

10

# Generiká

# Pojmy zavedené v 9. prednáške<sub>(1)</sub>

- štandardný vstup a výstup
- textové súbory
- binárne súbory
- objektové prúdy

# Pojmy zavedené v 9. prednáške<sub>(2)</sub>

- štandardný vstup a výstup
- čítanie: triedy InputStream in a Scanner
- zapisovanie: trieda PrintStream out
- členenie na riadky

# Pojmy zavedené v 9. prednáške<sub>(3)</sub>

- súbory
- trieda File
- spolupráca so súborovým systémom
- práca so súborom
  - otvorenie – vytvorenie príslušnej inštancie
  - čítanie alebo zapisovanie
  - zatvorenie – správa close()

# Pojmy zavedené v 9. prednáške<sub>(4)</sub>

- textové súbory
  - čitateľné pre človeka
- čítanie: Scanner (+ Scanner)+ File
- zápisovanie: PrintWriter (+ File)
- formátovanie textového súboru
- bežne členenie na riadky

# Pojmy zavedené v 9. prednáške<sub>(5)</sub>

- binárne súbory
  - nečitateľné pre človeka
- čítanie: DataInputStream + FileInputStream (+ File)
- zapisovanie: DataOutputStream + FileOutputStream (+ File)
- String a primitívne dátové typy

# Pojmy zavedené v 9. prednáške<sub>(6)</sub>

- objektové prúdy
  - nečitateľné pre človeka
- čítanie: ObjectInputStream + FileInputStream (+ File)
- zapisovanie: ObjectOutputStream + FileOutputStream (+ File)
- celé objekty

# Ciel' prednášky

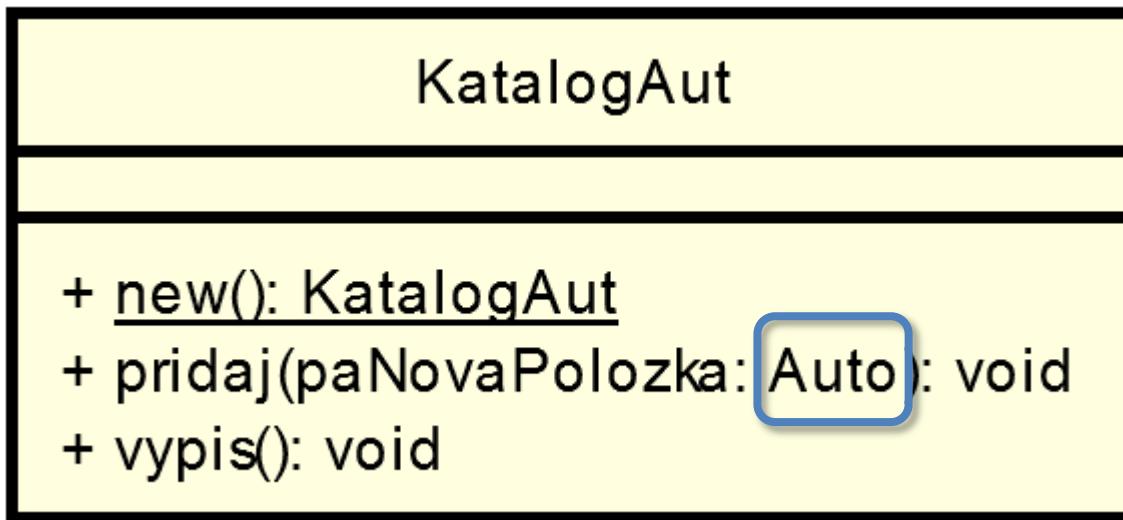
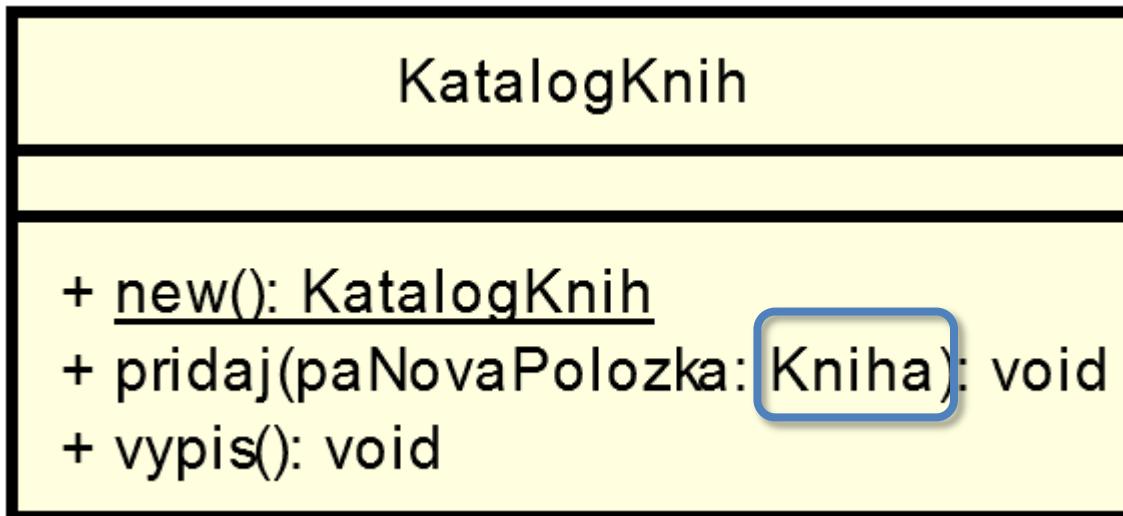
- generické triedy
  - generické metódy
- 
- príklad: všeobecný katalóg



