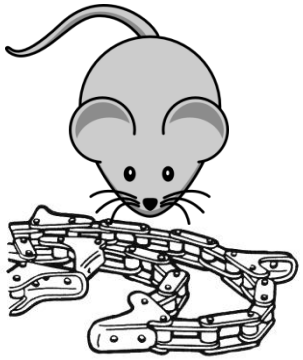
PAZ

4. prednáška (9.10.2023)

Reťazce a myšacie udalosti



Myši v reťaziach?

Čo už vieme...

- **Vytvoriť objekt** nejakej triedy pomocou **new**
  - vieme, že objekt môže mať viacero konštruktorov líšiacich sa parametrami (*WinPane*, *String*, …)
- **Vytvoriť vlastnú triedu** rozširujúcu triedu ***Turtle*** a popridávať do nej nové metódy
- Poznáme **podmienkový príkaz** (if-else)
- Poznáme **cykly**: **`for`**, **`while`**, **`do-while`**
  - **`break`** a **`continue`** na ich prerušenie
- Metódy vracajúce hodnoty a príkaz **`return`**
- Komentáre, debugovanie

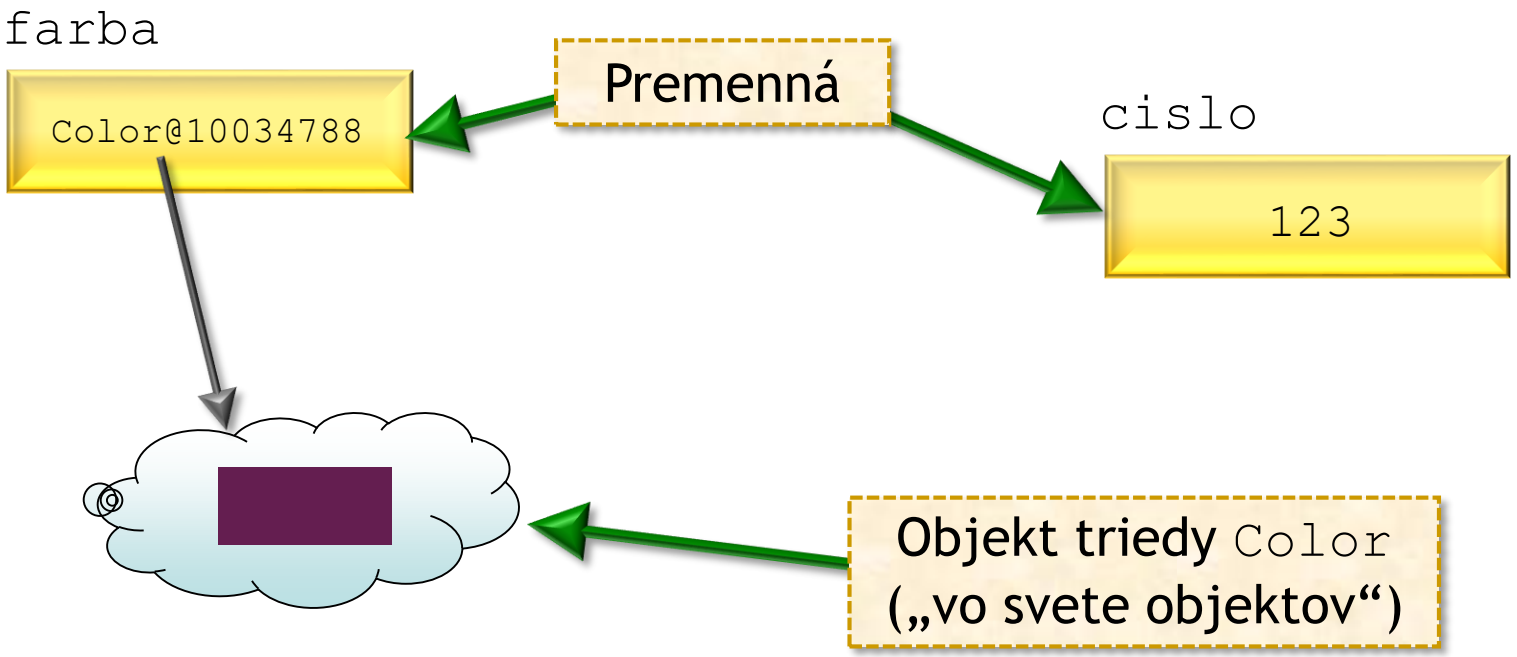
Čo vieme o premenných...

- Premenné (a typy hodnôt):
  - **primitívneho typu** – uchovávajú konkrétnu hodnotu zadaného typu (celkom 8 rôznych typov)
    - hodnoty niektorých typov je možné pretypovaním zmeniť na iný typ (implicitné vs. explicitné pretypovanie)
  - **referenčného typu** – uchovávajú referenciu („rodné číslo“) na nejaký objekt alebo hodnotu `null`

Žiadne iné typy premenných v Jave neexistujú.

Referenčná vs. prim. premenná

```
int cislo = 123;
```




```
Color farba = new Color(100, 30, 80);
```

Typy primitívnych premenných

- **Celočíselné hodnoty:**
  - `byte` (-128 až 127)
  - `short` (-32 768 až 32 767)
  - `int` (-2 147 483 648 až 2 147 483 647)
  - `long` (-9 223 372 036 854 775 808 až 9 223 372 036 854 775 807)
- **Reálne čísla:**
  - `double` – lepšia presnosť, zaberá viac pamäte
  - `float` – menšia presnosť, zaberá menej pamäte
- **Logická (pravdivostná) hodnota:**
  - `boolean` – dve možné hodnoty `true`/`false`

Ôsmy typ?

