23 Konštruktory

23.1 Konštrukcia objektov.

Prvý objekt, ktorý sme v Jave vedome vytvorili, bol objekt *konzolovyVstup*, v kapitole 8.1. Neskôr sme vytvárali reťazce, čo sú tiež objekty, potom polia až sme prešli k samotnému objektovo orientovanému programovaniu, kde sa vytváraniu objektov nevyhneme už v žiadnej aplikácii.

Objekty teda vytvárať vieme, ale čo sa deje v skutočnosti v pozadí pri konštruovaní samotného objektu, tomu sme sa venovali len veľmi okrajovo. Teraz nadišiel čas , aby sme si vysvetlili aj tento, veľmi dôležitý aspekt OOP.

Veľmi dobre vieme, že ak vytvárame primitívnu premennú, ktorá nie je členskou premennou triedy, tak Java nám s ňou dovolí pracovať až vtedy, keď ju nejakým spôsobom inicializujeme – či už vstupom z klávesnice, alebo priradením hodnoty.

Toto isté platí aj pre referenčné premenné – aj tie ak chceme používať, musia na niečo odkazovať.

Zaujímavá situácia je v prípade členských premenných tried. Pri nich sme ako fakt prijali to, že ich nemusíme inicializovať, lebo pri vytvorení objektu získajú konkrétnu hodnotu – číselné hodnotu 0 resp. 0.0, znakové Unicodeovú hodnotu 0, boolovské hodnotu false a referenčné hodnotu null. Prečo práve tieto hodnoty a nie iné. Má to na „svedomí“ špeciálna metóda, ktorá sa volá **konštruktor**.

23.2 Konštruktor.

Konštruktor je metóda a ako už samotný jej názov hovorí, niečo konštruuje. Tým niečím nie nič iné ako objekt. Takže úlohou konštruktora je skonštruovať objekt, presnejšie vytvoriť premenné a inicializovať ich na konkrétne hodnoty. Kde sa však táto metóda vzala, nikde sme ju totiž nevytvorili. V tomto prípade ju vytvoril samotný kompilátor.

Takže v zápise

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

je `Vozidlo()` názov konštruktora.

Konštruktoru, ktorý je vytvorený kompilátorom hovoríme **implicitný konštruktor**. Implicitný konštruktor vždy nastaví hodnoty členských premenných tak, ako bolo spomenuté vyššie - číselné na hodnotu 0 resp. 0.0, znakové na Unicodeovú hodnotu 0, boolovské na hodnotu false a referenčné na hodnotu null. Oveľa častejšie ale budeme vytvárať svoje vlastné konštruktory. V tom prípade **kompilátor implicitný konštruktor nevytvorí!**

Keďže odteraz budeme vytvárať svoje vlastné konštruktory, zhrňme si základné charakteristiky konštruktora:

- konštruktor je metóda, ktorá inicializuje členské premenné
- je volaná v momente vytvárania objektu
- má rovnaký názov, ako je názov triedy
- nikdy nič neodovzdáva; pozor v hlavičke implementácie nepíšeme void!
- nemusí mať alebo môže mať formálne parametre
- ako každá metóda, dá sa preťažovať
- ak nevytvoríme sami konštruktor, kompilátor vytvorí implicitný konštruktor
- konštruktor nemusí inicializovať všetky členské premenné

Teraz si ukážeme vytvorenie a použitie konštruktorov. Opäť sa vrátime k triede *Vozidlo* a doplníme do nej svoje vlastné konštruktory:

```
public class Vozidlo
{
 int pocetKolies;
 int rokVyroby;
 double najazdeneKM;

 //Konštruktory
 Vozidlo()                  //bezparametrický konštruktor vhodný na vytvorenie
 {                          //automobilu
    pocetKolies = 4;
    rokVyroby = 1990;
    najazdeneKM = 0;
 }

 Vozidlo(int pocKolies,int rVyroby) //konštruktor s dvomi parametrami
 {
    pocetKolies = pocKolies;
    rokVyroby = rVyroby;
    najazdeneKM = 0;
 }
 //Ďalej by mohli nasledovať ďalšie konštruktory, za nimi
 //prístupové metódy atď.
}
```

V metóde main() teraz vytvoríme objekty využitím nami navrhnutých konštruktorov:

```
...
Vozidlo automobil = new Vozidlo();
Vozidlo motocykel = new Vozidlo(2,2011);
Vozidlo nakladnyAutomobil = new Vozidlo(8,2003);
...
```

V prvom prípade sme použili bezparametrický konštruktor, v druhom a treťom prípade konštruktor s dvomi parametrami. Pozor, keďže kompilátor nevytvoril implicitný konštruktor, objekt *automobil* bude mať taký počet kolies aký sme zadali v tele vlastného bezparametrického konštruktora, teda 4 (mohli sme zadať akýkoľvek iný počet, všetko závisí od konkrétneho problému, ktorý riešime).

V predchádzajúcom príklade sme si súčasne predviedli aj preťaženie konštruktorov, keďže sme ich vytvorili dva. Doplniť by sme mohli aj ďalšie, len by sa museli líšiť počtom a typom formálnych parametrov.

Hodnoty členských premenných vytvorených objektov môžeme ďalej meniť už len pomocou prístupových metód, nie konštruktorom – ten, ako už bolo uvedené, sa volá len pri vytváraní daného objektu!

