@RoboNovotny

UINF/PAZ1c

epizóda 8

11/nov/15

divoký západ dedičnosti

Dediť či nedediť?

- naozaj je vzťah **is-a**? ✅
- implementujete **interfejs**? 🚫
- je **rodič explicitne navrhnutý** na dedenie? ✅
- máte dosah na **implementáciu** rodiča? ✅
- dedíte od tried **v rovnakom balíčku**? 🚫

viď tutoriály + oficiálna dokumentácia

Liskovovej substitučný princíp (1987)

Ak pre každý objekt o1 typu T1 existuje objekt o2 typu T2 taký, že pre všetky programy P využívajúce T2 platí, že po nahradení objektu o2 objektom o1 sa správanie P nezmení, potom T1 je podtypom T2

- ak v programe nahradíme triedu podtriedami, správanie sa zachová.
- ak nahradíme inštancie elíps inštanciami kružníc, správanie sa zrejme poruší

Formálnejšie zásady pre Liskovovej princíp

- **preconditions nemožno** v podtriede **zosilniť**
  - v podtriede **Kružnica** polos *e* = polos *f*
- **postconditions nemožno** v triede **zoslabiť**
  - **Elipsa#**zmeňPolosF(): postpodmienka `nováPolosE == predošláPolosE`
  - **Kružnica#**zmeňPolos: podmienka nemusí platiť
- **invarianty** musia ostať nezmenené

Základný kritický bod dedičnosti

Dedičnosť narúša zapúzdrenie!
(Inheritance breaks encapsulation!)

- potomok môže meniť interný stav rodičovskej triedy…
- …a tým narúšať invarianty rodiča!

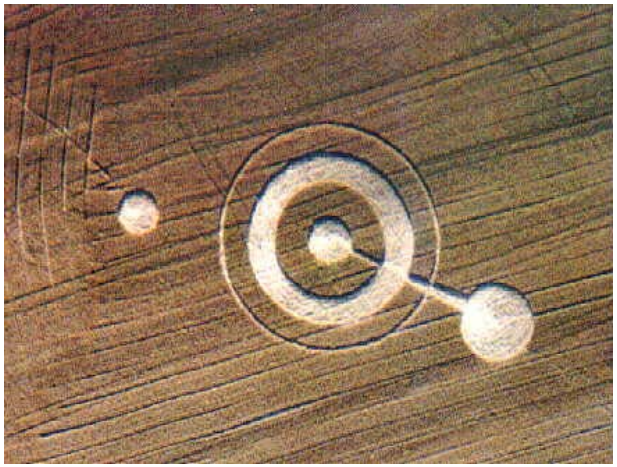
Dedičnosť narúša zapúzdrenie

- Klient uzatvára **kontrakt** s triedou, ktorý je reprezentovaný **verejnými** metódami
- Oddedená trieda je však tiež len klient rodičovskej triedy…
- …má sa riadiť **kontraktom**

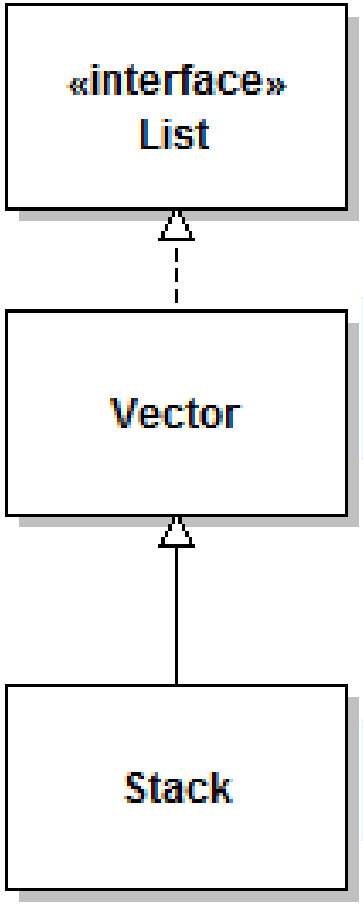
Problém je v meniteľnosti

- problém spočíva v **meniteľnosti inštancií**
  - keby sme zabránili zmene atribútov inštancie, problém by sa vyriešil
  - raz vytvorená kružnica bude navždy kružnicou

Riešenie: zrušíme settre, atribúty možno nastaviť *len* v konštruktore.



Zlý príklad!



- narúša to pravidlo **„is-a"**
  - „zásobník nie je vektorom založeným na poli"
- narúša to **LSP**
  - zásobník je zoznam, do ktorého možno vkladať len na koniec

od JDK6:
interface Deque + implementácia LinkedList

Liskovovej princíp je hrôza!

- ale veď to **úplne popiera dedičnosť**!
- prečo som sa to učila, keď je to zbytočne?
- dedičnosť nie je zlá, len ju treba používať s rozvahou

Trieda má jasne stanoviť, kedy a za akých podmienok je vhodné dediť!

Dve protibežné zásady

Dedičnosť

Trieda zdedí správanie a stav od rodiča

Zapúzdrenie

Trieda môže meniť stav inej triedy len skrz špecifikované metódy

Dizajn = balansovanie

- ideálny svet:
  - platí LSP, inštančné premenné sú **private**, oddedená trieda pristupuje k rodičovskému stavu cez metódy
- v praxi:
  - LSP sa narúša, lebo komplikuje veci
  - **protected** premenné a metódy

Uprednostňujte kompozíciu pred dedičnosťou!

-- Gang of Four, autori Design Patterns

nástroje pre dedičnosť

príkaz dedičnosti

```
abstract class AbstractService
{
    abstract run();
}
```

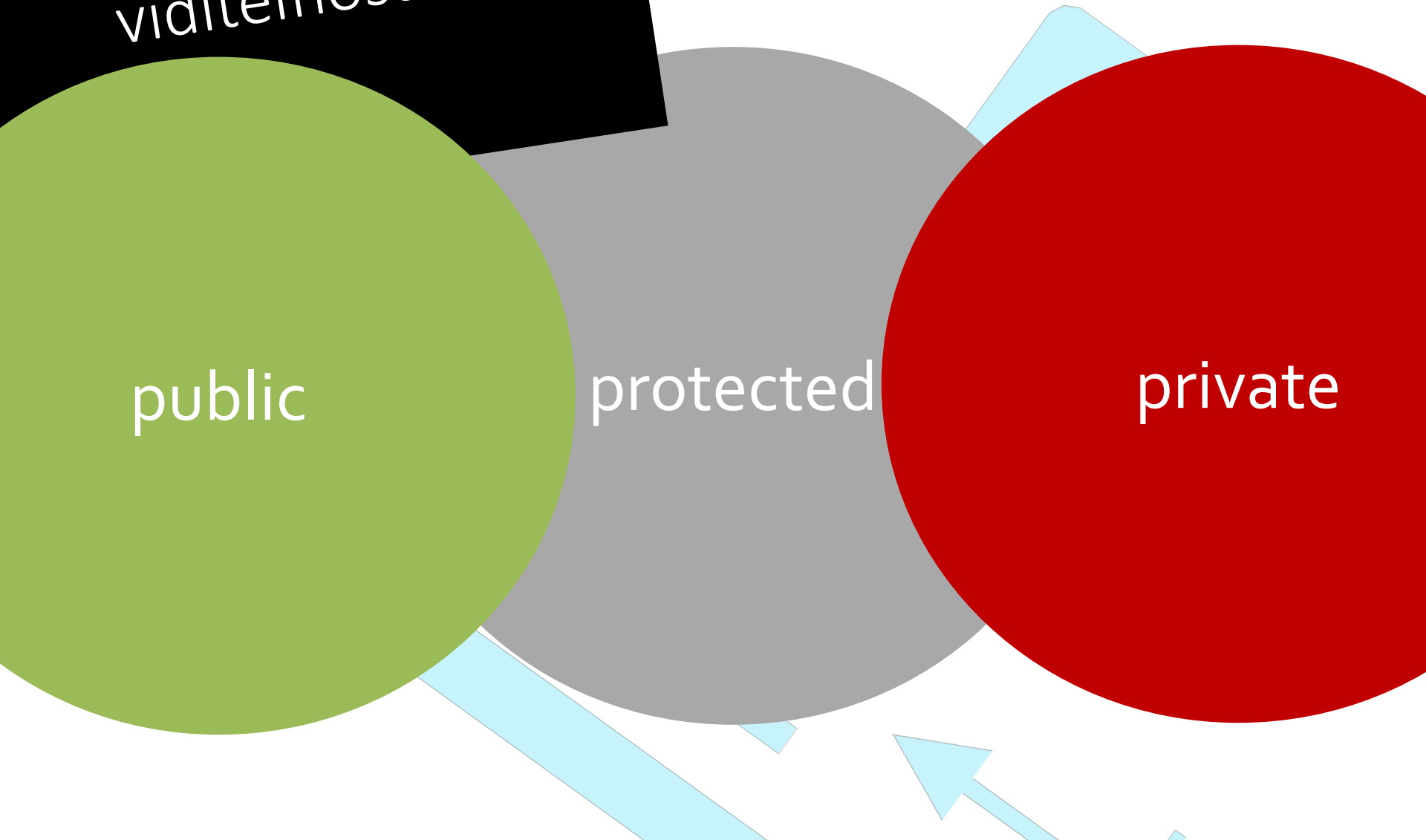
zákaz dedičnosti

```
final class String{
        ...
}
```

zákaz dedičnosti

```
class Boháč{
        public final void vydeď()
}
```

modifikátory viditeľnosti



rozšírenie cez dohodnutý kontrakt

Interfejs ako spôsob rozšírenia

- všeobecný algoritmus, ktorý potrebuje komunikovať s klientom
- **filter:** kedy prvok vyhodiť a kedy akceptovať?
- **triedenie:** kedy je prvok menší než iný?
- **vyťahovanie z SQL**: ako namapovať riadok na objekt?

Filtrovanie: rozšírenie cez interfejsy

```
interface FileFilter {
  public boolean accept(File file);
}
```

java.io.File

```
public File[] listFiles(FileFilter filter)
```

algoritmus záleží na prijatí/zamietnutí prvku

Vyťahovanie z SQL

```
interface RowMapper {
  Object mapRow(ResultSet rs, int index);
}
```

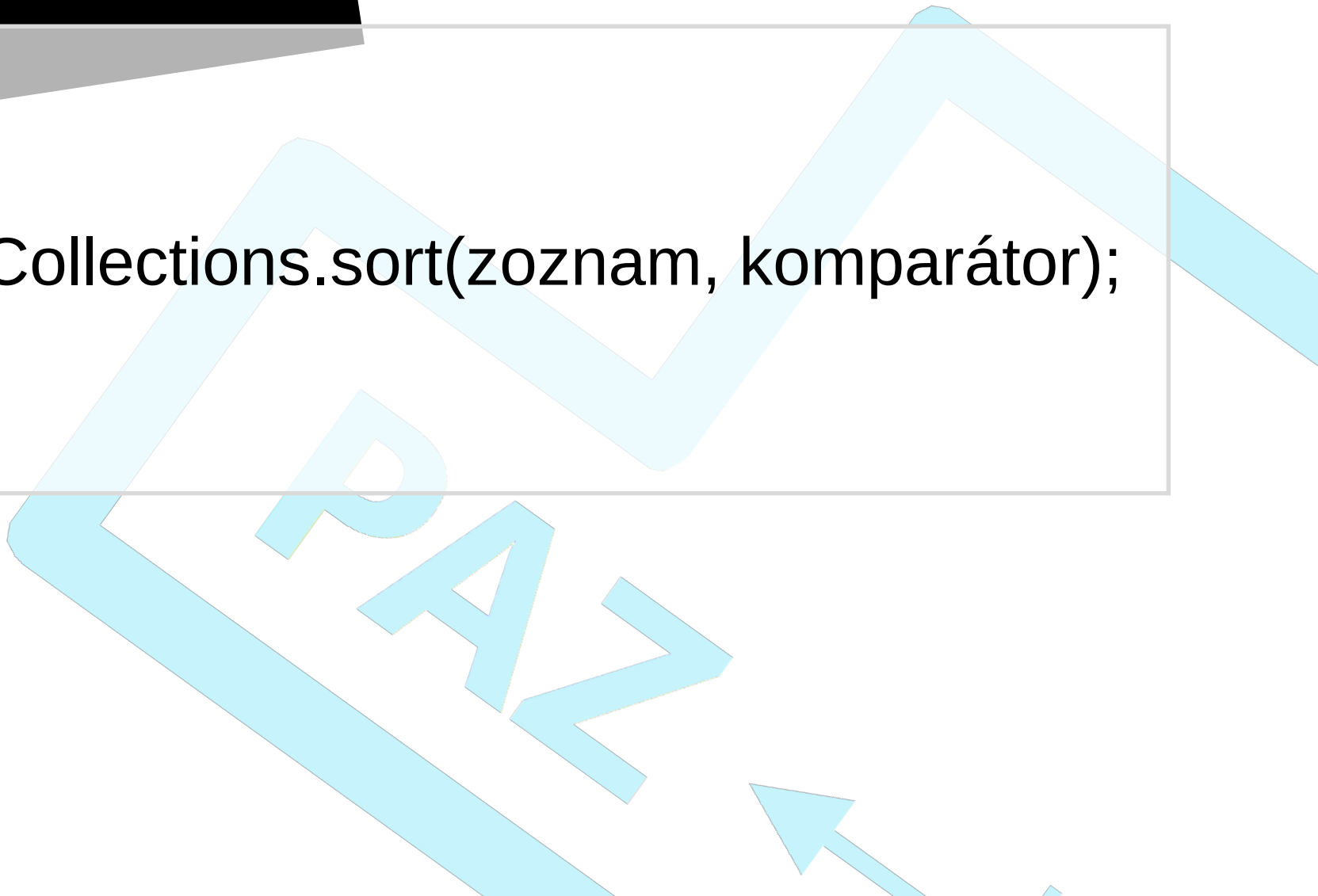
Spring JdbcTemplate

```
public List<Object> query(String sql, RowMapper rm);
```

algoritmus záleží na spracovaní riadkov

triedenie v Jave

```
Collections.sort(zoznam, komparátor);
```



triedenie v Jave

```
class FileComparator
        implements Comparator<File>
{
        int compare(File f1, File f2) {
                ...
        }
}
```

triedenie v Jave

```
Comparator<File> cmp
        = new FileComparator();

Collections.sort(zoznam, cmp);
```

Syntaktický cukor

- môžeme na jednom riadku
  1. **deklarovať** triedu,
  2. tá bude **implementovať** interfejs
  3. a automaticky vytvoriť jej **inštanciu**

triedenie v Jave

```
Comparator<File> cmp
        = new Comparator<File>();

Collections.sort(zoznam, cmp);
```

comparator

```
Comparator<File> cmp = new Comparator<File>() {



};
```

comparator

```
Comparator<File> cmp = new Comparator<File>() {
    public int compare(File f1, File f2) {
        return Long.compare(f1.length(), f2.length());
    }
};
```

comparator

```
Comparator<File> cmp
   = new /* C, kde C extends */ Comparator<File>() {
    public int compare(File f1, File f2) {
          return Long.compare(f1.length(), f2.length());
    }
};
```

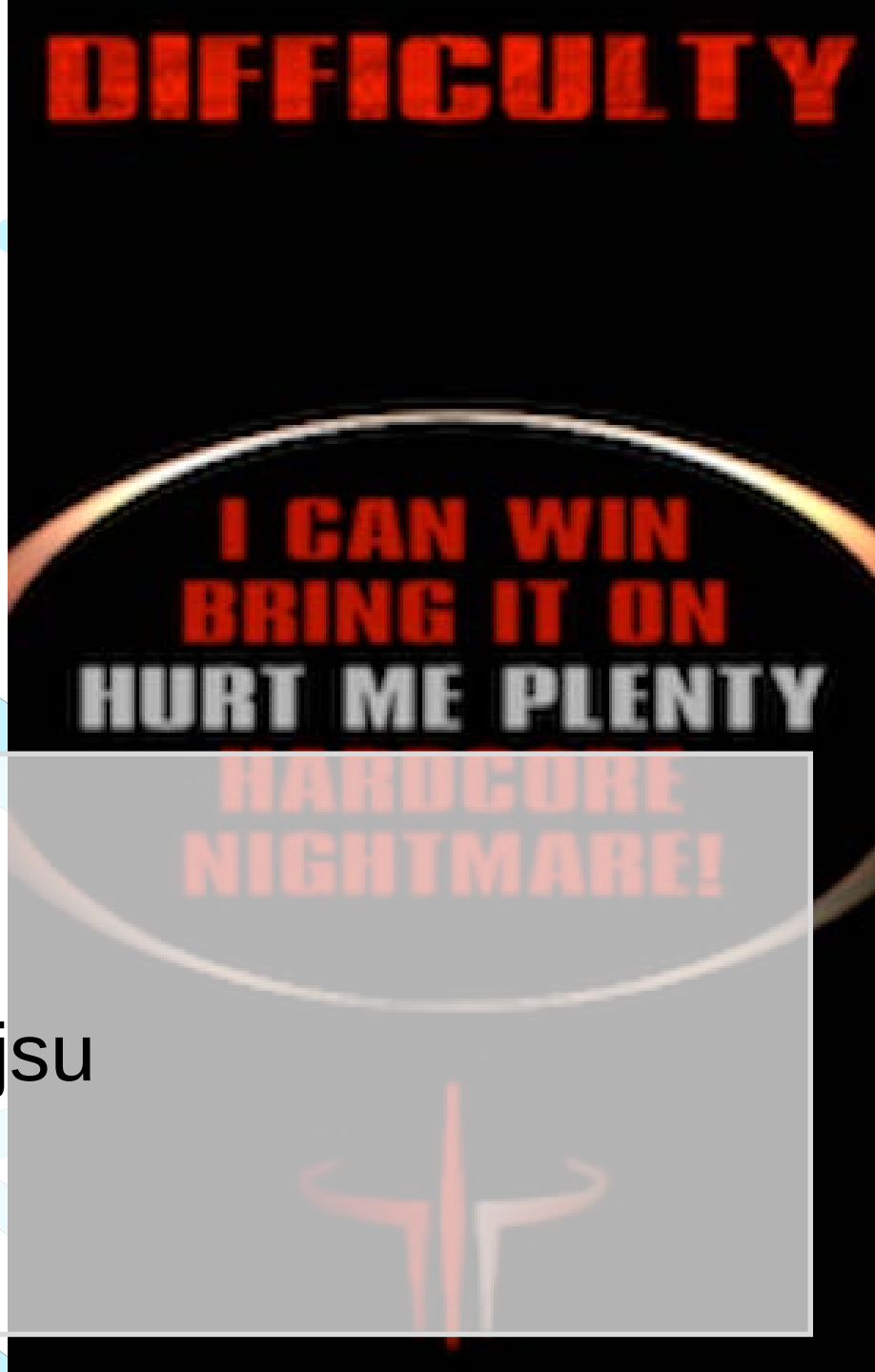
anonymná vnútorná trieda

```
Comparator<File> cmp
   = new /* C, kde C extends */ Comparator<File>() {
    public int compare(File f1, File f2) {
          return Long.compare(f1.length(), f2.length());
    }
};
```

anonymná vnútorná trieda

```
Comparator<File> cmp
   = new /* C, kde C extends */ Comparator<File>() {
    public int compare(File f1, File f2) {
          return Long.compare(f1.length(), f2.length());
    }
};
```

anonymná vnútorná trieda



V jednom kroku:

- implementácia interfejsu
- vytvorenie inštancie

IT'S JAVA TIME!
NO, DAD! NO!

demo!
[avt]

vnútorná trieda vidí metódy i premenné svojho obalu

vnútorná trieda vidí len finálne lokálne premenné z vonkajšej metódy

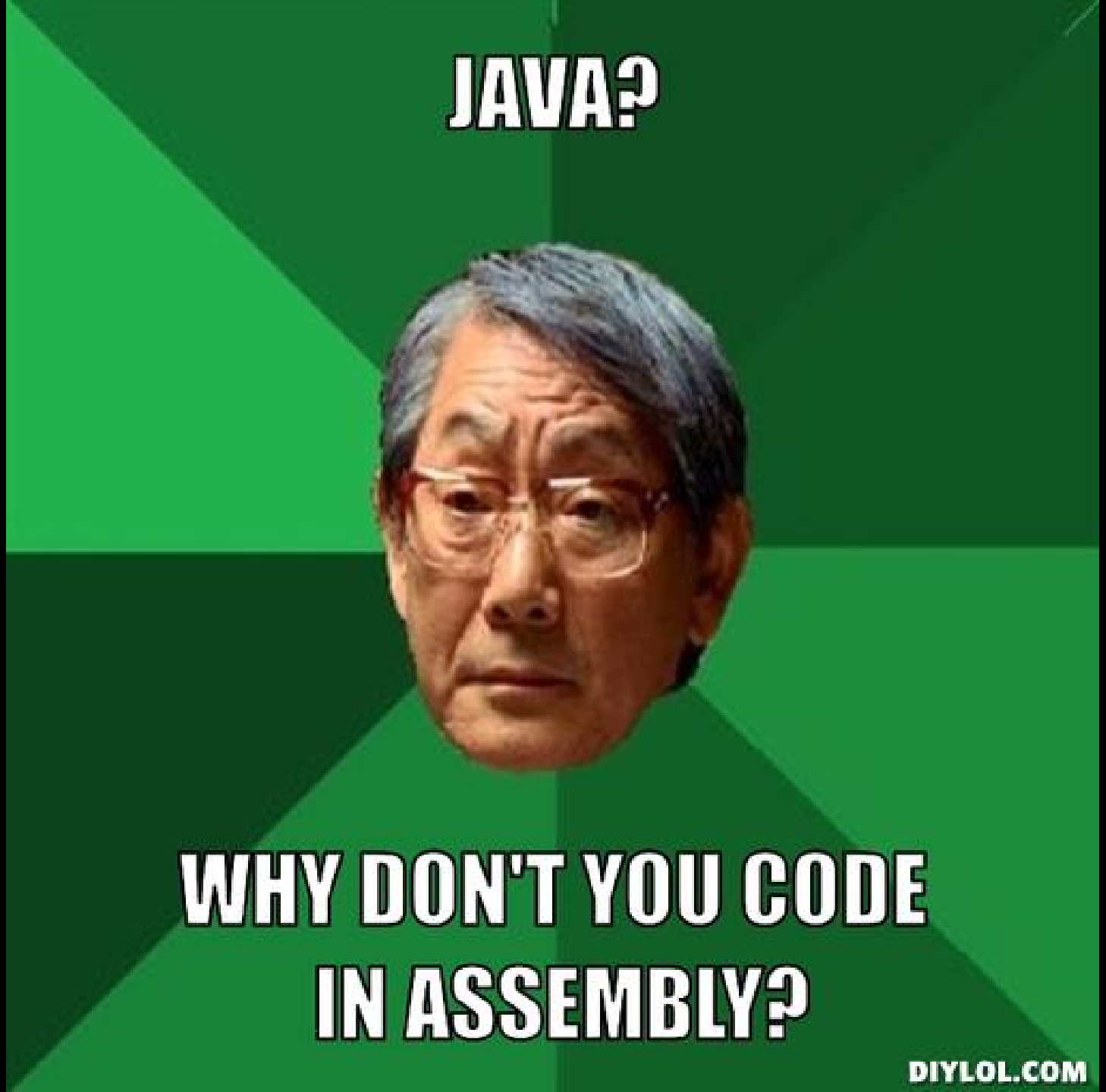
Cannot refer to a non-final variable inside an inner class

ak vnútorná trieda potrebuje meniť obsah lokálnej premennej, musí použiť obal

The final local variable cannot be assigned, since it is defined in an enclosing type

čítanie

http://ics.upjs.sk/~novotnyr/blog/1174/dizajnove-criepky-anonymne-vnutorne-triedy


JAVA?
WHY DON'T YOU CODE IN ASSEMBLY?
DIYLOL.COM