УНИВЕРЗИТЕТ У БЕОГРАДУ

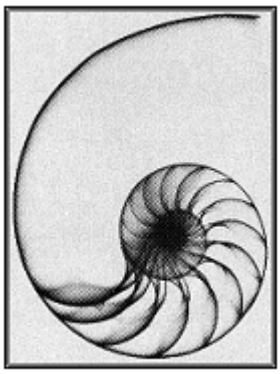
ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

Катедра за софтверско инжењерство –
Лабораторија за софтверско инжењерство

СОФТВЕРСКИ ПАТЕРНИ

РАДНИ МАТЕРИЈАЛ СА ПРЕДАВАЊА

Аутор: Др Синиша Влајић доц.



Београд - 2011.

docsity.com

Садржај

docsity.com

1. Увод

Једног лепог дана давне 2000-те године, мој тадашњи шеф проф. др Видојко Ћирић, донео ми је из Канаде књигу Design Patterns и рекао ми је: "*Ова књига је број један данас на западу, погледај је, можда ћеш у њој наћи нешто интересантно за твој докторат!* ". Узео сам књигу, не схватајући да узимам нешто што ће у значајној мери да утиче на моје стручне и животне погледе и ставове. Тада сам се први пут упознао са појмом патерна и једноставно они су постали део неке моје животне приче.
Искрен да будем, када сам први пут прочитао ту књигу, био сам у некој врсти полу-кошмара. Осећао сам да се ту дешава нешто веома битно али нисам могао да схватим шта је то. После другог, трећег, не знам ни сам ког читања, коначно је почело "*да ми свиће*". Схватио сам шта је суштина, и у следеће три године радио сам докторат у коме сам се бавио математичком формализацијом патерна. Идеја доктората се односила на схватање патерна у општем смислу и његова математичка формализација. У то време једино се Амнон Еден озбиљно, у светским оквирима, бавио формализацијом патерна. Међутим Еден се бавио појединачном формализацијом патерна пројектовања. Тај приступ ми се није свидео, јер нисам схватио шта добијам тиме ако патерн, опишем математичким формализмом а не програмским кодом. Еден је то радио како би у програмском коду препознао да ли постоји неки патерн пројектовања. Тада сам се питао, као што се и сада питам: "Шта се тиме добија?". Можда грубо звучи, али рекао бих ништа специјално. Шта мени значи ако констатујем да се у програму налази нпр. state или visitor патерн. Мене интересује да ли у програму постоји патерн. Није ми важно како се он зове. Такав начин размишљања ме је водио препознавању и схватању општих особина које има сваки патерн. Битно је препознати места у програмском коду, где се могу уградити те особине, односно препознати места где се уграђују (укључују) патерни...

...

Са ове дистанце могу одговорно да кажем да су се патерни показали као веома користан и оперативан концепт у развоју одрживих софтверских система.

2. Основе о патернима

Сврха патерна и њихово место у процесу развоја софтвера

Патерни или узори (како смо их превели) имају за циљ да нам помогну у одржавању и надоградњи софтверског система. У основи сваког софтверског система налази се нека архитектура. Архитектура се у најопштијем смислу састоји од компоненти које су између себе повезане преко њихових интерфејса. Постоји макро и микро архитектура. **Макро архитектуру** је реализована нпр. преко **ECF** (Enterprise Component Framework) или **MVC** (Model-View-Controler) патерна, док је **микро архитектура** реализована преко узора пројектовања и то: **креационих**, **структурних** и **патерна понашања**. Поред наведених патерна који покривају пре свега фазу пројектовања у развоју софтверског система, постоје и други патерни који покривају и друге фазе у развоју софтвера, као што су патерни захтева, патерни анализе и имплементациони патерни (идиоми) који су везани за конкретне технологије, као што су нпр. Јава или C#.
Циљ овога предмета је да студенти у општем смислу схвате, шта су патерни, независно од технологије у којој ће они да буду реализовани. Патерни пројектовања су независни од технологије, у којој ће бити имплементирани, тако да су они погодни да преко њих схватимо патерне у општем смислу.

Шта је патерн

Када говоримо о патернима и узорима тада о њима у најопштијем смислу можемо да кажемо да они представљају *решења неког проблема, у неком контексту, који се може поново искористити за решавање неких нових проблема*.

docsity.com

То значи да три елемента дефинишу патерн: проблем, решење и контекст. Контекст у суштини дефинише ограничења која морају бити задовољена када се решава неки проблем.

Поновна употребљивост патерна

Једно од основних својстава патерна јесте његова могућност да се може применити у решавању различитих проблема. Како се долази до наведеног својства, или је можда још прецизније питање, *како треба пројектовати софтверску компоненту да се она може применити за различите проблеме*?
Када се направи нека компонента, која решава неки проблем, потребно је направити допунски напор, како би се дошло до општијег или универзалнијег решења које неће бити примењиво само за један проблем, већ ће бити примењиво за један скуп или једну класу проблема.
Шта је оно, чему ми у суштини тежимо када развијамо софтверски систем?
Идеја је у томе, да ми направимо такве софтверске системе или софтверске производе, који ће се лако прилагодити сваком новом корисничком захтеву. То значи да ћемо без велике промене постојеће структуре и понашања софтверског система, моћи да додамо нову или променимо постојећу функционалност софтверског система сходно новим корисничким захтевима.
Идеално би било да имамо параметризован софтверски систем, који се може прилагодити (кастомизовати) различитим доменима проблема. Домен проблема се описује са неким скупом вредности, којима се параметри софтверског система иницијализују, пре покретања софтвера. Треба нагласити да промена вредности наведених параметара не тражи промену програмског кода софтверског система.

Књига Design Patterns

Патерне у општем смислу ћемо објаснити, коришћењем патерна пројектовања. Патерни пројектовања су своју афирмацију доживели са књигом Design Patterns, коју су написали Gamma и група аутора. Ова књига је једна од десет најцитиранијих књига у области софтверског инжењерства и она је поставила темеље схватању улоге патерна у фази пројектовања софтверског система. Ова књига представља аксиом или полазиште приче о патернима.
Књига Design Patterns је направила, по мом скромном мишљењу, кључни искорак у правцу прављења одрживих софтверских система. Када кажем одрживи софтверски системи, онда мислим на софтверске системе који се могу лако одржавати и надограђивати. Због свега тога, топло препоручујем да набавите и прочитате наведену књигу. Ја ћу се потрудити на предавањима, да вам из наведене књиге, објасним све оно, што мислим да је битно, како би схватили кључне идеје и концепте који се налазе у основи патерна пројектовања.

Примењивост патерна у различитим сферама живота

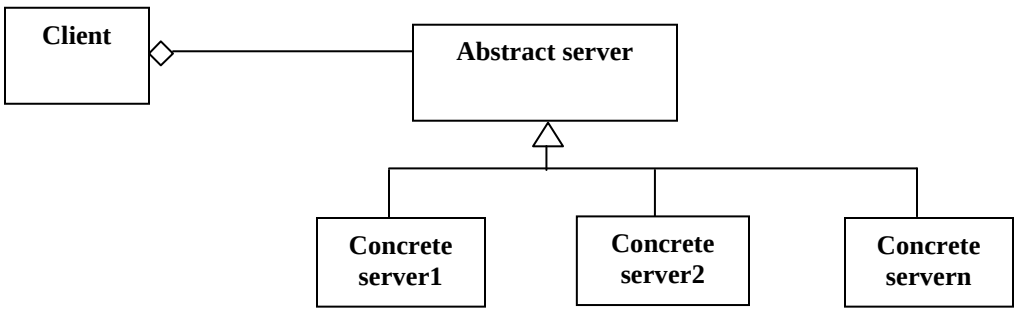
Ако посматрате живот, он се састоји од много процеса. Уколико не желимо да нам живот буде хаотичан, ми треба да управљамо са свим тим процесима. Патерни помажу да се лакше управља различитим животним процесима. Коришћењем патерна се лакше решавају, прате и управљају разни животни процеси. *Шта то значи?*
Када се деси неки проблем, тада се обично нађе неко решење. Поставља се следеће питање: *Шта треба урадити са решењем?* Треба одвојити мало времена и направити од решења, које је обично неко специфично решење, генеричко решење. Тиме добијате могућност да то генеричко решење примените за различите проблеме који се могу десити у животу. Генеричко решење, за разлику од специфичног решења, је непроменљиво у току развоја програма.
Минимално што треба урадити јесте да се опише решење, макар оно било и специфично. Када се у будућности деси неки нови сличан проблем, ви ћете препознати проблем који сте већ решили и потражићете постојеће решење. Оно не мора у потпуности да реши ваш проблем, али сигурно се из тог решења може пронаћи доста делова који се могу користити и код неког новог проблема. Уколико се не забележи решење, тада ћете више пута у животу решавати један те исти, или сличан проблем, сваки пут изнова. На тај начин ћете потрошити пуно више времена и енергије, него да сте запамтили решење и користили га као помоћ у решавању неког новог проблема. Десиће се после годину, две неки *déjà vu* (дежа ви), неки веома познат проблем, који знате да сте решавали, али нећете моћи да се сетите како сте га решили. Или ћете се присећати решења, што је прилично

docsity.com

непоуздано и непрецизно. Тако ћете опет изгубити много сати, можда и дана у решавању нечега што сте већ решили. Решења треба по могућству јасно и прецизно написати како би она могла поново да се користе. Нама је циљ да коришћењем патерна са што мање утрошеног времена и енергије, у дужем временском периоду, постигнемо што је могуће већи ефекат.
Прављење генеричких решења дуже траје него што је то случај код специфичних решења. Међутим ефекат тога јесте да ће се нови проблеми решавати све лакше и лакше. Да нагласим, патерни у себи садрже потенцијал да могу да реше скуп проблема или класу проблема а не само један специфичан проблем.

3. Општи облик патерна

У књизи Design Patterns постоје 23 GOF (Gang of Four) патерна пројектовања. Они су подељени у три основне групе: креационе патерне, патерне структуре и патерне понашања. Када се посматра 20 од тих 23 патерна, може да се примети једна структура[1] која постоји код сваког од тих патерна. Та структура у потпуности описује патерн или неки његов део. Наведена структура је **кључни механизам или својство** патерна пројектовања. Управо та структура омогућава лако одржавање и надоградњу програма. Та структура изгледа овако:

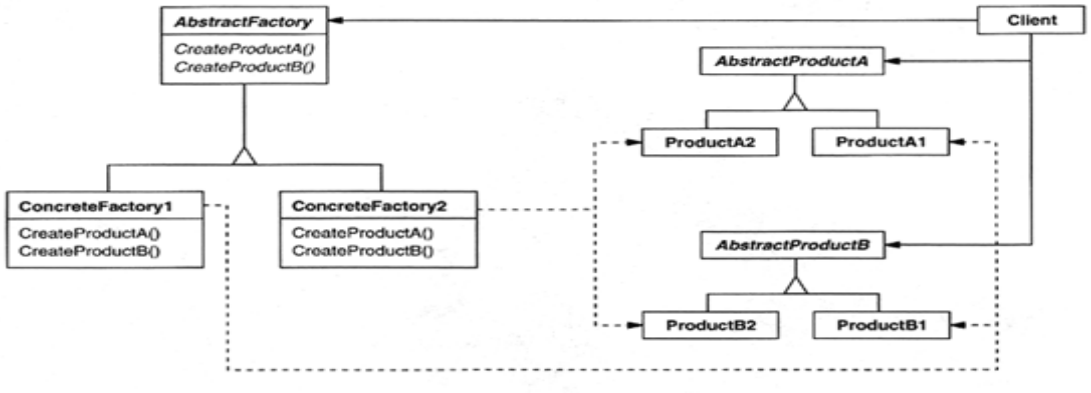


Клијент је везан за апстрактни сервер, док су из апстрактног сервера изведени различити конкретни сервери. У време компајлирања програма клијент се везује за апстрактни сервер[2]. То значи да ће се тек у време извршења програма разрешити који конкретни сервер ће да реализује захтев клијента. Управо ово повезивање клијента са конкретним сервером у време извршења програма, даје самом програму флексибилност, да један клијентски захтев може бити реализован на различите начине, преко различитих сервера. Такође додавање новог конкретног сервера неће променити клијента.

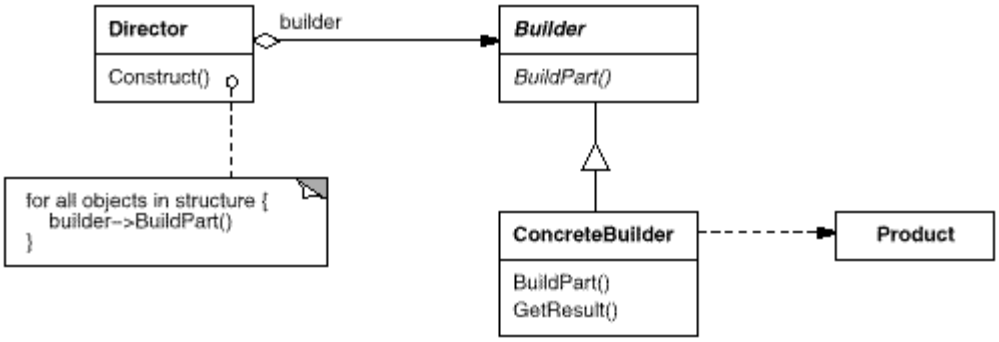
Навешћу неке од патерна чија структура садржи у потпуности или неким делом наведену *одрживу структуру:*
Abstract factory патерн има следећу структуру:

[1] Наведена структура омогућава да софтверски систем буде одржив. Због тога ћемо наведену структуру назвати: **одржива структура (viable structure)**
[2] Апстрактни сервер, уколико програмски посматрамо, може бити интерфејс (interface) или апстрактна класа (abstract class).

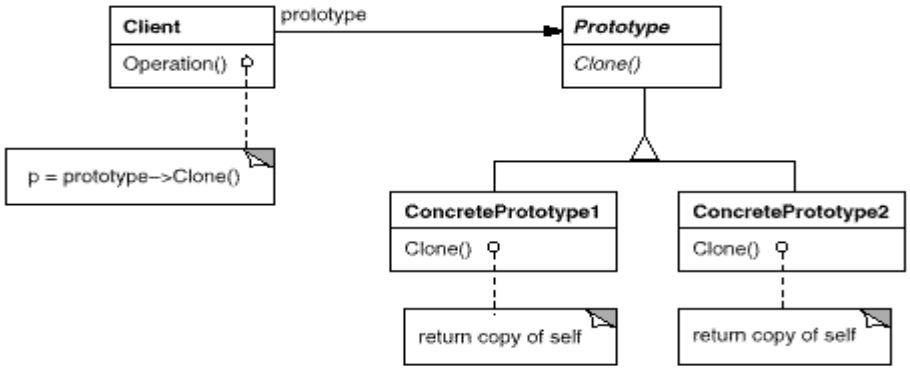
docsity.com



Builder патерн има следећу структуру:

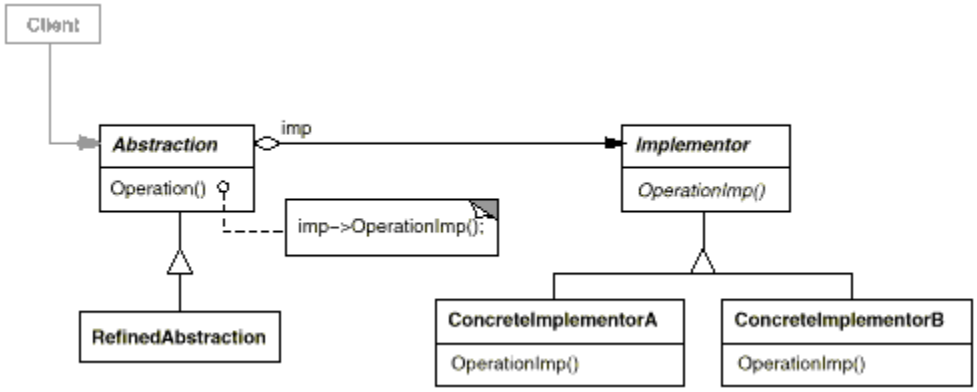


Prototype patern има следећу структуру:

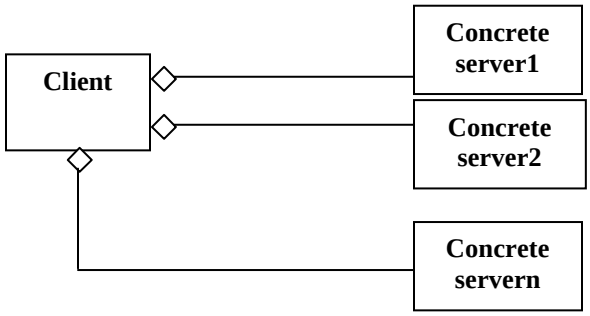


docsity.com

Bridge патерн има следећу структуру:



Наведена одржива структура решава проблем структуре[3] код које је клијент директно повезан са конкретним серверима:



Наведена структура је тешка за одржавање јер се при додавању новог конкретног сервера мења клијент. Такође, клијент се у време компајлирања програма везује за конкретни сервер, што онемогућава флексибилност програма у току његовог извршавања.

