Доклад

По Увод в обектно - ориентираното програмиране (ООП) – въведение

***1.ООП-определение:***

Обектно ориентираното програмиране (ООП) е парадигма в компютърното програмиране, при която една програмна система се моделира като набор от обекти, които взаимодействат помежду си, за разлика от традиционното виждане, в което една програма е списък от инструкции, които компютърът изпълнява. Всеки обект е способен да получава съобщения, обработва данни и праща съобщения на други обекти.

***2.Клас.Дефиниране на класове:***

Съвкупност от променливи и функции, които са обвързани в логическа структура и работят заедно. Класът служи като модел за представяне на реални обекти и софтуерни обекти, описвайки атрибути (свойства) и методи (поведение) на обектите.

Декларирането на клас следва строго определени правила (синтаксис):

**[<access\_modifier>] class <class\_name>**

Когато декларираме клас, задължително трябва да използваме ключовата дума class. След нея трябва да стои името на класа <class\_name>.

Освен ключовата дума class и името на класа, в декларацията на класа могат да бъдат използвани някои модификатори, например разгледаните вече модификатори за достъп.

***3.Полета и свойства:***

А) ПОЛЕТА: Както стана дума в началото на главата, когато декларираме клас, описваме обект от реалния свят. За описанието на този обект, се фокусираме само върху характеристиките му, които имат отношение към проблема, който ще решава нашата програма.

Б) СВОЙСТВА: В света на обектно-ориентираното програмиране съществува елемент на класовете, наречен свойство (property), който е нещо средно между поле и метод и служи за по-добра защита на състоянието в класа. В някои езици за обектно-ориентирано програмиране, като С#, Delphi / Free Pascal, Visual Basic, JavaScript, D, Python и др., свойствата са част от езика, т.е. за тях съществува специален механизъм, чрез който се декларират и използват. Други езици, като например Java, не подържат концепцията за свойства и за целта програмистите, трябва да декларират двойка методи (за четене и модификация на свойството), за да се предостави тази функционалност.

***4.Конструктор:***

Конструкторите са специални методи, които се извикват автоматично при създаването на обект от даден клас. Конструкторите могат да имат параметри, но обикновено не връщат стойност. Един клас може да има повече от един конструктор, като всеки да е с различен набор параметри. Тяхната задача е да инициализират обекта който се създава, което може да включва:

* задаване на стойности по подразбиране на членове на класа
* заделяне на ресурси
* присвояване на специфични стойности в зависимост от подадените параметри

Конструкторите на един клас в повечето езици за програмиране имат същото име като името на класа. Компилатора автоматично създава конструктор за даден клас дори и да няма дефиниран от потребителя такъв.

***А) Елементарна дефиниция на обект с конструктор:***

#include <iostream>

using namespace std;

class x {

int id;

public:

x(); //Прототип на конструктора

void hey();

} y;

x::x() //Дефиниция на конструктора

{

id = 1;

}

void x::hey()

{

cout << "This is object " << id << " speaking!" << endl;

}

int main()

{

y.hey();

}

Това ще изпише:

**This is object 1 speaking!**

- При инициализацията на обект y на id се задава стойност 1. При извикването на функцията y.hey() на екрана се изписва съдържанието на id оградено от низовете "This is object " и " speaking!".

***5.Методи:***

А) Метод в програмирането е съставна част от една програма, която решава конкретен проблем.

* Може както да приема параметри така и да връща стойност. Целта на писането на дадена програма е решаването на дадена задача. За да бъде ефективно решена дадена задача в програмирането, тя се разделя на подзадачи, разработват се алгоритми за решение на тези подзадачи и накрая тези подзадачи се сглобяват в цялостна програма. Обособените парчета код, решаващи дадената подзадача се наричат подпрограми (subroutines). В различните езици подпрограмите се срещат под други имена като функции (functions) или процедури (procedures). В C#, те се наричат методи (methods).
* Най-простият пример за метод е "Main(…)", който винаги го декларираме между отварящата и затварящата скоба на нашия клас**:**

class HelloCSharp

{ // Отваряща скоба на класа

// Декларираме нашият метод между скобите на класа

static void Main(string[] args)

