PB162 Programování v jazyce Java

8. cvičenie

vnútrosemestrálka a úloha

- písomka
  - ušli bez opravenia!
  - nejde skompilovať –> žiaľ nula (**nápoveda!**)
  - inak dobré :-)
- úloha
  - časté chyby

písomka

- zadanie v ISe
  - /el/1433/podzim2009/PB162/cv/mo/za/
- čas 15 minút

javadoc

- `/** class Pigeon implemens interface Messaging...`
  - to je zrejmé z hlavičky triedy!

píšte **sémantický** javadoc, nie syntaktický!

- nezačínajte
  - Class represents, ...simulates
  - This is a...

sémantický javadoc

- správny javadoc:
  - Pigeon is a flyer that can deliver messages. Delivery time depends on it's fitness and distance.
  - Mail is a common mail delivering messages. Standard delivery time can be shorten with use of express delivering.
  - DeliveryServiceImpl is a setupable delivery service: you can choose type of service provider. Each change of configuration introduces new provider (they are not chached)

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];
```

- najlepšie spojiť deklaráciu a inicializáciu

```
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

od pochybenia k zlyhaniu

- pochybenie (*fault*)
  - (v prostredí programových systémov) vzniká (najmä) nedokonalosťou ľudských mentálnych pochodov
  - napr. programátor očakáva, že `x % 4` vracia čísla 0..3
  - rozpor medzi očakávanou a skutočnou sémantikou aritmetických výrazov, podmienok, konštrukcií, funkciou komponent...

od pochybenia k zlyhaniu

- defekt (*defect*)
  - miesto v systéme, ktoré vyzerá inak, ako by malo
  - napr. vo výraze chýba `+ 4`
- chyba (*error*)
  - manifestácia (prejav) pochybenia vnútri hraníc systému
  - napr. v premennej `y` je záporná hodnota

od pochybenia k zlyhaniu

- zlyhanie (*faulire*)
  - systém sa nespráva podľa očakávaní
  - preruší poskytovanie služby
  - službu poskytuje v rozpore so špecifikáciou
  - napr. kadmiové tyče sa úplne vysúvajú z aktívnej zóny...

hranice systému

- vždy záleží na hraniciach systému
- ak berieme ako systém celú elektráreň:
- chyba
  - „kadmiové tyče...“.
- zlyhanie
  - prerušenie dodávky elektrickej energie
  - výskyt ionizujúceho žiarenia a spadu
    - „...došlo k neuspokojivej radiačnej situácii...“

pochybenia, defekty

- boli, sú, budú...
- pochybenia programátora (*bugs*, boty), problémy v okolí (hardware, sieť)...
- musíme zabrániť, aby prerástli v zlyhanie
- vo vnútri komponenty – hľadáme **testovaním**
- v okolí – ošetrujeme **výnimkami**

výnimky

- záležitosť komunikačného rozhrania
- komponent reaguje na **neprípusté okolnosti**, z ktorých sa na úrovni komponentu nemožno rozumne zotaviť
  - nevhodné argumenty metódy
  - nevhodný stav komponenty (tj. nekorektná postupnosť volania)
  - problém na nižších vrstvách
  - ...

výnimky – delenie

1. behové (*runtime*)
   - potomok `RuntimeException`
2. kontrolované

- v prípade, že sa klient môže z daného stavu zotaviť, je vhodné použiť kontrolovanú výnimku

výnimky – životný cyklus

- vyhodenie výnimky

```
throw new SomeException(parameters...);
```

  - kontrolované výnimky musia byť uvedené klauzulou **throws**

```
public void myDangerousMethod() throws
        RecoverableException {
    …
}
```

výnimky – životný cyklus

- spracovanie výnimky
  - poslať vyššie (do volajúcej metódy)
  - zachytiť

```
try {
    myDangerousMethod();
} catch (RecoverableException e) {
    doSomething(e);
} finally {
    cleanImportatnResource();
}
```

„obľúbené“ výnimky

- behové
  - `NullPointerException`
  - `IllegalArgumentException`
  - `InvalidStateException`
  - `ArrayIndexOutOfBounds`
  - `ClassCastException`