Objektově orientované programování

Filip Rosa

2025

Obsah

1 Modularita

1.1 Co je hlavním motivem pro vývoj programovacího paradigmatu od imperativního k objektovému?

Hlavným motívom pre prechod od imperatívneho programovania k objektovo orientovanému programovaniu je zjednodušenie riadenia komplexity pri vývoji a údržbe veľkých softvérových systémov. Tento prechod priniesol niekoľko kľúčových výhod, ktoré sa týkali najmä organizácie kódu, opakovateľnosti, rozšíriteľnosti a správy stavu aplikácií.

1.2 Co je imperativní programování?

Poznáme ho z bežne používaných jazykov. Môžeme ho rozdeliť na dva druhy:

- **procedurálne programovanie** - postupnosť krokov, ktorými meníme stav premenných programu
- **štruktúrované programovanie** - sekvencie, iterácie, vetvenie, skoky, abstrakcia

1.3 Co je modulární programování?

Je to návrh zhora-dolu. Rozdeľuje program na nezávislé, zameniteľné moduly, ktoré zaisťujú jednotlivé drobné funkčnosti. Modul obsahuje všetko potrebné pre zaistenie funkčnosti (dáta a algoritmy).

1.4 Jaké jsou hlavní faktory kvality software?

- vnútorné sú schované pred používateľom (AKO)
- vonkajšie popisujú správanie navonok - správnosť, robustnosť, rýchlosť, rozšíriteľnosť... (ČO)
- musíme byť schopní merať kvalitu software

1.5 Co je pochopitelnost modulu? Uveďte příklad.

Modul by mal vykonávať jednu jasne definovanú a pochopiteľnú úlohu, prípadne niekoľko málo jasne definovaných úloh.
Napr. modul pre prácu s dátumami funkciou is_leap_year jednoducho zisťuje, či je rok priestupný, a days_in_month vracia počet dní v danom mesiaci.

1.6 Co je samostatnost modulu? Uveďte příklad.

Každý modul musí byť relatívne samostatný a mal by mať čo najmenej väzieb na ostatné moduly. Nebolo by vhodné, aby všetky moduly programu boli navzájom prepojené a na sebe závislé.
Napr. modul pre základné operácie s textom. Vykonáva dve nezávislé operácie: konverziu textu na veľké písmená a počítanie samohlások. Nevyžaduje žiadne externé knižnice alebo konfigurácie. Môže byť použitý samostatne v akomkoľvek projekte.

1.7 Co je kombinovatelnost modulu? Uveďte příklad.

Moduly musia byť navzájom kombinovateľné. Musí byť možné modul vziať a použiť v inom kontexte alebo v inom projekte.
Napr. máme dva moduly: jeden na spracovanie textu a druhý na spracovanie čísiel. Oba moduly budú kombinovateľné v rámci väčšej aplikácie, kde budú spolupracovať. Každý z nich vykonáva jasnú úlohu a má jednoduché rozhranie.

1.8 Co je zapouzdření modulu? Uveďte příklad.

Moduly musia mať právo na isté súkromie: je prípustné a žiadúce, aby všetky informácie, ktoré nie sú potrebné pre klientov modulu zostaly skryté vnútri modulu. Väčšina funkcionality modulu je skrytá a len malá časť je viditeľná zvonku. Skrytej časti vravíme **implementácia** modulu a verejnej časti **rozhranie** modulu.
Napr. Máme triedu Car, ktorá reprezentuje automobil. Tento automobil má súkromné údaje, ako je aktuálny počet paliva v nádrži. Chceme, aby tieto údaje boli prístupné iba prostredníctvom metód triedy a neboli priamo meniteľné z vonkajšieho kódu.

1.9 Co je explicitní rozhraní modulu? Uveďte příklad.

Z deklarácie modulu musí byť všeobecne zrejmé, aké predpoklady pre vyokonávanie svojej úlohy potrebuje.
Napr. Vytvárame modul na spracovanie jednoduchých matematických operácií. Tento modul bude mať jasne definované funkcie pre sčítať, odčítať, vynásobiť a vydeliť čísla, ktoré tvoria jeho explicitné rozhranie.

1.10 Co je syntaktická podpora modularity?

Moduly počítačového programu musia byť jasne vymedzené syntaktickými jednotkami programu. Zo zápisu programovacieho jazyka musí byť zrejmé, kde končí a začína zápis modulu.

