Analýza a návrh informačných systémov II 2

objektovo orientovaný návrh

Peter Bednár

Metódy

Metódy

- Operácie nad členskými premennými objektov definované v danej triede
- Definujú sa podobne ako funkcie v C:

```
typ návratovej hodnoty názov(zoznam parametrov) {
    telo metódy
}
```

- Definícia parametrov: zoznam dvojíc typ parametra názov parametra oddelený `,` (čiarkou)
- Metóda môže byť:
  - bez parametrov (prázdne zátvorky)
  - bez návratovej hodnoty: typ `void`

Metódy – príklad 1

```
class Motor {

    String typ;
    ... // ostatné členské premenné

    void nastartuj() {
        ...
    }

    void nastavPlyn(float plyn) {
        ...
    }

    float aktualnyVykon() {
        ...
    }
}
```

Motor
typ: String objem: float maxVykon: float otacky: int teplota: float
nastartuj(): void nastavPlyn(plyn: float): void aktualnyVykon(): float

Metódy a členské premenné

- V rámci metódy môžeme priamo pristupovať k členským premenným triedy

```
int otacky;
...
void nastartuj() {
    otacky = 2000;
}

void nastavPlyn(float plyn) {
    if (otacky == 0) {
        return;
    }
    otacky = (int)(50 * plyn + 2000);
}
```

Nastavíme hodnotu členskej premennej

Príkaz return môžeme použiť kedykoľvek na návrat z metódy

Metódy – príklad 2

```
class DruhyProgram {

    public static void main(String args[]) {
        Motor motor1 = new Motor();
        Motor motor2 = new Motor();

        motor1.nastartuj();
        motor2.nastartuj();
        motor2.nastavPlyn(100); // nastav plyn na 100%

        System.out.println("motor1 otáčky: " + motor1.otacky);
        // 2000 (nastavené v metóde nastartuj)
        System.out.println("motor2 otáčky: " + motor2.otacky);
        // 7000 = 50 * 100 + 2000 (nastavené v metóde nastavPlyn)
    }
}
```

Volanie metód

Lokálne premenné

- V tele metódy môžeme definovať ľubovoľné lokálne premenné

```
float aktualnyVykon() {
    // vykon je lokálna premenná
    float vykon = 100 * (otacky / 5000)
    if (vykon > maxVykon) {
        return maxVykon;
    }
    return vykon;
}
```

Členské premenné

Návratovú hodnotu
vrátime v príkaze return

Prekrývanie členských premenných (1)

- Lokálne premenné, alebo parametre metód sa môžu volať rovnako ako členské premenné
- Pre rozlíšenie je možné vždy pristupovať k členským premenným cez kľúčové slovo `this`

```
int otacky;      // členská premenná
...
void nastartuj() {
    int otacky;
    otacky = 2000;
    this.otacky = 1000;
}
```

Vytvoríme lokálnu premennú s rovnakým menom, ako má členská premenná - prekrytie

Zmeníme iba lokálnu premennú

Ak chceme zmeniť prekrytú členskú premennú, použijeme kľúčové slovo this

Prekrývanie členských premenných (2)

- Podobne je to pri parametroch metód

```
int otacky;     // členská premenná
...
void nastavOtacky(int otacky) {
    this.otacky = otacky;
}
```

Nastavíme členskú premennú podľa parametra metódy

- **POZOR** na chyby:

```
void nastavOtacky(int otacky) {
    otacky = otacky;
}
```

Chýba this, len sme nastavili lokálny parameter na tú istú hodnou, členská premenná sa po volaní metódy nezmení!

Preťažovanie metód (1)

- V jednej triede je možné definovať viacero metód s tým istým menom – musia sa odlišovať typom a/alebo počtom parametrov (na názvoch parametrov nezáleží)

```
void nastartuj() {
    this.otacky = 2000;
}

void nastartuj(int otacky) {
    this.otacky = otacky;
}
```

```
motor1.nastartuj();
// motor1.otacky = 2000

motor1.nastartuj(3000);
// motor1.otacky = 3000
```

Preťažovanie metód (2)

```
void nastartuj(float plyn) {
    nastartuj(2000, plyn);
}
```

```
motor1.nastartuj(50.0);
// motor1.otacky = 2000 + 2500
// motor1.teplota = 60
```

