Polymorfismus a dědičnost

OMO, LS 2013/14

Typy

- Základní
  - logické hodnoty, znaky, čísla, adresy, …
- Slovene
  - kartézský součin (součinové)
  - disjunktní sjednocení (součtové)
  - zobrazení (pole, funkce) (mocninné)

Kartézský součin

- A x B
- Množina všech uspořádaných dvojic (a,b), kde
  a : A, b : B
- Projekční funkce (selektory):

  fst : A x B -> A

  fst (x,y) = x

  snd : A x B -> B

  snd (x,y) = y
- z = (x,y) : A x B, pak x = fst z, y = snd z

Record (Agregát)

- Datum = {den : Int, mesic : String, rok : Int}
- d : Datum
- d = Datum {den = 5, mesic = “listopad”, rok = 2013}
- d.den, d.mesic, …

Disjunktní sjednocení (Sum)

- *A + B*

- Sjednocení množiny všech dvojic (**L**, *a*) s množinou všech dvojic (**R**, *b*), kde *a : A, b : B.* L, R jsou příznaky původu prvku.

- Inserční funkce (konstruktory):

  inl : A -> A + B
  inl *x* = (**L**, *x*)
  inr : B -> A + B
  inr *y* = (**R**, *y*)

- *x* : A, *y* : B, *u* = (**L**, *x*), *v* = (**R**, *y*), pak *u* = inl *x*, *v* = inr *y*.

Union

- V typu union jsou definovány inverze konstruktorů.

```
Cislo = { cele : Int | desetinne : Float }
c : Cislo
c = Cislo { cele = 42 }
```

pak c.cele = 42 odpovídá inl 42 = c.

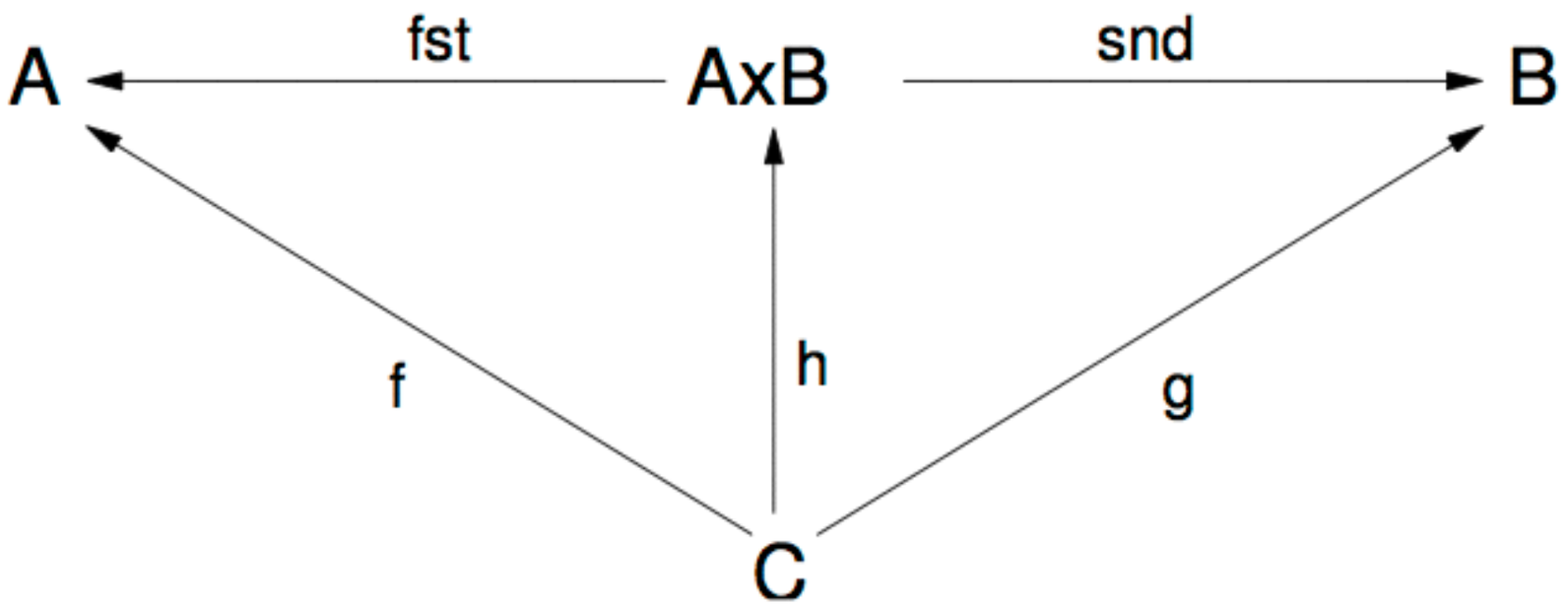
Obecné vlastnosti kartézského součinu a disjunktního sjednocení

