Projektavimo šablonai

Maksim Norkin
maksim.norkin@ieee.org

I. Įžanga

Programų inžinerijoje, projektavimo šablonas yra bendras, daug kartų naudojamas sprendimas nuolat kylančiai problemai išspręsti. Projektavimo šablonas nėra galutinis programinės įrangos sprendimas. Tai yra šablonas arba aprašymas kaip galima išspręsti kažkokią problemą skirtingose situacijose. Šablonus galima pervadinti į sprendimų rinkinius, kurie yra išgryninti programuotojų bendruomenės ir sąlygoja geriausią sprendimą. Objektiškai orientuoti projektavimo šablonai dažniausiai nurodo susiejimus ir sąveikas tarp klasių arba objektų, nesileidžiant į galutinių objektų ar klasių aprašymo. Daugelis šablonų įgyvendina objektiškai orientuotą arba paveldimą būseną, todėl jie gali būti neįmanomi pritaikyti funkciniame programavime.

Projektavimo šablonai priklauso modulių ir sujungimų srityje. Aukštesniam lygmenyje yra naudojami architektūriniai šablonai, kurie dažniausiai nusako bendrą šabloną, kurio privalo laikytis visa likusi sistema.

Egzistuoja labai daug projektavimo šablonų, pavyzdžiui:

- Algoritminės strategijos šablonai, kurie nusako galimus sprendimus, kurie yra susiję su aukšto lygio strategijomis – kaip išnaudoti programos charakteristikas skaičiavimo mašinoje.
- Skaičiuojamieji projektavimo šablonai, kurie nusako svarbiausius skaičiavimo proceso vietas.
- Paleidimo šablonas, nusako programos paleidimo palaikymą, įtraukiant strategijas, kurios nusako kaip paleisti programinės sistemos srautus, taip remiant užduočių sinchronizavimą.
- Įgyvendinimo strategijos šablonai, nusako kaip galima pateikti programinį kodą programos organizavimui ir bendrus duomenų tipus, konkrečiai lygiagrečiam programavimui.
- Struktūriniai projektavimo šablonai, nusako kaip galima įvykdyti aukšto lygio struktūras programos kūrimo metu.

II. Istorija

Šablonai atsirado kaip architektūrinė idėją, kurią pristatė Christopher Alexander. 1987 metais, Kent Beck ir Ward Cunningham pradėjo idėjos eksportavimą į kitas šakas, tarp kurių ir programavimas. Savo darbo rezultatus Jie paskelbė OOPSLA konferencijoje tais pačiais metais. Vėlesniais metais, jų darbas buvo tęsiamas toliau.

Projektavimo šablonai gavo labai didelį populiarumą kompiuterių moksle, kai 1994 buvo išleista knyga “Design Patterns: Elements of Reusable Object-Oriented Software”, kurių autoriai yra “Gang of Four” (Gamma et al.), kurie dažniausiai yra trumpinami iki “GOF”. Tais pačiais metais, buvo surengta pirmoji *Pattern Languages of Programming* konferencija. Sekančiais metais, buvo įsteigta *Portland Pattern Repository*, kurios paskirtis buvo projektavimo šablonų dokumentavimas.

Turint omenyje, kad projektavimo šablonai buvo taikomi praktikoje labai ilgą laiką, formalus jų aprašymas yra nyksta jau keletas metų.

Tik 2009, Tomas Erl kartu su 30 inžinierių vieningomis jėgomis išleido knygą “SOA Design Patterns”, kurios tikslas buvo nustatyti *de facto* projektavimo šablonų apibūdinimus SOA ir paslaugų tipo sprendimams.

III. Praktika

Projektavimo šablonai gali labai paspartinti darbo procesą, pateikdami išbandytus, įrodytus projektavimo pavyzdžius. Efektyvus programinio paketo šablonas reikalauja turėti omenyje problemas, kurios gali nepasirodyti iki tada, kai žymi darbo dalis jau bus atlikta. Projektavimo šablonų pakartotinis taikymas leidžia išvengti subtilių problemų, kurios gali peraugti į dideles problemas. Taip pat yra pagerinamas kodo skaitymas programuotojams ir architektams, kurie jau turi projektavimo šablonų žinių.

