21 Úvod do objektovo orientovaného programovania (OOP)

21.1 Charakteristika objektovo orientovaného programovania

Java je **objektovo orientovaný** programovací jazyk. Na rozdiel od jazyka C++ (C++ nie je typický objektovo orientovaný jazyk, hovoríme, že podporuje OOP) v Jave nevieme vytvoriť žiadny program, v ktorom by sme nevyužili vlastnosti OOP.

Princípy OOP boli rozpracované už v 70. rokoch 20. storočia, k širšiemu použitiu OOP pri vývoji softvéru však došlo až ku koncu storočia.

Potreba využiť pri vývoji softvéru iné postupy a technológie, také ktoré by sa viac približovali reálnemu svetu a jeho objektom (programy vždy riešia problémy reálneho sveta), vzišla z neschopnosti dosiahnuť to procedurálnym programovaním.

V procedurálnych jazykoch (Algol, Cobol ale aj C atď.) sa problémy riešia tak, že sa „rozložia“ na menšie časti a pre každú z častí sa vytvorí príslušný podprogram (procedúru, funkcia, metóda), ktorá by daný čiastkový problém vyriešila.

Programy písané týmto štýlom sa však pri určitej veľkosti kódu začínajú veľmi ťažko udržiavať a nie je možné ich rozšíriť. Vývojári postupne dospeli k znepokojujúcemu záveru: procedurálne programy sú pomalé, plné chýb, nespoľahlivé a nákladné. Ich údržba je neúnosná a takmer nemožná.

Východiskom sa ukázalo byť OOP, v ktorom sa neuvažuje v pojmoch údajové štruktúry a manipulujúce funkcie, ale uvažuje sa v pojmoch **objektov**, to znamená vecí reálneho sveta.

Skutočný svet je tvorený osobami, automobilmi, zvieratami, stromami, budovami atď. Každý objekt je niečím charakteristický: osoba je vysoká, auto je modré, zviera má štyri nohy atď. Okrem toho sa každý objekt vyznačuje určitým chovaním: osoba kráča, automobil zrýchľuje, zviera spí, strom rastie.

V OOP môžeme každému objektu reálneho sveta prisúdiť určité charakteristiky a správanie, a spolu ich zapuzdriť do jedného celku. Jazyk Java na to využíva objektový typ **trieda (class).**

Medzi najdôležitejšie vlastnosti OOP patria:

- Abstrakcia
- Zapuzdrenie a ukrytie údajov
- Polymorfizmus
- Dedičnosť
- Znovu použiteľný kód

Súčasné moderné programy, ako sú grafické rozhrania operačných systémov a používateľských aplikácií, internetové aplikácie a množstvo nových technológií, sa vyznačujú takou komplexnosťou, že bez OOP by ich nebolo možné prakticky vytvoriť.

21.2 Trieda (class)

Trieda je najdôležitejším pojmom OOP. Je to v podstate plán, podľa ktorého sa vytvárajú objekty. Od začiatku programovania v Jave sme vždy používali aspoň jednu triedu - do nej sme umiestňovali metódu main(). Keď sme začali vytvárať a používať vlastné metódy, tieto sme umiestnili do samostatných tried. Okrem metód však trieda môže obsahovať aj premenné, ktoré nie sú súčasťou metód. Takýmto premenným hovoríme **členské premenné** (member variables). Členským premenným hovoríme aj **údajové zložky** alebo tiež **atribúty** a je v nich uložený **stav objektu** (bude vysvetlené neskôr).

Metódy umožňujú manipulovať s členskými premennými a meniť tak stav objektu. Metódy vlastne popisujú **schopnosti** objektu - čo dokáže objekt robiť.

Samotná trieda je len objektový typ a nie je jej pridelená žiadna pamäť. Je to len akási šablóna, pomocou ktorej, bude neskôr vytvorený objekt.