Znaky

- Na ukladanie **znakov** (primitívna hodnota) slúžia premenné typu **`char`**

```
char znak = 'a';
```

- **Znakové literály** (konkrétne znakové hodnoty) píšeme medzi apostrofy `'a'`, `'A'`, `','`, `' '`,...
- Do premennej typu **`char`** vieme uložiť ľubovoľný z prvých *65536* znakov kódovania UNICODE - (národné znaky, azijské znaky, ...)
  - Čo je to znak, znaková sada a čo UNICODE?
  - Je možné **`char`** pretypovať na niečo iné?

Znaková sada

- Každý znak má svoje **poradové číslo** – kód.
- V UNICODE sú cifry a písmena abecedy usporiadané súvisle za sebou:
  - znaky cifier
  - veľké písmena
  - malé písmena
  - národné znaky

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Ako sa kódujú znaky?

- **Kódovanie** (znaková sada)
  - dohodnutá tabuľka, v ktorej v i-tom riadku je znak prislúchajúci číslu i (číslujeme od 0)
  - **ASCII** (128 znakov, 7 bitov)
  - **Windows-1250** (256 znakov, 8 bitov = 1 bajt)
    - 0..127 – ako ASCII
    - 128 – 256 – znaky národných abecied v strednej Európe
    - alternatívy: ISO-8859-1, ISO-8859-2
  - **UNICODE** (veľmi veľa znakov ~ 144000)
    - 0..127 – ako ASCII
    - od 128 – znaky národných abecied (Európa, Ázia, ...)
    - kódovania: UTF-8, UTF-16, …

Špeciálne znaky

- Niektoré znaky musia byť zapísané ako **kombinácie viacerých znakov** ...
  - Znak medzery: `' '`
  - Znak tabulátora: `'\t'`
  - Znak konca riadka: `'\n'`
  - Znak lomítka: `'\\'`
  - Znak úvodzovky: `'\"'`
  - Znak apostrofu: `'\''`

Pretypovanie znakov

- Znaky sú len poradové čísla v dohodnutej znakovej sade
- **Znak** vieme previesť na **číslo**:

```
char znak = 'a';
int cislo = znak;
```

V premennej `cislo` bude poradové číslo znaku v UNICODE tabuľke.

- **Číslo** vieme previesť na **znak**:

```
int cislo = 65;
char znak = (char)cislo;
```

V premennej `znak` bude znak s poradovým číslom 65 - znak `'A'`

- Znaky vieme **porovnávať**:

```
znak >= 'a'
```

V skutočnosti porovnávame poradové čísla znakov v UNICODE tabuľke.

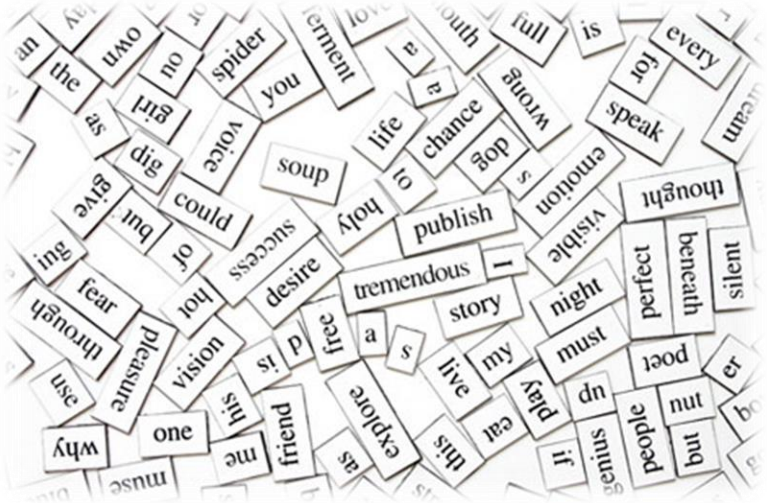
Typy primitívnych premenných

- **Celočíselné hodnoty:**
  - `byte` (-128 až 127)
  - `short` (-32 768 až 32 767)
  - `int` (-2 147 483 648 až 2 147 483 647)
  - `long` (-9 223 372 036 854 775 808 až 9 223 372 036 854 775 807)
- **Reálne čísla:**
  - `double` – lepšia presnosť, zaberá viac pamäte
  - `float` – menšia presnosť, zaberá menej pamäte
- **Logická (pravdivostná) hodnota:**
  - `boolean` – dve možné hodnoty `true`/`false`
- **Znak:**
  - `char` – jeden znak (z prvých 2^{16} znakov) v kódovaní UNICODE

Keď jeden znak nestačí

- V praxi je jeden znak málo, potrebujeme uchovávať postupnosti znakov...
  - postupnosti znakov vytvárajú slová, vety, ...
- **Znakový reťazec** = postupnosť znakov

- V Jave sa znakové reťazce uchovávajú **v objektoch**
  - kým niektoré objekty kreslia, iné zas slúžia na uchovávanie údajov...



Znakové reťazce

- Reťazec vieme uložiť v objekte triedy **String**

```
String s = new String("Ahoj");
```

Referenčná premenná `s` referencuje vytvorený objekt triedy `String` (uchováva jeho „rodné číslo“)

Vytvoríme objekt triedy **String**, ktorého „životným cieľom“ bude uchovanie 4-znakového reťazca `"Ahoj"`

Reťazcový literál (konkrétna hodnota), znaky sa píšu medzi úvodzovky `" "`.

