JAVA programavimo kalba

Projektavimo šablonai

Klasių kompozicija ir paveldimumas

Vienas iš pagrindinių objektinio programavimo principų yra kuriant programą nusistatyti kaip dvi klasės yra priklausomos viena nuo kitos.

Yra du pagrindiniai klasių priklausomybės būdai:

- Klasių paveldimumas (inheritance) - “yra” (“is-a”) principas
- Klasių kompozicija (composition) - “turi” (“has-a”) principas

Klasių paveldimumas

Klasių paveldimumas (inheritance) - “**yra**” (“**is-a**”) principas

class A { … }

class B extends A { … }

- Klasės B objektas “**yra**” laikomas ir klasės A objektu
- Sakoma, kad klasė A yra super-klasė klasei B, o klasė B yra klasės A sub-klasė
- **Privalumas** - labai lengva išplėsti klases ir galima panaudoti super-klasėse aprašytus metodus nieko nekeičiant. Bet, jei reikia, tai galima sub-klasėse perrašyti (override) super-klasės metodą. T.y. programos modifikavimai lengvai daromi, jei tai liečia naujas sub-klases.
- **Trūkumas** - pasikeitus super-klasės interfeisui (pvz: metodo parametrų skaičiui arba grąžinamos reikšmės tipui) pasikeičia ir visų sub-klasių ir jų sub-klasių interfeisai, t.y. juos visus reikia taisyti.

Klasių kompozicija

Klasių kompozicija (composition) - “**turi**” (“**has-a**”) principas

class A { … }

class B {A a;…}

- Klasės B objektas “**turi**” ir klasės A objektą
- Sakoma, kad klasė B yra priekinė (front-end) klasė, o klasė A yra galinė (back-end) klasė.
- Jei reikia padaryti iš išorės prieinamus galinės klasės metodus, priekinė klasė realizuoja atitinkamus metodus.
- **Privalumas** - kadangi galinės klasės metodai tiesiogiai nėra prieinami iš išorės, tai galinės klasės interfeiso pakeitimai nesugriauna priekinės klasės interfeiso.
- **Trūkumas** - priekinėje klasėje reikia realizuoti metodus kurie nieko kito nedaro, tik kviečia atitinkamus galinės klasės metodus.

Klasių paveldimumas ir kompozicija - skirtumai

1. Žymiai lengviau pakeisti galinės klasės interfeisą nei super-klasės galinės klasės interfeiso pakeitimai niekaip neįtakoja priekinės klasės interfeisą.

2. Žymiai lengviau pakeisti priekinės klasės interfeisą nei sub-klasės. Sub-klasės metodas gali “netyčia” perrašyti super-klasės metodą sugriaudamas funkcionalumą. Taip pat sub-klasė negali turėti turėti tokį pat metodą kaip ir super klasės kuris grąžina kito tipo reikšmę.

3. Lengviau yra pridėti naujas sub-klases nei naują priekinę klasę. Tai todėl kad sub-klasėje užtenka perrašyti ar realizuoti tik tuos metodus kurie skiriasi nuo super-klasės.

Istorija

- 1994 taip vadinamas “Gang of Four” (GoF), t.y. Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides išleido knygą “Design Patterns: Elements of Reusable Object-Oriented Software”, kurioje aprašė 23 programavimo projektavimo šablonus dabar vadinamus klasikiniais.
- Jie buvo aprašyti su pavyzdžiais C++ ir Smaltalk kalbose bet labai lengvai realizuojami ir kitose kalbose.
- Šablonų naudojimas pagreitina programavimo procesą, kodą daro labiau suprantamą ir lengviau testuojamą.
- Kritikai sako, kad šablonai iš esmės yra tik priemonė pasiekti tai, ko nėra standartinėje C++ kalboje ir kad kitose kalbose (pvz Lisp) tas funkcionalumas jau iš karto yra kaip kalbos dalis.

Šablonų tipai

- Paprastai šablonai skirstomi į grupes:
  - Sukūrimo (Creational)
  - Struktūriniai (Structural)
  - Elgsenos (Behavioral)

Sukūrimo šablonai (creational patterns)

Sukūrimo (creational) šablonai - tai šablonai kurie abstrahuoja objektų kūrimo procesą, t.y. padeda programos kodą padaryti nepriklausomą nuo objektų kūrimo proceso perduodant tai atlikti kitam objektui.