Preťažené metódy sa môžu navzájom volať

```
void nastartuj(int otacky,
               float plyn) {
    nastartuj(otacky, plyn, 60);
}
```

```
motor1.nastartuj(3000, 50.0);
// motor1.otacky = 2000 + 2500
// motor1.teplota = 60
```

```
void nastartuj(int otacky,
               float plyn,
               int teplota) {
    this.otacky = otacky;
    this.teplota = teplota
    nastavPlyn(plyn);
}
```

```
motor1.nastartuj(2000, 10.0, 50);
// motor1.otacky = 2000 + 500
// motor1.teplota = 50
```

Konštruktory

Konštruktory (1)

- Špeciálne metódy určená na inicializáciu objektov po vytvorení
- Nemajú definovaný typ návratovej hodnoty a majú rovnaké meno ako trieda

```
class Motor {

    String typ;
    float objem;
    ...

    Motor() {
        typ = "EcoBoost";
        objem = 1.0;
    }
    ...
}
```

Motor
typ: String objem: float maxVykon: float otacky: int teplota: float
Motor() nastartuj(): void nastavPlyn(plyn: float): void aktualnyVykon(): float

Konštruktory (2)

- Konštruktory môžu mať parametre

```
class Motor {
    ...
    Motor(String typ, float objem, float maxVykon) {
        this.typ = typ;
        this.objem = objem;
        this.maxVykon = maxVykon;
    }
    ...
}
```

- Vytvorenie objektu:

```
Motor ecoboost1 = new Motor("EcoBoost", 1.0, 92.0);
// inicializuje sa typ, objem a maxVykon, ostatné majú
// prednastavené hodnoty 0
```

Konštruktory (3)

- Konštruktory je možné preťažovať podobne ako metódy

```
Motor(String typ, float objem) {
    this(typ, objem, objem * 70);
}

Motor(String typ, float objem, float maxVykon) {
    this.typ = typ;
    this.objem = objem;
    this.maxVykon = maxVykon;
}
...

Motor motor1 = new Motor("Zetec", 1.0);
Motor motor2 = new Motor("EcoBoost", 1.2, 79);
```

Aj preťažené konštruktory sa môžu navzájom volať, odkazuje sa na nich cez kľúčové slovo this

Statické metódy a statické premenné

Statické metódy a premenné (1)

- Niekedy chceme naprogramovať kód bez toho aby sme vytvorili nejaký objekt
  - Napr. hlavná funkcia programu `main`
- Môžeme definovať statickú metódu pomocou slova **`static`**
- Podobne je možné definovať aj statické premenné
  - Často sa používajú na definovanie konštantných hodnôt, ktoré sa zdieľajú pre všetky objekty danej triedy

Statické metódy a premenné (2)

- V normálnych metódach môžeme používať aj statické metódy a členské premenné
- V statických metódach iba statické (keďže neexistuje žiaden objekt, nemôžeme použiť `this`)

Statické metódy a premenné – príklad 1

```
class Motor {

    static float MAX_TEPLOTA = 150;

    int otacky;
    float teplota;
    ...

    static float vypocitajTeplotu(int otacky) {
        // teplota a otacky sú lokálne premenné, nie členské
        float teplota = (otacky - 2000) / 50.0 + 20;
        if (teplota > MAX_TEPLOTA) {
            return MAX_TEPLOTA;
        }
        return teplota;
    }
}
```

Motor
MAX_TEPLOTA: float = 150 otacky: int teplota: float
vypocitajTeplotu(otacky: int): float

Zo statickej metódy sa môžeme odkazovať na statické premenné

Statické metódy a premenné – príklad 2

```
class Motor {

    static float MAX_TEPLOTA = 150;
    ...

    void nastavOtacky(int otacky) {
        this.otacky = otacky;
        this.teplota = vypocitajTeplotu(otacky);
        if (this.teplota == MAX_TEPLOTA) {
            System.out.println("prehrievanie motora");
        }
    }

    static float vypocitajTeplotu(int otacky) {
        ...
    }
}
```

V nestatickej metóde môžeme volať statické metódy a odkazovať sa na statické premenné