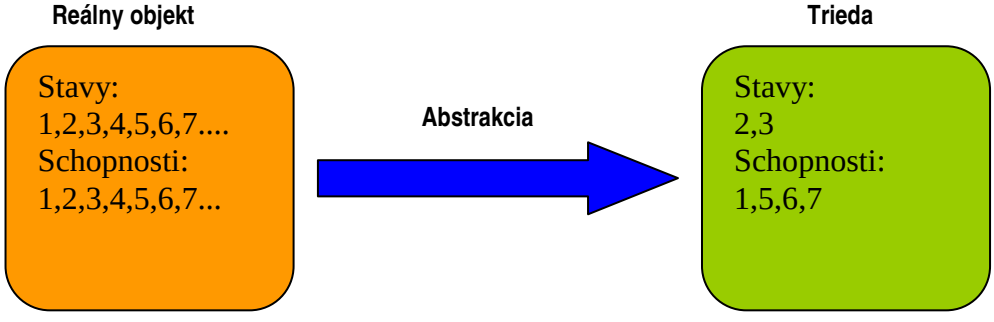
21.3 Objekt (object)

Objekt je vlastne údajový prvok, vytvorený podľa triedy. Hovoríme mu aj **inštancia** triedy. Pojmy objekt a inštancia sa voľne zamieňajú.

Podľa vzoru jednej triedy je možno vytvoriť ľubovoľný počet objektov (inštancií), pričom každý objekt bude mať rovnaké metódy. Objekty jednej triedy budú mať tiež rovnaké členské premenné, ale v každom objekte môžu tieto premenné obsahovať iné hodnoty. Pre každý objekt sa tak vytvoria v pamäti vlastné premenné.

21.3 Abstrakcia

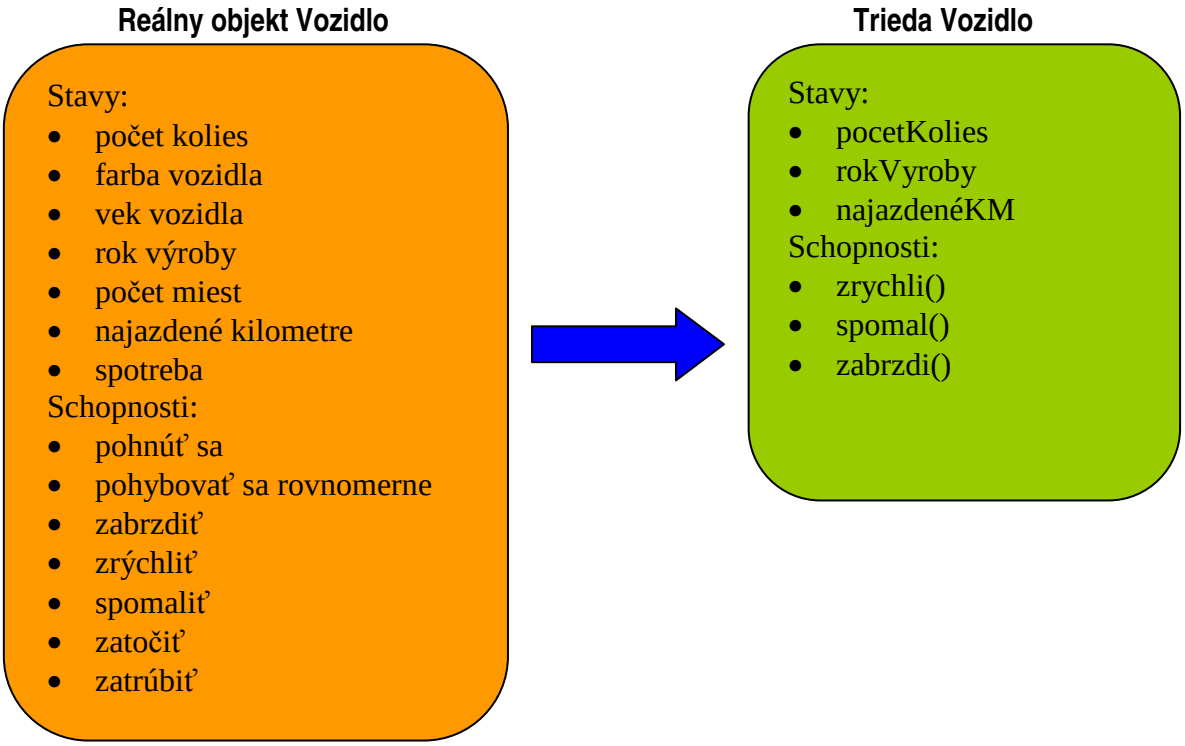
Ako už bolo skôr uvedené, objekty reálneho sveta sa vyznačujú tým, že sú v nejakom stave a sú niečoho schopné – nejako sa správajú, niečo vedia. Ak chceme naprogramovať stav a správanie reálneho objektu, tak zo všetkých stavov a schopností objektu vyberieme len tie, ktoré sú pre riešenie danej úlohy nevyhnutné. Preto do triedy, ktorá má popisovať daný reálny objekt, zaradíme len tie stavy (do premenných) a tie schopnosti (vyjadrené metódami), ktoré potrebujeme pri riešení daného problému. Hovoríme o **abstrakcii**. Abstrakcia je teda základným krokom k reprezentácií stavov a schopností reálneho objektu v triede.