23.3 Deštrukcia objektov.

Deštrukciou objektu uvoľňujeme dynamicky alokovanú pamäť. A ako už vieme uvoľnenie dynamicky alokovanej pamäte má na starosti garbage collector. Keďže garbage collector automatický zbiera neplatné objekty, t.j. také, na ktoré už neexistuje odkaz, nemusíme sa o deštrukciu objektov vôbec starať.

Sami nevieme ovplyvniť to, kedy má garbage collector zasiahnuť a objekt z pamäte odstrániť. Niekedy to neurobí počas celého behu programu a to aj napriek tomu, že sú v pamäti neplatné objekty.

Sú však situácie, kedy by sme chceli vedieť, či daný objekt ukončil svoju existenciu korektným spôsobom. Napríklad, či došlo k uzatvoreniu súboru, ktorého vlastníkom je deštruovaný objekt.
V takomto prípade môže poslúžiť špeciálna metóda s názvom *finalize()*. Metódu *finalize()* volá virtuálny stroj Javy vždy, keď sa chystá deštruovať objekt danej triedy. My sami ale nevieme zabezpečiť kedy presne a či vôbec sa metóda *finalize()* vykoná. Z tohto dôvodu ju používať nebudeme a spoľahneme sa na to, že virtuálny stroj zabezpečí všetko potrebné k deštrukcii objektu, a že ho prevedie korektne.

23.4 Rezervované slovo this.

Jednoducho povedané **this** je odkaz na objekt. My ho budeme využívať v situáciách, keď budeme potrebovať pracovať s členskými premennými objektov v metódach, ktoré budú mať deklarované premenné s tými istými názvami ako členské premenné objektu. Vyhneme sa tak vymýšľaniu ďalších a ďalších mien premenných.

Všetko si vysvetlíme opäť na triede *Vozidlo*. Trieda *Vozidlo* bude obsahovať tri členské premenné. Vytvoríme jeden parametrický konštruktor a jednu prístupovú metódu set. V oboch metódach použijeme formálne parametre s rovnakými názvami aké majú členské premenné. Aby kompilátor vedel navzájom rozlíšiť premenné metód od členských premenných triedy, pred menom každej členskej premennej použijeme rezervované slovo this:

```
public class Vozidlo
{
 int pocetKolies;
 int rokVyroby;
 double najazdeneKM;

 //Konštruktor
 Vozidlo(int pocetKolies,int rokVyroby,double najazdeneKM)
 {
    this.pocetKolies = pocetKolies;
    this.rokVyroby = rokVyroby;
    this.najazdeneKM = najazdeneKM;
 }

 //Prístupová metóda
 void setPocetKolies(int pocetKolies)
 {
    this.pocetKolies = pocetKolies;
 }
 ...
}
```

23.5 Aplikácia VzdialenostBodov

Vytvoríme aplikáciu s názvom **VzdialenostBodov**. V nej deklarujeme triedu s názvom **Bod** s dvomi súkromnými členskými premennými *suradnicaX*, *suradnicaY.* Vytvoríme dva konštruktory. Jeden bezparametrický, ktorý nám umožní vytvoriť bod v počiatku súradnicovej sústavy a druhý, ktorý nám vytvorí bod v ľubovoľnej časti súradnicovej sústavy. Potom vytvoríme dve inštancie triedy Bod a vypočítame vzdialenosť oboch bodov.

23.6 Aplikácia PravouhlyTrojuholnik

Vytvoríme aplikáciu s názvom **PravouhlyTrojuholnik**. V nej deklarujeme triedu s názvom **Trojuholnik** s tromi súkromnými členskými premennými *odvesna1*, *odvesna2 a prepona.* Vytvoríme konštruktor s dvomi parametrami, ktoré budú predstavovať dĺžky odvesien. V konštruktore sa dopočíta veľkosť prepony, tak, že si konštruktor zavolá metódu dlzkaPrepony(). Potom vytvoríme niekoľko inštancií triedy Trojuholník a pre každú vypočítame obsah trojuholníka.

23.7 Cvičenie

1. Vytvorte aplikáciu s názvom **Trojuholníky**. Deklarujte triedu s názvom **Trojuholnik** s trojparametrickým konštruktorom, ktorého parametre budú dĺžky jednotlivých strán trojuholníka. Konštruktor zistí, či zadané strany sú stranami trojuholníka. Ak sú, tak nastaví príslušné hodnoty, ak nie sú, tak nastaví všetky strany na nulu. Vytvorte niekoľko inštancií a dajte vypísať dĺžky jednotlivých strán.

2. Vytvorte aplikáciu s názvom **BankovyUcet**. Deklarujte triedu **Ucet** s členskými premennými, ktoré umožnia evidovať číslo účtu, stav na účte, počet vkladov a počet výberov. Vytvorte príslušné konštruktory a metódy, ktoré umožnia manipulovať s účtom – vytvoriť účet, vložiť na účet, vybrať z účtu, zistiť stav na účte a zrušiť účet. Vytvorte niekoľko inštancií, tak že budú ukladané do dynamického poľa. Preveďte niekoľko bankových operácií.
3. Vytvorte aplikáciu s názvom **StudijneVysledky.** Program nech umožní zadať meno, priezvisko a triedu žiaka, ďalej známky z troch predmetov a počet vymeškaných hodín. Po zadaní všetkých hodnôt ich program vypíše spolu s priemerom známok. Riešte využitím OOP, pričom priemer známok nech je tiež členskou premennou.

23.8 Otázky

1. Vymenujte základné charakteristiky konštruktora.
2. Čo je to implicitný konštruktor..
3. Čo je to parametrický konštruktor?
4. Na čo slúži rezervované slovo *this*?