Projektavimo šablonai dažniausiai įveda papildomus netiesioginius lygius, kuomet reikia lankstumo. Kai kuriais atvejais tai gali sukelti atlikimo spartos sumažėjimą.

Remiantis projektavimo šablono sąvoka, projektavimo šablonai turi būti iš naujo realizuojami kiekvienoje naujoje situacijoje. Kai kurie autoriai nusprendė, kad toks darbo pobūdis yra žingsnis atgal nuo pakartotinio programinio kodo panaudojimo, kaip yra daroma modulinio programavimo atveju, projektavimo šablonai buvo perdaryti moduliniu pagrindu. Meyer ir Arnout sugebėjo panaudoti arba pilną, arba dalinį komponentų panaudojimą, tais projektavimo atvejais, kuriais jie taikė projektavimo šablonus.

IV. Klasifikavimas

Pradžioje projektavimo šablonai buvo sugrupuoti į kategorijas: kūrimo šablonai, struktūriniai šablonai ir elgesio šablonai.

A. Kūrimo šablonai

- *Abstract factory* – sukuria sąsaja panašios šeimos ar susijusių objektų, jų priklausomybių kūrimui, nenusakant tiksliai visų reikalavimų.
- *Builder* – sudėtingo konstruktoriaus atskyrimas nuo jo pateikimo, leidžiant skirtingus objektus kurti su tuo pačiu konstruktoriumi.
- *Factory method* – objektų kūrimo sąsajos įgyvendinimas, kur dukterinė klasė nusprendžia kokia klasė yra paveldima.
- *Lazy initialization* – taktinis objekto kūrimo, sudėtingo skaičiavimo ar kito sudėtingo proceso paleidimo užlaikymas iki tada, kai jo prireikia.
- *Multiton* – klasės baigtinių egzempliorių užtikrinimas ir globalus jų priėjimas.
- *Object pool* – šablonas nusako, kaip elgtis norint išvengti sudėtingų išteklių priskyrimų ir paleidimo, kuomet jie nėra reikalingi.
- *Prototype* – nusakoma kokio tipo objektus sukurti, naudojantis prototipo egzemplioriumi. Taip pat galima klonuoti objektus, kopijuojant esamą prototipo egzempliorių.
- *Resource acquisition* – šablonas užtikrina resursų atlaisvinimu, kuomet jie nėra reikalingi, surišdamas juos su objektų gyvavimo trukme.
- *Singleton* – vieno objekto egzemplioriaus užtikrinimas. Taip pat pridedamas globalus priėjimas prie duoto objekto.

B. Struktūriniai šablonai

- *Adapter* – sąsajos konvertavimas iš vienos, kurios klientas nesupranta, į tą, kurią jis supranta. Adapteris leidžia objektams bendrauti kartu, kas būtų neįmanoma dėl nesutampančių sąsajų.
- *Bridge* – abstrakcijos priklausomybių (angl. *decoupling*) atskyrimas nuo jos realizacijos, taip sudarant lankstumo sąlygas.
- *Composite* – objektų sudarymas į medžio šakas, siekiant pateikti atskiro modulio ar visos sistemos hierarchiją. *Composite* leidžia klientams vienodai elgtis tiek su vienu objektu, tiek su objekto medžiu.
- *Decorator* – dinaminis papildomos atsakomybės priskyrimas prie objekto, paliekant tą pačią sąsają. *Decorator* yra labai gera alternatyva klasės paveldėjimui.
- *Facade* – subendrintos sąsajos pateikimas sąsajų grupės posistemei. *Facade* apibrėžia aukšto lygio sąsają, kuri leidžia lengvai naudotis posisteme.
- *Flyweight* – efektyvus panašių objektų dalinimas tarp sistemos.
- *Front Controller* – centralizuoto įrašo pateikimas, apdorojant užklausas.
- *Module* – grupė susijusių elementų: klasės, singleton'ai, metodai, naudojami globaliai, iš vieno bendro taško.
- *Proxy* – pakaitalo ar rezervato patekimas objekto kontrolei.