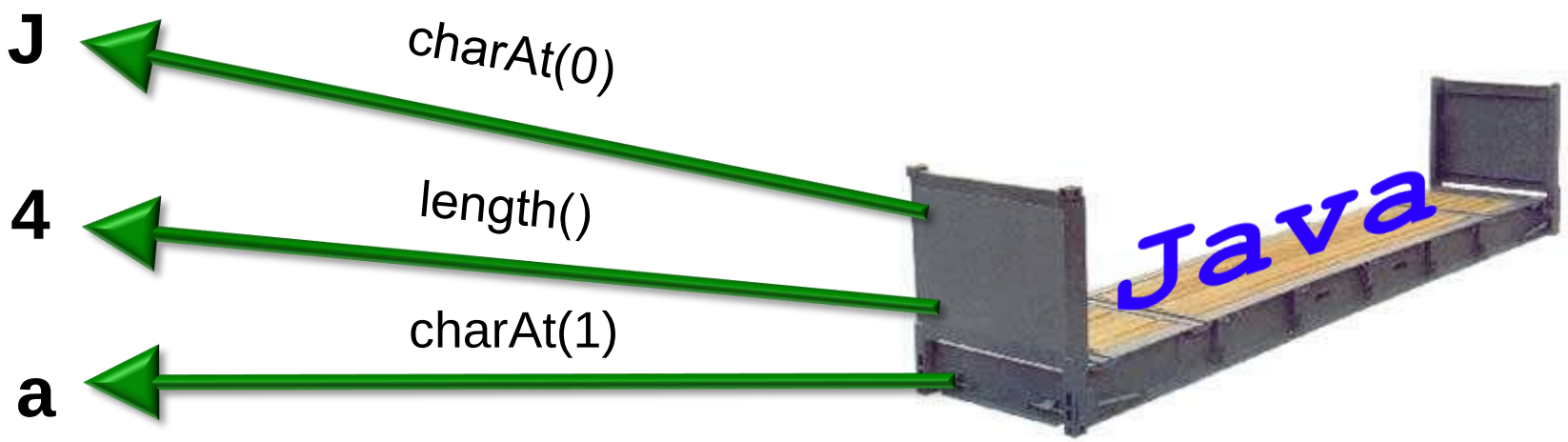
Prieskum triedy String

```
ObjectInspector oi = new ObjectInspector();
String s = new String("Java");
oi.inspect(s);
```

- Skúmajme metódy objektov triedy String ...
  - Čo robia metódy charAt a length?

Výsledky krátkeho výskumu

- *charAt(int)* – vráti **znak** na i-tej **pozícii**, znaky sú číslované od **0**
- *length()* – vráti **dĺžku reťazca**



Počet výskytov písmena (1)

- Chceme naučiť korytnačku spočítať počet výskytov písmena `'a'` v reťazci ...
  - V reťazci `"Java"` sa `'a'` vyskytuje 2-krát

```
public int countA(String s)
```

Vrátiť chceme (celé číslo) vyjadrujúce počet výskytov znaku `a` v „spracovávanom“ reťazcovom objekte.

Parametrom je referencia na objekt reťazca („rodné číslo reťazca“), v ktorom počítame výskyty znaku `a`

Počet výskytov písmena (2)

```
public int countA(String s) {
    int result = 0;


    for (int i=0; i<s.length(); i++) {
        if (s.charAt(i) == 'a') {
            result++;
        }
    }

    return result;
}
```

Využijeme premennú `result` ako počítadlo toho, koľkokrát nám reťazec odpovedal na `charAt` znakom `a`.

Skúšame všetky indexy znakov v reťazci *(0, 1, …, dĺžka-1)* a pýtame si písmeno, na danom indexe. Ak je to `a`, tak zvýšime počítadlo o 1.

Je to číselný reťazec?

- Chceme naučiť korytnačky metódu, ktorá overí, či zadaný reťazec je číselný
  - číselný reťazec je reťazec zo znakov cifier `'0'`, `'1'`, `'2'`, … , `'9'` s kódmi 48 až 57.

```
public boolean isNumeric(String s)
```

Vrátime `true` práve vtedy, keď `s` referencuje číselný reťazec

Parametrom je referencia na objekt reťazca („rodné číslo reťazca“), ktorý testujeme, či obsahuje len znaky cifier.

Je to číselný reťazec?

```
public boolean isNumeric(String s) {
   if (s == null) {
      return false;
   }

   for (int i = 0; i < s.length(); i++) {
       char c = s.charAt(i);
       if (!(('0' <= c) && (c <= '9'))) {
          return false;
       }
    }

    return true;
}
```

Overíme, či máme referenciu na objekt triedy `String`

Ak sme našli znak, ktorý nie je znakom cifry, ihneď končíme s **`false`**

Ak žiaden znak vo **`for`**-cykle nespôsobil **`return`**, potom všetky znaky reťazca sú znakmi cifier.

Triedy, ktoré poznáme

- Turtle – korytnačka
- WinPane – kresliaca plocha
- ObjectInspector – prieskumník objektov

Objekty týchto tried môžeme **vidieť**

- String – reťazec=postupnosť znakov
- Color – farba namiešaná z RGB zložiek
  - `Color farba = new Color(100, 30, 80);`

Objekty týchto tried **nevidíme** (slúžia na uchovanie údajov)

Premenná `farba` neuchováva farbu, iba referenciu („rodné číslo“) na objekt, ktorý nejako vnútorne uchováva namiešanú farbu.

Ďalší prieskum triedy String

```
ObjectInspector oi = new ObjectInspector();
String s = new String("Java");
oi.inspect(s);
```

- Skúmajme metódy objektov triedy String ...
  - Pozorovanie: niektoré metódy majú rovnaké meno, ale iné parametre ...

Výsledky výskumu

- Zaujímavé metódy:
  - *charAt(int)* – vráti **znak** na i-tej **pozícii**, znaky sú číslované od 0
  - *length()* – vráti **dĺžku reťazca**
  - *endsWith(String)* – vráti, **či** reťazec **končí** zadaným reťazcom
  - *concat(String)* – vyrobí nový objekt triedy String, ktorý vznikne pridaním znakov so zadaného reťazca za znaky pôvodného reťazca (**zreťazením**). Vráti referenciu na vytvorený objekt.