Nechť A, B, C jsou typy, $f : C \to A$, $g : C \to B$ nějaké dvě funkce.

Pak existuje jediná funkce $h : C \to A \times B$ tak, že

$$\mathit{fst} \circ h = f$$

$$\mathit{snd} \circ h = g$$

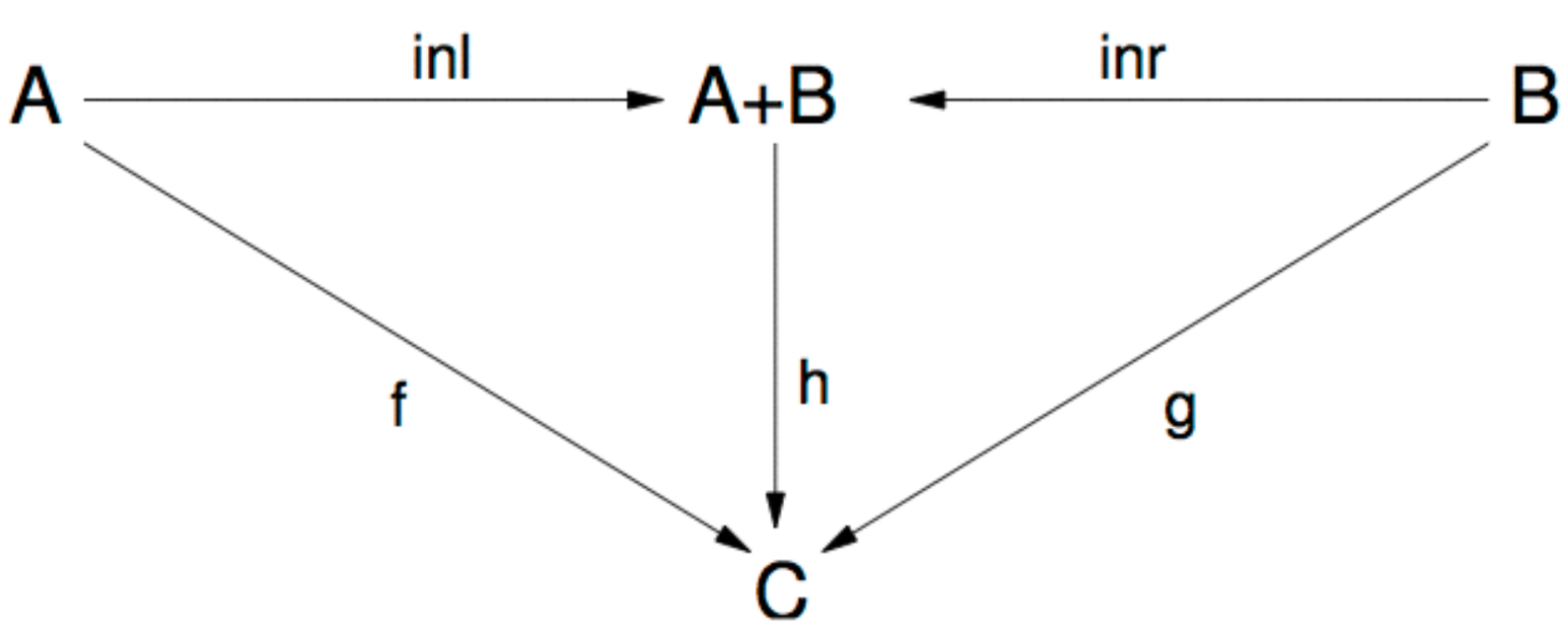


Nechť A, B, C jsou typu, $f : A \to C$, $g : B \to C$ nějaké dvě funkce.

