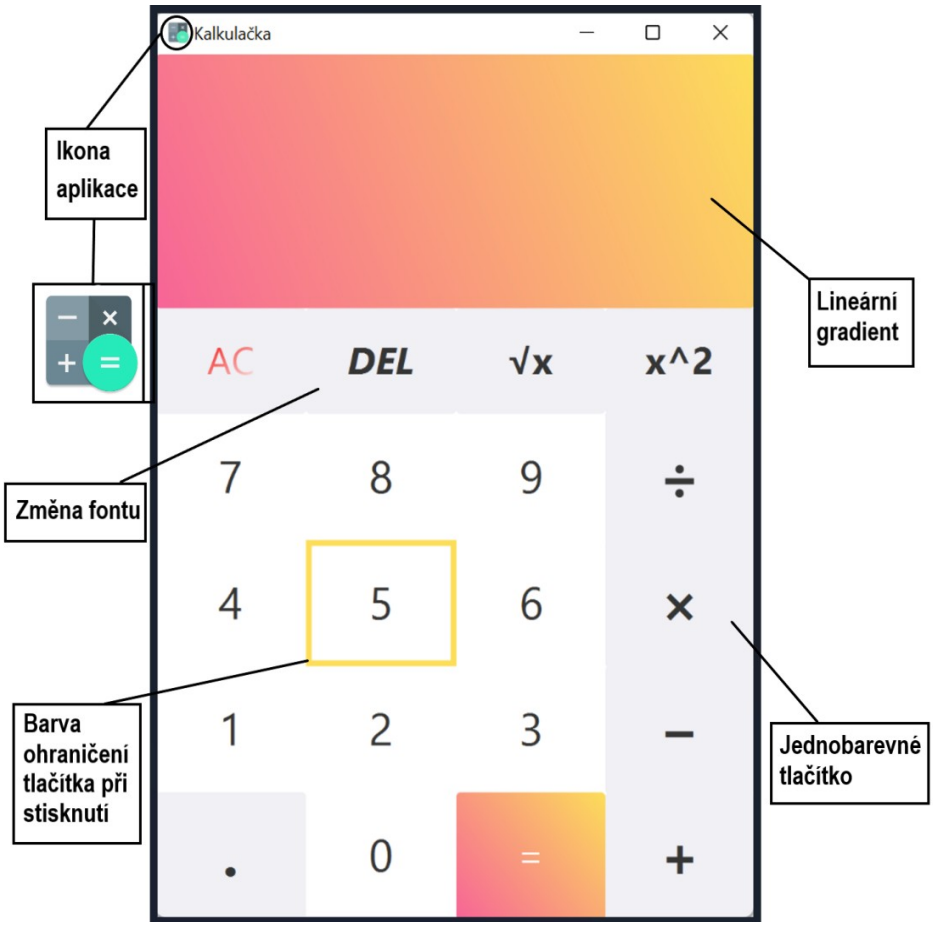
Každá JavaFX aplikace má stanovenu standardní ikonu, jenž může být nahrazena obrázkem z počítače. Ikona aplikace probíhá v momentu, když je zavolána metoda *start*, která zahájí provádění aplikace.

Obrázek musí být uložen ve složce daného projektu, jinak ikona aplikace nebude funkční a aplikace nebude spuštěna kvůli chybě. Grafický symbol lze změnit pomocí metody *getIcons().add()*, která umožní přidat obrázek do ikony. Tato metoda získá obrázky ikon. Ikona se vytvoří při stanovení okna *stage* v JavaFX. Pro přesné určení vybrané fotografie je nutné, aby funkce obsahovala další podrobnosti, do které přidáme další metodu:

```
stage.getIcons().add(new Image(KalkulackaFX.class.getResourceAsStream("Kalkulacka.png")));
```

Další metodou, která je uvedena výše je *getResourceAsStream()*. Tato metoda se používá k získání prostředku se zadaným způsobem třídy *KalkulackaFX*. Metoda zároveň vrací zadaný prostředek této třídy ve formě objektu *InputStream*. [8]

Výše je napsán program, který je schopen zvolit určený obrázek ze složky projektu, v tomto případě *Kalkulacka.png*.
Na obrázku 2.5 je uvedena ikona kalkulátoru.




