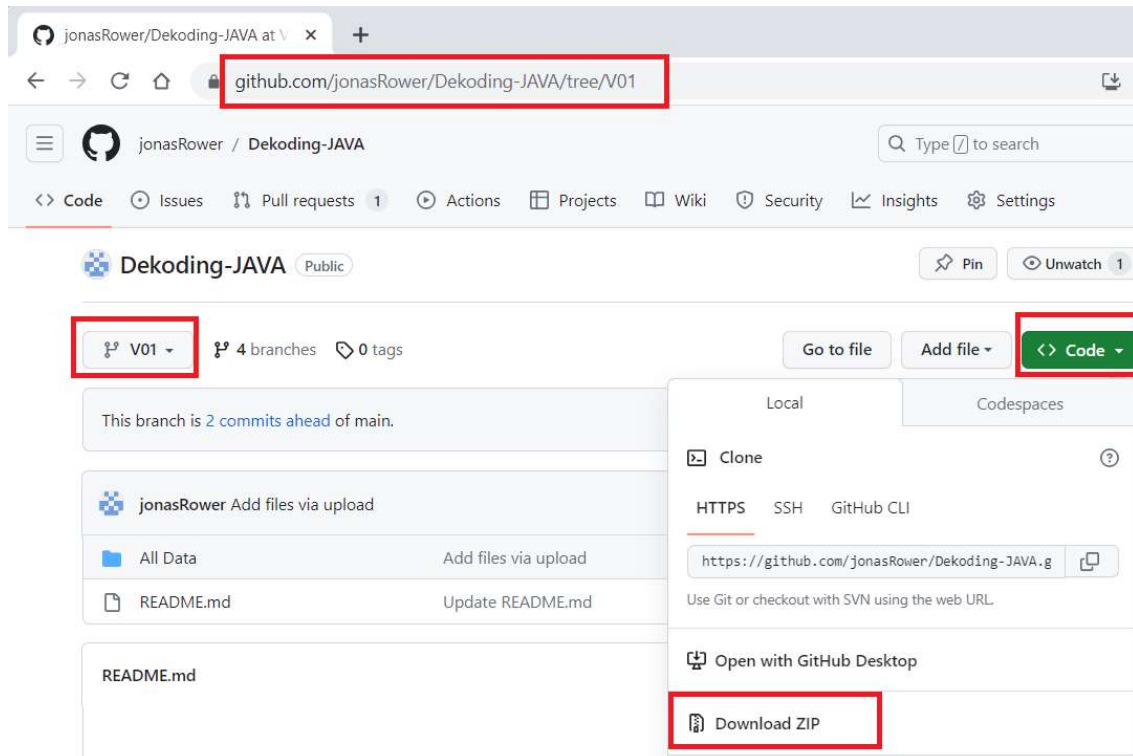


**Rozvíjení zdrojového kódu do stromové struktury.**  
*Dokumentace.*

1.	Verze V01.....	2
1.1.	Stažení verze V01.....	2
1.2.	První start.....	2
1.3.	Start debugu.....	4
1.4.	Objekt DataVsechSouboru.....	5
1.5.	Vytváření kódu do výstupu txt.....	10
2.	Verze V02.....	15
2.1.	Stažení verze V02.....	15
2.2.	První start V02.....	15
2.3.	Implementovaný kód V02.....	16
2.4.	Třída createIdArr():.....	19
2.5.	Algoritmus rozpoznávání zanoření závorek – teorie.....	22
2.6.	Mírná odlišnost od teorie.....	25
2.7.	Algoritmus rozpoznávání zanoření závorek – zdrojový kód.....	26

## 1. Verze V01

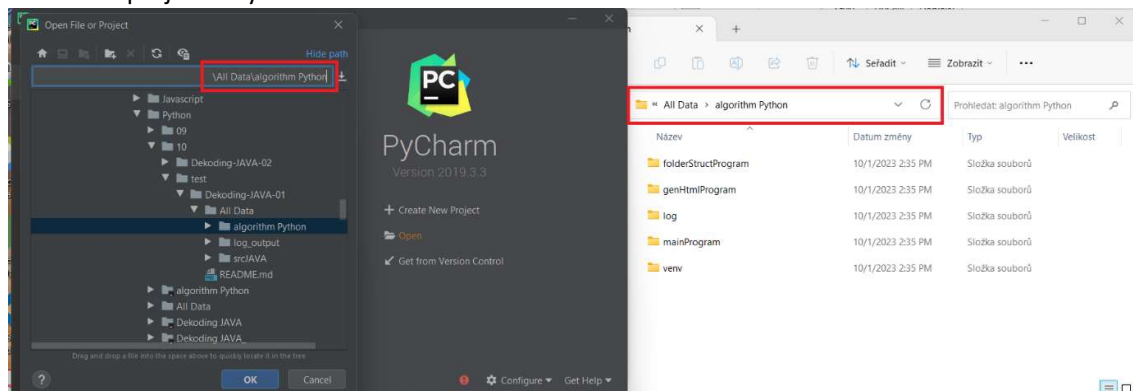
### 1.1. Stažení verze V01



Obr. 1 – stažení verze V01

### 1.2. První start

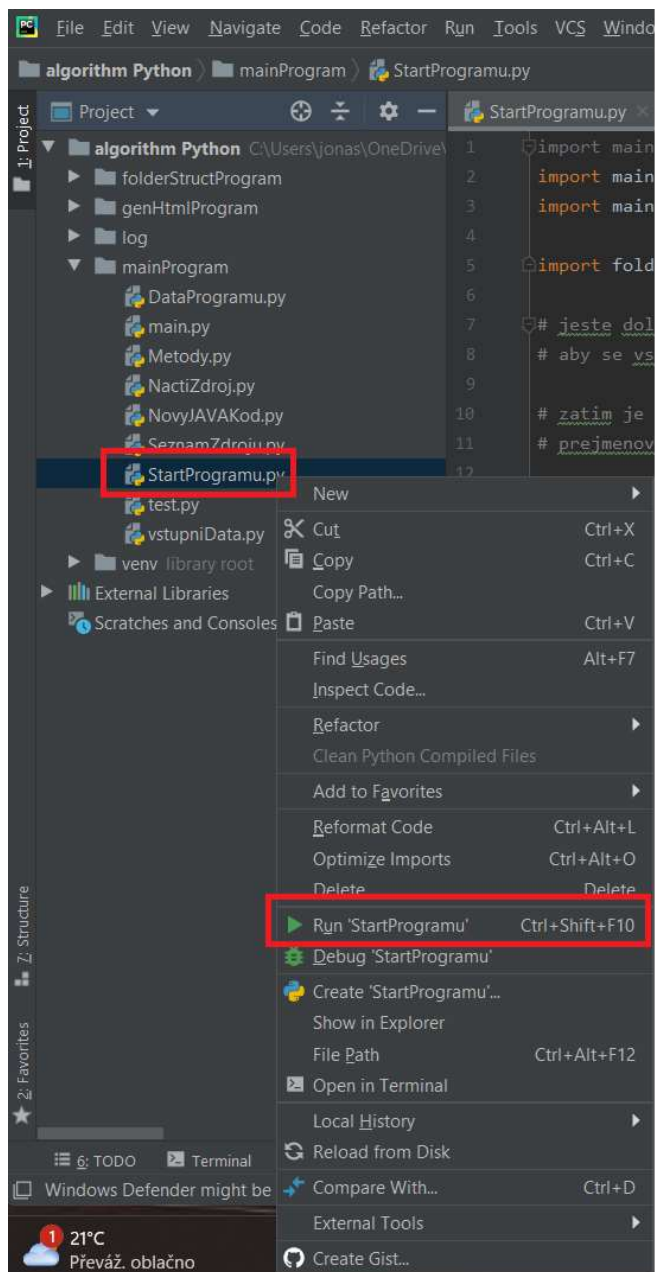
Otevřeme projekt v PyCharm.



Obr. 2 – otevření v PyCharm

## Rozvíjení zdrojového kódu do stromové struktury.

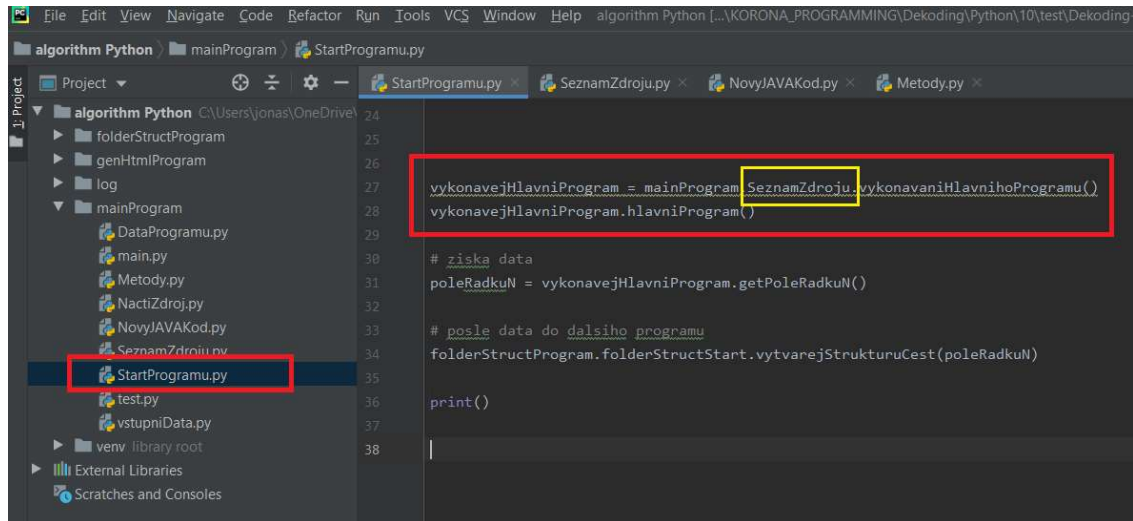
Projekt spustíme vybráním modulu StartProgramu.py a pak následně Run „StartProgramu“.



Obr. 3 – PyCharm – první start

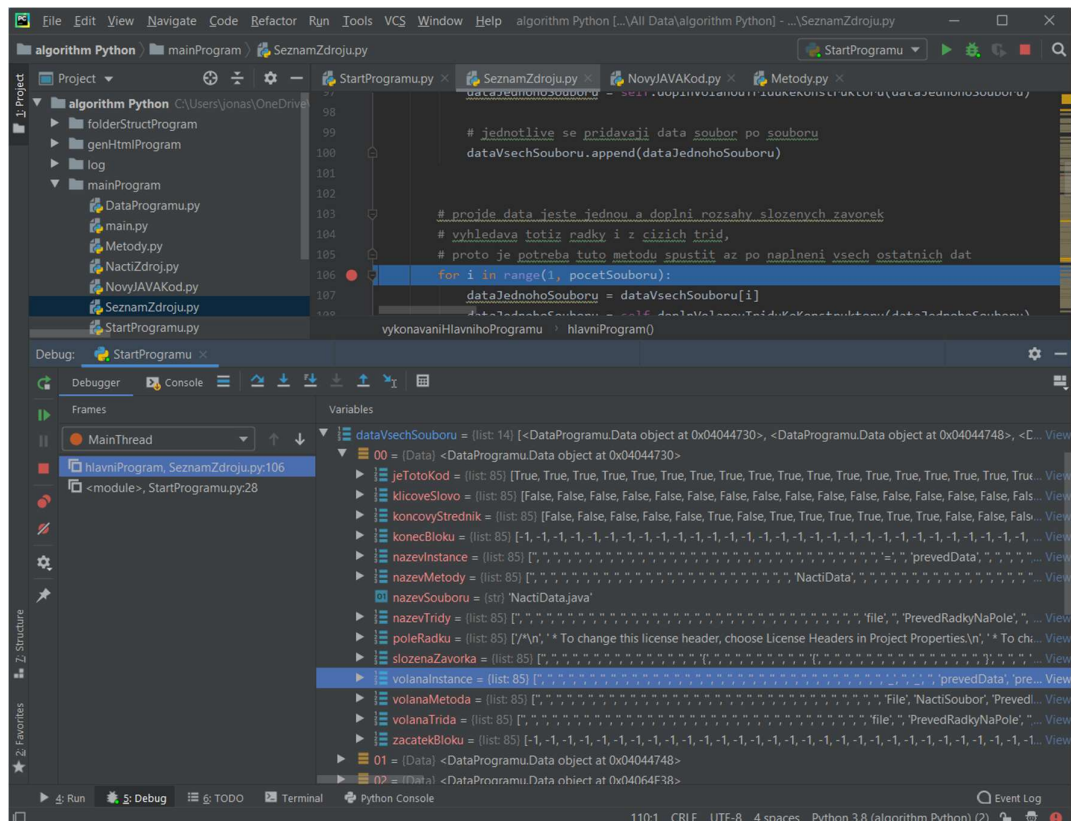
### 1.3. Start debugu.

