Informatika 2

Svetlé a temné stránky dedičnosti

doc. Ing. Ján Janech, PhD.
18. 3. 2024

ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia
a informatiky

Pojmy zavedené v 4. prednáške (1)

- odstránenie duplicít medzi triedami
  - skladanie – kompozícia
  - dedičnosť
- dedičnosť
  - typológia
- trieda dedí
  - vonkajší pohľad
  - vnútorný pohľad
  - nededí konštruktory

Pojmy zavedené v 4. prednáške (2)

- interné ukrývanie informácií

- vzťahy pri dedičnosti
  - predok
  - potomok
  - priamy predok
  - priamy potomok
  - absolútny predok

Pojmy zavedené v 4. prednáške (3)

- typy dedičnosti
  - jednoduchá – strom dedičnosti
  - jednoduchá – les dedičnosti
  - viacnásobná – mriežky dedičnosti

Pojmy zavedené v 4. prednáške (4)

- porovnanie skladania a dedičnosti
  - znovupoužiteľnosť
  - implementačná závislosť

- dedičnosť a interface

Pojmy zavedené v 4. prednáške (5)

- dedičnosť a typová kompatibilita
- dedičnosť a polymorfizmus
- prekrývanie metód
  - kľúčové slovo super

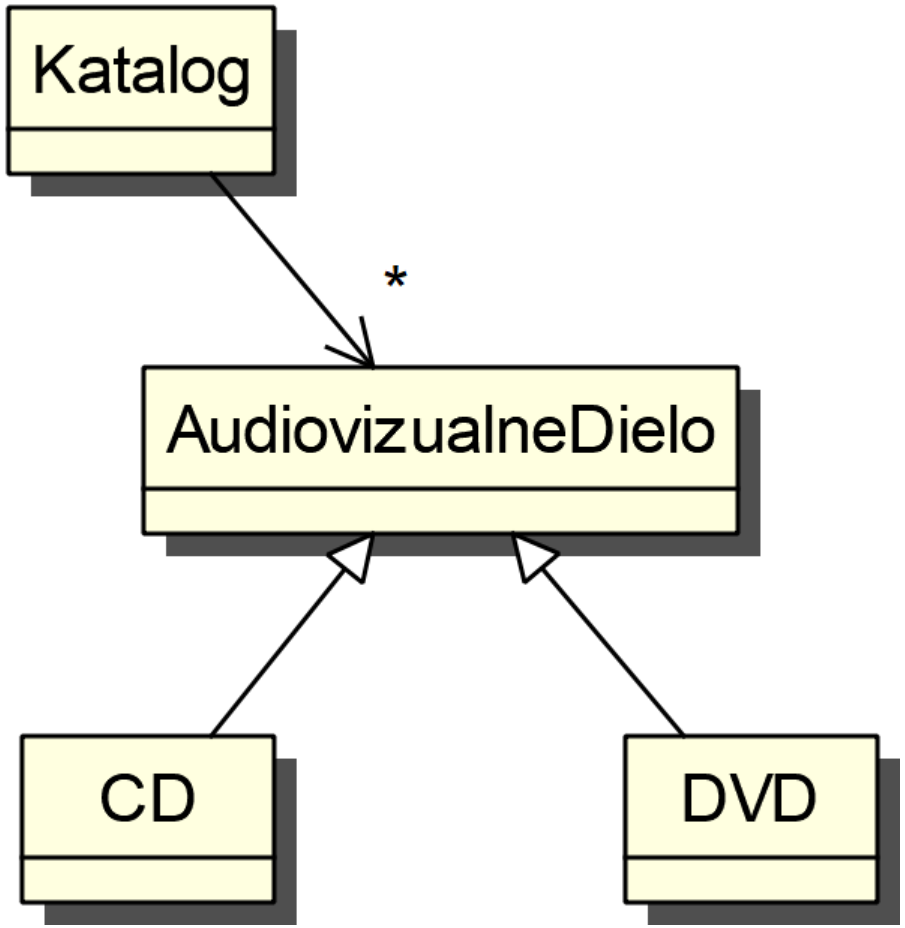
- jazyky – implementácia polymorfizmu

Cieľ prednášky

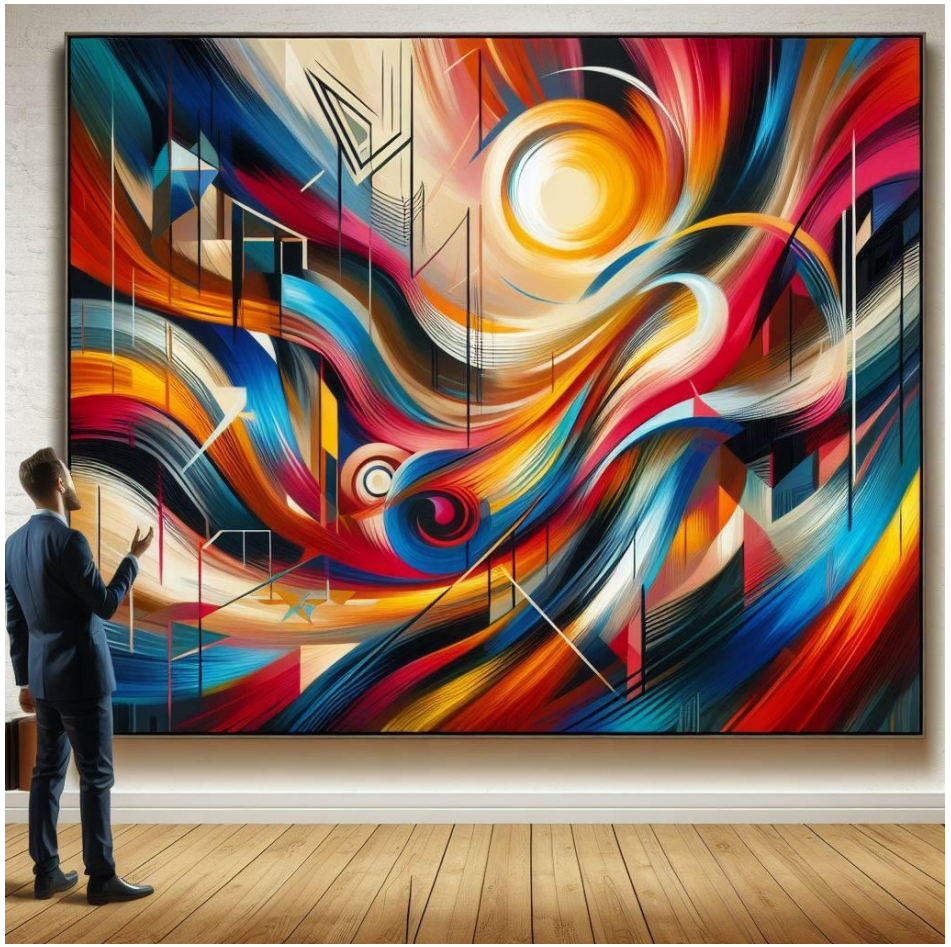
- abstraktná trieda
- vzťahy is-a, has-a
- extenzia triedy
- LSP pri dedičnosti
- trieda Object

- príklad: KCalB

KCaIB – súčasný stav



Abstraktná trieda



Abstraktná trieda – motivácia

- inštancie vytvorené triedou AudiovizualneDielo
  - nemajú význam
  - v skutočnosti neexistujú
  - vytvárať je technicky možné
  - abstraktná trieda
- trieda AudiovizualneDielo
  - prvotne odstránenie duplicity v triedach
  - predok pre rôzne typy avi diel
  - koreň hierarchie typov (domény)

