Fakulta riadenia a informatiky ŽU

Dedičnosť II

Pojmy zavedené v 6. prednáške(1)

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

Pojmy zavedené v 6. prednáške(2)

- interné ukrývanie informácií
- vzťahy pri dedičnosti
  - predok
  - potomok
  - priamy predok
  - priamy potomok
  - absolútny predok

Pojmy zavedené v 6. prednáške(3)

- typy dedičnosti
- graf dedičnosti
  - jednoduchá - strom dedičnosti
  - jednoducha - les dedičnosti
  - viacnásobná - mriežky dedičnosti

Pojmy zavedené v 6. prednáške(4)

- porovnanie skladania a dedičnosti
  - znovupoužiteľnosť
  - implementačná závislosť
- dedičnosť a interface

Pojmy zavedené v 6. prednáške(5)

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

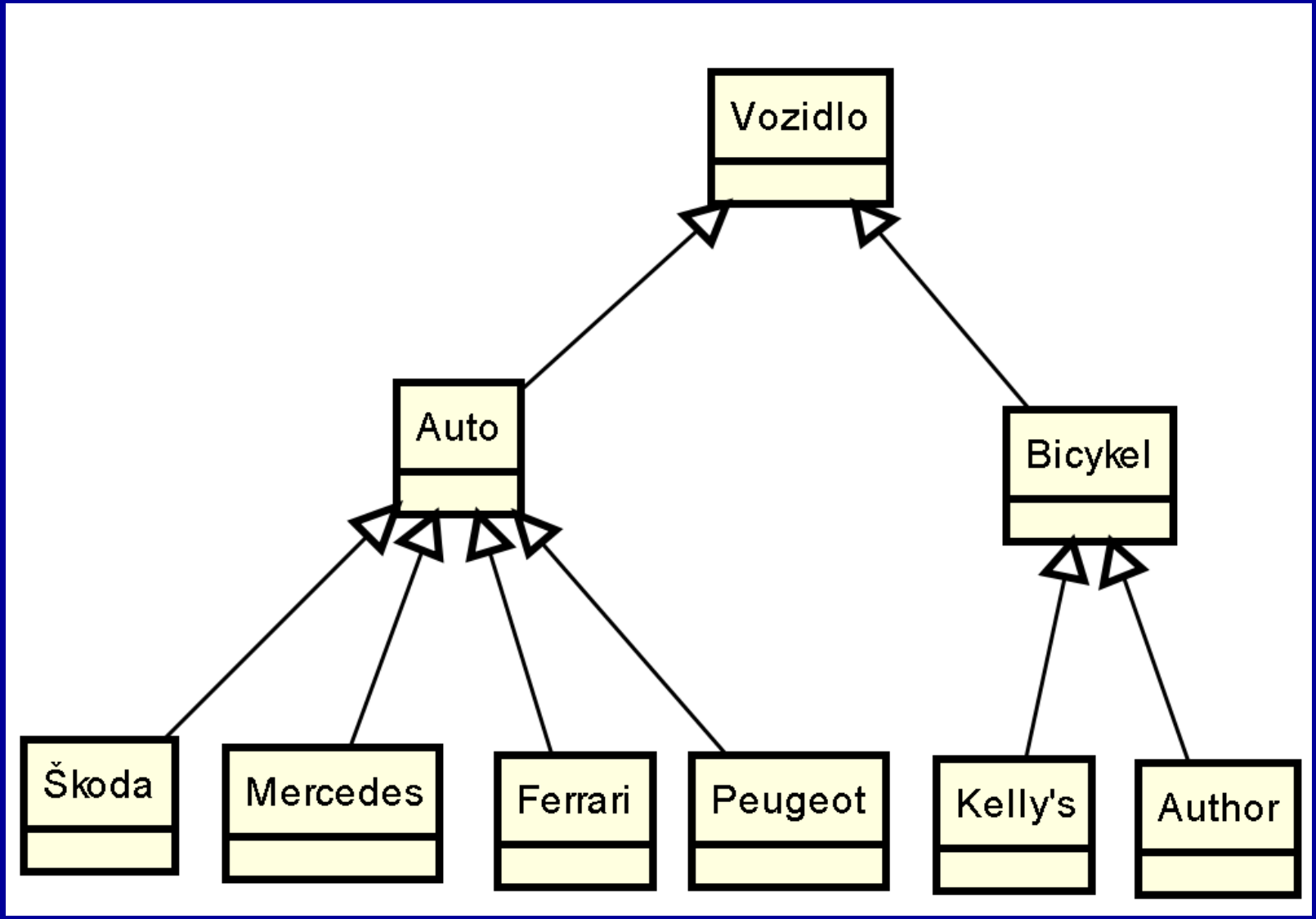
- jazyky - implemetácia polymorfizmu

Cieľ

- abstraktná trieda
- vzťahy is-a, has-a
- extenzia triedy
- Liskovej princíp substitúcie
- trieda Object

- projekt: KCaIB

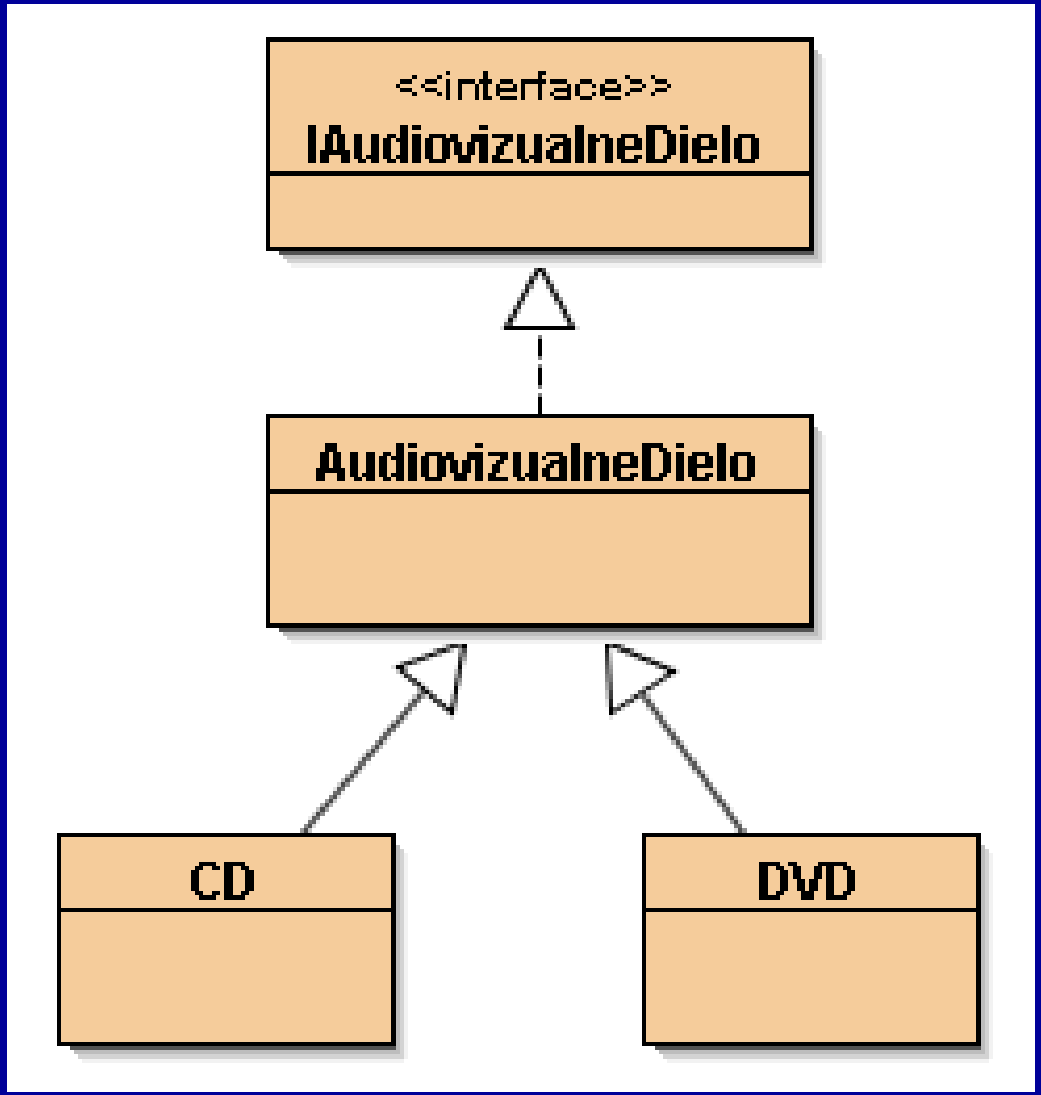
Abstraktné triedy v reálnom svete(1)