Statické metódy a premenné – príklad 3

- V inej triede môžeme volať statické metódy alebo sa odkazovať na statické premenné cez Názov triedy.
- Pomocou názvu triedy môžeme rozlíšiť statickú premennú a v tej istej triede, ak je prekrytá lokálnou, alebo členskou premennou

```
class TretiProgram {

    public static void main(String args[]) {
        int teplota = Motor.vypocitajTeplotu(5000);
        System.out.println("maximálna teplota: " +
                           Motor.MAX_TEPLOTA);
    }
}
```

Premenné zhrnutie

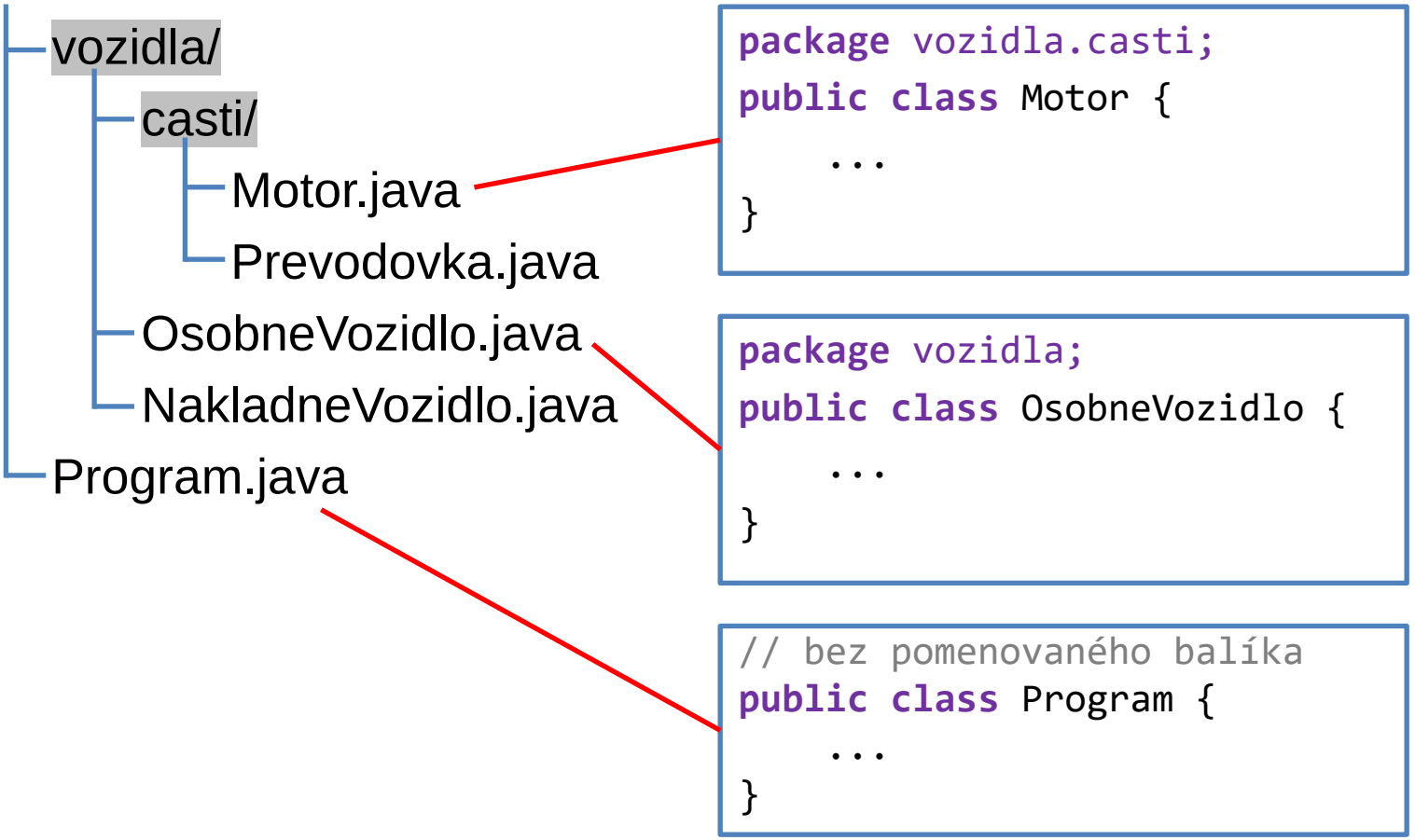
Lokálna	**Členská**	**Statická**
	„Patrí“ objektu	„Patrí“ triede
Definuje sa v tele metódy	Definuje sa v triede	Definuje sa v triede
Je prístupná iba v danej metóde, kde je definovaná	Je prístupná v nestatických metódach danej triedy	Je prístupná v statických aj nestatických metódach danej triedy
	Je prístupná cez `this.`	Je prístupná cez Názov triedy`.`
Môže prekryť členskú alebo statickú	Môže prekryť statickú	