- Naudotini kada:
  - Objektų kūrimas turi būti nepriklausomas
  - Turi būti naudojami kartu keli skirtingų klasių objektai (kompozicija)
  - Norima paslėpti klasių realizavimus paskelbiant tik jų interfeisus
  - Objekto funkcionalumas gali būti praplėstas nekeičiant objekto
  - Gali būti sukurtas ir naudojamas tik vienas duotos klasės objektas

Abstraktaus fabriko šablonas (abstract factory pattern)

Abstraktaus fabriko šablonas sprendžia šias problemas:

- atskirti arba padaryti nepriklausomu objekto sukūrimo procesą
- padaryti klasę nepriklausomą nuo to kaip jai reikalingi objektai yra sukuriami.
- sukurti priklausomus arba susijusius objektus. Tai atliekama paslepiant objekto kūrimą specialaus metodo, taip vadinamo fabriko, realizacijoje, kuri aprašoma tam skirtoje sąsajoje (interface)
- Objektas kuriamas kviečiant ne klasės konstruktorių, o metodą – fabriką.
- https://en.wikipedia.org/wiki/Abstract_factory_pattern

Kūrėjo šablonas (builder pattern)

Kūrėjo šablonas sprendžia šias problemas:

- supaprastinti sudėtingo objekto kūrimo procesą
- Tai atliekama objekto kūrimą patikint kitai klasei, vadinamai kūrėju (Builder), kurio metodų pagalba objektas palaipsniui konstruojamas/aprašomas ir gale sukuriamas.

- **Privalumas** - lengvai kuriami skirtingų parametrų objektai. Jų kūrimo procesas paslėptas.
- **Trūkumas** - reikia sukurti atskiras statytojo klases kiekvienai statomų objektų klasei.
- https://en.wikipedia.org/wiki/Builder_pattern

Fabriko-metodo šablonas (factory method pattern)

Fabriko-metodo šablonas sprendžia šias problemas:

- sukurti objektą nežinant ir nenurodant kokio tiksliai klasės objekto reikia ir koks objektas sukuriamas yra apsprendžiama sub-klasėse. Tai atliekama kuriant objektą kviečiant metodą-fabriką pagal interfeisą, kurį realizuoja kažkokios sub-klasės arba kviečiant bazinės klasės metodą-fabriką, kuris yra sub-klasėse perrašomas.
- https://en.wikipedia.org/wiki/Factory_method_pattern

Prototipo šablonas (prototype pattern)

Prototipo šablonas sprendžia šias problemas:

- sukurti objektą ne konstruojant naują, kas gali būti labai imli resursams operacija, o kopijuojant (klonuojant) egzistuojantį prototipinį objektą.
- Tai atliekama aprašant abstrakčią klasę su clone() metodu, kuris po to perrašomas sub-klasėse.
- https://en.wikipedia.org/wiki/Prototype_pattern

Vienintelio objekto šablonas (singleton pattern)

Vienintelio objekto šablonas sprendžia šias problemas:

- kaip užtikrinti, kad duotos klasės būtų galima sukurti tik vieną objektą. Tai atliekama “paslepiant” klasės konstruktorių ir “atveriant” statinį metodą (paprastai vadinamą getInstance()), kuris grąžina visą laiką tą patį objektą.
- https://en.wikipedia.org/wiki/Singleton_pattern

Struktūriniai šablonai (structural patterns)

Struktūriniai šablonai skirti palengvinti įvairių dviejų klasių objektų ryšių realizaciją. Jie aprašo įvairius tipinius variantus kaip tokie ryšiai turi būti realizuoti.

Adapterio šablonas (adapter pattern)

Adapterio objekto šablonas sprendžia šias problemas:

- kai norima naudoti klases, realizuojančios interfeisus kurie mums netinka. Tai atliekama aprašant papildomą klasę Adapter, kurioje atliekamas interfeisų.
- suderinimas, t.y. norimas naudoti interfeisas (Adaptee) verčiamas į tinkamą naudojimui (Target).
- https://en.wikipedia.org/wiki/Adapter_pattern

Tilto šablonas (bridge pattern)

Tilto objekto šablonas sprendžia šias problemas:

- kai abstrakcija (Abstraction) ir jos implementacija (Implementor) yra aprašomos ir realizuojamos (išplečiamos) nepriklausomai viena nuo kitos. Tai atliekama abstrakcijos klasę ir jos realizavimo klasę aprašant skirtingose klasių hierarchijose, t.y. šios klasės nėra viena kitos super ar sub klasės.
- https://en.wikipedia.org/wiki/Bridge_pattern

Sudėtinis šablonas (composite pattern)

Sudėtinis objekto šablonas sprendžia šias problemas:

- kai kažkokių objektų kolekcija traktuojama kaip vienas toks pat objektas, t.y. objektų visuma ir vienas objektas yra traktuojami vienodai. Tai atliekama aprašant Component interfeisą ir jį realizuojant ir objekto dalyje (Leaf) ir objekto visumoje (Composite). Objekto dalys (Leaf) realizuoja interfeisą tiesiogiai, o pats objektas nukreipia interfeiso metodus kažkuriai daliai.
- https://en.wikipedia.org/wiki/Composite_pattern