Abstraktná trieda – vlastnosti

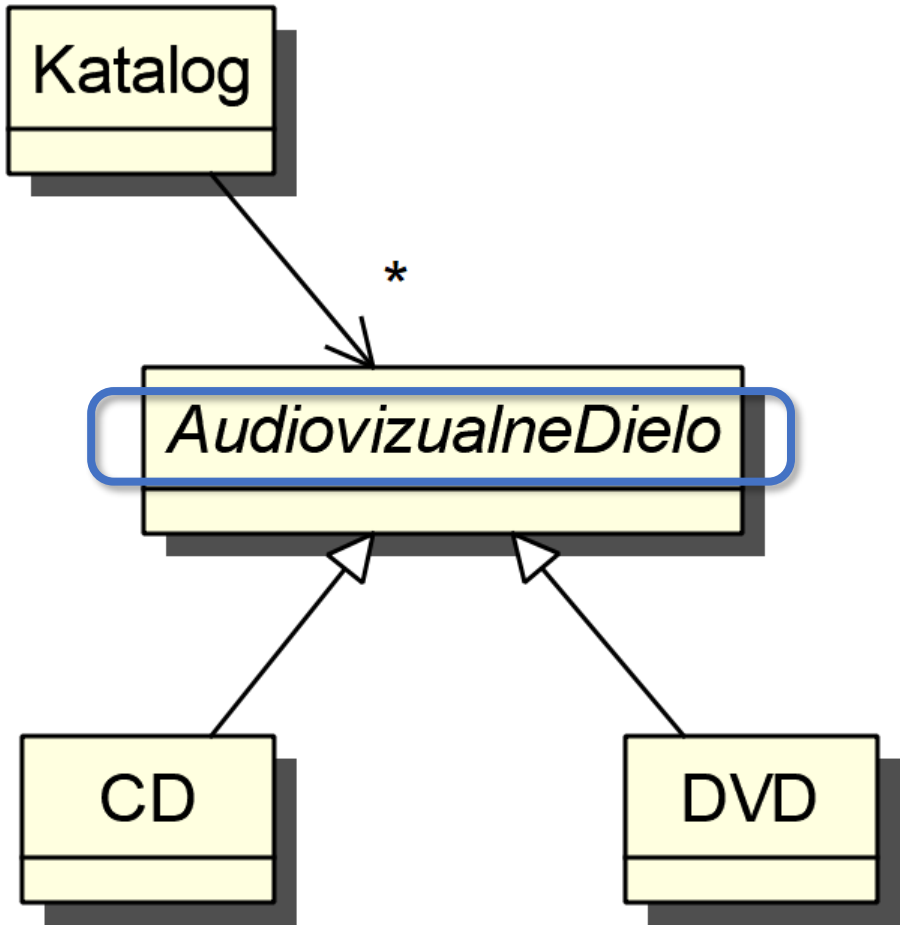
- charakteristická vlastnosť – nevytvára inštancie
- podľa jazyka
  - dobrovoľná – možno vytvoriť, nemá význam
  - povinná – nemožno vytvoriť, zákaz
- Java – možnosť označiť triedu ako abstraktnú

Konkrétna trieda

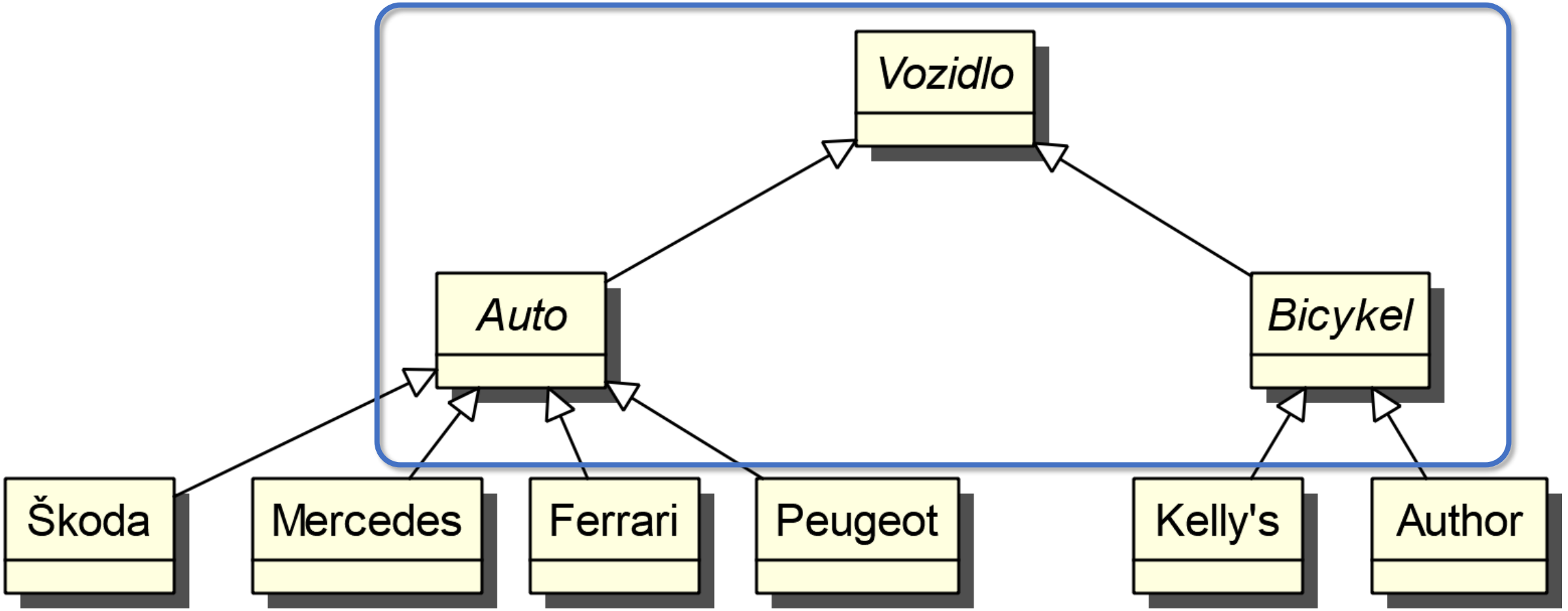
- inštancie vytvorené triedami CD a DVD
  - majú význam
  - existujú aj v skutočnosti

- konkrétna trieda
  - bežne vytvára inštancie

Abstraktná trieda – UML



Abstraktné triedy v reálnom svete (1)



Abstraktné triedy v reálnom svete (2)

- Trieda Vozidlo
  - definované chovanie (pohyb dopredu, odviesť človeka...)
  - nevieme si predstaviť jej inštanciu – vybavíme si konkrétne auto, alebo bicykel
  - abstraktná trieda

Abstraktná trieda – Java

- kľúčové slovo abstract
- hlavička triedy

```
public abstract class AudiovizualneDielo {
    // telo triedy
}
```

KCaIB – metódy vypis (1)

- AudiovizualneDielo – definícia metódy

- CD, DVD – prekrytie metódy

- polymorfizmus
- konečná podoba metódy – potomok

KCaIB – metódy vypis (2)

- AudiovizualneDielo – metóda nemá konečnú podobu
- CD, DVD – doplnenie informácie
  - môže vyžadovať iné usporiadanie informácií
- iné možnosti návrhu metódy v AudiovizualneDielo
  - prázdne telo metódy
  - abstraktná metóda

