

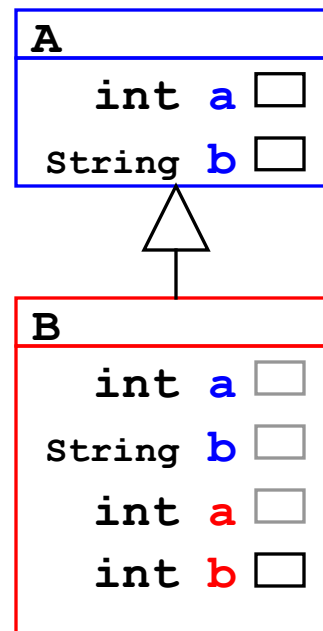
Atribúty a statické metódy

Atribúty (a statické metódy) metódy s rovnakým menom sú v podtriede nové!

Príklad:

```
class A
{ int    a;
  String b;
}
```

```
class B extends A
{ int  a;
  int  b;
}
```



B

int **a** ☐

String **b** ☐

int **a** ☐

int **b** ☐

ζ V objekte triedy **B** budú dva rôzne atribúty **a** a dva rôzne atribúty **b**:

ψ Zdedené atribúty triedy **A** (**a** a **b**) a

ψ Nové atribúty triedy **B** (**a** a **b**)

Príklad

```
B v1 = new B();
```

```
v1.a = 27;
```

```
v1.b = 2;
```

```
A v2 = v1;
```

```
System.out.println(v2.a);
```

```
// Vypise: 0 !!!!
```

```
v2.b = 25;
```

Ktorý atribút sa použije závisí od typu premennej!

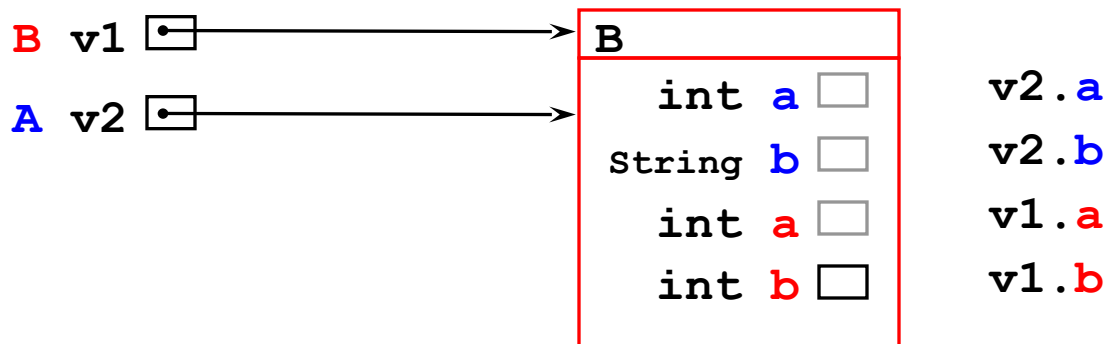
Zatienenie atribútov



ζ V podtriede **B** môžu byť definované atribúty s rovnakými menami ako v triede **A**! Typ týchto atribútov nemusí byť rovnaký, ako v rodičovskej triede!

Pri volaní premennou typu triedy **B** budú použité nové atribúty, pri volaní premennou typu triedy **A** pôvodné atribúty!

Prístup k atribútom závisí od typu premennej!



Pozor!



ζ Pri priradení premennej rodičovskej triedy môže prísť k „čudným“ výsledkom

```
B v1 = new B();  
A v2 = v1;  
// v1 a v2 ukazujú na ten  
// istý objekt!  
v1.a = 27;  
// v2.a == 0 !!!
```

Tip:
Pokúste sa predchádzať
použitiu rovnakých mien
atribútov rodičovskej
triedy a jej podtriedy

Zatienenie statických metód



To isté platí pre statické metódy!

Konštruktory a dedenie



- ζ Konštruktory triedy sa nededia!!
- ζ Každá trieda musí definovať vlastný konštruktor

Problém:



Niekedy by však bolo vhodné použiť konštruktor rodičovskej triedy.