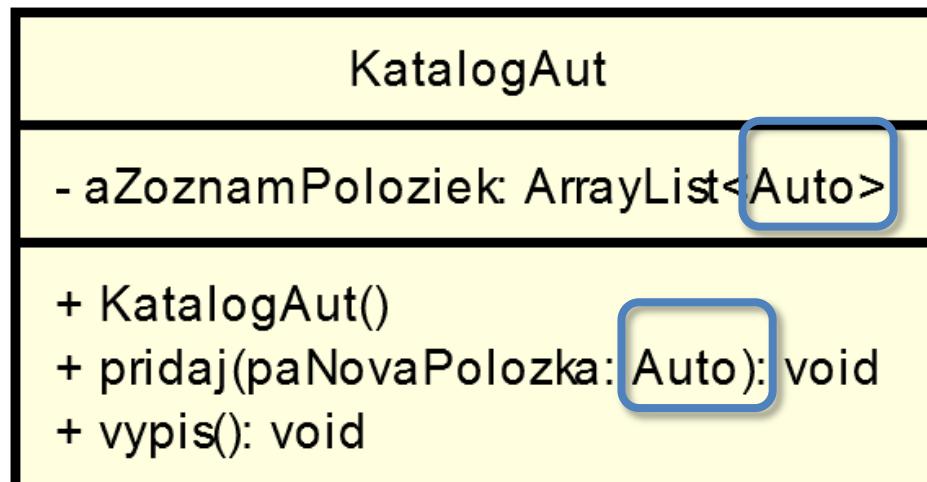
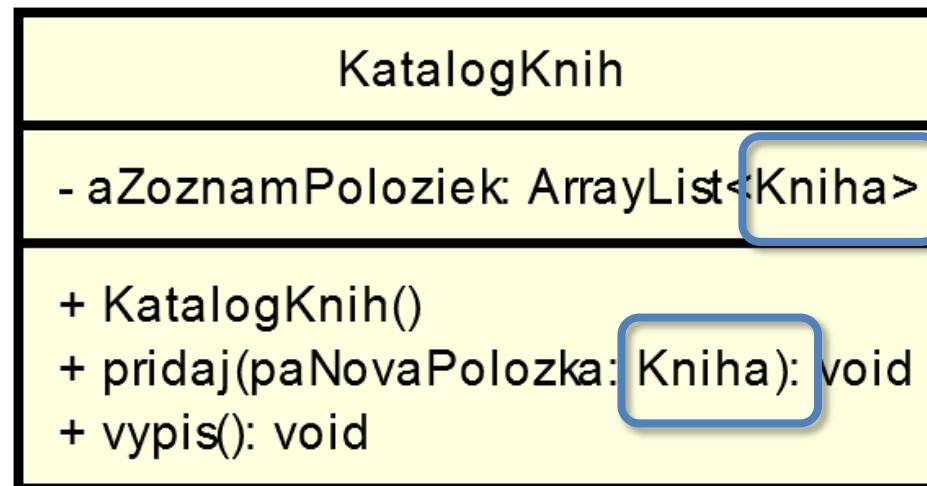
# Všeobecný katalóg – zadanie

- rozšírenie katalógu z KCaIB
- univerzálnejšie riešenie
  - katalóg kníh
  - katalóg automobilov

# Katalóg kníh vs. katalóg áut<sub>(1)</sub>



# Katalóg kníh vs. katalóg áut<sub>(2)</sub>



# Trieda KatalogAut

```
public KatalogAut() {  
    aZoznamPoloziek = new ArrayList<Auto>();  
}  
  
public void pridaj(Auto paNovaPolozka) {  
    aZoznamPoloziek.add(paNovaPolozka);  
}  
  
public void vypisPolozky() {  
    for (Auto polozka : aZoznamPoloziek) {  
        System.out.println(položka);  
        System.out.println("=====");  
    }  
}
```

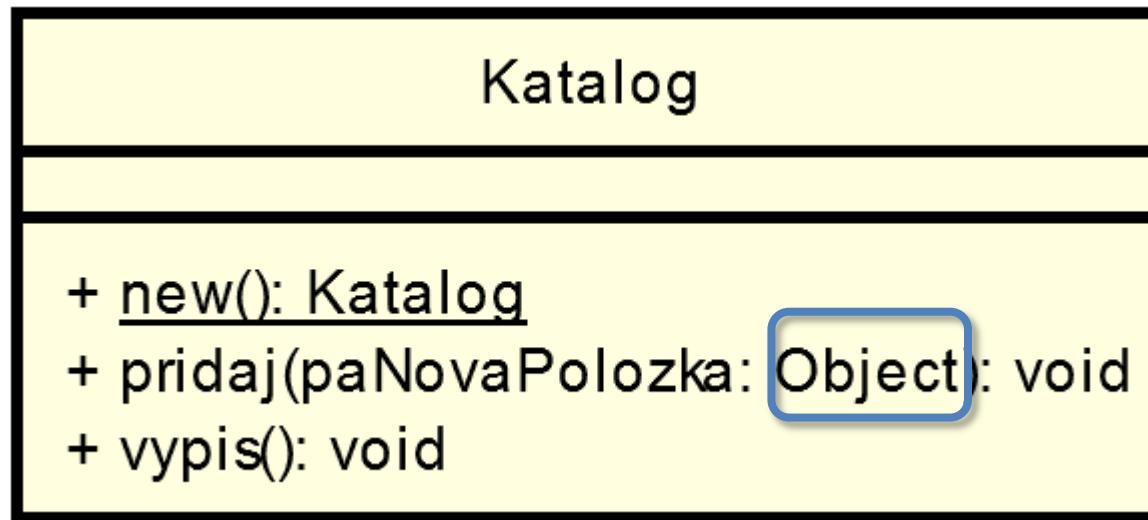
# Trieda KatalogKnih

```
public KatalogKnih() {  
    aZoznamPoloziek = new ArrayList<Kniha>();  
}  
  
public void pridaj(Kniha paNovaPolozka) {  
    aZoznamPoloziek.add(paNovaPolozka);  
}  
  
public void vypisPolozky() {  
    for (Kniha polozka : aZoznamPoloziek) {  
        System.out.println(položka);  
        System.out.println("=====");  
    }  
}
```

# Riešenie

- polymorfizmus
- čo použiť ako spoločný typ?
  - interface IPolozkaKatalogu
  - abstraktná trieda PolozkaKatalogu
  - trieda Object
- riešenie s Object vyhovuje
  - katalóg posielá položkám len správu `toString`

# Riešenie pomocou Object



# Použitie (katalóg kníh)

```
Katalog mojKatalog = new Katalog();  
mojKatalog.pridaj(  
    new Kniha("Bram Stocker", "Drakula"));  
mojKatalog.pridaj(  
    new Kniha("Sharon Zakhour", "Java 6"));  
...  
Kniha drakula = mojKatalog.dajNaPozicii(0);  
// chyba pri preklade  
  
Kniha drakula = (Kniha)mojKatalog.dajNaPozicii(0);
```