Zdrojový kód začíná souborem StartProgramu.py v balíčku mainProgram.



Obr. 4 – Kód začíná v modulu StartProgrtamu.py v balíčku mainProgram.

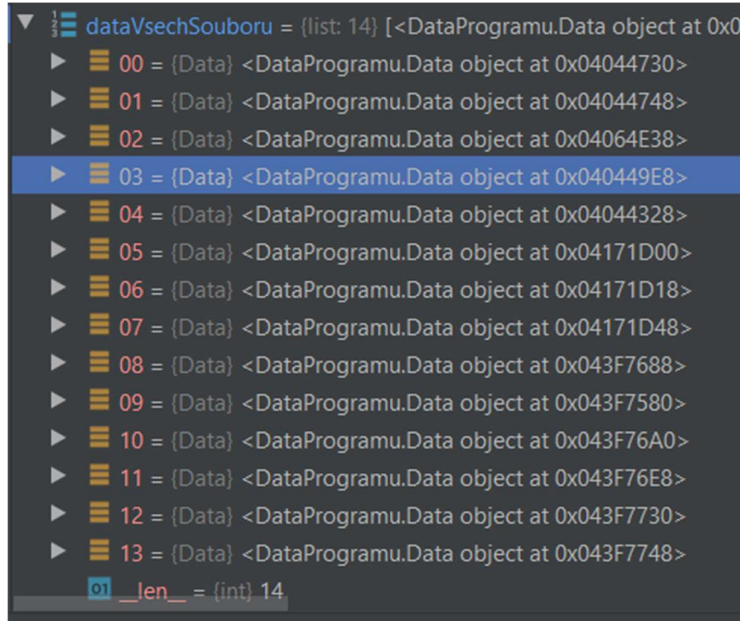
Umístíme následně breakpoint na řádek 106 v souboru SeznamZdroju.py.



Obr. 5 – Náhled na objekt **dataVsechSouboru**.

## Rozvíjení zdrojového kódu do stromové struktury.

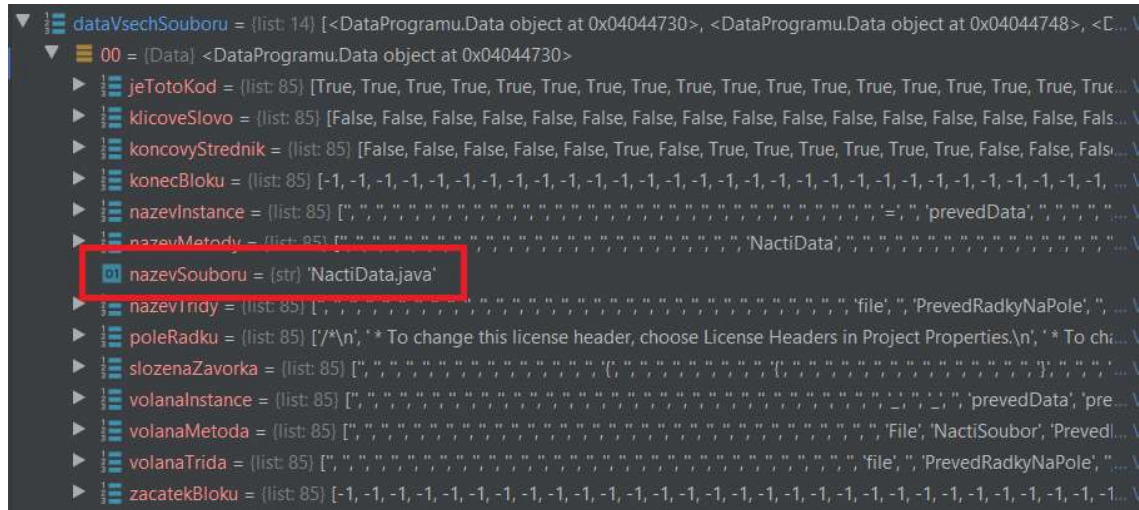
Níže si popíšeme objekt **dataVsechSouboru**.



Obr. 6 – Objekt **dataVsechSouboru** obsahuje 14 sad dat, pro 14 souborů java.

#### 1.4. Objekt DataVsechSouboru.

**DataVsechSouboru** obsahuje 14 položek objektu. Každá položka odpovídá jednomu java-souboru, např.



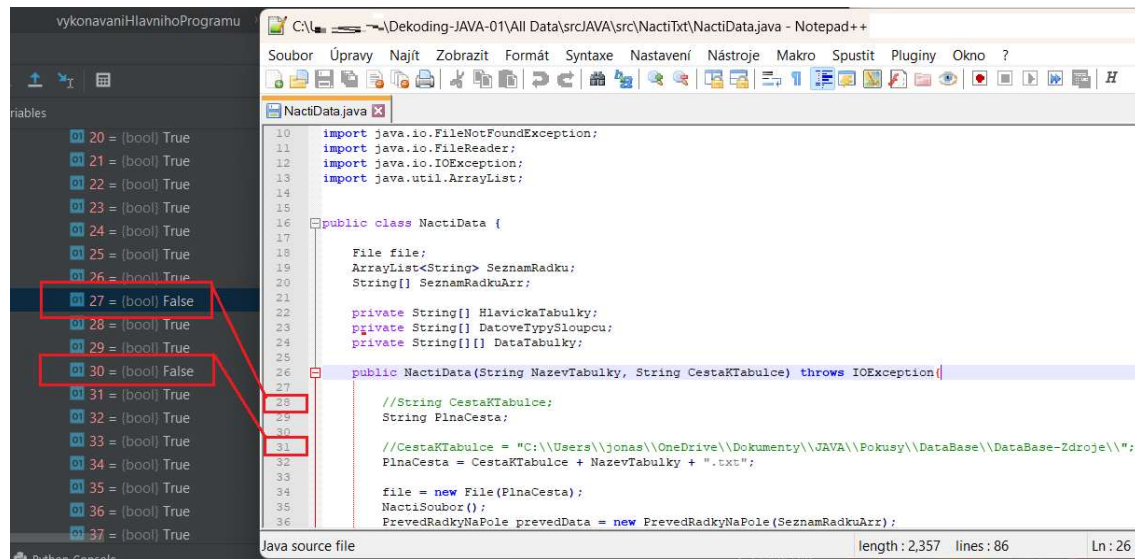
Obr. 7 – **nazevSouboru** v objektu **dataVsechSouboru**.



## Rozvíjení zdrojového kódu do stromové struktury.

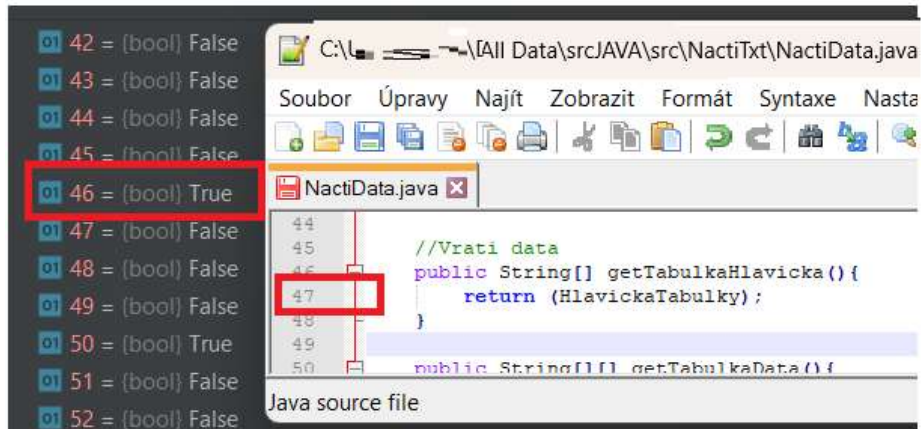
Níže si popíšeme jednotlivé pod-objekty.

**jeToKod** - pokud na daném řádku je False, pak se jedná o komentář, je-li True, pak se jedná o řádek bez komentáře. Např.



Obr. 8 – **jeToKod** v objektu **dataVsechSouboru**.

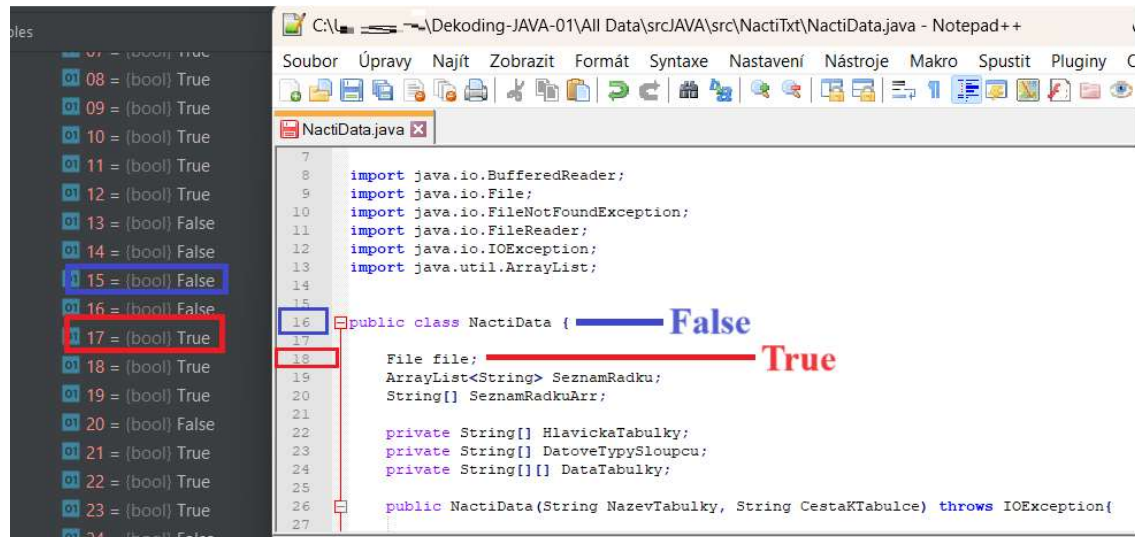
**klicoveSlovo** - je-li True, pak na řádku je klíčové slovo, např. **return**.



Obr. 9 – **klicoveSlovo** v objektu **dataVsechSouboru**.

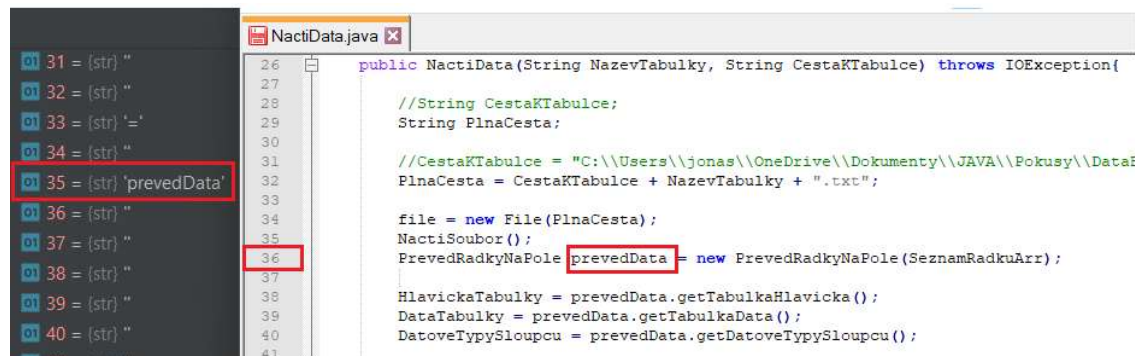
## Rozvíjení zdrojového kódu do stromové struktury.