**Obrázek 2.5:** Popis jednotlivých úprav plochy kalkulátoru.

# 3. Matematické číslice, početní operátory a funkce tlačítek

## 3.1 Desetinná čísla

Tato podkapitola popisuje desetinná čísla, které matematická kalkulačka využívá pro provádění výpočtů. Mezi desetinná čísla řadíme cifry, jejichž způsob zápisu obsahuje čísla pomocí celé části a desetinné části oddělené desetinnou čárkou. [9]

Pro použití vstupní hodnoty prvního nebo druhého desetinného čísla do textového komponentu *TextField*, je potřeba aplikovat primitivní datové typy. V jazyce Java rozeznáváme pouze dva takové typy: *double* a *float*. Mezi hlavní rozdíly patří rozsah hodnoty a počet desetinných míst. Pro kalkulátor je, s výjimkou odčítání, při výpočtu výhodnější použít datový typ *double*, který má dvojnásobnou přesnost oproti *float*. Rozsah *double* se vykazuje na 15 až 16 čísel, zatímco přesnost *float* je pouze na 7 čísel. [10] Operátory a počet obsahují celočíselný datový typ *int*. U desetinných číslic je uplatňován *DecimalFormat*. Jedná se o specifikátor formátu převádějící výsledné číslo výpočtu na řetězec desetinných cifer. Je důležitý zejména pro udávání minimálního požadovaného počtu číslic v finálním výsledku počtu. [11]

Pro vypsání každé číslice umístěné v komponentě *Button* aplikujeme metodu *public void Vypocet*, která je podrobněji popsána v části 3.2.1. V metodě použijeme funkci *appendText(String text)*, do které lze napsat obsah znaků v textovém řetězci *String*.

Na obrázku 3.1 je uveden příklad, ve kterém je komponenta Button s číslicí „1“ vyobrazena na displeji s pomocí funkce *appendText()*.

```
public void Vypocet(ActionEvent event) {

    if (event.getSource() == jedna) { // Načtení tlačítka s fx: id - jedna
        VysledekVypoctu.appendText("1"); // Vypsání textu "1" na displej
```

**Obrázek 3.1:** Zobrazení číslice „1“ v textovém poli *TextField*.

## 3.2 Početní operátory

Rozlišujeme několik zásadních početních symbolů. Jedná se o čtyři základní početní operátory a dva rozšířené početní operátory. V jazyce Java se základní a rozšířené početní operátory značně liší.

### 3.2.1 Základní početní operátory

Do této skupiny operátorů řadíme čtyři početní symboly a operace. Tabulka uvedena níže znázorňuje základní početní operátory a operace v jazyce Java.

| **Početní operátory** | **Početní operace** |
|---|---|
| + | Sčítání |
| − | Odčítání |
| × | Násobení |
| ÷ | Dělení |

Jak už v podpodkapitole 2.1.2 bylo uvedeno, jednotlivé početní operátory mají v *FXML* stanovenu *fx: id*. Pro určení kláves *Button* je potřeba propojení, podle kterého lze provádět úpravy v normálním Java souboru.

V jazyce Java vytvoříme metodu *public void Vypocet*, do které přidáme událost *ActionEvent event*, která je generována komponentou *Button*. Po propojení události *ActionEvent event* s tlačítkem použijeme metodu *event.getSource()*, která vrátí objekt, na kterém došlo k události. [12]
Pro příklad základní početní operace si zvolíme sčítání.