V predchádzajúcej schéme má reálny objekt stavy vyjadrené číslami a schopnosti, tiež vyjadrené číslami. Pri abstrahovaní sme zo všetkých stavov vybrali iba stavy 2 a 3. Zo všetkých schopností iba schopnosti 1,5,6 a 7.

21.4 Návrh konkrétnej triedy

Na konkrétnom príklade si teraz ukážeme návrh triedy, ktorá bude reprezentovať reálny objekt s názvom Vozidlo. Vieme, že vozidlom môže byť napríklad automobil, motocykel a vlastne všetko, čo nás odvezie. U spomenutých vozidiel môžeme evidovať množstvo stavov, napríklad: počet kolies, farbu, vek, rok výroby, počet miest, počet prejdených kilometrov, spotrebu (ak má vozidlo motor) a pod. Tiež môžeme evidovať množstvo schopností, napríklad: schopnosť pohnúť sa, pohybovať sa rovnomerne, zabrzdiť, zrýchliť, zatočiť, zatrúbiť atď. My si však zo stavov a schopností vyberieme iba niektoré, tak ako to ukazuje nasledovná schéma:



V uvedenej schéme obsahuje trieda Vozidlo už konkrétne názvy členských premenných a členských metód.

V ďalšej časti si už konečne ukážeme ako by vyzerala trieda Vozidlo v programe. Ďalej si ukážeme vytvorenie samotného objektu a naplnenie členských premenných hodnotami a ich výpis. V triede nebudeme implementovať žiadne metódy, chceme zatiaľ pracovať iba z členskými premennými.

21.5 Aplikácia EvidenciaVozidiel

Na konkrétnej aplikácii s názvom **EvidenciaVozidiel** si ukážeme prácu s členskými premennými, konkrétne zápis a čítanie hodnôt.

Triedu Vozidlo vytvoríme v samostatnom súbore:

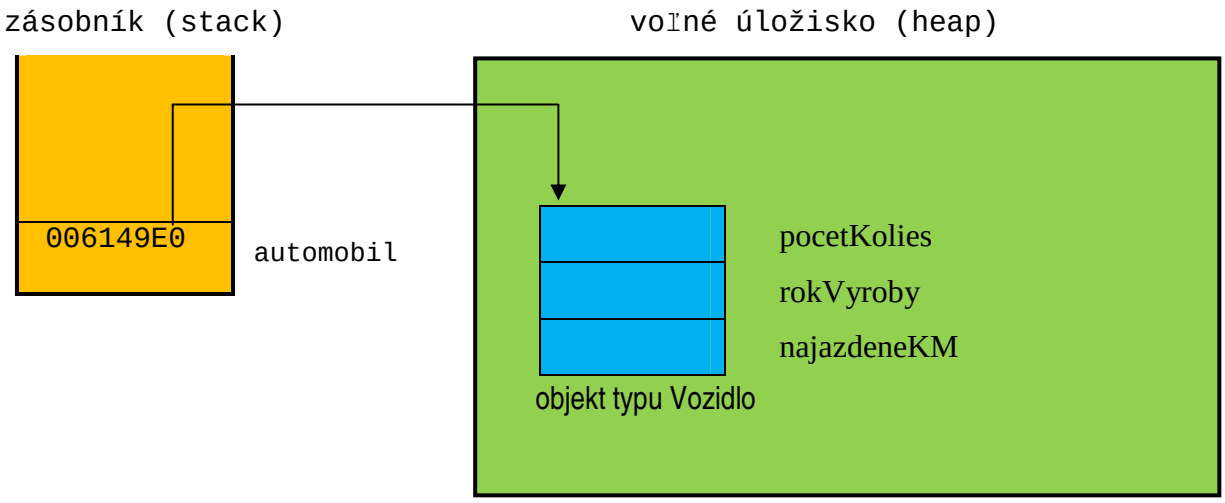
```
public class Vozidlo        //public sa musí uviesť, lebo trieda je v samostatnom
{                           //súbore
 int pocetKolies;
 int rokVyroby;             } členské premenné – nie sú súčasťou žiadnej metódy
 double najazdeneKM;

 //V tejto časti by boli implementácie metód:
 //zrychli()
 //spomal()
 //zabrzdi()
}
```

Aj napriek tomu, že trieda obsahuje tri členské premenné, nezaberá v pamäti žiadne miesto. Až vytvorením objektu dôjde k alokácii pamäte.

V metóde main() vytvoríme inštanciu, teda objekt triedy *Vozidlo* a nazveme ho *automobil*.

```
...
Vozidlo automobil = new Vozidlo();
...
```



Objekt *automobil* má tri členské premenné. Pristupujeme k nim cez operátor bodka (.). Tento prístup je priamy, lebo všetky premenné sú **verejné (public)**. Ak ich neinicializujeme my, tak kompilátor im priradí hodnotu 0 resp. 0.0. (typu boolean hodnotu false).

```
...
automobil.pocetKolies = 4;
automobil.rokVyroby = conIN.nextInt();
automobil.najazdeneKM = 56820.8;
...
```

V metóde main() si vytvoríme si ďalší objekt triedy *Vozidlo* a nazveme ho *motocykel*. Do premenných zapíšeme hodnoty.

```
...
Vozidlo motocykel = new Vozidlo();
motocykel.pocetKolies = 2;
motocykel.rokVyroby = conIN.nextInt();
motocykel.najazdeneKM = 512.5;

...
```

Ako vidíme, pomocou jednej triedy sme vytvorili dva samostatné objekty, pričom každý z nich má vlastné premenné.

21.6 Cvičenie

1. Vytvorte aplikáciu s názvom **EvidenciaZiakov**. Vytvorte triedu s názvom Osoba. Deklarujte v nej 5 členských premenných, kvôli evidencii základných údajov, napr. meno, rok narodenia, výška atď.. Vytvorte 3 objekty, zapíšte do premenných hodnoty a dajte vypísať.
2. Vytvorte aplikáciu s názvom **SkladPC.** Vytvorte triedu s premennými na evidenciu počítačov a 3 objekty. Zapíšte údaje a dajte vypísať.

21.7 Otázky

1. Uveďte hlavné výhody OOP oproti procedurálnemu programovaniu.
2. Uveďte najdôležitejšie vlastnosti OOP.
3. Charakterizujte pojem trieda.
4. Charakterizujte pojem objekt.
5. Na konkrétnom príklade uveďte, čo rozumieme pod abstrakciou.