**koncovyStrednik** - je-li True, pak řádek je zakončen „ ; “



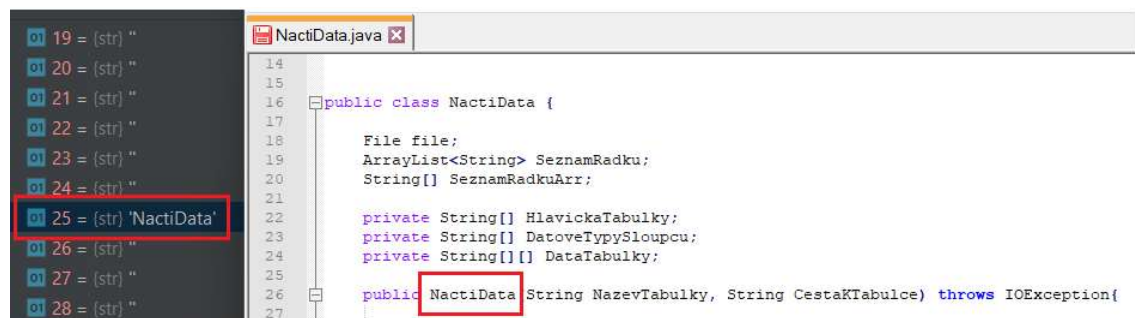
Obr. 10 – **koncovyStrednik** v objektu **dataVsechSouboru**.

### nazevInstance



Obr. 11 – **nazevInstance** v objektu **dataVsechSouboru**.

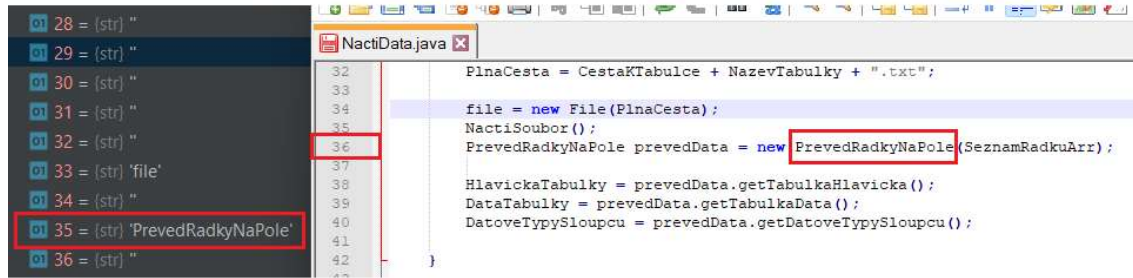
### nazevMetody



Obr. 12 – **nazevMetody** v objektu **dataVsechSouboru**.

## Rozvíjení zdrojového kódu do stromové struktury.

nazevTridy

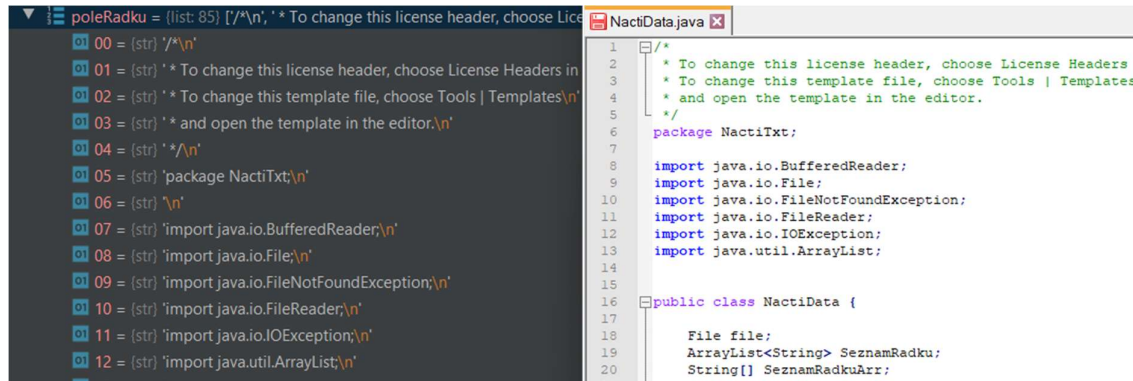


```
01 28 = {str} ""
01 29 = {str} ""
01 30 = {str} ""
01 31 = {str} ""
01 32 = {str} ""
01 33 = {str} 'file'
01 34 = {str} ""
01 35 = {str} 'PrevedRadkyNaPole'
01 36 = {str} ""
```

```
32 PlnaCesta = CestaKTabulce + NazevTabulky + ".txt";
33
34 file = new File(PlnaCesta);
35 NactiSoubor();
36 PrevedRadkyNaPole prevedData = new PrevedRadkyNaPole(SeznamRadkuArr);
37
38 HlavickaTabulky = prevedData.getTabulkaHlavicka();
39 DataTabulky = prevedData.getTabulkaData();
40 DatoveTypySloupce = prevedData.getDatoveTypySloupce();
41
42 }
```

Obr. 13 – **nazevTridy** v objektu **dataVsechSouboru**.

poleRadku – obsahuje seznam řádků celého souboru.

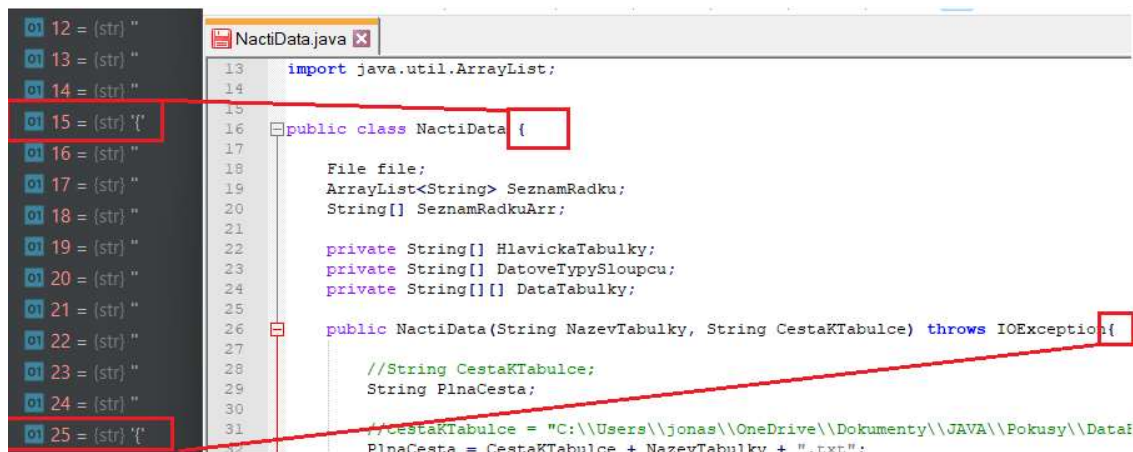


```
01 00 = {str} '/^\\n'
01 01 = {str} '* To change this license header, choose License Headers in
01 02 = {str} '* To change this template file, choose Tools | Templates\\n'
01 03 = {str} '* and open the template in the editor.\\n'
01 04 = {str} '*\\n'
01 05 = {str} 'package NactiTxt;\\n'
01 06 = {str} '\\n'
01 07 = {str} 'import java.io.BufferedReader;\\n'
01 08 = {str} 'import java.io.File;\\n'
01 09 = {str} 'import java.io.FileNotFoundException;\\n'
01 10 = {str} 'import java.io.FileReader;\\n'
01 11 = {str} 'import java.io.IOException;\\n'
01 12 = {str} 'import java.util.ArrayList;\\n'
```

```
1 /*
2  * To change this license header, choose License Headers
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6 package NactiTxt;
7
8 import java.io.BufferedReader;
9 import java.io.File;
10 import java.io.FileNotFoundException;
11 import java.io.FileReader;
12 import java.io.IOException;
13 import java.util.ArrayList;
14
15
16 public class NactiData {
17
18     File file;
19     ArrayList<String> SeznamRadku;
20     String[] SeznamRadkuArr;
```

Obr. 14 – **poleRadku** v objektu **dataVsechSouboru**.

slozenaZavorka – pokud je na řádku složená závorka, pak příslušná položka obsahuje pouze ji a žádný jiný text.



```
01 12 = {str} ""
01 13 = {str} ""
01 14 = {str} ""
01 15 = {str} '{'
01 16 = {str} ""
01 17 = {str} ""
01 18 = {str} ""
01 19 = {str} ""
01 20 = {str} ""
01 21 = {str} ""
01 22 = {str} ""
01 23 = {str} ""
01 24 = {str} ""
01 25 = {str} '{'
```

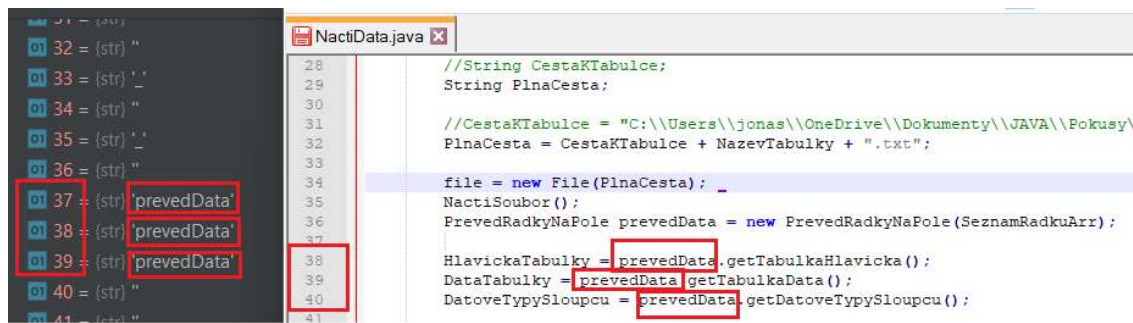
```
13 import java.util.ArrayList;
14
15
16 public class NactiData {
17
18     File file;
19     ArrayList<String> SeznamRadku;
20     String[] SeznamRadkuArr;
21
22     private String[] HlavickaTabulky;
23     private String[] DatoveTypySloupce;
24     private String[][] DataTabulky;
25
26     public NactiData(String NazevTabulky, String CestaKTabulce) throws IOException {
27
28         //String CestaKTabulce;
29         String PlnaCesta;
30
31         //CestaKTabulce = "C:\\Users\\jonas\\OneDrive\\Dokumenty\\JAVA\\Pokusy\\Data\\";
32         PlnaCesta = CestaKTabulce + NazevTabulky + ".txt";
```

Obr. 15 – **slozenaZavorka** v objektu **dataVsechSouboru**.



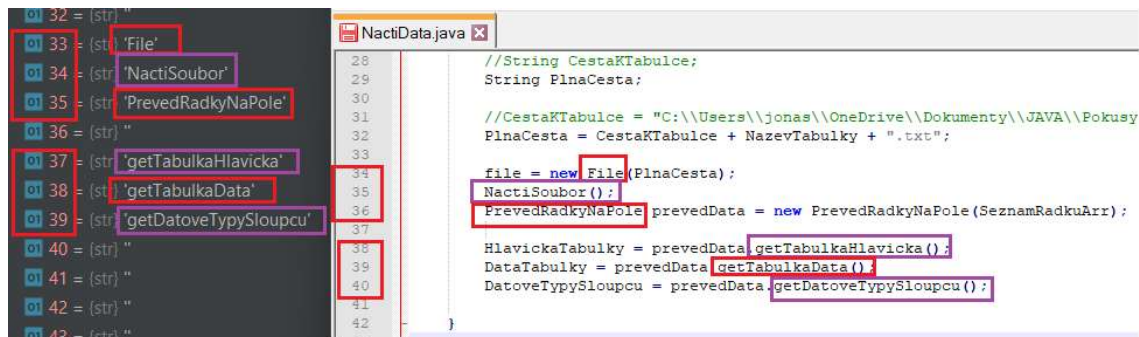
## Rozvíjení zdrojového kódu do stromové struktury.

### volanaInstance



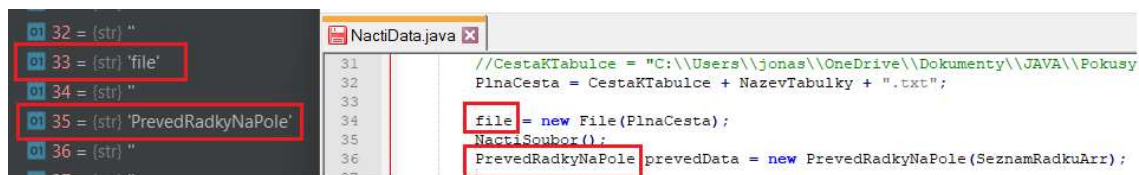
Obr. 16 – **volanaInstance** v objektu **dataVsechSouboru**.