Na obrázku 3.2 je zobrazena ukázka kódu pro početní operátor sčítání, která zároveň vykazuje jednotlivé popisy k řádkům. Na obrázku jsou zároveň zobrazeny metody *getText()* a *setText()*. Metoda *getText()* vrátí text z jednořádkového textového pole *TextField*. [13]

Tato metoda *getText()* je zároveň umístěna v metodě *parseDouble()* vracející nový *double* na hodnotu zadaným řetězcem. [14]

S prostřednictvím metody *setText()* je možno nastavit text v textovém poli *TextField*. U ostatních základních operátorů platí stejný způsob zápisu programu, akorát je nutné zaměnit volbu početního operátoru.
Na obrázku 3.2 je napsáno, že znaménko plus spadá pod operátor s číslem 1, který je zapotřebí vytvořit.

Pro vytvoření početního operátoru aplikujeme podmíněné označení *case* k příkazu *switch*, který umožní otestovat proměnnou. Příkaz *switch* může obsahovat více případů označení *case*. K označení případu lze využít další příkaz *break*, který ukončí provádění procesu. [15]
Je zapotřebí si zároveň vytvořit proměnnou *Pocet*, jenž obsahuje celočíselný datový typ *int*.

Na obrázku 3.3 je ukázáno schéma uplatňující příkaz *switch* a podmíněné označení *case*, které je použito jako volba sčítání. Taktéž je na obrázku zobrazena metoda *String.valueOf()*. Pomocí metody *String.valueOf()* lze převést typ hodnoty *int* na řetězec a stanovit tak výsledek v *TextField*.

```
} else if (event.getSource() == plus) { // Volba: Sčítání

    cislo1 = Double.parseDouble(VysledekVypoctu.getText());
    // Vypsání výsledku po operaci sčítání

    VysledekVypoctu.setText(""); // TextField nic nezobrazí po stisknutí tlačítka (+)
    operator = 1; // Zvolení operátoru: Operátor č.1 spadá pod sčítání
    Pocet = 0; // Sčítání je možno použít nekonečno krát
```

**Obrázek 3.2:** Příklad použití základního početního operátoru plus („+“) při propojení s komponentem *Button* a *TextField*. Zvolení základního početního operátoru pro další část kódování, které je ukázáno na obrázku 3.3.

```
DecimalFormat x = new DecimalFormat("#.###############");

switch (operator) { // Pod příkazem switch uvedeme operátor

    case 1: // Volba: Sčítání

        DoubleVysledek = cislo1 + cislo2; // Výsledek je roven sčítání dvou čísel

        if (DoubleVysledek == cislo1 + cislo2) {
        // Pokud uživatel zvolí sčítání

        VysledekVypoctu.setText(String.valueOf(x.format(DoubleVysledek)));
        // Výsledek se zobrazí podle formátu x
        }
        break; // Ukončení provádění procesu
```

**Obrázek 3.3:** Navázanost na předchozí schéma 3.2. Použití příkazu *switch* a podmíněné označení *case* v matematické operaci sčítání.

Aplikace s základními matematickými početními operátory probíhá v níže uvedených krocích:

| |
|---|
| 1) Zadání čísla č.1 |
| 2) Zvolení základního operátoru |
| 3) Zadání čísla č.2 |
| 4) Stisknutí rovnítka |
| 5) Výsledek zobrazen v *TextField* |

### 3.2.2 Rozšířené početní operátory

Do této skupiny operátorů řadíme dva početní symboly a operace. Mezi rozšířené početní operátory zahrnujeme odmocniny a mocniny. Odmocnina i mocnina mají v kalkulačce podobný význam, jelikož fungují podle stejného principu.

