Programovanie v jazyku C#

Abstrakcia, konštruktory a enkapsulácia

prednáška 2

Ing. Ján Magyar, PhD.

ak. rok. 2022/2023 ZS

Princípy OOP

abstrakcia

krytie informácií - zapuzdrenie

dedenie

polymorfizmus

Abstrakcia

modelovanie zložitej reality do programovej reprezentácie

abstrakcia oddelí správanie objektu od implementácie

dôležité sú schopnosti objektu a nie implementačné detaily

abstraktné údajové typy

Trieda vs objekt

trieda definuje abstraktné vlastnosti objektov

každá trieda má definované vlastnosti (**properties**) a schopnosti (**capabilities**)

objekt je konkrétny príklad triedy s konkrétnymi vlastnosťami

Príklad - študijná skupina

Chceme vytvoriť programovú reprezentáciu študijnej skupiny. Aké komponenty/objekty k tomu potrebujeme?

Príklad - študijná skupina

študijná skupina sa skladá zo študentov

skupinu objektov vieme reprezentovať rôznymi spôsobmi v programe

namiesto základnej implementácie vytvoríme novú triedu študijnej skupiny (a študentov)

Príklad - študijná skupina

`Student`

- properties: `firstName`, `familyName`, `grade`
- capabilities: create new object, learn

`StudyGroup`

- properties: `number`, list of students
- capabilities: create new object, add student, remove student, get list of students

UML diagram

členské premenné a metódy majú priradené prístupové možnosti

Objekty ako model sveta

objekt je stavebným blokom programu v OOP

objekt môže modelovať hmatateľné, konceptuálne veci ale aj procesy

každý objekt má vlastnosti (**properties**) a schopnosti (**capabilities**)

Stav objektu

stav objektu je definovaný hodnotami **properties**

na implementačnej úrovni sú to zvyčajne členské premenné

členská premenná je definovaná iba pre triedu

stav objektu sa mení počas života objektu

Typy členských premenných

atribút - popisuje objekt

komponent - je súčasťou objektu

asociácia - objekt používa túto hodnotu, tá ale nie je jeho súčasťou

Správanie objektu

definované schopnosťami, umožňuje objektu vykonávať úlohy

na implementačnej úrovni sú to zvyčajne metódy

metódy sú definované iba pre triedu

správanie objektu sa nemení (účinky ale sú iné)

Typy metód

konštruktor - pre vytvorenie objektu

príkazy - menia hodnotu property

dotazy - poskytujú odpoveď vo forme návratovej hodnoty alebo vedľajšieho účinku (niekedy sa nazývajú ako **funkcie**)

Enkapsulácia

mali by sme zabrániť tomu, aby k properties jedného objektu priamo pristupoval iný objekt

properties sú vnútornou záležitosťou objektu

Vytvorenie inštancií

objekt je inštanciou triedy

objekt sa vytvorí volaním **konštruktora**

konštruktor je špeciálny typ metódy, ktorý sa zavolá na začiatku života objektu

konštruktor zavoláme pomocou kľúčového slova `new`

Vytvorenie objektu

najprv sa alokuje pamäť (automaticky)

zavolá sa konštruktor

inicializuje sa stav objektu (na základe konštruktora)

konštruktor “vráti” referenciu na nový objekt

Konštruktor

rovnaký názov ako trieda

nemá návratový typ (nie je `void`!)

každá trieda musí definovať konštruktor a ten musí byť dostupný

ak nedefinujete vlastný konštruktor, použije sa defaultný konštruktor

Preťaženie konštruktorov

pre jednu triedu sa pomerne často definuje viac konštruktorov s rôznymi parametrami

správny konštruktor sa zavolá na základe počtu parametrov

veľmi často jeden konštruktor zavolá druhý konštruktor (predvolené hodnoty)