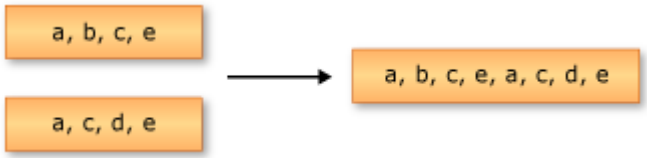
Spájanie reťazcov (1)

```
String s1 = new String("Ahoj");
String s2 = new String("Svet");
String s3 = s1.concat(s2);
```

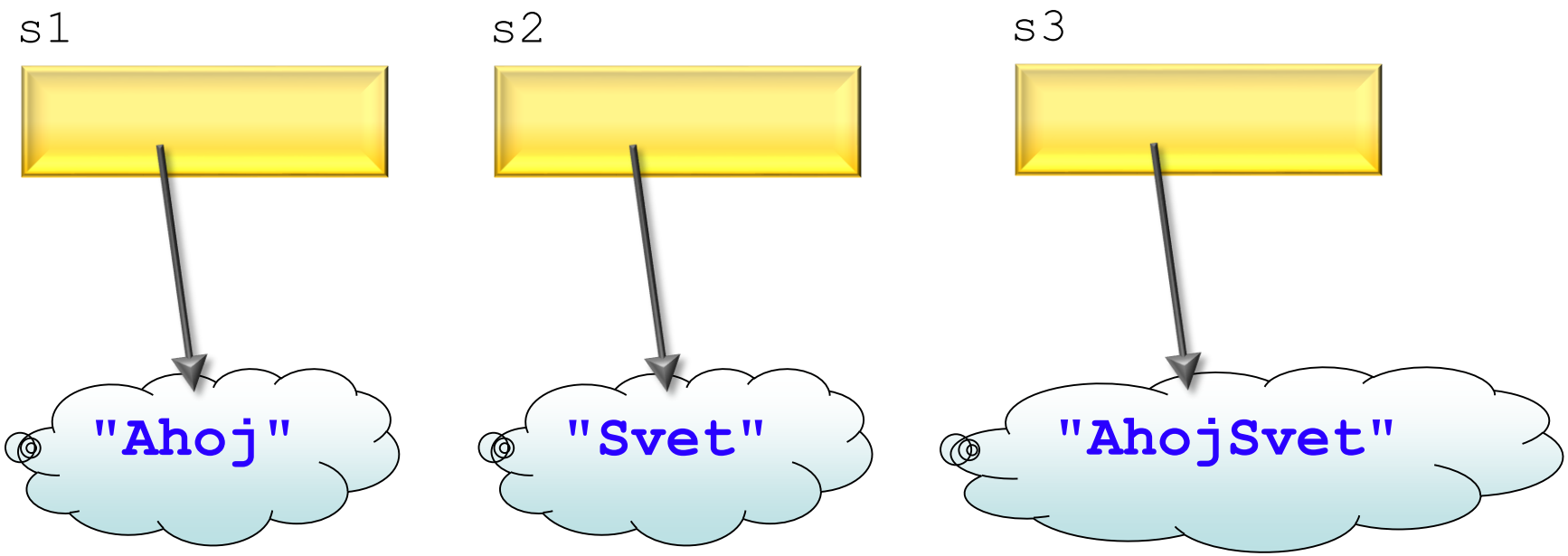
Vrátená referencia sa uloží do referenčnej premennej *s3*

Vytvorí nový reťazec s obsahom `"AhojSvet"` a vráti referenciu naň

```
System.out.println(s3);
```