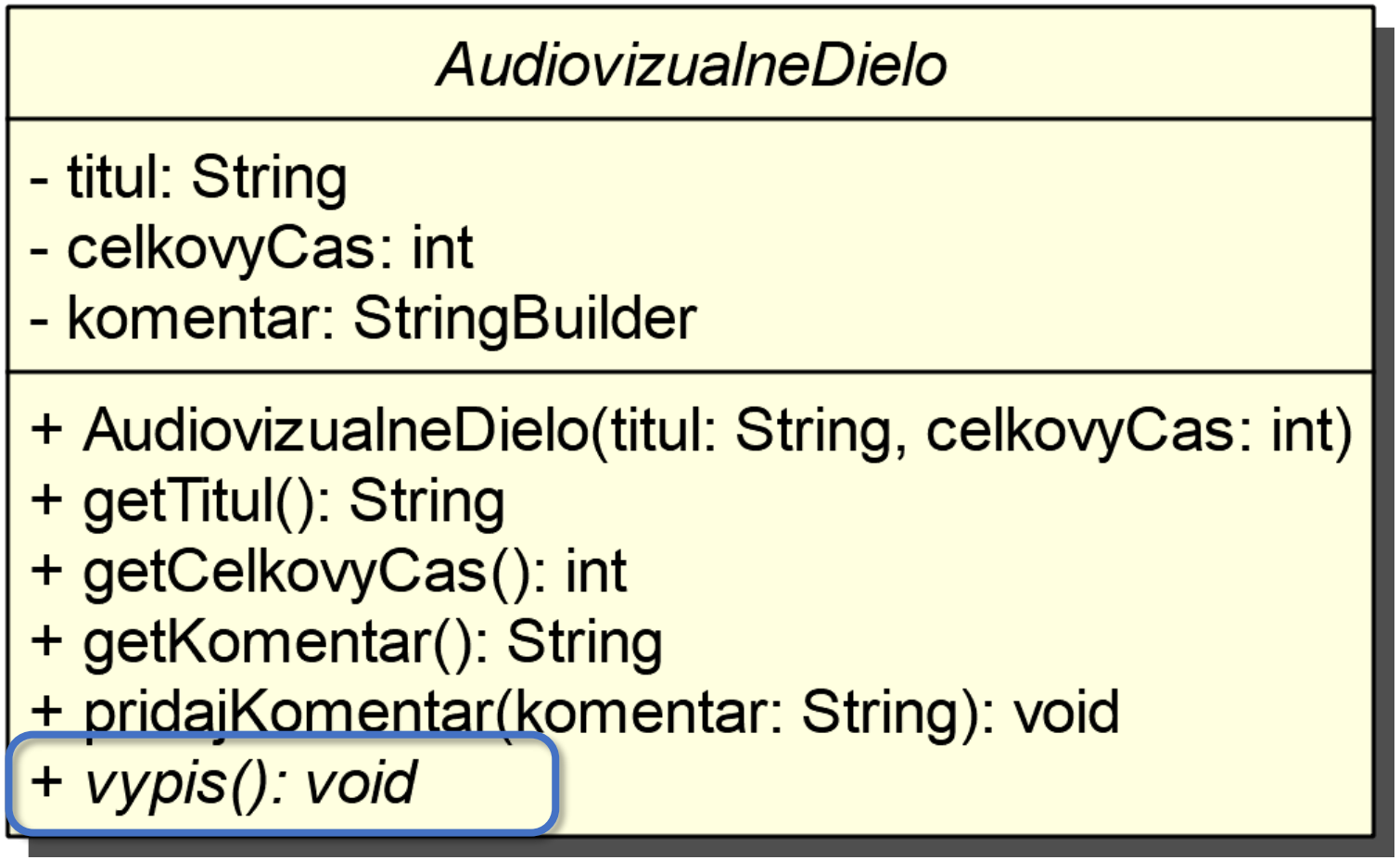
Abstraktná a konkrétna metóda

- metódy v abstraktnej triede
  - konkrétne
    - definícia tela metódy v tele triedy
    - definícia správy v rozhraní triedy
  - abstraktné
    - definícia správy v rozhraní triedy
- metódy v konkrétnej triede
  - len konkrétne
    - definícia tela metódy v tele triedy
    - definícia správy v rozhraní triedy

Abstraktná a konkrétna metóda – príklad

- konkrétna metóda
  - bicykel ide dopredu po potiahnutí pedálmi – bez ohľadu na konkrétny typ
- abstraktná metóda
  - bicykel vie zabrzdiť – konkrétna implementácia (čeľusťové brzdy, V-brzdy, kotúčové brzdy, hydraulické brzdy...) závisí na konkrétnom type

Abstraktná metóda – UML



Abstraktná metóda – Java

- jazyková konštrukcia
- zabezpečuje správu do rozhrania
- nemá žiadne telo

- kľúčové slovo abstract v hlavičke metódy
- hlavička ukončená bodkočiarou

```
public abstract void vypis();
```

AudiovizualneDielo – metóda vypis

```
public abstract class AudiovizualneDielo {
    ...
    public abstract void vypis();
    ...
}
```

Implementácia abstraktnej metódy

- ako bežná metóda

- nemá zmysel prekrývanie abstraktnej metódy
  - neexistujúca metóda sa nedá prekryť

=>

- nepoužíva kľúčové slovo super
  - super.abstraktnaMetoda() – syntaktická chyba

Abstraktná trieda vs. metóda

- abstraktná metóda – trieda musí byť abstraktná

- abstraktná trieda
  - nechceme vytvárať inštancie – voliteľná
  - definuje abstraktnú metódu – povinná
  - dedí abstraktnú metódu – povinná
  - implementuje abstraktnú metódu – podľa potreby

CD – metóda vypis (1)

```
public class CD extends AudiovizualneDielo {
    ...
    @Override
    public void vypis() {
        // príkazy tela metódy
    }
    ...
}
```

CD – metóda vypis (2)

```
System.out.println("CD:");
System.out.println("    Autor: " + this.autor);
System.out.println("    Titul: " + this.getTitul());
System.out.println("    Pocet skladieb: " +
                        this.pocetSkladieb +
                       " (celkovo " +
                        this.getCelkovyCas() + " minut)");
this.vypisKomentar();
```

Vzťah is-a

- is-a – „is a“ – anglicky „je“
- dedičnosť je realizáciou vzťahu is-a
- každé CD „je“ audiovizuálne dielo

=>

- každá inštancia CD je inštanciou AudiovizualneDielo

Dedičnosť a extenzia triedy

- extenzia triedy
  - množina všetkých inštancií
  - vzťah „is-a“ => aj inštancií potomkov

- dôsledok
  - abstraktné triedy môžu mať neprázdnu extenziu

Operátor instanceof – extenzia

```
prvyOperand instanceof DruhyOperand
```

- vracia true
  - prvyOperand je inštanciou triedy DruhyOperand

- rozšírenie:
- vracia true
  - prvyOperand patrí do extenzie triedy DruhyOperand

CD – metóda vypis – komentáre

```
System.out.println("CD:");
System.out.println("    Autor: " + this.autor);
System.out.println("    Titul: " + this.getTitul());
System.out.println("    Pocet skladieb: " +
                        this.pocetSkladieb +
                       " (celkovo " +
                        this.getCelkovyCas() + " minut)");

this.vypisKomentar();
```