Да би смо схватили патерн у општем смислу потребно је да извршимо малу анализу постојећих дефиниција патерна.

Christopher Alexander је рекао[4]:"*Сваки патерн описује **проблем** који се јавља изнова (непрестано) у нашем окружењу, а затим описује суштину **решења** тог проблема, на такав начин да ви можете користити ово решење милион пута, а да никада то не урадите два пута на исти начин.*"

Из наведене дефиниције се може приметити да патерн има два важна дела: проблем и решење. Такође се може видети да патерни имају особину поновљивости, што значи да се решење неког проблема може поновити више пута код другиh, различитих проблема.

Александар је такође рекао[5]: "*Патерн је, у најкраћем, у исто време **ствар** која се дешава у свету и **правило** које нам говори како се креира та ствар и када се креира та ствар, то је у исто време и*

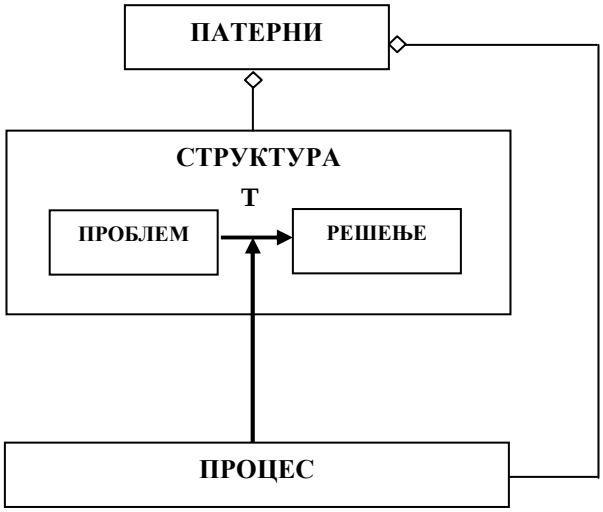
[3] Наведену структуру ћу назвати: **неодржива структура (unviable structure)**
[4] *"Each pattern describes a **problem** which occurs over and over again in our environment, and then describes the core of the **solution** to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.*[AIS]"

docsity.com

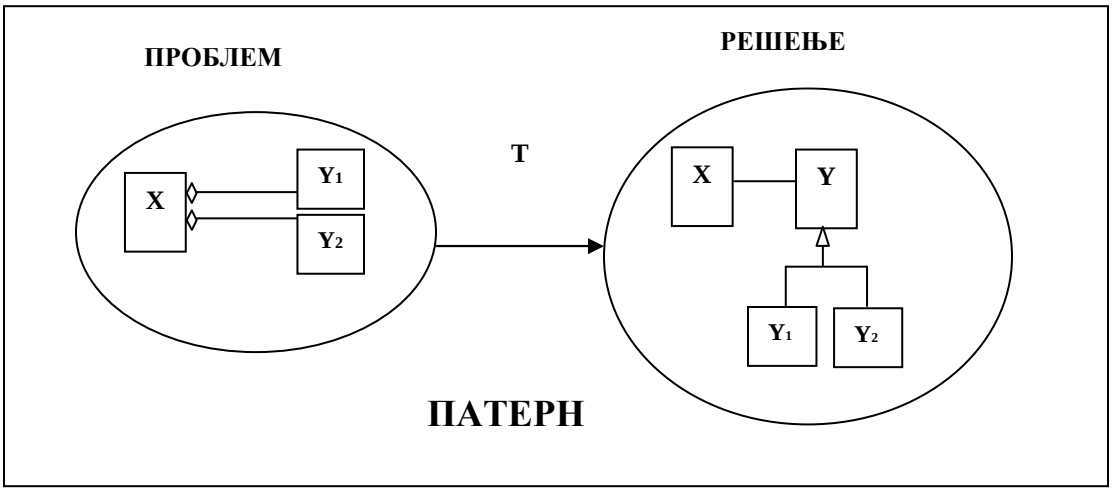
процес *и ствар; описује се ствар која је жива (активна) и описује се процес који генерише ту ствар.*"

Jim Coplien [Cop1], је такође објашњавајући патерне рекао[6]: "… *то је правило за израду ствари, али је такође, са много поштовања, и сама ствар* ."

Наведени експерти за патерне нам говоре да је патерн у исто време и структура (thing) и процес. Патерн има структуру проблема и решења. Патерн је процес који објашњава **када** и **како** се креира структура решења из структуре проблема.



Патерн би могао да се представи у најопштијем смислу на следећи начин:



Као што смо рекли, проблем и решење имају неку структуру. За структуру проблема смо рекли да је "неодржива структура", док је структура решења "одржива структура" .

---

[5] *The pattern is, in short, at the same time a **thing**, which happens in the world, and the **rule** which tells us **how** to create that thing, and **when** we must create it. It is both a **process** and a **thing**; both a description of a thing which is alive, and a description of the process which will generate that thing*

[6] *"…it is the rule for making the thing, but it is also,in many respect, the thing itself."*

docsity.com

Процес описује како се структура проблема трансформише (T) у структуру решења:

T
проблем --------> решење

То у суштини значи да је *патерн процес трансформације структуре проблема у структуру решења*. Такође можемо рећи да је *патерн процес који трансформише "неодрживу структуру" у "одрживу структуру"*.

Поставља се питање, када се дешава тај процес?

Проблем је, ако би говорили из перспективе развоја софтвера, неки програм који има структуру која је тешка за одржавање и надоградњу.
Када се деси да је структура програма (проблем код патерна) тешка за одржавање и надоградњу, тада се прави нова структура програма (решење код патерна) која је лакша за одржавање и надоградњу.
То значи да се наведени процес трансформације структуре проблема у структуру решења дешава када је структура проблема тешка за одржавање и надоградњу (то је услов који одређује када се дешава трансформација). Ми смо покушали у нашој теорији да формализујемо наведену трансформацију проблема у решење како би схватили шта је суштина патерна[7].

Где треба поставити патерне код софтверских система
Када се нађе решење неког проблема треба препознати шта је у решењу опште а шта специфично, односно шта је непроменљиво а шта је променљиво. Наведена констатација се директно односи на писање програмског кода у току имплементације софтверског система. Уколико се препознају она места у софтверском систему, која су променљива и која се непрекидно мењају са новим корисничким захтевима, ту треба поставити патерне како би се зауставио потенцијални хаос који та променљива места у програму могу да направе. Та места, или те тачке, су по аналогији једнаке бифуркационим тачкама у теорији хаоса. Бифуркационе тачке воде систем у потпуни хаос, јер се у њима нагомилавају различитости. Нешто слично се може десити код развоја софтверских система. Уколико се правовремено не препознају "бифуркационе тачке" софтверског система, систем може да почне ортогонално да развија своју сложеност у односу на постојећи софтверски систем. Та ортогонална сложеност може да потпуно обори постојећи софтверски систем. Патерни се постављају у наведеним тачкама како се не би дозволило да софтверски систем оде у хаос. Ко развија софтвер, треба да има знање, искуство и осећај, да препозна те бифуркационе тачке софтверског система. Ко то препозна, он ће пре свој систем довести у ред и омогућиће да се систем лако одржава и надограђује.
"Шпагети код", појам познат из софтверског инжењерства, где имате више грана у оквиру једне од метода (више if услова), представља пример "бифуркационе тачке" код софтверских система. Поставља се питање, *зашто се дешава "шпагети код"*?
Сваки нови проблем се решава новог граном у оквиру постојеће методе. То доводи то тога да имамо методе које су велике и које се тешко могу одржавати.
Када приметите да се нешто овако дешава у вашем програму (кога ћемо преставити једном линијом), ви у суштини препознајете, тачке хаоса, које теже да сруше ваш програм. У једном тренутку, те тачке хаоса, са њиховом тенденцијом развоја сложености, се ортогонално развијају у односу на ваш програм. У тим тачкама се дешавају непрекидне промене. Те тачке се непрекидно "кувају" и "расту". Ако у тим тачкама поставите патерне, ви ћете зауставити хаос у процесу развоја вашег софтверског система. Уколико не зауставите хаос, вероватно ће, у неком тренутку да вам буде јефтиније да развијате нови софтверски систем, него да одржавате постојећи софтверских систем.
Како препознати тачке које воде софтверски систем у хаос?

[7] На конференцији ASC2011 на Криту у јуну 2011 год. смо изложили рад под насловом: *The Explanation of the Design Patterns by the Symmetry Concepts*, где смо формално коришћењен симетријских концепата објаснили општи облик патерна пројектовања.

docsity.com

Прате се кориснички затеви и њихов утицај на програмски код. Ако приметите да се неки делови програмског кода непропорционално развијају у односу на друге делове програма, то може да буде показатељ да су то потенцијалне тачке хаоса у које треба поставити патерн.
Више грана у програму не значи аутоматски да ту треба поставити патерн. Ако се број грана не мења у току развоја и одржавања програма, тада не треба уводити патерн, јер патерни повећавају сложеност софтверског система.

Раздвајање генералних од специфичних делова програмског кода
Како долази до тога да патерн представља опште решење за класу проблема? Када правимо неки софтверски систем или у ужем смислу неку софтверску компоненту, то радимо на основу неког корисничког захтева. Софтверски систем настаје као резултат процеса развоја софтвера, који пролази кроз све фазе развоја софтвера, која започиње дефинисањем корисничких захтева а завршава се имплементацијом софтверског система. Софтверски систем који је направљен, односно прва верзија софтвера, обично у себи садржи измешане генералне и специфичне делове програмског кода. Генерални делови се могу користити не само за решавање текућег проблема, већ и за решавање неких других проблема. Специфични делови програмског кода су везани за текући проблем и они се не могу користити за неке друге проблеме. Можемо да кажемо да се генерални (***gen***) и специфични (***spec***) делови програмског кода налазе у једном модулу (***ModulA***), односно у једној логичкој целини програма:

ModulA = (gen + spec)

Наведена измешаност генералних и специфичних делова програма, који се налазе у једном модулу, у суштини представља проблем уколико би покушали да се наведени програм користи у решавању неких других проблема. Због тога се временом, наведени генерални и специфични програмски код раздвајају и постављају се у различите модуле (***ModulB*** и ***ModulC***):

ModulB = (gen)
ModulC = (spec)

Наведени процес, у слободном облику, би могли да представимо на следећи начин:

Lim modulA = modulB + modulC
t-->∞

односно,

Lim (gen + spec) = (gen) + (spec)
t-->∞

На основу наведеног можемо да закључимо да је крајњи циљ или сврха процеса развоја софтвера, изградња софтверског система код кога ће бити одвојени генерални од специфичних делова програмског кода[8]. На тај начин је омогућено да се генерални делови кода могу користити за класу проблема, при чему се специфични делови кода прилагођавају различитим проблемским ситуацијама.

...

Када развијате софтвер, ви ћете временом, свесно или несвесно ићи у смеру раздвајања генералног од специфичног кода. То је тенденција сваког искусног програмера који иза себе има много развијених софтверских система. Ова предавања и концепт патерна имају за циљ да схватите који је то механизам, који сваки програмер временом схвати, али је велико питање које је то време (које

[8] Код *Јединственог процеса развоја софтвера (Unified Software Development Process)* која представља методу развоја софтвера јасно су раздвојени генерални од специфичних слојева код архитектуре софтверског система.

docsity.com

може ићи од неколико месеци до неколико година) за које ће он то схватити. Ми желимо да вам помогнемо, да пре него што постанете искусни програмери, знате концепт патерна, који ће вам пуно помоћи да схватите суштину одрживих софтверских система.

Да поновимо шта је суштина наведене приче: - У почетку имамо програм код кога су измешани генерални и специфични делови програмског кода, који се налазе у једном модулу. То значи да имамо програм код кога су измешани различити нивои апстракције[9].Како пролази време и како се појављују нови кориснички захтеви, генерални и специфични делови програма се раздвајају и смештају у различите модуле. Идеално би било да софтверски систем има само генералне делове програмског кода, док би специфични делови програмског кода требало да буду, "потиснути" из програма и смештени у датотеке односно табеле базе података. Тако би дошли до параметризираног софтверског система, који се прилагођава (кастомизује) различитим проблемима, променом датотеке или табеле (у којој се налазе параметри) који се иницијализује са вредностима којима се дефинише специфичан проблем[10]. То значи да уколико се деси нови проблем, улази се у наведене датотеке и табеле, и оне се пуне са вредностима који дефинишу тај проблем. Наглашавам оно што је генеричко, то је непроменљиво и на тај део програмског кода не утичу нови проблеми.
Шта добијамо као резултат коришћења патерна?
Патерни омогућавају да генерални и специфични делови софтверског система буду раздвојени.

Недостаци софтверског система при увођењу патерна
Увођењем патерна, повећава се сложеност софтверског система и смањује се брзина извршења програма јер се уводе нови слојеви[11] и нивои[12] у архитектури софтверског система[13]. Поред тога, увођење нивоа и слојева у архитектури отежава тестирање и контролу извршења програма(debug програма). Овај проблем је посебно наглашен када су различити нивои и слојеви архитектуре реализовани различитим технологијама. Као што се види, патерни обарају неке перформансе софтверског система, али са друге стране повећавају лакоћу одржавања и надоградње софтверског система. Наведени проблеми су пре свега технички проблеми и сматрам да ће током времена брзина рачунара бити довољно велика да се неће приметити значајна разлика између апликација које имају различит број нивоа и слојева.

Објашњење механизма одрживе структуре програма

Када смо правили математички формализам за опис патерна пажњу смо усмерили на то да формално опишемо патерне, односно да направимо језик којим ћемо описати патерне. У основи тог формализма су се налазили ***симетријски концепти*** и то симетријска трансформација и симетријска група. Они су нам у суштини помогли да схватимо који је то механизам у патернима који омогућава да структура програма буде одржива. Математички смо показали да сваки пут када се појави несиметријска група коју образују клијент и конкретни сервери), што представља проблем код патерна, уводе се две релације које представљају решење код патерна:
а) симетријска трансформација између клијента и апстрактног сервера.
б) симетријска група коју образују апстрактни сервер и конкретни сервери.
То значи, концептуално гледајући, да *одржива структура настаје тако што се несиметријска структура трансформише у симетријску структуру*. ***Патерни, у суштини представљају процес трансформације несиметријске у симетријску структуру.*** Понављам, симетријска структура је

[9] Ако непрекидно мењате нивое апстракције код излагања, ваше излагање је нејасно, јер обично губите основни ток мисли, непрекидно понирете у детаље, остајете на њима и заборављате шта сте почели да објашњавате. Излаз из наведеног проблема јесте држања једног нивоа апстракције код објашњавања и избегавање да се превише улази у детаље. Ако се улази у детаље то треба урадити тако да се не изгуби и занемари основни ток (нит) размишљања.
[10] Пример за то је локализација неког софтверског система, којом се врши прилагођавање софтверског система различитим светским језицима.
[11] Слојеви су хијерархијски организовани, при чему се на врху хијерархије налази најапстрактнији слој, док се на дну хијерархије налази најконкретнији слој.
[12] Макро патерни као што је нпр. MVC или микро патерн facade уводе допунске нивое у архитектури софтверског система.
[13] *Butler Lampson* је рекао: "Сви проблеми у рачунарству могу бити решени другим нивоом индирекције (All problems in computer science can be solved by another level of indirection) ". Наведена констатација је речена у ироничном смислу, али она посредно говори да свако побољшање неке перформансе софтверског система (у овом случају одржавање система) са једне стране, обара неке друге перформансе (у овом случају повећава се сложеност и брзина извршења софтверског система.)

docsity.com

одрживија у односу на несиметријску структуру, јер се таква структура лакше одржава и надограђује.