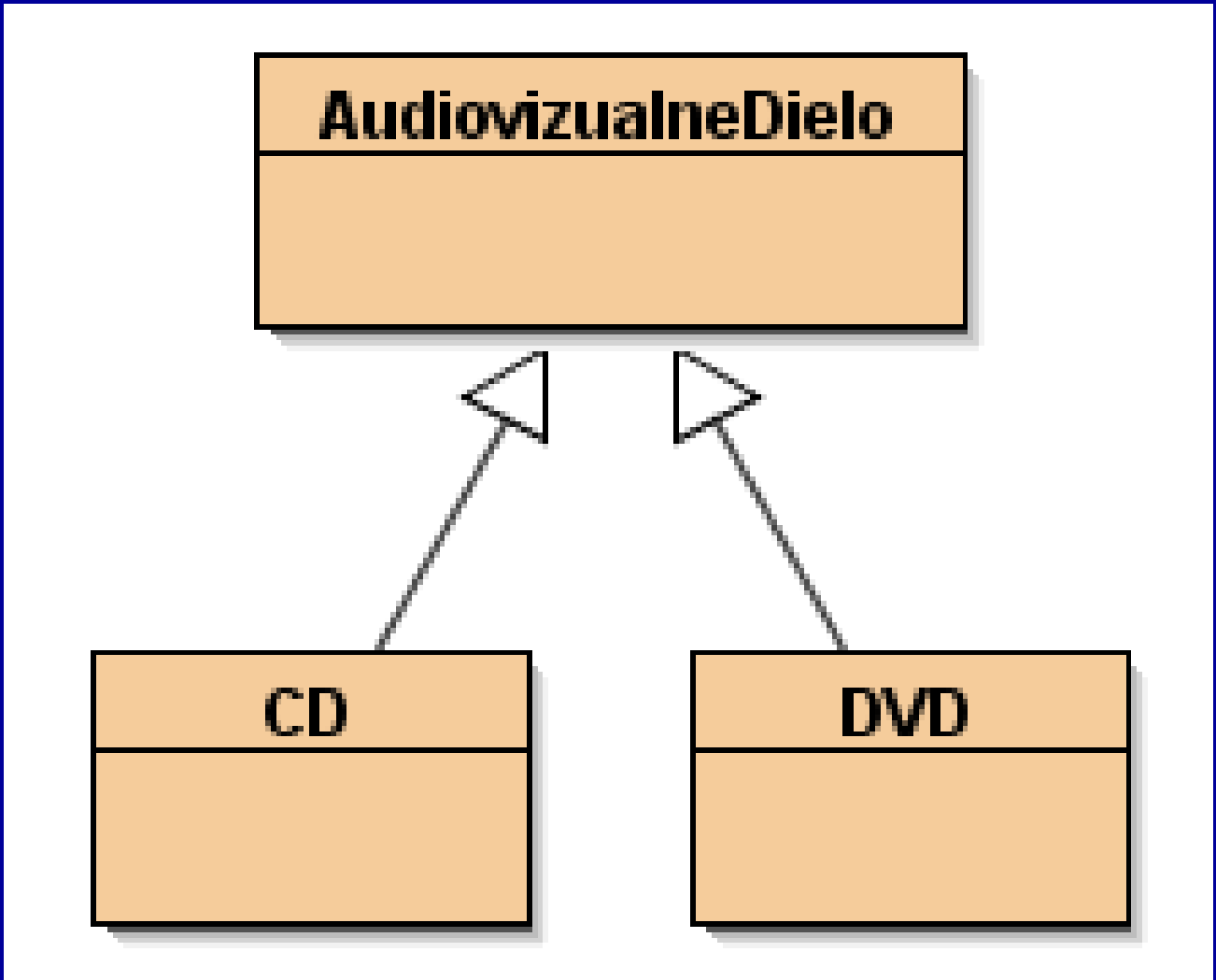
Abstraktné triedy v reálnom svete(2)

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

Projekt DCaIB(1)



Projekt DCaIB(2)



Abstraktná trieda i motivácia

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

Konkrétna trieda

- Inštancie vytvorené triedami CD a DVD
  - majú význam
  - existujú aj v skutočnosti
- Konkrétna trieda
  - bežne vytvára inštancie