# Použitie (katalóg áut)

```
Katalog mojKatalog = new Katalog();
mojKatalog.pridaj(new Auto("Peugeot", "207"));
mojKatalog.pridaj(new Auto("Škoda", "Oktávia"));
...
Auto oktavia = (Auto)mojKatalog.dajNaPozicii(1);
```

# Problém (znovu katalóg kníh)

```
Katalog mojKatalog = new Katalog();
mojKatalog.pridaj(
    new Kniha("Bram Stocker", "Drakula"));
mojKatalog.pridaj(
    new Kniha("Sharon Zakhour", "Java 6"));

... // (o 100 riadkov ďalej a 3 týždne neskôr)
mojKatalog.pridaj(new Auto("Škoda", "Oktávia"));

...
Kniha drakula = (Kniha)mojKatalog.dajNaPozicii(2);
```

# Výsledok riešenia

- odstránili sme dublicity
- katalóg je príliš „univerzálny“
  - umožňuje vkladať ľubovoľné objekty
- katalóg kníh = len pre knihy
- katalóg áut = len pre autá
- katalóg audiovizuálnych diel = len pre diela

# Generické triedy

- riešením je generická trieda
- definícia typu položky pri definícii premennej
- definícia typu položky pri vytváraní inštancie

# Typový parameter

- syntax:

```
public class NazovTriedy<TypoveParametre>
```

- zoznam typových parametrov je čiarkami oddelený
- jedná sa o typy – konvencia – prvé veľké
- definuje „typ“ prístupný v celej triede

# Typový parameter – konvencie

- Java
  - E – element kontajnera
  - K – kľúč v Map
  - V – hodnota v Map
  - N – číslo
  - T – všeobecný typ
  - S, U, V... – ďalšie typy
- všeobecnejšia konvencia
  - prvé písmeno T, pokračuje popisný názov
  - TPolozka

# Trieda Katalog<sub>(1)</sub>

```
public class Katalog<TPolozka>
{
    private ArrayList<TPolozka> aZoznamPoloziek;
    ...
}
```

# Trieda Katalog<sub>(2)</sub>

```
public Katalog() {  
    aZoznamPoloziek = new ArrayList<TPolozka>();  
}  
  
public void pridaj(TPolozka paNovaPolozka) {  
    aZoznamPoloziek.add(paNovaPolozka);  
}  
  
public void vypisPolozky() {  
    for (TPolozka polozka : aZoznamPoloziek) {  
        System.out.println(polozka);  
        System.out.println("=====");  
    }  
}
```

# Použitie (katalóg kníh)

```
Katalog<Kniha> mojKatalog = new Katalog<Kniha>();  
mojKatalog.pridaj(  
    new Kniha("Bram Stocker", "Drakula"));  
mojKatalog.pridaj(  
    new Kniha("Sharon Zakhour", "Java 6"));  
...  
Kniha drakula = mojKatalog.dajNaPozicii(0);
```

# Použitie (katalóg áut)

```
Katalog<Auto> mojKatalog = new Katalog<Auto>();  
mojKatalog.pridaj(new Auto("Peugeot", "207"));  
mojKatalog.pridaj(new Auto("Škoda", "Oktávia"));  
...  
Auto oktavia = mojKatalog.dajNaPozicii(1);
```

# Vyriešený problém (znovu katalóg kníh)

```
Katalog<Kniha> mojKatalog = new Katalog<Kniha>();  
mojKatalog.pridaj(  
    new Kniha("Bram Stocker", "Drakula"));  
mojKatalog.pridaj(  
    new Kniha("Sharon Zakhour", "Java 6"));  
  
... // (o 100 riadkov ďalej a 3 týždne neskôr)  
mojKatalog.pridaj(new Auto("Škoda", "Oktávia"));  
// chyba pri preklade
```