Дефиниција **одрживости**: *способност нечега да расте и да се развија; способност нечега да живи (да траје).*
Ако бисмо дефиницију одрживости применили на софтвер онда би могли рећи да је одржив софтвер онај софтвер који може да расте и да се развија. Да би софтвер могао да се развија он мора да се:
а) **одржава**, да би обезбедио или променио постојећу функционалност и да се
б) **надограђује**, како би обезбедио допунске функционалности.
Патерни обезбеђују структуру која је одржива, односно структуру која може лако да се одржава и надограђује. Патерни обезбеђују механизам који ће структуре које су тешке за развој (неодрживе структуре) трансформисати у структуре које се могу лако развијати (одрживе структуре). Неодрживе структуре имају краћи век трајања од одрживих структура и веома се брзо деле и доводе систем у хаотично стање. У њих мора да се улаже велика спољна енергија како би систем могао да функционише и да се развија. Код одрживих система се улаже мања спољашња енергија код одржавања и надоградње система.
Код одрживих система додавање или промена неког елемента структуре неће утицати на остале елементе структуре. Код неодрживих структура, додавање или промена неког елемента структуре утиче на промену других елемената структуре. што чини да је дата структура непостојана и нестабилна. Међузависност елемената система је превише велика и такви системи су тешки за одржавање.

Патерни, ред и хаос

У сваком систему различитости имају тенденцију да се остваре. Уколико две или више различитости не пронађу заједнички именитељ (заједничке особине) који ће их окупити, оне ће имати тенденцију да се даље деле (унутар њих самих) што води ка хаосу и нереду. То значи да ће се **апсолутни хаос** десити уколико се све различитости (до најситнијих различитости) у некој појави остваре. Патерни имају механизам који не дозвољава да систем уђе у апсолутни хаос. Патерни уводе одрживе структуре на местима које могу систем да уведу у хаос. Систем ће ући у хаос ако се дозволи да различитости (различите вредности) доминирају у односу на заједништво (заједничке вредности). Различитост има тенденцију да наруши заједништво. Заједништво има тенденцију да неутралише различитост. Систем ће постати тоталитаран ако се дозволи да заједништво неутралише различитости, односно да ред постане апсолутан, јер би тада имали тоталитарни систем који не прихвата различитости. Не треба заустављати почетак настанка неке различитости[14] јер се тиме спречава и успорава развој система. Хаос и ред се непрекидно смењују и то је нормалан процес у развоју било ког система. *Патерни држе хаос и ред у непрекидној равнотежи и не дозвољавају да било ко од њих постане апсолутан.*
...

Патерни описују процес у коме систем никада неће отићи у апослутни хаос или апсолутни ред. Патерни омогућавају да систем из реда пређе у "ограничени хаос" како би се десиле различитости које прилагођавају систем његовом окружењу. Такође патерни омогућавају да систем из хаоса пређе у "ограничени ред" како би се десило заједништво које јача систем изнутра. Хаос са једне стране слаби систем али га са друге стране чини прилагодљивијим окружењу. Ред са једне стране јача систем али га са друге стране чини крутим и мање прилагодљивим окружењу.

Из наведеног изводим следећу хипотезу: *Патерни омогућавају да производ реда и хаоса буде увек нека константа rh:*

rh = Red * Haos

[14] Право на различитост не подразумева наметање те различитости другима. То се посебно односи на оне који не подржавају ту различитост. Треба разликовати подржавање права на различитост од подржавања различитости. Неко може да подржи право некога да буде различит и у исто време да лично не подржи ту различитост. Неко може да подржи нечије право да се бори за различитост а у исто време да не подржи ту различитост. Право на различитост није право на наметање различитости. Ако неко има право на различитост, он нема право да намеће другима ту различитост. Нпр. Свако има право да слуша музику у своме стану, али нема право да појача ту музику и да је намеће другима који не воле (не подржавају) ту музику.

docsity.com

4. Патерни пројектовања

4.1 Објектно-оријентисани концепти потребни за схватање патерна пројектовања

Да би патерни пројектовања могли да се схвате потребно је разумети неке од основних концепата објектно-оријентисаног програмирања[15]:
а) Класе и објекте
б) Наслеђивање
ц) Динамички полиморфизам
ц) Апстрактне класе
д) Интерфејси

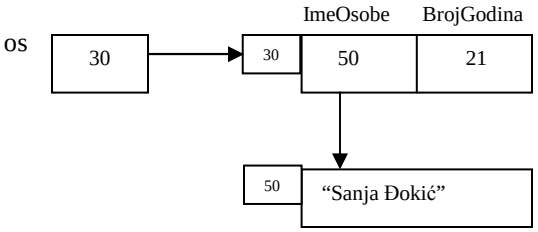
КЛАСА И ОБЈЕКАТ
Као што је познато **класа** се састоји од скупа атрибута и метода. Атрибути описују **структуру** класе док методе описују њено **понашање**. На пример класа Student:

```
class Osoba
  { String ImeOsobe;
    int BrojGodina;
    void Postavi(String ImeOsobe1, int BrojGodina1)  {ImeOsobe = ImeOsobe1; BrojGodina =
            BrojGodina1;}
    String Prikazi() { System.out.println("Ime osobe: " + ImeOsobe + " Broj godina:" + BrojGodina);}
    public static void main(String arg[])
          {  Osoba os;
             os = new Osoba();
             os.Postavi("Sanja Đokić",21);

          }
  }
```

има атрибуте *ImeOsobe* и *BrojGodina* и методе *Postavi()* и *Prikazi()*. По дефиницији, ***класа је општи представник неког скупа објеката који имају исте особине (атрибуте и методе).*** Из тога следи да објекат представља једно појављивање (примерак, инстанцу) класе. У наведеном примеру објекат се креира са наредбом:

```
os = new Osoba();
```

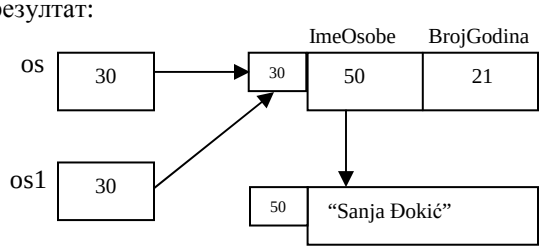
Тада објектна променљива *os* добија адресу (нпр. 30) објекта који је креиран.



Наглашавам да се у слободном жаргону обично за *os* каже да је објекат. То није тачно јер је *os* **показивач на објекат**, односно **објектна променљива** чији је садржај адреса објекта на који показује. Ово треба узети озбиљно у обзир јер ћемо у случају следеће наредбе,

```
Osoba os1 = os;
```

имати резултат:



[15] Наведене концепте ћу објаснити у програмском језику **Јава**. Препоручујем да погледате скрипту ***Основни концепти Јаве*** у којој су детаљно објашњени концепти: *класе, објекти, методе, наслеђивање, апстрактне класе, интерфејси, рад са стринговима, пакети и изузеци*. Адреса где се налази скрипта:
http://silab.fon.rs/index.php?option=com_docman&task=doc_download&gid=706&&Itemid=56

docsity.com

НАСЛЕЂИВАЊЕ И ДИНАМИЧКИ ПОЛИМОРФИЗАМ

Један од најважнијих концепата објектно-оријентисаног програмирања је концепт **наслеђивања.** Уколико имамо основну класу *Osoba,*

```
class Osoba
  { String ImeOsobe;
    int BrojGodina;
    Osoba(String ImeOsobe1, int BrojGodina1)
          {ImeOsobe = ImeOsobe1; BrojGodina =  BrojGodina1;}
    void Prikazi() { System.out.println("Ime osobe: " + ImeOsobe + "Broj godina:" + BrojGodina);}
    void PrikaziOsobu() { System.out.println("Ime osobe: " + ImeOsobe);}
  }
```

коју наслеђује класа *Student,*

```
class Student extends Osoba
  { String BrojIndeksa;
    Student(String ImeOsobe1, int BrojGodina1,String BrojIndeksa1)
          { super(ImeOsobe1,BrojGodina1); BrojIndeksa = BrojIndeksa1;}

    void Prikazi() { super.Prikazi(); System.out.println("Broj indeksa:" + BrojIndeksa);}
    void PrikaziStudenta() { super.PrikaziOsobu(); System.out.println("Broj indeksa:" + BrojIndeksa);}

  }
```

тада можемо да кажемо да класа *Student* наслеђује све атрибуте и јавне методе класе *Osoba*. Из концепта наслеђивања директно произилази концепт **компатибилности објектних типова** код кога *објектна променљива надкласе може да добије референцу на било који објекат који припада класи која наслеђује надкласу*. У нашем примеру објектна променљива класе *Osoba* може да добије референцу на објекат класе *Student* што можемо представити следећим примером:

```
Osoba os;
Student st = new Student();
os = st;
```

Компатибилност објектних типова је посебно важна код прављења **генеричких програма** када треба да се обезбеди могућност да *једна наредба програма има различито понашање у зависности од логике програма*. На пример, уколико дефинишемо главни програм који користи наведене класе *Osoba* и *Student*:

```
public static void main(String arg[])
        {   Osoba os;
            Osoba os1 = new Osoba("Sanja Djokic",21);
            Student st = new Student("Maja Stanilovic", 22,"12/09");

            if (arg[0].equals("1"))
                    os = os1;
            else
                    os = st;

            os.Prikazi();
        }
```

за наредбу

```
os.Prikazi();
```

можемо да кажемо да се различито понаша у зависности од логике извршења програма. Уколико је задовољен услов да је улазни аргумент arg[0] једнак 1 тада ће објектна променљива *os* да добије референцу на објекат класе *Osoba*. Код наредбе *os.Prikazi()* биће позвана метода *Prikazi()* класе *Osoba*. Уколико није задовољен наведени услов објектна променљива *os* ће добити референцу на објекат класе *Student.* У том случају наредба *os.Prikazi()* ће позвати методу *Prikazi()* класе *Student.*

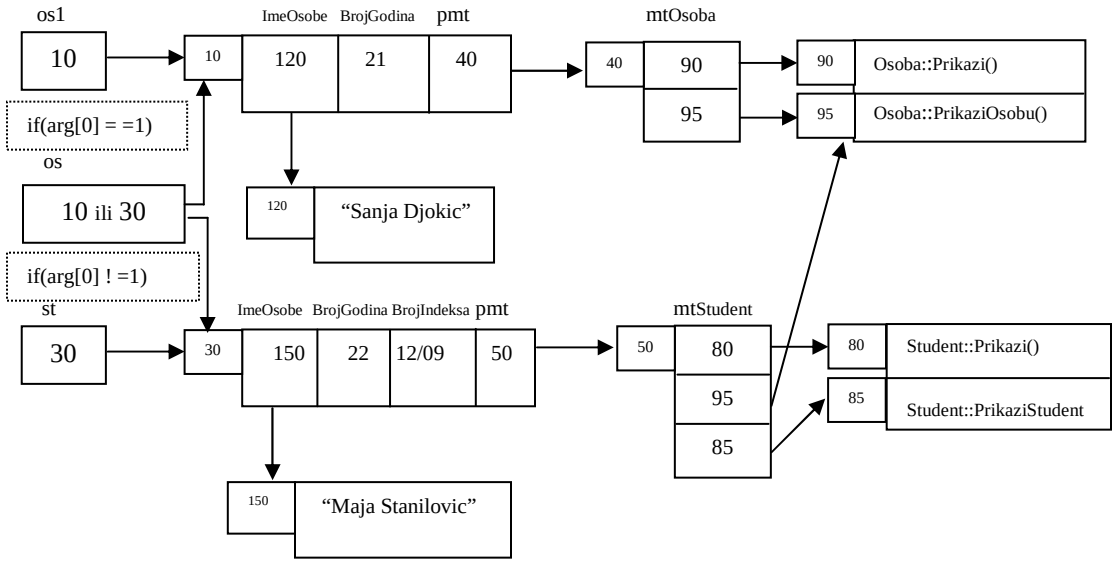
docsity.com

...

Уколико једна наредба може да има различито понашање за њу се може рећу да је **полиморфна**[16]**.** На овај начин смо дошли до ткз. **динамичког**[17] **полиморфизма,** код кога објектна променљива у време извршења програма (run-time) добија адресу објекта који ће да изврши дефинисану методу.
То значи да се тек у време извршења програма одређује (разрешава) понашање неке полиморфне наредбе.

...

КАСНО ПОВЕЗИВАЊЕ ОБЈЕКТА СА МЕТОДОМ
Методе у Јава програмском језику се повезују са објектима у време извршења програма. Овакво повезивање се зове **касно повезивање** (late binding) или **динамичко повезивање** објекта са методом. За наведени пример дајем изглед оперативне меморије[18], након извршења главног програма, како бих објаснио како се остварује касно повезивање објекта са методом:



Сваки објекат (на адреси 10 и 30) има невидљиви атрибут *pmt* који представља показивач на табелу метода класе (*mtOsoba*) тог објекта. У наведеном примеру објекат класе *Osoba* има атрибут *pmt* чији је садржај (40), што је адреса табеле метода класе *Osoba* (*mtOsoba*). Табела метода класе *Osoba* (*mtOsoba)* садрже низ показивача на адресе (90,95) где се налазе методе класе *Osoba*.
Објекат класе *Student* има атрибут *pmt* чији је садржај (50), што је адреса табеле метода класе *Student* (*mtStudent*). Табела метода класе *Student* (*mtStudent)* садрже низ показивача на адресе (80,95,85) где се налазе методе класе *Student*. Будући да класа *Student* наслеђује класу *Osoba* самим тим наслеђује и њене методе. Пошто је класа *Student* прекрила методу *Prikazi()* класе Osoba, метода *Prikazi()* класе *Student* налази се на посебној адреси (80), док се метода *Prikazi()* класе *Osoba* налази на адреси (90).
Методу *PrikaziOsobu* од класе *Osoba* класа *Student* није прекрила (није је имплементирала) тако да табеле метода класа *Osoba* и *Student* показују на исту адресу (95) где се налази метода *PrikaziOsobu* класе *Osoba*. Класа *Student* поред две наведене методе *Prikazi()* и *PrikaziOsobu()* има и додатну методу *PrikaziStudent()* која се налази на адреси 85. Табеле метода (*mtOsoba* и *mtStudent*) се пуне по редоследу наведених метода у класама.

---

[16] Појам **полиморфизам** су први користили стари Грци када су желели да објасне да *нешто може да има више облика.*

[17] Појам "**динамички**" се односи на време извршења програма. Насупрот томе појам "**статички**" се односи на време компајлирања програма. Каже се да се објекат **динамички** повезује са методом у време извршења програма, док се објекат повезује **статички** са методом у време компајлирања програма.

[18] Адресе на којима се налазе променљиве и објекти су фиктивни.

docsity.com

...

У време компајлирања наредба:

```
os.Prikazi();
```

се преводи у следећи облик

```
os.pmt[0];
```

У време компајлирања се не може разрешити која метода *Prikazi()* ће бити позвана јер се не зна на који ће објекат *os* да добије референцу (адресу).

...

У време извршења програма наредба:

```
os = os1;
```

има следећи ефекат на наредбу:

```
os.pmt[0];
```

Преко садржаја од *os* се приступа садржају од *pmt* који приступа садржају [0] индекса од *mtOsoba*.

S(os).S(pmt).S([0]) ------> 10.40.90 – Приступа се до методе *Prikazi ()* класе *Osoba*[19].

У време извршења програма објекат који се налази на адреси 10 (коме се приступа преко *os*) се повезује са методом која је на адреси 90.

