Programovanie v jazyku C#

Polymorfizmus

prednáška 5

Ing. Ján Magyar, PhD.

ak. rok. 2021/2022 ZS

Polymorfizmus

jeden objekt môže mať veľa foriem

OOP podporuje polymorfizmus cez dedičnosť a implementáciu rozhraní

invokovaná metóda sa môže zmeniť pri používaní polymorfizmu

Typy polymorfizmu

subtype polymorfizmus - podtypy, dedičnosť, implementácia

ad-hoc polymorfizmus - preťaženie metód

parametrický polymorfizmus - generické programovanie

Preťaženie metód

definícia viacerých metód s rôznymi typmi parametrov

správna implementácia sa zavolá na základe statickej kontroly

zvýši sa zložitosť funkcionality triedy

Preťaženie a prekrytie metód

prekrytie metód je správny prístup

preťaženie metód by sme nemali používať ak sa dá

používajte jedinečné názvy metód

Downcasting

je možné zmeniť polymorfickú referenciu počas behu programu?

```
Person p = new Student(“Janko”, “Hrasko”);

Student s = (Student) p;
```

Problémy s downcastingom

kontrola typov sa uskutoční počas behu

ak objekt bol pôvodne inštanciou triedy, do ktorej chceme robiť downcasting, je to v poriadku

ak skutočný typ objektu je iný ako typ, do ktorého robíme downcasting, vygeneruje sa runtime chyba

Používanie downcastingu

nerobte to

ide proti zásadám dobrého objektového návrhu

porušuje Liskov Substitution Principle

riešenie v C# - generické programovanie (neskôr)

Liskov Substitution Principle

Funkcie, ktoré používajú referencie na nadtriedu (base) musia byť schopné pracovať s objektmi podtried (odvodených) bez toho, aby o tom vedeli.

Ukážka - obdĺžniky a štvorce

Majme jednoduchú reprezentáciu obdĺžnika s dvoma členskými premennými (`width`, `height`), so základnou funkcionalitou getterov a setterov a s metódou pre výpočet obsahu.

Vieme reprezentovať štvorec ako podtyp obdĺžnika?

Ukážka - implementácia štvorca

štvorec zdedí funkcionalitu od obdĺžnika

členské premenné `width` a `height` budú čiastočne zbytočné

potrebujeme prekryť metódy `SetWidth` a `SetHeight`

Ukážka - problémy s implementáciou

nedodržali sme LSP

`SetWidth` (a `SetHeight`) v jednom prípade mení iba jednu premennú, v prípade štvorca aktualizuje obe členské premenné

programátor, ktorý píše testy, nepredpokladá, že setter by mal aktualizovať aj inú premennú

aj keď jednotlivé triedy sú správne navrhnuté, nedávajú zmysel v kontexte celkového riešenia

LSP zásady

dedičnosť a vzťahy typu IS-A sú nielen o reprezentácii ale aj o správaní objektov

všetky podtriedy sa musia správať rovnakým spôsobom ako sa očakáva od nadtriedy

podtrieda nesmie mať viac ohraničení ako nadtrieda

ak chceme dodržiavať LSP, podtrieda musí byť použiteľná všade tam, kde sa používa nadtrieda

Princíp substitúcie

Kedy bude B podtypom A s istotou?

- všade kde kód očakáva A, môžeme použiť aj B
- ak máme funkciu *f(A)*, ktorá sa vykoná správne ak dostane objekt typu A, musí mať rovnakú funkcionalitu ak dostane objekt typu B

Subtype podmienka na úrovni deklarácií

návratový typ musí byť podtriedou pri náhrade

typ parametra musí byť nadtriedou pri náhrade

v C# a ostatných objektových jazykoch riešená cez rozdelenie prekrytia a preťaženia

Subtype podmienka na úrovni metód

predpoklady pre metódu podtriedy musia byť voľnejšie ako pre nadtriedu

podmienka pre výsledok metódy musí byť silnejšia v podtriede ako v nadtriede

Subtype podmienka pre vlastnosti

podtrieda musí zachovať všetky vlastnosti nadtriedy, môže ich ale rozširovať

Typové operátory - `is`

`a is T`

kontroluje, či objekt `a` je typu `T`

`true` ak je to priamy typ, podtyp alebo implementácia, vieme ho konvertovať pomocou boxingu alebo unboxingu

Typové operátory - `as`

```
a as T
```

konvertuje objekt `a` na typ `T` ak je to možné

v porovnaní s pretypovaním nikdy negeneruje chybu

ak transformácia zlyhá, vráti `null`

Typové operátory - `typeof`

```
object p = new Person(“Janko”, “Hrasko”)
if (p.GetType() == typeof(Person))
```

vráti inštanciu `System.Type` pre typ, užitočné pre porovnávanie

Kompozícia objektov

spojenie objektov a dátových typov do zložitejších štruktúr

rozšírenie vlastností a správania objektu zabezpečená pridaním nových členov

preferovaný spôsob v moderných architektúrach

Agregácia

vzťah typu “má”

vyjadruje to, že objekt je súčasťou iného objektu

binárny vzťah

trieda je kontajnerom iných tried, ale súčasti nie sú závislé od kontajnera

Kompozícia

podobná agregácii

kompozit obsahuje jednotlivé časti fyzicky

vysoká závislosť od životného cyklu kompozitu

Agregácia

reprezentácia softvérového a databázového stavu

ak zanikne kontajner, súčasti neprestanú existovať (študijná skupina sa skladá zo študentov)

Kompozícia

reprezentácia súčastí (motor je súčasťou auta)

ak zanikne kontajner, obsah sa vymaže (univerzita sa skladá z fakúlt)

otázky?