C. Elgesio šablonai

- *Blackboard* – daugelio rašytojų/skaitytojų suteikimas, subendrinant stebėtoją. Duomenų perdavimas atliekamas visos sistemos atžvilgiu.
- *Chain of responsibility* – Vengimas siuntimą priskirti kažkokiam vienam objektui. Keliama galimybė keliems objektams apdoroti ateinančią informaciją.
- *Command* – užklausos yra pateikiamos kaip uždari objektai, kas leidžia klientams pateikti skirtingas užklausas, eiles ar žurnalą.
- *Interpreter* – turint kalbą, apibūdinti jos žodyną kartu su interpretatoriumi, kuris naudoja žodyną ir supranta kalbos sakinius.
- *Iterator* – pateikti sprendimą, kaip galima pateikti visus esamo objekto elementus, nesigilinant į objekto vidinę struktūrą.
- *Mediator* – nurodyti objektą, kuris reguliuoja kaip objektų grupę tarpusavyje sąveikauja. *Mediator* skatina laisvą jungimą, taip išvengiant tiesioginį vieno objekto priklausomybę nuo kito objekto, kurie priklauso *mediator* objektui.
- *Memento* – leidimas objekto būsenos atstatymui uždelstam laike, nepažeidžiant objekto priklausomybių.
- *Null object* – null tipo vengimas, priskiriant reikšmę kūrimo metu.
- *Observer* – apibūdinti vienas-daugelis sąryšį, kur vieno objekto atnaujinimas skatina jam priklausančių objektų automatinį atnaujinimą.
- *Servant* – objektų grupės bendro funkcionalumo aprašymas.
- *Specification* – loginiais ryšiais jungiama verslo logika.
- *State* – leidimas objektui keisti jo elgesį, pasikeitus jo vidiniai būsenai. Objektas gali pakeisti savo klasę.
- *Strategy* – algoritmo šeimos apibūdinimas, vieno kito paveldėjimas ir priklausomybė. *Strategy* leidžia keisti algoritmus, priklausomai nuo kliento kreipinio.
- *Template method* – operacijos skaldymas į dukterines klases, pradžioje aprašant tik algoritmo griaučius. Metodas leidžia dukterinėms klasėms iš naujo deklaruoti tam tikras operacijas iš naujo, nekeičiant bendro algoritmo logikos.
- *Visitor* – pateikia operacijas, kurios yra įvykdomos su objekto elementais. *Visitor* leidžia pateikti naują operaciją, nepakeičiant jo operuojamų elementų klases.

D. Lygiagretumo šablonai

- *Active Object* – atskiria metodo iškvietimą nuo metodo kreipinio, kuris egzistuoja kontroliuojamoje šakoje. Tikslas yra pateikti lygiagretumą, naudojant asinchroninius kreipinių iškvietimą ir kontroliuojamą užklausų apdorojimą.
- *Balking* – objekto operacija kviečiama tik tuomet, kai objektas yra tam tikroje būsenoje.
- *Binding properties* – priverstinis savybių sinchronizavimas tarp stebėtojų, juos sujungiant.
- *Double-checked locking* – lengvinamas užrakto gavimas, pirmiausia bandant uždarymo kriterijų nesau-

gioje būsenoje. Jeigu bandymas sėkmingas – resursas užrakinamas. Toks šablonas gali būti nesaugus, todėl jį galima traktuoti kaip priešpriešinis šablonas.
- *Event-based asynchronous* – problemų adresavimas ne sinchroniniam režime. Nutinka lygiagrečiose programose.
- *Guarded suspension* – valgo veiksmus, kuriems riekia rakto ir tam tikros sąlygos, norint jas įvykdyti.
- *Lock* – vienos gijos rakto uždėjimas ant tam tikro resurso. Tokiu atveju kitos gijos negali prieiti prie resurso, kol nebus paleistas raktas.
- *Messaging* – leidžia informacijos perdavimą tarp sistemos modulių ir komponenčių.
- *Monitor object* – objekto metodų kontroliuojamas paleidimas. Taip išvengiama kelių objekto metodų kreipinių lygiagrečioje sistemoje.
- *Reactor* – Rektoriaus objektas suteikia ne sinchroninį priėjimą prie duomenų, kai juos reikia pateikti sinchroniškai.
- *Read-write lock* – suteikti galimybę skaityti iš objekto laisvai, tačiau rašymo operacijai reikalingos ypatingos teisės.
- *Scheduler* – labai griežtas vienos gijos metodų paleidimas, kelių gijų skaitymo atveju.
- *Thread pool* – sukuriamas tam tikras gijų skaičius, organizuotas į eilę, operacijai atlikti. Specialus *Object pool* atvejis.
- *Thread-specific storage* – statinės ar globalinis priėjimas prie lokalių gijos duomenų.