У време извршења програма наредба

```
os = st;
```

има следећи ефекат на наредбу:

```
os.pmt[0];
```

Преко садржаја од *st* се приступа садржају од *pmt* који приступа садржају [0] индекса од *mtStudent.*

S(st).S(pmt).S([0]) ------> 30.50.80 – Приступа се до методе *Prikazi ()* класе *Student.*

У време извршења програма објекат који се налази на адреси 30 (коме се приступа преко *os*) се повезује са методом која је на адреси 80.

Уколико се стави кључна реч static испред имена методе, таква метода се повезује са објектом у време компајлирања програма. Такво повезивање се зове **рано повезивање** (early binding) или статичко повезивање објекта са методом. Код раног повезивање објекат је повезан са једном методом и не може се у току извршења програма повезати са неком другом методом.

[19] Нотација нпр. S(os) указује на садржај од *os*, што је у наведеном примеру адреса 10 где се налази објекат класе *Osoba*.

docsity.com

АПСТРАКТНЕ КЛАСЕ И ИНТЕРФЕЈСИ

Уколико желимо да понашање неке класе издигнемо на општији ниво тада користимо **апстрактне класе** и **интерфејсе**. Апстрактне класе поред атрибута и обичних метода имају и **апстрактне методе**, док интерфејси имају само **операције**. И апстрактне методе и операције имају *само потпис без имплементације*. Класе које наслеђују апстрактне класе или имплементирају интерфејс морају да реализују апстрактне класе и операције.

На пример, уколико имамо апстрактну класу *Osoba* која има апстрактну методу *Prikazi()*,

```
abstract class Osoba
  { String ImeOsobe;
    int BrojGodina;
    Osoba(String ImeOsobe1, int BrojGodina1)
          {ImeOsobe = ImeOsobe1; BrojGodina =  BrojGodina1;}
    abstract void Prikazi();
    void PrikaziOsobu() { System.out.println("Ime osobe: " + ImeOsobe);}
  }
```

потребно је да изведена класа *Student* имплементира (реализује) наведену апстрактну методу:

```
class Student extends Osoba
  { String BrojIndeksa;
    Student(String ImeOsobe1, int BrojGodina1,String BrojIndeksa1)
          { super(ImeOsobe1,BrojGodina1); BrojIndeksa = BrojIndeksa1;}

    void Prikazi() { System.out.println("Ime osobe: " + ImeOsobe + "Broj godina:" + BrojGodina + " Broj indeksa:" +
                    BrojIndeksa);}
    void PrikaziStudenta() { super.PrikaziOsobu(); System.out.println("Broj indeksa:" + BrojIndeksa);}

  }
```

На сличан начин уколико имамо интерфејс *Osoba*,

```
interface Osoba
  {  void Prikazi();
     void PrikaziOsobu();
  }
```

класа *Student* која имплементира интерфејс *Osoba* треба да реализује методе интерфејса:

```
class Student implements Osoba
  {  String ImeStudenta;
    int BrojGodina;
    String BrojIndeksa;
    Student(String ImeOsobe1, int BrojGodina1,String BrojIndeksa1)
          { ImeOsobe = ImeOsobe1; BrojGodina =  BrojGodina1; BrojIndeksa = BrojIndeksa1;}

    void Prikazi() { System.out.println("Ime osobe: " + ImeOsobe + "Broj godina:" + BrojGodina + " Broj indeksa:" +
                    BrojIndeksa);}
    void PrikaziOsobu() { System.out.println("Ime osobe: " + ImeStudenta);}

    void PrikaziStudenta() { super.PrikaziOsobu(); System.out.println("Broj indeksa:" + BrojIndeksa);}

  }
```

Могуће је направити објектну променљиву типа апстрактне класе или интерфејса,

```
Osoba os;
```

али није могуће направити појављивање апстрактне класе или интерфејса:

```
Osoba os = new Osoba();
```

Наведена наредба није коректна.

docsity.com

За објектну променљиву типа апстрактне класе или интерфејса важи правило компатибилности објектних типова, што значи да она може да добије референцу на објекат класе која наслеђује апстрактну класу или имплементира интерфејс. Нпр:

```
Osoba os;
Student st = new Student();
os = st;
```

4.2 Имплементација општег облика патерна

Као што је речено патерн је процес трансформације структуре проблема у структуру решења. У наставку ћу дати имплементацију структуре проблема и структуре решења код патерна у општем случају. Пре тога ћу објаснити пример програма који је тежак за одржавање и надоградњу и објаснићу главни разлог зашто уводимо патерне.

Уколико имамо класу *BrokerBazePodataka* и њену методу *brisiSlog()*,

```
class BrokerBazePodataka
 {   Student st;
     Predmet pr;
      ...
          BrokerBazePodataka(Student st1, Predmet pr1)  {st = st1; pr = pr1;}

          public boolean brisiSlog(String ImeKlase)
               {  String upit;
                  String UslovZaNadjiSlog;

                   If ImeKlase.equals("Student")
                         UslovZaNadjiSlog = st.vratiUslovZaNadjiSlog();

                   If ImeKlase.equals("Predmet")
                         UslovZaNadjiSlog = pr.vratiUslovZaNadjiSlog();


                 try {  st = con.createStatement();
                       upit ="DELETE *  FROM " + imeKlase + " WHERE " + UslovZaNadjiSlog;
                       st.executeUpdate(upit);
                        st.close();
                    } catch(SQLException esql)
                       { porukaMetode = porukaMetode + "\nNije uspesno obrisan slog u bazi: " + esql;
                         return false;
                          }
                 porukaMetode = porukaMetode + "\nUspesno obrisan slog u bazi.";
                 return true;
               }


          public boolean otvoriBazu(String imeBaze)
            {  String Urlbaze;
               try {  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                      Urlbaze = "jdbc:odbc:" + imeBaze;
                      con = DriverManager.getConnection(Urlbaze);
                      con.setAutoCommit(false); // Ako se ovo ne uradi nece moci da se radi roolback transakcije.
                   } catch(ClassNotFoundException e)
                          { porukaMetode = "Drajver nije ucitan:" + e; return false;}
                        catch(SQLException esql)
                             { porukaMetode = "Greska kod konekcije:" + esql; return false;}
                     catch(SecurityException ese)
                          { porukaMetode = "Greska zastite:" + ese; return false;}
                 porukaMetode = "Uspostavljena je konekcija sa bazom podataka."; return true;
            }

…
}
```

која се позива из главног програма,

docsity.com

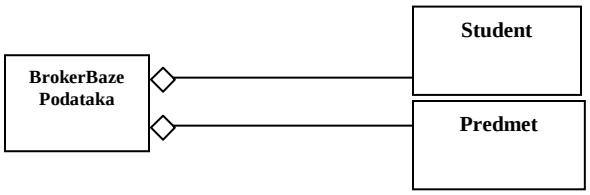
```
class Main
{  public static void main(String arg[])
    {  Student st = new Student();
       Predmet pr = new Predmet();
       BrokerBazePodataka bbp = new BrokerBazePodataka(st,pr);
      bbp.otvoriBazu("Fakultet");

       st.BrojIndeksa = "123-09";
       bbp.brisiSlog("Student");

       pr.SifraPredmeta = 11;
       bbp.brisiSlog("Predmet");

    }
}
```

и која треба да обришу слогове из табела *Student* и *Predmet*, на основу вредности објеката *st* и *pr* класа *Student* и *Predmet*.

```
class Student
{
  String BrojIndeksa;
  String ImeStudenta;
   …
  public String vratiUslovZaNadjiSlog() { return " BrojIndeksa = '"+ BrojIndeksa + "'"; }
  ...
}
```

```
class Predmet
{ int SifraPredmeta;
  String NazivPredmeta;
  ...
  public String vratiUslovZaNadjiSlog() { return " SifraPredmeta = " + SifraPredmeta; }
}
```

Наведени програм има следећу структуру:



У наведеном примеру *BrokerBazePodataka* је клијент, док су *Student* и *Predmet* конкретни сервери. Наведена структура је тешка за одржавање и надоградњу јер свако додавање нове класе, нпр.класе *Profesor*

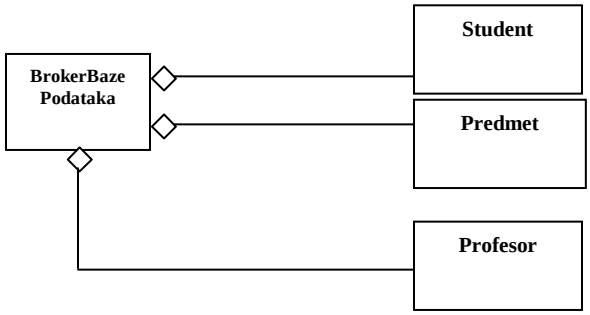
```
class Profesor
{
  String ImeProfesora;
   …
  public String vratiUslovZaNadjiSlog() { return " ImeProfesora = '"+ ImeProfesora + "'"; }
  ...
}
```

која би била повезана са брокером базе података захтева промену брокера (клијента):

```
class BrokerBazePodataka
 {   Student st;
     Predmet pr;
```


docsity.com

```
    Profesor prof;
     ...

          BrokerBazePodataka(Student st1, Predmet pr1)  {st = st1; pr = pr1;}

          public boolean brisiSlog(String ImeKlase)
               {  String upit;
                  String UslovZaNadjiSlog;

                   If ImeKlase.equals("Student")
                         UslovZaNadjiSlog = st.vratiUslovZaNadjiSlog();

                   If ImeKlase.equals("Predmet")
                         UslovZaNadjiSlog = pr.vratiUslovZaNadjiSlog();

                  If ImeKlase.equals("Profesor")
                        UslovZaNadjiSlog = prof.vratiUslovZaNadjiSlog();
                  ...
...
}
```



Наведени проблем се решава тако што се уводи апстрактна класа *OpstaDomenskaKlasa*, која има апстрактну методу *vratiUslovZaNadjiSlog()*

```
abstract OpstaDomenskaKlasa
 {
   public String vratiUslovZaNadjiSlog();
   public String vratiImeKlase();

}

class Student extends OpstaDomenskaKlasa
{
   String BrojIndeksa;
   String ImeStudenta;
    …
   public String vratiUslovZaNadjiSlog() { return " BrojIndeksa = '"+ BrojIndeksa + "'"; }
   public String vratiImeKlase() {return "Student";}

   ...
}


class Predmet extends OpstaDomenskaKlasa
{ int SifraPredmeta;
   String NazivPredmeta;
   ...
   public String vratiUslovZaNadjiSlog() { return " SifraPredmeta = " + SifraPredmeta; }
   public String vratiImeKlase() {return "Predmet;}

}
```


docsity.com

```
class Profesor extends OpstaDomenskaKlasa
{
   String ImeProfesora;
    …
   public String vratiUslovZaNadjiSlog() { return " ImeProfesora = '"+ ImeProfesora + "'"; }
   public String vratiImeKlase() {return “Profesor;}

   ...
}
```

док ће брокер базе података да има следећи изглед:

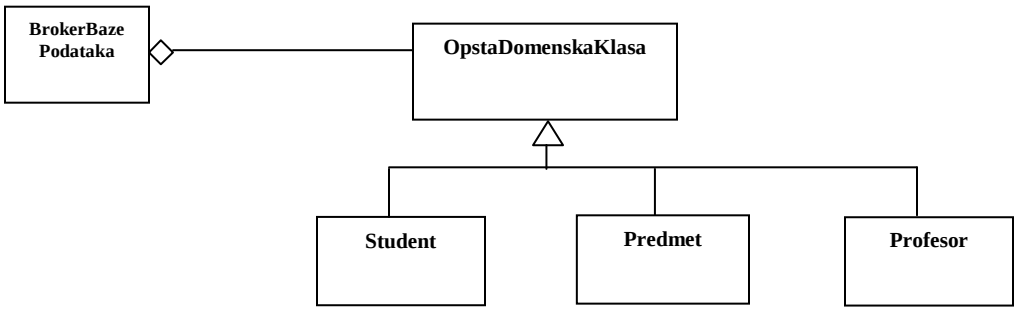
```
class BrokerBazePodataka
 {   OpstaDomenskaKlasa odk;
       ...

            public boolean brisiSlog()
                 {  String upit;
                    String UslovZaNadjiSlog;
                    String ImeKlase;


                      ImeKlase = odk.vratiImeKlase();
                      UslovZaNadjiSlog = odk.vratiUslovZaNadjiSlog();


                   try {  st = con.createStatement();
                         upit ="DELETE *  FROM " + imeKlase + " WHERE " + UslovZaNadjiSlog;
                         st.executeUpdate(upit);
                          st.close();
                      } catch(SQLException esql)
                          { porukaMetode = porukaMetode + "\nNije uspesno obrisan slog u bazi: " + esql;
                            return false;
                             }
                   porukaMetode = porukaMetode + "\nUspesno obrisan slog u bazi.";
                   return true;
                 }

…
}
```

док ће главни програм имати следећи изглед:

```
class Main
{  public static void main(String arg[])
    {  Student st = new Student();
       Predmet pr = new Predmet();
       Profesor prof = new Profesor();

       BrokerBazePodataka bbp = new BrokerBazePodataka();
       bbp.otvoriBazu(“Fakultet“);

       st.BrojIndeksa = “123-09“;
       bbp.odk = st;
       bbp.brisiSlog();

       pr.SifraPredmeta = 11;
       bbp.odk = pr;
       bbp.brisiSlog();

      pr.ImeProfesora = “Milan Petrovic“;
       bbp.odk = prof;
       bbp.brisiSlog();


    }
}
```

Наведени програм има следећу структуру:

docsity.com



Ова структура је лакша за одржавање јер додавање нове класе не мења клијента, односно брокер базе података. Уколико се додаје нова класа, нпр. Sala, она треба само да имплементира апстрактне методе апстрактне класе *OpstaDomenskaKlasa.*

```
class Sala extends OpstaDomenskaKlasa
{
  int SifraSale;
   …
  public String vratiUslovZaNadjiSlog() { return " SifraSale = "+ SifraSale; }
  public String vratiImeKlase() {return “Sala”;}

  ...
}
```

Можемо да приметимо да се метода *brisiSlog()* класе *BrokerBazePodataka* не мења.

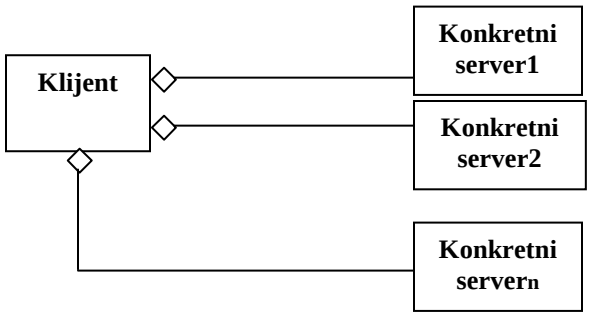
На основу наведеног примера можемо да објаснимо општи облик патерна. Проблем код патерна можемо да представимо на следећи начин:

```
class Klijent
{ KonkretniServer1 ks1;
  KonkretniServer2 ks2;
  …
  KonkretniServern ksn;

  Klijent(KonkretniServer1 ks11, KonkretniServer1 ks22,..., KonkretniServer1 ksn1){ ks1=ks11;ks2=ks21;…;ksn = ksn1;}

 void op(uslov)
    {   if (uslov1 = uslov)
          ks1.op();

         if (uslov2 = uslov)
           ks2.op();
         …

        if (uslovn = uslov)
           ksn.op();

    }
}

class KonkretniServer1
{ op() {…}  // имплементација методе op() класе KonkretniServer1
}

class KonkretniServer2
{ op() {…}  // имплементација методе op() класе KonkretniServer2
}
```


docsity.com

```
...

class KonkretniServern
{ op() {...}  // имплементација методе op() класе KonkretniServern
}


class Main
{
   public static void main(String arg[])
    {
      KonkretniServer1 ks1 = new KonkretniServer1();
      KonkretniServer2 ks2 = new KonkretniServer2();
      …
      KonkretniServern ksn = new KonkretniServern();
      Klijent kl = new Klijent(ks1,ks2,…,ksn);

       kl.op(uslov1);
       kl.op(uslov2);
       ...
       kl.op(uslovn);
    }

}
```

Структура проблема код патерна је:



За наведену структуру је речено да је тешка за одржавање јер се при додавању новог конкретног сервера, нпр. **KonkretniServern+1**, мења клијент. Такође, клијент се у време компајлирања програма везује за конкретни сервер, што онемогућава флексибилност програма у току његовог извршавања.