Balíky v Jave

Balíky (1)

- Umožňujú rozdeliť zložitejšie programy na moduly
- Jeden balík zoskupuje viacero súvisiacich tried
- Hierarchická štruktúra:
  - balík môže mať ďalšie pod-balíky
- Trieda sa do balíka zaradí kľúčovým slovom **package**
  - musí byť ako prvý príkaz pred definíciou triedy
- Na disku sú balíky reprezentované ako adresáre/podadresáre v ktorých sú zdrojové .java súbory tried

Balíky – príklad (1)



Balíky (2)

- Ak chceme používať triedy z iného balíka, musíme ich importovať kľúčovým slovom **`import`**
- Pre importovanie jednej triedy:

```
import cesta.Trieda;
```

- Pre importovanie všetkých tried z daného balíka:

```
import cesta.*;
```

- Môže byť uvedených viacero importov
- Musia byť uvedené za definíciou **`package`** a pred definíciou triedy

Balíky – príklad 2

```
package vozidla;

import vozidla.casti.*;

public class OsobneVozidlo {
    ...
}
```

Pre triedu `OsobneVozidlo` importujeme všetky triedy z balíka `vozidla.casti` (`Motor`, `Prevodovka`)

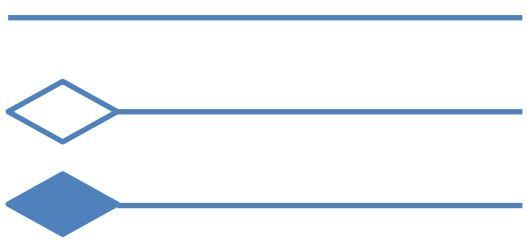
```
import vozidla.*;
import vozidla.casti.Motor;

public class Program {
    ...
}
```

Pre triedu `Program` všetky triedy z balíka vozidla (`OsobneVozidlo`, `NakladneVozidlo`) a triedu `Motor` z balíka `vozidla.casti`

Relácie medzi objektami

Relácie medzi objektami v UML

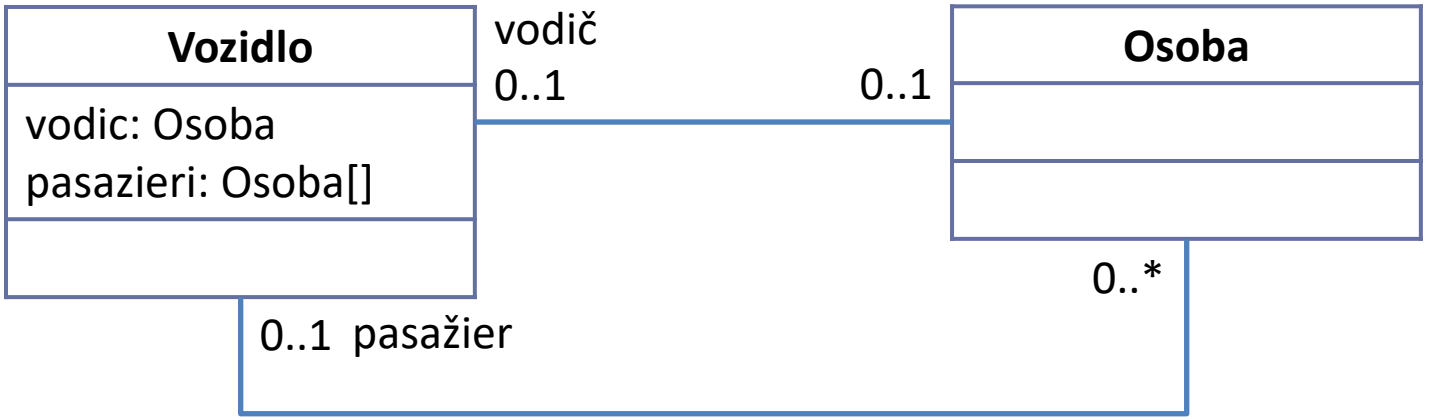
- Popisujú vzťahy medzi objektami
- Základné typy relácií:
  - Asociácia
  - Agregácia
  - Kompozícia