V. Kūrimo šablonai

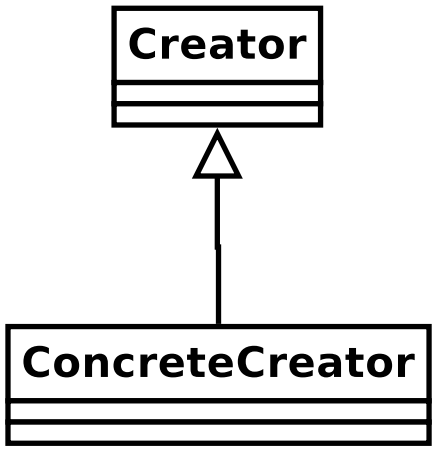
A. Kūrimo šablonas

Programų inžinerijoje, kūrimo šablonai yra specialūs projektavimo šablonai, kurių paskirtis yra kontroliuoti objektų sukūrimą, priklausomai nuo tam tikrų aplinkybių. Paprasčiausias objekto sukūrimas gali privesti prie projektavimo problemų ar pridėti labai daug nereikalingo kompleksiškumo. Kūrimo šablonai sprendžia šitą problemą, pateikiant kontrolės mechanizmus objekto kūrimui.

Kūrimo šablonai susideda iš dviejų pagrindinių idėjų:
- Sistemos naudojamų klasių sąrašo pateikimas
- Slėpti sistemos naudojamų modelių kūrimo ir jų sujungimo logiką

Kūrimo šablonai gali būti suskirstyti toliau į objekto kūrimo šablonus ir klasės kūrimo šablonus. Objekto kūrimo šablonai daugiau orientuoti objektui priklausančių objektų kūrimą, o klasės kūrimo šablonai rūpinasi objekto kūrimui dukterinėse klasėse.

Penki labiausiai žinomi projektavimo šablonai yra:
- *Abstract factory*, kuris leidžia kurti objektus, nesirūpinant objekto priklausomybėmis.
- *Builder* šablonas, kuris atskiria sudėtingo projekto konstrukciją nuo jo atvaizdavimo.
- *Factory method*, kuris leidžia nesirūpinti dukterinės klasės priklausomybėmis.
- *Prototype*, kuris nusako kuriamo objekto tipą per prototipą.



1 pav.. Klasės kūrimo projektavimo šablono diagrama

- *Singleton* šablonas, kurio uždavinys yra palaikyti tos pačios klasės vieną egzempliorių.

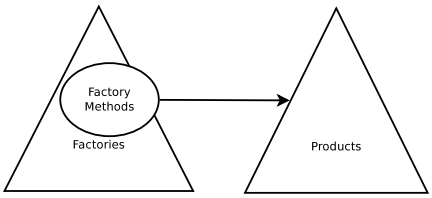
1) Apibrėžimas: Kūrimo šablonų tikslas yra atskirti sistemą nuo objekto kūrimo proceso. Jie suteikia sistemai daugiau lankstumo ir neriboja kada, kur ir kaip objektai turi būti sukurti.