Napr. pri inicializácii atribútov rodičovskej triedy pred inicializáciou vlastných atribútov.

Príklad:



```
class Person
{ int    userId;
  String name;
  String telNr;

  Person() { }

  Person(int id, String name, String telNr)
  { this.userId = id;
    this.name = name;
    this.telNr = tel; }

  ...
}
```

Príklad:



```
class Pracovnik extends Person
```

```
{
```

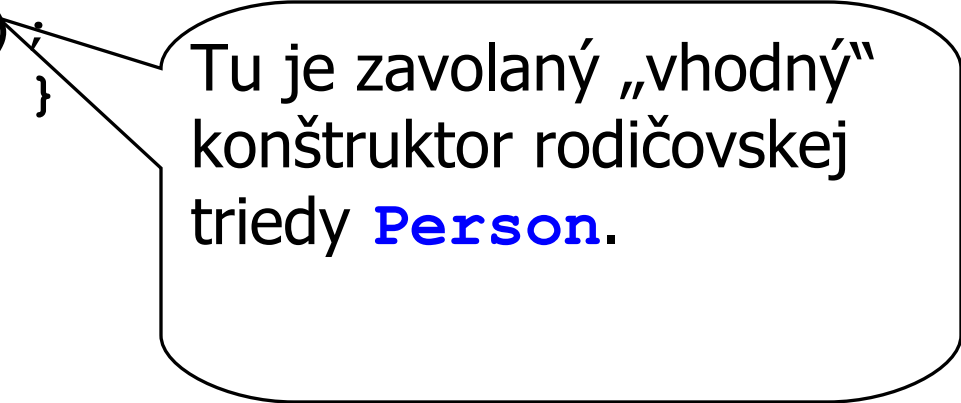
```
    String titul;
```

```
    Pracovnik(int id, String name,  
                String tel, String titul)
```

```
{    super(id,name,telNr);  
    this.titul = titul; }
```

```
    ...
```

```
}
```



Tu je zavolaný „vhodný“
konštruktor rodičovskej
triedy **Person**.

Volanie `super (. . .)`

- § Na začiatku konštruktora je možné pomocou
`super (. . .)`
zavolať konštruktor rodičovskej triedy!
- § Volanie `super (. . .)` je možné povoliť iba na začiatku konštruktora!
- § Pomocou `this (. . .)` je možné zavolať iný konštruktor tej istej triedy.

Implicitné volanie `super()`

- ζ Keď nie je explicitne zavolaný žiadny konštruktor rodičovskej triedy, je implicitne zavolaný štandardný konštruktor rodičovskej triedy:
`super()` ;

Príklad:

```
class Person
{ int    userId;
  String name;
  String telNr;
```

```
Person() { }
```

```
Person(int id, String name, String telNr)
{ this.userId = id;
  this.name = name;
  this.telNr = tel; }
```

```
...
```

```
}
```

Doteraz bola definícia štandardného konštruktora nutná, keďže triedy **Student** a **Pracovník** nevolali žiadny konštruktor svojej rodičovskej triedy **Person**; preto java implicitne volala štandardný konštruktor a preto tento musel byť definovaný. S nasledujúcou definíciou triedy **Pracovník** (podobne **Student**) môžeme definíciu štandardného konštruktora vynechať .

Príklad:



```
class Pracovnik extends Person
{
    String titul;

    Pracovnik(int id, String name,
               String telNr, String titul)
    { super(id,name,telNr);
      this.titul = titul; }

    ...
}
```

Poznámka:

super



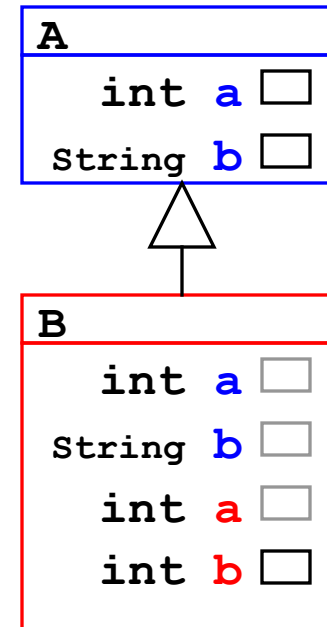
- ζ Pomocou **super** môžeme pristúpiť tiež k zatieneným atribútom alebo zatieneným statickým metódam rodičovskej triedy.
- ζ Pomocou **super** môžeme taktiež zavolať prepísané metódy rodičovskej triedy.