Jak již bylo zmíněno v části 3.2.1, pro propojení komponentů *Button* je potřeba využít událost *ActionEvent event*. U rozšířených početních operátorů využijeme taktéž označení *case* s příkazem *switch*. Program pro odmocňování i umocňování je napsán pouze pro jedno číslo, nikoli pro více čísel. Průběh aplikace je naprosto odlišný na rozdíl od základních operátorů. Tentokrát uplatníme odmocninu jako typ rozšířeného operátoru.

```
} else if (event.getSource() == odmocnit) { // Volba: Odmocňování

    cislo1 = Math.sqrt(DoubleVysledek);
    // Vypsání výsledku po operaci odmocňování

    VysledekVypoctu.setText("");
    // TextField nic nezobrazí po stisknutí tlačítka (√x)

    operator = 6; // Zvolení operátoru: Operátor č.6 spadá pod odmocňování
    Pocet = 0; // Odmocninu lze použít nekonečno krát
```

**Obrázek 3.4:** Příklad použití rozšířeného početního operátoru plus („√x“) při propojení s komponentem *Button* a *TextField*. Zvolení rozšířeného početního operátoru pro další část programu, které je ukázáno na obrázku 3.5.

```
case 6:  // Operátor: Druhá Odmocnina
    DoubleVysledek = Math.sqrt(cislo2);
    // Výsledek je roven druhé odmocnině z druhého čísla

    if (DoubleVysledek == Math.sqrt(cislo2)) {
    // Pokud je zvolena jako první hodnota odmocnina a druhá hodnota jako číslo

        VysledekVypoctu.setText(String.valueOf(x.format(DoubleVysledek)));
    // TextField zobrazí výsledný text v formátu x
    }
    break; // Ukončení provádění procesu
```

**Obrázek 3.5:** Navázanost na předchozí obrázek 3.4. Zvolení matematického operátoru – odmocnina a uplatnění příkazu *switch* a podmíněné označení *case*.

Aplikace s rozšířenými matematickými početními operátory probíhá v níže uvedených krocích:

| |
|---|
| 1) Zvolení rozšířeného operátoru |
| 2) Zadání čísla |
| 3) Stisknutí rovnítka |
| 4) Výsledek zobrazen v *TextField* |

## 3.3 Funkce tlačítek

Do této podkapitoly patří tlačítka, která vykonávají činnosti pro standardní kalkulačku. Mezi tyto tlačítka spadají klávesy *DEL*, *AC*, *Tečka*, *Rovnítko*. Nyní si popíšeme jednotlivá tlačítka popíšeme detailněji v následujících podpodkapitolách.

### 3.3.1 DEL

Zkratka *DEL* neboli celým jménem *Delete* vymaže jednotlivé znaky z pravého konce displeje. Pro aplikování klávesy je zapotřebí využít metody *getText() a substring()*. Metoda *substring()*, jenž spadá pod třídu *String, v*rátí nový řetězec, který spadá podřetězec tohoto řetězce. [16]

Do metody *substring()* se zahrnuje i finální metoda *length()*, která najde délku řetězce. V případě klávesy *Delete* odečteme číslo „1“, jelikož je potřeba zkrátit text při stisknutí tlačítka vždy o jedno místo v jednořádkovém poli *TextField*.
Na obrázku 3.6 je ukázán popis kódu k tlačítku *DEL*, pomocí něhož klávesa *DEL* je schopna fungovat.

```
} else if (event.getSource() == delete) { // Volba: Tlačítko Delete
    VysledekVypoctu.setText(VysledekVypoctu.getText().substring(0, VysledekVypoctu.getText().length() - 1));
    // Displej vymaže jeden znak po kliknutí na tlačítko DEL
```

**Obrázek 3.6:** Znázornění programu k funkci klávesy *Delete* využívající metody *getText() a substring()*.

### 3.3.2 AC

AC („*All Clear*“), z angličtiny přeloženo „Vymazat Vše“, slouží k vymazání aktuálního výpočtu a resetování každé funkce kalkulátoru. Na obrázku 3.7 je zobrazen jednoduchý program pro tlačítko *AC*.

```
} else if (event.getSource() == AC) { // Volba: Tlačítko AC
    VysledekVypoctu.setText(""); // Smazání veškerého textu

    cislo1 = 0; //Resetování prvního čísla
    cislo2 = 0; // Resetování druhého čísla
    operator = 0; // Není zvolen žádný operátor
    Pocet = 0; // Funkci AC lze použít nekonečno krát
```