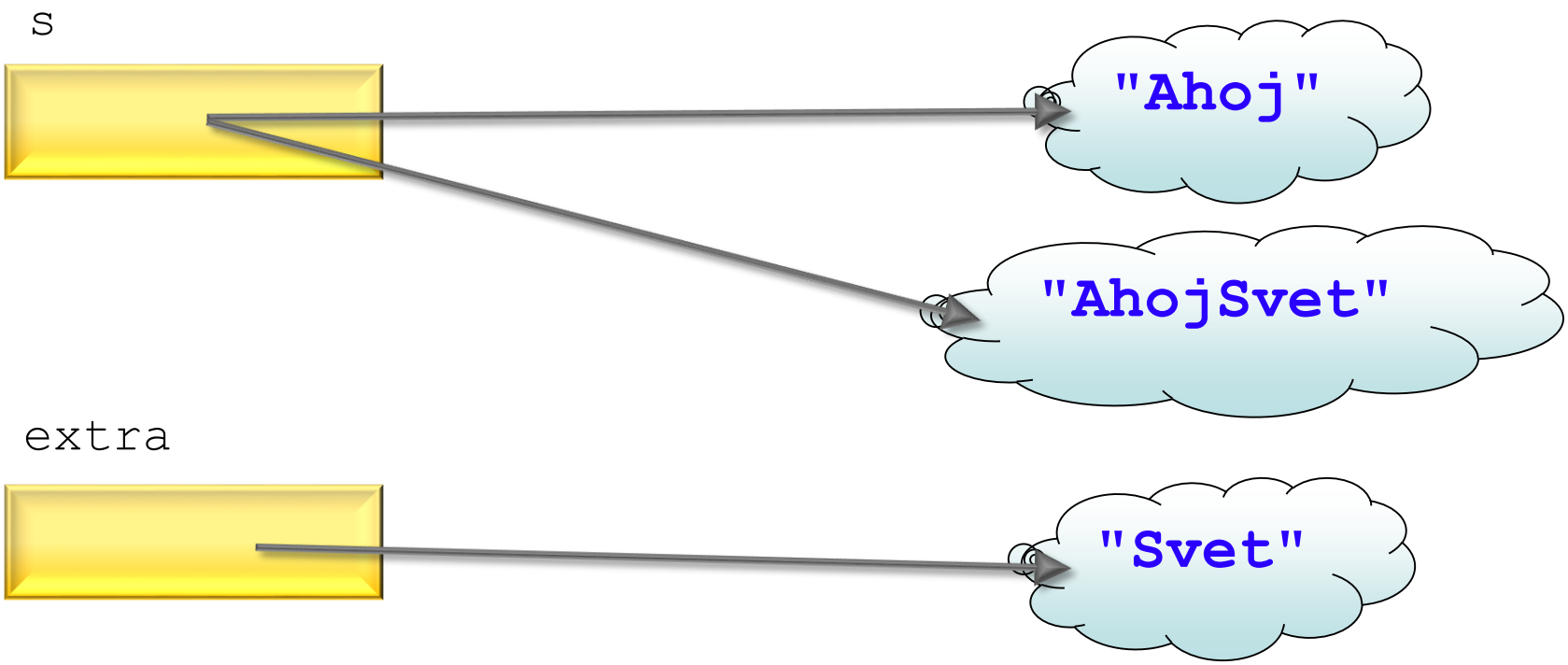


Spájanie reťazcov (2)




```
String s1 = new String("Ahoj");
String s2 = new String("Svet");
String s3 = s1.concat(s2);
```

Spájanie reťazcov (3)




```
String s = new String("Ahoj");
String extra = new String("Svet");
s = s.concat(extra);
```

Duplikovanie reťazcov (1)

- Chceme naučiť korytnačku zduplikovať reťazec zadaný počet krát: reťazec `"Svet"` zduplikovaný 3 krát je: `"SvetSvetSvet"`

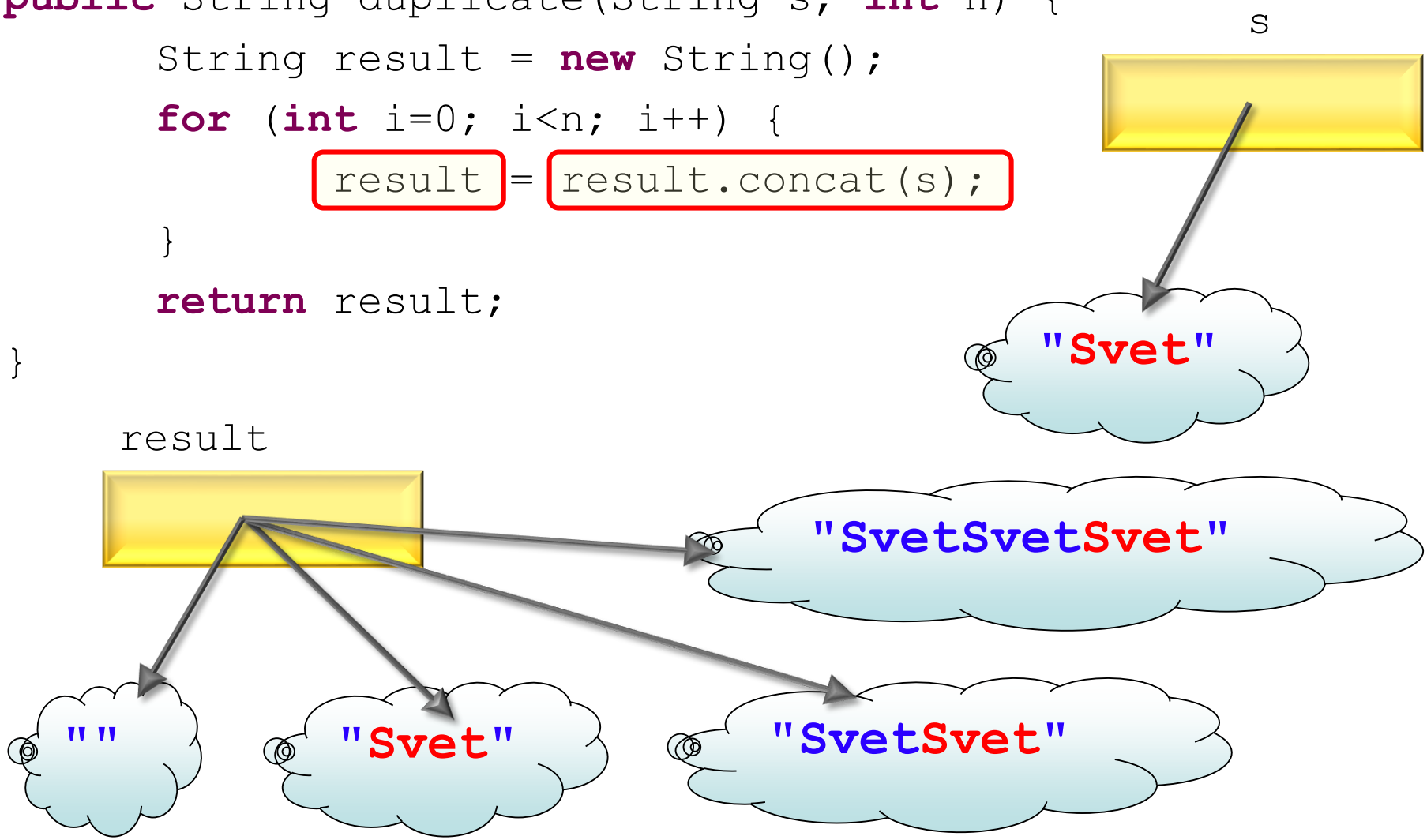
```
public String duplicate(String s, int n)
```