Príklad 1

```
class A
{ int    a;
  String b; }

class B extends A
{ int a;
  int b;

  B()
  { a = 1;
    b = 1;
    super.a = 1;
    super.b = ""; } }
```



Príklad 2



```
class Person
{ int    userId;
  String name;
  String telNr;

  Person(int id, String name, String telNr)
  { \* ako doteraz *\}

  public String toString()
  { return name + ": " + telNr}
}
```

Příklad 2

```
class Pracovník extends Person
{
```

```
    String titul;
```

```
    Pracovník(int id, String name,
               String telNt, String titul)
    { \* ako doteraz *\ }
```

```
    public String toString()
    { return super.toString() + ", " + titul; }
}
```

Tu je volaná
metóda
`toString`
rodičovskej
triedy `Person`!

Trieda Object



- ζ V Java existuje preddefinovaná trieda **Object**
- ζ **Každá** trieda dedí (implicitne) z triedy **Object**; to znamená najmä, že
 - ψ Objekt ľubovoľnej triedy môže byť vždy priradený k premennej typu **Object**
 - ψ Metódy, ktoré sú definované pre triedu **Object**, sú zdedené každou triedou (napr. `toString()`)

Metódy triedy Object



V triede Object je definovaná celá paleta metód (pozri

<https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html>)

ζ Pre nás sú zaujímavé najmä metódy:

```
public boolean equals(Object obj)
```

```
public String toString()
```

public boolean equals (Object obj)



- ζ Definuje rovnosť objektov
- ζ Táto metóda môže byť samozrejme v každej triede prepísaná (takto je možné definovať, čo znamená rovnosť priamo v programe)
- ζ Relácia **equals** by však mala byť reláciou ekvivalencie, teda mala by byť reflexívna, symetrická a tranzitívna)
- ζ Prednastavenie: identita

`public String toString()`

- ζ Definuje textovú podobu objektu (Objekt typu `String`)
- ζ Táto metóda je implicitne zavolaná, keď sa očakáva textová podoba objektu (`println`, sčítanie objektov `+`, ...); metódu je však samozrejme možné zavolať aj explicitne
- ζ Každá trieda ju môže prepísať, výsledok by mal predstavovať obsah objektu v textovej podobe

Typová konverzia - pretypovanie

- ζ Priradenie výrazu určitého typu na premennú typu predok triedy (v smere generalizácie) je vždy prípustné
- ζ Príklad:

```
Person pers = new Pracovnik(...);
```
- ζ Ako je to s opačným smerom (v smere špecializácie)?

Beispiel:

```
Person[] pers = new Person[3];
```

```
pers[0] = new Pracovnik(1, "Nový", "0905 ...", "Ing.");
```

```
pers[1] = new Student(2, "Prvák", "0907 ...", "0815");
```

```
pers[2] = new Person(5, "Externý", "0918 ...");
```

```
Pracovnik prac = pers[0];
```

Tento program sa nepodarí preložiť, to znamená, že priradenie nie je možné, aj keď `pers[0]` ukazuje na objekt triedy **Pracovnik**!

Riešenie:

Explicitné pretypovanie

```
Pracovnik prac = (Pracovnik) pers[0];
```

Táto operácia sa pokúsi skonvertovať typ výrazu na typ **Pracovnik** umzuwandeln; čo je možné oiba vtedy, ak výraz ukazuje na objekt triedy **Pracovnik** alebo špeciálnej triedy.