**Obrázek 3.7:** Znázornění programu k funkci klávesy *AC* využívající metody *setText()* a jednotlivých proměnných.

### 3.3.3 Desetinná tečka

Desetinná tečka je v jazyce Java používána jako desetinný oddělovač. Na obrázku 3.8 je vyobrazen popis k programu desetinné tečky, ve kterém je použita proměnná *Pocet*.

```
} else if (event.getSource() == tecka && Pocet == 0) {
// Volba: Tečka s počtem nula

    VysledekVypoctu.appendText(".");
    // Po stisknutí je vypsán symbol desetinné tečky

    Pocet = 1; // Pouze jednu tečku lze použít v číslu
```

**Obrázek 3.8:** Zobrazení programu k desetinné tečce s využitím proměnné *Pocet* a metody *appendText()*.

### 3.3.4 Rovnítko

Rovnítko je v kalkulačce aplikován jako matematický symbol, jenž je použit k zjištění výsledku po matematické operaci. Obrázek 3.9 ukazuje popis programu pro rovnítko, které jsou použity metody *getText()* a *parseDouble()*.

```
} else if (event.getSource() == rovnat && operator > 0) {
// Po stisknutí rovnítka a operátoru, který musí být větší něž 0
// => Operátory jsou od 0 do 6

    cislo2 = Double.parseDouble(VysledekVypoctu.getText());
// V případě, že první číslo je operace, druhé číslo musí být rovnítko
```

**Obrázek 3.9:** Ukázka kódu ukazující program pro rovnítko s využitím metod *getText()* a *parseDouble()*.

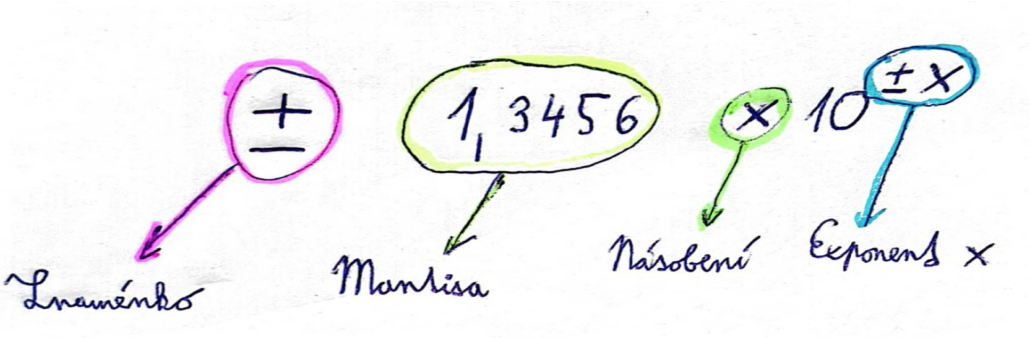
# 4. Zobrazení výsledku

Tato kapitola se zabývá druhy zobrazení výsledku v textovém poli *TextField*. Výsledek uvádějící hodnotu matematické operace lze rozdělit podle zvětšování a zmenšování textu a dle vědecké notace čísel.

## 4.1 Vědecká notace čísel

Tato podkapitola zkoumá matematické výpočty, při kterých je výsledné číslo větší než kladné nebo záporné číslo přesahující limit textového pole.