Pak existuje jediná funkce $h : A + B \to C$ tak, že

$$h \circ \mathit{inl} = f$$

$$h \circ \mathit{inr} = g$$



Pole

- Zobrazení (konečného) ordinálního typu *I* do typu *T*.
  (Konečný typ *I* je ordinální <=> existuje bijekce ord : *I* -> {1, ..., n}, kde n = |*I*|)

- Array *I T*

Vícerozměrná pole

$$p = \overbrace{\begin{matrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \end{matrix}}^{J} \; I$$

Array $(I \times J)\, T$

$$q = I \left\{ \begin{array}{c} \overbrace{\begin{matrix} 11 & 12 & 13 & 14 \end{matrix}}^{J} \\ \overbrace{\begin{matrix} 21 & 22 & 23 & 24 \end{matrix}}^{J} \end{array} \right.$$

Array I (Array $J\, T$)

$$p[i, j] \cong q[i][j]$$

Funkce

- Zobrazení typu *A* do typu *B.*
- *A -> B*
- Funkce více proměnných

$$f = \begin{pmatrix} (1,1) \mapsto 11 & (1,2) \mapsto 12 & (1,3) \mapsto 13 & (1,4) \mapsto 14 \\ (2,1) \mapsto 21 & (2,2) \mapsto 22 & (2,3) \mapsto 23 & (2,4) \mapsto 24 \end{pmatrix} \quad A \times B \to C$$

$$g = \begin{pmatrix} 1 \mapsto & \big(1 \mapsto 11 & 2 \mapsto 12 & 3 \mapsto 13 & 4 \mapsto 14\big) \\ 2 \mapsto & \big(1 \mapsto 21 & 2 \mapsto 22 & 3 \mapsto 23 & 4 \mapsto 24\big) \end{pmatrix} \quad A \to (B \to C)$$

$$f(x,y) \cong g\ x\ y$$

Potenční množiny

Mocninný typ, speciální případ zobrazení.

Set A	typ všech podmnožin A
$A \to$ Bool	typ všech predikátů nad A

Každé hodnotě a : Set A odpovídá tzv. *charakteristická funkce* $\chi_a : A \to$ Bool taková, že pro každé $x : A$ platí $x \in a \Leftrightarrow \chi_a(x) =$ *true*.

Výčtové typy

- Zvláštní případ disjunktního sjednocení, kdy všechny inserce jsou typu Unit -> *T*.

  Bool = { false : Unit | true : Unit }
  Tyden = { po : Unit | ut : Unit | … | ne :Unit }
  { false = () }, { true = () }, { po = () }

- Běžně zapisujeme jako

  Bool = false | true
  false, true, …

Zvláštní typy

- Prázdný typ
  - Void (disjunktní sjednocení nulového počtu typů)
    - Neobsahuje žádnou hodnotu.
- Jednotkový typ
  - Unit (kartézský součin nulového počtu typů)
  - Má jedinou hodnotu - uspořádanou nultici ().

Rekursivní typy

- Je-li F(t) typový výraz obsahující typovou proměnnou t, pak

FIX t.F(t)

- je (množinově) nejmenší typ vyhovující rovnici

t = F(t)

- Definici typu T = FIX t.F(t) zapisujeme stručněji

T = F(T) - rekursivní definice typu

Příklad: přirozená čísla

$$\mathsf{Nat} = \mathrm{FIX}\, n.\, \{\mathit{nula} : \mathsf{Unit} \mid \mathit{naslednik} : n\}$$

tj.

$$\mathsf{Nat} = \{\mathit{nula} : \mathsf{Unit} \mid \mathit{naslednik} : \mathsf{Nat}\}$$

je typ s hodnotami

$$\{\mathit{nula} = ()\}$$

$$\{\mathit{naslednik} = \{\mathit{nula} = ()\}\}$$

$$\{\mathit{naslednik} = \{\mathit{naslednik} = \{\mathit{nula} = ()\}\}\}$$

$$\vdots$$

Příklad: seznamy

Je-li T nějaký typ, pak typ $\mathrm{FIX}\ l.\ \{\mathit{nil} : \mathrm{Unit} \mid \mathit{cons} : T \times l\}$ označujeme T^* a nazýváme typem všech seznamů nad T.

