**ÚVOD DO OOP**

Objektovo orientované programovanie ponúka praktickejší pohľad na tvorbu kódu. Proces programovania pripodobňuje okolitému svetu = všetko vôkol nás sú objekty a navzájom na seba pôsobia.

Základ tvorí trieda a objekt. Trieda je šablóna, z ktorej tvoríme objekty. Napr. existuje veľa konkrétnych ľudí (objektov), ale všetci vychádzajú zo spoločnej definície človeka (triedy). Objekty majú vlastnosti a schopnosti, ktoré predpisuje trieda.

Triedu sa tvorí predpisom „class“ a názvom. Je možné vytvoriť ju vedľa inej triedy, alebo dokonca do nej (vnorená trieda). Zaužívané pravidlo však je, že každá trieda by sa mala tvoriť do vlastného súboru, ktorý sa rovnako volá. Taktiež je dobrým zvykom písať názov triedy veľkým písmenom a objektu malým.

Ak chceme využívať triedu v inom priestore (súbore/triede), musia byť v rovnakom mennom priestore – „namespace“, alebo prilinkované pomocou „using“.

Nasledovný príklad vytvára triedu Zviera, ktorá obsahuje dve verejné vlastnosti: typ, vek. Keby neboli verejné (public), nedalo by sa k ním pristúpiť cez objekt – dali by sa používať len v rámci vlastnej triedy. Trieda tiež obsahuje metódu: zapis(), ktorá očakáva dva vstupné parametre. Cez túto metódu vieme nastaviť hodnoty atribútom.

namespace ConsoleApp3

{

class Zviera

{

public string typ;

public int vek;

public void zapis(string t, int v)

{

typ = t;

vek = v;

}

}

}

Objekt vytvárame podobne, ako premennú, čiže najskôr napíšeme názov triedy (akoby typ), a potom názov objektu. Takýto názov však neukazuje na miesto v pamäti, preto treba cez príkaz „new“ alokovať pamäťový priestor. Za „new“ nasleduje konštruktor, čiže metóda, ktorá sa volá rovnako, ako trieda.

Zviera z1 = new Zviera(); // alebo Zviera z1; z1 = new Zviera();

Presná terminológia znie, že zápisom „Zviera z1;“ sme vytvorili len objekt z1. Pomocou zápisu „new Zviera();“ sme vytvorili tzv. inštanciu, čiže objekt, s vyhradeným priestorom v pamäti. Mali by sme teda väčšinou hovoriť o inštanciách, avšak zaužívaný je spoločný názov objekt.

Následne cez bodku (bodkovú notáciu) pristupujeme k verejným atribútom (field) a metódam.

Pri práci s objektami by bolo y vhodné pochopiť aj prácu s pamäťami halda (heap) a zásobník (stack). Objekt len ukazuje (ako odkaz) na pamäť, kde sú dáta. Zápisom objekt1 = objekt2 preto nezapisujeme dáta do pamäťového priestoru objektu1, len prepisujeme odkaz/ukazovateľ na pamäť a tak objekt1 a aj objekt2 ukazujú na tú istú časť pamäte. Toto je častá chyba začiatočníkov v OOP.

Napríklad:

int a = 5; Trieda objekt1 = new Trieda(); objekt1.vek = 5;

int b = 1; Trieda objekt2 = new Trieda(); objekt2.vek = 1;

a = b; objekt1 = objekt2;

b = 10; objekt2.vek = 10;

// v premennej a je 1 a b je 10 // v objekt1.vek aj objekt2.vek je 10

Nasledovný príklad vytvára tri objekty podľa triedy Zviera. Priraďuje im hodnoty cez atribúty, alebo cez metódu (pri druhom spôsobe by mohli byť atribúty súkromné). Následne sa vytvára List pre objekty typu Zviera, takže do Listu zoo môžeme zapisovať jednotlivé celé objekty. Tým, že sa nachádzajú v liste, vieme masovo/hromadne spracovať ich dáta pomocou cyklu. Keďže nepotrebujeme poznať poradie a ani ich meniť, môžeme použiť cyklus foreach().

using System;

using System.Collections.Generic;

namespace ConsoleApp3

{

internal class Program

{

static void Main(string[] args)

{

Zviera z1 = new Zviera();

Zviera z2 = new Zviera();

Zviera z3 = new Zviera();

z1.typ = "macka";

z1.vek = 10;

z2.typ = "pes";

z2.vek = 15;

z3.typ = "mys";

z3.vek = 2;

//ekvivalent

z1.zapis("macka", 10);

z2.zapis("pes", 15);

z3.zapis("mys", 2);

List<Zviera> zoo = new List<Zviera>();

zoo.Add(z1);

zoo.Add(z2);

zoo.Add(z3);

foreach (Zviera ii in zoo)

Console.WriteLine("Zviera " + ii.typ + " má rokov " + ii.vek.ToString());

// ekvivalent

for (int i = 0; i < zoo.Count; i++)

Console.WriteLine("Zviera " + zoo[i].typ + " má rokov " + zoo[i].vek.ToString());

}

}

}

Ako bolo spomenuté, z1.typ a zoo[0].typ je ten istý údaj v pamäti. Takže úpravou jedného sa akoby zmení automaticky aj druhý (v skutočnosti sú však oba názvy nasmerované na to isté miesto v pamäti a teda žiaden druhý nie je).

**KONŠTRUKTOR**

Aaa

**PREŤAŽOVANIE**

Bbb