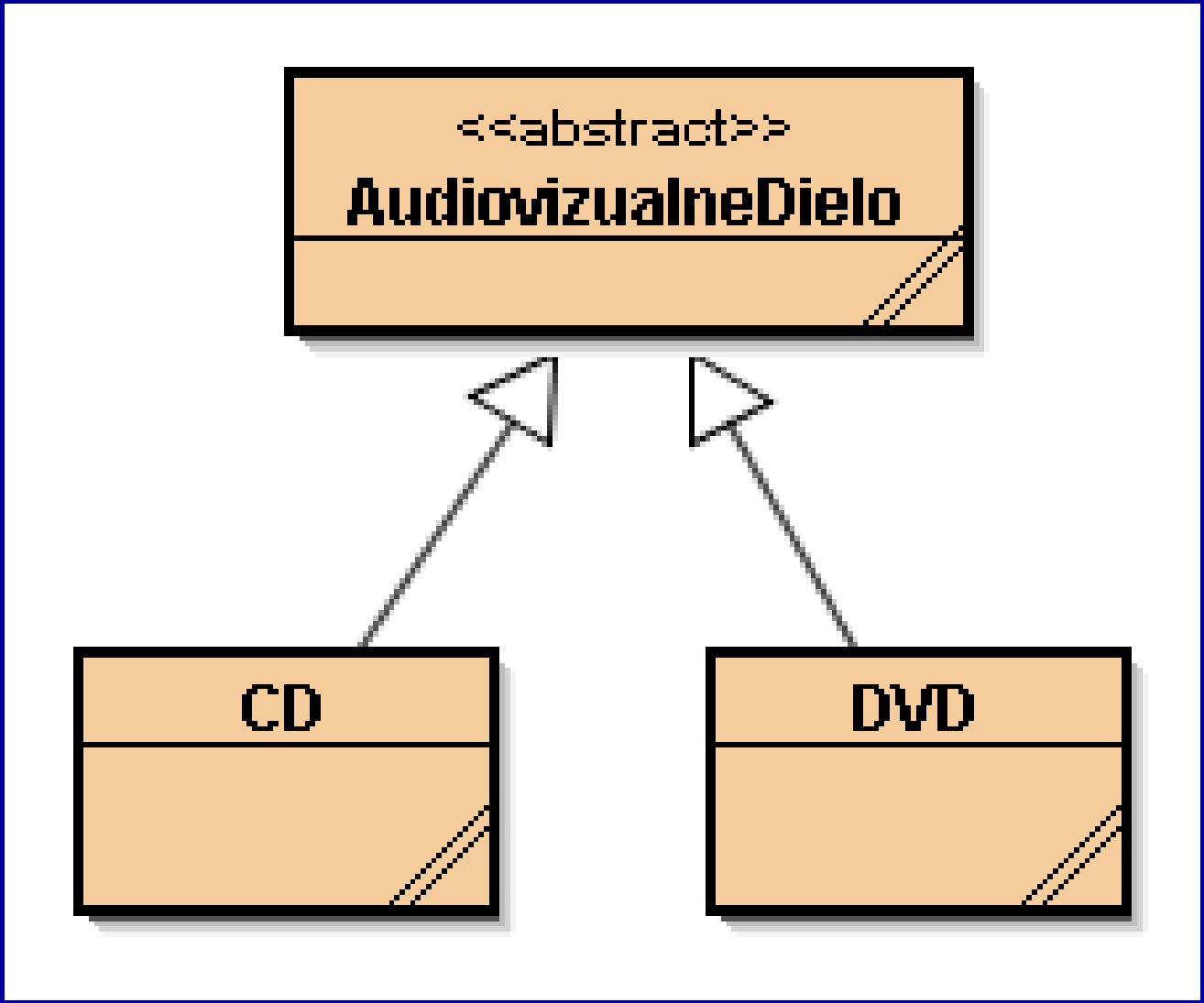
Abstraktná trieda – vlastnosť

- charakteristická vlastnosť – nevytvára inštancie
  - dobrovoľná – možno vytvoriť, nemá význam
  - povinná – nemožno vytvoriť, zákaz

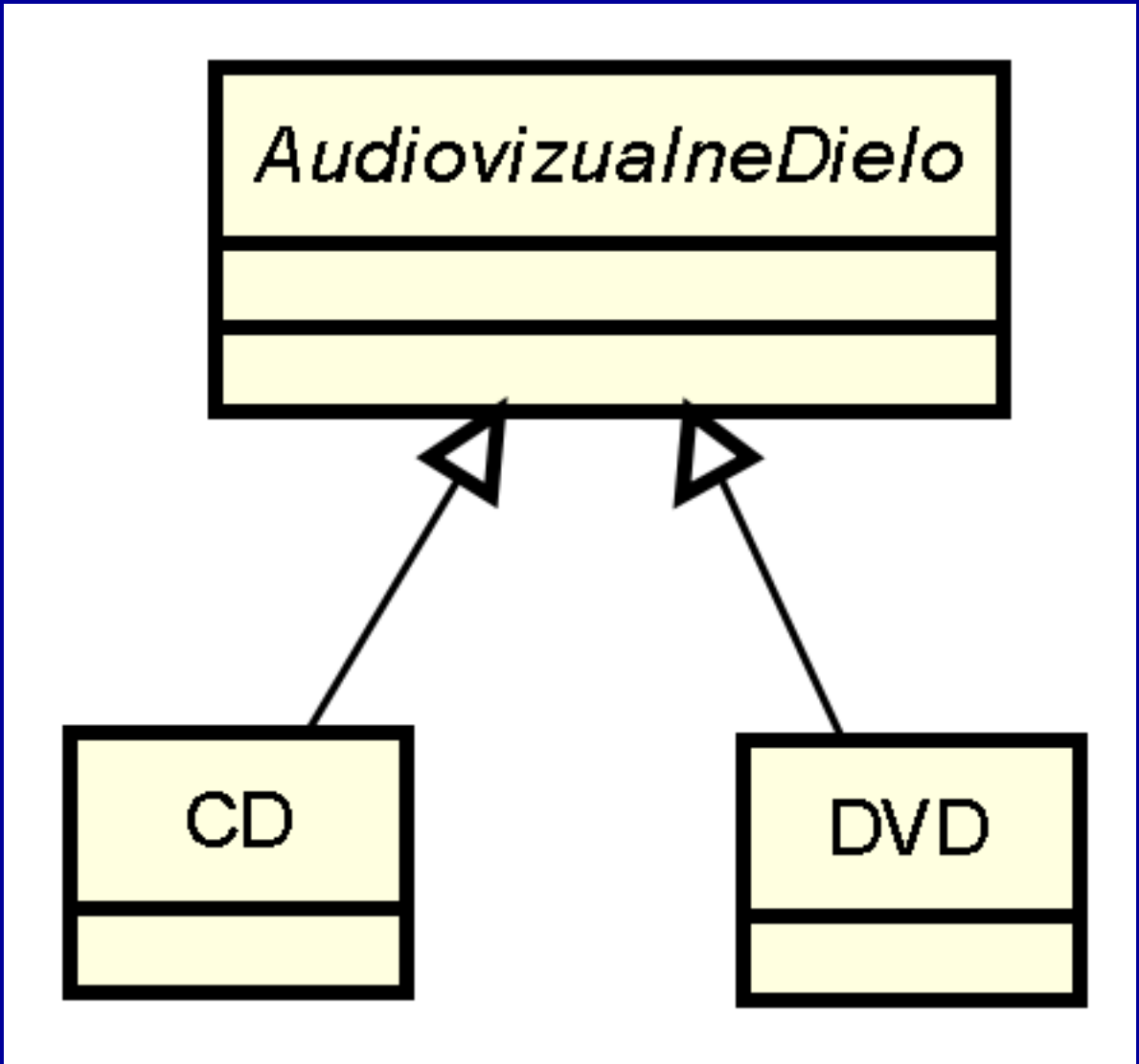
Abstraktná trieda – modelovanie

- BlueJ
  - stereotyp <<abstract>>
  - nad menom triedy
- UML
  - meno triedy – *kurzíva*

Abstraktná trieda – BlueJ



Abstraktná trieda – UML



Abstraktná trieda – Java

- kľúčové slovo abstract
- hlavička triedy

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

DCaIB – metódy vypis(1)

- AVIDielo – definícia metódy
- CD, DVD – prekrytie metódy
- polymorfizmus
- konečná podoba metódy – potomok

DCaIB – metódy vypis(2)

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

Abstraktná a konkrétna metóda

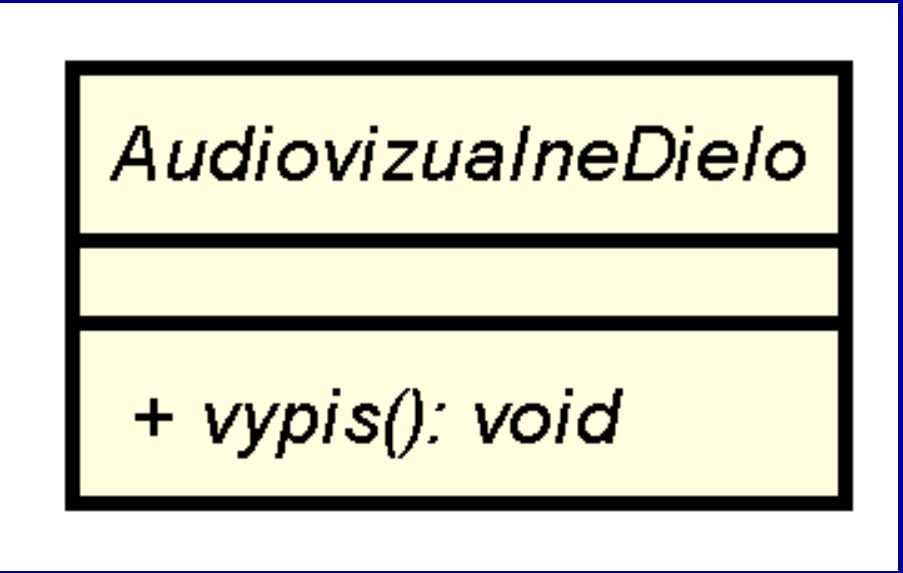
- metódy v abstraktnej triede
  - konkrétne
    - definícia tela metódy v tele triedy
    - definícia správy v rozhraní triedy
  - abstraktné
    - definícia správy v rozhraní triedy
- metódy v konkrétnej triede
  - len konkrétne

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

- meno metódy *kurzívou*



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();
```

Abstraktná metóda – Java

- abstraktná metóda – trieda musí byť abstraktná

- abstraktná trieda
  - nechceme vytvárať inštancie – voliteľná
  - definuje abstraktnú metódu
  - dedí abstraktnú metódu

AVIDielo – 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

CD – metóda vypis

```
public class CD
{
    ...
    public void vypis()
    {
        // príkazy tela metódy
    }
    ...
}
```

CD – telo metódy vypis

```
System.out.println("CD:");
System.out.println("    Autor: " + aAutor);
System.out.println("    Titul: " + this.dajTitul());
System.out.println();
System.out.println("    Pocet skladieb: " +
                        aPocetSkladieb +
                       " (celkovo " +
                        this.dajCelkovyCas() + " 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(1)

- extenzia triedy
  - množina všetkých inštancií
  - vzťah „is-a“ => aj inštancií potomkov
- dôsledok
  - abstraktné triedy môžu mať neprázdnu extenziu

Dedičnosť a extenzia triedy(2)

- podľa filozofie už z čias Aristotela
- trieda má triádu: *meno*, *intenziu* a *extenziu*
- *meno* – označenie triedy, napr. Ryba
- *intenzia* – čo to je ryba, aké ma ryba chovanie (implementácia triedy Ryba; sú rôzne triedy rýb)
- *extenzia* – všetky inštancie, ktoré sú rybami, majú chovanie podľa intenzie

- zdroj. doc. Vojtěch Merunka

Operátor instanceof – extenzia

```
prvyOperand instanceof DruhyOperand
```

- vracia true
  - prvyOperand je inštanciou triedy DruhyOperand
- rozšírenie
- vracia true
  - prvyOperand patrí do extenzie triedy DruhyOperand

Vzťah has-a

- vyjadrenie vzťahu celku a jeho častí
- celok má časti
- obdoba vzťahu is-a pre kompozíciu (skladanie)

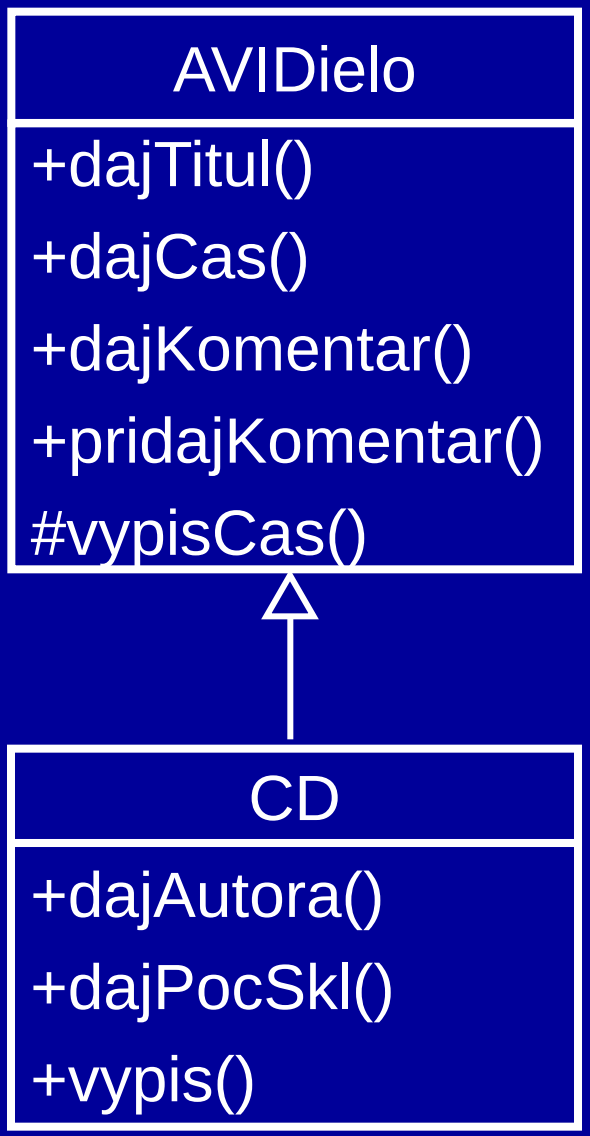
- DigitalneHodiny „má“ CiselnyDisplej

Prístupové právo protected

- selektívne prístupové práva
  - „protekcia“ pre potomkov
- public – verejný prístup všetkým iným objektom
- private – zákaz prístupu všetkým iným objektom
- doména – podstrom dedičnosti, daný koreň
- protected
  - public – každý objekt typu z domény
  - private – každý objekt typu mimo domény

Protected – UML

– protected



Dedičnosť a návrh tried

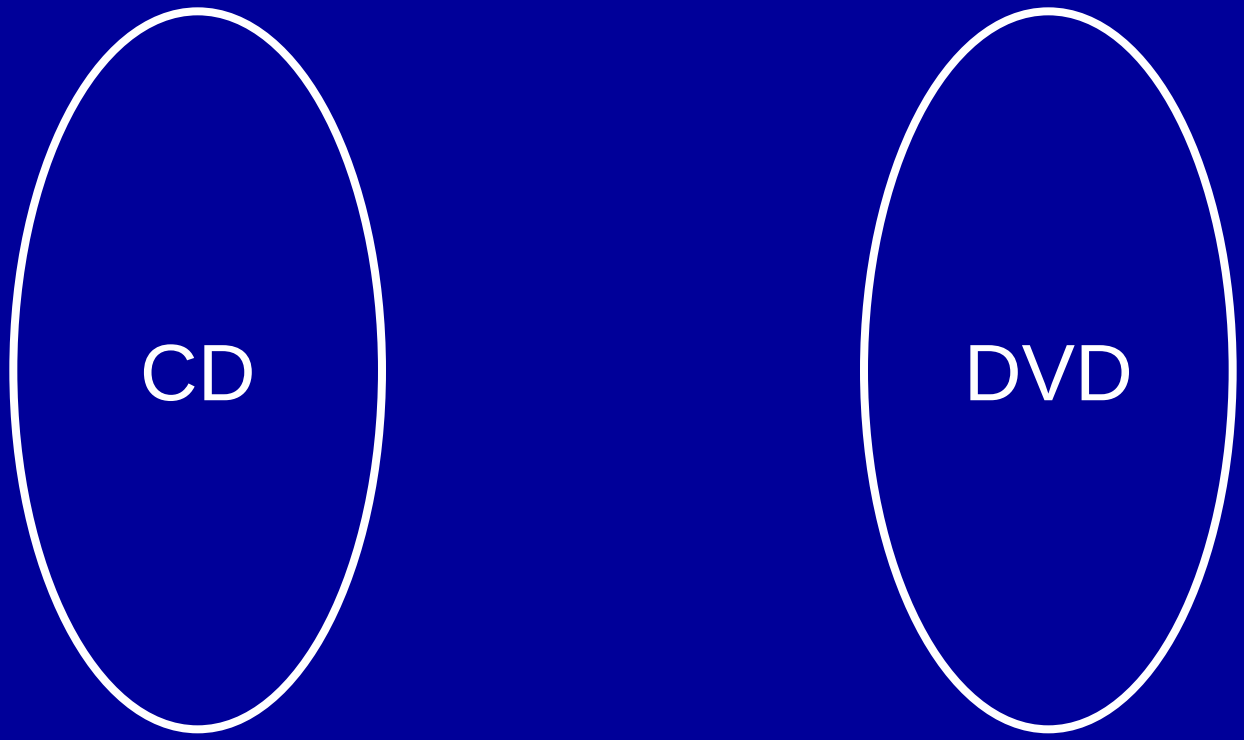
- generalizácia
- špecializácia

- dedičnosť – vzťah „genspec“

Generalizácia

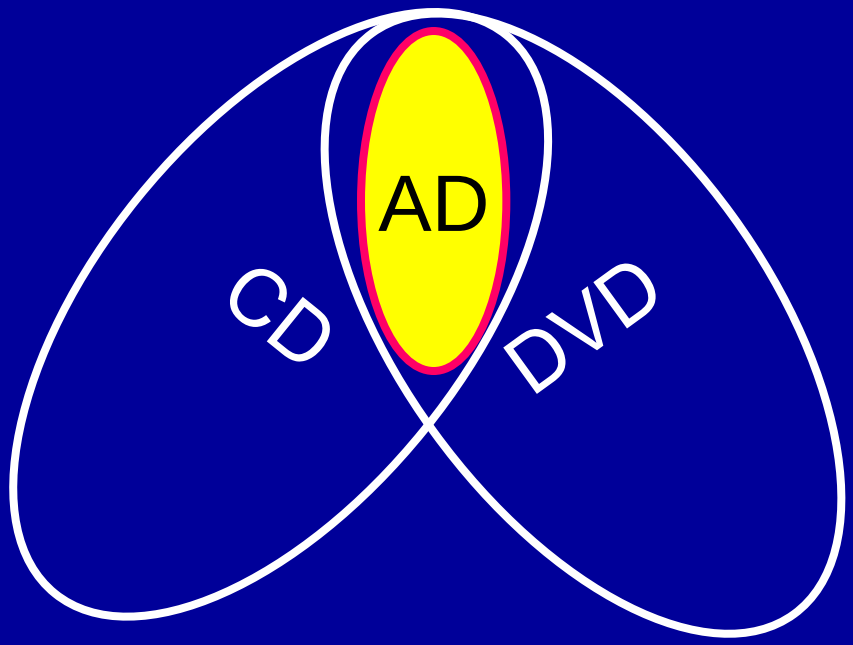
- KCalB
  - trieda CD
  - trieda DVD
  - z nich
  - trieda AudiovizualneDielo

Samostatné triedy CD a DVD



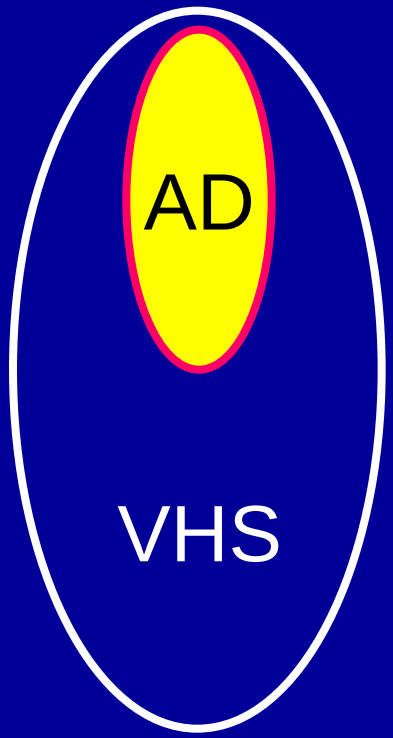
Vytvorenie spoločnej časti AD

generalizácia: (CD, DVD) → AD

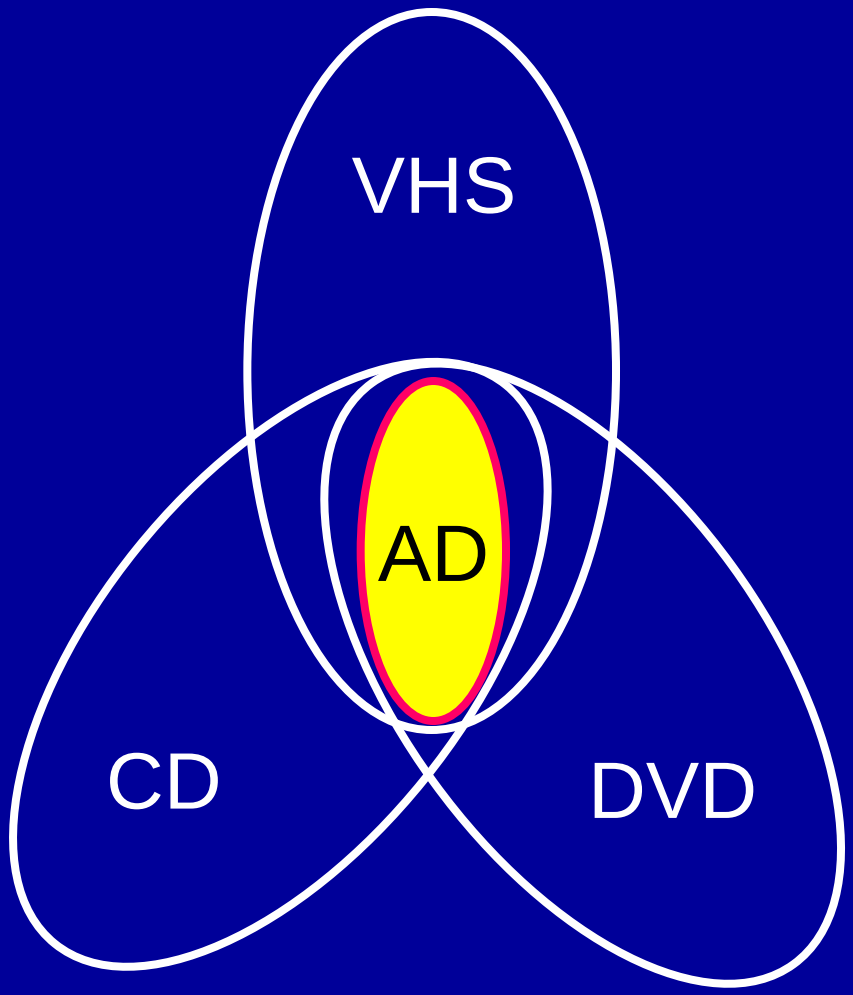


Nová trieda VHS

špecializácia: AD → VHS



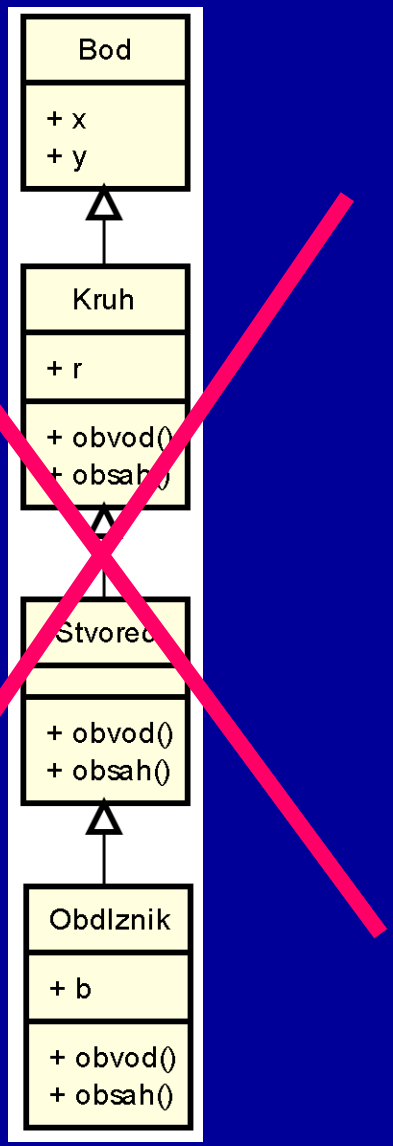
Nová trieda VHS



Nebezpečenstvá dedičnosti

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

„Technická“ dedičnosť

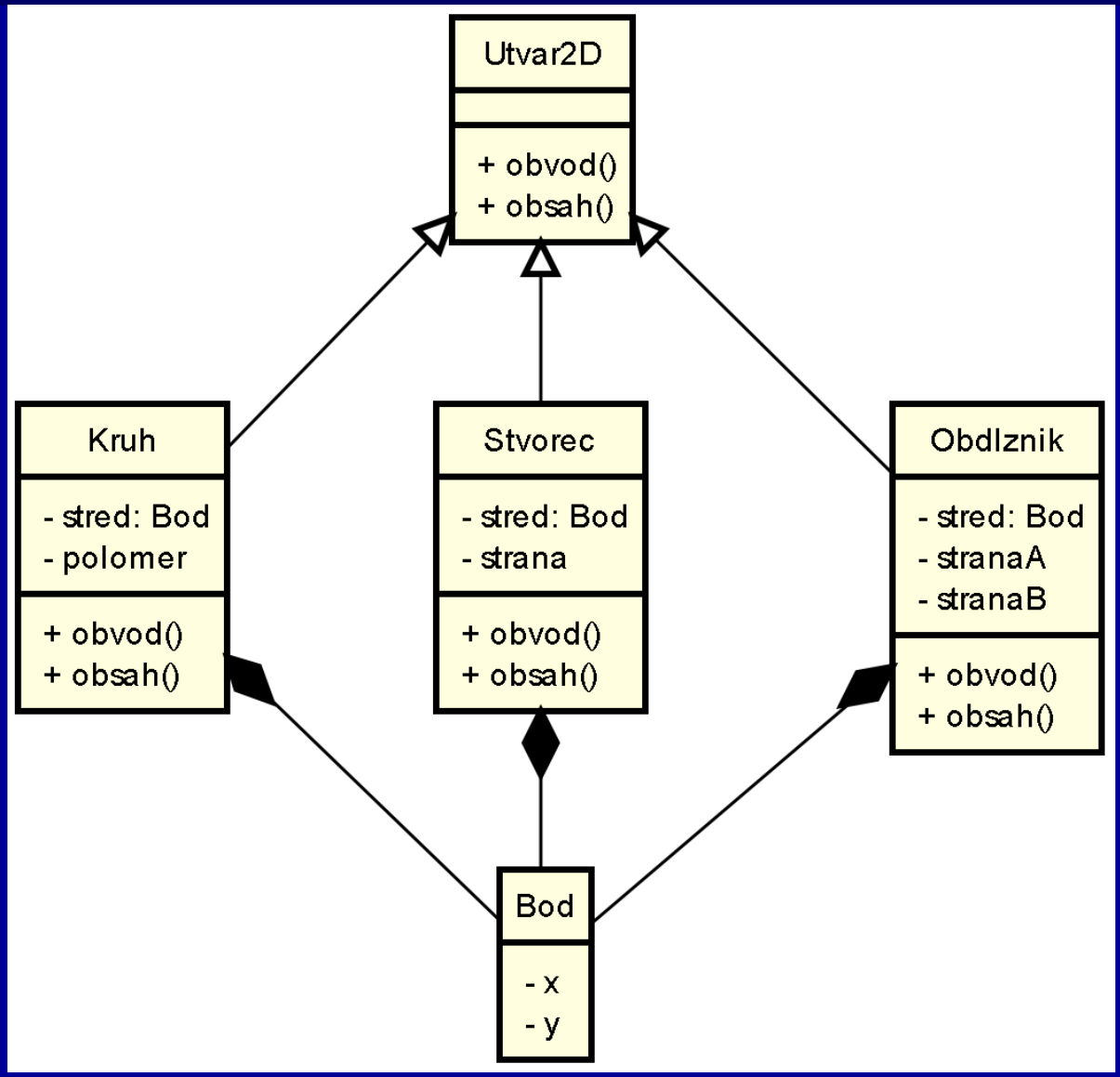


$$O = 2\pi r$$
$$S = \pi r^2$$

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

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

„Logická“ dedičnosť



Barbara Liskov(*1939)

- 1987 – princíp substitúcie pre typy
- 2004 – medaila Jon von Neumana
- 2008 – cena A. M. Turinga

The Liskov Substitution Principle (LSP)

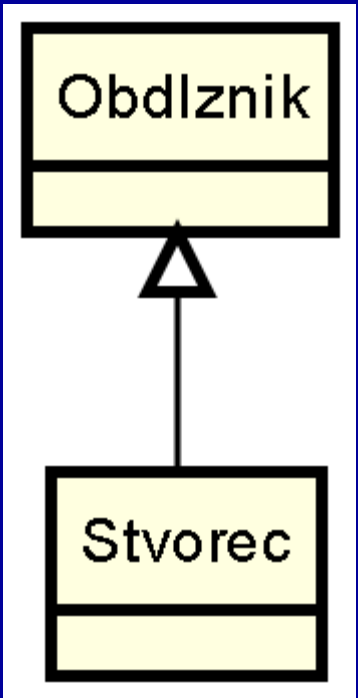
- *Let q(x) be a property provable about objects x of type T. Then q(y) should be true for objects y of type S where S is a subtype of T.*

- Nech *q*(*x*) je preukázateľná vlastnosť objektu *x* typu *T*. Potom *q*(*y*) by mala platiť pre objekt *y* typu *S*, kde *S* je podtyp typu *T*.

Liskovej princíp substitúcie

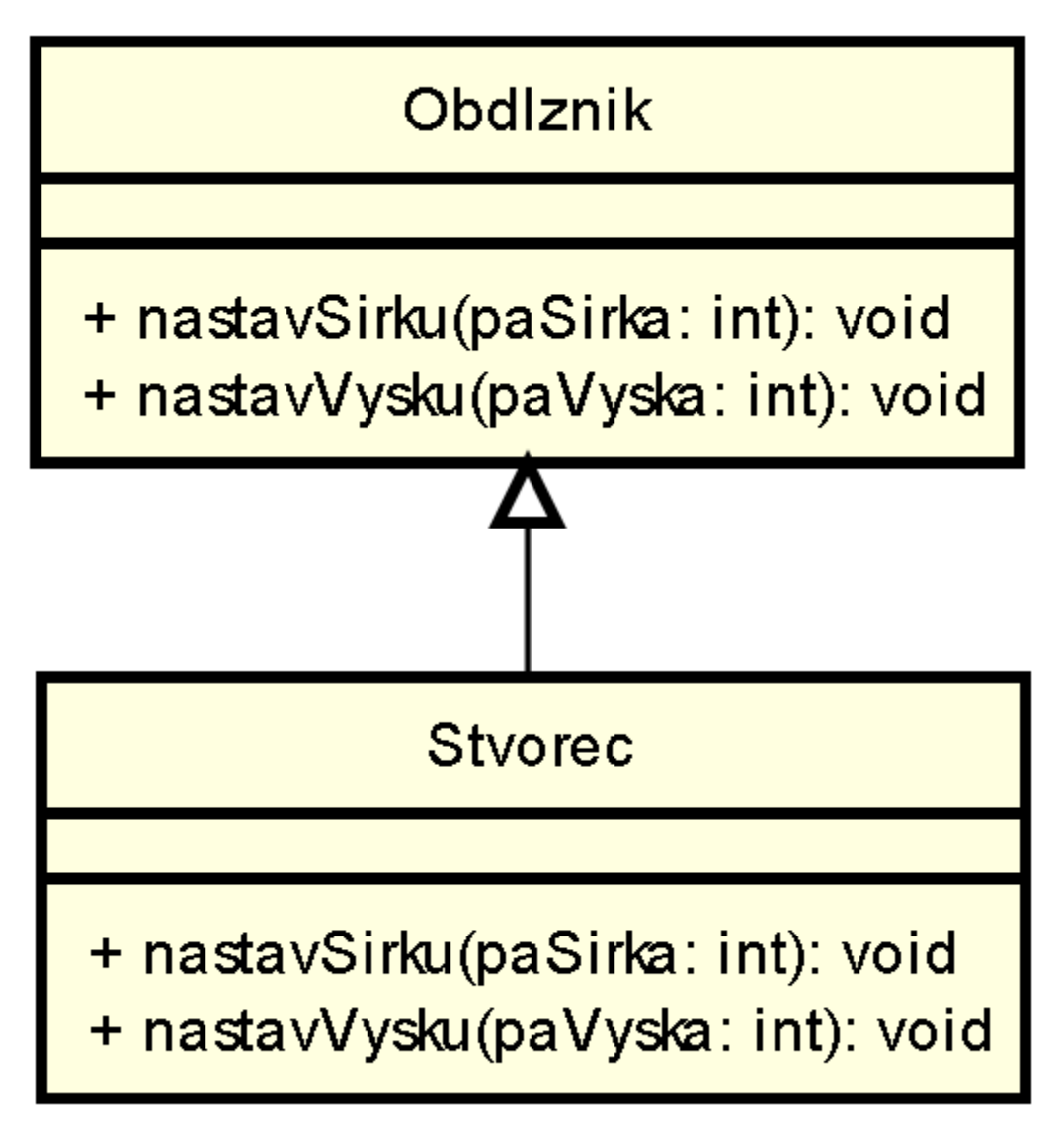
- Ak existuje vlastnosť predka, ktorú nemôže potomok splniť = porušený substitučný princíp.
- Ľubovoľná referencia na inštanciu predka by mala byť nahraditeľná referenciou na inštanciu potomka bez ovplyvnenia funkcionality.
- vlastnosti – uvedené v dokumentácii

Dedičnosť: Obdlznik – Stvorec



- štvorec – špeciálny prípad obdĺžnika
- obe strany rovnaké

Dedičnosť: Obdlznik – Stvorec



metóda nastavSirku

```
public void nastavSirku(int paSirka)
{
    super.nastavVysku(paSirka);
    super.nastavSirku(paSirka);
}
```

Test objektov typu Obdlznik

```
private void pomocObsah(Obdlznik paObdlznik)
{
    paObdlznik.nastavSirku(5);
    paObdlznik.nastavVysku(4);
    assertEquals(
        20,
        paObdlznik.dajSirku() * paObdlznik.dajVysku()
    );
}
```

Test objektov typu Obdlznik

```
public void testObsahTriedaObdlznik()
{
    this.pomocObsah(new Obdlznik());
}

public void testObsahTriedaStvorec()
{
    this.pomocObsah(new Stvorec());
}
```