Vraciame referenciu na vytvorený objekt triedy `String` („rodné číslo vytvoreného tohto objektu“)

Parametrom je referencia na objekt reťazca, ktorý chceme duplikovať

Duplikovanie reťazcov (2)

```
public String duplicate(String s, int n) {
        String result = new String();
        for (int i=0; i<n; i++) {
                result = result.concat(s);
        }
        return result;
}
```



Duplikovanie reťazcov (3)

```
public String duplicate(String s, int n) {
        String result = new String();

        for (int i=0; i<n; i++) {
                result = result.concat(s);
        }

        return result;
}
```

Požiadame objekt referencovaný z premennej `result`, aby vytvoril nový reťazcový objekt so „zlepeným obsahom“

Počas vykonávania `duplicate("Svet", 3)` vzniknú celkom 4 objekty s obsahom: `""`, `"Svet"`, `"SvetSvet"`, `"SvetSvetSvet"`

Rovnaké reťazce

```
String s1 = new String("Ahoj");
String s2 = new String("Ahoj");
```

2 rôzne objekty uchovávajúce rovnaký obsah

```
boolean rovnakeReferencie = (s1 == s2);
boolean rovnakyObsah = (s1.equals(s2));

System.out.println(rovnakeReferencie);
System.out.println(rovnakeObsah);
```

Pozor na reťazce: == vs. equals!!!

Všetky objekty sú si rovné ...

- ... ale niektoré sú si rovnejšie.

- Objekty t vyvolených tried) ma Java ...
- Namiesto

```
Stri                    j");
```

stačí nap

"Metóda concat() pre lenivých"

- Jazyk Java ponúka **operátor pre zreťazovanie** (spájanie reťazcov):

```
String s1 = "Ahoj";
String s2 = "Svet";
String s3 = s1 + s2;
String s4 = s1 + " " + s2;
```

Varovanie pre pokročilých: (nielen) na slajdoch o zreťazovaní tak trochu klameme...

Ak je jeden z operandov operácie **+** reťazec (objekt triedy `String`), chápe sa `+` ako operácia zreťazenia. Výsledkom je **nový objekt** triedy `String`, ktorého obsah vznikne zreťazením obsahu operandov.

"Metóda concat() pre lenivých"

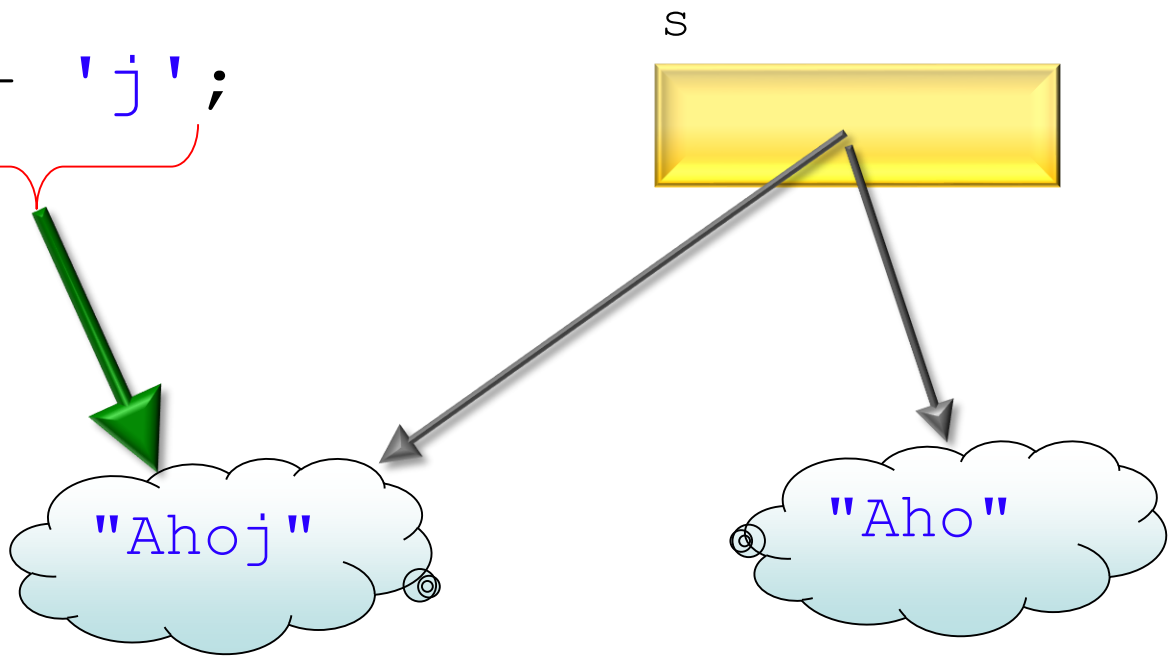
- Ak je jeden z operandov operácie + reťazec (objekt triedy String), **+** funguje **ako operácia zreťazenia**. Výsledkom je nový objekt, ktorého obsah vznikne zreťazením obsahu operandov.
- Ak je druhý z operandov nie reťazec, Java sa ho **pokúsi prerobiť na reťazec** !!!
- Možeme písať:

```
int c = 10;

String s = "V c je cislo: " + c;
```

Vyrobí reťazec:
"V c je cislo: 10"

Hra s referenciami

```
String s = "Aho";

s = s + 'j';
```



Obrátený reťazec (1)

- Chceme naučiť korytnačky metódu, ktorá prevráti znaky v reťazci ...
  - `"Java" » "avaJ"`

```
public String reverse(String s)
```

Metóda vráti referenciu na objekt triedy `String`

Obrátený reťazec (2)

```
public String reverse(String s) {
      if (s == null)
            return null;

      String result = "";
      for (int i=0; i<s.length(); i++) {
            result = s.charAt(i) + result;
      }

      return result;
}
```

Ďalšie reťazcové metódy

- substring - vráti referenciu na reťazec, ktorý je podreťazcom reťazca
- trim - vráti referenciu na reťazec, v ktorom odstráni medzery z oboch koncov reťazca
- indexOf - vráti pozíciu prvého výskytu zadaného reťazca v reťazci, -1 ak nie je podreťazcom

```
String s = "Java je super";

s.toUpperCase() » "JAVA JE SUPER";

s.substring(2, 7) » "va je";
```

Čo s odpadom?

