Programovanie, algoritmy, zložitosť / ÚINF/PAZ1c

PAZ1C

Róbert Novotný
robert.novotny@upjs.sk

23. 9. 2009

Formality a byrokracie

- Teoretické cvičenie („prednáška")
  - streda, 18.05, ??
- Praktické cvičenia
  - štvrtok, ??:??, P3, Ib
  - štvrtok, ??:??, P4, Xib

Každý musí byť zaradený na jedno praktikum a jedno teoretické cvičenie!

- Všetky inštrukcie na

http://ics.upjs.sk/~novotnyr/wiki/Java/PAZ1c

Vstupné vedomosti

- toto je tretí semester programovania
- od študenta sa očakáva
  - znalosť základného procedurálneho programovania
    - viem, čo je cyklus, podmienka, premenná, dátový typ, metóda (procedúra, funkcia)
  - mlžné predstavy o OOP
  - základné predstavy o programovacích technikách a algoritmoch
    - rekurzia...

Témy a motívy predmetu

- Pokročilé princípy objektovo-orientovaného programovania
- Oproti minulému roku väčší dôraz na dizajn a udržovateľnosť programov
- Zamerané na príklady z praxe
- Programovací jazyk Java

Požiadavky na hodnotenie

- **Účasť** na praktických cvičeniach (30%)
- Záverečný **projekt** (40%) – nutná podmienka
- Dva **testy**: jeden v polovici semestra, druhý na konci (30%)
  - jedna možnosť opravy jedného testu

Požiadavky na hodnotenie

- Extra požiadavka
  - domáca príprava

Nejdůležitější: vlastní praxe
Na přednáškách se nikdo nikdy programovat nenaučil

-- Filip Zavoral, MFF UK Praha

And now...

PAZ1C

...a teraz...

Reálny príklad

„Z textového súboru načítajte dáta o zamestnancoch a ich platoch. Zistite meno zamestnanca s najvyšším platom."

Programovanie = dáta + algoritmy

„Ako bude vyzerať algoritmus?"

1. Načítaj riadok
2. Vytrieskaj z neho hodnotu pre plat
3. Pamätaj si priebežnú najväčšiu hodnotu platu a osobu, ktorá ju má
4. Ak je hodnota na riadku väčšia ako priebežná, aktualizuj priebežnú hodnotu a osobu
5. Na konci vypíš meno.

Programovanie = dáta + algoritmy

„Ako bude vyzerať algoritmus?"

Alebo inak, stručnejšie a jasnejšie, ale výpočtovo zložitejšie:

1. načítaj riadky do nejakej štruktúry
2. zotrieď ju podľa platu
3. vezmi prvú položku a vypíš

Programovanie = dáta + algoritmy

„Ako budú vyzerať dáta?"

Dohoda:

data.txt

```
Don Fitzgerald-Kennedy 1000
Peter Spielberg 7650
```

Ako reprezentovať dáta?

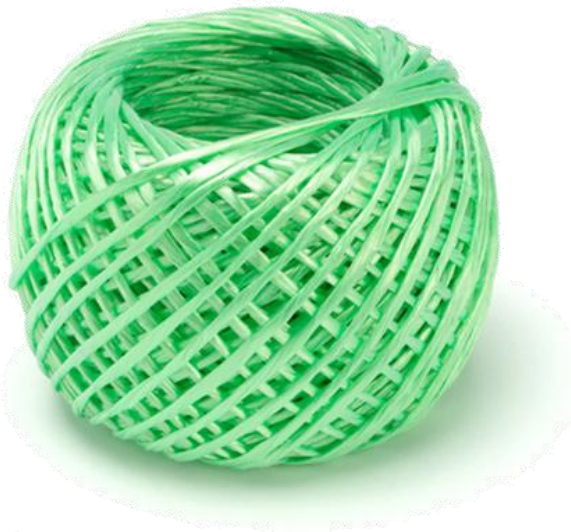
- vhodná reprezentácia dát ušetrí čas a sprehľadní kód
- máme riadok, teda osobu a jej plat
- nápad č. 1: osoba s platom je **String**.
  - ak treba zistiť hodnotu platu, vysekáme príslušný podreťazec, prevedieme na číslo...

data.txt

```
Don Fitzgerald-Kennedy 1000
Peter Spielberg 7650
```

Ako reprezentovať dáta? String

- **String** (reťazec) je veľmi užitočná štruktúra
- Má neobmedzenú kapacitu
- Dá sa doň napchať všetko – čísla, znaky, hocičo
- Dá sa na ňu dívať ako na pole (pole znakov)
- Každý objekt možno reprezentovať ako String



Ako reprezentovať dáta? String

- Lenže nejeden programátor podľahne kúzlu a začne používať Stringy na všetko
  - „Veď typová kontrola ma zdržiava!"

```
void zaevidujOsobuDoSystému(String osoba)
void porovnajDveMatice(String m1, String m2)
void upravTelefón(String predvoľba, String číslo)
String dajRodnéČíslo(String menoAPriezvisko)
```

Ako reprezentovať dáta? String

String používame len na to, na čo boli určené!

