PB162 Programování v jazyce Java

7. cvičenie

dnešné cvičenie

- trieda `Object` a jej metódy
  - `equals`, `hashCode`, `toString`
- konštanty
- `final`
- volanie konštruktoru v konštruktore
  - `this()`, `super()`
- **3. úloha** – max 1 hodina, musí ísť skompilovať
- cykly (`for`, `while-do`, `do-while`)
- pole

trieda `Object` a jej metódy

- implicitný predok všetkých tried
- obsahuje informáciu o identite objektu
- metódy (najčastejšie používané/prekrývané):
  - `equals` – definuje ekvivalenciu na triede
  - `hashCode` – počíta hash objektu
  - `toString` – poskytuje textovú informáciu o objekte

`equals`

- definuje ekvivalenciu objektov
- používa sa na zistenie, či ide o rovnaký objekt
  - operátor `==` rozhoduje, či ide o dve referencie na jeden objekt – rovnosť identity objektov
  - pre `equals` je smerodajné ekvivalencia objektov
    - reťazce sú ekvivalentné, ak ide o rovnakú postupnosť znakov, nezávisle na tom, či je to ten istý objekt
    - dve inštancie reprezetujú jednu osobu, ak majú rovnaké učo

equals

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

if (s1==s2){
    System.out.println("s1 == s2");
} else {
    System.out.println("s1 != s2");
}

if (s1.equals(s2)) {
    System.out.println("s1.equals(s2)");
} else {
    System.out.println("!s1.equals(s2)");
}
```

`hashCode`

- počíta ***hash*** objektu
- používa sa v hashovaných dátových štruktúrach
- dva „`equals`“ objekty musia mať rovaký *hash*

=> ak prekryjete **`equals`**,

prekryte i **`hashCode`**

- `hashCode` by mal byť netriviálny
  - nie `return 1;`

`toString`

- krátka textová informácia o objekte
- napr. pre ladiace účely
- default:

```
<deskriptor><plné meno triedy>#hash
```

- je užitočné prekryť

konštanty

- akoby atribúty
- avšak
  - `static` – konštanty triedy, jedna hodnota v jvm
  - `final` – nemožno meniť za behu
  - NAZOV_KAPITALKAMI_S_PODTRZITKOM
- príklad:

```
public static final int
    LIGHT_VELOCITY = 299 792 458;
```

`final`

- dobrá ochrana pred programátorovými chybami
- sémantika závisí od „operandu“
- prvé dva obmedzujú dedičnosť
- `final class`
  - nie je možné rozširovať
  - mali by ste vždy, ak triedu nenavrhujete na rozširovanie [Bloch]
- `final method`
  - nie je možné prekryť v podtriede

`final`

- ďalšie umožňujú niektoré **runtime optimalizácie** – používajte, kde to len ide
- `final atribute`
  - hodnotu je nutné nastaviť v konštruktore
  - potom nie je možné meniť
- `final variable`
  - hodnotu je možné priradiť len raz
- `final parameter`
  - hodnotu parametra nie je možné v metóde meniť

volanie konštruktoru v konštruktore

- viac konštruktorov v triede – všeobecnejšie i špeciálne
  - všeobecnejšie – veľa parametrov
  - špeciálne – predvolené hodnoty za parametre

  –> volanie jedného konštruktoru z iného pomocou **`this(param...)`**
- zabraňuje **duplikácii kódu**

  => ľahšia udržiavateľnosť, body :-)

volanie konštruktoru v konštruktore

- dedičnosť
- konštruktor v nadtriede i podtriede

  –> volanie konštruktoru predka z konštruktoru potomka pomocou **`super(param...)`**

cykly

- používajú sa, ak sa má nejaký kus kódu vykonávať opakovane
  - `for`
    - pevný počet opakovaní
    - najbežnejší a najprehľadnejší cyklus
  - `while-do`
    - telo cyklu sa vykonáva, kým podmienka platí
  - `do-while`
    - detto, avšak podmienka sa kontroluje na konci => telo sa vykoná minimálne raz
- každý zmysluplný program obsahuje cyklus

pole

- lineárna homogénna dátová štruktúra
  - ? lineárna
  - ? homogénna
- pravdepodobne najjednoduchšia
- programátorov pohľad – „očíslované“ premenné

pole

- pamäťový pohľad – súvislý blok pamäte
  - adresa prvku = báza + n· veľkosť prvku
- predpoklad
  - všetky prvky majú rovnakú veľkosť (homogénnosť)
- dôsledok
  - polia číslujeme od nuly

pole – syntax

- deklarácia

```
int[] elements;
```

- inicializácia

```
elements = new int[20];
```

- optimálne

```
int[] elements = new int[20];
```

- prístup k prvku

```
elements[4] = 5;
int x = elements[4];
```

prechádzanie poľa

- cyklus for

```
int sum = 0;
for (int i = 0; i < elements.length; i++) {
   sum += elements[i];
}
```

- alebo elegantnejšie – for-each

```
int sum = 0;
for (int i : elements) {
   sum += elements[i];
}
```

veľkosť poľa

- veľkosť prvku
  - primitívny typ (`int`, `double`, `boolean`...)
    - veľkosť prvku = veľkosť typu
  - objektový typ
    - v poli len odkaz na haldu (pár bytov)
- ak veľkosť poľa nevyhovuje

```
elements =
    Arrays.copyOf(elements, newLength);
```

  - lineárna zložitosť => zväčšovať aspoň na dvojnásobok