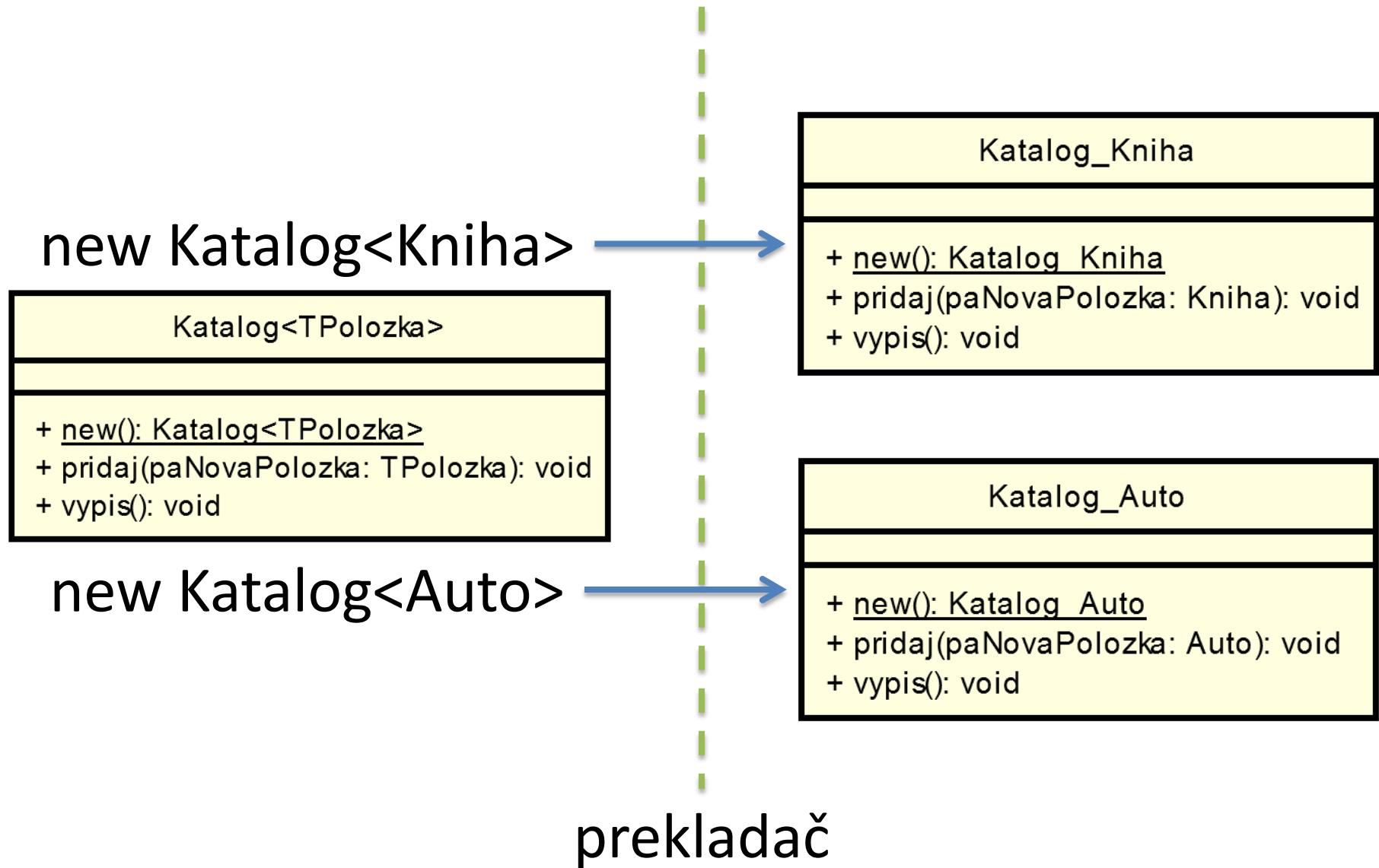
# Výhody využitia generickej triedy

- úplná typová kontrola pri preklade
- netreba pretypovávať
- nie je možné vložiť položku iného typu

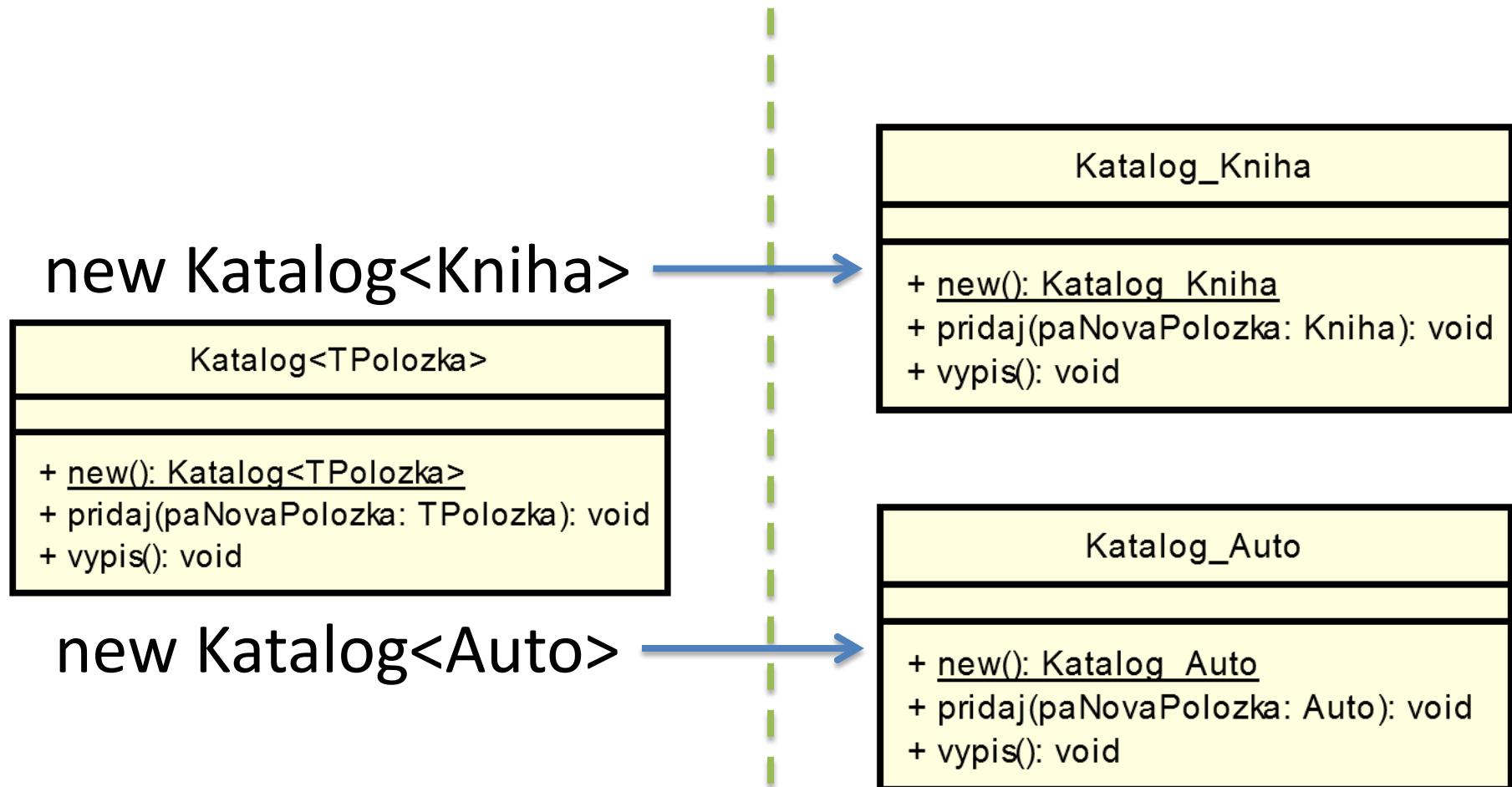
# Riešenia generických tried v jazykoch

- pri preklade
  - preklad na viac tried – C++
  - preklad na jednu triedu – Java
- za behu
  - preklad na viac tried – C#