- Postupnosť znakov: slová, vety **bez štruktúry**.
- Ako dočasné úložisko pre dáta, ktoré sa o chvíľu spracujú do štruktúrovanej podoby...
  - načítavam riadok do Stringu, o chvíľu ho spracujem
- ...alebo sa konvertujú na správny typ
  - získam reťazec z textového políčka a hneď ho prevediem na číselný typ

Ako reprezentovať dáta?

- nápad č. 2: osoba s platom je trojprvkové **pole**
  - v prvom prvku je meno
  - v druhom priezvisko
  - v treťom plat

Don	Fitzgerald-Kennedy	1000

Ako reprezentovať dáta?

- Lenže akého typu má byť pole?
- Potrebujeme typ, ktorý obsiahne aj meno, aj číselný plat

```
String[] osoba = new String[3];
```

- lenže plat ako String?
  - sme pri predošlom probléme
    - vždy, keď chceme robiť s platom osoby, musíme vykonať typovú konverziu

Ako reprezentovať dáta?

- Pole má ďalší folklórny problém

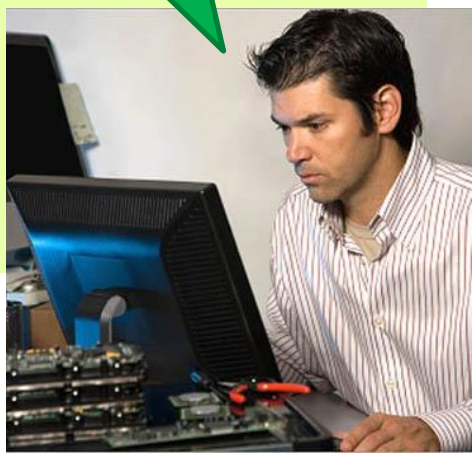
Indexy polí sú magické čísla.

```
String[] osoba = new String[3];
osoba[0] = "Don"
osoba[1] = "Fitzgerald-Kennedy";
osoba[2] = "1000"
```

Ako reprezentovať dáta?

- ...o tisíc riadkov neskôr....

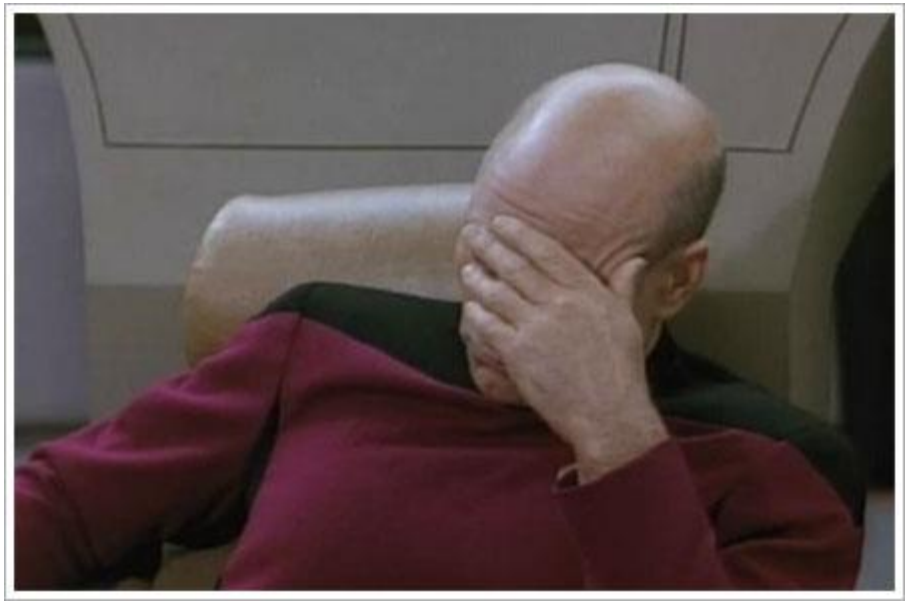
```
void vypíšPlat(String[] osoba) {
  System.out.println("Plat: "
  + osoba[3]);
}
```



- výnimka **ArrayIndexOutOfBoundsException**
  - lebo sme vyleteli z rozsahu poľa

Ako reprezentovať dáta?