Mezi nejdelší čísla kalkulátoru je možné zařadit číslo přesahující délku šestnácti cifer. U hodnot, jež přesahují limit délku čísla je zapotřebí použít vědeckou notaci čísel. Lehčeji řečeno, vědecká notace slouží k „zkrácení“ předešlého výsledku. Vědecká notace používá takzvaný semilogaritmický tvar, jenž lze taktéž využít při výpočtech. Zápis vědecké notace je složen z jednotlivých částí, které jsou zobrazeny na obrázku 4.1.




**Obrázek 4.1:** Na obrázku jsou zobrazeny části vědecké notace: znaménko(±), mantisa (jakékoliv reálné nebo komplexní číslo), násobení číslem $10^{\pm x}$.

Pro použití vědecké notace v jazyce Java je potřeba u každé matematické operace stanovit pouze jednu podmínku, jež se týká limitu výsledku. Je potřeba zahrnout metody *getText()* a *length()*, jenž dohromady vytvoří podmínku délky pro textové pole *TextField*. Po podmínce je použita hlavní metoda *setText()* a zároveň nahrazen specifikátor formátu čísel *DecimalFormat x* metodou *String.valueOf*, která vypíše vědeckou notaci z výsledku.

Na obrázku 4.2 je zobrazen program pro uplatnění vědecké notace výsledku, který je použit mezi jednotlivými operacemi.

```
if (VysledekVypoctu.getText().length() > 16) {
// Pokud výsledek přesáhne délku 16 znaků

    VysledekVypoctu.setText(String.valueOf(DoubleVysledek));
// Stanovení výsledku bez formátu x, použití vědecké notace
}
```

**Obrázek 4.2:** Vyobrazení kódu pro převedení výsledku do vědecké notace při limitu.

## 4.2 Zvětšování a zmenšování čísel

Zvětšování a zmenšování čísel je použito v případě, když výsledek je zapsán s pomocí vědeckého zápisu čísel. Zmenšování textu je automaticky aplikováno, jelikož výsledná hodnota může přesáhnout délku výsledného textového pole *TextField*. Naopak zvětšování může být uplatněno v momentě, kdy výsledek je zmenšený, lze pomocí komponenty *Button* stisknout klávesu *AC*, která vymaže celý obsah displeje a zároveň resetuje velikost textu v displeji.

Do původního programu 4.2 je nutné přidat pouze metodu *setFont()*, do které přidáme třídu *Font* s další metodou *font()*. Metoda *setFont()* slouží k nastavení písma. Pro upřesnění použijeme třídu *Font*, do které lze přidat metodu *font()* k zvolení rodiny, tloušťky a velikosti písma.
Na finálním kódu 4.3 s zobrazením výsledku lze vidět kód pro vědeckou notaci s automatickým zmenšováním textu v displeji.

```
if (VysledekVypoctu.getText().length() > 16) {
// Pokud výsledek přesáhne délku 16 znaků

    VysledekVypoctu.setFont(Font.font("System", FontWeight.BOLD, 31.5));
// Stanovení fontu písma

    VysledekVypoctu.setText(String.valueOf(DoubleVysledek));
// Stanovení výsledku bez formátu x, použití vědecké notace
}
```

**Obrázek 4.3:** Výsledný program při vědecké notaci se zmenšením textu.

# Závěr

Výsledný kalkulátor funguje výborně. V průběhu výpočtů se neobjevují žádné chyby s kalkulačkou. Vytváření programu bylo relativně rychlé a obešlo se bez velkých potíží. Při průběhu projektu bylo ve vývojovém prostředí uplatněno spousta způsobů, podle kterých byl kalkulátor vylepšen. Projekt by šlo rozšířit o několik dalších funkcí kalkulátorů, mezi které například patří historie výpočtů.

## Splnění zadání

Výsledný kalkulátor splňuje zadání, které bylo uvedeno v úvodu. Při projektu bylo nejtěžší vytvořit vzhled kalkulátoru a změna barvy ohraničení tlačítka, protože pro stylování plochy byl využit designový jazyk *CSS* se kterým autor doposud neměl takové zkušenosti.