# Riešenie pri preklade – na viac tried

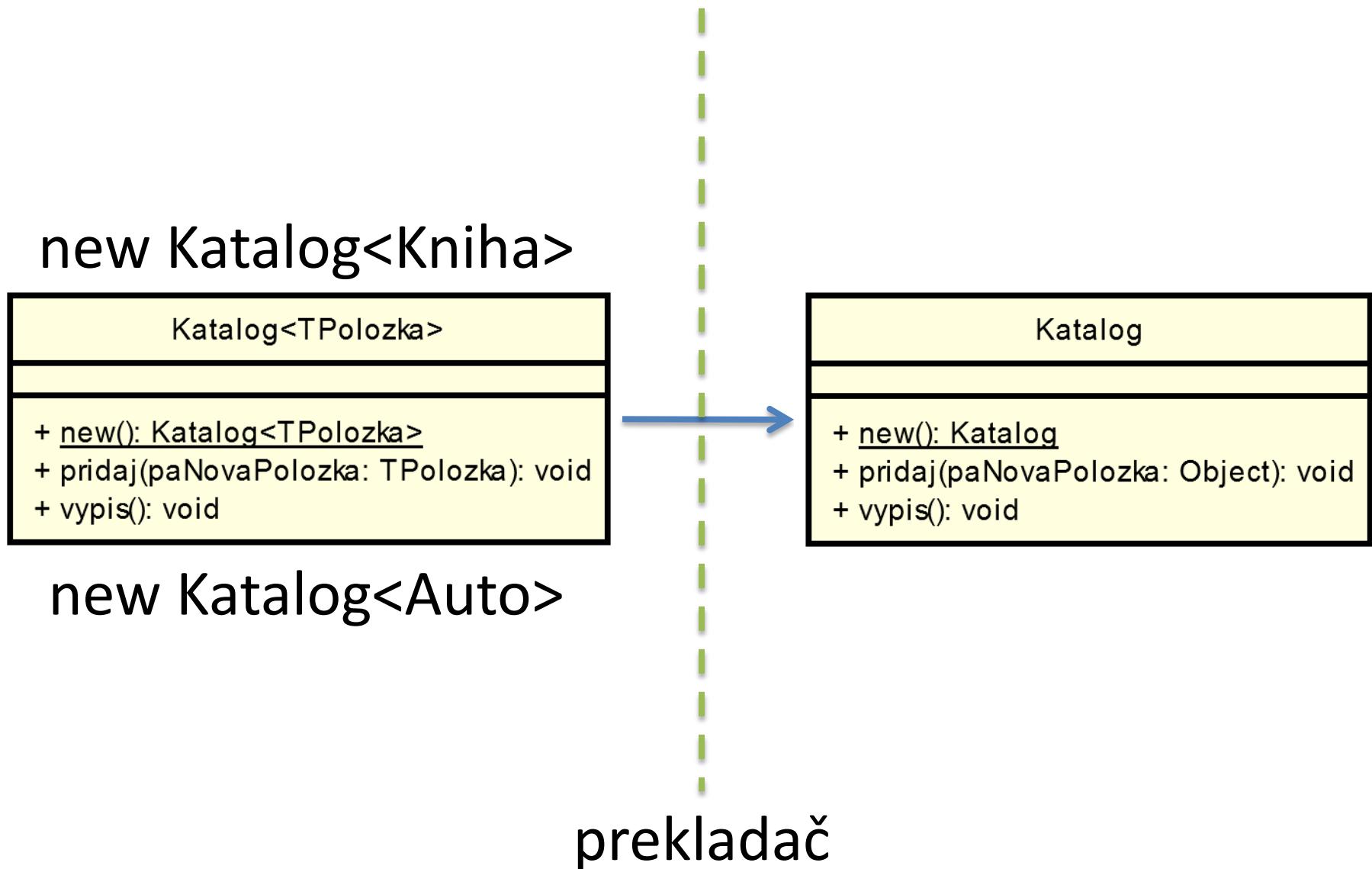


# Riešenie za behu – na viac tried



behové prostredie  
(napr. JVM pre platformu Java)

# Riešenie pri preklade – na jednu triedu



# Riešenie v jazyku Java

- tretia možnosť
- špeciálny proces pri preklade
  - type erasure – odstraňovanie typov
  - zmena typových parametrov na typ Object
  - pridanie pretypovaní

# Problémy použitého riešenia

- nefunguje „new TypovyParameter“
- nefunguje „instanceof TypovyParameter“
- dá sa vytvoriť príliš všeobecná implementácia pomocou

**new Katalog()**

# Príklad – prvé dva problémy

```
public class Katalog<TPolozka>
```

```
{
```

```
...
```

```
    TPolozka implicitnaHodnota = new TPolozka();
```

```
...
```

```
    if (objekt instanceof TPolozka) {
```

```
        TPolozka polozka = (TPolozka)objekt;
```

```
}
```

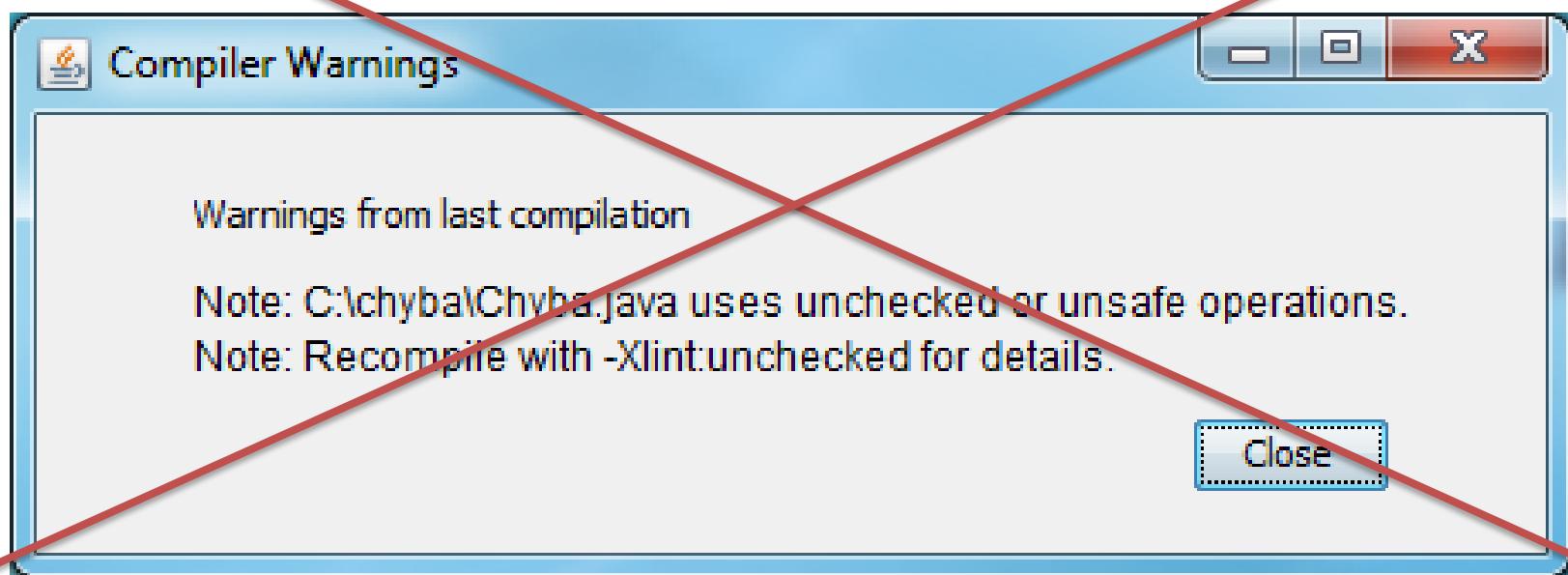
```
...
```

```
}
```

# Tretí problém

Katalog vseobecny = **new** Katalog();

- nerobí sa typová kontrola
- zobrazí sa varovanie:



# Rozšírenie zadania

- chceme v katalógu vyhľadávať
  - nájst' knihu podľa titulu/autora
  - nájst' auto podľa ŠPZ
  - ...

# Klasické riešenie

- každá položková trieda má v rozhraní správu obsahuje(retazec)
  - retazec = porovnávaný reťazec
- katalóg sa každej položky spýta, či obsahuje hľadaný reťazec
- vráti prvú položku, pre ktorú obsahuje(retazec) vráti true
  - retazec = hľadaný reťazec

# Metóda Kniha.obsahuje

```
public boolean obsahuje(String paHodnota)  
{  
    return aAutor.equals(paHodnota)  
        || aTitul.equals(paHodnota);  
}
```

# Máme problém

The screenshot shows a Java code editor window titled "Katalog". The menu bar includes "Class", "Edit", "Tools", and "Options". The toolbar contains buttons for "Compile", "Undo", "Cut", "Copy", "Paste", "Find...", and "Close". A dropdown menu labeled "Source Code" is open. The code editor displays the following Java code:

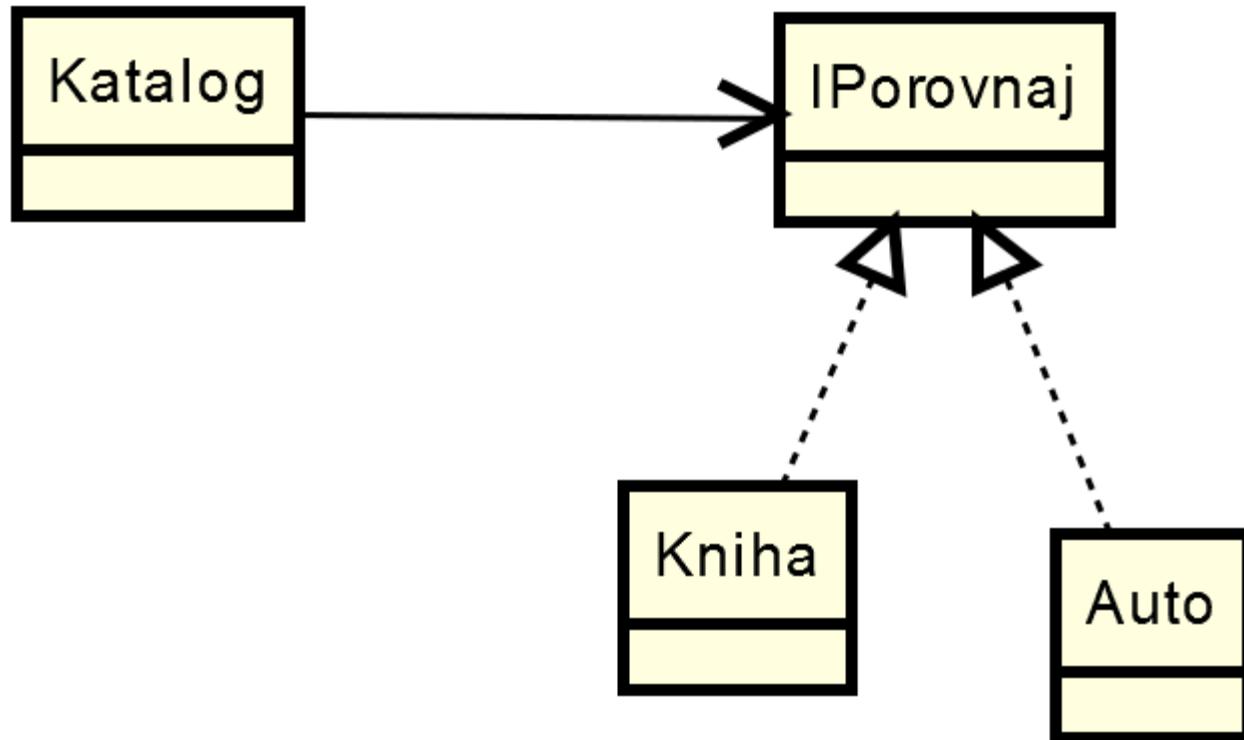
```
25     public TPolozka vyhľadaj(String paHodnota)
26     {
27         for (TPolozka polozka : aZoznamPoloziek) {
28             if (polozka.obsahuje(paHodnota))
29                 return polozka;
30         }
31     }
```

The word "obsahuje" is highlighted in red, indicating a syntax error. The status bar at the bottom of the IDE window displays the message: "cannot find symbol - method obsahuje(java.lang.String)".

# Chyba pri preklade

- prekladač nevie, že všetky položky budú mať správu obsahuje
- riešenie:
  - polymorfizmus
- čo s generikami?

# Generiká a polymorfizmus



# Tri možnosti riešenia

- interface ako typ do a ZozamPoloziek
- nahradíť TPolozka za IPorovnaj
- obmedzenie typového parametra

# Interface ako typ do aZozamPoloziek

```
private ArrayList<TPolozka> aZoznamPoloziek;
```

=>

```
private ArrayList<IPorovnaj> aZoznamPoloziek;
```

- zachováme generickú triedu
- musíme pridať pretypovania z TPolozka na IPorovnaj
- pri preklade sa nekontroluje, či sú položky IPorovnaj

# Metóda Katalog.pridaj

```
public class Katalog<TPolozka>
{
    private ArrayList<IPorovnaj> aZoznamPoloziek;
    ...
    public void pridaj(TPolozka paNovaPolozka)
    {
        aZoznamPoloziek.add((IPorovnaj)paNovaPolozka);
    }
}
```

# Metóda Katalog.pridaj – lepšie riešenie

```
public void pridaj(TPolozka paNovaPolozka)
{
    if (paNovaPolozka instanceof IPorovnaj) {
        aZoznamPoloziek.add((IPorovnaj)paNovaPolozka);
    }
}
```

# Nahradenie TPolozka za IPorovnaj

- prestávame používať generickú triedu
- nahrádzame za klasický polymorfizmus

# Metóda Katalog.pridaj

```
public class Katalog {  
    private ArrayList<IPorovnaj> aZoznamPoloziek;  
  
    public void pridaj(IPorovnaj paNovaPolozka) {  
        aZoznamPoloziek.add(paNovaPolozka);  
    }  
    ...  
}
```

# Obmedzenie typového parametra

- špeciálna syntax

```
<typovyParameter extends typ>
```

- definovanie, aké typy môžu byť použité ako hodnota typového parametra
- kontrola typového parametre pri preklade

# Trieda Katalog, obmedzenie

```
public class Katalog<TPolozka extends IPorovnaj>
{
    private ArrayList<TPolozka> aZoznamPoloziek;

    public void pridaj(TPolozka paNovaPolozka)
    {
        aZoznamPoloziek.add(paNovaPolozka);
    }

    ...
}
```

# Nové zadanie

- prechádzanie katalógu pomocou foreach

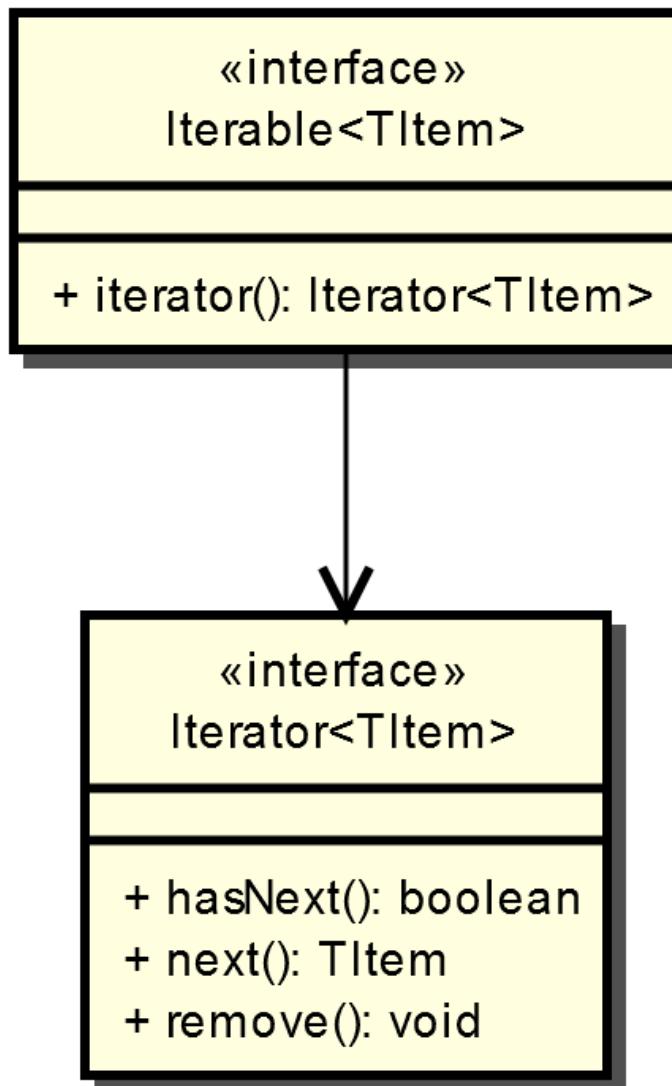
# Konštrukcia foreach (opakovanie)

```
for (TypPrvkov prvok : kontajner) {  
    // telo cyklu  
}
```

- je to isté ako

```
Iterator<TypPrvkov> prst = kontajner.iterator();  
while (prst.hasNext()) {  
    TypPrvkov prvok = prst.next();  
    // telo cyklu  
}
```

# Iterable a Iterator (opakovanie)



# Prechádzanie vlastného kontajnera

- treba implementovať interface Iterable<TItem>
- generický interface

# Implementácia interface Iterable

```
public class Katalog<TPolozka extends IPorovnaj>
    implements Iterable<TPolozka>
{
    ...
    public Iterator<TPolozka> iterator()
    {
        return aZoznamPoloziek.iterator();
    }
    ...
}
```

# Generické interface

- Syntax
  - podobne ako u triedy

```
public interface NazovInterface<TypoveParametre>
```

- typy sa kontrolujú aj pri implementácii interface triedou

# Implementácia interface Iterable

```
public class Katalog<TPolozka extends IPorovnaj>
    implements Iterable<TPolozka>
{
    ...
    public Iterator<TPolozka> iterator()
    {
        return aZoznamPoloziek.iterator();
    }
    ...
}
```

# Generické metódy

- syntax:

```
modifikatory <TypoveParametre>
typNavratovejHodnoty nazovMetody(parametre);
```

- príklad:

```
private <T> void vypisVsetko(Iterable<T> paZoz);
```

# Poslanie generickej správy

- syntax:

```
adresat.<TypoveParametre>selektor(parametre);
```

- príklad:

```
private ArrayList<Integer> aCisla;
```

```
...
```

```
this.<Integer>vypisVsetko(aCisla);
```

# Automatické odvodzovanie typov

- pri poslaní správy

```
private ArrayList<Integer> aCisla;
```

...

```
this.vypisVsetko(aCisla);
```

- automaticky sa určí
- musí byť Integer
- iba ak sú všetky typové parametre použité vo formálnych parametroch metódy