- Naše metódy vytvárajú veľa „odpadu“ vo forme dočasne potrebných objektov triedy `String`.
- Riešenie: objekty triedy **`StringBuilder`**
- „Zmysel života“ **`StringBuilder`**-ov:
  - uchovávať postupnosť znakov, ktorú ide meniť (narozdiel od String-ov, ktoré žijú s tým obsahom, s ktorým sa „narodia“)
- Niektoré metódy **`StringBuilder`**-ov:
  - *append* – prilepí reťazec/znak/číslo... na koniec
  - *insert* – vloží reťazec/znak/číslo na zadaný index
  - *toString* – vytvorí reťazec podľa aktuálneho obsahu

Duplikovanie reťazcov (3)

```
public String duplicate(String s, int pocet) {
        StringBuilder sb = new StringBuilder();

        for (int i=0; i<n; i++) {
                sb.append(s);
        }

        return sb.toString();
}
```

Vytvoríme objekt na uchovanie postupnosti znakov, ktorý možno „meniť“

Prilepíme na koniec postupnosti znakov v `sb` celý obsah reťazca `s`

Necháme „meniteľnú“ postupnosť znakov `sb` vytvoriť objekt triedy `String` podľa aktuálneho obsahu

Kde nájsť viac informácií

Google Slovensko

java 8 string

Na čo treba pamätať

- **Reťazce** v Jave (oproti iným jazykom) **sú objekty** triedy `String`
- Java má pre „lenivých“ programátorov „skratky“
- Obsah 2 reťazcov sa porovnáva na zhodu pomocou metódy **equals** (nie ==)
- Objekt triedy `String` nikdy počas svojho života **nemení svoj obsah**
- Reťazcové metódy zvyčajne vrátia **referenciu na novovytvorený** reťazec so „správnym“ obsahom

Rozširovanie WinPane

- Nielen korytnačky (triedu *Turtle*) vieme rozširovať, ale vieme to spraviť (skoro) s každou Java triedou …
- Ukážka ...

Kreslenie bodiek

- Chceme, aby plocha mala metódu na nakreslenie bodky na zadanej pozícii...

- Postup:
  - Vytvoríme korytnačku na kreslenie
  - **Pridáme** ju do kresliacej plochy
  - Necháme ju nakresliť to, čo treba
  - Korytnačku **odstránime** z kresliacej plochy metódou *remove* objektov triedy *WinPane*

Myšacie udalosti v JPAZe

- Ak pridáme do triedy rozširujúcej triedu *WinPane* metódu so **správny menom a parametrami**, vykoná sa vždy pri kliknutí do kresliacej plochy:

```
protected void onMouseClicked(int x, int y,
                              MouseEvent detail)
```

Ako to funguje ?

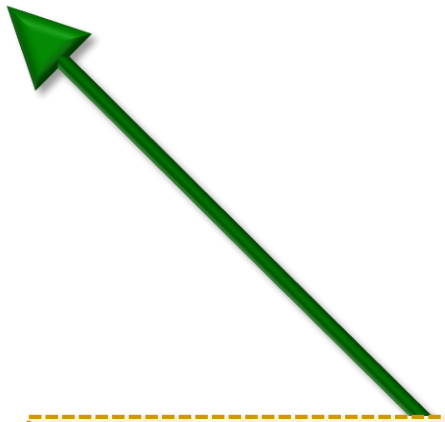
- JPAZ je naprogramovaný tak, že po kliknutí do kresliacej plochy sa zavolá metóda **onMouseClicked** pričom v parametroch sú užitočné informácie:
  - **x** – x-ová súradnica miesta, kam sa kliklo
  - **y** – y-ová súradnica miesta, kam sa kliklo
  - **detail** – referencia na objekt triedy **MouseEvent**, kde sú doplňujúce informácie o tom, čo sa stalo
- Kreslime červené bodky tam, kam sa kliklo ...

MouseEvent objekt

- Čo všetko sa vieme dozvedieť z MouseEvent objektu:
  - `boolean isAltDown()` - či je zatlačený **Alt** kláves
  - `boolean isControlDown()` - či je zatlačený **Ctrl**
  - `boolean isShiftDown()` - či je zatlačený **Shift**
  - `int getButton()` - ktorým tlačidlom sa kliklo:
    - `MouseEvent.BUTTON1` – ľavé tlačidlo myši
    - `MouseEvent.BUTTON2` – stredné tlačidlo myši
    - `MouseEvent.BUTTON3` – pravé tlačidlo myši

Využitie MouseEvent

```
protected void onMouseClicked(int x, int y,
                              MouseEvent detail) {

    if ((detail.getButton() == MouseEvent.BUTTON1) &&
        (detail.isAltDown()) {
        // prikazy
    }
}
```



Príkazy sa vykonajú, ak sa kliklo ľavým tlačidlom myši a zároveň je zatlačený *Alt* kláves.