Наведени проблем код патерна се решава увођењем апстрактног сервера:

```
class ApstraktniServer
{
    op();

}
```

из кога су изведени конкретни сервери:

```
class KonkretniServer1 extends ApstraktniServer
{ op() {...}  // имплементација методе op() класе KonkretniServer1
}

class KonkretniServer2 extends ApstraktniServer
{ op() {...}  // имплементација методе op() класе KonkretniServer2
}
...
```


docsity.com

```
class KonkretniServern extends ApstraktniServer
{ op() {...}  // имплементација методе op() класе KonkretniServern
}
```

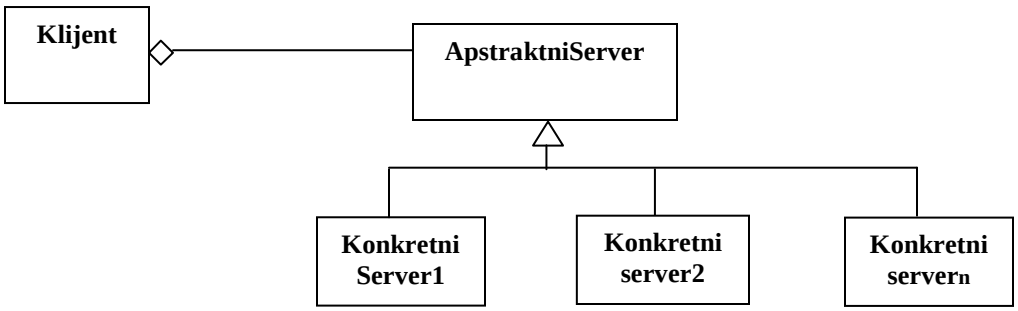
док клијент има следећи изглед:

```
class Klijent
{
  ApstraktniServer as;

 void op()
    {   as.op();
    }
}
```

Главни програм је:

```
class Main
{
   public static void main(String arg[])
    {
      KonkretniServer1 ks1 = new KonkretniServer1();
      KonkretniServer2 ks2 = new KonkretniServer2();
      …
      KonkretniServern ksn = new KonkretniServern();
      Klijent kl = new Klijent();

       kl.as = ks1; kl.op();
       kl.as = ks2; kl.op();
       ...
       kl.as = ksn; kl.op();
    }

}
```

Структура решења код патерна је:



Додавање новог конкретног сервера *KonkretniServer$_{n+1}$* не мења клијента:

```
class KonkretniServern+1 extends ApstraktniServer
{ op() {...}  // имплементација методе op() класе KonkretniServern+1
}
```


docsity.com

Задаци:

1. Шта је потребно урадити у програму да би се на стандардном излазу добила порука “Sutra ce biti lep dan.” . Наредбе програма не смеју да се бришу. Наредбе програма могу да се додају на местима где има три тачке.

```
class X  …
{    Y y;
     X() { y = new Y(); }
     void op(Z z) { z.Prikazi(); }
     void Prikazi(){System.out.println(“Sutra ce biti lep dan”);}
}

class Y …
{   void Prikazi() {System.out.println(“Danas je lep dan”);}
}

abstract class Z
{  abstract void Prikazi ();
}

class Main
{  public static void main(String arg[])
     { X x = new X();
       …
       x.op(…);
    }
}
```

2. Шта је потребно урадити у програму да би се на стандардном излазу добила порука: “Danas je lep dan.” . Наредбе програма не смеју да се бришу. Наредбе програма могу да се додају на местима где има три тачке.

```
class X …
{
     void Prikazi(){System.out.println(“Sutra ce biti lep dan”);}
}

class Y …
{
      void Prikazi() {System.out.println(“Danas je lep dan”);}
}

interface Z
{  void Prikazi ();
}

class Main
{  public static void main(String arg[])
     { X x = new X();
       Y y = new Y();
       Z z = …;
       z.Prikazi();
    }
}
```


docsity.com

3. Шта је потребно урадити у програму да би на стандардном излазу добили поруке “Danas je lep dan. Sutra ce biti lep dan.” у наведеном редоследу. Наредбе програма не смеју да се бришу. Наредбе програма могу да се додају на местима где има три тачке.

```
class X ...
{
    void Prikazi(){System.out.println(“Sutra ce biti lep dan”);}
}

class Y ...
{
     void Prikazi() {System.out.println(“Danas je lep dan”);}
}

interface Z
{  void Prikazi ();
}

class Main
{  public static void main(String arg[])
    { X x = new X();
      Y y = new Y();
       Z z ;
       ...
       z.Prikazi();
        ...
       z.Prikazi();
   }
}
```


docsity.com

4. Променити наведени програм који је тежак за одржавање и надоградњу у одрживи програм у складу са концептом патерна.

```
class Y1
{
    Prikazi(){System.out.println("Sutra ce biti lep dan");}
}

class Y2
{
     Prikazi() {System.out.println("Danas je lep dan");}
}


class X
 {  Y1 y1;
    Y2 y2;
     X() { y1 = new Y1(); y2 = new Y2(); }
     void op(int y)
          {  if (y = 1)
               y1.Prikazi();
             if (y = 2)
               y2.Prikazi();
          }
 }



class Main
{  public static void main(String arg[])
    {
      X x = new X();
      x.op(1);
    x.op(2);
    }
}
```

4.3 Прављење генеричких метода

Као што је раније речено, када развијамо одржив програм потребно је да раздвојимо генеричке од специфичних делова програма. Генерички делови програма су представљени методама које се могу користити у различитим доменима проблема. Нпр. брокер базе података је пример класе која садржи скуп генеричких метода које омогућавају перзистентност доменских објеката при раду са базом података.
Када правимо генеричку методу потребно је да у методи препознамо места која се мењају са променом домена проблема и места која су непромењива. На промењивим местима треба поставити механизам који ће омогућити непромењивост програмског кода. Промењива места у програму се обично реализују коришћењем операција интерфејса или апстрактних метода апстрактних класа.

На пример уколико имамо две методе које сортирају низ у растућем и опадајућем редоследу:

```
class Sortiranje
 {
    static void sortRastuci(int n[])
      { int pom = 0;
        for(int i=0; i< n.length-1;i++)
              for(int j=0; j<n.length; j++)
                    { if(n[i]>n[j])
                          {pom = n[i];  // prvi jednako drugi
                           n[i] = n[j]; // drugi jednako treci
                       n[j] = pom;  // treci jednako prvi
                      }
        }
   }
```


docsity.com

```
static void sortOpadajuci(int n[])
      { int pom = 0;
        for(int i=0; i< n.length-1;i++)
            for(int j=0; j<n.length; j++)
             { if(n[i]<n[j])
                 { pom = n[i];  // prvi jednako drugi
                   n[i] = n[j]; // drugi jednako treci
                   n[j] = pom;  // treci jednako prvi
                 }

             }
      }

static void Prikazi(int n[])
  {   System.out.println("Elementi niza su:");
      for(int i=0; i<n.length;i++)
          { System.out.println(n[i]);}
  }

public static void main(String arg[])
   { int n[] = {7,1,2,8,3};
     sortOpadajuci(n);
     Prikazi(n);
     sortRastuci(n);
      Prikazi(n);
   }

}
```

можемо да приметимо да се наведене две методе разликују у наредби која одређује услов када се два елемента низа мењају место. Прецизније речено те две методе се разликују у оператору > односно <. Од наведене две методе треба направити једну генеричку методу:

```
static void sort (int n[])
      {  int pom = 0;
        for(int i=0; i< n.length-1;i++)
              for(int j=0; j<n.length; j++)
                 { if(n[i] © n[j])  // симбол © може бити > или <.  Уколико се изабере оператор >
                                    // низ ће бити сортиран у растућем редоследу. Уколико се изабере
                                    // оператор < низ ће бити сортиран у  опадајућем редоследу
                     { pom = n[i];
                       n[i] = n[j];
                       n[j] = pom;
                     }
                 }
      }
```

У наведеној методи симбол © у програму указује на оно што је променљиво у програму.
Наредба **if(n[i] © n[j])** је променљива. Она може да има два облика:

a) **if (n[i] > n[j]) –** сортирање низа у растућем редоследу.
b) **if (n[i] < n[j]) –** сортирање низа у опадајућем редоследу.

Наведени симбол ће бити имплементиран коришћењем операције ***Poredi()*** интерфејса ***OperatorPoredjenja.***

```
class Sortiranje1
{
   static void sort(int n[], OperatorPoredjenja op)
      { int pom = 0;
        for(int i=0; i< n.length-1;i++)
            for(int j=0; j<n.length; j++)
             { if(op.poredi(n[i],n[j]))
                 { pom = n[i];
                   n[i] = n[j];
                   n[j] = pom;
                 }
             }
      }
```

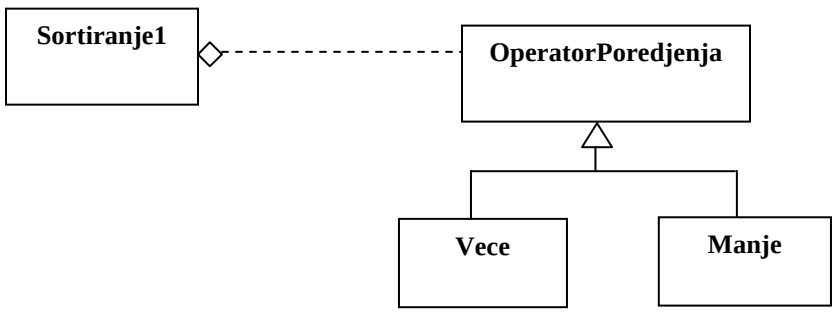

docsity.com

```
    static void Prikazi(int n[])
      {   System.out.println("Elementi niza su:");
          for(int i=0; i<n.length;i++)
               { System.out.println(n[i]);}
               }

     public static void main(String arg[])
        { int n[] = {7,1,2,8,3};
          sort(n,new Manje());
          Prikazi(n);
          sort(n,new Vece());
          Prikazi(n);
       }
}

interface OperatorPoredjenja
{boolean poredi(int a, int b);}

class Vece implements OperatorPoredjenja
 { public boolean poredi(int a, int b)
    {  if (a>b) return true;
       return false;
    }
 }

class Manje implements OperatorPoredjenja
 { public boolean poredi(int a, int b)
    {  if (a<b) return true;
       return false;
    }
 }
```

На основу наведеног можемо да изведемо правило:

Уколико желимо да метода постане генеричка, потребно је да се уоче промењиве наредбе у телу методе (if(n[i]>n[j]), if(n[i]<n[j])) и и да се замене са генеричким наредбама (op.poredi(n[i],n[j])). Оно што је специфично (промењиве наредбе) у методи постаје параметар методе (OperatorPoredjenja op) који је обично типа апстрактне класе или интерфејса (interface OperatorPoredjenja).

Можемо да приметимо да је у процесу прављења генеричких метода коришћен концепт патерна, прецизније речено структура решења код патерна:



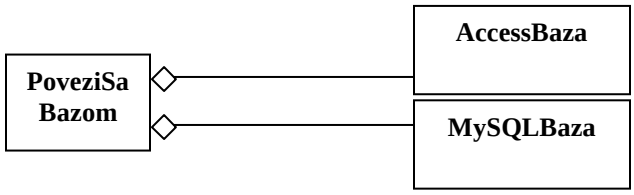
У следећем примеру се Јава програм повезује са два различита система за управљање базом података помоћу две различите методе ***Povezi()*** класа ***AccessBaza*** и ***MySqlBaza***:

```
import java.sql.*;

class AccessBaza
 {
   public void povezi()
     {
```


docsity.com

```
            try{   String dbUrl="jdbc:odbc:student";
                   String user= "root";
                   String pass="root";

                    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); // C:\Program Files\Java\jdk1.5.0_06\jre\lib\rt.jar
                   Connection c=DriverManager.getConnection(dbUrl,user,pass);
                   System.out.println("Program je povezan sa MS Access SUBP!!!");
                    c.close();
                } catch(ClassNotFoundException cnfe){ System.out.println("Nije ucitan upravljacki program: "+cnfe);    }
            catch(SQLException sqle){  System.out.println("Greska kod konekcije: "+sqle);  }
    }
}

class MySqlBaza
 {
   public void povezi()
    {
            try{
                  String dbUrl = "jdbc:mysql://127.0.0.1:3306/student";
                  String user= "root";
                  String pass="root";
                  Class.forName("com.mysql.jdbc.Driver"); // C:\Install\MySQL5.0\mysql-connector-java-3.1.12\mysql-connector-java-3.1.12.jar
                   Connection c = DriverManager.getConnection(dbUrl,user,pass);
                   System.out.println("Program je povezan sa MySQL SUBP!!!");
                   c.close();

                } catch(ClassNotFoundException cnfe){  System.out.println("Nije ucitan upravljacki program: "+cnfe);    }
                  catch(SQLException sqle){  System.out.println("Greska kod konekcije: "+sqle);  }
  }
}

class PoveziSaBazom
{ AccessBaza ab;
  MySqlBaza ma;
  public static void main(String arg[])
  { PoveziSaBazom psb;
    psb.ab = new AccessBaza();
    psb.ab.povezi();
    psb.ma = new MySqlBaza();
    psb.ma.povezi();
  }
}
```

Структура наведеног програма је:



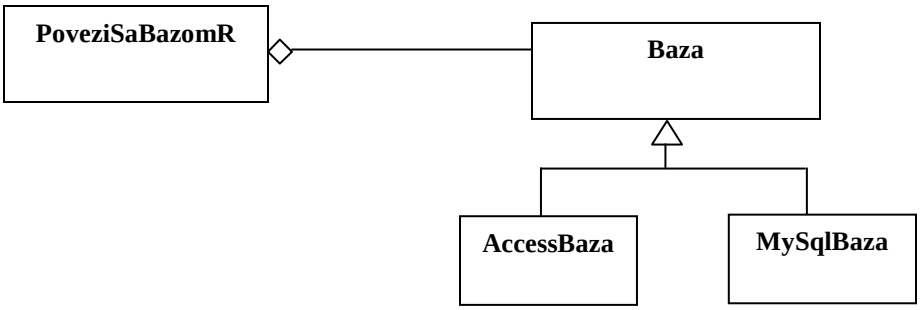
Уколико желимо да направимо генеричку методу помоћу које ће бити омогућено повезивање Јава програма са различитим системима за управљање базама података потребно је да препознамо променљива места у наведеним методама и да на њихово место поставимо генеричке наредбе (*vratiURL(), vratiDrajver(),…*). Наведене методе су апстрактне методе које се имплементирају у оквиру класа (*AccessBaza, MySqlBaza*) које се односе на конкретне системе за управљање базом података са којима ће Јава програм бити повезан.

```
import java.sql.*;
abstract class Baza
{
     public void povezi()
        {
          try{   String dbUrl = vratiUrl();
                 String user = vratiUserName();
```


docsity.com

```
                String pass = vratiPassword();
                Class.forName(vratiDrajver());
                Connection c = DriverManager.getConnection(dbUrl,user,pass);
                 System.out.println(vratiPoruku());
                 c.close();
                 } catch(ClassNotFoundException cnfe){ System.out.println("Nije ucitan upravljacki program: "+cnfe);    }
                   catch(SQLException sqle){  System.out.println("Greska kod konekcije: "+sqle);  }
            }

   abstract String vratiUrl();
   abstract String vratiDrajver();
   abstract String vratiPoruku();
   abstract String vratiUserName();
   abstract String vratiPassword();
}

class AccessBaza extends Baza{
   String vratiUrl(){return "jdbc:odbc:student";}
   String vratiDrajver(){return "sun.jdbc.odbc.JdbcOdbcDriver";}
   String vratiPoruku(){return "Program je povezan sa MS Access SUBP!!!";}
   String vratiUserName(){return "";}
   String vratiPassword(){return "";}
}

class MySqlBaza extends Baza{
   String vratiUrl(){return "jdbc:mysql://127.0.0.1:3306/student";}
   String vratiDrajver(){return "com.mysql.jdbc.Driver";}
   String vratiPoruku(){return "Program je povezan sa MySQL SUBP!!!";}
   String vratiUserName(){return "root";}
   String vratiPassword(){return "root";}
  }


class PoveziSaBazomR
{ Baza ba;

  public static void main(String arg[])
  { PoveziSaBazomR psb;
    AccessBaza ab = new AccessBaza();
    psb.ba = ab;
    psb.ba.povezi();
    MySqlBaza ma = new MySqlBaza();
    psb.ba = ma;
    psb.ba.povezi();
  }
}
```