2) Naudojimas: Šiuolaikinės programinės įrangos vystymas vis labiau priklauso nuo objektų kūrimo, negu nuo klasės paveldėjimo, apibrėžimas stumiamas toliau nuo sunkaus programavimo įpročių iki sistemos skaldymo į mažesnius objektus, kuriuos galima sudėti į vieną kompleksinį objektą. Sunkus programavimas priveda prie mažo sistemos lankstumo, kadangi tokiu atveju daugelyje vietų tenka kopijuoti tą patį kodą, kurį gal būt galima būtų panaudoti iš vienos vietos. Taip pat mažėja kodo suprantamumas, kodą tampa sunkiau pakeisti. Klaidos atveju sunkiau yra rasti kur būtent sistemoje įvyko klaida. Dėl tokių priežasčių, kūrimo šablonai yra žymiai pranašesni, už sunkaus programavimo įpročius.

Kūrimo šablonus rekomenduojama naudoti sekančiose situacijose:
- Sistema turi būti nepriklausoma nuo sistemoje dalyvaujančių objektų
- Grupė objektų, kuri turi dirbti kartu
- Klasės bibliotekos metodų slėpimas. Pateikti tik sąsaja su biblioteka
- Skirtingų atvaizdavimų sudėtingų objektų kūrimo
- Dukterinės klasės įgyvendinimas pagrindinio objekto kūrimo metu
- Klasės atvaizdo formavimas veikimo metu
- Tam tikros klasės vieno atvaizdo užtikrinimas
- Atvaizdas turi būti plečiamas, nekeičiant pagrindinės klasės struktūros.

3) Struktūra: Pavyzdyje 1 yra pateikiama bendra kūrimo šablono struktūra.

Šablonas susideda:
- *Creator*: sukuria objekto atvaizdą, grąžindamas objektą
- *ConcreteCreator*: įgyvendina objekto atvaizdą



2 pav.. Abstract factory projektavimo šablono struktūra

B. Abstract factory šablonas

Abstract factory šablonas yra projektavimo šablonas, kuris leidžia apimti skirtingus objektus su skirtingais *factory* į vieną objektą. Standartiniu atveju, programuotojas įgyvendina *abstract factory* ir paskui naudoja bendrą sąsają objektams kurti. Tokiu atveju programuotojui nereikia žinoti specifinių reikalavimų objektui sukurti. Objektai yra sukuriami per *factory*, ir būtent *factory* pasirūpina objektų reikalavimais. Šablonas atskirai objekto kūrimo specifikas nuo objekto naudojimo specifikos.

1) Apibrėžimas: Bendrinis apibrėžimas gali būti aprašytas taip – sąsajos suteikimas, kurios pagalba galima kurti objektus, nežinant jų kūrimo specifikos.

2) Naudojimas: Factory nusako kokie būtent yra reikalavimai objekto sukūrimui ir būtent šioje vietoje objektas yra sukuriamas. Klientui yra grąžinamas to objekto abstrakti nuoroda. Taip yra išvengiama kliento pusėje kurti iš *factory* metodo gražinamo objekto kopiją. Taip pat, klientas taip visiškai nežino kokie yra sukurto objekto posistemėje esantys objektai.

Per *factory* sukuriamų objektų posistemės papildymas yra labai lengvas uždavinys. Klientas visiškai gali nežinoti ar sukuriamo objekto posistemės reikalavimai pasikeitė bet kokiu laiko momentu. Taip labai sumažėja laiko sąnaudos objekto modifikacijomis.

3) Struktūra: Šablono struktūra pateikiama 2 pav. *Factories* yra aprašomi visi *factory methods*, kurie sukuria naujus objektus, *products*.

C. Builder šablonas

Šablono tikslas yra objekto kūrimo reikalavimų skaldymas į žingsnius. Taip objekto kūrimas gali būti skirtingas, priklausomai nuo praeitų žingsnių, to objekto kūrimo metu.