Извикване на метод от друг клас

{

Console.WriteLine("Hello C#!");

}

} // Затваряща скоба на класа

***Б) Извикване на метод от друг клас -*** За да извикаме метод дефиниран в един клас от друг, то тогава той трябва да е или internal (достъпван само в съответното assembly) или public. Достъпът до метода става чрез името на класа. Private методи са видими само в съответния клас и не могат да бъдат достъпвани от друг.

using System;

class SimpleCalculations

{

double maxValue = GetMax.GetMaxCalculation(1.02, 3);

}

Ако искаме да разберем кой метод извиква нашият код може да използваме StackTrace(using System.Diagnostics;).

StackTrace stackTrace = new StackTrace();

var traceMethod = stackTrace.GetFrame(1).GetMethod().Name;

По този начин получаваме името на метода който извиква.

***6.Статични методи и конструктори:***

***А)Статични методи:***

▪ Статичните методи в класа:

▪ Принадлежат на самия клас

▪ Могат да бъдат достъпени само чрез класа - без създаване на обект от този клас

▪ Удобни са за извършване на действия върху всички обекти от класа или за извършване на действия, които нямат пряко отношение към обектите

***Задача: Аритметични действия***

Създайте клас, който поддържа статични методи за аритметични действия върху две цели числа:

- Add(int, int) – събира числата

- Multiply(int, int) – умножава числата.

Използвайте методите от този клас в Main метода да извършите засичане на команда и извършете операцията.

***Решение: Аритметични действия***

class Arithmetics {

public static int Add(int a, int b){

return a+b;

}

public static int Multiply(int a, int b) {

return a \* b;

}

}▪ Извиквайте методите по аналогичен начин в Main(): Arithmetics.Add(10, 15);

***Б)Статични конструктори:***

Конструкторите в един клас също могат да бъдат статични

▪ Ако един конструктор е статичен той се изпълнява, когато едно от тези събития се случи за първи път:

▪ Създаде се обект от класа (ако той е нестатичен)

▪ Достъпва се статичен член от класа

Най-често статични конструктори се използват за инициализация на статични полета

***Задача: Заявка за корен***

Напишете клас, който съдържа метод, който връща корен квадратен при подадена заявка. Възможно е да получитеголям брой заявки, така че трябва да отговаряте бързо на всяка една от тях.

***Решение: Заявка за корен***

public static class SquareRootPrecalculator {

public const int MaxValue = 1000;

private static double[] sqrtValues;

static SquareRootPrecalculator() {

sqrtValues = new double[MaxValue+1];

for (int i = 1; i <= MaxValue; i++)

sqrtValues[i] = Math.Sqrt(i);

}

public static double GetSqrt(int value) {

return sqrtValues[value];

}

}

***7.Статични полета и свойства:***

Статични полета:

* Когато създаваме обекти от даден клас, всеки един от тях има различни стойности в полетата си.

public class Dog

{

// Instance variables

private string name;

private int age;

}

* Той има две полета съответно за име – name и възраст – age. Във всеки обект, всяко едно от тези полета има собствена стойност, която се съхранява на различно място в паметта за всеки обект.
* Понякога обаче, искаме да имаме полета, които са общи за всички обекти от даден клас. За да постигнем това, трябва в декларацията на тези полета да използваме модификатора static. Както вече обяснихме, такива полета се наричат статични полета (static fields). В литерату­рата се срещат, също и като променливи на класа.
* Казваме, че статичните полета са асоциирани с класа, вместо с който и да е обект от този клас. Това означава, че всички обекти, съз­дадени по описанието на един клас споделят статичните полета на класа.

Изготвил: Валентин , 10А клас

Източници: https://introprogramming.info/intro-csharp-book/read-online/glava14-definirane-na-klasove/#\_Toc298864371