# Bibliografie

[1] ČÁPKA, David. *FXML* https://www.itnetwork.cz/java/javafx/java-tutorial-uvod-do-javafx [cit. 2022-04-08].

[2] *Using FXML to Create a User Interface* https://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm [cit. 2022-04-21].

[3] *TextField* https://docs.oracle.com/javafx/2/ui_controls/text-field.htm [cit. 2022-04-08].

[4] JENKOV, Jakob. *JavaFX Button* https://jenkov.com/tutorials/javafx/button.html [cit. 2022-04-08].

[5] VITA. *Kaskádové styly (CSS)* https://www.itnetwork.cz/java/javafx/tutorial-javafx-quickstart-udalosti-a-css [cit. 2022-04-10].

[6] *JavaFX - CSS* https://www.tutorialspoint.com/javafx/javafx_css.htm [cit. 2022-04-10].

[7] JAHODA, Bohumil. *Gradienty v CSS* https://jecas.cz/gradient [cit. 2022-04-10].

[8] SRINAM, *Class getResourceAsStream() method in Java* with *Examples* https://www.geeksforgeeks.org/class-getresourceasstream-method-in-java-with-examples/ [cit. 2022-04-10].

[9] *Desetinná čísla* https://cs.wikipedia.org/wiki/Desetinn%C3%A9_%C4%8D%C3%ADslo [cit. 2022-04-12].

[10] ČÁPKA, David. *Datové typy* https://www.itnetwork.cz/java/zaklady/java-tutorial-typovy-system-podruhe-datove-typy-string [cit. 2022-04-12].

[11] Standard numeric format strings https://docs.microsoft.com/en-us/dotnet/standard/base-types/standard-numeric-format-strings [cit. 2022-04-13].

[12] *ActionEvent* https://www.oreilly.com/library/view/java-for-dummies/9781118239742/118239742-ch05.html [cit. 2022-04-13].

[13] *GetText Function (JavaJFCTextField)*
https://www.microfocus.com/documentation/silk-test/195/en/silktestclassic-195-help-en/STCLASSIC-7FB80E93-GETTEXTFUNCTIONJAVAJFCTEXTFIELD-REF.html
[cit. 2022-04-14].
[14] *Java Double parseDouble() Method* https://www.javatpoint.com/java-double-parsedouble-method [cit.2022-04-15].
[15] *Java case keyword*
https://www.javatpoint.com/case-keyword-in-java [cit. 2022-04-13].
[16] REDDY, Ankitha. *Java String substring() Method example*
https://www.tutorialspoint.com/Java-String-substring-Method-example

# Seznam obrázků

2.1 Zobrazení základního příkladu *FXML* souboru . . . . . 6
2.2 Dvě části kalkulátoru vytvořené s pomocí *GridPane* . . . . . 7
2.3 Vytvoření výsledné plochy kalkulačky v aplikaci SceneBuilder . . . . . 9
2.4 Lineární přechod dvou různých barev a ohraničení tlačítka . . . . . 11
2.5 Popis jednotlivých úprav plochy kalkulátoru . . . . . 12
3.1 Zobrazení číslice v displeji . . . . . 13
3.2 Příklad použití základního početního operátoru . . . . . 15
3.3 Použití příkazu v matematické operaci sčítání . . . . . 16
3.4 Příklad použití rozšířeného početního operátoru . . . . . 17
3.5 Použití příkazu v matematické operaci odmocňování . . . . . 17
3.6 Tlačítko *Delete* . . . . . 18
3.7 Tlačítko *AC* . . . . . 19
3.8 Tlačítko *Desetinná tečka* . . . . . 19
3.9 Matematický symbol *rovnítko*. . . . . 19
4.1 Části vědecké notace . . . . . 20
4.2 Program pro převedení výsledku do vědecké notace . . . . . 21
4.3 Program pro vědeckou notaci se zmenšením textu. . . . . 21