- Iný problém:
  - ...o dva roky neskôr
  - *„Ľudia, odteraz je v dátach vymenené priezvisko s menom!"*
- Môžeme prepisovať všetky indexy a modliť sa, že sa nepomýlime.



Ako reprezentovať dáta?

- Ako to riešiť v prípade núdze?
  - Konštanty!

```
public static final int INDEX_MENA = 0;
public static final int INDEX_PRIEZVISKA = 1;
public static final int INDEX_PLATU = 2;
...
void spracuj(String[] osoba) {
  System.out.println(osoba[INDEX_MENA]);
}
```

Ako reprezentovať dáta?

- nápad č. 3 (pre zdatných v kolekciách): osoba bude položka v mape

```
Map<String, String> osoby
    = new HashMap<String, String>();
osoby.put("Madonna", "22000");
```

Ako reprezentovať dáta?

- mapa však nedáva žiadne extrémne výhody
  - akurát vieme rýchlo vyhľadávať podľa mena
    - ale to nám netreba
  - rovnako nie je robustná voči zmenám

Ako reprezentovať dáta?

- nápad č. 4: osoba s platom je trieda s troma inštančnými premennými!
  - meno: String
  - priezvisko: String
  - plat: int

Osoba.java

```
class Osoba {
  String meno;
  String priezvisko;
  int plat;
}
```

Ako reprezentovať dáta?

- jasne vidíme, ktoré inštančné premenné máme k dispozícii a ako sa volajú
- typová kontrola!
- automatické dopĺňanie v Eclipse!
- prehľadné názvy metód využívajúcich triedu

```
vypíš(Osoba cvičiaci) {
  System.out.println(cvičiaci.meno);
}
```

ČO SME ZISTILI?

Triedy sú úložiskom dát

- Na triedy sa môžeme dívať ako na štruktúrovaný dátový typ
  - dátové typy v Jave sú **primitívne**
    - **int**, **boolean**, všetky tie, čo sa píšu malými písmenami
    - obsahujú jeden druh dát
  - štruktúrované (triedy)
    - pozostávajú zo zložiek **primítívnych** alebo **štruktúrovaných** typov

Triedy sú úložiskom dát

- Reálny príklad na rekurziu!
  - „To iterate is human, to recur divine!"
  - „Rekurzia je definícia pomocou seba samého!"
- Krok 1: primitívny typ je int, boolean...
- Krok 2: štruktúrovaný typ pozostáva zo zložiek
  a) buď **jednoduchého** typu
  b) alebo **štruktúrovaného** typu

- Triedy z predošlého príkladu nemajú metódy – nesú len dáta
  - „sú degenerované"
  - **DTO** („data transfer object") – objekty na prenos dát
  - **VO** („value object") – objekt s hodnotou
- V iných programovacích jazykoch existujú pre takéto triedy samostatné typy

Struct a record

jazyk C

```
struct Osoba {
  char meno[8];
  char priezvisko[32];
  int plat;
}
struct Osoba cvičiaci;
cvičiaci.plat = 10;
```

Pascal/Delphi

```
type TOsoba = record
  meno:string;
  priezvisko:string;
  plat:integer;
end;

var cvičiaci:TOsoba;
cvičiaci.plat := 10
```

- netreba vytvárať inštancie

Degenerované triedy v Jave

- v Jave neexistuje ani **struct**, ani **record**
- je to dizajnérsky zámer, aby sa zachovávala jednoduchosť
- drobná nevýhoda: musíme vytvárať inštancie

```
Osoba o = new Osoba();
o.meno = "Don";
o.priezvisko = "Fitzgerald-Kennedy";
o.plat = 100;
```

Iný, filozofický pohľad na objekty

- prirodzený pohľad
- objekty sú všade naokolo
  - študent(i), prednášajúci, tabuľa,
  - počítač, vypínač, strom,
  - chlap, hrdina, dub, stroj...

