Programovanie v jazyku C#

Základy objektovo orientovaného programovania

prednáška 1

Ing. Ján Magyar, PhD.

ak. rok. 2021/2022 ZS

Paradigmy programovania

sekvenčné programovanie

štrukturované programovanie

modulárne programovanie

objektovo orientované programovanie

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

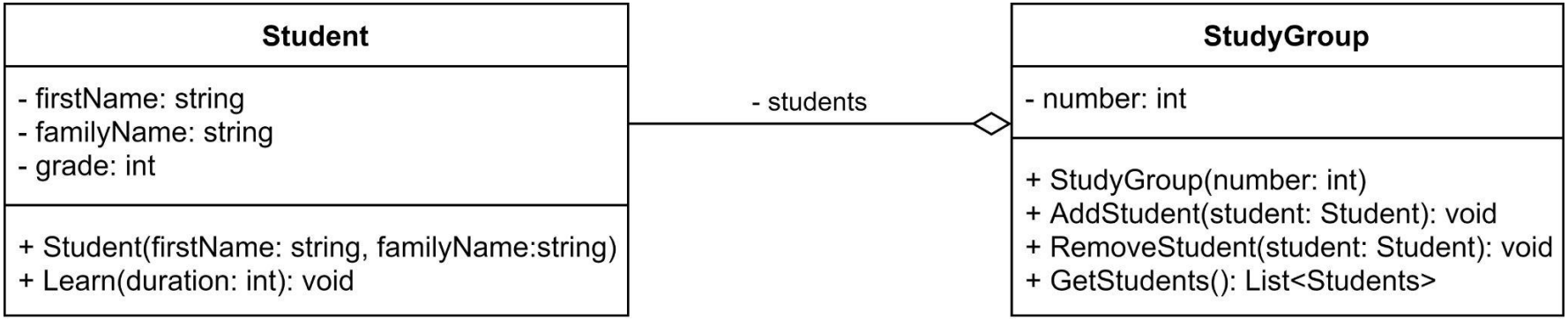
`Student`

- 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

Enkapsulácia

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

properties sú vnútornou záležitosťou objektu

viac na nasledujúcej prednáške

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

definovaná 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**)

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

otázky?