### volanaMetoda



Obr. 17 – **volanaMetoda** v objektu **dataVsechSouboru**.

### volanaTrida



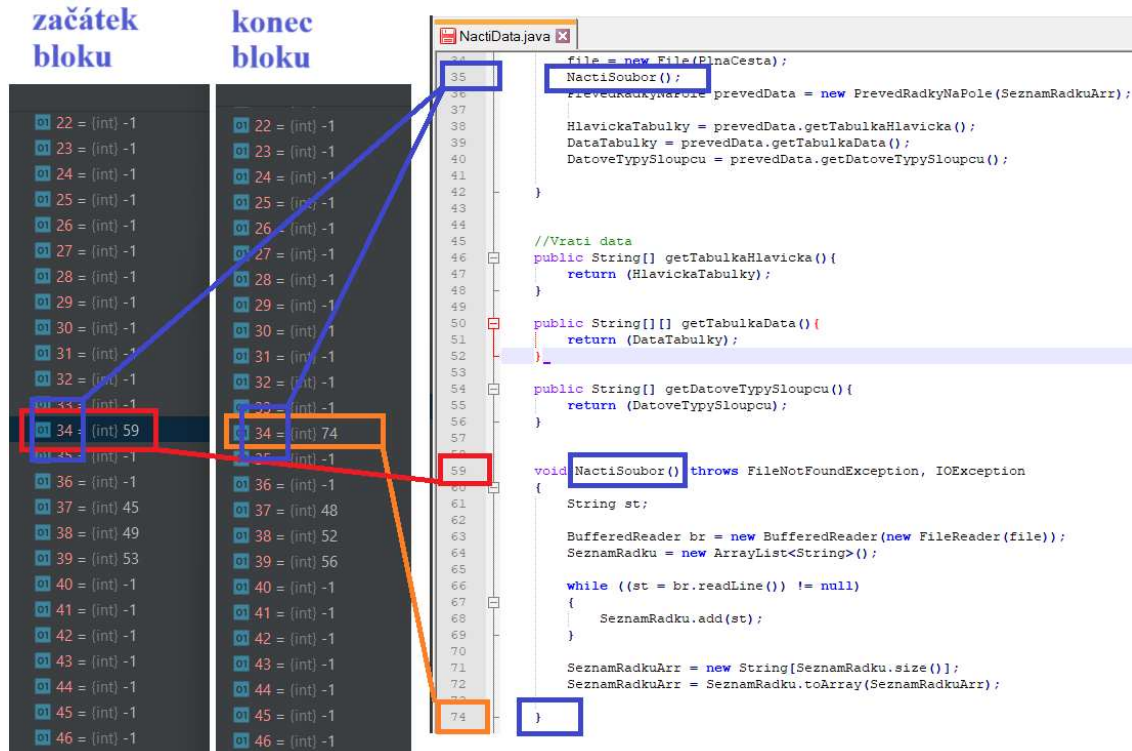
Obr. 18 – **volanaTrida** v objektu **dataVsechSouboru**.

**zacatekBloku, konecBloku** – udává začátek a konec bloku dané metody. V následujícím příkladu máme NactiSoubor na řádce 35 (tomu odpovídá index 34, jelikož číslujeme od nuly), tudíž:

**zacatekBloku** – obsahuje, na indexu 34, hodnotu 59, což udává index prvního indexu řádku metody NactiSoubor()

**konecBloku** – obsahuje, na indexu 34, hodnotu 74, což udává index posledního indexu řádku metody NactiSoubor()

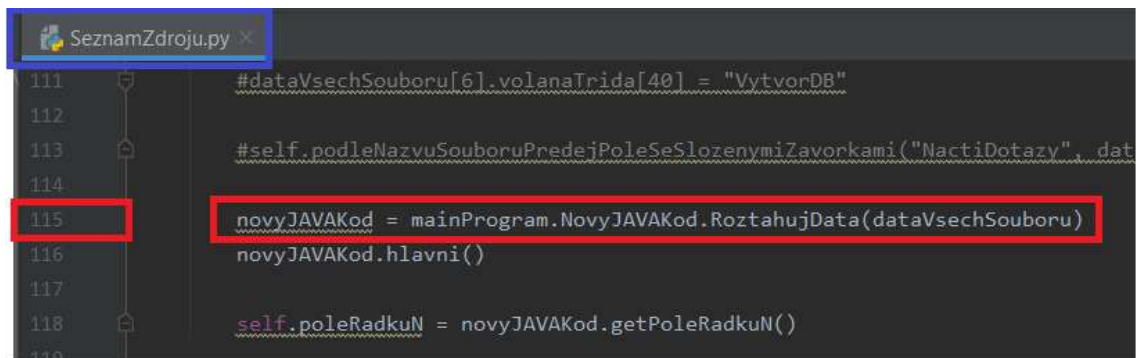
## Rozvíjení zdrojového kódu do stromové struktury.



Obr. 19 – začátekBloku , konecBloku v objektu dataVsechSouboru.

### 1.5. Vytváření kódu do výstupu txt.

Na řádku 115 voláme:



Obr. 20 – instance **NovyJAVAKod**

V této třídě nastavujeme manuálně, počínaje kterou požadovanou metodou si přejeme vytvářet stromovou strukturu.

## Rozvíjení zdrojového kódu do stromové struktury.

```
SeznamZdroju.py x NovyJAVAKod.py x
21
22 # zacatek programu v teto tride
23 def hlavni(self):
24
25     # nazev tridy, kde je umistena metoda, která se bude dokumentovat
26     pozadovanaTrida = "TextReader3.java"
27     pozadovanaTrida = "SQL_GUI_Frame.java"
28
29     # nazev metody, která se bude dokumentovat
30     nazevMetody = "main"
31     nazevMetody = "TlacDopravaActionPermed"
32
33     # pocatecni kod, který se bude teprve roztahovat
34     self.vratPocatecniKod(pozadovanaTrida, nazevMetody)
```

Obr. 21 – Výchozí metoda a třída se nastavují ručně.

Následně umístíme breakpoint na řádek 61.

```
# Ziska pole radku kodu dane dokumentovane metody v dane tride  
# Jedna se o originalni data - neroztazena  
KodPozadovaneMetody = ostatniMetody.nactiSubKod(vyjmiKodOdRadku, vyjmiKodDoRadku, 0, dataPozadovaneTridy.poleRadku) KodPozadovaneMetody:  
  
VolaneMetodyUvnitrMetodyPozadovane = ostatniMetody.nactiSubKod(vyjmiKodOdRadku, vyjmiKodDoRadku, 0, dataPozadovaneTridy.volanaMetoda)  
VolaneTridyUvnitrMetodyPozadovane = ostatniMetody.nactiSubKod(vyjmiKodOdRadku, vyjmiKodDoRadku, 0, dataPozadovaneTridy.volanaTrida)
```

RoztahujData | hlavni()

Obr. 22 – Veškerá data nalezneme v objektu `dataPozadovaneTridy`.

Zarámované proměnné vymezují indexy, mezi kterými se nachází metoda zobrazená na obr. 21.

**vyjmiKodOdRadku = 289**

**vyjmiKodDoRadku = 331**

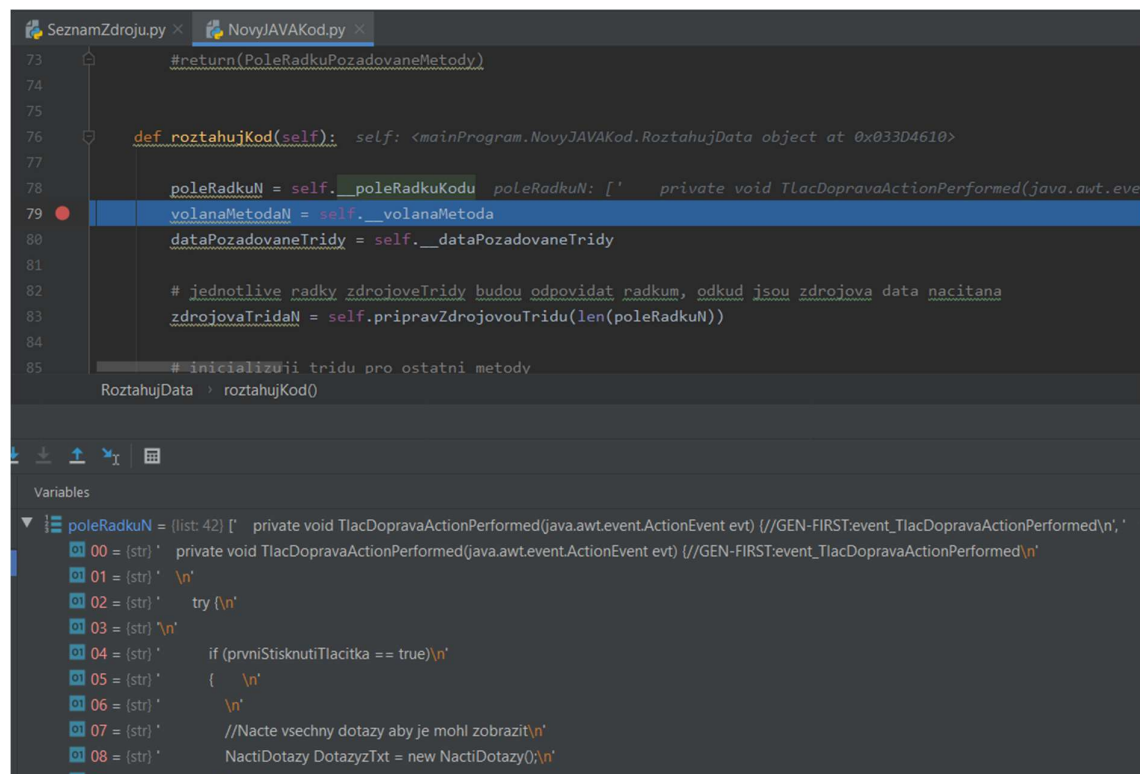
## Rozvíjení zdrojového kódu do stromové struktury.

Můžeme se přesvědčit, že `vyjmiKodOdRadku` je skutečně 289 (resp. 290 – počítáno od 1).



Obr. 23 – `vyjmiKodOdRadku` odpovídá indexu řádku, kde je nalezena metoda zadaná na obr. 21.

Následně umístíme breakpoint na řádek 79, `poleRadkuN` by měl obsahovat vyjmutá data. Délka pole je 42.

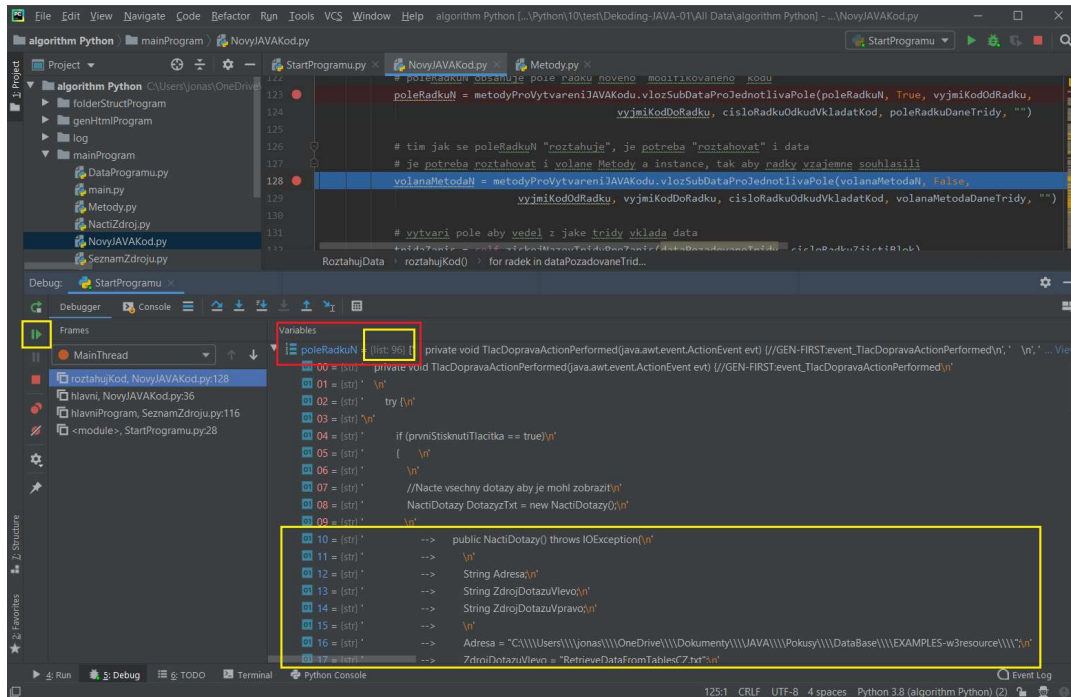


Obr. 24 – vyjmutá data mezi `vyjmiKodOdRadku` a `vyjmiKodDoRadku` = `poleRadkuN`.  
Zde: `poleRadkuN` v nultém kroku.

Umístíme breakpoint na řádek 128. Po zastavení kódu na tomto řádku se můžeme přesvědčit, že byla vložena data první vnořené úrovně. Pole bylo rozšířeno a nyní jeho délka je 96.



## Rozvíjení zdrojového kódu do stromové struktury.



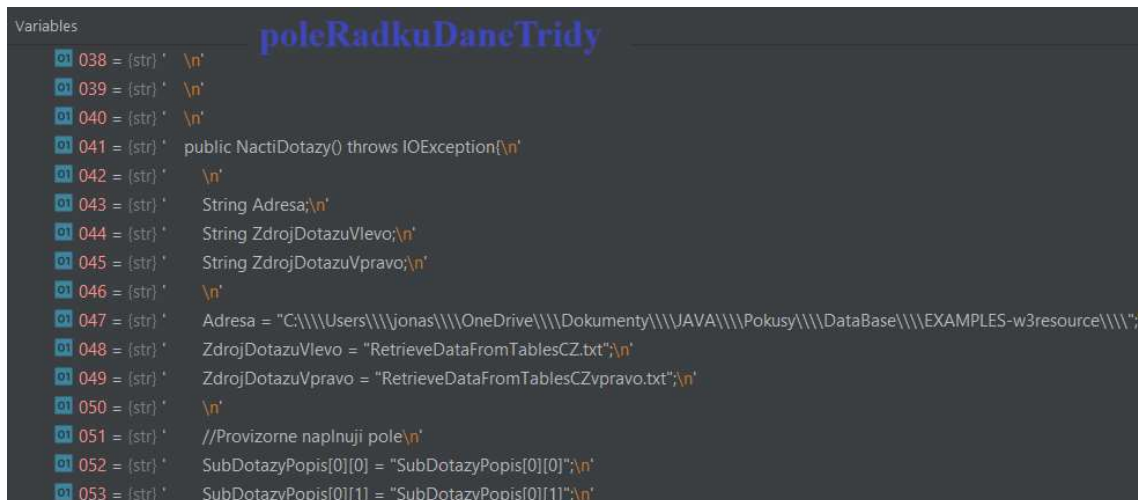
Obr. 25 – **poleRadkuN** v prvním kroku.

Též se můžeme přesvědčit, že:

```
01 vyjmiKodDoRadku = {int} 95
01 vyjmiKodOdRadku = {int} 41
```

Obr. 26 – **vyjmiKodOdRadku** a **vyjmiKodDoRadku** v prvním kroku.

Opravdu odpovídá:



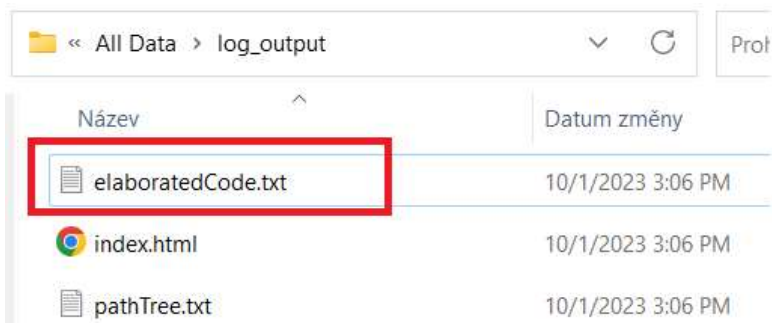
Obr. 27 – vyjmutá data mezi **vyjmiKodOdRadku** a **vyjmiKodDoRadku** v prvním kroku.






## Rozvíjení zdrojového kódu do stromové struktury.

Při opakovaném průchodem cykly, se můžeme přesvědčit, že poleRadkuN se postupně rozrůstá.

Při ponechání doběhnutí kódu do konce můžeme nalézt výsledný rozvinutý soubor.



« All Data » log_output		▼ ↺	Prost
Název	Datum změny		
 elaboratedCode.txt	10/1/2023 3:06 PM		
 index.html	10/1/2023 3:06 PM		
 pathTree.txt	10/1/2023 3:06 PM		

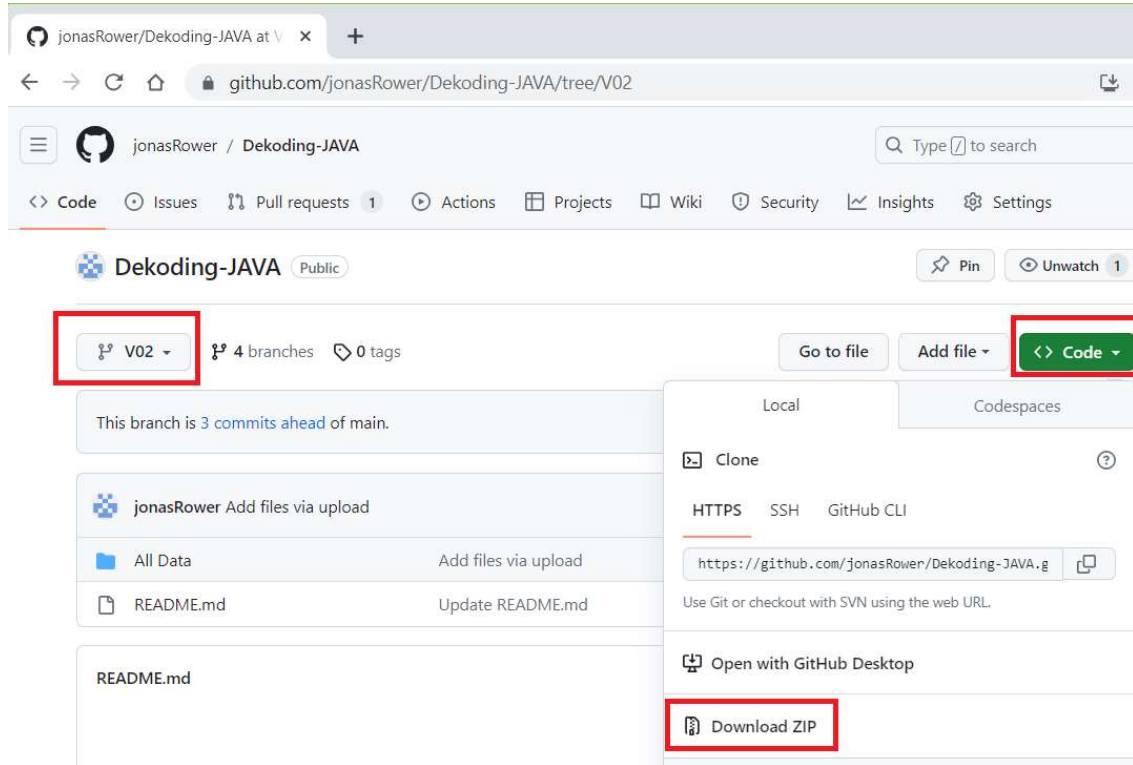
Obr. 28 – Vygenerované soubory.

Ostatními soubory se zde nebudeme zabývat, jelikož ve verzi V02 bude kód přestavěn.

## 2. Verze V02

### 2.1. Stažení verze V02

Verzi stáhneme obdobným způsobem, jako V01



Obr. 29 – Stažení verze V02.

### 2.2. První start V02.

Program spustíme, obdobně, jako V01 (StartProgramu.py).

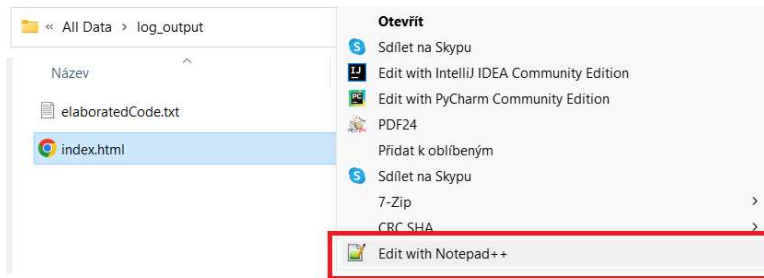
O úspěšnosti proběhnutí kódu se můžeme mimo jiné také přesvědčit tak, že ve složce AllData/log\_output nalezneme aktualizovaný soubor index.html, obsahující strom zdrojového kódu.



Obr. 30 – Vygenerované soubory.

Soubor index.html otevřeme v textovém editoru.

## Rozvíjení zdrojového kódu do stromové struktury.



Obr. 31 – Otevření index.html v textovém editoru.

Jehož obsah je následující:

```
index.html x
1 <!DOCTYPE html>
2 <html lang="en" xmlns="http://www.w3.org/1999/xhtml">
3 <head>
4 <meta charset="utf-8" />
5 <title>Simple jsTree</title>
6 <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/jstree/3.2.1/themes/default.min.css">
7 <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/1.12.1/jquery.min.js"></script>
8 <script src="https://cdnjs.cloudflare.com/ajax/libs/jstree/3.2.1/jstree.min.js"></script>
9
10 <script type="text/javascript">
11 $(function () {
12
13     var jsonData = [
14         { "id": "0", "parent": "#", "text": "private void TlacDopravaActionPerform",
15         { "id": "1", "parent": "0", "text": "    " },
16         { "id": "2", "parent": "1", "text": "        try { " },
17         { "id": "3", "parent": "1", "text": "            " },
18         { "id": "4", "parent": "1", "text": "                if (prvniStisknutiTlacitka ==",
19         { "id": "5", "parent": "4", "text": "                    { " },
20         { "id": "6", "parent": "4", "text": "                        " },
21         { "id": "7", "parent": "4", "text": "                            //Nacte vsechny dotazy al",
22         { "id": "8", "parent": "4", "text": "                                NactiDotazy DotazyzTxt =",
23         { "id": "9", "parent": "4", "text": "                                    " },
24         { "id": "10", "parent": "9", "text": "                                        public NactiDotazy() throws IOExcepti",
25         { "id": "11", "parent": "9", "text": "                                            " },
26         { "id": "12", "parent": "9", "text": "                                                String Adresa;" },
27         { "id": "13", "parent": "9", "text": "                                                    String ZdrojDotazuVlevo;" },
```

Obr. 32 – Obsah index.html.

### 2.3. Implementovaný kód V02.

Verzi V01 jsme rozšířili o generování js-tree. Počáteční algoritmus je implementován přímo do metody **roztahujKod**, která je v souboru **NovyJavaKod.java**.

Umístíme breakpoint na řádek 101, pusťme a zastavme kód zde. Získali jsme poleId, které vypadá následovně:

## Rozvíjení zdrojového kódu do stromové struktury.

The screenshot shows a Visual Studio Code editor window with a C# file named `RoztahujData.cs`. The code is as follows:

```
95  
96     # vytvori poles id  
97     poleId = poleIdClass.getPoleId() poleId: [-1, -1, 1, 1, 1, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,  
98  
99 |  
100     # nasledne jde program smyckou a postupne roztahuje data  
101     r = -1  
102     for radek in dataPozadovaneTridy.poleRadku:  
        RoztahujData > roztahujKod()
```

The variable explorer at the bottom shows the following variables:

- `ostatniMetody`: [UstatniMetody] <mainProgram.Metody.UstatniMetody object at 0x0000021A65149AC0>
- `poleId`: ([list: 42] [-1, -1, 1, 1, 1, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 38, 38, 40])

The `poleId` variable is expanded, showing its elements:

- 00 = (int) -1
- 01 = (int) -1
- 02 = (int) 1
- 03 = (int) 1
- 04 = (int) 1
- 05 = (int) 4
- 06 = (int) 4
- 07 = (int) 4
- 08 = (int) 4
- 09 = (int) 4
- 10 = (int) 4
- 11 = (int) 4
- 12 = (int) 4
- 13 = (int) 4
- 14 = (int) 4
- 15 = (int) 4

Obr. 33 – **poleId** v nultém cyklu.

Délka pole je rovna 42. Porovnáme-li s obr. 24, dospějeme k závěru, že délka pole je stejná. Každá položka na daném indexu objektu **poleRadkuN** odpovídá položce objektu **poleId**.

Následně umístíme breakpoint na řádek 135 a zastavme kód zde.

## Rozvíjení zdrojového kódu do stromové struktury.

[illegible]

Obr. 34 – **poleld** v prvním cyklu.

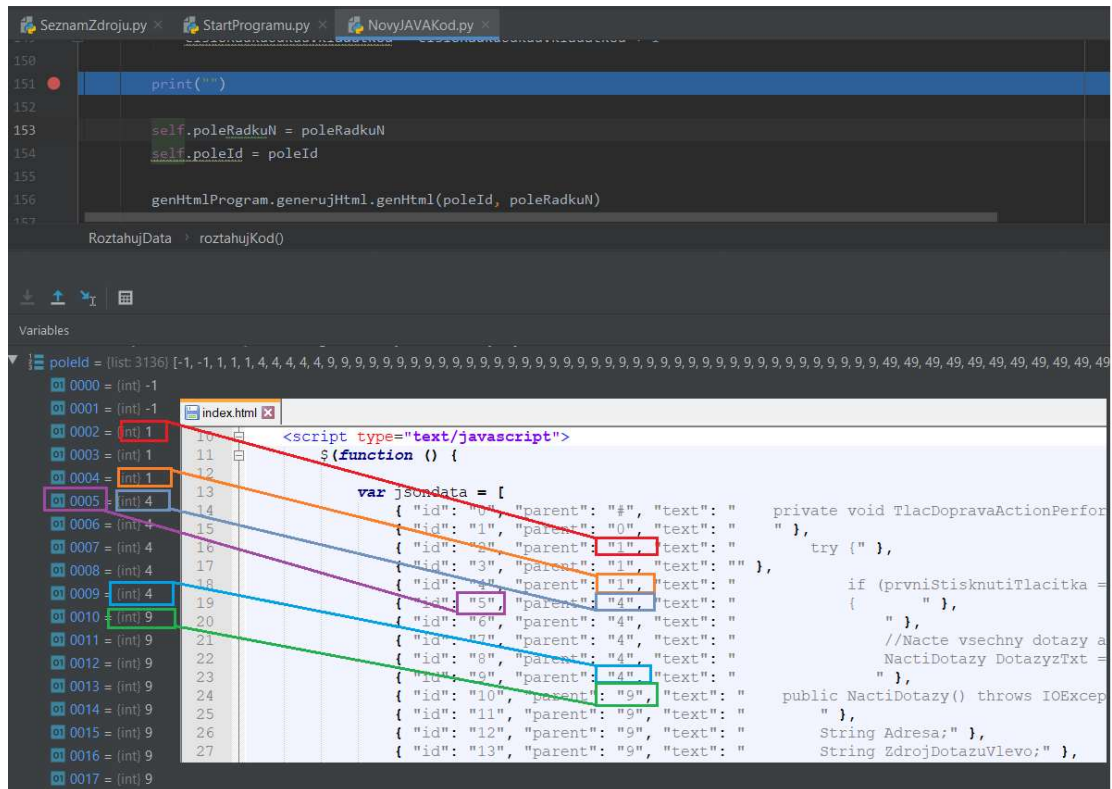
Délka pole je zde rozšířena na hodnotu 96. Stejně, jako na obr. 25.

Kód běží obdobným mechanismem, jako ve verzi V01 (obr. 25). Objekt **cisloRadkuOdkudVkladatKod** udává index, odkud se vkládá kód.

Necháme doběhnout kód do poslední smyčky, tj. umístíme breakpoint na řádek 151. Obrázek níže porovnává proměnnou poleld a soubor index.html. Je tedy zřejmé, že indexy poleld odpovídají hodnotám klíčů „id“ a hodnoty v poleld odpovídají hodnotám klíčů „parent“.



## Rozvíjení zdrojového kódu do stromové struktury.

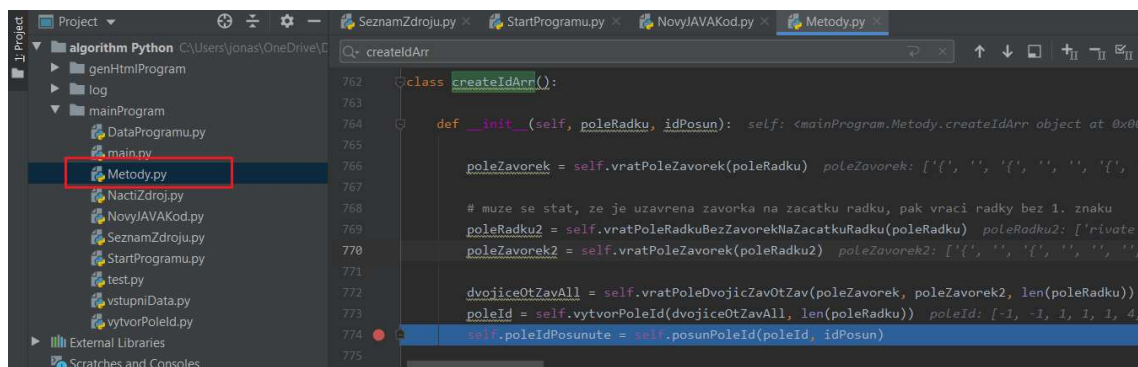


Obr. 35 – Porovnání **poleId** s daty souboru **index.html**.

## 2.4. Třída createIdArr():

S každou proběhnutou smyčkou, (ilustrovanou na obr. xxx) je třeba vytvořit nové poleld, které je posléze vkládáno na index **cisloRadkuOdkudVkladatKod**.

O to se stará právě třída `createIdArr()`. Umístěte tedy breakpoint na řádek 774 v modulu `Metody.py`. Restartujeme debug.

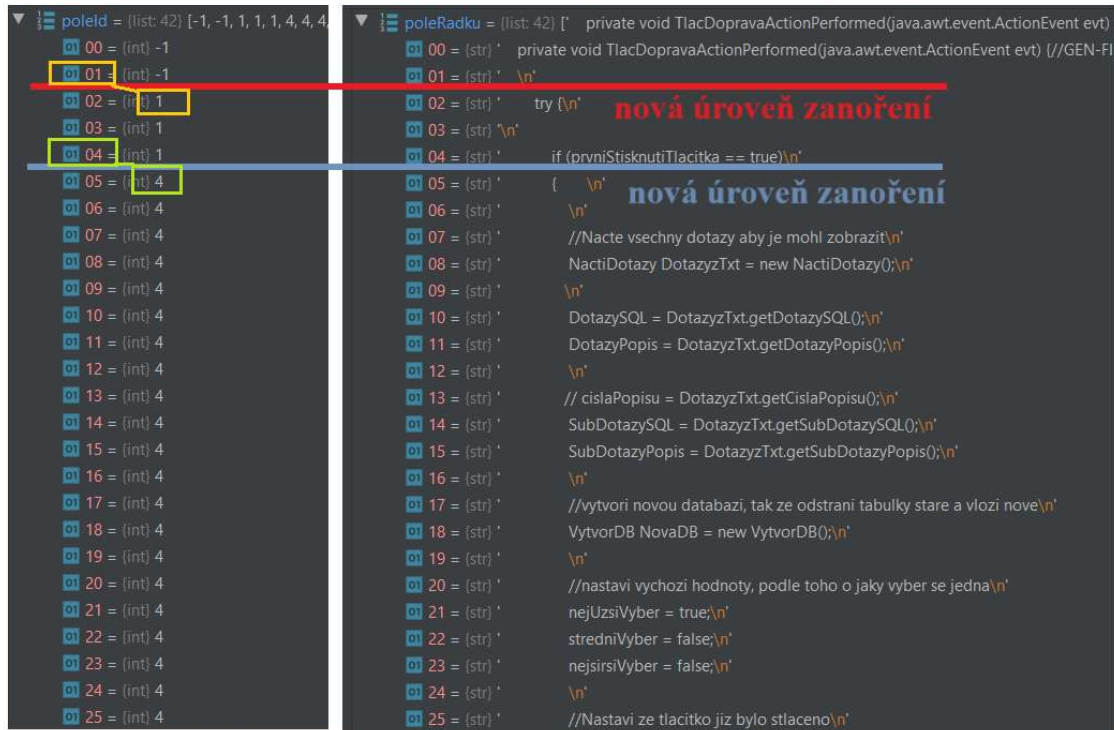


Obr. 36 – Constructor třídy `createIdArr()`.

## Rozvíjení zdrojového kódu do stromové struktury.

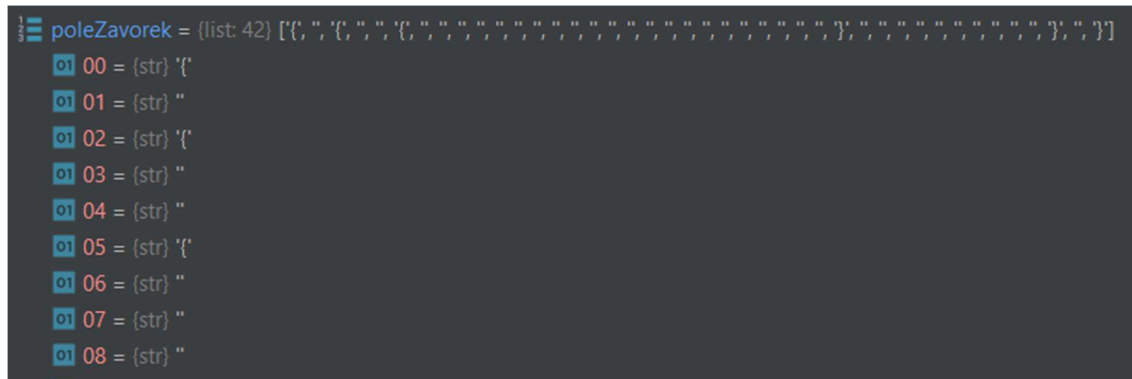
V objektech **poleId** a **poleRadku** můžeme nalézt souvislost:

**jednotlivé úrovně jsou dány pomocí znaků '{' a '}'**



Obr. 37 – Význam hodnot v **poleld.**

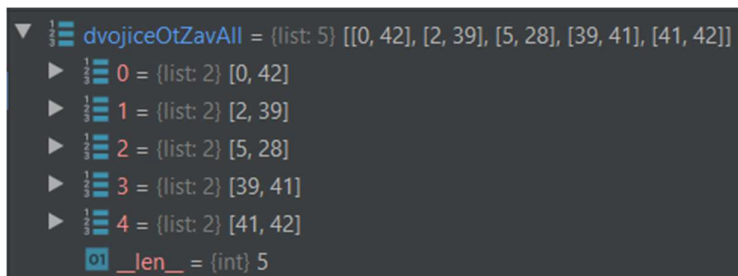
Objekt **poleZavorek** zobrazuje znaky ‘{’ nebo ‘}’ po jednotlivých řádcích.



Obr. 38 – Objekt **poleZavorek**.

## Rozvíjení zdrojového kódu do stromové struktury.

Taktéž si můžeme povšimnout objektu **dvojiceOtZavAll**:



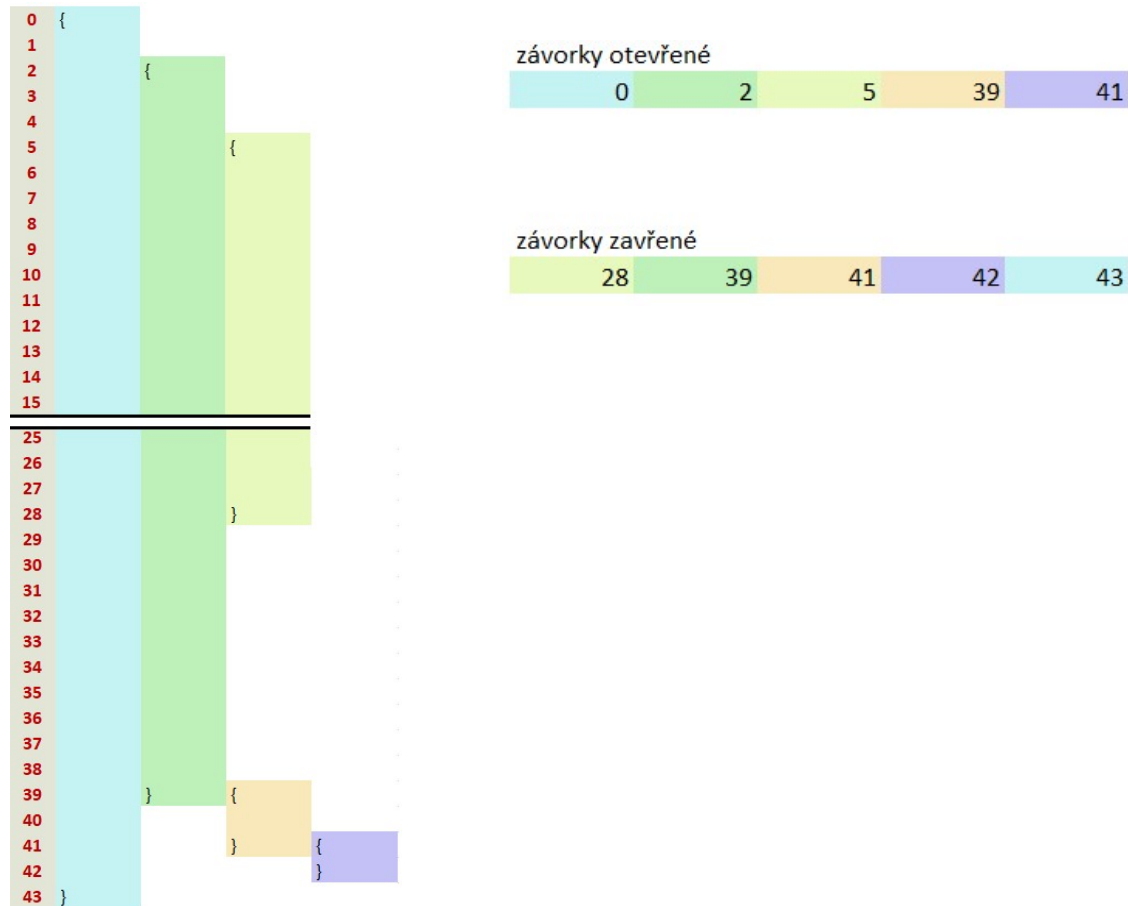
Obr. 39 – Objekt **dvojiceOtZavAll**.

**dvojiceOtZavAll** tedy udává dvojice indexů řádku na otevřené a zavřené složené závorky.

## 2.5. Algoritmus rozpoznávání zanoření závorek – teorie.

Je tedy třeba rozpoznat která dvojice indexů otevřených a zavřených závorek je zanořena do té které jiné. Algoritmus si popíšeme níže:

Obrázek níže zachycuje barevné označení intervalů jednotlivých závorek



Obr. 39 – Grafické znázornění složených závorek na jednotlivých řádcích pole.

Obrázek níže zachycuje rozdíly indexů všech souřadnic, tedy:

Index závorky zavřené – index závorky otevřené.

0	2	5	39	41	
28	26	23	-11	-13	28
39	37	34	0	-2	39
41	39	36	2	0	41
42	40	37	3	1	42
43	41	38	4	2	43

Obr. 40 – Rozdíly indexů závorek – nultý cyklus.

## Rozvíjení zdrojového kódu do stromové struktury.

Tedy konkrétně, položka o hodnotě 28 je **index závorky zavřené 28** – **index závorky otevřené 0**.

$$28 - 0 = 28$$

Vzhledem k tomu, že rozdíly indexů souřadnic musí být  $> 0$  pak všechny záporné a nulové zneplatníme a nebudeme je v následujícím výpočtu již uvažovat. Na obrázku níže jsou barevně označeny.

0	2	5	39	41	
28	26	23	-11	-13	28
39	37	34	0	-2	39
41	39	36	2	0	41
42	40	37	3	1	42
43	41	38	4	2	43

Obr. 41 – Rozdíly indexů závorek – označení neplatných hodnot – nultý cyklus.

V následujícím výpočtu vyhledáme minimální hodnotu, tu zde zvýrazníme **tučně červeně**.

0	2	5	39	41	
28	26	23	-11	-13	28
39	37	34	0	-2	39
41	39	36	2	0	41
42	40	37	3	<b>1</b>	42
43	41	38	4	2	43

Obr. 42 – Rozdíly indexů závorek – nalezení nejmenší hodnoty – 1. cyklus.

Námi minimální hodnota odpovídá minimálnímu rozdílu indexů závorek. A jedná se tedy o dvojici indexů závorek 41 a 42. Následně zneplatníme všechny hodnoty na daném řádku a sloupci této minimální hodnoty.

0	2	5	39	41		
28	26	23	-11	-13	28	
39	37	34	0	-2	39	
41	39	36	2	0	41	
42	40	37	3	<b>1</b>	42	41 42
43	41	38	4	2	43	

Obr. 43 – Rozdíly indexů závorek – označení neplatných hodnot – 1. cyklus.



## Rozvíjení zdrojového kódu do stromové struktury.

V dalším kroku postupujeme stejně. Opět vyhledáme minimální hodnotu. Tou je hodnota **2**. Indexy závorek jsou 39 a 41. Současně zneplatníme hodnoty na daném řádku a sloupci ve kterém je hodnota **2**. Tím zajistíme, že opět v dalším kroku již nemůžeme vyhledávat minimální hodnotu stejného sloupce či řádku.

0	2	5	39	41		
28	26	23	-11	-13	28	
39	37	34	0	-2	39	
41	39	36	<b>2</b>	0	41	39 41
42	40	37	3	1	42	
43	41	38	4	2	43	

Obr. 43 – Rozdíly indexů závorek – nalezení nejmenší hodnoty – 2. cyklus.

V dalším kroku je minimální hodnota 23. To odpovídá dvojici indexů závorek 5 a 28.

0	2	5	39	41		
28	26	<b>23</b>	-11	-13	28	5 28
39	37	34	0	-2	39	
41	39	36	2	0	41	
42	40	37	3	1	42	
43	41	38	4	2	43	

Obr. 44 – Rozdíly indexů závorek – nalezení nejmenší hodnoty – 3. cyklus.

V dalším kroku je minimální hodnota 37. Indexy závorek 2 a 39.

0	2	5	39	41		
28	26	23	-11	-13	28	
39	<b>37</b>	34	0	-2	39	2 39
41	39	36	2	0	41	
42	40	37	3	1	42	
43	41	38	4	2	43	

Obr. 45 – Rozdíly indexů závorek – nalezení nejmenší hodnoty – 4. cyklus.

Zbývá nám poslední hodnota 43, odpovídající indexům řádků 0 , 43.

0	2	5	39	41		
28	26	23	-11	-13	28	
39	37	34	0	-2	39	
41	39	36	2	0	41	
42	40	37	3	1	42	
43	41	38	4	2	43	0 43

Obr. 45 – Rozdíly indexů závorek – nalezení poslední hodnoty – 5. cyklus.

### 2.6. Mírná odlišnost od teorie.

Můžeme nalézt mírnou neshodu mezi indexy závorek zde teoreticky vypočítanými a obr. 39. Obr. 39 udává dvojice indexů [0, 42], namísto [0, 43].

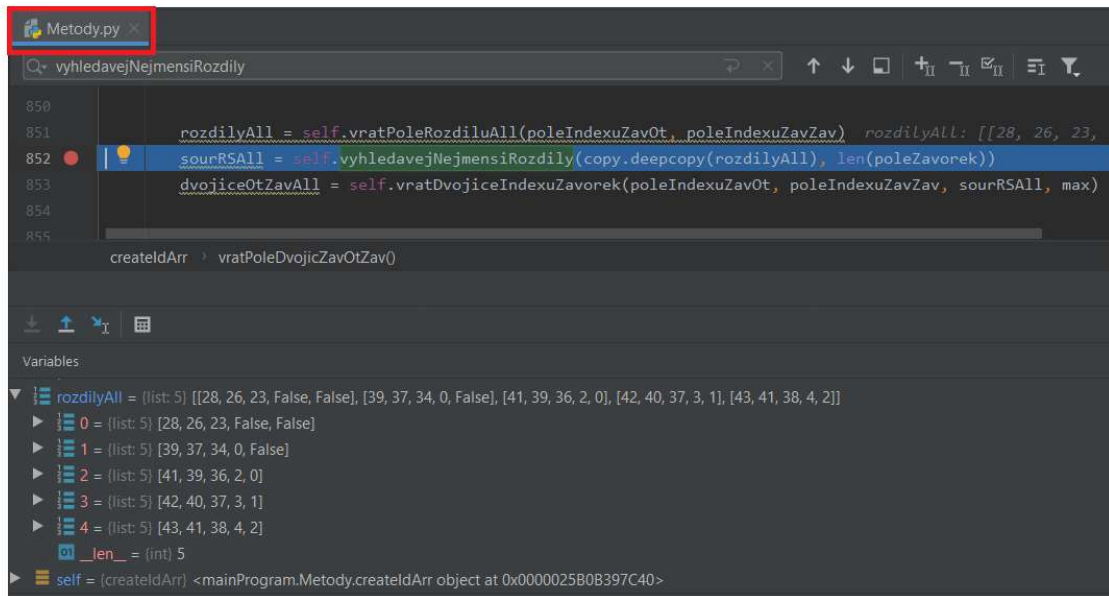
Opravami zdrojového kódu v této verzi se již nezabýváme (bude ve verzi příští). Pokud si detailně prohlédneme objekt **poleRadkuN** (obr. 24), můžeme nalézt chybu, že poslední řádek je:

```
} catch (ClassNotFoundException ex) {
```

Tudíž pravděpodobně je někde chyba v indexaci a není vyjmut kód celý. Verze V02 sice neopravuje tuto chybu, avšak všechny neuzavřené závorky doplňuje. Jelikož je délka pole rovna 42, přepisujeme interval mezi otevřenou a zavřenou závorkou na [0, 42].

## 2.7. Algoritmus rozpoznávání zanoření závorek – zdrojový kód.

Umístíme breakpoint na řádek 852 a 964 v modulu Metody.py. Restartujeme debug.



```

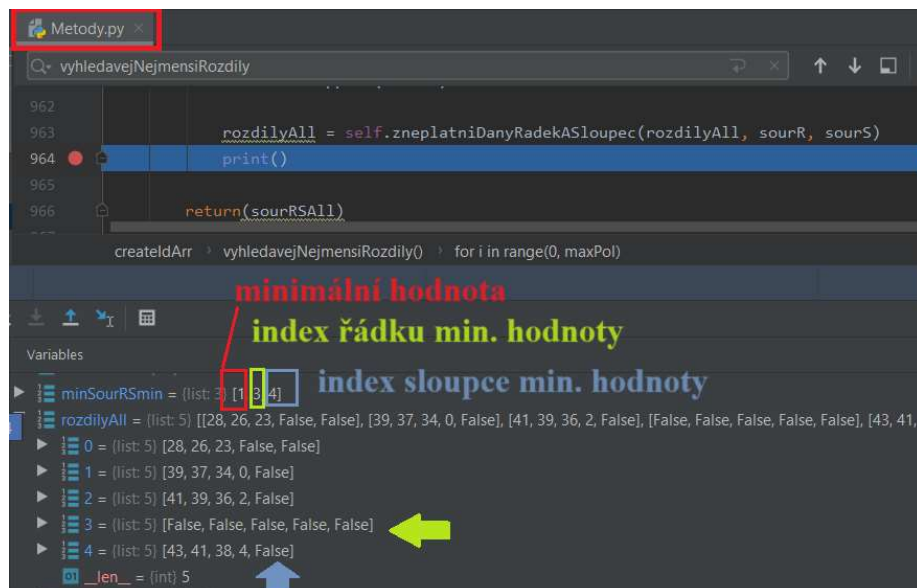
850
851 rozdilyAll = self.vratPoleRozdiluAll(poleIndexuZavOt, poleIndexuZavZav) rozdilyAll: [[28, 26, 23, f
852 sourRSAll = self.vyhledavejNejmensiRozdily(copy.deepcopy(rozdilyAll), len(poleZavorek))
853 dvojiceOtZavAll = self.vratDvojiceIndexuZavorek(poleIndexuZavOt, poleIndexuZavZav, sourRSAll, max)
854
855
createdArr → vratPoleDvojicZavOtZav()

Variables
rozdilyAll = (list: 5) [[28, 26, 23, False, False], [39, 37, 34, 0, False], [41, 39, 36, 2, 0], [42, 40, 37, 3, 1], [43, 41, 38, 4, 2]]
0 = (list: 5) [28, 26, 23, False, False]
1 = (list: 5) [39, 37, 34, 0, False]
2 = (list: 5) [41, 39, 36, 2, 0]
3 = (list: 5) [42, 40, 37, 3, 1]
4 = (list: 5) [43, 41, 38, 4, 2]
_len_ = (int) 5
self = (createdArr) <mainProgram.Metody.createdArr object at 0x0000025B0B397C40>
    
```

Obr. 46 – Hodnoty objektu **rozdilyAll** v nultém cyklu.