Структура наведеног решења је:



Из наведеног примера можемо да изведемо следеће правило:
Генеричка метода се имплементира у окружењу структуре решења код патерна.

docsity.com

У следећем примеру се приказују слогови из две различите табеле коришћењем методе *prikazi()* која није генеричка, јер се мења сваки пут када се додаје нова табела, односно када се желе приказати слогови нове табеле.

```
import java.sql.*;

class AccessBaza
 {
   Connection c;
   public void povezi()
     {
          try {   String dbUrl="jdbc:odbc:student";
                  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); // C:\Program Files\Java\jdk1.5.0_06\jre\lib\rt.jar
                  c=DriverManager.getConnection(dbUrl,"","");
                  System.out.println("Program je povezan sa MS Access SUBP!!!");
              } catch(ClassNotFoundException cnfe){ System.out.println("Nije ucitan upravljacki program: "+cnfe);    }
                catch(SQLException sqle){  System.out.println("Greska kod konekcije: "+sqle);  }
      }

 void prikazi(String imeTabele)
     {   try {
               Statement naredba =c.createStatement();
               String upit="SELECT * FROM " + imeTabele;
               ResultSet rs=null;
                try { rs=naredba.executeQuery(upit);
                    } catch(SQLException sqle){ System.out.println("Greska u izvr. upita: "+sqle); }
                System.out.println("Trenutan izgled tabele " + imeTabele);
                 while(rs.next())
                     {  if (imeTabele.equals("Student"))
                           System.out.println(rs.getString("brind")+" " + rs.getString("ime")+" "+rs.getString("prezime"));
                        if (imeTabele.equals("Predmet"))
                           System.out.println(rs.getString("sifraPredmeta")+" " + rs.getString("nazivPredmeta"));
                     }
               naredba.close();
               c.close();
             } catch(SQLException se){ System.out.println("Nedozvoljena operacija: "+se);}
     }
}

class PrikaziSlogoveTabele
{
 public static void main(String arg[])
 { AccessBaza ab = new AccessBaza();
   ab.povezi();
   ab.prikazi("Student");

   ab.povezi();
   ab.prikazi("Predmet");
 }
}
```

Уколико желимо да направимо генеричку методу потребно је да следећи програмски код методе *prikazi()* учинимо непромењивим:

```
if (imeTabele.equals("Student"))
       System.out.println(rs.getString("brind")+" " + rs.getString("ime")+" "+rs.getString("prezime"));
if (imeTabele.equals("Predmet"))
    System.out.println(rs.getString("sifraPredmeta")+" " + rs.getString("nazivPredmeta"));
```

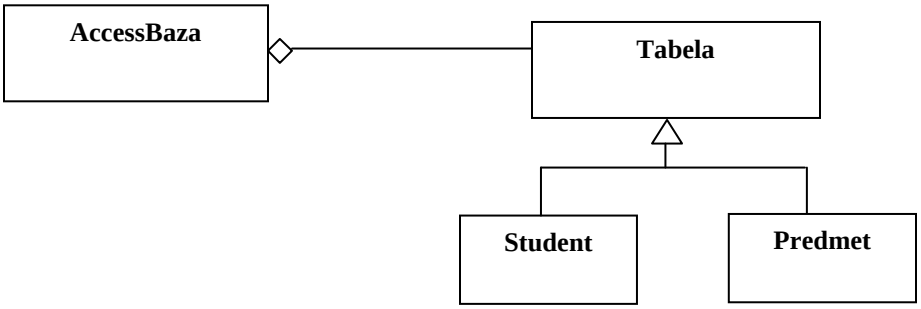
Наводим програм код кога је направљена метода *Prikazi()*.

docsity.com

```
import java.sql.*;

class AccessBaza
{
   Connection c;
   public void povezi(){
   try{   String dbUrl="jdbc:odbc:student";
          Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
          c=DriverManager.getConnection(dbUrl,"","");
          System.out.println("Program je povezan sa MS Access SUBP!!!");
     } catch(ClassNotFoundException cnfe){ System.out.println("Nije ucitan upravljacki program: "+cnfe);    }
       catch(SQLException sqle){  System.out.println("Greska kod konekcije: "+sqle);  }
}

  void prikazi(Tabela ta)
   {   try {
              Statement naredba =c.createStatement();
              String upit="SELECT * FROM " + ta.vratiImeTabele();
              ResultSet rs=null;
              try { rs=naredba.executeQuery(upit);
                  } catch(SQLException sqle){ System.out.println("Greska u izvr. upita: "+sqle); }
               System.out.println("Trenutan izgled tabele " + ta.vratiImeTabele());
               while(rs.next())
                 {  System.out.println(ta.vratiSlog(rs));
                 }
              naredba.close();
              c.close();
         } catch(SQLException se){ System.out.println("Nedozvoljena operacija: "+se);}
   }

}

class PrikaziSlogoveTabeleR
{
  public static void main(String arg[])
  { AccessBaza ab = new AccessBaza();
    ab.povezi();
    ab.prikazi(new Student());

    ab.povezi();
    ab.prikazi(new Predmet());
  }
}


interface Tabela
{
              String vratiImeTabele();
              String vratiSlog (ResultSet rs) throws SQLException;
}


class Student implements Tabela
{ public String vratiImeTabele(){return "Student";}
  public String vratiSlog (ResultSet rs) throws SQLException
            {return rs.getString("brind")+" " + rs.getString("ime")+" "+rs.getString("prezime");}
}


class Predmet implements Tabela
{ public String vratiImeTabele(){return "Predmet";}
  public String vratiSlog (ResultSet rs) throws SQLException
            {return rs.getString("sifraPredmeta")+" " + rs.getString("nazivPredmeta");}
}
```


docsity.com

Структура наведеног решења:



У наведеном примеру је такође генеричка метода имплементирана у окружењу структуре решења код патерна.

Задаци:

1. Довршити следећи програмски код, како би се на стандардном излазу добила следећа порука: *Збир два броја је: 8*
У постојећем програму не сме се избацити или ставити под коментар ниједна линија програма. Користити објекте из главног програма за добијање жељеног резултата. Додати наредбе на местима где су дате три тачке (...).

```
class Izraz
{ int clan1;   char operator;   int clan2;
  Izraz(int clan1p,int clan2p, char operatorp){clan1 = clan1p; clan2 = clan2p; operator = operatorp;}
}

interface KRacunaj { int Racunaj(Izraz iz);}
class Saberi implements KRacunaj {    public int Racunaj(Izraz iz) {return iz.clan1 + iz.clan2;}}
class Oduzmi implements KRacunaj {  public int Racunaj(Izraz iz) {return iz.clan1 - iz.clan2;}}

class T1Z1Zadatak
{  public static void main(String str[])
   {    KRacunaj rac = null;
        Saberi sa = new Saberi();
        Oduzmi od = new Oduzmi();
        Izraz iz = new Izraz(5,3,'+');
        if (iz.operator == '+')
              {...                                  }
        else
             if (iz.operator == '-')  { ...                                  }
             else {}

      System.out.println(rac.Racunaj(iz));
  }
}
```


docsity.com

2. Довршити следећи програмски код, како би се на стандардном излазу добила следећа порука: *Збир два броја је: 8*
У постојећем програму не сме се избацити или ставити под коментар ниједна линија програма. Користити објекте из главног програма за добијање жељеног резултата. Додати **једну** наредбу на месту где су дате три тачке (...)

```
class Izraz
{ int clan1;
  KRacunaj operator;
  int clan2;
  Izraz(int clan1p,int clan2p, KRacunaj operatorp){clan1 = clan1p; clan2 = clan2p; operator = operatorp;}
  int Racunaj(){return operator.Racunaj(this);}
  int vratiClan1(){return clan1;}
  int vratiClan2(){return clan2;}
}

interface KRacunaj
{ int Racunaj(Izraz iz);}

class Saberi implements KRacunaj
{public int Racunaj(Izraz iz) {System.out.print("Zbir dva broja je:"); return iz.vratiClan1() + iz.vratiClan2();}}

class Oduzmi implements KRacunaj
{public int Racunaj(Izraz iz) {System.out.print("Razlika dva broja je:"); return iz.vratiClan1() - iz.vratiClan2();}}

class T1Z2
{
  public static void main(String str[])
  { Saberi sa = new Saberi();
    Oduzmi od = new Oduzmi();
    Izraz iz = new Izraz(5,3,sa);
    ...
  }
}
```

3. Довршити следећи програмски код, како би се на стандардном излазу приказао низ бројева у растућем редоследу.
У постојећем програму не сме се избацити или ставити под коментар ниједна линија програма. Додати наредбу на месту где су дате три тачке (...).

```
class Niz
{   int niz[];
    Niz(int niz1[]) {niz=niz1;}
    void Sortiraj(OperatorPoredjenja op)
      { int pom = 0;
        for(int i=0; i<niz.length-1;i++)
              for(int j=i+1; j<niz.length;j++)
                  if( ...                                    )
                    { pom = niz[i]; niz[i]=niz[j];  niz[j]=pom; }
              }
             void prikaziNiz()
               { System.out.println("Elementi niza su:");
                 for(int i=0;i < niz.length;i++)    {System.out.println(niz[i]);}
                }
}

interface OperatorPoredjenja { boolean Poredi(int x,int y);}

class Vece implements OperatorPoredjenja
{ public boolean Poredi(int x,int y) {...                                    }}

class Manje implements OperatorPoredjenja
{ public boolean Poredi(int x,int y) {...                                    }}
```


docsity.com

```
class T1Z3
{ public static void main(String str[])
            { int n[] = {7,3,9,2,1,4,8};
              Niz niz = new Niz(n);
              Vece ve = new Vece();
              Manje ma = new Manje();
              niz.prikaziNiz();
             niz.Sortiraj(...);
             niz.prikaziNiz();
            }
}
```

4. У ниже наведеном примеру написати програмски код који неће мењати main() методу ако се дода нови геометријски облик (npr. Kvadrat) за који ће моћи да се види површина. Преко main() методе звати методу Povrsina() за било који геометријски облик.

```
import java.io.*;

class T1Z4P
{   public static void main(String str[]) throws IOException
     {  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        Krug kr = new Krug(7);
        Pravugaonik pr = new Pravugaonik(5,2);
        System.out.println("Izaberi geometrijski oblik za koji se racuna povrsina (1-krug, 2-pravugaonik):");
        String izbor = br.readLine();
             if (izbor.equals("1")==true)
                 kr.Povrsina();
             if (izbor.equals("2")==true)
                 pr.Povrsina();
     }
 }

 class Krug
  { int r;
    Krug(int r1) {r=r1;}
    void Povrsina(){System.out.println("Povrsina kruga je:" + r*r*3.14);}
  }

 class Pravugaonik
   { int a,b;
     Pravugaonik(int a1,int b1) {a=a1;b=b1;}
     void Povrsina(){System.out.println("Povrsina pravugaonika je:" + a*b);}
  }
```


docsity.com

4.4 Микро архитектура

4.4.1: GOF УЗОРИ ПРОЈЕКТОВАЊА

GOF узори се користе у фази пројектовања софтверског производа. Они помажу у именовању и опису **генеричких решења**, која могу бити примењена у различитим проблемским ситуацијама.

Код развоја софтвера, уопштено гледајући, прво треба да се схвати и разуме разматрани проблем (систем). Затим се врши његова анализа, да би се на крају вршило његово пројектовање и имплементација. У току пројектовања уочавају се класе пројектовање. Уколико се жели да наведене класе буду **флексибилне** (у смислу њиховог једноставног одржавања и надоградње), њих треба организовати помоћу узора пројектовања. Узоре треба користити и онда када се жели **динамичка измена функционалности** програма у току његовог извршавања.
Узори пројектовања омогућавају флексибилност класа и динимичку измену функционалности али истовремено они повећавају **сложеност** система.

Узори су подељени у 3 групе:

- **Креациони** узори помажу да се изгради систем независно од тога како су објекти, креирани, компоновани и репрезентовани.
- **Структурни** узори описују сложене структуре међусобно повезаних класа и објеката.
- **Узори понашања** описују начин на који класе или објекти сарађују и распоређују одговорности.

У даљем тексту ћемо детаљно објаснити све три групе узора.

Интересантни делови из књиге Design patterns:

- Ваше решење (design) треба да буде:
  а) специфично (**specific**), односно прилагођено проблему и
  б) довољно опште (**general**) да укаже на будуће проблеме и захтеве.
  Треба направити такво решење да се избегне или минимизира поновно пројектовање решења (**redesign**).
  Искуства објектно-оријентисаних пројектаната говоре да *поновно употребљиво* (**reusable**) и *прилагодљиво* решење (**flexible design**) је тешко да се добије из првог пута. Пре него што се добије такво решење потребно је да се оно неколико пута користи (**reuse**) и модификује док се не добије довољно генеричко решење које може да покрије класу проблема.

- Колико пута се десило да решење неког проблема код вас пробуди *dežavi* осећај, да сте већ раније решили неки проблем али не знате тачно где и како? Уколико се сећате детаља предходног проблема и начина како сте га решили тада ви можете да користите то *искуство* у решавању нових проблема уместо да их опет изнова истражујете. Сврха књиге ***Design patterns*** јесте да *запамти искуства* у пројектовању објектно-оријентисаног софтвера и та искуства се називају **узори пројектовања** (**design patterns.**)

- **Узори пројектовања [**GOF,str.3**]** *су описи комуникација између објеката, односно класа који су прилагођени* ***(customized)*** *да реше генерални проблем у посебном контексту.*

Узори пројектовања идентификују класе и појављивања, њихове улоге и сарадњу и расподелу (**distribution**) одговорности (**responsibilities**).

docsity.com

ПК: УЗОРИ ЗА КРЕИРАЊЕ ОБЈЕКАТА (CREATIONAL PATTERNS)

*Узори за креирање објеката апстракују процес инстанцијације (**instantiation process**), односно процес **креирања** објеката. Они дају велику прилагодљивост (**flexibility**) у томе шта (**what**) ће бити креирано, ко (**who**) ће то креирати, како (**how**) ће то бити креирано и када (**when**).*

Постоје следећи узори за креирање објеката:
1. ***Abstract Factory*** - Обезбеђује интерфејс за креирање фамилије повезаних или зависних објеката без навођења (специфицирања) њихових конкретних класа.

2. ***Builder*** - Дели конструкцију сложеног (комплексног) објекта од његове репрезентације, тако да исти конструкциони процес може да креира различите репрезентације.

3. ***Factory Method*** - Дефинише интерфејс за креирање објеката, али омогућава подкласама да донесу одлуку коју класу ће истанцирати. Фацтору метход преноси надлежност инстанцирања објеката са класе на подкласе.

4. ***Prototype*** - Одређује (специфицира) врсте објеката које ће бити креиране коришћењем прототипског појављивања и креира нове објекте копирањем тог прототипа.

5. ***Singleton*** - Обезбеђује класи само једно појављивање и глобални приступ до ње.

У ниже наведеном тексту даћемо детаље везане за наведене узоре.

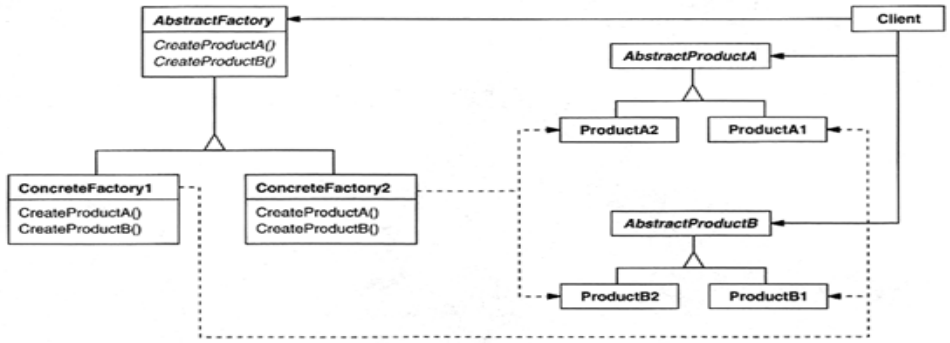
docsity.com

ПК1: Abstract Factory патерн

Дефиниција:
Обезбеђује интерфејс за креирање фамилије повезаних или зависних објеката без навођења (специфицирања) њихових конкретних класа.

Појашњење дефиниције:
Обезбеђује интерфејс (*AbstractFactory*) за креирање *(CreateProductA(), CreateProductB())* фамилије повезаних или зависних објеката (*AbstractProductA, AbstractProductB*) без навођења (специфицирања) њихових конкретних класа *(ProductA1, ProductA2,ProductB1,ProductB2).*

Структура[20] factory патерна:



Учесници*:*

- **AbstractFactory**
  Декларише интерфејс за операције које креирају <<производе>>.
- **ConcreteFactory**
  Имплементира операције <<AbstractFactory>> интерфејса, којима се креирају конкретни <<производи>>.
- **AbstractProduct**
  Декларише интерфејс за <<производе>>.
- **ConcreteProduct**
  - Дефинише <<производе>> који ће бити креирани преко конкретних <<faktory-a>>.
  - Имплементира операције <<AbstractProduct>> интерфејса.
- **Client**
  Користи <<AbstractFactory>> и <<AbstractProduct>> класе (интерфејсе) за креирање <<финалног производа>>.

[20] Када је писана књига Design Patterns коришћена је Booch-ова нотација код описа дијаграма класа. Наведену нотацију ми смо преузели код спецификације GOF узора.

docsity.com

Пример AbstractFactory патерна:

Кориснички захтев PAF1: *Управа Факултета је послала захтев Лабораторији за софтверско инжењерство да пошаље елементе понуде за израду софтверског система последипломских студија ФОН-а:*
а) Програмски језик у коме ће се развијати програм.
б) Систем за управљање базом података у коме ће се чувати подаци.
Након прихватања елемената понуде Управа Факултета ће направити (саставити) понуду у целини[21].

Уколико посматрамо кориснички захтев можемо приметити да Управа тражи од Лабораторије за С.И. да креира елементе понуде, односно програмски језик и СУБП у којима ће се реализовати софтверски систем. Лабораторију за СИ (SILAB) можемо представити на следећи начин:

```
// Улога: Декларише интерфејс за операције које креирају <<елементе понуде >>.
interface SILAB // AbstractFactory
{ ProgramskiJezik kreirajProgramskiJezik();
  SUBP kreirajSUBP();
}
```

Интерфејс *SILAB* је одговоран за дефинисање операција *kreirajProgramskiJezik* и *kreirajSUBP* помоћу којих се креирају елементи понуде[22].
У наведеном интерфејсу се може приметити да операције *kreirajProgramskiJezik* и *kreirajSUBP* враћају програмски језик и СУБП, који ће такође бити представљени преко интерфејса:

```
/*Улога: Декларише интерфејс за <<елементе понуде>>. */
interface ProgramskiJezik  // AbstractProductA
{String vratiProgramskiJezik();}

interface SUBP // AbstractProductB
{String vratiSUBP();}
```

Сви елементи захтева за понудом (понуда, програмски језик, СУБП) представљени су преко интерфејса[23].

Лабораторија за софтверско инжењерство је направила два тима. Један је оријентисан ка Јави, док је други оријентисан ка VB-у. Оба тима треба да одреде конкретан програмски језик и СУБП који ће дати у понуди[24].

```
/* Улога: Имплементира операције <<SILAB>> интерфејса, којима се креирају конкретни <<елементи понуде>>.*/
class JavaTimPonuda implements SILAB // ConcreteFactory1
{ public ProgramskiJezik kreirajProgramskiJezik(){return new Java();}
  public SUBP kreirajSUBP() {return new MySQL();}
}

class VBTimPonuda implements SILAB // ConcreteFactory1
{ public ProgramskiJezik kreirajProgramskiJezik(){return new VB();}
  public SUBP kreirajSUBP() {return new MSAccess();}
}
```

---

[21] **Шта** ће бити креирано: **Понуда**
Ко ће креирати понуду: **Управа**
Како ће се креирати понуда: видети методу *Kreiraj()*.
Када ће се креирати понуда: видети методу *main()*.

[22] AbstractFactory интерфејс (*SILAB*) преноси одговорност за креирање објеката до његових ConcreteFactory подкласа (*JavaTimPonuda*, *VBTimPonuda*).

[23] Интерфејс је концепт у ОО програмирању који дефинише шта треба да се ради, без улажења у имплементацију, како ће то да се уради. Класе које имплементирају интерфејс су одговорне за имплементацију операција интерфејса.

[24] Сваки конкретан faktory има посебну (специфичну) имплементацију код креирања производа.

docsity.com

У методама *kreirajProgramskiJezik()* и *kreirajSUBP()* су креирани конкретни програмски језици: *Java* и *VB:*

```
/* Улоге: а) Дефинише <<елементе понуде (Java,VB,MySQL и MSAccess)>> који ће бити креирани
          преко конкретних <<тимова за понуде(JavaTimPonuda и VBTimPonuda)>>.
          б) Имплементира операције <<ProgramskiJezik i SUBP>> интерфејса. */
class Java implements ProgramskiJezik // Product A1
{ public String vratiProgramskiJezik(){return "Java";}}

class VB implements ProgramskiJezik // Product A2
{ public String vratiProgramskiJezik(){return "VB";}}
```

као и конкретни СУБП: *MySQL* и *MSAccess*:

```
class MySQL implements SUBP // Product B1
{ public String vratiSUBP(){return "MySQL";}}

class MSAccess implements SUBP // Product B2
{ public String vratiSUBP(){return "MS Access";}}
```

На крају управа Факултета (Client[25]) приказује обе понуде преко *main* методе класе *UpravaFakulteta.* У методи *kreiraj()* се прави конкретна понуда, прво за Јава тим а после тога и за VB тим.

```
// Улога: Користи <<Ponuda>> и <<ProgramskiJezik и SUBP>> интерфејсе за креирање <<понуде>>.
class UpravaFakulteta  // Client
{
  SILAB sil; // Abstract Factory
  UpravaFakulteta(SILAB sil1){sil = sil1;}

   public static void main(String args[])
            {  UpravaFakulteta uf;
               JavaTimPonuda jat = new JavaTimPonuda();  // ConcreteFactory1
               uf = new UpravaFakulteta(jat);
               System.out.println("Ponuda java tima: " + uf.Kreiraj());

                VBTimPonuda vbt = new VBTimPonuda(); // ConcreteFactory2
                uf = new UpravaFakulteta(vbt);
                 System.out.println("Ponuda VB tima: " + uf.Kreiraj());
            }

      String Kreiraj()[26]
            {   ProgramskiJezik pj = sil.kreirajProgramskiJezik();
                SUBP subp = sil.kreirajSUBP();
                return "Programski jezik-" + sil.vratiProgramskiJezik() + "  SUBP-" + sil.vratiSUBP();
          }
}
```

Метода *kreiraj()* је генерички урађена јер је креирање програмског језика и СУБП-а везано за операције интерфејса. Управа факултета (Client) креира понуду преко апстрактног интерфејса (*SILAB*) и она не види како конкретне класе креирају објекте. Ово је добар пример једног од главних принципа поновног коришћења програмског кода у ОО пројектовању:

Програмирати према интерфејсу а не према имплементацији (Program to an interface, not an implementation [GOF, стр.18])

Предности AbstractFactory узора

[25] Klijent treba da koristi različite konkretne faktore kada želi da kreira različite proizvode.

[26] 1. Управа је одговорна за контролу креирања понуде.
2. Тимови су одговорни за креирање елемената понуде.
3. Управа је одговорна за састављање понуде у целини.

docsity.com

Предности AbstractFactory узора се огледају у томе што додавање нове ConcreteFactory класе (у нашем случају то је нови тим који даје понуду, нпр: CTimPonuda) не захтева промену у постојећим класама и интерфејсима.

```
class CTimPonuda implements SILAB // novi ConcreteFactory
{ public ProgramskiJezik kreirajProgramskiJezik(){return new C();}
  public SUBP kreirajSUBP() {return new Oracle();}
}
```

Задатак ZAF1: Извршити промене у програму *PAF1* када се дода нови тим CTimPonuda.

Недостаци AbstractFactory узора
Тешко се додају нове врсте производа до AbstractFactory узора. То је због тога што AbstractFactory (Ponuda) интерфејс има фиксан скуп производа који може да креира (Programski jezik и SUBP). Увођење нове врсте производа (нпр. OperativniSistem) захтева проширење интерфејса AbstractFactory класе и промену свих њених подкласа.

Задатак ZAF2: Извршити промене у програму *PAF1* када се дода нови елемент понуде Operativni sistem.

docsity.com

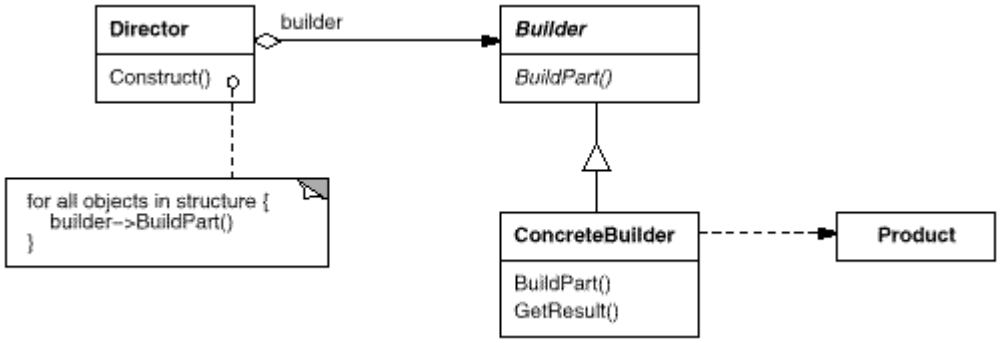
ПК2: Builder патерн

Дефиниција*:*
Дели конструкцију сложеног (комплексног) објекта од његове репрезентације, тако да исти конструкциони процес може да креира различите репрезентације.

Појашњење дефиниције:
Дели одговорност за контролу конструкције (*Director*) сложеног (комплексног) објекта од одговорности за реализацију његове репрезентације-конкретне конструкције (*Builder*), тако да исти конструкциони процес (*Direktor.Construct()*) може да креира различите репрезентације конкретне конструкције (*ConcreteBuilder.BuildPart()*) у зависности од ConcreteBuilder-а.

Напомена: Код GOF дефиниције појам репрезентација указује на конкретну конструкцију (производ) који ће се добити као резултат контрукционог процеса.

Структура Builder патерна:



Учесници:

- Builder
  Специфицира апстрактни интерфејс за креирање <<делова производа[27]>>.
- ConcreteBuilder
  Конструише и групише <<делове производа>> имплементирајући <<Builder>> интерфејс.
  Обезбеђује интерфејс за узимање <<производа>>.
- Director
  Конструише објекат (<<производ>>) коришћењем <<Builder>> интерфејса.
  Контролише конструкцију <<производа>> коришћењем <<Builder>> интерфејса.
- Product
  - Репрезентује сложен (комплексан) објекат (<<производ>>) који се конструише.
  - Укључује класе (интерфејсе) који дефинишу <<делове производа>>, укључујући интерфејсе за груписање делова у финални резултат (<<производ>>).

Пример Builder патерна:

Кориснички захтев PBU1: *Управа Факултета је послала захтев Лабораторији за софтверско инзењерство да направи (састави) 2 понуде за израду софтверског система последипломских студија ФОН-а. У понуди треба се наведе:*
а) Програмски језик у коме ће се развијати програм.
б) Систем за управљање базом података у коме ће се чувати подаци.
Управа Факултета ће надзирати (контролисати) израду понуда.

```
package Builder;

// Улога: Контролише конструкцију <<понуде>> коришћењем Builder интерфејса.
```

[27] На слици то је Product.

docsity.com

```
class UpravaFakulteta   // Director
{
  SILAB sil; // Builder
  UpravaFakulteta(SILAB sil1){sil = sil1;}

  // Контролише конструкцију <<понуде>> коришћењем Builder интерфејса (pon).
  void Konstruisi()[28]
    { sil.kreirajProgramskiJezik();
      sil.kreirajSUBP();
      sil.kreirajPonudu();
    }

   public static void main(String args[])
             {  UpravaFakulteta uf;
                JavaTimPonuda jat = new JavaTimPonuda();  // ConcreteBuilder1
                uf = new UpravaFakulteta(jat);
                uf.Konstruisi();
                 System.out.println("Ponuda java tima: " + jat.vratiPonudu());
                VBTimPonuda vbt = new VBTimPonuda(); // ConcreteBuilder2
                uf = new UpravaFakulteta(vbt);
                uf.Konstruisi();
                System.out.println("Ponuda VB tima: " + vbt.vratiPonudu())
             }
}

// Улога: Специфицира апстрактни интерфејс за креирање <<елемената понуде>>.
interface SILAB // Builder
{ void kreirajProgramskiJezik();
  void kreirajSUBP();
   void kreirajPonudu();
  String vratiPonudu();}
/*Улоге: а) Репрезентује сложену  <<понуду>>  која се конструише.
          б) Укључује класе (интерфејсе) које дефинишу  <<елементе понуде>> */
class PonudaS { ProgramskiJezik pj;  SUBP subp;}

/* Улоге: а) Конструише и групише  <<елементе понуде>> имплементирајући
              <<SILAB>>интерфејс.
          б) Обезбеђује интерфејс за узимање <<понуде>>.*/
class JavaTimPonuda implements SILAB // ConcreteBuilder1
{  // Обезбеђује интерфејс за узимање <<понуде>>.
  PonudaS elpon; // elementi ponude
  String ponuda;
  JavaTimPonuda() {elpon = new PonudaS();}
  // Конструише и групише делове <<понуде>>.
  public void kreirajProgramskiJezik(){elpon.pj = new Java();}
  public void kreirajSUBP() {elpon.subp = new MySQL();}
  public void kreirajPonudu() { ponuda = "Programski jezik-" + elpon.pj.vratiProgramskiJezik() + "
                                  SUBP-" + elpon.subp.vratiSUBP();}
  public String vratiPonudu(){return ponuda;}
}

class VBTimPonuda implements SILAB // ConcreteBuilder2
{ PonudaS elpon;
  String ponuda;
  VBTimPonuda(){elpon = new PonudaS();}
  public void kreirajProgramskiJezik(){elpon.pj = new VB();}
  public void kreirajSUBP() {elpon.subp = new MSAccess();}
  public void kreirajPonudu() { ponuda = "Programski jezik-" + elpon.pj.vratiProgramskiJezik() + "
                                      SUBP-" + elpon.subp.vratiSUBP();
  public String vratiPonudu(){return ponuda;}
}

// Наведени интерфејси и класе су преузети из примера за Abstract Factory узор.
// ***************************************************************
interface ProgramskiJezik
```

[28] 1. Управа је одговорна за контролу креирања понуде .
2. Тимови су одговорни за креирање елемената понуде.
3. Тимови су одговорни за састављање понуде у целини.

docsity.com

```
{String vratiProgramskiJezik();}

class Java implements ProgramskiJezik
{ public String vratiProgramskiJezik(){return "Java";}}

class VB implements ProgramskiJezik
{ public String vratiProgramskiJezik(){return "VB";}}

// ***************************************************************

interface SUBP // AbstractProductB
{String vratiSUBP();}

class MySQL implements SUBP
{ public String vratiSUBP(){return "MySQL";}}

class MSAccess implements SUBP
{ public String vratiSUBP(){return "MS Access";}}

// ***************************************************************
```

Питање PBU1: Које су предности Builder узора?