Hodnotový a referenčný typ

niektoré údajové typy sa viažu na hodnotu, iné na smerník

každý objekt je referenčného typu

premenné hodnotových typov sa ukladajú v zásobníku

premenné referenčných typov sa ukladajú v halde

Uvoľnenie pamäte

pri zásobníku žiadny problém - vieme vopred kedy uvoľniť premenné

pri halde nevieme vopred, kedy objekt už nepotrebujeme - spomeňte si na `malloc()` a `free()`

vzniká **reachability problem**

Riešenia problému prístupu

ignorujeme problém a veríme, že máme dostatok pamäte

manuálne uvoľňujeme pamäť, ktorú už nepotrebujeme

automatické uvoľnenie pamäte pomocou **garbage collectora**

Managed vs unmanaged code

vykonávanie kódu riadi Common Language Runtime

kód riadený CLR je manažovaný

nemanažovaný kód je riadený programátorom a vykonáva sa priamo v OS (kľúčové slovo `unsafe`)

Garbage collector

vieme vyvíjať programy bez uvoľnenia pamäte

alokuje objekty efektívnym spôsobom

automaticky vymaže už nepoužívané objekty

zabezpečuje pamäť - jeden objekt nevie zasiahnuť do iného objektu (pretečenie pamäte)

Používanie garbage collectora

najčastejšie automaticky

málo fyzickej pamäte

veľkosť využívanej pamäte presiahne istú hodnotu

zavoláme metódu `GC.Collect()`

Definícia uvoľnenia pamäte

finalizer (deštruktor) - špeciálna metóda zavolaná garbage collectorom

metóda `Dispose` - z rozhrania `IDisposable`, môžeme zavolať explicitne

Finalizer

každá trieda môže mať iba jeden finalizer

bez parametrov

zavolá sa pri garbage collection a pri ukončení programu

ideálny pre zrušenie pripojenia s databázou, na zatvorenie súboru a podobne

názov: `~NazovTriedy`

Dispose

trieda musí implementovať rozhranie `IDisposable`

ak je definovaná metóda `Dispose()`, tak finalizer ju zavolá

môžeme ju volať explicitne, vtedy musíme zablokovať finalizer:
`GC.SuppressFinalize(this);`

blok definovaný kľúčovým slovom `using` tiež zavolá `Dispose()` automaticky

Interface a contract

interface (rozhranie) formálne popisuje, čo môžeme zadať a využiť

contract (kontrakt) je dohoda, za akých podmienok niečo použijeme

pre metódu kontrakt definuje:
- parametre a ich typy
- návratovú hodnotu
- správanie, ak príde neplatný vstup

Enkapsulácia (zapuzdrenie)

kým sa abstrakcia sústreďuje na vonkajší pohľad na objekt (jeho rozhranie), enkapsulácia skrýva vnútorné implementačné detaily

enkapsulácia je hlavným nástrojom abstrakcie

dobre navrhnutá a implementovaná trieda je nepriehľadná s dobrým rozhraním

Pravidlá zapuzdrenia

nezverejňujte vnútornú štruktúru triedy

nezverejňujte detaily o internom stave triedy

nezverejňujte rozdiel medzi uloženým a vypočítanom stavom

nezverejňujte implementačné detaily triedy

dáta a operácie nad nimi nech sú v rovnakej triede

Modifikátory viditeľnosti

`private` (privátny)

- iba v rámci tej istej triedy

`public` (verejný)

- dostupný z ľubovoľnej triedy

`protected` (chránený)

- v triede a podtriedach, ako aj v podtriedach vonkajšej triedy

Ďalšie modifikátory

`internal`
- defaultný prístup, v rámci assembly (skupina súvisiacich komponentov)

`protected internal` - podtriedy alebo v rámci assembly

`private protected` - trieda a podtriedy v rámci assembly

`readonly`
- nedefinuje viditeľnosť, ale prístup (uvádza sa po modifikátore viditeľnosti)

Skrývanie informácií

základný princíp: použiť privátne členské premenné a verejné gettre (accessor) a settre (mutator)

```
private double price;
public double GetPrice() {
   return this.price;
}
public void SetPrice(double price) {
   this.price = price;
}
```

Hlavné výhody gettrov a settrov

dokážeme obmedziť hodnoty

ak by používatelia pristupovali k premenným priamo, boli by zodpovední za overenie obmedzení

ak používateľ priamo prepisuje hodnoty, je zodpovedný za vykonanie vedľajších efektov