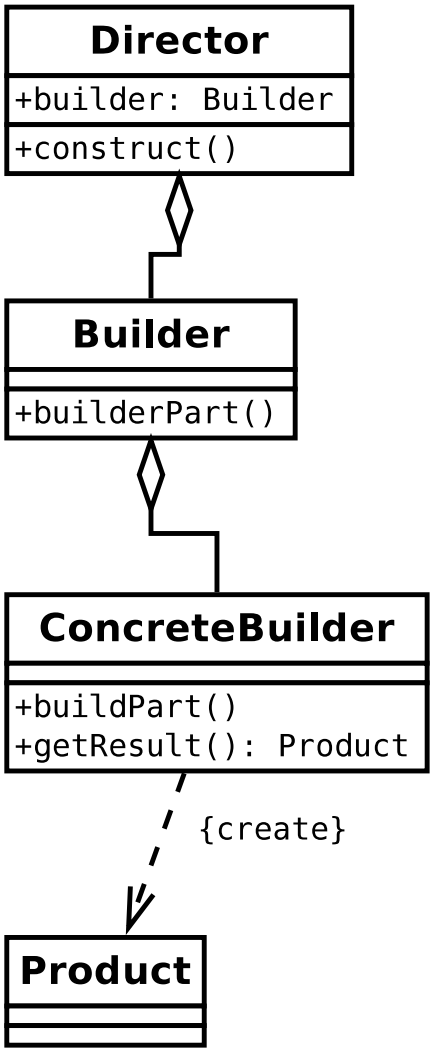
1) Apibrėžimas: Builder šablono tikslas yra atskirti objekto pateikimą, nuo kompleksinio objekto kūrimo. Tai įgyvendinus, kiekvienas kuriamas objektas gali būti skirtingas, priklausomai nuo kūrimo proceso.

2) Naudojimas: Šablonas yra dažnai taikomas, kuomet egzistuoja labai panašios objektų grupės, kurios skiriasi tik tam tikrų objekto parametrų būsena.

3) Struktūra: Šablono struktūra pateikiama 3 pav. *Builder* suteikia abstrakčią sąsaja objekto kūrimui. *Concrete Builder* suteikia *Builder* realizaciją. Tai yra objektas, kuris geba kurti kitus objektus.

D. Factory method šablonas

1) Apibrėžimas:



3 pav.. Builder projektavimo šablono struktūra

2) Naudojimas:

3) Struktūra:

E. Lazy initialization šablonas

1) Apibrėžimas:

2) Naudojimas:

3) Struktūra:

F. Multiton šablnas

1) Apibrėžimas:

2) Naudojimas:

3) Struktūra:

G. Object pool šablonas

1) Apibrėžimas:

2) Naudojimas:

3) Struktūra:

H. Prototype šablonas

1) Apibrėžimas:

M. State šablonas

1) Apibrėžimas:
2) Naudojimas:
3) Struktūra:

N. Strategy šablonas

1) Apibrėžimas:
2) Naudojimas:
3) Struktūra:

O. Template method šablonas

1) Apibrėžimas:
2) Naudojimas:
3) Struktūra:

P. Visitor šablonas

1) Apibrėžimas:
2) Naudojimas:
3) Struktūra:

VIII. Lygiagretūs šablonai

A. Lygiagretus šablonas

1) Apibrėžimas:
2) Naudojimas:
3) Struktūra:

B. Active Object šablonas

1) Apibrėžimas:
2) Naudojimas:
3) Struktūra:

C. Balking šablonas

1) Apibrėžimas:
2) Naudojimas:
3) Struktūra:

D. Messaging šablonas

1) Apibrėžimas:
2) Naudojimas:
3) Struktūra:

E. Double-checked šablonas

1) Apibrėžimas:
2) Naudojimas:
3) Struktūra:

F. Asinchroninio metodo kreipimo šablonas

1) Apibrėžimas:
2) Naudojimas:
3) Struktūra:

G. Saugomas sulaikymas

1) Apibrėžimas:
2) Naudojimas:
3) Struktūra:

H. Uždarymas

1) Apibrėžimas:
2) Naudojimas:
3) Struktūra:

I. Stebėjimas

1) Apibrėžimas:
2) Naudojimas:
3) Struktūra:

J. Reactor šablonas

1) Apibrėžimas:
2) Naudojimas:
3) Struktūra:

K. Scheduler šablonas

1) Apibrėžimas:
2) Naudojimas:
3) Struktūra:

L. Thread pool šablonas

1) Apibrėžimas:
2) Naudojimas:
3) Struktūra: