

Informatika 2

Generiká



Pojmy zavedené v 6. prednáške (1)

- behové chyby
- komunikácia klient-server

Pojmy zavedené v 6. prednáške (2)

- defenzívna programovanie
- server – informovanie o chybách
- používateľa
- klienta
 - návratová hodnota
 - výnimka

Pojmy zavedené v 6. prednáške (3)

- výnimky – hierarchia
- druhy
 - Error
 - Exception
 - RuntimeException
- kontrolované výnimky
- nekontrolované výnimky

Pojmy zavedené v 6. prednáške (4)

- vyhadzovanie výnimiek
- príkaz throw
- klauzula throws

Pojmy zavedené v 6. prednáške (5)

- zachytávanie výnimiek
- príkaz try
- try-catch
- try-catch-finally
- try-finally

Cieľ prednášky

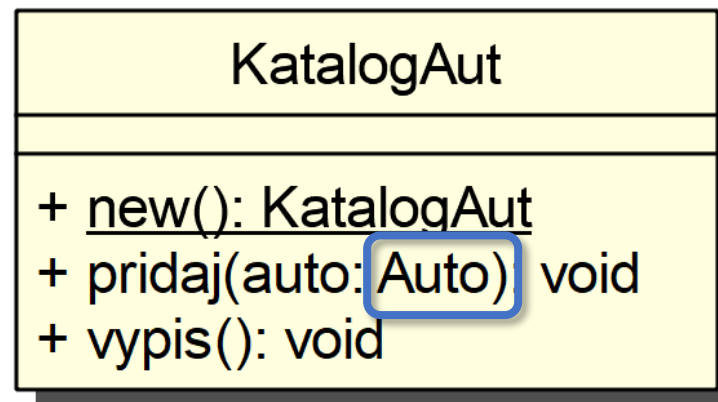
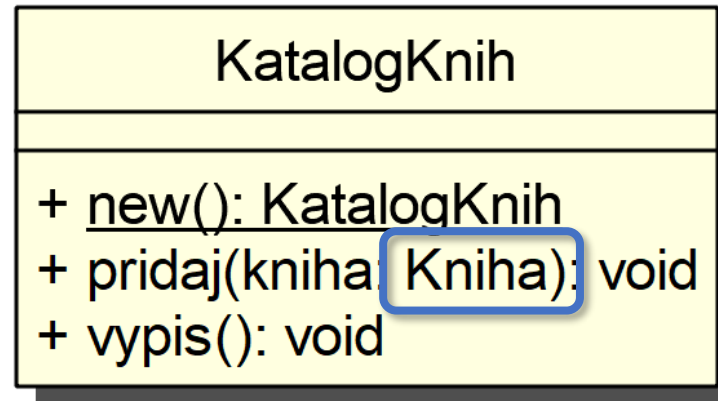
- generické triedy
- generické metódy
- príklad: všeobecný katalóg



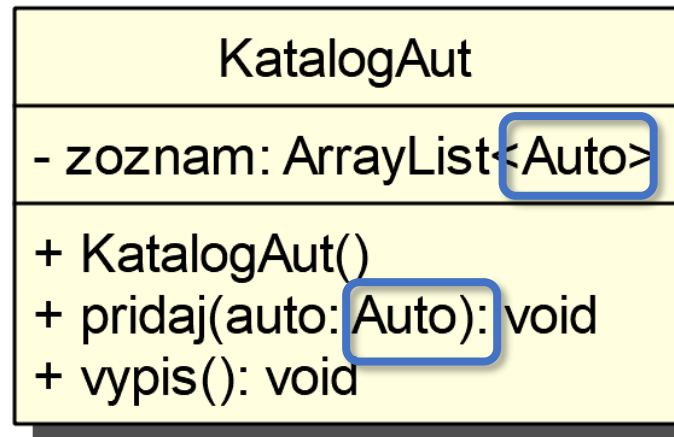
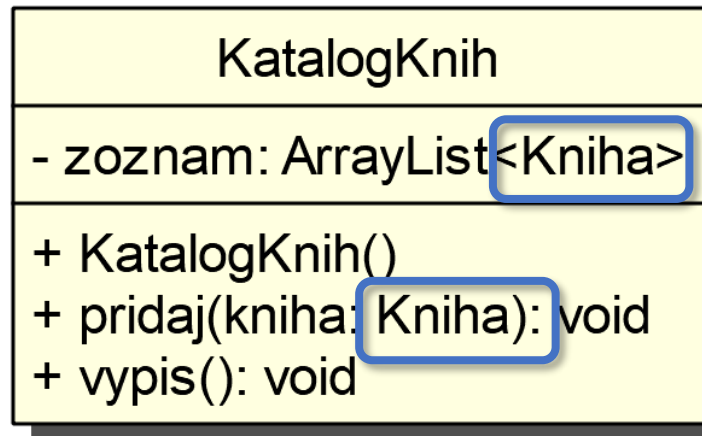
Všeobecný katalóg – zadanie

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

Katalóg kníh vs. katalóg áut (1)



Katalóg kníh vs. katalóg áut (2)



Trieda KatalogAut

```
public KatalogAut() {  
    this.zoznam = new ArrayList<Auto>();  
}  
public void pridaj(Auto auto) {  
    this.zoznam.add(auto);  
}  
public void vypisPolozky() {  
    for (var auto : this.zoznam) {  
        System.out.println(auto);  
    }  
}
```

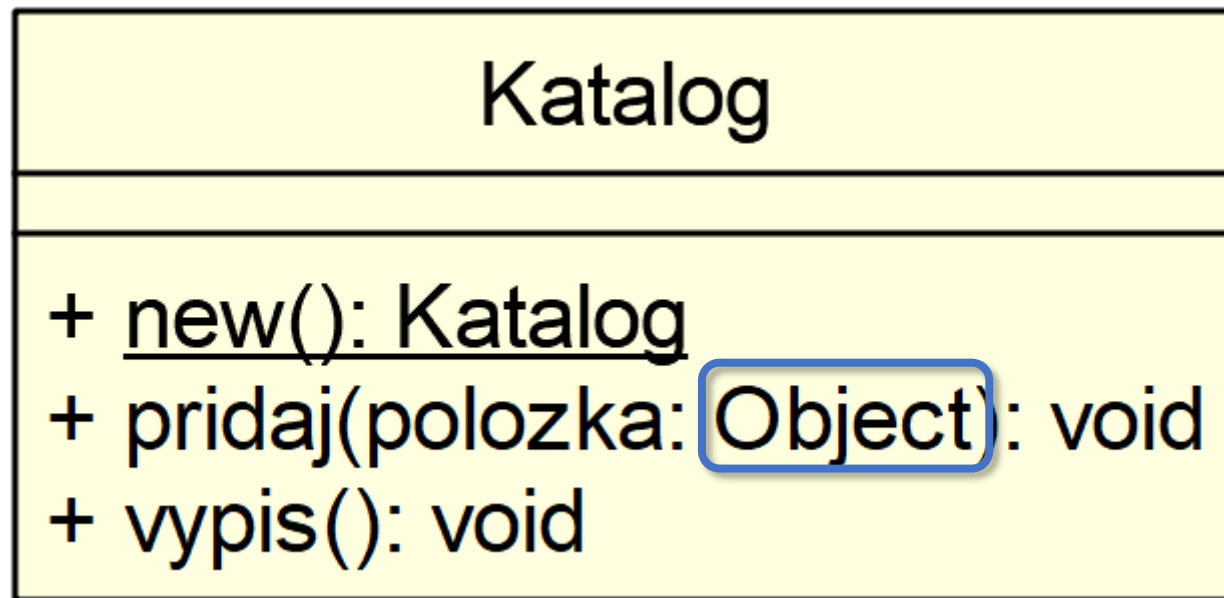
Trieda KatalogKnih

```
public KatalogKnih() {  
    this.zoznam = new ArrayList<Kniha>();  
}  
public void pridaj(Kniha kniha) {  
    this.zoznam.add(kniha);  
}  
public void vypisPolozky() {  
    for (var kniha : this.zoznam) {  
        System.out.println(kniha);  
    }  
}
```

Riešenie

- polymorfizmus
- čo použiť ako spoločný typ?
 - interface PolozkaKatalogu
 - 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 Stoker", "Drakula"));  
mojKatalog.pridaj(new Kniha("Sharon Zakhour", "Java 6"));  
...  
Kniha drakula = mojKatalog.getNaPozicii(0); // chyba pri preklade  
  
Kniha drakula = (Kniha)mojKatalog.getNaPozicii(0);
```

Použitie (katalóg áut)

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


Problém (znovu katalóg kníh)

```
Katalog mojKatalog = new Katalog();  
mojKatalog.pridaj(new Kniha("Bram Stoker", "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.getNaPozicii(2);
```

Výsledok riešenia

- odstránili sme duplicity
- 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

Trieda Katalog (1)

```
public class Katalog<E> {  
    private ArrayList<E> zoznamPoloziek;  
    ...  
}
```

Trieda Katalog (2)

```
public Katalog() {  
    this.zoznam = new ArrayList<E>();  
}  
public void pridaj(E polozka) {  
    this.zoznam.add(polozka);  
}  
public void vypisPolozky() {  
    for (var polozka : this.zoznam) {  
        System.out.println(polozka);  
    }  
}
```

Použitie (katalóg kníh)

```
Katalog<Kniha> mojKatalog = new Katalog<Kniha>();  
mojKatalog.pridaj(new Kniha("Bram Stoker", "Drakula"));  
mojKatalog.pridaj(new Kniha("Sharon Zakhour", "Java 6"));  
...  
Kniha drakula = mojKatalog.getNaPozicii(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.getNaPozicii(1);
```

Vyriešený problém (znovu katalóg kníh)

```
Katalog<Kniha> mojKatalog = new Katalog<Kniha>();  
mojKatalog.pridaj(new Kniha("Bram Stoker", "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
```

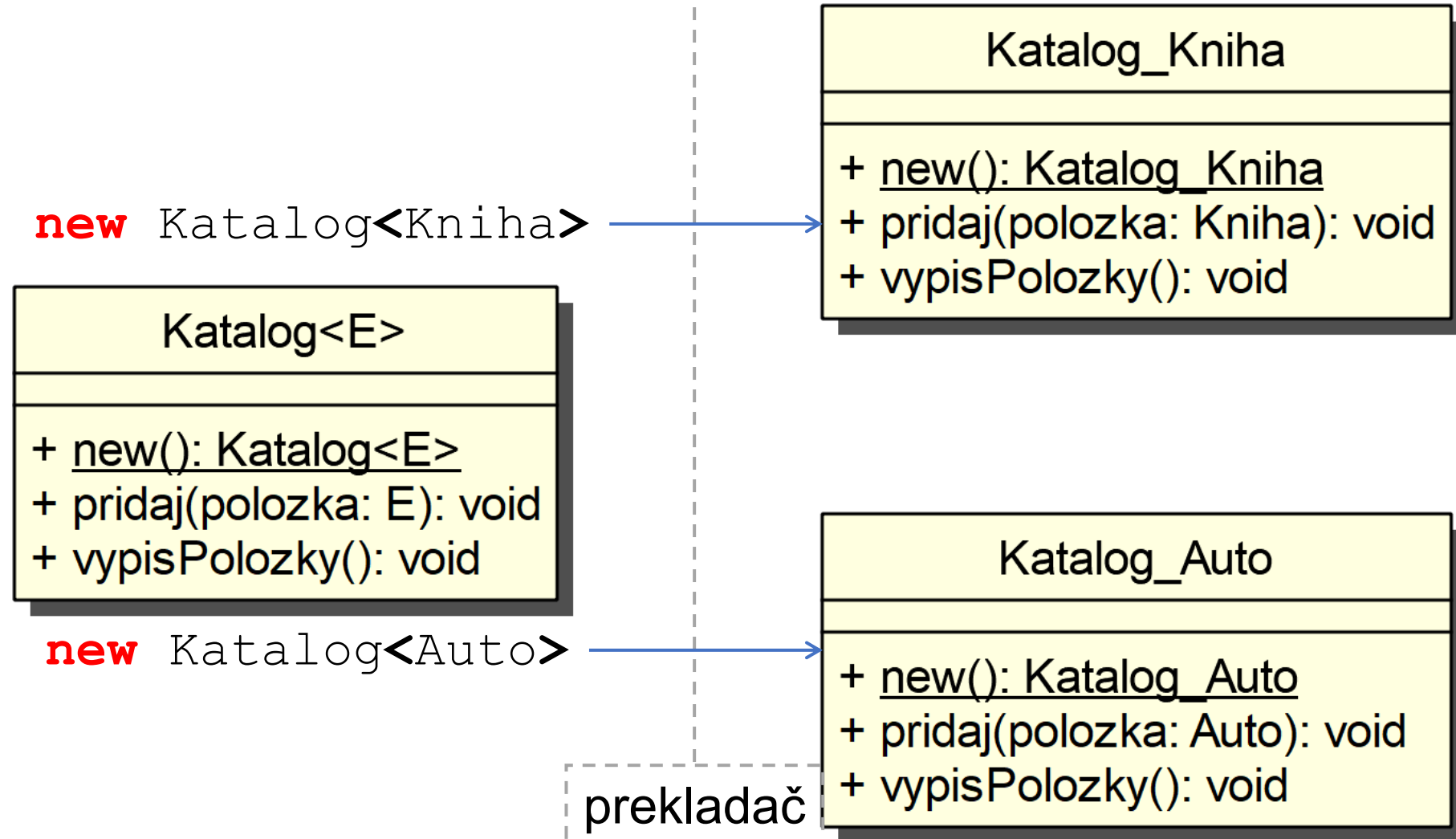
Výhody využitia generickej triedy

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

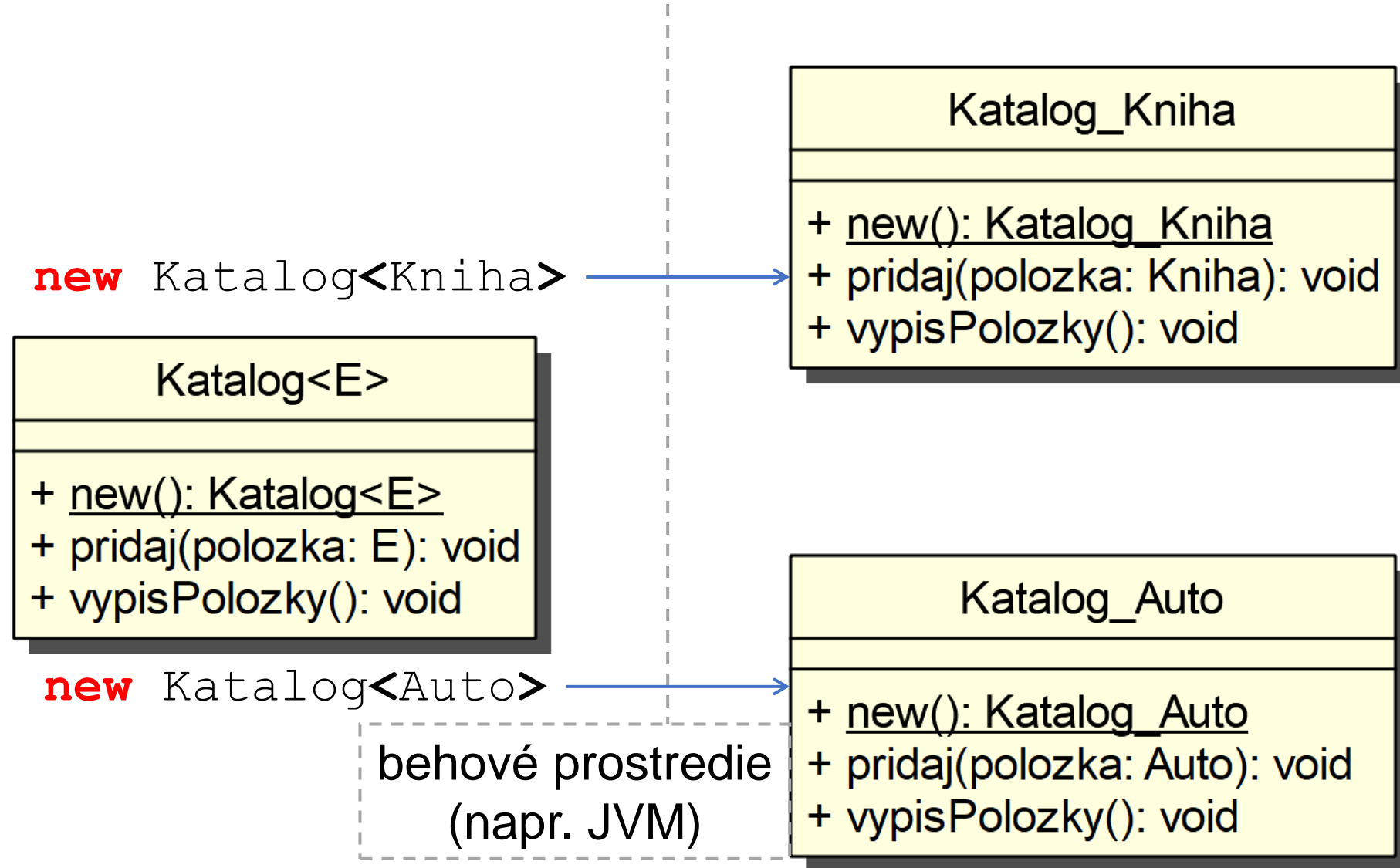
Riešenia generických tried v jazykoch

- 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

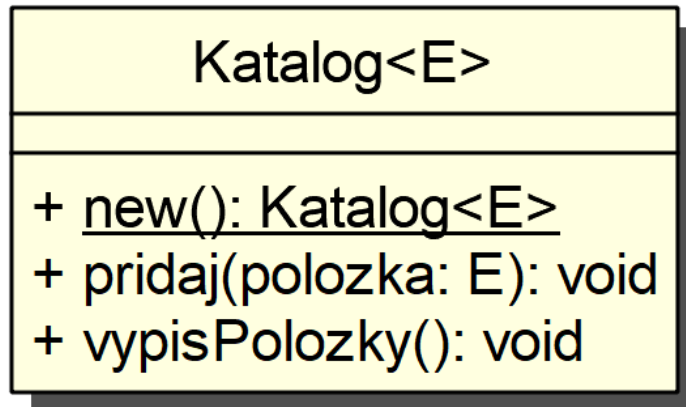


Riešenie za behu – na viac tried

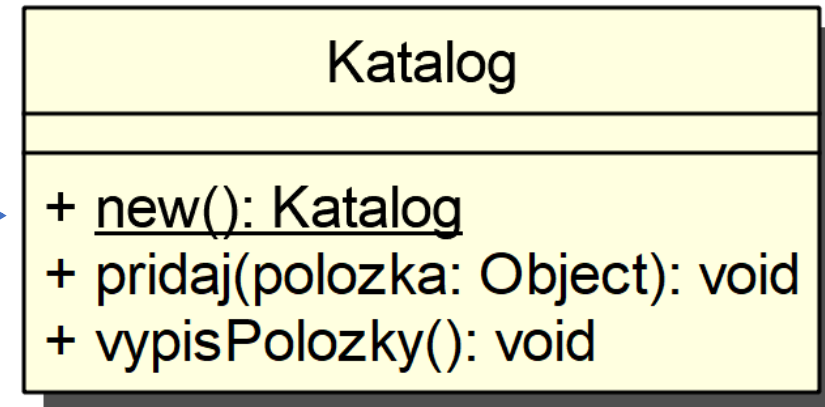


Riešenie pri preklade – na jednu triedu

new Katalog<Kniha>



new Katalog<Auto>



prekladač

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<E> {  
    ...  
    E implicitnaHodnota = new E();  
    ...  
    if (objekt instanceof E) {  
        E polozka = (E)objekt;  
    }  
    ...  
}
```

Tretí problém

```
Katalog vseobecny = new Katalog();
```

- nerobí sa typová kontrola
- zobrazí sa poznámka pri preklade:

Note: ... \Kcaib.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

Rozšírenie zadania

- chceme v katalógu vyhľadávať
 - nájsť knihu podľa titulu/autora
 - nájsť 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 hodnota) {  
    return this.autor.equals(hodnota) || this.titul.equals(hodnota);  
}
```

Máme problém

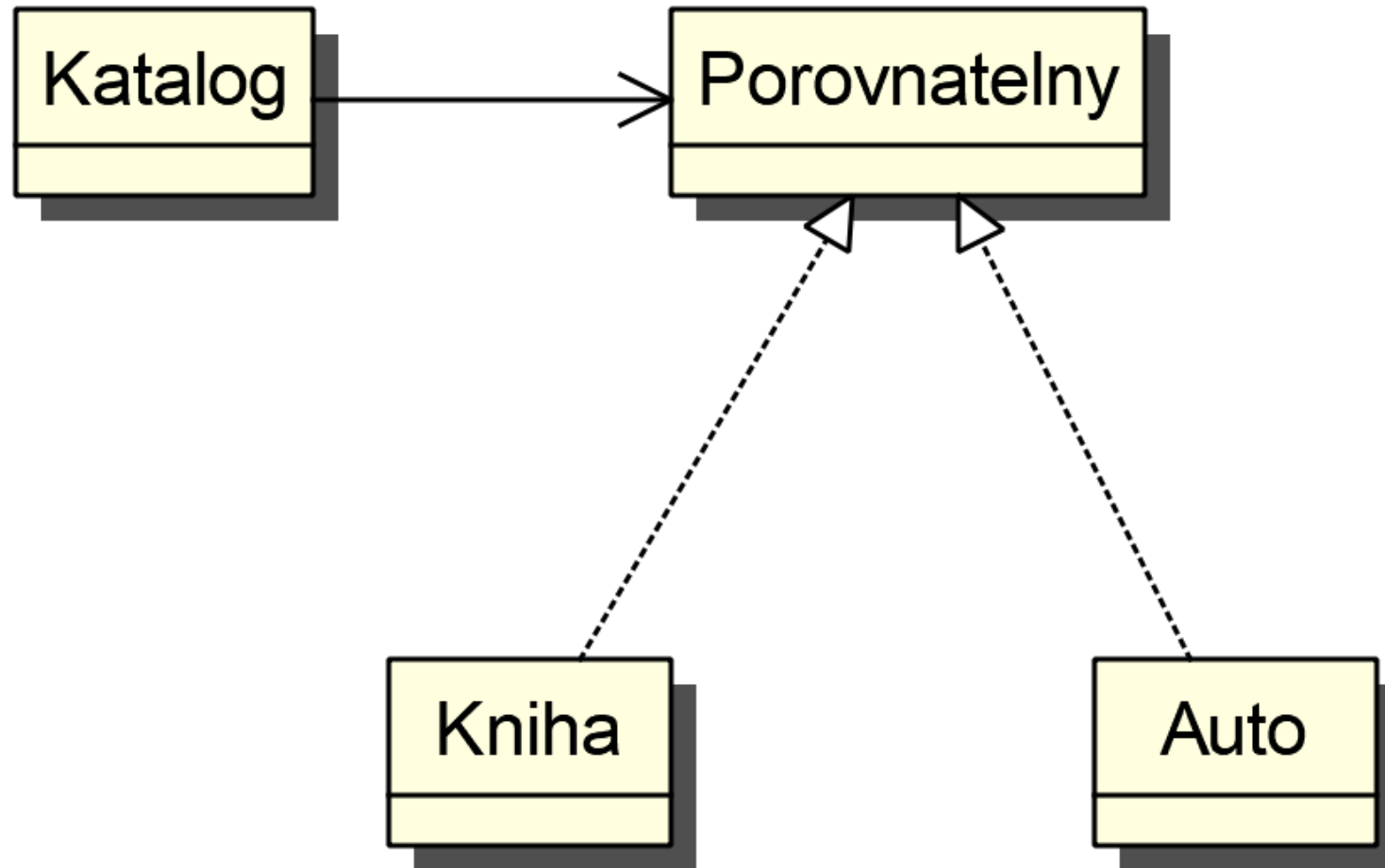
```
public E vyhladaj(String hodnota) {  
    for (var polozka : this.zoznam) {  
        if (polozka.obsahuje(hodnota)) {  
            return polozka;  
        }  
    }  
    return null;  
}
```

Cannot find symbol
symbol: method obsahuje(String)
location: variable polozka of type E

Chyba pri preklade

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

Generiká a polymorfizmus



Štyri možnosti riešenia

- Porovnateľný ako typ do zoznamPoloziek
- nahradiť E za Porovnateľný
- bezpečné pretypovanie vo vyhľadaj
- obmedzenie typového parametra

Interface ako typ do zoznamPoloziek

```
private ArrayList<E> zoznamPoloziek;
```

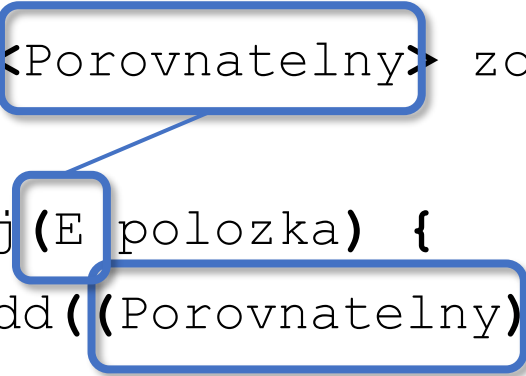
=>

```
private ArrayList<Porovnatelny> zoznamPoloziek;
```

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

Metóda pridaj

```
public class Katalog<E> {  
    private ArrayList<Porovnatelny> zoznam;  
    ...  
    public void pridaj(E polozka) {  
        this.zoznam.add((Porovnatelny)polozka);  
    }  
}
```



Metóda pridaj – možné riešenie?

```
public void pridaj(E polozka) {  
    if (polozka instanceof Porovnatelny porovnatelnaPolozka) {  
        this.zoznam.add(porovnatelnaPolozka);  
    }  
}
```

Nahradenie E za Porovnatelny

- prestávame používať generickú triedu
- nahrádzame za klasický polymorfizmus

Metóda Katalog.pridaj

```
public class Katalog  {  
    private ArrayList<Porovnatelny> zoznam;  
  
    public void pridaj( polozka) {  
        this.zoznam.add(polozka);  
    }  
    ...  
}
```

Bezpečné pretypovanie

- upravíme metódu vyhľadaj
 - pridáme pretypovanie na Porovnateľny

Úprava na bezpečné pretypovanie

```
public E vyhladaj(String hodnota) {  
    for (var polozka : this.zoznam) {  
        if (polozka instanceof Porovnatelny porovnatelnaPolozka) {  
            if (porovnatelnaPolozka.obsahuje(hodnota)) {  
                return polozka;  
            }  
        }  
    }  
    return null;  
}
```

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<E extends Porovnatelny> {  
    private ArrayList<E> zoznam;  
  
    public void pridaj(E polozka) {  
        this.zoznam.add(polozka);  
    }  
    ...  
}
```

Nové zadanie

- prechádzanie katalógu pomocou foreach

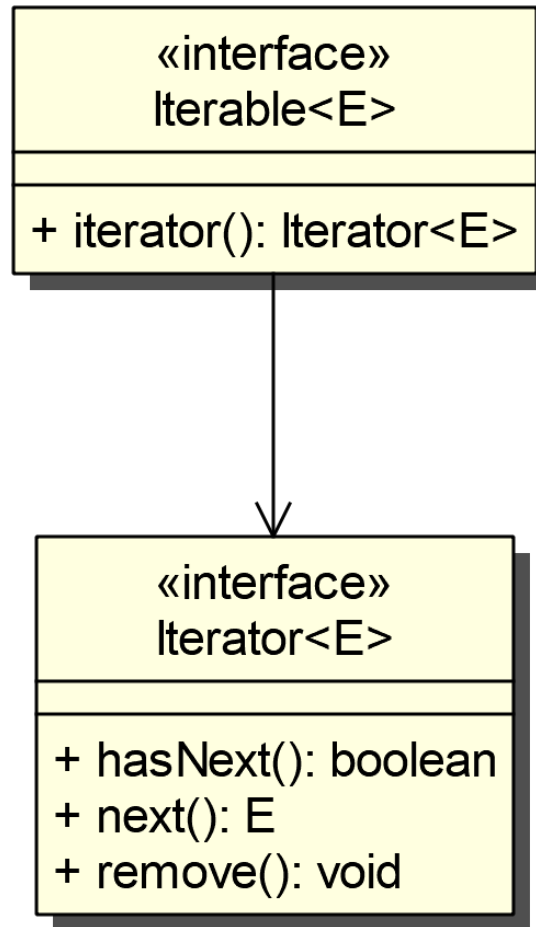
Konštrukcia foreach

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

- je to isté ako

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

Iterable a Iterator



Prechádzanie vlastného kontajnera

- treba implementovať interface `Iterable<E>`
- generický interface

Implementácia interface Iterable

```
public class Katalog<E extends Porovnateľný> implements Iterable<E> {  
    ...  
    public Iterator<E> iterator() {  
        return this.zoznamPoloziek.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<E extends Porovnatelny> implements Iterable<E> {  
    ...  
    public Iterator<E> iterator() {  
        return this.zoznamPoloziek.iterator();  
    }  
    ...  
}
```

Generické metódy

- syntax:

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

- príklad:

```
private <T> void vypisVsetko (Iterable<T> zoznam)
```

Poslanie generickej správy

- syntax:

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

- príklad:

```
private ArrayList<Integer> cisla;  
...  
this.<Integer>vypisVsetko (this.cisla) ;
```

Automatické odvodzovanie typov

- pri poslaní správy

```
private ArrayList<Integer> cisla;  
...  
this.vypisVsetko(this.cisla);
```

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

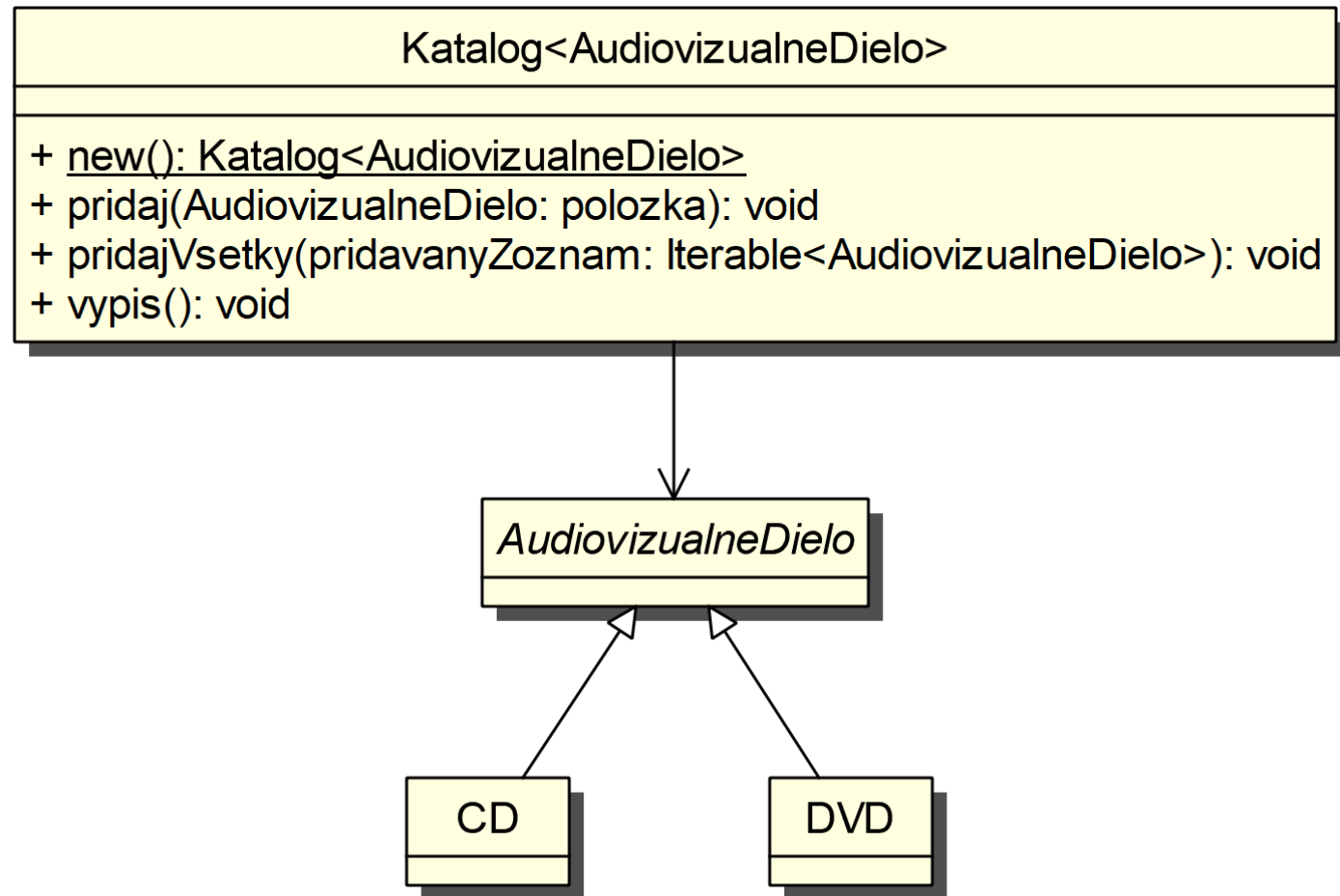
Nové zadanie

- hromadné pridanie položiek z kontajnera ArrayList

Metóda Katalog.pridajVsetky

```
public void pridajVsetky(Iterable<E> pridavanyZoznam) {  
    for (var polozka : pridavanyZoznam) {  
        this.zoznam.add(polozka);  
    }  
}
```

Katalóg audiovizuálnych diel



Použitie pridajVsetky

```
var katalog = new Katalog<AudiovizualneDielo>();  
...  
var zoznam = new ArrayList<AudiovizualneDielo>();  
zoznam.add(new CD("Beatles"));  
...  
katalog.pridajVsetky(zoznam);
```

Použitie pridajVsetky, nefunguje

```
var katalog = new Katalog<AudiovizualneDielo>();  
...  
var zoznam = new ArrayList<CD>();  
zoznam.add(new CD("Beatles"));  
...  
katalog.pridajVsetky(zoznam);
```

incompatible types: ArrayList<CD> cannot be converted to
Iterable<AudiovizualneDielo>

Divoké karty – wildcards

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

```
Iterable<? extends E> polozky
```

- miesto

```
Iterable<E> polozky
```

Metóda pridajVsetky, divoké karty

```
public void pridajVsetky(Iterable<? extends E> pridavanyZoznam) {  
    for (var polozka : pridavanyZoznam) {  
        this.zoznam.add(polozka);  
    }  
}
```