Properties

getter alebo setter je volaný implicitne

prístup je možné obmedziť podobne ako pri členských premenných

pri automatických nemusíme definovať členské premenné, pre zložitejšie prípady už áno

Definícia properties

```
public class Student
{
    private int age;
    public int Age
    {
        get => age;
        set => age = value;
    }
}
```

```
public class Student
{
    private int age;
    public int Age
    {
        get
        {
            return age;
        }
        set
        {
            age = value;
        }
    }
}
```

Definícia automatických properties

```
public class Student
{
   public int Age { get; set; }
}
```

iba ak nemajú riešiť zložitú logiku

je možné aj inicializovať hodnotu

Používanie properties

```
student.Age = 23;
int age = student.Age;
```

Modifikátory viditelnosti properties

je možné zadať rôzne modifikátory pre `get`, resp. `set`

getter a setter defaultne zoberie modifikátor property

pre read-only property nezadefinujeme `set`

Kľúčové slovo `this`

`this` je odkaz na aktuálny objekt - pre členské premenné, metódy a konštruktor

k prístupu na samotný objekt

k prístupu k “shadowed” členom triedy

Shadowing

členská premenná je “zatienená” parametrom metódy alebo konštruktora

```
public Student(string name, int age) {
   this.name = name;
   this.age = age;
}
```

Použitie `this` v konštruktore

pre explicitné volanie konštruktora (preťaženie)

po novom dá sa riešiť cez predvolené hodnoty (čiastočne)

volanie druhého konštruktora musí byť prvým príkazom

A čo globálne premenné?

keď sa dá, tak ich nepoužívajte

zavedenie statickej pamäte a statických členov

statický člen je dostupný aj bez existencie inštancie triedy

prístupný pod menom triedy

iba jedna kópia

Statické premenné

statické polia by mali byť inicializované (napr. cez statický konštruktor)

môžu byť využité ako globálne premenné (zvyčajne konštanty)

uložené v statickej pamäti, jedna kópia

deklarované pomocou kľúčového slova `static`

Statické metódy

volateľné z akejkoľvek inštancie

deklarované pomocou kľúčového slova `static`

nedokážu pristupovať k členským premenným svojej triedy

môžu volať iba statické metódy

nemôžu využiť `this` a `base`

jeden jednoznačný príklad

Členské vs statické premenné

každá inštancia triedy má svoje vlastné kópie členských premenných

iba jedna kópia statickej premennej

Best practices

statické premenné by zvyčajne mali byť nemeniteľné (pre konštanty)

meniteľné statické premenné: počítadlo inštancií, asi aj iné…

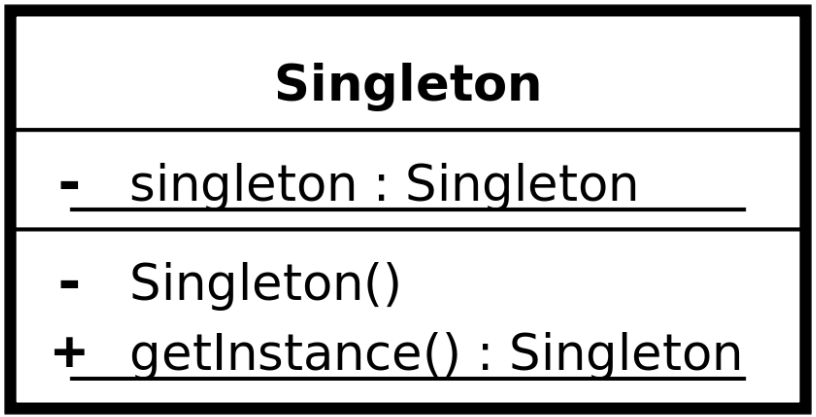
nepoužívajte statickú metódu ak je potrebné zapamätať si niečo medzi volaniami

statická metóda by mala byť sebestačná

statická metóda by mala využívať iba nemeniteľné hodnoty alebo parametre

Singleton

návrhový vzor pre zabezpečenie vytvorenia jednej jedinej inštancie istej triedy



otázky?