Ak nie je toto pretypovanie možné, program je prerušený chybou.

Explicitné pretypovanie



Pre každú definovanú triedu **Príklad** zmení operácia (**Príklad**) typ výrazu, ktorý nasleduje na danú triedu, ak je to možné.

- ζ Pretypovanie je teda možné, môže však spôsobiť chybu počas samotného behu programu

Overenie pretypovania

Aby sa zabránilo chybám pri behu spôsobeným neúspešným pretypovaním, je možné pomocou operácie `instanceof` overiť, či daný výraz je inštancia daného typu.

Príklad:

```
if (pers[0] instanceof Pracovnik)
{ Pracovnik prac = (Pracovnik) pers[0]; }
```

Kombinácia triedy `Object` s pretypovaním



- ζ Kombinácia triedy `Object` s pretypovaním umožňuje ľubovoľný objekt vložiť do nejakej dátovej štruktúry (pole, zret'azený zoznam)
- ζ Príslušná dátová štruktúra je potom definovaná pre prvky typu `Object`

Abstraktné triedy a metódy

Príklad: Triedy pre geometrické objekty

```
class Obdlznik
{ double xpos, ypos;

    double vyska, sirka;

    double plocha()
    { return vyska * sirka;

    }

    ...
}
```

```
class Kruh
{ double xpos, ypos;

    double r;

    double plocha()
    { return r * r * Math.PI;

    }

    ...
}
```

Rodičovská trieda:

Možná rodičovská trieda pre obĺžnik a kruh:

```
class GeomObjekt
{ double xpos, ypos;

    double plocha()
    { return ... ;
    }

    ...
}
```

Problém:
Čo dať sem???

Pozorovanie



- ζ Keď sa extrahujú niektoré spoločné črty tried do rodičovskej treidy, niekedy nie je možné určiť metódy **konkrétne** definovať!
- ζ Napriek tomu patrí takáto metóda k rodičovskej triede, keďže existuje v každej podtriede!

Riešenie:

Abstraktné metódy

Možná rodičovská trieda pre obĺžnik a kruh:

```
abstract class GeomObjekt  
{ double xpos, ypos;  
  
    abstract double plocha();  
  
    ...  
}
```

Ak má trieda aspoň jednu abstraktnú metódu, potom je trieda sama abstraktná.

Abstraktná metóda:
Metóda bez konkrétnej implementácie

Abstraktné metódy



- ζ **Abstraktné metódy** sú „rezervácie“ pre metódy, ktoré sú v podtriedach prepísané konkrétnymi metódami
- ζ Trieda s aspoň jednou abstraktnou metódou sa volá **abstraktná trieda**
- ζ Nie je možné vytvárať inštancie abstraktnej triedy.

Implementácia abstraktnej metódy

Príklad: Implementácia metódy plocha

```
class Obdlznik
    extends GeomObjekt
{
    double vyska, sirka;

    double plocha()
    { return sirka * vyska;

    }

    ...
}
```

```
class Kruh
    extends GeomObjekt
{
    double r;

    double plocha()
    { return r * r * Math.PI;

    }

    ...
}
```

Abstraktné triedy s konkrétnymi metódami

ζ V abstraktnej triede môžu byť definované aj konkrétne metódy.

Príklad:

```
abstract class GeomObjekt
{ double xpos, ypos;

    abstract double flaeche();

    void move(int xpos, int ypos)
    { this.xpos = xpos;
      this.ypos = ypos;
    }
}
```

Premenné pre abstraktnú triedu

ζ Aj keď nie je možné vytvárať inštancie abstraktnej triedy, je možné definovať premenné typu abstraktnej triedy.

Príklad:

```
GeomObjekt[] obj = new GeomObj[3];
```

```
obj[0] = new Obdlznik();
```

```
obj[1] = new Kruh();
```

```
obj[2] = new GeomObjekt();
```

Trieda je
abstraktná, preto
nie je možné
vytvárať jej objekty