1.11 Co je pět kritérií pro dobrou modularitu?

- dekomponovateľnosť
- kombinovateľnosť
- pochopiteľnosť
- kontinuita
- ochrana

1.12 Co se rozumí pěti pravidly zajišťující dobrou modularitu?

- priame mapovanie
- pár rozhraní
- malé rozhrania

- explicitné rozhrania
- skrývanie informácií

1.13 Popište jednotlivá kritéria dobré modularity. Uveďte příklady.

- dekomponovateľnosť - schopnosť rozložiť veľký a komplexný systém na menšie, samostatné komponenty. Predstavme si, že vyvíjame webovú aplikáciu pre správu objednávok. Celkový systém môžeme rozdeliť na menšie moduly, ako sú: odul pre správu používateľov: Registrácia, prihlásenie, správa profilov. Modul pre spracovanie objednávok: Vytváranie objednávok, platby, správa stavu objednávky. Modul pre spracovanie platieb.
- kombinovateľnosť - schopnosť spájať rôzne moduly alebo komponenty do väčších celkov, ktoré spolupracujú, aby vykonávali zložitejšie úlohy. Zoberme si modul na spracovanie platieb v online obchode. Tento modul môže byť kombinovaný s rôznymi ďalšími modulmi ako: Modul pre spracovanie objednávok: Keď používateľ vytvorí objednávku, modul pre spracovanie objednávok využije modul na spracovanie platieb na dokončenie transakcie. Modul pre notifikácie.
- pochopiteľnosť - systém alebo modul je ľahko čitateľný a zrozumiteľný pre vývojárov alebo iných používateľov systému - premenné sú popisné, názov funkcie je jasný, kód je čitateľný
- kontinuita - systém alebo modul by mal fungovať bez prerušenia aj pri zmenách alebo aktualizáciách. Ak aktualizujeme databázový systém, musíme zabezpečiť, že staršie verzie systému budú fungovať aj po implementácii novej verzie.
- ochrana - zabezpečenie systému pred neautorizovaným prístupom, chybami alebo nepredvídanými situáciami. - autentifikácia, ochrana údajov, validácia vstupov

1.14 Popište jednotlivá pravidla pro dobrou modularitu. Uveďte příklady.

- priame mapovanie - existuje jednoznačný a priamy vzťah medzi internými komponentmi alebo dátovými štruktúrami a rozhraním modulu. Predstavme si triedu na reprezentovanie bankového účtu. Ak by sme použili priame mapovanie, poskytli by sme metódy, ktoré priamo manipulujú s internými hodnotami (ako je zostatok).
- pár rozhraní - modul poskytuje dve alebo viac rozhraní, ktoré sa vzájomne dopĺňajú a používajú sa spoločne. Predstavme si systém na správu súborov, ktorý poskytuje rozhranie na čítanie a zapisovanie do súborov, ale každé z týchto rozhraní môže byť prispôsobené pre rôznych používateľov (napr. pre čitateľov a zapisovačov).
- malé rozhrania - rozhranie modulu je minimalizované na čo najmenší počet metód alebo funkcií, ktoré vykonávajú jasne definovanú a konkrétnu úlohu. Predstavme si modul na validáciu používateľských údajov. Tento modul môže mať malé a špecifické rozhranie. Modul poskytuje len dve metódy, ktoré vykonávajú veľmi špecifické úlohy – validáciu e-mailu a validáciu hesla.

- explicitné rozhrania - rozhranie modulu je jasne definované a dokumentované. Rozhranie modulu je explicitné, pretože používateľ má jasne definované metódy na pridávanie položiek a získanie celkovej ceny. Modifikácia interných dát (napríklad priamy prístup k zoznamu položiek) nie je povolená.
- skrývanie informácií - interné detaily implementácie sú skryté pred používateľmi modulu. redstavme si triedu na správu bankového účtu, ktorá skrýva interný stav (ako je zostatok) a poskytuje len verejné metódy na interakciu.

1.15 K čemu je konstruktor? Uveďte příklad.

Konštruktor inicializuje dáta objektu hodnotami parametrov v konštruktori (naplní pamät dátami).

1.16 K čemu je destruktor, kdy ho potřebujeme a kdy ne? Uveďte příklad.

Deštruktor odstráni v pamäti dáta objektu (čistí pamäť). Nie je potrebný pokiaľ sú dáta objektu statické.