## Volání metody

V této chvíli je již vytvořen objekt nad třídou Pes a to umožňuje použít jeho metody stekej a behej. Volání metod se od volání konstruktoru liší. Pokud chceme použít metodu objektu, musíme ji **zavolat** **nad daným vytvořeným objektem**. Použije se k tomu opět tečková notace: za název objektu se přidá tečka a název metody, s konkrétními parametry, které jí mají být předány, v kulatých závorkách.

**Úkol: Metoda stekej vypíše na monitor text „haf“. Metodu poté vyzkoušejme, např. na hnědém psu.**

public void stekej() { //metoda štěkej

System.out.println("haf");

}

Do metody main pak přidáme příkaz:

hnedy.stekej();

**Úkol: uživatel zadá, kolikrát má hnědý pes zaštěkat.**

Nejprve přidáme odkaz na balíček Scanner, který obsahuje třídu Scanner, která zajišťuje načtení dat zadaných uživatelem:

package Pes;

import java.util.Scanner;

poté upravíme metodu stekej, aby přijímala číselný parametr:

public void stekej(int nactistekot) { //metoda štěkej

for (int i=0; i<nactistekot; i++){

System.out.printf("haf ");

}

}

a konečně upravíme metodu main:

Scanner sc = new Scanner( System.in );

System.out.println("Kolikrát má pes zaštěkat?");

int nactistekot = sc.nextInt();

hnedy.stekej(nactistekot);

**Úkol: uživatel zadá, jakou vzdálenost má hnědý pes uběhnout, než zaštěká.**

Nejprve upravíme metodu behej, aby přijímala číselný parametr:

public void behej(int nactivzdalenost){ //metoda běhej

for (int i=0; i<nactivzdalenost; i++){

System.out.printf(" ");

}

Poté upravíme metodu main, skener již máme aktivovaný, není nutné jej znovu inicializovat:

Scanner sc = new Scanner( System.in );

System.out.println("Jakou vzdálenost má pes uběhnout?");

int nactivzdalenost = sc.nextInt();

System.out.println("Kolikrát má pes zaštěkat?");

int nactistekot = sc.nextInt();

hnedy.behej(nactivzdalenost);

hnedy.stekej(nactistekot);

**Úkol: uživatel zadá nový věk psa stary, ten bude uložen do atributu daného objektu a následně vypsán na monitor přepočten na lidský věk (tj. krát 7). Využijte getterů a setterů, atribut vek upravte na private, aby objekt splňoval princip zapouzdření.**

Pozn.: Kód z minulých úkolů můžeme zakomentovat, aby nás nyní nezdržoval.

Upravíme atribut věk na private

private int vek; //číslo udávající věk

Vytvoříme gettery a settery pro atribut věk.

public int getVek() {

return this.vek \* 7;

}

public void setVek(int vek) {

this.vek = vek;

}

Upravíme metodu main:

Scanner sc = new Scanner( System.in );

/\*

System.out.println("Jakou vzdálenost má pes uběhnout?");

int nactivzdalenost = sc.nextInt();

System.out.println("Kolikrát má pes zaštěkat?");

int nactistekot = sc.nextInt();

hnedy.behej(nactivzdalenost);

hnedy.stekej(nactistekot);

\*/

System.out.println("Zadej nový věk starého psa:");

int nactivek = sc.nextInt();

stary.setVek(nactivek);

System.out.println(stary.getVek());

Kód je možné ještě zestručnit:

místo:

System.out.println("Zadej nový věk starého psa:");

int nactivek = sc.nextInt();

stary.setVek(nactivek);

pouze:

System.out.println("Zadej nový věk starého psa:");

stary.setVek(sc.nextInt()); //výsledek skeneru se přidá přímo metodě setVek, není nutné //zakládat proměnou nactivek

**Důležité:** Statické metody se nevolají tečkovou notací s názvem objektu, ale tečkovou notací s názvem třídy, ve které jsou deklarovány. Týká se to například již zmíněné třídy **Math**, jejíž metoda maximum by se v programu volala zápisem **Math.max(a, b);**. Statické metody nacházející se ve stejné třídě, ze které jsou volány, lze volat jen zápsáním názvu metody a parametrů: **statickaMetoda(parametr);**.

## Princip zapouzdření

Pro přístup k atributům objektu je stejně jako pro volání metod možné použít tečkovou notaci. Za název objektu se přidá tečka a jméno atributu, jehož hodnotu je potřeba použít. Proč tak jednoduchá věc nebyla zmíněna už dříve? **Přistupovat k atributům objektu přímo** z ostatních částí programu **je** totiž v naprosté většině případů **nevhodné** a porušuje se tím takzvaný **princip** **zapouzdření**.

Punta.vek = 5;

Princip zapouzdření jednoduše doporučuje, aby ze tříd programu bylo vidět navenek co nejméně. Pokud jsou některé metody volány pouze uvnitř třídy, a ne z ostatních míst programu, měly by mít modifikátor přístupu private. Většina metod přesto zůstane veřejná, protože právě metody umožňují potřebnou práci s objekty.

To ovšem neplatí o atributech. Ke zvolení přímého přístupu k atributům je jen velmi málo důvodů a téměř ve všech případech lze tento přístup nahradit použitím speciálních metod – **getterů a setterů**. Atributy by obecně neměly být mimo třídu viditelné, k upravení jejich hodnoty slouží settery (z anglického set – nastavit) a k získání jejich hodnoty slouží gettery (z anglického get – získat). Ve většině vývojových prostředí lze tyto metody k atributům vygenerovat automaticky.

private int vek;

public int getVek() {

return this.vek;

}

public void setVek(int vek) {

this.vek = vek;

}

Kromě zvýšení bezpečnosti usnadňují gettery a settery také úpravy programu. Kdyby se k atributům přistupovalo přímo, musel by se při jeho změně přepsat všechen kód, který atribut používá. Při použití obalujících metod by se nová logika jednoduše připsala do nich a navenek by se nic nezměnilo.

public int getVek() {

//pomocí jednoduché úpravy je změněna návratová hodnota,

//aniž by se zasahovalo do zbytku programu

return this.vek**\*7**

}

int psiVek = Punta.getVek();