- Označenie kardinality:
  - 1 – práve jeden
  - 0..1 – 0, alebo jeden
  - 0..* – neobmedzený počet (aj žiaden)
  - *n*..*m* – od *n* do *m* (*n* a *m* sú čísla)

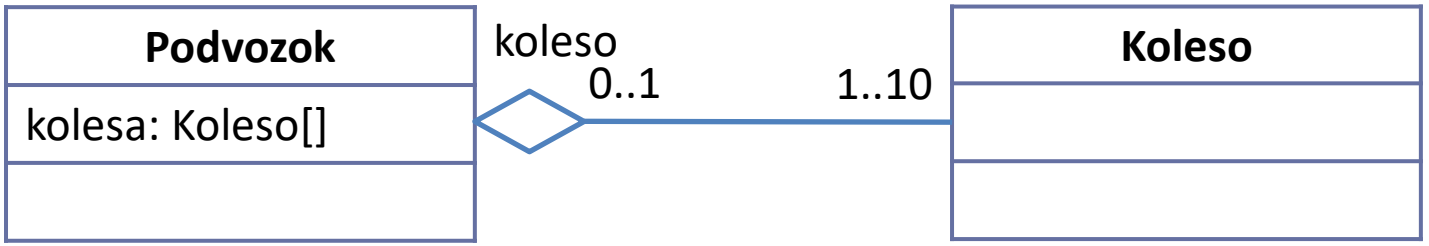
Asociácia - príklad

- Základný vzťah medzi dvoma typmi objektov
  - Jedno vozidlo môže mať max. jedného vodiča a neobmedzený počet pasažierov (môže byť aj prázdne)
  - Jedna osoba môže byť vodičom, alebo pasažierom max. v jednom vozidle



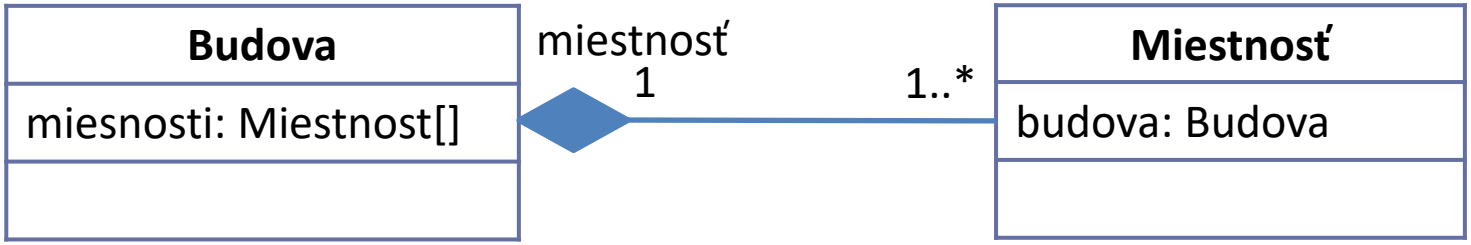
Agregácia - príklad

- Vzťah typu celok/časť, požíva sa ak časť môže existovať samostatne
  - Podvozok musí mať aspoň jedno a max. 10 kolies
  - Koleso musí byť namontované na max. jednom podvozku
  - Koleso môže existovať samostatne a môžeme ho preniesť na iný podvozok



Kompozícia - príklad

- Vzťah typu celok/časť, požíva sa ak časť nemôže existovať samostatne
  - Budova musí mať aspoň jednu miestnosť
  - Miestnosť patrí práve do jednej budovy (nemôže existovať bez budovy)



Zhrnutie

- Metódy
- Konštruktory
- Preťažovanie metód a konštruktorov
- Statické metódy a premenné
- Balíky
- Vzťahy medzi objektami
  - Asociácia
  - Agregácia
  - Kompozícia