Prístupové právo protected

- selektívne prístupové práva
  - „protekcia“ pre potomkov ☺

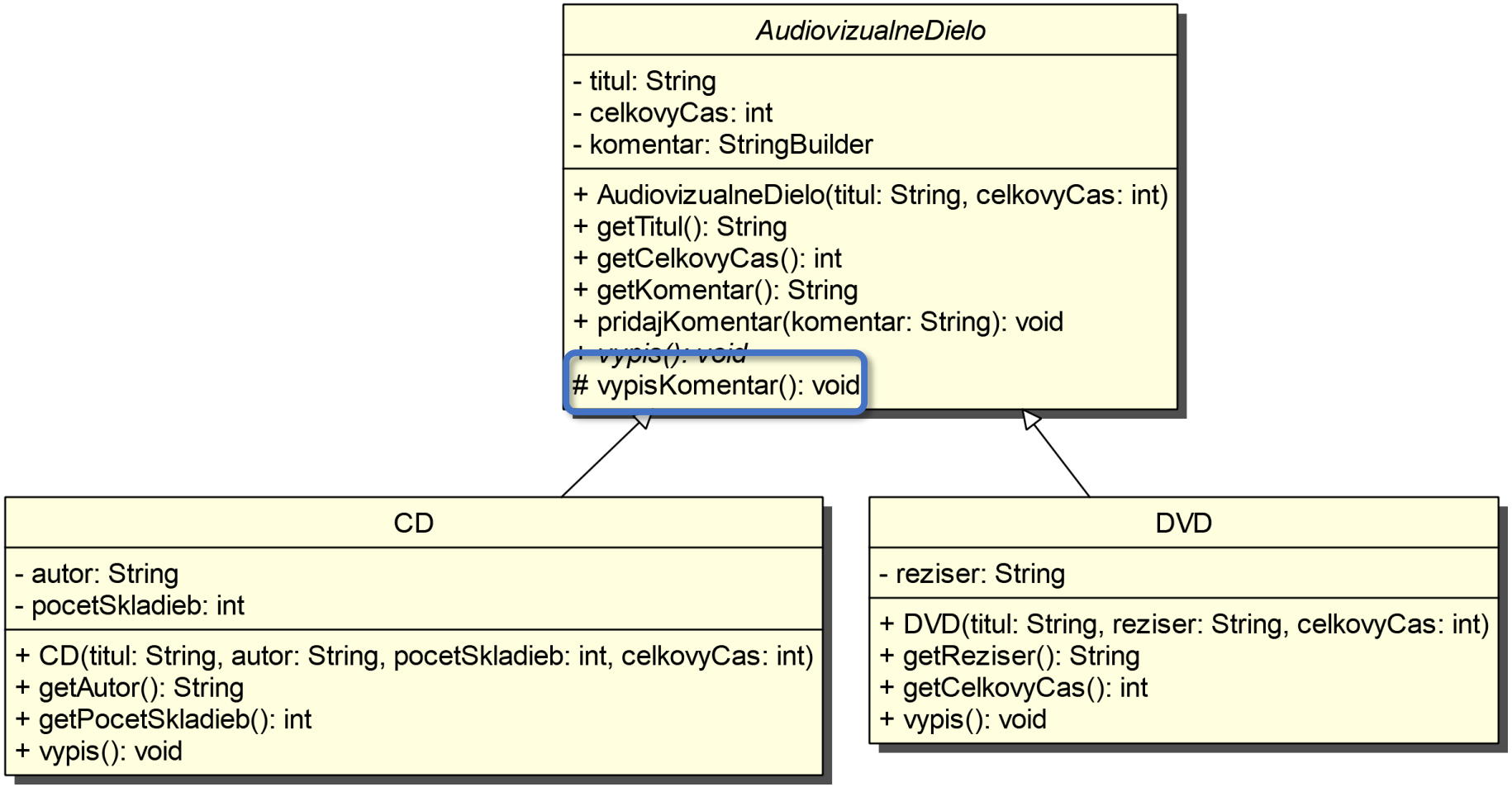
- public – verejný prístup všetkým iným objektom
- private – súkromný prístup samotného objektu

- protected
  - verejný pre každý objekt z extenzie
  - súkromný pre každý objekt mimo extenzie

Protected – Java

```
protected void vypisKomentar() {
    if (this.komentar.length() > 0) {
        System.out.println("Komentar ku dielu:");
        System.out.println(this.komentar);
    }
}
```

Protected – UML



Dedičnosť a návrh tried

- generalizácia
- špecializácia

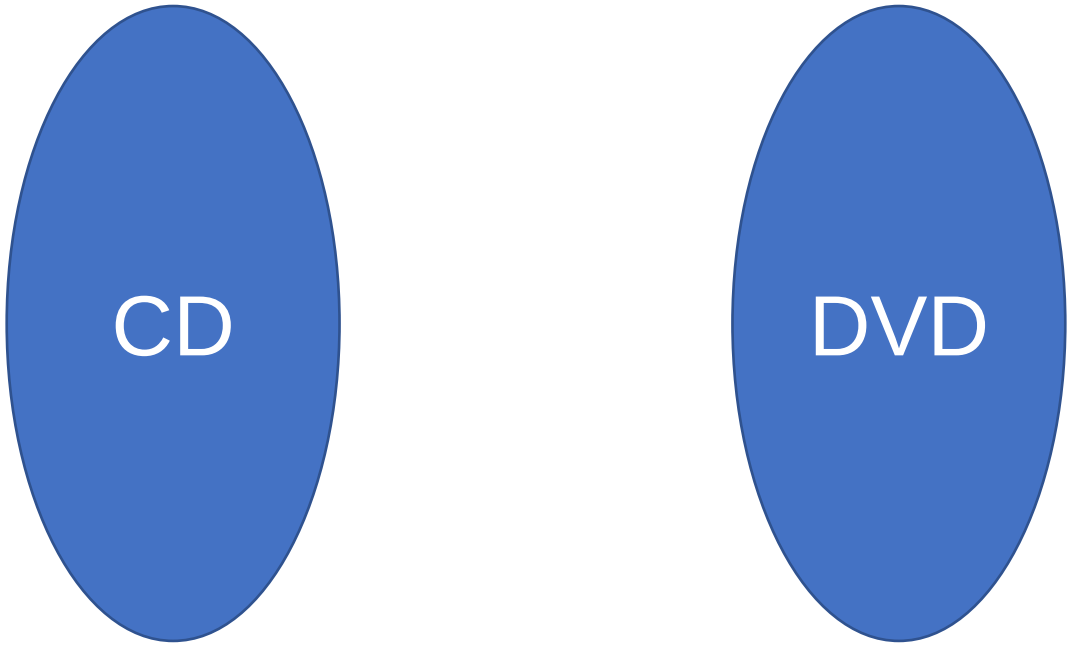
- dedičnosť – vzťah „gen-spec“

Generalizácia

- spoločný predok z konkrétnych tried

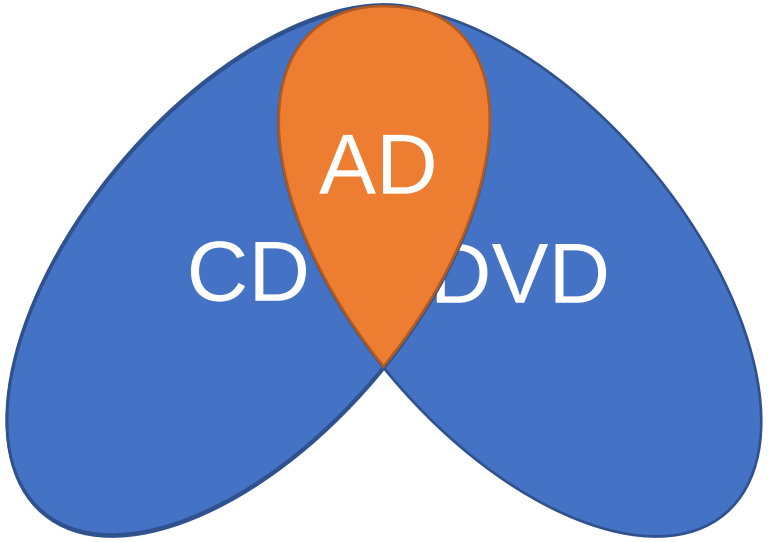
- KCaIB
  - trieda CD
  - trieda DVD
  - z nich trieda AudiovizualneDielo

