# PB162 Programování v jazyce Java

9. cvičenie

# chyby v 4. úlohe

- funkčnosť
  - chýbalo vynulovanie času
  - `getWinner()`: do lokálnej premennej ste si mohli rovno ukladať zatiaľ najlepšieho
- javadoc
  - úplne chýbal alebo bol nezmyselný
- výstavba metód
  - výnimočné stavy ošetrujte na začiatku, po jednom – zjednoduší to kód

# chyby v 4. úlohe – výnimky

- Výnimkám nastavujte správy. Ak test zlyhá na zlej hodnote, treba ju do správy zahrnúť
- výnimkám nastavujte správy. Ak test zlyhá na zlej hodnote, treba ju do správy zahrnúť
- nezachytávať `NullPointerException`, kontrolovať pomocou `==`
- avšak nekontrolovať znova, čo kontroluje volaná metóda
- za `throw` nemusí byť `else`, vyhodenie výnimky beh preruší

# pretypovanie

- test na príslušnosť k typu: **`instanceof`**

```
if (obj instanceof CompactDisk) ...
if (obj.getClass() == CompactDisk.class) ...
```

  - rozdiel?
- pretypovanie na typ: **`()`**

```
CompactDisk cd = (CompactDisk)obj;
```

- vyhadzuje výnimku `ClassCastException`

# `equals` a `hashCode`

- ak sú dva objekty `equal`, musia mať rovnaký `hashCode`!
- ukážka na príklade

```
public class CompactDisk {
    private final String albumName;
    private final String[] songs =
        new String[16];
```

# dynamické dáta

- ...ako modelovať štartovaciu listinu?
  - použiť pole
    - pre programátora zložité na obsluhu
      - musíte programovať `isEnrolled` pomocou cyklu...
      - pridávanie pretekára
        - na koniec – zväčšovanie poľa
        - doprostred – detto + presúvanie prvkov
      - odoberanie pretekára
    - náchylné na chyby
      - napr hrozí memory leak
- dôvod problémov – štartovacia listina je dynamická

# dynamické dáta

- potrebujeme dynamickú dátovú štrukúru
  - vybudovanú nad poľom, binárnym stromom, pomocou hashovania
  - vybudovanú dobre
    - bez chýb, odskúšanú
    - efektívnu / so známym výkonom
- toto existuje v podobe rámca kolekcií – Java Collections Framework
  - hlavná stránka , triedy, tutoriál

# Java Collections Framework

- súčasť Java Core API od v. 1.2
- poskytuje pohodlné rozhranie na prácu s dynamickými dátovými štruktúrami
- vyššia úroveň abstrakcie
  - množina, zoznam, fronta...
- usporiadanosť kolekcií
  - explicitne, implicitne usporiadané
  - neusporiadané

# štartovacia listina

- štartovacia listina – papier s menami v očíslovaných riadkoch
- avšak pretekár môže byť najviac raz
- ide **množinu štartujúcich pretekárov**

# rozhranie `java.util.Set` = množina

- matematické poňatie množiny
  - neusporiadané dáta
  - objekt nanajvyš raz
- metódy
  - `add`
  - `contains`
  - `remove`
- implementované triedami
  - `HashSet` – pomocou hashovania
  - `TreeSet` – pomocou čierno-bieleho stromu

# `java.util.Set` – použitie

- použitie `HashSet`
  - ak prekyjeme `equals`, musíme i `hashCode`
  - lepšia časová zložitosť
    - elementárne operácie (`add`, `contains`, `remove`) v *O(1)*
- použitie `TreeSet`
  - preberieme o týždeň
  - automaticky a priebežne sa usporiadava
  - cena: časová zložitosť je horšia
    - elementárne operácie (`add`... `remove`) v *O(log n)*

# `java.util.Set` – použitie

- import balíčkov

```
import java.util.Set;
import java.util.HashSet;
```

- vytvorenie inštancie

```
private final Set<Racer> racers = new HashSet<Racer>();
```

- pridanie linky
  - `racers.add(racer);`
- dopyt na prítomnosť
  - `racers.contains(racer);`

# `java.util.Set` – použitie

- odobratie linky
  - `racers.remove(racer);`
- vyprázdnenie množiny
  - `racers.clear()`
- iterácia – for-each

```
for (Racer racer: racers) {
    System.out.println(racer);
}
```

# nemodifikovateľné kolekcie

- kolekcia ako atribút triedy
- chceme sprístupniť len na čítanie
  - nedostatočné riešenie:

```
private Set<Racer> racers;
…
public Set<Racer> getRacers() {
    return racers;
}
```

klient môže volať `getRacers().clear();`

# nemodifikovateľné kolekcie

- kolekcia len na čítanie
  - použitie obálky

```
…
public ???<Racer> getRacers() {
    return Collections.
        unmodifable???(racers);
}
```

  volanie modifikujúcich metód spôsobí **výnimku**
- trieda Collections – statické metódy na všeobecnú prácu s kolekciami

# mapy = asociatívne polia

- pole – indexované číslami `0..length - 1`
- asociatívne pole – indexované **objektami**
- množina objektov – **kľúče**
  - každý iba raz, na poradí nezáleží
  - `getKeySet()`
- na každý kľúč viazaný nejaký objekt – **hodnota**
  - všeobecná kolekcia
  - `getValues()`

# mapy = asociatívne polia

- metódy
  - `put`, `remove`
  - `containsKey`, `containsValue`
  - `get`
- implementované triedami
  - `HashMap` – pomocou hashovania
  - `TreeMap` – pomocou čierno-bieleho stromu

# `java.util.Map` – použitie

- import balíčkov

```
import java.util.Map;
import java.util.HashMap;
```

- vytvorenie inštancie

```
private final Map<Person, Ticket> tickets =
  new HashMap<Person, Ticket>();
```

- pridanie lístku
  - `tickets.put(me, ticket);`
- dopyt na prítomnosť
  - `tickets.contains(me);`

# java.util.Map – použitie

- odobratie lístku
  - `tickets.remove(me);`
- vyprázdnenie mapy
  - `tickets.clear()`
- iterácia – for-each

```
for (Ticket ticket: tickets.values()) {
    System.out.println(ticket);
}
for (Person p: tickets.keySet()) {
    System.out.println(tickets.get(p));
}
```

# 5. úloha

- musia prejsť testy!
- domácu časť odovzdať do soboty polnoci (ako vždy)
- pri vkladaní do odovzdavárne:
  - Název: doma
- nemeňte balíček, rar nie je jar...
- v `ResultsImpl` najprv implementuje `StartingList`, potom `Results`
  - použite už existujúce metódy