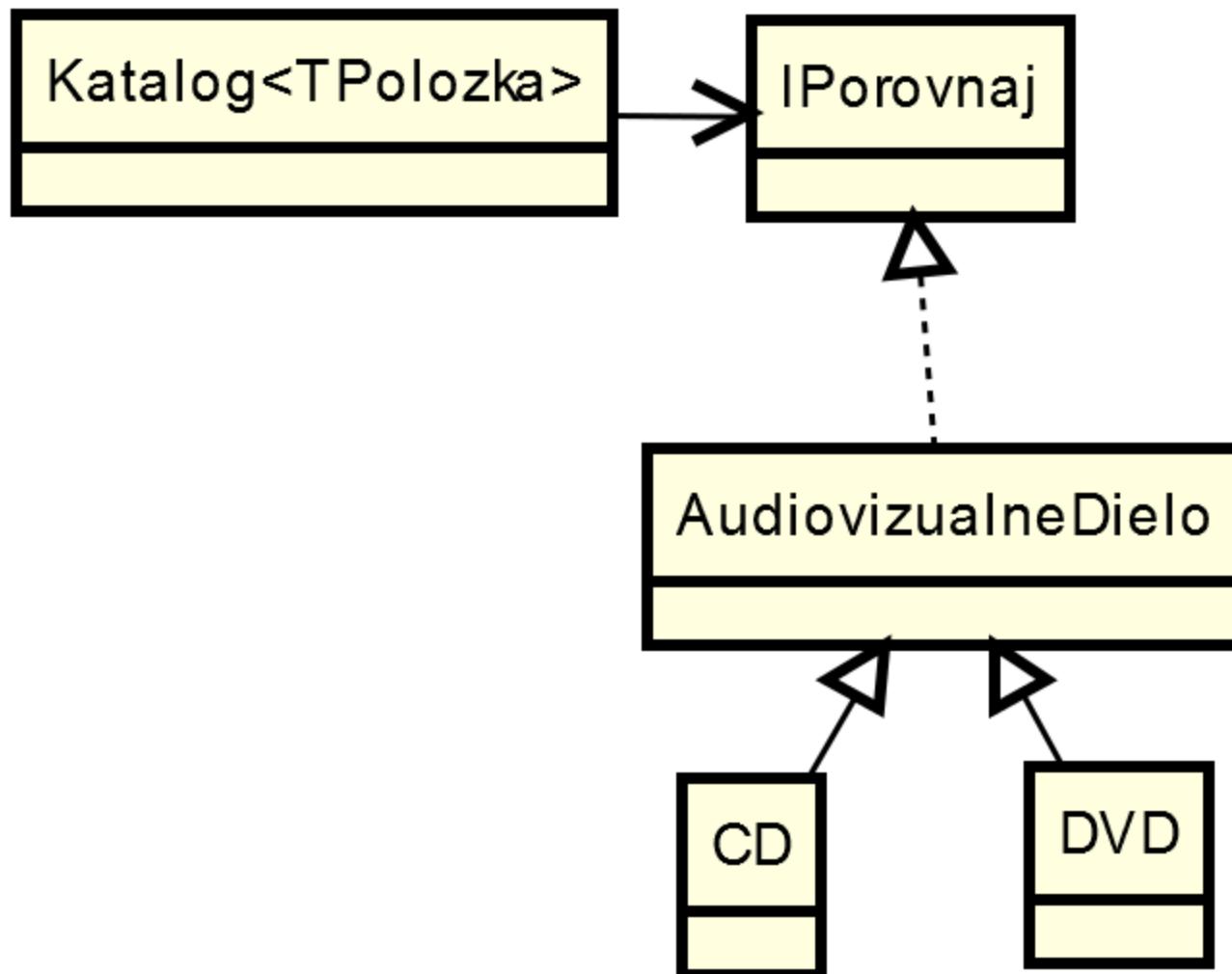
# Nové zadanie

- hromadné pridanie položiek z kontainera  
ArrayList

# Metóda Katalog.pridajVsetky

```
public void pridajVsetky(  
    Iterable<TPolozka> paPolozky)  
{  
    for (TPolozka pol : paPolozky) {  
        aZoznamPoloziek.add(pol);  
    }  
}
```

# Katalóg audiovizuálnych diel



# Použitie pridajVsetky

Katalog<AudiovizualneDielo> katalog

= **new** Katalog<AudiovizualneDielo>();

...

ArrayList<AudiovizualneDielo> zoznam

= **new** ArrayList<AudiovizualneDielo>();

zoznam.add(**new** CD("Beatles"));

...

katalog.pridajVsetky(zoznam);

# Použitie pridajVsetky, nefunguje

Katalog<AudiovizualneDielo> katalog

= **new** Katalog<AudiovizualneDielo>();

...

```
ArrayList<CD> zoznam = new ArrayList<CD>();  
zoznam.add(new CD("Beatles"));
```

...

```
katalog.pridajVsetky(zoznam);
```

# Použitie pridajVsetky, nefunguje

```
8 Katalog<AudiovizualneDielo> katalog
9         = new Katalog<AudiovizualneDielo>();
10 ArrayList<CD> zoznam = new ArrayList<CD>();
11 zoznam.add(new CD("Beatles"));
12
13     katalog.pridajVsetky(zoznam);
14
```

pridajVsetky(java.util.ArrayList<AudiovizualneDielo>) in Katalog <AudiovizualneDielo> cannot be applied to (java.util.ArrayList<CD>)

# Divoké karty – wildcards

- definícia premennej
- typový parameter bez konkrétneho typu
- napr.

ArrayList<? **extends** TPolozka> paPolozky

– miesto

ArrayList<TPolozka> paPolozky

# Metóda pridajVsetky, divoké karty

```
public void pridajVsetky  
        (Iterable<? extends TPolozka> paPolozky)  
{  
    for (TPolozka pol : paPolozky) {  
        aZoznamPoloziek.add(pol);  
    }  
}
```

# Java 7 – diamantový operátor

- zjednodušenie zápisu
- príklad:

ArrayList<String> poznamky

= **new ArrayList<String>();**

- sa dá zapísat' ako

ArrayList<String> poznamky = **new ArrayList<>();**



diamond operator

# Vďaka za pozornosť