Filozofický pohľad na objekty

- zamyslime sa nad objektom:
  - o čom má objekt vedomosti (**stav**)
  - aké **činnosti** dokáže vykonávať
- príklad: ***vypínač***
  - **stav**: je zapnutý/vypnutý
  - **činnosti**: dokáže sa (= dokážeme ho) zapnúť a vypnúť

Filozofický pohľad na objekty

- príklad: empétrojka
  - stav:
    - má názov
    - má interpreta
    - má dĺžku
    - veľkosť dátového toku (128 kbps, 190 kbps...)
  - schopnosť:
    - dokáže sa prehrať
    - dokáže sa preniesť do MP3 walkmana™

Triedy, objekty a inštancie

- trieda: predstavuje koncept, resp. pojem
- príklad: pes

- čo majú spoločné všetky psy
- aký je koncept „psovitosti". Dohodnime sa:
  - 4 nohy, chvost, chlpy
  - breše

Triedy, objekty a inštancie

- **trieda:** všeobecný abstraktný pojem
  - Pes: 4 nohy, chvost, chlpy, breše
- **objekt:** konkrétny hmatateľný *objekt* spĺňajúci podmienky danej triedy:
  - Rex, Ariel Hviezdička, Lajka
- trieda je predlohou pre inštancie
  - pojmy objekt a inštancia sa zamieňajú

„Rex je inštancia triedy Pes" alebo „Rex je objekt typu Pes"

Triedy vs objekty

- trieda vs. objekt
- definícia vs. príklad
  - def: *bodom nazývame usporiadanú dvojicu (x, y)...*
  - príklad: *A = (2, 3)*
- abstraktný pojem vs. konkrétna vec
- slovo v slovníku vs. obrázok
- tlačivo vs. dáta v ňom

Triedy

- jemne upravíme našu definíciu psa
  - 4 nohy – nie až taká dôležitá vlastnosť
    - *černobyľský pes?*
  - chvost, chlpy… – rovnako neveľmi dôležité
    - *možno farba*
  - **rasa** – vyzerá dôležitá
  - **vek**
  - … iné podľa dohody

stav

  - **breše** – dôležitá: dva psy môžu brechať inak
  - … iné podľa dohody

schopnosti

Triedy


```
class Pes {
   String rasa;
   float vek;
}
```

- stavové premenné
- každý pes má rasu
- každý pes má vek

- definujeme vlastný **dátový typ Pes**
- *stav* (o čom má objekt vedomosti) je reprezentovaný v **stavových premenných** (alias inštančné premenné, alias *fields*)
- o tom, ako sa zapíše to, čo objekt dokáže, viac neskôr

Pôrod psa (vytváranie inštancie)

Pes.java

```
class Pes {
   String rasa;
   int vek;
}
```

PesTester.java

```
class PesTester {
  public static void main(String[] args) {
    Pes dunčo;
    dunčo = new Pes();
   }
}
```

Načo dva súbory?

Pes.java

```
class Pes {
  String rasa;
  int vek;
  public static void main(String[] args) {
    Pes dunčo = new Pes();
  }
}
```

- dva súbory rozdeľujú aplikáciu na ,,aplikačnú logiku" a "používateľské rozhranie" (testovaciu časť)
- v našej smiešnej aplikácii je to zbytočné, ale je to zárodok filozofie **unit testov**
- triedu Pes je zároveň možno priamo recyklovať v iných projektoch bez nutnosti mazať "testovaciu logiku"

Pôrod psa (vytváranie inštancie)

vlastný dátový typ

deklarácia

- `Pes dunčo;`
  - trieda je užívateľom definovaný dátový typ

---

iniciali-zácia

- `dunčo = new Pes();`
  - inicializácia premennej
  - `new Pes()` = vytvor novú inštanciu (objekt) typu Pes
  - priraď ju do premennej `dunčo` typu `Pes`.

Pôrod psa (vytváranie inštancie)

- Deklaráciu (**Pes dunčo;**) a inicializáciu (**dunčo = new Pes();**) vieme zapísať jedným riadkom.
- Je to štandardný **idiom** (zaužívaný výraz)

PesTester.java

```
class PesTester {
  public static void main(String[] args) {
    Pes dunčo = new Pes();
  }
}
```

Štelujeme psa

- na modifikovanie a čítanie používame bodkovú notáciu

PesTester.java

```
class PesTester {
   public static void main(String[] args) {
        Pes dunčo = new Pes();
        dunčo.rasa = "slovenský čuvač";
        dunčo.vek = 8;

        System.out.println(dunčo.rasa);
        System.out.println(dunčo.vek);
   }
}
```

```
slovenský čuvač
8
```