Můžeme se přesvědčit, že hodnoty **rozdilyAll** odpovídají s teoretickým výpočtem v nultém cyklu (obr. 41)

Krokujeme kód na řádek 964.



```

962
963 rozdilyAll = self.zneplatniDanyRadekASloupec(rozdilyAll, sourR, sourS)
964 print()
965
966 return(sourRSAll)
createdArr → vyhledavejNejmensiRozdily() → for i in range(0, maxPol)

Variables
minSourRSmin = (list: 3) [1 3 4]
rozdilyAll = (list: 5) [[28, 26, 23, False, False], [39, 37, 34, 0, False], [41, 39, 36, 2, False], [False, False, False, False, False], [43, 41, 38, 4, 2]]
0 = (list: 5) [28, 26, 23, False, False]
1 = (list: 5) [39, 37, 34, 0, False]
2 = (list: 5) [41, 39, 36, 2, False]
3 = (list: 5) [False, False, False, False, False]
4 = (list: 5) [43, 41, 38, 4, False]
_len_ = (int) 5
    
```

minimální hodnota  
index řádku min. hodnoty  
index sloupce min. hodnoty

Obr. 46 – Hodnoty objektu **rozdilyAll** v prvním cyklu.

## Rozvíjení zdrojového kódu do stromové struktury.

Jednotlivými kroky se můžeme přesvědčit, že výpočet je v souladu s teoretickým postupem.

Při umístění breakpointu na řádek 856 si můžeme prohlédnout data v objektu **dvojiceOtZavAll**, zobrazené na obr. 39.

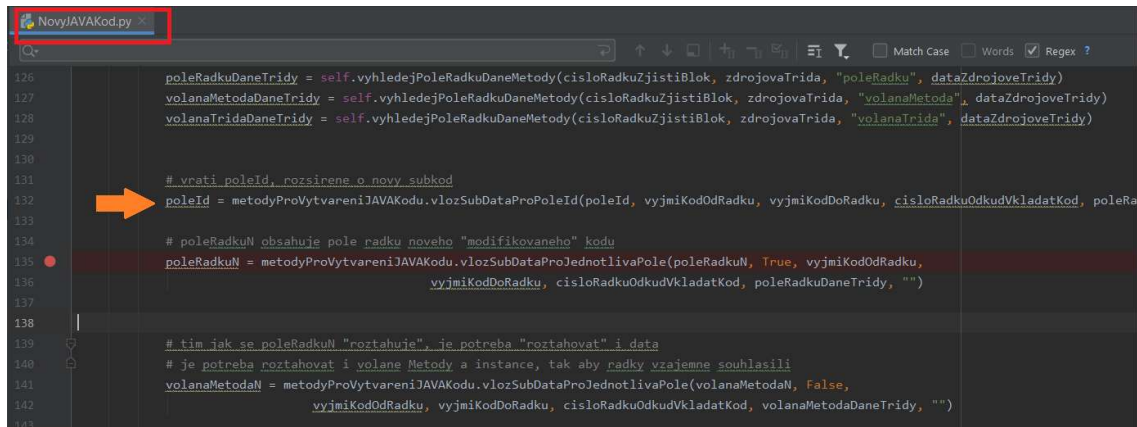
V této fázi již není problém vytvořit poleld. Obrázek níže ukazuje pozastavení kódu na řádce 774. Jedná se o stejná data, jako obr. 37.

[illegible]

Obr. 47 – Hodnoty objektu **rozdilyAll** v prvním cyklu.

## Rozvíjení zdrojového kódu do stromové struktury.

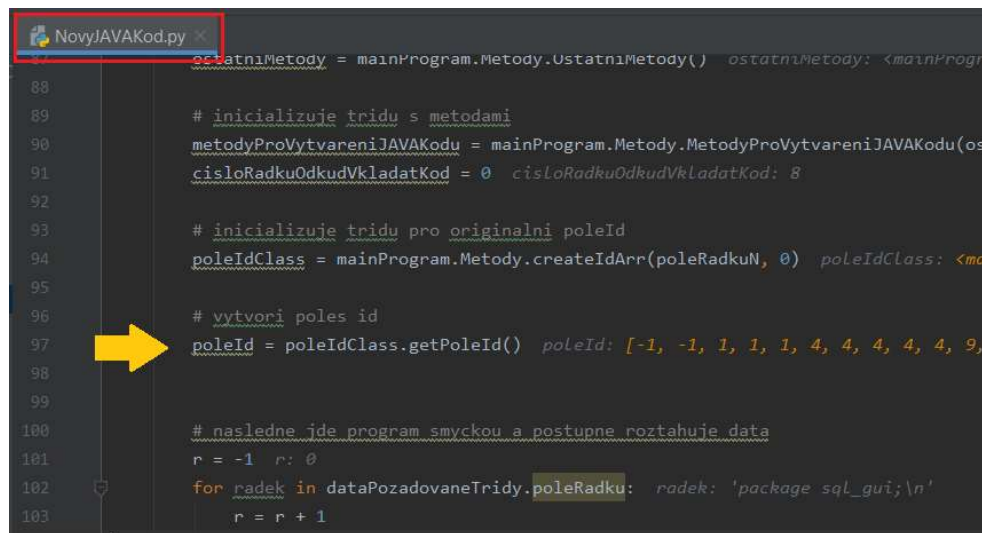
**PoleId** tedy vytváříme v třídě **createIdArr()**, kterou inicializujeme v každém cyklu. Data v každém cyklu získáváme zde:



```
126 poleRadkuDaneTridy = self.vyhledejPoleRadkuDaneMetody(cisloRadkuZjistiBlok, zdrojovaTrida, "poleRadku", dataZdrojoveTridy)
127 volanaMetodaDaneTridy = self.vyhledejPoleRadkuDaneMetody(cisloRadkuZjistiBlok, zdrojovaTrida, "volanaMetoda", dataZdrojoveTridy)
128 volanaTridaDaneTridy = self.vyhledejPoleRadkuDaneMetody(cisloRadkuZjistiBlok, zdrojovaTrida, "volanaTrida", dataZdrojoveTridy)
129
130
131 # vrati poleId, rozsiřene o nový subkod
132 poleId = metodyProVytvareniJAVAKodu.vlozSubDataProPoleId(poleId, vyjmiKodOdRadku, vyjmiKodDoRadku, cisloRadkuOdkudVkladatKod, poleRa
133
134 # poleRadkuN obsahuje pole radku nového "modifikovaného" kodu
135 poleRadkuN = metodyProVytvareniJAVAKodu.vlozSubDataProJednotlivuPole(poleRadkuN, True, vyjmiKodOdRadku,
136 vyjmiKodDoRadku, cisloRadkuOdkudVkladatKod, poleRadkuDaneTridy, "")
137
138
139 # tím jak se poleRadkuN "roztahuje", je potřeba "roztahovat" i data
140 # je potřeba roztahovat i volané Metody a instance, tak aby řádky vzájemně souhlasili
141 volanaMetodaN = metodyProVytvareniJAVAKodu.vlozSubDataProJednotlivuPole(volanaMetodaN, False,
142 vyjmiKodOdRadku, vyjmiKodDoRadku, cisloRadkuOdkudVkladatKod, volanaMetodaDaneTridy, "")
143
```

Obr. 48 – Získání objektu **poleId**. Porovnej s obr. 25.

Z kontextu vyplývá, že je třeba získat výchozí **poleId**. To se děje před začátkem cyklu.



```
ostatniMetody = mainProgram.Metody.UstatniMetody() ostatniMetody: <mainProgr
88
89 # inicializuje tridu s metodami
90 metodyProVytvareniJAVAKodu = mainProgram.Metody.MetodyProVytvareniJAVAKodu(os
91 cisloRadkuOdkudVkladatKod = 0 cisloRadkuOdkudVkladatKod: 8
92
93 # inicializuje tridu pro originalni poleId
94 poleIdClass = mainProgram.Metody.createIdArr(poleRadkuN, 0) poleIdClass: <ma
95
96 # vytvori poles id
97 poleId = poleIdClass.getPoleId() poleId: [-1, -1, 1, 1, 1, 4, 4, 4, 4, 4, 9,
98
99
100 # nasledne jde program smyčkou a postupne roztahuje data
101 r = -1 r: 0
102 for radek in dataPozadovaneTridy.poleRadku: radek: 'package sql_gui;\n'
103 r = r + 1
```

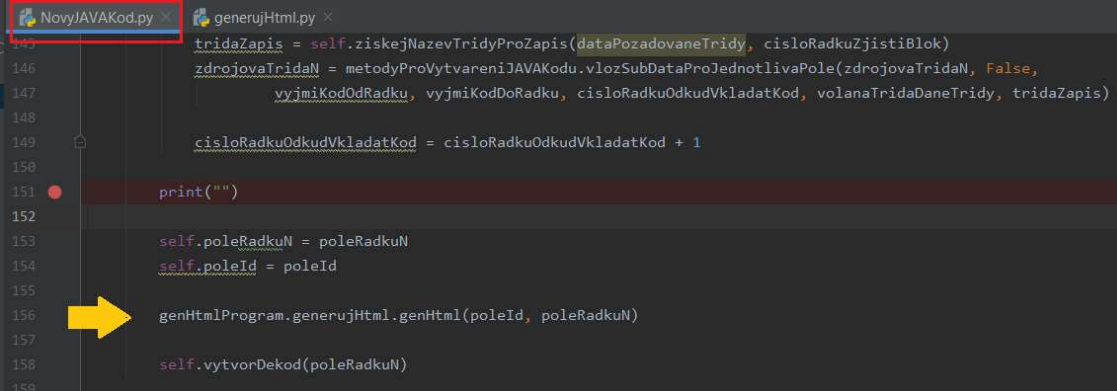
Obr. 49 – Získání objektu **poleId** – výchozí data.

Po průchodu všemi smyčkami, získáme **poleId** a **poleRadkuN** současně. Obrázek níže ukazuje kam odesíláme data. Problematika získání **poleRadkuN** je popsána ve verzi V01 (ve V02 nezměněna).

Význam dat **poleId** jsme objasnili na obr. 35.



## Rozvíjení zdrojového kódu do stromové struktury.



```
145 tridaZapis = self.ziskejNazevTridyProZapis(dataPozadovaneTridy, cisloRadkuZjistiBlok)
146 zdrojovaTridaN = metodyProVytvoreniJAVAKodu.vlozSubDataProJednotlivaPole(zdrojovaTridaN, False,
147 vyjmiKodOdRadku, vyjmiKodDoRadku, cisloRadkuOdkudVkladatKod, volanaTridaDaneTridy, tridaZapis)
148
149 cisloRadkuOdkudVkladatKod = cisloRadkuOdkudVkladatKod + 1
150
151 print("")
152
153 self.poleRadkuN = poleRadkuN
154 self.poleId = poleId
155
156 genHtmlProgram.generujHtml.genHtml(poleId, poleRadkuN)
157
158 self.vytvorDekod(poleRadkuN)
159
```

Obr. 50 – Odeslání dat pro vytvoření html souboru.