Pro $a_1, a_2, a_3, \ldots$ typu T píšeme

$$[\,] = \{\mathit{nil} = ()\}$$

$$[a_1] = \{\mathit{cons} = (a_1, \{\mathit{nil} = ()\})\}$$

$$[a_1, a_2] = \{\mathit{cons} = (a_1, \{\mathit{cons} = (a_2, \{\mathit{nil} = ()\})\})\}$$

$$\vdots$$

Ukázka typového systému

Typy jazyků

- Programovací jazyky můžeme rozdělit z hlediska typů do dvou skupin na:
  - monomorfní - každá hodnota nebo proměnná je právě jednoho typu
  - polymorfní - hodnoty a proměnné mohou mít více typů
    - parametrický polymorfismus - v typových výrazech se vyskytují typové proměnné, za něž lze dosadit libovolný typ
    - podtypový polymorfismus - v typovém systému se zavede mezi typy relace <: a každá hodnota má kromě svého (nejmenšího) typu i všechny nadtypy

Princip subsumpce

- Nechť *a* je typu *A* a *A*, *B* jsou typy takové, že *A* je podtypem *B*, pak *a* je i hodnotou typu *B*.

$$\frac{a : A \qquad A <: B}{a : B}$$

Substituční princip (LSP)

- Nechť q(x) je vlastnost, která platí pro všechny objekty typu T. Pak q(y) by měla být platná pro objekty y, které jsou typu T', kde T' je podtypem T.

$$\frac{\forall\, x : T \;\; : q(x) \qquad T' <: T}{\forall y : T' \;\; : q(y)}$$

Co je polymorfismus?

- Obecně se polymorfismem rozumí schopnost vyskytovat se v mnoha formách.

- Polymorfismus v Java

  - Schopnost reference chovat se dle skutečně odkazovaného objektu.

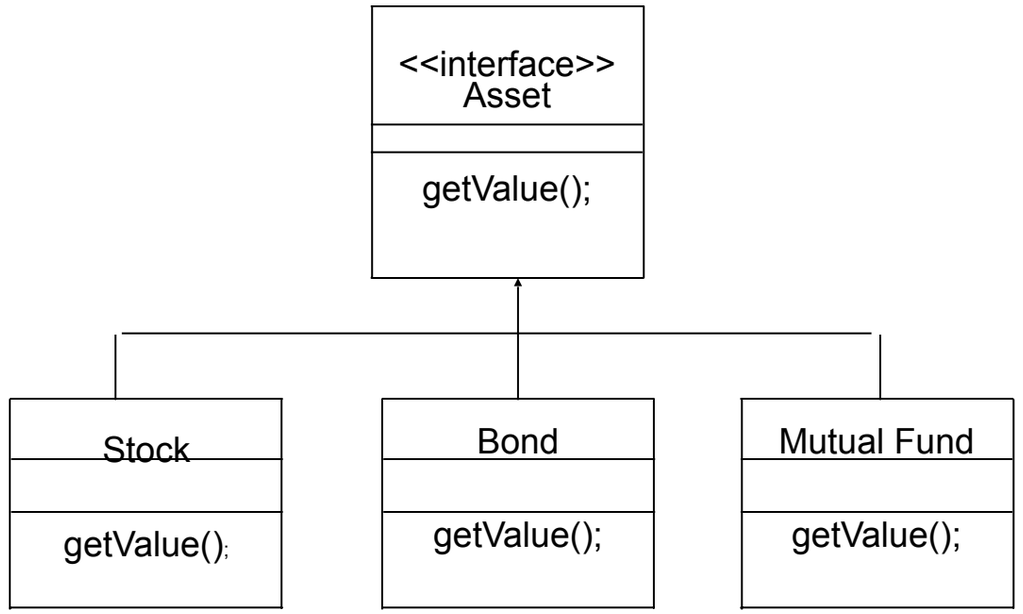
  - Výběr metody, která se volá probíhá za běhu programu podle toho na který objekt reference odkazuje.

Polymorfismus v Java

- V Java máme dostupný jak podtypový polymorfismus, tak parametrický.
- Podtypový polymorfismus je realizován pomocí rozhraní a dědičnosti tříd.
- Parametrický polymorfismus je realizován pomocí generic.
- Podtypový polymorfismus vychází z principu **subsumpce**.

Polymorfismus

- Je to způsob jak skrýt více implementací za jedno rozhraní.
- Umožňuje aby stejná zpráva byla obsloužena různě v závislosti na konkrétním objektu.



Příklad

- Mějme abstraktní třídu *Shape*, polymorfismus umožňuje programátorovi definovat různé metody *area* ve zděděných třídách Circle, Rectangle, ….

- Zavolání metody *area* na referenci typu shape vrátí správný výsledek s ohledem na skutečně odkazovaný objekt.

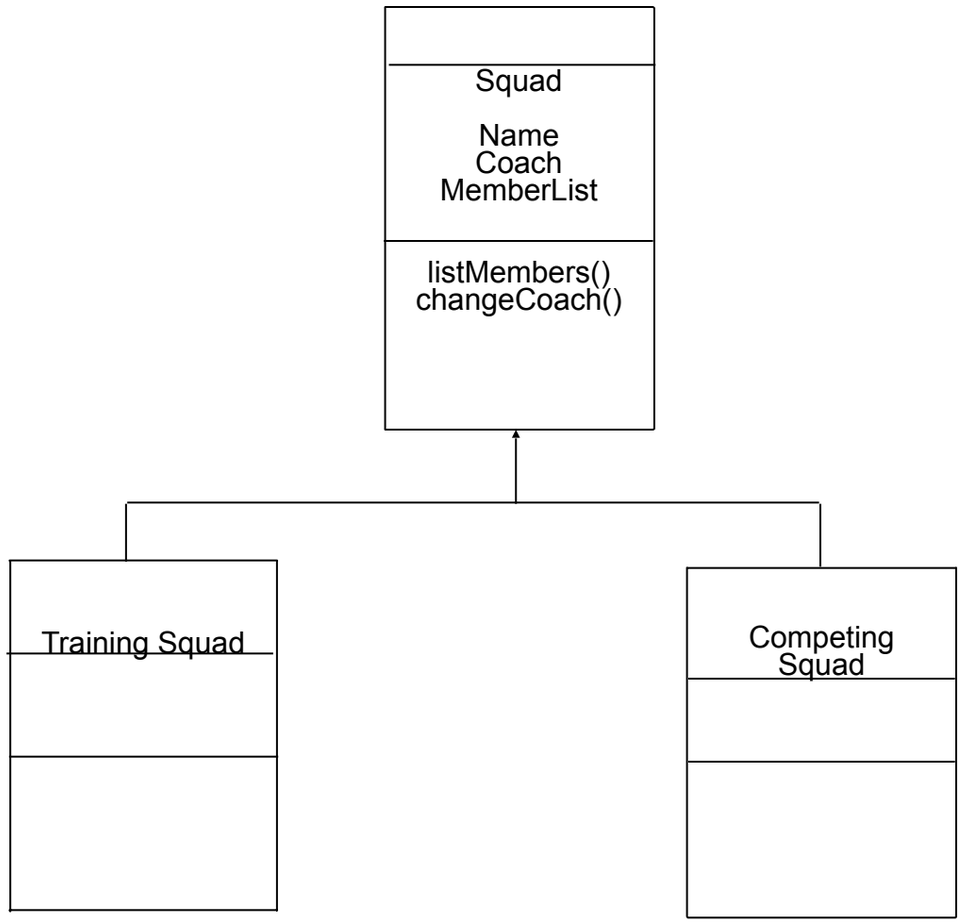
Generalizace

- Forma asociace, kde třída sdílí strukturu anebo chování jiných/jiné tříd/třídy.
- Definuje hierarchii.
  - Jednoduchá dědičnost
  - Vícenásobná dědičnost
- Je to vztah typu *kind of*.

Dědičnost

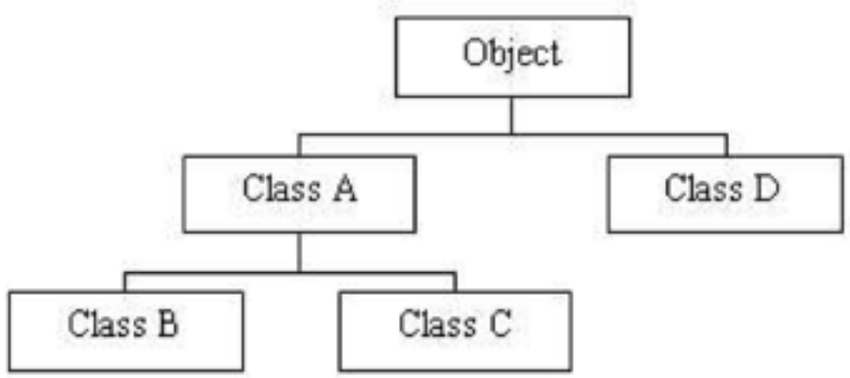
- Mechanismus jak více specifické třídy přejímají chování a strukturu obecnějších tříd.

- Třída dědí atributy, operace a vazby.



Dědičnost

- V Java jsou všechny třídy zděděné z třídy *Object*.
- V Java je k dispozici jednoduchá dědičnost.
- Hierarchie tříd se znázorňuje obvykle stromem



Dědičnost

- Výhoda dědičnosti v OOP - Znovupoužitelnost
  - Chování (metody) definované v nadtřídě je dostupné ve všech podtřídách.
  - Není nutné funkčnost metod definovat znovu.
  - Podtřídy mohou implementaci metod měnit a přidávat metody nové.

Dědičnost

- V Java používáme pro specifikaci dědičnosti klíčové slovo *extends* u třídy, která dědí.

```
public class Person {
      protected String name;
      protected String address;

      /**
       * Default constructor
       */
      public Person(){
            System.out.println(“Inside
Person:Constructor”);
            name = ""; address = "";
      }
      . . . .
}

public class Student extends Person {
      public Student(){
            System.out.println(“Inside Student:Constructor”);
      }
      . . . .
}
```

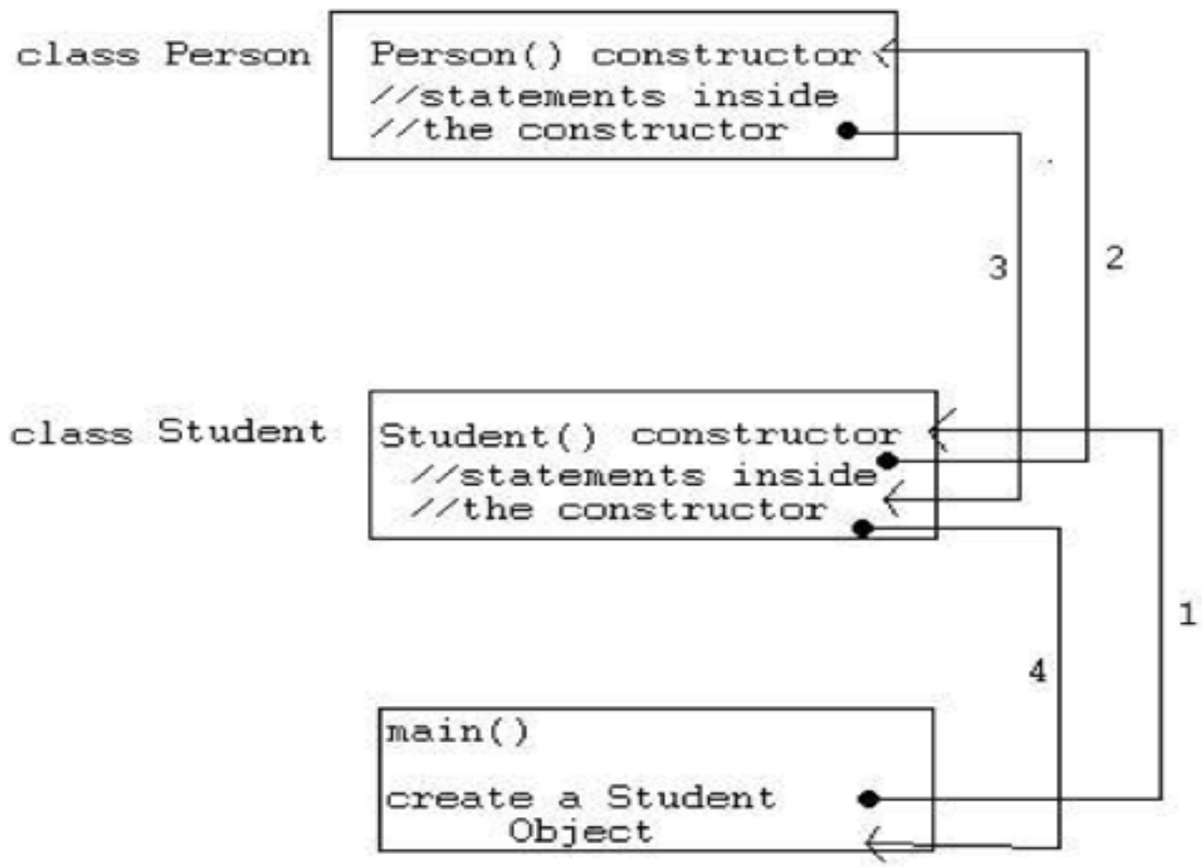
Dědičnost

Metoda main:

```
public static void main( String[] args ){
     Student anna = new Student();
}
```

Výstup programu:

```
Inside Person:Constructor
Inside Student:Constructor
```



Klíčové slovo “super”

- Podtřída může explicitně zavolat konstruktor nejbližší nadtřídy pomocí klíčového slova *super.*

```
public Student(){
        super( "SomeName", "SomeAddress" );
        System.out.println("Inside
Student:Constructor");
}
```

Klíčové slovo “super”

- Několik věcí na které je třeba myslet při používání klíčové slova super pro volání konstruktoru nadtřídy:
  - *super()* musí být jako první příkaz konstruktoru.
  - *super()* může být použito pouze v konstruktoru.
  - Ve stejném konstruktoru nemůžeme použít *this()* a *super()* zároveň.

Klíčové slovo “super”

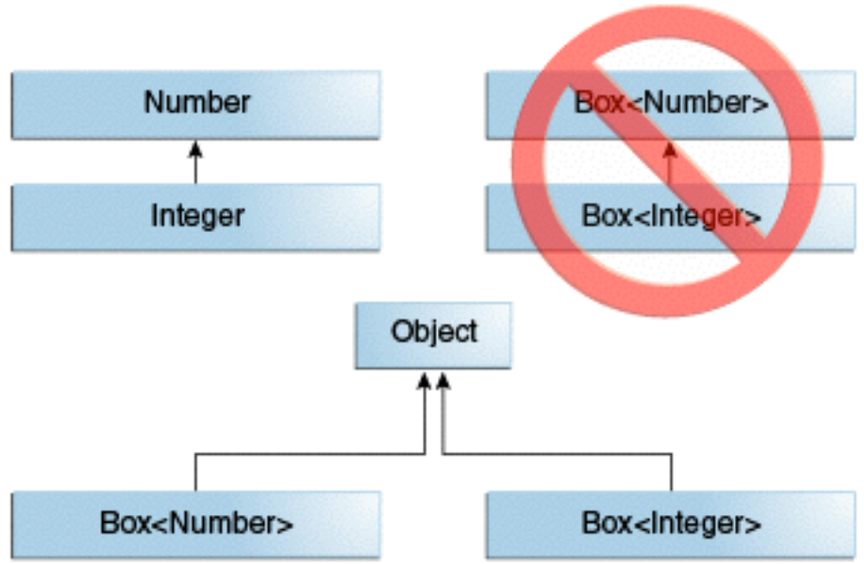
- Další použití je pokud chceme přistupovat ke členským proměnným nadtřídy.

```
public Student() {
     super.name = “somename”;
     super.address = “some address”;
     super.s();
}
```

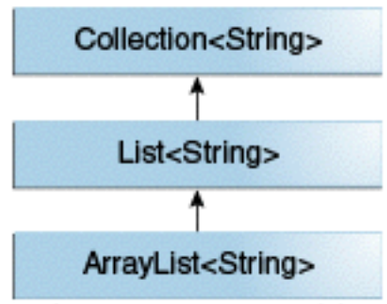
Generické typy

```
public interface Pair<K, V> {

    public K getKey();

    public V getValue();

}

public class OrderedPair<K, V> implements Pair<K, V> {

    private K key;

    private V value;

    public OrderedPair(K key, V value) {

     this.key = key;

     this.value = value;

    }

    public K getKey()    { return key; }

    public V getValue() { return value; }

}
```

Generické typy a podtypy



`Box<Integer>` is not a subtype of `Box<Number>` even though `Integer` is a subtype of `Number`.



A sample `Collections` hierarchy

Literatura

- http://fi.muny.cz/data/PB006/PB006_slides2009.pdf

- https://edux.feld.cvut.cz/courses/X36OBP/_media/lectures/02-subtyping.pdf