Dekoratoriaus šablonas (decorator pattern)

Dekoratoriaus objekto šablonas sprendžia šias problemas:

- kai objekto funkcionalumas gali būti keičiamas vykdymo metu dinamiškai. Tai atliekama aprašant Decorator objektą, kuris išplečia objektą (Component) modifikuojant vienus metodus ir nukreipiant kitus to objekto metodus į jį patį.
- https://en.wikipedia.org/wiki/Decorator_pattern

Fasado šablonas (Facade pattern)

Fasado objekto šablonas sprendžia šias problemas:

- kai sudėtingo objekto funkcionalumas supaprastinamas paskelbiant paprastus metodus, o sudėtingi veiksmai atliekami viduje. Tai atliekama aprašant Facade objektą, kuris realizuoja paprastą interfeisą nukreipiantį metodus į vidinę sistemą ir galintį atlikti tarp tų metodų reikiamus veiksmus.
- https://en.wikipedia.org/wiki/Facade_pattern

Lengvojo objekto šablonas (flyweight pattern)

Lengvojo objekto šablonas sprendžia šias problemas:

- kai reikia taupyti objektų naudojamus resursus (atmintį), tai kai kurios objektų dalys naudojamos bendrai visiems to tipo objektams. Tai atliekama išskiriant objekte bendro naudojimo dalį (invariant) ir specifinę kiekvienam objektui dalį (variant), kuri negali būti naudojama bendrai.
- https://en.wikipedia.org/wiki/Flyweight_pattern

Pavaduotojo šablonas (proxy pattern)

Pavaduotojo šablonas sprendžia šias problemas:

- kai reikia kontroliuoti metodų prieinamumą. Tai atliekama sukuriant atskirą objektą (Proxy), kuris pakeičia objektą (Subject) ir kontroliuoja pastarojo metodų iškvietimus.
- https://en.wikipedia.org/wiki/Proxy_pattern

Elgsenos šablonai (behavioral patterns)

Elgsenos šablonai skirti realizuoti įvairių komunikacijų tarp objektų variantus. Jie aprašo kaip objektai komunikuoja (keičiasi informacija) vienas su kitu.

Atsakomybės grandinės šablonas ( )

Atsakomybės grandinės šablonas sprendžia šias problemas:

- kai reikia kad siuntėjas (sender) nebūtų griežtai surištas su gavėju (receiver)
- kad siuntėjas galėtų komunikuoti ne tik su vienu bet su keliais gavėjais
- Tai atliekama aprašant grandinę gavėjų, paprastai užtenka aprašyti tik dviejų gavėjų susirišimą jei jie sudaro hierarchiją, kuriems paeiliui perduodamas pranešimas tol kol einamasis gavėjas jį priima.
- https://en.wikipedia.org/wiki/Chain-of-responsibility_pattern

Komandos šablonas (command pattern)

Komandos šablonas sprendžia šias problemas:

- kai reikia kad užklausą generuojantį objektą atrišti nuo pačios užklausos
- Šablone veikia keturi objektai:
  - **Komanda (command)** - objektas žinantis gavėją (receiver) ir vykdantis gavėjo metodus. Komanda vykdoma kai kviečiantysis vykdo jos metodą execute().
  - **Gavėjas (receiver)** - tai objektas su kuriuo atliekami veiksmai, t.y. komandų vykdymo metu vykdomi jo metodai.
  - **Kviečiantysis (invoker)** - objektas kuris žino kaip vykdyti komandą pagal jos interfeisą, bet nežino nieko apie konkrečią komandą ir jos gavėją.
  - **Klientas (client)** - tai objektas kuris laiko savyje ir komandų ir gavėjų objektus. Jis apsprendžia kuris gavėjas bus vykdomas kurios komandos ir kuri konkreti komanda bus priskiriama kviečiančiajam.
- ▸ https://en.wikipedia.org/wiki/Command_pattern

Stebėtojo šablonas (observer pattern)

Stebėtojo šablonas sprendžia šias problemas:

- kai reikia sekti objekto būsenos pakitimus, t.y. pasikeitus stebimo objekto kažkuriam laukui, kiti objektai stebėtojai (observers) turi gauti pranešimus – būtų iškvieti atitinkami jų metodai.
- Reikia aprašyti stebimąjį objektą (subject) ir stebėtojus (observers), kurie bus įspėjami automatiškai kai stebimo objekto būsena pasikeis.
- https://en.wikipedia.org/wiki/Observer_pattern

Uždaviniai

- Pasikartoti ir išsinagrinėti kodą.