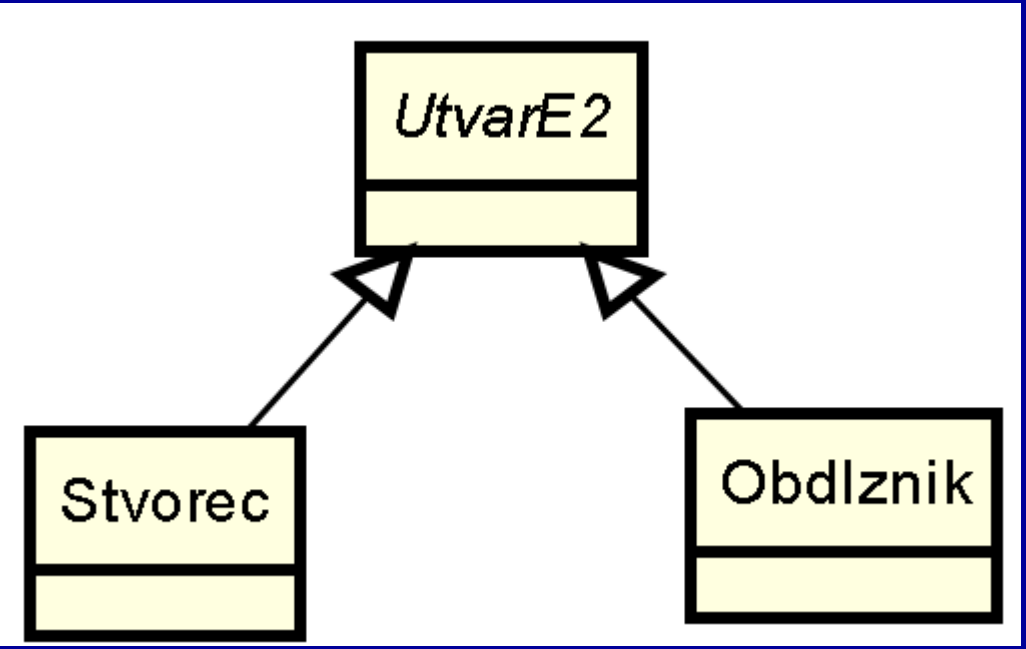
Dedičnosť: Obdlznik – Stvorec(1)

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

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

- polymorfizmus – abstraktné metódy predka implementujú potomkovia svojím spôsobom

Dedičnosť: Obdlznik – Stvorec(2)



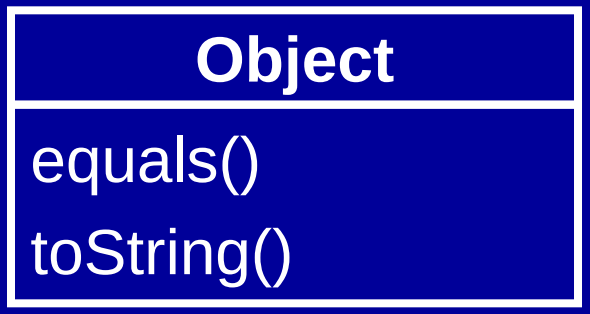
Riziká porušenia LSP

- 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

Java – trieda Object



- boolean equals(Object paObject)
- String toString()

Object – toString()

- textová reprezentácia objektu
- String toString()

- názov Triedy objektu (dynamický typ)
- znak@
- adresa objektu v pamäti

- System.out.println(objekt)
- reťazcový operátor +
- objekt nie je reťazec – automatické použitie toString

Object – equals(Object paObjekt)

- relácia ekvivalencie pre dva objekty (referencie)
- trieda Object: 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(Object paObjekt)

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

Trieda Object a LSP(1)

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

- porušenie LSP?

Trieda Object a LSP(2)

- zdedená toString od Object – má zmysel pre potomka?
- „Vytvára textovú reprezentáciu inštancie.“
  - dokumentácia Java
- vlastnosť inštancie je definovaná dokumentáciou.
- LSP neporušujeme pri takomto chápaní významu metódy toString.

Dedičnosť – zvyšovanie závislosti

- nutnosť poznania implementácie predka

  =>

- porušenie zapúzdrenia
- zvyšovanie implementačnej závislosti medzi triedami

Príklad závislosti(1)

```
public class Katalog
{
    private List<AVIDielo> aDiela;

    public Katalog()
    {
        aDiela = new List<AVIDielo>();
    }
    ...
}
```

Príklad závislosti(2)

```
public pridaj(AVIDielo paDielo)
{
    aDiela.add(paDielo);
}
public pridajZoznam(List<AVIDielo> paZoznamDiel)
{
    for (AVIDielo dielo : paZoznamDiel) {
        this.pridaj(dielo);
    }
}
```

Potomok s počítaním

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

    public PocitaciKatalog()
    {
        aPocet = 0;
    }
    ...
}
```

Príklad závislosti(2)

```
public pridaj(AVIDielo paDielo)
{
    aPocet++;
    super.pridaj(paDielo);
}
public pridajZoznam(List<AVIDielo> paZoznamDiel)
{
    aPocet += paZoznamDiel.count();
    super.pridajZoznam(paZoznamDiel);
}
```

V čom je problém?

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

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
- môže zahŕňať aj polymorfizmus
  - prekrývanie metód
  - implementácia abstraktných metód

- „zmiešaný“ princíp – štruktúra + chovanie

Názov „dedičnosť“(1)

- dedičnosť dovoľuje zaviesť hierarchiu typov
- dedičnosť – nesprávny názov
  - neznamená dedenie génov
  - neznamená dedenie majetku

Názov „dedičnosť“(2)

- vzťah is-a, generalizácia/špecializácia
- dokumentácia UML – prevažuje generalizácia
- generalizácia používajú aj nástroje

- odvádza pozornosť od podstaty – hierarchia typov – k implementačným detailom – čo trieda dedí

Vytváranie inštancií tried

- štyri spôsoby
  - klasická trieda
  - singleton
  - enum
  - abstraktná trieda

Klasická trieda

- správa new – verejné rozhranie triedy
- verejný konštruktor
- ľubovoľný počet inštancií
- extenzia – všetky inštancie triedy a jej potomkov

Singleton

- správa new – neverejné rozhranie triedy
- neverejný konštruktor
- žiadosť o inštanciu – správa dajInstanciu() – verejné rozhranie
- najviac jedna inštancia
- extenzia – najviac jedna inštancia

(Objektový) enum

- bez správy new
- konštruktor bez definovaných prístupových práv
- počet inštancií je pevne daný
- priamo definovaný zoznam inštancií
- extenzia – pevne daná množina inštancií

Abstraktná trieda

- bez správy new
- verejný konštruktor
- ľubovoľný počet inštancií
- extenzia – všetky inštancie potomkov

Zabránenie prekrytia metódy

- ak sa nesmie prekrývať konkrétna metóda
- zabránenie polymorfizmu danej metódy
- nevzťahuje sa na preťaženie
- kľúčové slovo final v hlavičke metódy

```
public final int dajCelkovyCas()
{
    return aCelkovyCas;
}
```

Koncová trieda

- odvodená trieda nemá zmysel
- vždy konkrétna

- všetky metódy automaticky final
- kľúčové slovo final v hlavičke triedy

```
public final class CD
{
    ...
}
```

Zhrnutie(1)

- Abstraktná trieda
  - v reálnom svete
  - v jazyku Java
  - v BlueJ/UML
- Konkrétna trieda
- Abstraktná metóda
  - v reálnom svete
  - v jazyku Java

Zhrnutie(2)

- Vzťah is-a
- Dedičnosť a extenzia triedy
- Triáda triedy
  - meno
  - intenzia
  - extenzia
- instanceof v hierarchii dedičnosti

Zhrnutie(3)

- viac o LSP
- riešenie problému štvorec-obdĺžnik pomocou abstraktnej triedy
- riziká porušenia LSP
  - znižovanie rizika
- trieda Object a LSP

- polymorfizmus vs dedičnosť

Zhrnutie(4)

- vytváranie inštancií tried
  - klasická trieda
  - singleton
  - enum
  - abstraktná trieda
- modifikátor final
  - zabránenie prekrytia metódy
  - koncová trieda

Ďakujem za pozornosť