Samostatné triedy CD a DVD



Vytvorenie spoločnej časti AudiovizualneDielo

generalizácia: (CD, DVD) → AudiovizualneDielo



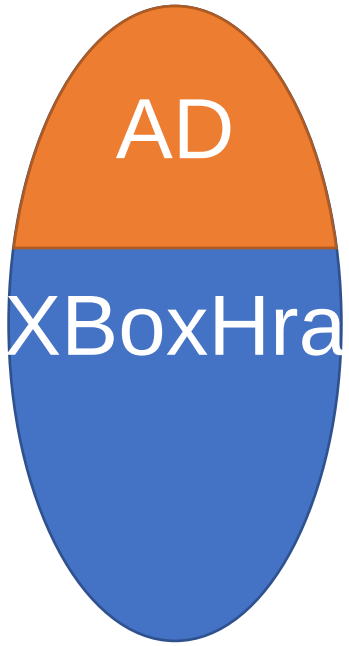
Špecializácia

- odvodenie potomka z predka

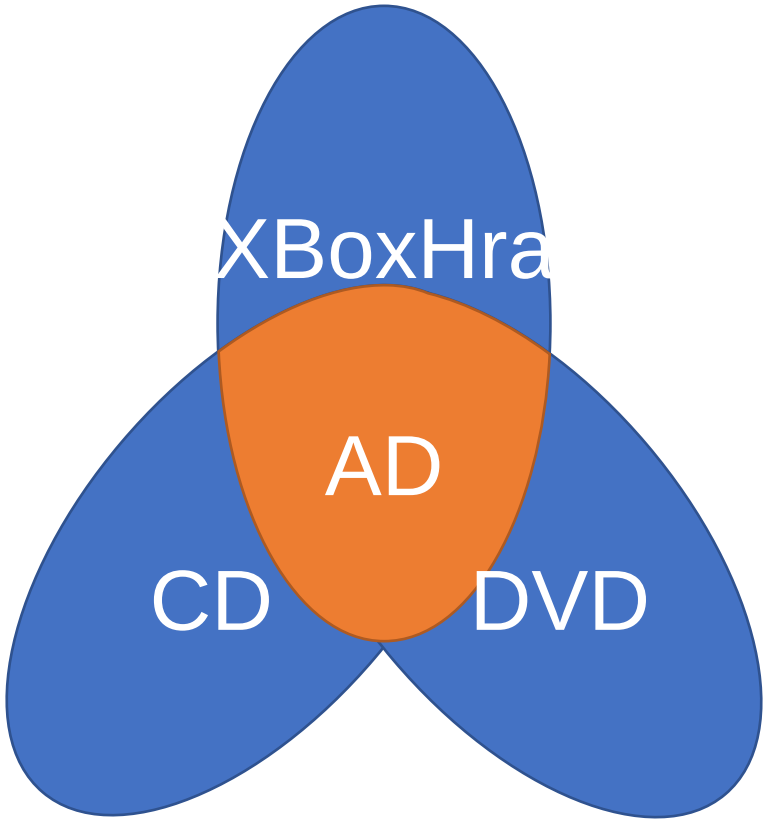
- KCaIB
  - trieda AudiovizualneDielo
  - z nej trieda XBoxHra

Nová trieda XBoxHra

špecializácia: AudiovizualneDielo → XBoxHra

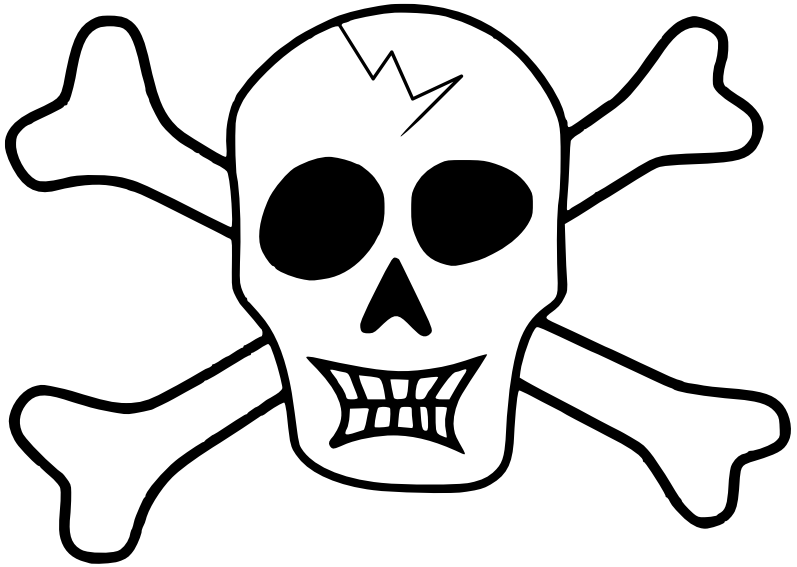


Nová trieda BlueRay

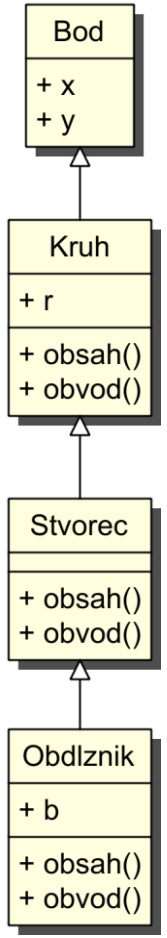


Nebezpečenstvá dedičnosti

- explózia tried
- nelogické použitie (zneužitie) dedičnosti
- porušenie Liskovej princípu substitúcie
- zvyšovanie implementačnej závislosti



„Technická“ dedičnosť

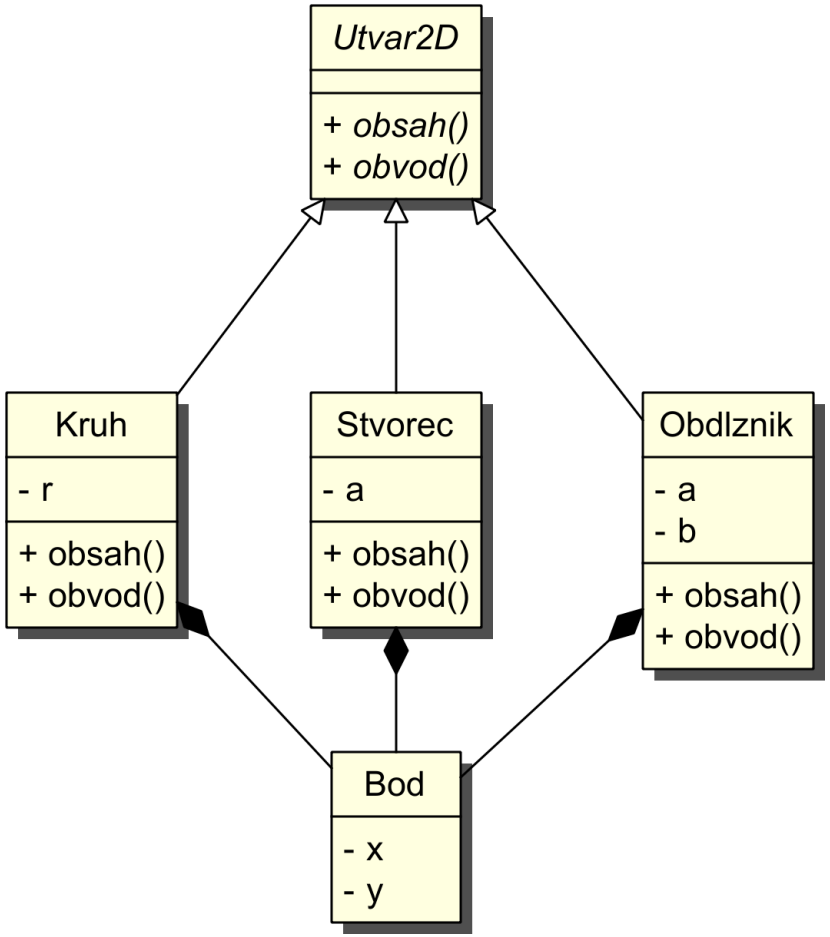


$O = 2\pi r$
$S = \pi r^2$

$O = 4r$
$S = r^2$

$O = 2r + 2b$
$S = rb$

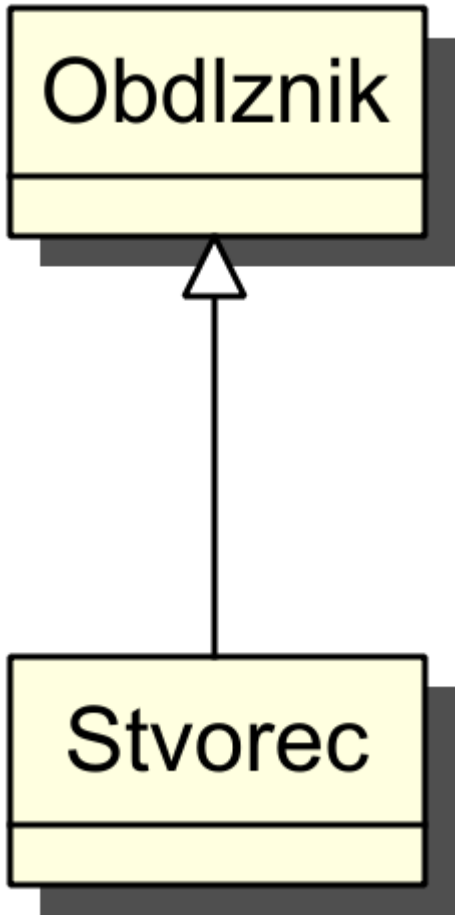
„Logická“ dedičnosť



Liskovej princíp substitúcie

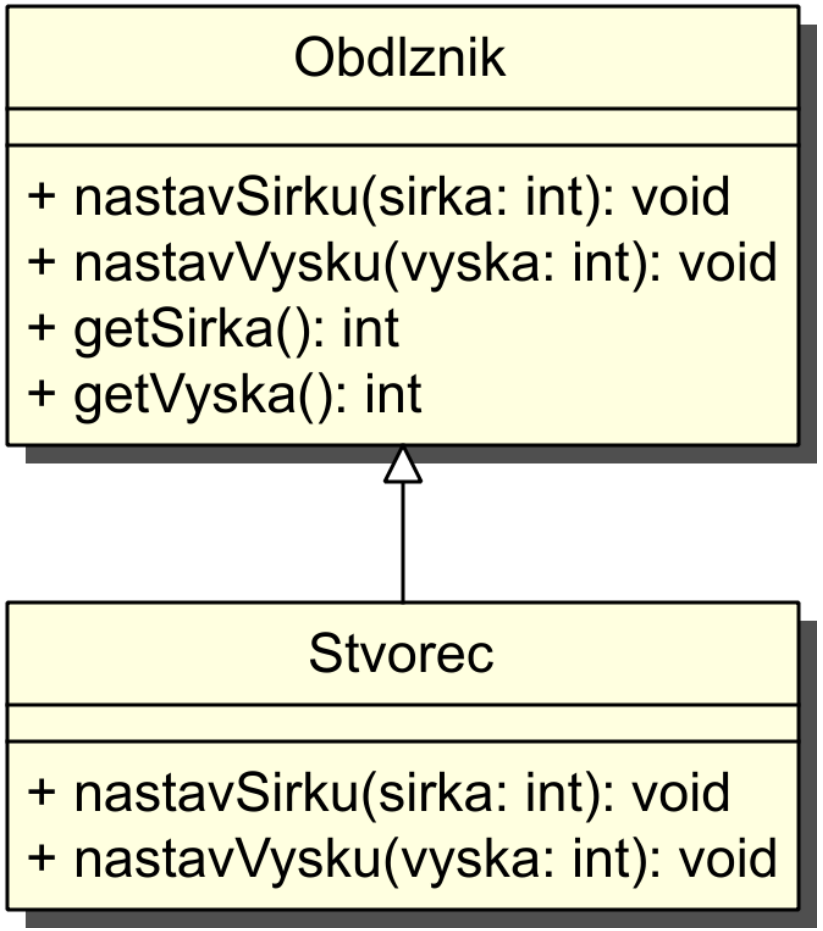
- ak existuje vlastnosť predka, ktorú nemôže potomok splniť = porušený substitučný princíp.
- ľubovoľná referencia na inštanciu predka by mala byť nahraditeľná referenciou na inštanciu potomka bez ovplyvnenia funkcionality.
- vlastnosti – uvedené v dokumentácii

Dedičnosť: Obdlznik – Stvorec (1)



- štvorec – špeciálny prípad obdĺžnika
- obe strany rovnaké

Dedičnosť: Obdlznik – Stvorec (2)



Metóda nastavSirku v triede Stvorec

```
@Override
public void nastavSirku(int sirka) {
    super.nastavVysku(sirka);
    super.nastavSirku(sirka);
}

@Override
public void nastavVysku(int vyska) {
    super.nastavVysku(vyska);
    super.nastavSirku(vyska);
}
```

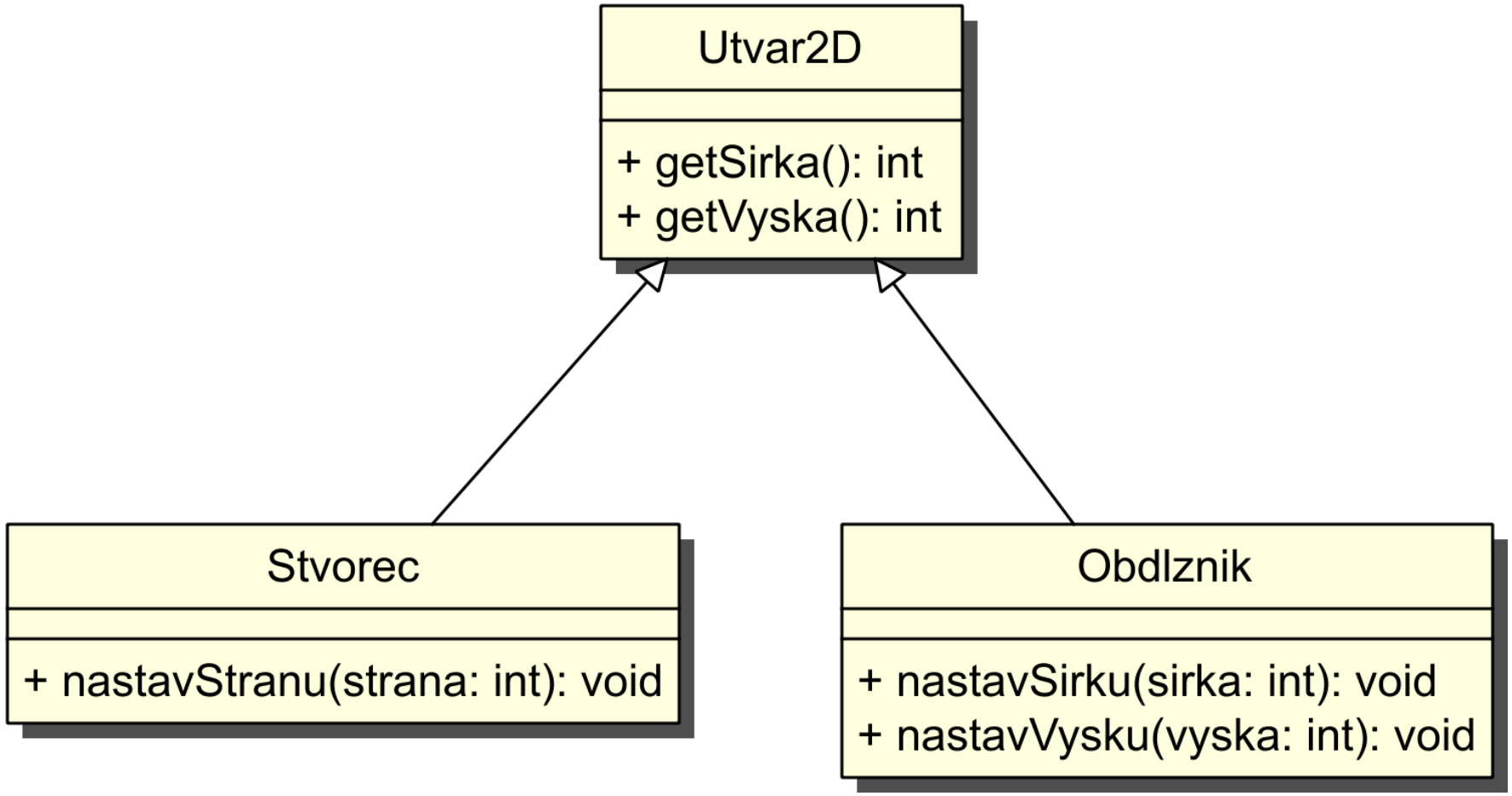
Dedičnosť: Obdlznik – Stvorec (1)

- obdĺžnik a štvorec sú rôzne útvary
- niektoré vlastnosti rovnaké
- niektoré vlastnosti rôzne

- rovnaké vlastnosti – abstraktný predok Utvar2D
- špecifické vlastnosti – konkrétni potomkovia

- polymorfizmus – abstraktné metódy predka implementujú potomkovia svojím spôsobom

Dedičnosť: Obdlznik – Stvorec (2)



Riziká porušenia LSP

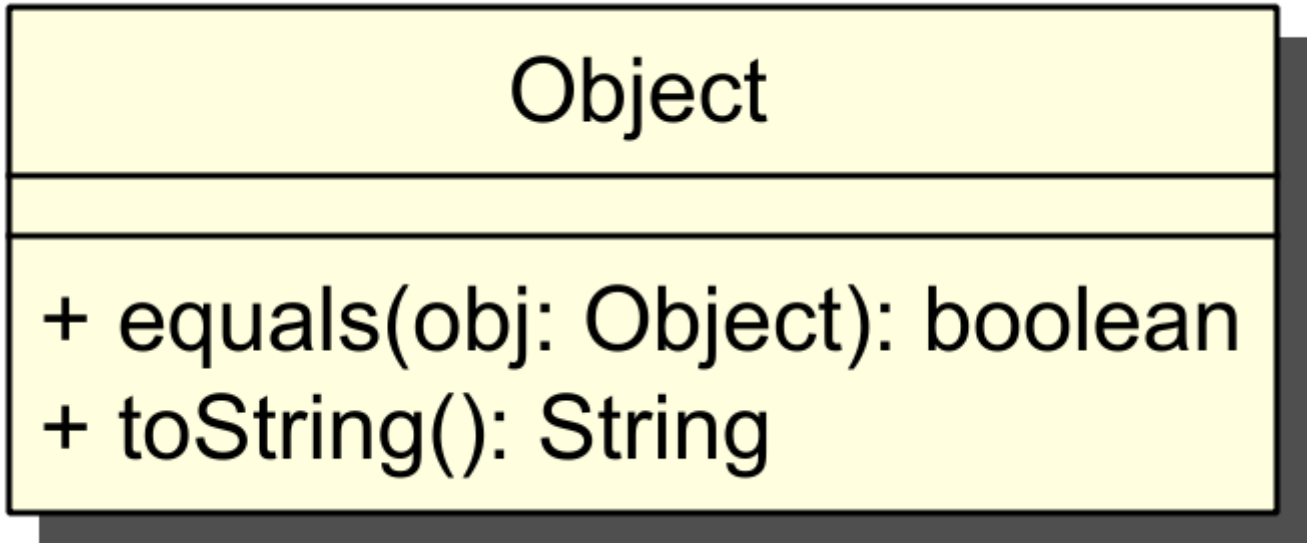
- predok a potomok sú konkrétne triedy
- potomok prekrýva metódy predka

Znižovanie rizika porušenia LSP

- implementácia abstraktných metód
- zvážiť vzťah dedičnosti konkrétnych tried
- zvážiť prekrývanie konkrétnej metódy predka
- zvážiť využívanie metódy predka pomocou super v prekrývajúcej metóde

- zvážiť = konzultovať dokumentáciu predka
  - čítať aj medzi riadkami ☺

Java – trieda Object



Trieda Object a LSP

- nie je definovaná ako abstraktná
- má zmysel vytvárať inštancie?
- väčšinou používame ako abstraktnú

- porušenie LSP?

Object – toString

- textová reprezentácia objektu
- štandardne
  - názov Triedy objektu (dynamický typ)
  - znak@
  - adresa objektu v pamäti
- System.out.println(objekt)
- reťazcový operátor +
  - objekt nie je reťazec – automatické použitie toString

Trieda Object a LSP – toString

- vlastnosť objektu je definovaná dokumentáciou.

- „Vytvára textovú reprezentáciu inštancie.“
  - dokumentácia štandardnej knižnice Java
- LSP neporušujeme pri takomto chápaní významu metódy toString.

Trieda Object a LSP – equals

- vlastnosť objektu je definovaná dokumentáciou.

- „Indikuje, či je iný objekt zhodný s týmto“
  - dokumentácia štandardnej knižnice Java
- dokumentácia definuje už spomenuté podmienky

- zachovanie podmienok v dokumentácii = neporušovanie LSP

Object – equals (1)

- relácia ekvivalencie pre dva objekty (referencie)
- štandardne
  - ekvivalentná s operátorom ==
- podmienky relácie:
  - x, y, z: rôzne od null
  - reflexívna: x.equals(x) = true
  - symetrická: x.equals(y) ↔ y.equals(x)
  - tranzitívna: x.equals(y) and y.equals(z) ↔ x.equals(z)
  - prístupová metóda, nemení stav porovnávaných objektov
  - x.equals(null) = false

Object – equals (2)

- == rovnosť identity dvoch referencii
  - implikuje rovnosť stavov
- equals – rovnosť stavu dvoch objektov
- pri prekrytí musia byť dodržané podmienky

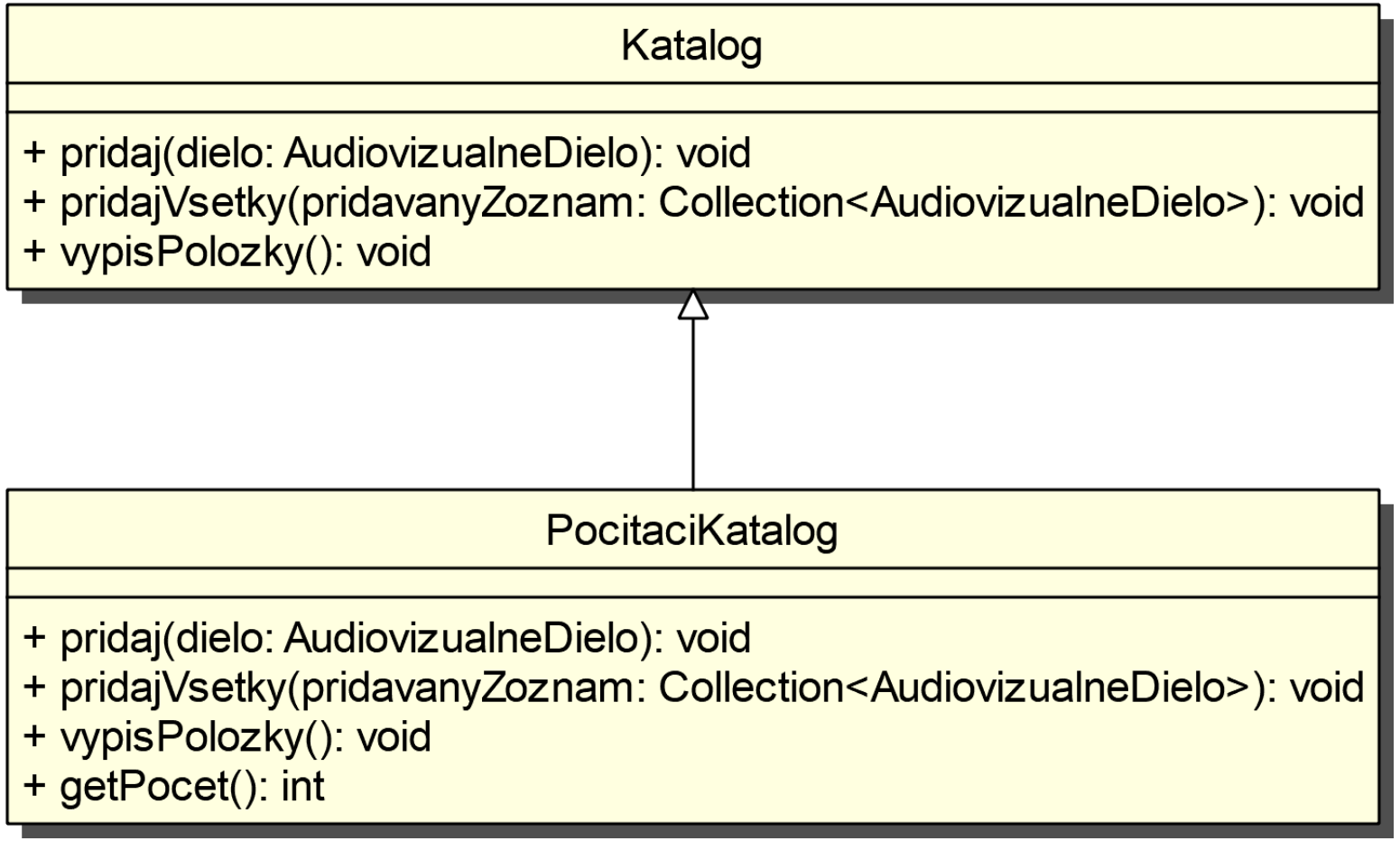
Dedičnosť – zvyšovanie závislosti

- nutnosť poznania implementácie predka

=>

- porušenie zapúzdrenia
- zvyšovanie implementačnej závislosti medzi triedami

Počítací katalóg



Príklad závislosti – predok (1)

```
public class Katalog {
    private ArrayList<AudiovizualneDielo> zoznam;

    public Katalog() {
        this.zoznam = new ArrayList<AudiovizualneDielo>();
    }

    ...
}
```

Príklad závislosti – predok (2)

```
public void pridaj(AudiovizualneDielo dielo) {
    this.zoznam.add(dielo);
}

public void pridajVsetky(Collection<AudiovizualneDielo> pridavanyZoznam){
    for (var dielo : pridavanyZoznam) {
        this.pridaj(dielo);
    }
}
```

Príklad závislosti – potomok (1)

```
public class PocitaciKatalog extends Katalog {
    private int pocet;

    public PocitaciKatalog() {
        this.pocet = 0;
    }

    ...
}
```

Príklad závislosti – potomok (2)

```
public void pridaj(AudiovizualneDielo dielo) {
    super.pridaj(dielo);
    this.pocet++;
}

public void pridajVsetky(Collection<AudiovizualneDielo> pridavanyZoznam){
    super.pridajVsetky(zoznam);
    this.pocet += pridavanyZoznam.size();
}
```

Funguje správne

```
PocitaciKatalog katalog = new PocitaciKatalog();
katalog.pridaj(new CD(...));
katalog.pridaj(new CD(...));
katalog.pridaj(new DVD(...));

System.out.println(katalog.getPocet());
// vypise „3“
```

Alebo aj nie

```
PocitaciKatalog katalog = new PocitaciKatalog();

var diela = new ArrayList<AudiovizualneDielo>();
diela.pridaj(new CD(...));
diela.pridaj(new CD(...));
diela.pridaj(new DVD(...));

katalog.pridajVsetky(diela);

System.out.println(katalog.getPocet());
// vypise „6“
```

V čom je problém?

- správne počíta, ak používame správu pridaj()
- ak použijeme pridajVsetky(), ráta zle
  - zaráta dvojnásobok

- kde je chyba?

Problém

```
public void pridaj(AudiovizualneDielo dielo) {
    super.pridaj(dielo);
    this.pocet++;
}

public void pridajVsetky(Collection<AudiovizualneDielo> pridavanyZoznam){
    super.pridajVsetky(zoznam);
    this.pocet += pridavanyZoznam.size();
}
```

Riešenia

- upraviť predka so znalosťou potomka
  - odstrániť polymorfizmus v pridajVsetky
    - this.pridaj(dielo) – problém
    - this.zoznam.add(dielo) – riešenie
- upraviť potomka so znalosťou predka
  - neprekryť metódu pridajVsetky
- odstrániť dedičnosť
  - skladanie

Polymorfizmus a dedičnosť (1)

- polymorfizmus – definuje chovanie objektov
- vychádza zo základného princípu – posielania správ
- nezávisí od dedičnosti

- „čistý“ princíp – len chovanie

Polymorfizmus a dedičnosť (2)

- dedičnosť – definícia hierarchie typov
- definuje štruktúru objektov
- napĺňa princíp reuse
- poskytuje implementačný komfort
- zahŕňa aj polymorfizmus
  - prekrývanie metód
  - implementácia abstraktných metód
- „zmiešaný“ princíp – štruktúra + chovanie

Názov „dedičnosť“

- dedičnosť dovoľuje zaviesť hierarchiu typov
- dedičnosť – nesprávny názov
  - neznamená dedenie génov
  - neznamená dedenie majetku
  - odvádza pozornosť od podstaty – hierarchia typov – k implementačným detailom – čo trieda dedí
- vzťah is-a, generalizácia/špecializácia
- UML – pojem Generalizácia