Informačný výpis o bodke

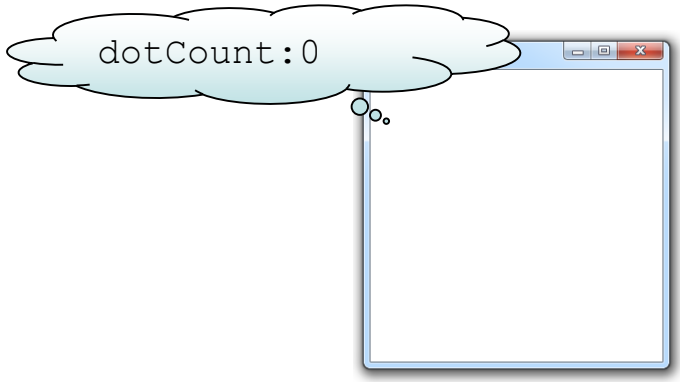
- Chceme vedľa každej bodky napísať, koľká je v poradí vytvorená ...

- Potrebovali by sme premennú, ktorá nezanikne po skončení metódy ...

Riešenie:

inštančné (objektové) premenné

Inštančné (objektové) premenné



- Deklarácia:

```
public class DotPane extends WinPane {
    private int dotCount;

    // priestor pre metody …
}
```

Deklarácia premennej, ktorá bude „žiť“ v každom objekte triedy `DotPane` (stane sa časťou objektu).

Inštančné premenné - metafora

- Premenné vytvorené v metóde „existujú“ len počas jej vykonávania
  - dočasný „papierik“ na poznámky
  - RAM pamäť – po vypnutí sa jej obsah stratí
- Inštančné premenné „existujú“ spolu s objektom
  - **mozog objektu**, kde si môže pamätať hodnoty natrvalo
  - pevný disk – trvalo uložený obsah

Inštančné (objektové) premenné

- Čo platí pre inštančné premenné:
  - deklarujeme ich tak ako premenné v metódach, len ich píšeme **do vnútra triedy mimo jej metód** s pridaním slovíčka **`private`**
  - z metód k nim pristupujeme cez **`this`**:

```
this.dotCount = 1;
```

  - inštančná premenná sa **vytvára** v objekte **pri jeho vzniku** cez **`new`**
  - ak ich neincializujeme, tak inštančné premenné sú **automaticky inicializované** na tzv. „defaultnú“ hodnotu

„Defaultné“ hodnoty

- Pre `int`, `byte`, `short`, `long`, `float`, `double` je to hodnota `0`
- Pre `boolean` je to `false`
- Pre `char` je to Unicode znak s kódom `0`
- Pre premenné referenčného typu (referencujúce objekty tried ako *String*, *Turtle*, ...) je to `null`.

Prístup k inštančným premenným

- Inštančné premenné sú „skryté v mozgu“ objektu
- Na získanie hodnoty vytvoríme metódy, ktoré nám povedia aktuálny obsah (ak chceme také metódy mať):

```
public int getDotCount() {
        return this.dotCount;
}
```

Zvykom je pomenúvať tieto metódy v tvare **getNazovPremennej**

Farebné retiazky

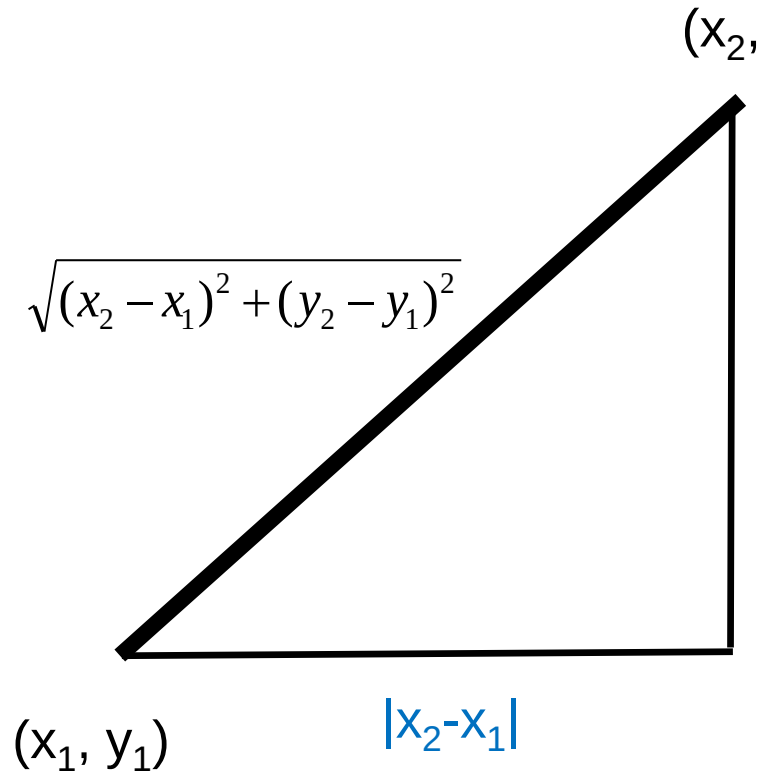
- Pri pohybe myškou so zatlačeným CTRL chceme kresliť farebné bodky:
  - bodky nie su príliš blízko seba – 2 za sebou idúce bodky v tej istej reťazi sa neprekrývajú

Programujeme ...

Ďalšie myšacie udalosti

- Okrem klikania môžeme obsluhovať:
  - **onMousePressed** – pri zatlačení tlačidla myši
  - **onMouseReleased** – po uvoľnení tlačidla myši
  - **onMouseClicked** – po kliknutí (postupnosť: Pressed, Released, Clicked)
  - **onMouseMoved** – pri pohybe myši bez zatlačeného tlačidla myši
  - **onMouseDragged** – pri pohybe myši so zatlačeným tlačidlom myši

Vzdialenosť dvoch bodov



Pytagorova veta (viď ZŠ)

```
double dx = x2-x1;
double dy = y2-y1;
```

```
double vzdialenost = Math.sqrt(dx*dx + dy*dy);
```

to be continued …

Ďakujem za pozornosť !