

## Dokumentace + motivace

1.	Motivace.....	2
2.	Konkrétní příklad.....	3
3.	Již použitelné řešení.....	4
4.	Dokumentace této aplikace.....	4
5.	Jak aplikace funguje.....	6
6.	Úvodní test.....	7
7.	Rámcový popis kódu.....	8

### 1. Motivace.

- Rád bych se bezproblémově pohyboval ve vlastním kódu.
- Rád bych přesně věděl, co má obsahovat daná proměnná v daném cyklu.

Nápad:

- Dokumentovat si všechny proměnné.

Jakou by to mělo i výhodu:

- Určitě by se snadněji vyhodnocovaly testy.

### Rozevírání kódu JAVA:

Projekt zde něco dělá.

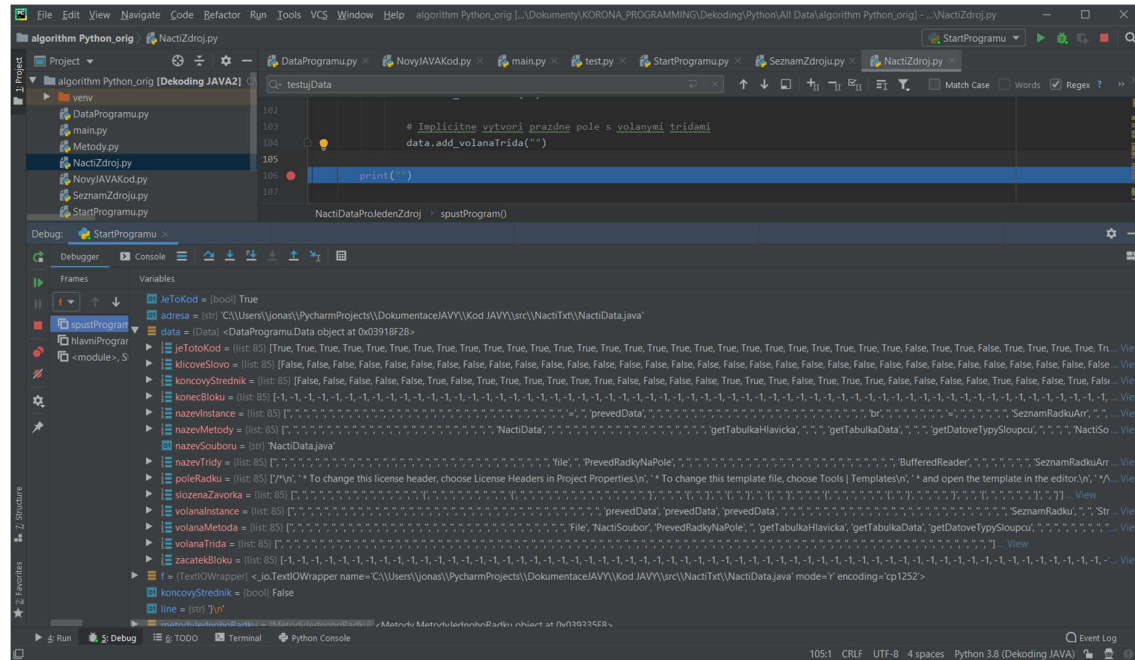
<https://github.com/jonasRower/Dekoding-JAVA>

Zatím jsem však nese-psal příliš podrobnou dokumentaci. Nějak na to nebyl čas.... Nebylo by dobré, kdyby mi něco přímo generovalo obsah jednotlivých proměnných, tak abych vygeneroval obsah všech proměnných velice rychle?

## 2. Konkrétní příklad

<https://github.com/jonasRower/Dekoding-JAVA/tree/main/All%20Data/algorithm%20Python>

Umístěte breakpoint na řádek 106 v modulu NactiZdroj.py. Data proměnné jsou následující:



Obr. 1 – Data testované aplikace

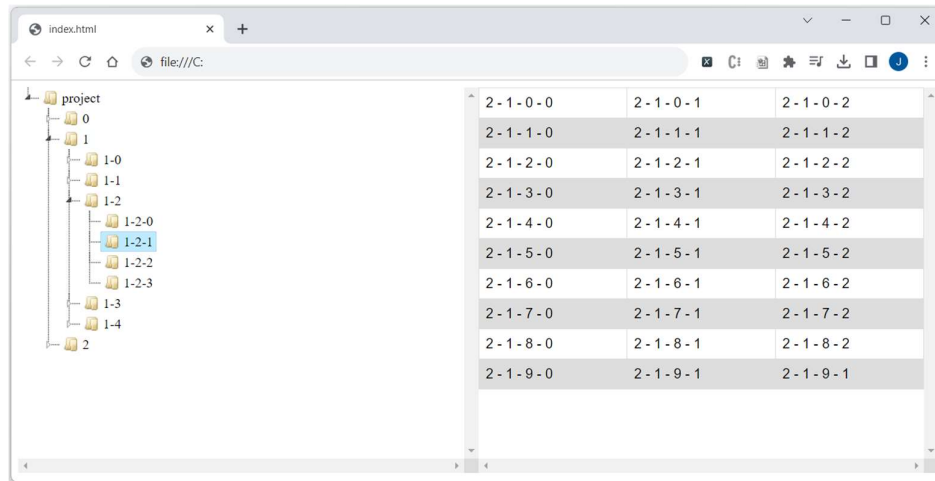
Co když však spustíme aplikaci na jiných datech? Pak samozřejmě budou data jiná. Je možné si někde bokem uchovávat vnitřní data aplikace. Jelikož když změním kód, i přes změnu, by kód měl vrátit stejné hodnoty. Znáš tyto hodnoty? Mám je někde uložené?

Umím si hodnoty efektivně dohledat?

### 3. Již použitelné řešení

Řešení by mohl částečně poskytovat již rozpracovaný projekt:

[https://github.com/jonasRower/docObjects\\_jsTree](https://github.com/jonasRower/docObjects_jsTree)



Obr. 2 – Zobrazení proměnných ve stromové struktuře

Objekt je zde zobrazen ve stromové struktuře, přičemž poslední 2 dimenze jsou zobrazené v tabulce. Klikneme-li na daný uzel ve stromu, data proměnné se aktualizují.

O aplikaci jako takové pojednávám zde:

[https://github.com/jonasRower/docObjects\\_jsTree/blob/main/docObjects.pdf](https://github.com/jonasRower/docObjects_jsTree/blob/main/docObjects.pdf)

### 4. Dokumentace této aplikace.

Na obrázku níže zobrazujeme část dat, z obrázku 1.

poleRadku	jeTotoKod	klicoveSlovo	<< koncovyStrednik >>
package PridejTabulku;	True	False	True
import sql_gui.*;	True	False	True
import java.sql.DatabaseMetaData;	True	False	True
import java.sql.DriverManager;	True	False	True
import java.sql.ResultSet;	True	False	True
import java.sql.SQLException;	True	False	True
import java.sql.Statement;	True	False	True
import java.util.ArrayList;	True	False	True

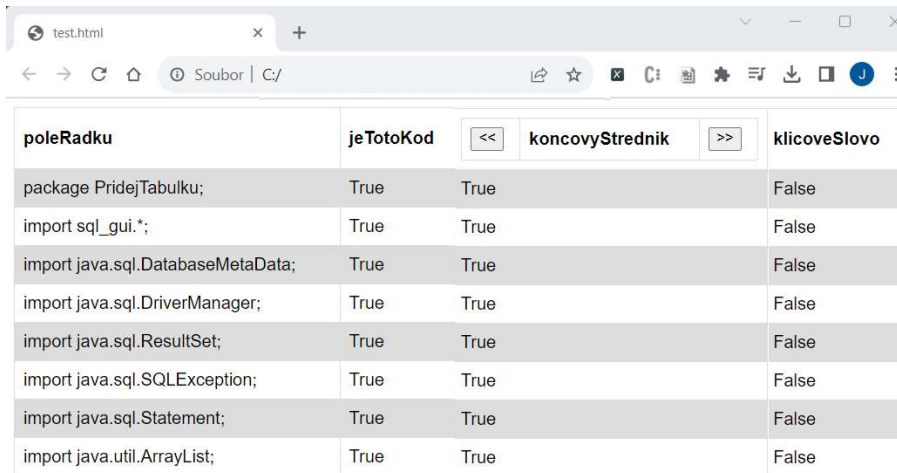
Obr. 3 – Část dat z obrázku 1.



## Dokumentace proměnných.

Pochopitelně, bude se jednat o tabulku v pravé části obrázku 2.



Předpokládaná funkcionálníta:

Po kliknutí na tlačítka   budeme posouvat daný sloupec doprava, či doleva.



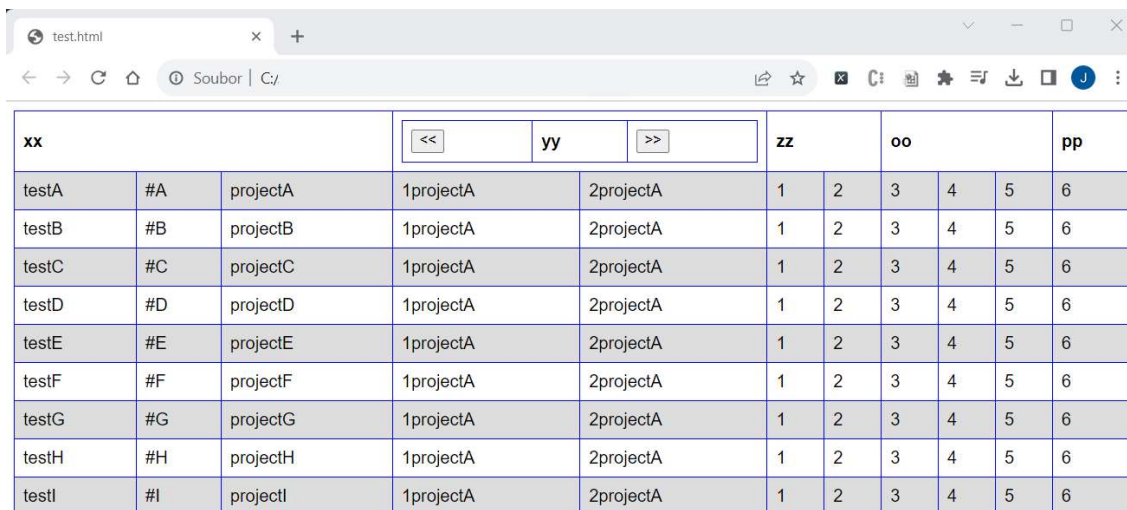
poleRadku	jeTotoKod		koncovyStrednik		klicoveSlovo
package PridejTabulku;	True	True			False
import sql_gui.*;	True	True			False
import java.sql.DatabaseMetaData;	True	True			False
import java.sql.DriverManager;	True	True			False
import java.sql.ResultSet;	True	True			False
import java.sql.SQLException;	True	True			False
import java.sql.Statement;	True	True			False
import java.util.ArrayList;	True	True			False



Obr. 4 – Přesunutý sloupec.

Tlačítka se   zobrazí kliknutím na daný sloupec.

V našem případě máme jeden sloupec = jedna proměnná pole (nebo objektu). Může nastat i případ, kdy se může jednat o dvojrozměrnou proměnnou. V tomto případě hlavička pro příslušné sloupce bude sloučená.

Obrázek níže již zobrazuje testovací data projektu.



xx				yy		zz		oo			pp
testA	#A	projectA	1projectA	2projectA		1	2	3	4	5	6
testB	#B	projectB	1projectA	2projectA		1	2	3	4	5	6
testC	#C	projectC	1projectA	2projectA		1	2	3	4	5	6
testD	#D	projectD	1projectA	2projectA		1	2	3	4	5	6
testE	#E	projectE	1projectA	2projectA		1	2	3	4	5	6
testF	#F	projectF	1projectA	2projectA		1	2	3	4	5	6
testG	#G	projectG	1projectA	2projectA		1	2	3	4	5	6
testH	#H	projectH	1projectA	2projectA		1	2	3	4	5	6
testI	#I	projectI	1projectA	2projectA		1	2	3	4	5	6

Obr. 5 – Testovací data aplikace. Hlavičky jsou zde pochopitelně sloučené.

## 5. Jak aplikace funguje

Zatím máme potřebné json-data uložena přímo ve scriptu:

```
class vratDataJson{
    constructor(){
        var jsonData = '[' +
            '{ "row": -1, "data":["testA", "#A", "projectA", "1projectA", "2projectA", "1", "2", "3", "4", "5", "6"] },' +
            '{ "row": 0, "data":["testB", "#B", "projectB", "1projectA", "2projectA", "1", "2", "3", "4", "5", "6"] },' +
            '{ "row": 1, "data":["testC", "#C", "projectC", "1projectA", "2projectA", "1", "2", "3", "4", "5", "6"] },' +
            '{ "row": 2, "data":["testD", "#D", "projectD", "1projectA", "2projectA", "1", "2", "3", "4", "5", "6"] },' +
            '{ "row": 3, "data":["testE", "#E", "projectE", "1projectA", "2projectA", "1", "2", "3", "4", "5", "6"] },' +
            '{ "row": 4, "data":["testF", "#F", "projectF", "1projectA", "2projectA", "1", "2", "3", "4", "5", "6"] },' +
            '{ "row": 5, "data":["testG", "#G", "projectG", "1projectA", "2projectA", "1", "2", "3", "4", "5", "6"] },' +
            '{ "row": 6, "data":["testH", "#H", "projectH", "1projectA", "2projectA", "1", "2", "3", "4", "5", "6"] },' +
            '{ "row": 7, "data":["testI", "#I", "projectI", "1projectA", "2projectA", "1", "2", "3", "4", "5", "6"] }' +
            '];

        var promenneAIndexy = '[' +
            '{ "variable": "xx", "columns": [0, 1, 2] },' +
            '{ "variable": "yy", "columns": [3, 4] },' +
            '{ "variable": "zz", "columns": [5, 6] },' +
            '{ "variable": "oo", "columns": [7, 8, 9] },' +
            '{ "variable": "pp", "columns": [10] }' +
            '];

        var objTxt = eval('jsonData');
        this.obj = JSON.parse(objTxt);

        var objTxtVar = eval('promenneAIndexy');
        this.objVar = JSON.parse(objTxtVar);
    }
}
```

Obr.6 – Zdrojová data aplikace jsou zatím uložena přímo ve scriptu.

Aplikace zatím začíná zde:

```
$(document).ready(function(){

    //ziska data
    var dataJsonu = new vratDataJson();
    var obj = dataJsonu.getJsonData();
    var objVar = dataJsonu.getObjVar();

    console.log(obj);

    //vykresli tabulku
    var zobrazUvodniStranku = new vytvorTabulku(obj, objVar)

    $("button").on( "click", function() {

        var id = $(this).attr("id");
        var objNewData = new prohodSloupceTabulky(obj, objVar, id);
        obj = objNewData.getObjNew();

        console.log(obj);

    });

});
```

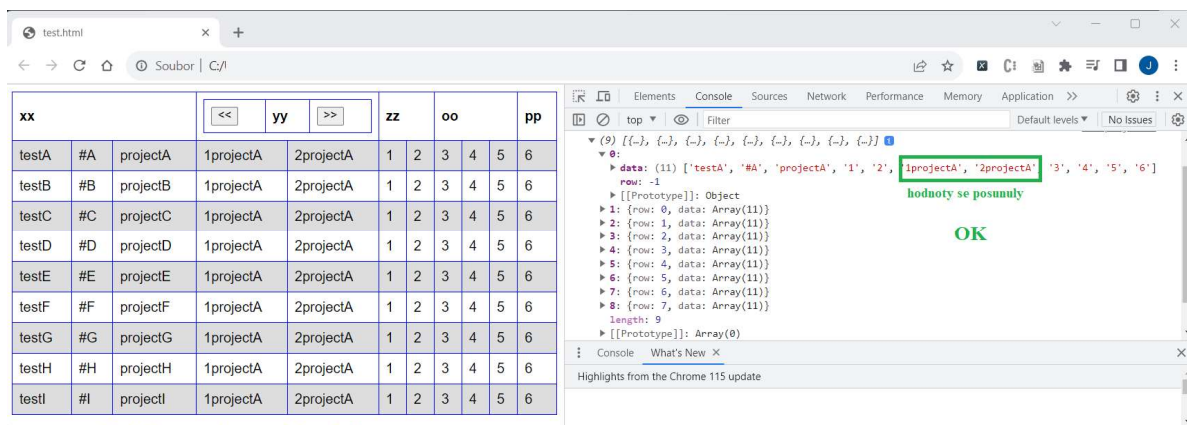
Obr.6 – Start běhu aplikace.



Následně klikneme na tlačítko:



Data na konzoli se posunuly správně, zatímco stránka se zatím nepřekresluje.



Stránka se zatím nepřekresluje

Obr. 9 – Data na konzoli se posunou správně, stránka se nepřekreslí.

Problematika bude řešena prostřednictvím webových služeb.

## 7. Rámcový popis kódu

Kód je zatím jednoduchý, nicméně co rámcově dělá:

```
class vytvorTabulku{
```

Vytváří tabulku, tak že z příslušných dat sestavuje appendString, který je následně vložen pomocí jQuery:

```
constructor(obj, objVar){
    //var tabAppendStr = this.vytvorAppendStr();
    var tabAppendStr = this.vytvorTabulkuZJson(obj, objVar);
    $('#table').append(tabAppendStr);
}
```





Tlačítka do buňky hlavičky jsou vložena pomocí subtabulky.

	<div><div>&lt;&lt;</div><div>yy</div><div>&gt;&gt;</div></div>		zz	
	1projectA	2projectA	1	2
	1projectA	2projectA	1	2
	1projectA	2projectA	1	2

Obr. 12 – (mini)Tabulka v tabulce. Tímto způsobem jsou vložena tlačítka.

Subtabulka je vložena do příslušné buňky, tím způsobem, že text v elementu je nahrazen kódem tabulky. To se děje funkcí:

```
sloucenýSloupecNew = sloucenýSloupec.replace(názevSloupce, subTabulka)
```

Celá metoda je zde:

```
vlozSubTabulku(sloucenýSloupec, názevSloupce, id){
    var subTabulka;
    var sloucenýSloupecNew;

    var idDoleva = "doleva_" + id + "";
    var idDoprava = "doprava_" + id + "";

    subTabulka = '<table>\n' +
        '    <tr>\n' +
        '        <th><button id=' + idDoleva + '>&lt;&lt;</button></th>\n' +
        '        <th>' + názevSloupce + '</th>\n' +
        '        <th><button id=' + idDoprava + '>&gt;&gt;</button></th>\n' +
        '    </tr>\n' +
        '</table>';

    sloucenýSloupecNew = sloucenýSloupec.replace(názevSloupce, subTabulka)

    return(sloucenýSloupecNew);
}
```

Obr. 13 – Metoda zajišťující vložení subtabulky.

Tabulku vkládáme pochopitelně do sloupce (hlavičky sloupce) s příslušným id. Metoda

```
vlozSubTabulku(sloucenýSloupec, názevSloupce, id){
```

je volána z metody níže. Zatím máme pevně nastaveno:

```
var idExp = "col_3";
```

Tudíž, vkládání tlačítek prozatím probíhá pouze ve sloupci s tímto id a neposouvá se.

```
vratPoleRadkuHlavicky(objVar){  
  
    var poleRadkuHlavicky = [];  
    var poleIdHlavicky = []  
  
    var idExp = "col_3";  
    var sloucenýSloupec;  
  
    for (let i = 0; i < objVar.length; i++) {  
  
        var objVarRadek = objVar[i];  
        var dataSlouceneSloupce = this.vratSlouceneSloupce(objVarRadek);  
        var idSloucenehoSloupce = dataSlouceneSloupce[1];  
        var názevSloupce = dataSlouceneSloupce[2];  
  
        sloucenýSloupec = dataSlouceneSloupce[0];  
  
        //prepise novou subtabulkou  
        if(idSloucenehoSloupce == idExp){  
            sloucenýSloupec = this.vlozSubTabulku(sloucenýSloupec, názevSloupce, idSloucenehoSloupce);  
        }  
  
        poleRadkuHlavicky.push(sloucenýSloupec);  
    }  
  
    return(poleRadkuHlavicky)  
}
```

Obr. 14 – Metoda odkud je volána metoda vkládající subtabulku.

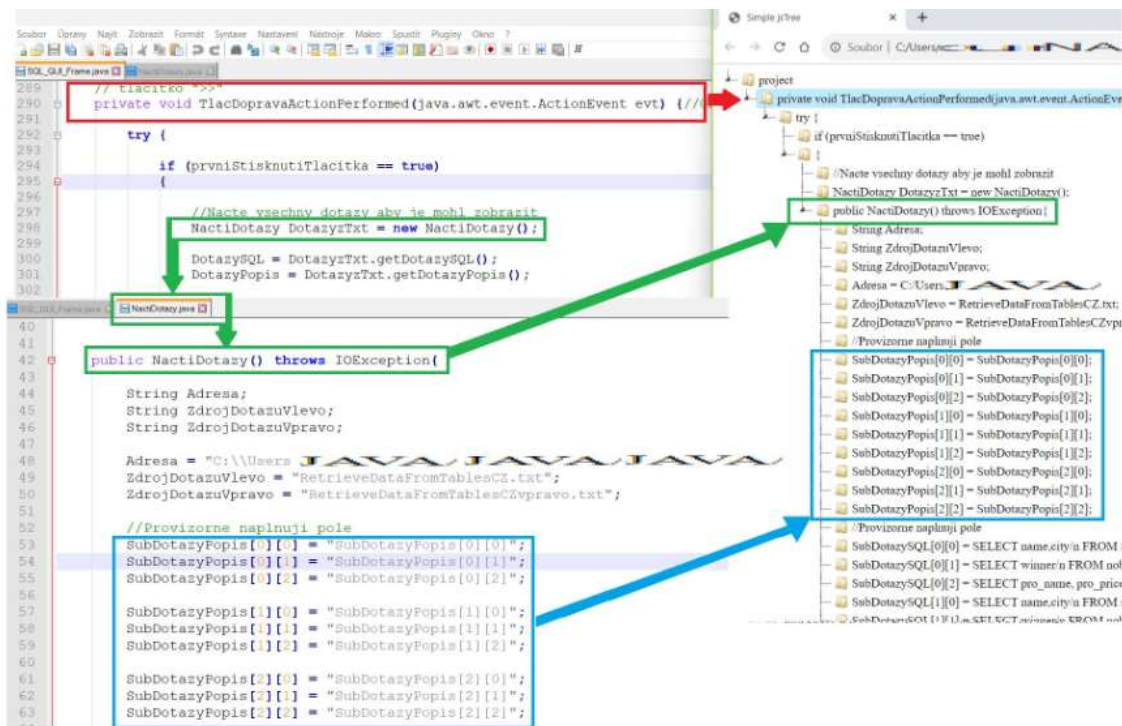
Motivace pro příští verzi.

- Osvojit si webové služby a pokračovat ve webových službách.

Motivace k dokončení tohoto projektu a projektů dalších:

- Vnímáme, že zde např. zdrojový kód na obr. 13 je volán metodou na obrázku 14. Určitě by bylo skvělé, když bychom dokázali zdrojový kód zobrazovat ve stromové struktuře. O tom však pojednává již projekt tento:

<https://github.com/jonasRower/Dekoding-JAVA/tree/V01>



- Budeme-li vědět, které metody jsou do kterých, zanořené, můžeme soustředit testování pouze na danou oblast a nehlédáme chyby tam, kde pro to není důvod.