Питање PBU2: Који су недостаци Builder узора?

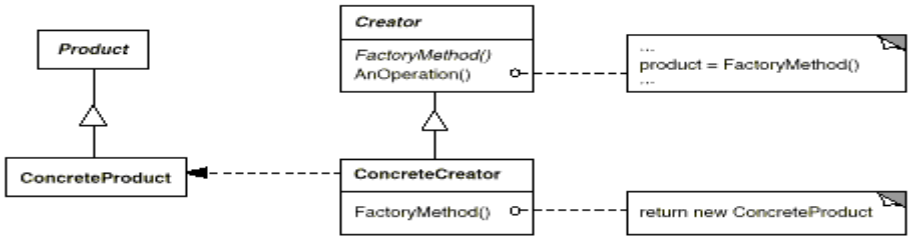
docsity.com

КП3: Factory method узор

Дефиниција
Дефинише интерфејс за креирање објеката, али омогућава подкласама да донесу одлуку коју класу ће истанцирати. Factory method преноси надлежност инстанцирања објеката са класе на подкласе.

Појашњење дефиниције:
Дефинише интерфејс за креирање објеката *(Creator)*, али омогућава подкласама *(ConcreteCreator)* да донесу одлуку коју класу (производ) ће истанцирати. Factory method преноси надлежност инстанцирања објеката са класе *(Creator)* на подкласе*(ConcreteCreator)*.

Структура Factory method патерна:



Учесници:

- **Product**
  Дефинише интерфејс <<објеката >> које <<factory>> метода креира.
- **ConcreteProduct**
  Имплементира <<Product>> интерфејс.
- **Creator**
  - Декларише <<factory>> методу, која враћа објекат класе <<Product>>.
  - Позива <<factory>> методу да креира <<Product>> објекат.
- **ConcreteCreator**
  Прекрива <<factory>> методу и враћа појављивање класе <<ConcreteProduct>>.

Пример Factory method узора:

/ **Кориснички захтев PFM1:** Управа Факултета је послала захтев Лабораторији за софтверско инжењерство да направи (састави) 2 понуде за израду софтверског система последипломских студија ФОН-а. У понуди треба се наведе:*
а) Програмски језик у коме ће се развијати програм.
б) Систем за управљање базом података у коме ће се чувати подаци.
*Управа Факултета **неће надзирати (контролисати)** израду понуда.*
***Надзор** израде понуде **је пренет** на Лабораторију за софтверско инзењерство.*
**/*

```
package FactoryMethod;

class UpravaFakulteta   // Client
{
  SILAB sil; // Creator
  UpravaFakulteta(SILAB sil1){sil = sil1;}

   void  kreiraj() { sil.kreiraj();}[29]

   public static void main(String args[])
            {  UpravaFakulteta uf;
```

[29] 1. Тимови су одговорни за контролу креирања понуде.
2. Тимови су одговорни за креирање елемената понуде.
3. Тимови суодговорни за састављање понуде у целини.

docsity.com

```
            JavaTimPonuda jat = new JavaTimPonuda();  // ConcreteCreator1
            uf = new UpravaFakulteta(jat);
            uf.kreiraj();
            System.out.println("Ponuda java tima: " + jat.vratiPonudu());

          VBTimPonuda vbt = new VBTimPonuda(); // ConcreteCreator2
          uf = new UpravaFakulteta(vbt);
          uf.kreiraj();
          System.out.println("Ponuda VB tima: " + vbt.vratiPonudu());
        }
}

/*Улога: Дефинише интерфејс <<понуда>>  које  <<kreirajPonudu()>> метода креира.*/
interface APonudaS {} // Product

/*Улога: Имплементира <<AponudaS>> интерфејс. */
class PonudaS implements APonudaS // ConcreteProduct
{ ProgramskiJezik pj;  SUBP subp;}

/* Улоге: а)Декларише <<kreirajPonudu()>> методу, која враћа објекат класе <<PonudaS>>.
          б) Позива <<kreirajPonudu()>> методу да креира <<Product>> објекат.
*/
abstract class SILAB // Creator
{ APonudaS elpon;
   String ponuda;

// Позива <<kreirajPonudu()>> методу да креира <<Product>> објекат.
 void kreiraj(){elpon = kreirajPonudu();}
  //Декларише <<kreirajPonudu()>> методу, која враћа објекат класе <<PonudaS>>.
  abstract APonudaS kreirajPonudu();
  abstract String vratiPonudu();
}

/* Улоге: Прекрива <<kreirajPonudu()>> методу   и враћа појављивање класе <<PonudaS>>.*/
class JavaTimPonuda extends SILAB // ConcreteCreator1
{
  APonudaS kreirajPonudu()
                    {  PonudaS elpon1 = new PonudaS();
                       elpon1.pj = new Java();
                       elpon1.subp = new MySQL();
                        ponuda = "Programski jezik-" + elpon1.pj.vratiProgramskiJezik() + "  SUBP-" +
                                  elpon1.subp.vratiSUBP();
                       return elpon1;
                    }
  String vratiPonudu(){ return ponuda;}
}

class VBTimPonuda extends SILAB // ConcreteCreator2
{
  APonudaS kreirajPonudu()
                    {  PonudaS elpon1 = new PonudaS();
                          elpon1.pj = new VB();
                          elpon1.subp = new MSAccess();
                       ponuda = "Programski jezik-" + elpon1.pj.vratiProgramskiJezik() + "  SUBP-" +
                                   elpon1.subp.vratiSUBP();
                       return elpon1;
                     }
  String vratiPonudu(){ return ponuda;}
}

// Наведени интерфејси и класе су преузети из примера за Abstract Factory узор.
// ***************************************************************
interface ProgramskiJezik
{String vratiProgramskiJezik();}

class Java implements ProgramskiJezik
{ public String vratiProgramskiJezik(){return "Java";}}

class VB implements ProgramskiJezik
{ public String vratiProgramskiJezik(){return "VB";}}
```


docsity.com

```
// ************************************************************

interface SUBP
{String vratiSUBP();}

class MySQL implements SUBP
{ public String vratiSUBP(){return "MySQL";}}

class MSAccess implements SUBP
{ public String vratiSUBP(){return "MS Access";}}

// ************************************************************
```

Питање PFM1: Које су предности Factory method узора?

Питање PFM2: Који су недостаци Factory method узора?

Компаративна анализа AbstractFactory, Builder и FactoryMethod узора

У примеру који је дат, Управа Факултета код:

- **AbstractFactory узора** шаље захтев Лабораторији за софтверско инжењерство да направи и пошаље елементе понуде. Управа Факултета преузима на себе обавезу да контролише израду понуде и да састави понуду у целини.
- **Builder узора** шаље захтев Лабораторији за софтверско инжењерство да направи елементе понуде и понуду у целини. Управа Факултета преузима на себе обавезу да надзире (контролише) израду понуда.
- **Factory method узора** шаље захтев Лабораторији за софтверско инжењерство да направи елементе понуде и понуду у целини и да надзире процес прављења понуде. То значи да Управа Факултета у потпуности преноси одговорност надзора и прављења понуде на Лабораторију за софтверско инжењерство.

На основу наведене анализе може да се закључи да код:

- **AbstractFactory узора - Client** надзире процес прављења производа и израђује производ у целини. Израда делова производа се преносе на **ConcreteFactory** класе.
- **Builder uzora – Direktor** надзире процес прављења производа. Израда делова и целине производа се преноси на **ConcreteBuilder** класе.
- **Factory method узора – ConcreteCreator** надзире процес прављења производа и израђује делове и целину производа.

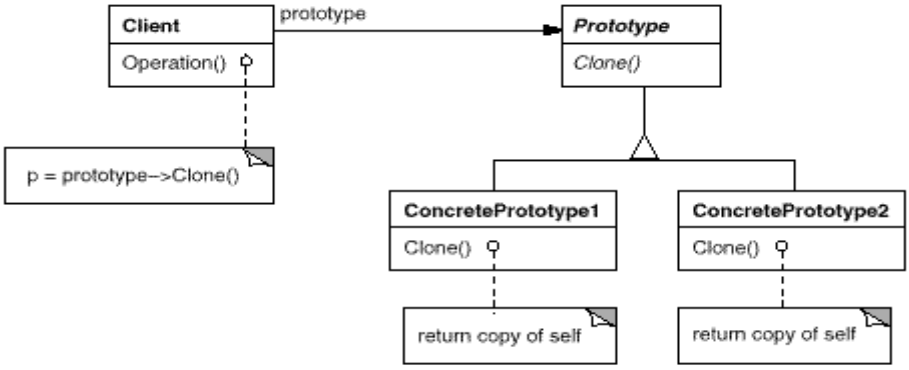
docsity.com

КП4: Prototype патерна

Дефиниција

Одређује (специфицира) врсте објеката које ће бити креиране коришћењем прототипског појављивања и креира нове објекте копирањем тог прототипа.

Структура Prototype патерна



Учесници:

- **Prototype**
  Декларише интерфејс за споствено клонирање.
- **ConcretePrototype**
  Имплементира операцију за сопствено клонирање.
- **Client**
  Креира нови <<објекат (производ)>> захтевајући од прототипа да се клонира.

Пример Прототупе узора:

Кориснички захтев PPR1: *Управа Факултета након пријема понуда од Јава и VB тима треба да направи копију сваке од понуда.*

```
package Prototype;

// Улога: Декларише интерфејс (апстрактну класу) PrototipTekstaPonude за сопствено клонирање.
abstract class PrototipTekstaPonude  // Prototype
{ String tekstPonude;
  abstract PrototipTekstaPonude Clone();
  abstract String vratiTekstPonude();
}

// Улога: Имплементира операцију Clone() за сопствено клонирање
class TekstPonude extends PrototipTekstaPonude // ConcretePrototype1
{ TekstPonude(String tekstPonude1) {tekstPonude = new String(tekstPonude1);
                                    tekstPonude = "Ponuda Java tima: " + tekstPonude;}
  TekstPonude(TekstPonude tekstPonude1) { tekstPonude = new
                                    String(tekstPonude1.tekstPonude);}
  TekstPonude Clone(){return new TekstPonude(this);}
  String vratiTekstPonude(){return tekstPonude;}
}

class TekstPonude1 extends PrototipTekstaPonude // ConcretePrototype2
{
  TekstPonude1(String tekstPonude1) {tekstPonude = new String(tekstPonude1);
                    tekstPonude = "Ponuda VB tima: " + tekstPonude;}
  TekstPonude1(TekstPonude1 tekstPonude1) { tekstPonude = new
```


docsity.com

```
                                                    String(tekstPonude1.tekstPonude);}
  TekstPonude1 Clone(){return new TekstPonude1(this);}
  String vratiTekstPonude(){return tekstPonude;}
}


// Улога: Креира нову копију <<понуде>> захтевајући од прототипа да се клонира.
class UpravaFakulteta   // Client
{
   public static void main(String args[])
            {
              JavaTimPonuda jat = new JavaTimPonuda();
              VBTimPonuda vbt = new VBTimPonuda();
              PrototipTekstaPonude tekstPonude;

              TekstPonude tekstPonude1 = new TekstPonude(jat.vratiPonudu());
              TekstPonude1 tekstPonude2 = new TekstPonude1(vbt.vratiPonudu());

              tekstPonude = tekstPonude1;
              // Kreira novu kopiju <<ponude>> zahtevajuci od prototipa da se klonira.
              tekstPonude = tekstPonude.Clone();
              // Isti efekat bi bio: tekstPonude = tekstPonude1.Clone();
              System.out.println(tekstPonude.vratiTekstPonude());

              tekstPonude = tekstPonude2;
              // Kreira novu kopiju <<ponude>> zahtevajuci od prototipa da se klonira.
              tekstPonude = tekstPonude.Clone();
              // Isti efekat bi bio: tekstPonude = tekstPonude2.Clone();
              System.out.println(tekstPonude.vratiTekstPonude());

   }
}

class JavaTimPonuda
{ String vratiPonudu(){return "Programski jezik - Java  SUBP - MySQL";}
}

class VBTimPonuda
{ String vratiPonudu(){return "Programski jezik - VB  SUBP - MSAccess";}
}
```

Питање **PPR1:** Које су предности Prototype патерна?

Питање **PPR2:** Који су недостаци Prototype патерна?

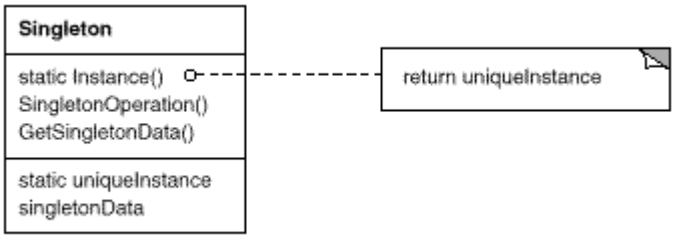
docsity.com

КП5: Singleton патерн

Дефиниција

Обезбеђује класи само једно појављивање и глобални приступ до ње.

Структура Singleton патерна:



Учесници:

- **Singleton** – дефинише **Instance()** операцију која омогућава клијентима приступ до њеног јединственог појављивања.

Пример Syngleton патерна:

```
/*Кориснички захтев PSI1: Онемогућити да се добију две понуде од Јава тима. */

package Singleton;

class UpravaFakulteta {
  public static void main(String args[])     {
                JavaTimPonuda jat = JavaTimPonuda.Instance();
                jat = JavaTimPonuda.Instance();
   }
}

// Улога: Дефинише  Instance() операцију која омогућава клијентима приступ до
// њеног јединственог појављивања (jtp).
class JavaTimPonuda
{ static boolean jedinstvenoPojavljivanje=false;
  static JavaTimPonuda jtp;
  static JavaTimPonuda Instance()
        {     if (jedinstvenoPojavljivanje==false)
                { System.out.println("Kreira se nova ponuda");
                  jedinstvenoPojavljivanje=true;
                  jtp = new JavaTimPonuda();
                  }
                else
                  { System.out.println("Ponuda je vec kreirana"); }
                return jtp;
        }
}
```

Питање **PSI1:** Које су предности Singletone узора?
Питање **PSI2:** Који су недостаци Singletone узора?

docsity.com

Задаци:

T2Z1: *Довршити следећи програмски код како би био задовољен следећи кориснички захтев: Направити сопствену класу Изузетак која ће обрадити следеће изузетке који се дешавају над низом:*
а) Прекорачење броја елемената низа.
б) Убацивање негативне вредности.
Над низом су дефинисане две операције: убаци и избаци. Користити лифо принцип код убацивања и избацивања елемената низа.

```
import java.io.*;

class Izuzetak extends Exception
{  String poruka;
    Izuzetak(String poruka1) { ...                                                    }
  public ...              toString(){return poruka;}
}

class T2Z1
{ int n[];
  int brojElemenataNiza=0;
  void ubaci(int novi) throws Izuzetak
    {    if (novi < 0)  { ...                                                    }
         if (brojElemenataNiza >= n.length)
                 { ...                                                          }
         for(int i=brojElemenataNiza; i>0; i--)
               { ...                                                            }
          n[0]=novi;
          brojElemenataNiza++;
    }

  int izbaci()
     { int pom;
       pom = n[0];
       for(int i=0; i<brojElemenataNiza-1; i++)  {...                                  }
       ...
       return pom;
     }

  void prikaziElemente()
    { System.out.println("Elementi niza:");
     for(int i=0; i<brojElemenataNiza; i++) {System.out.println("Element:" + n[i]);}
    }

    public static void main(String arg[]) throws IOException
     {
      T2Z1 t2z1 = new T2Z1();
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      System.out.println("Ubaci broj elemenata niza:");
      int brelniza = Integer.parseInt(br.readLine());
      t2z1.n = new int[brelniza];
      System.out.println("Ubaci elemente niza:");
      while(true)
       { try { int pom = Integer.parseInt(br.readLine());
                  if (pom == 99)  break;
                  t2z1.ubaci(pom);
             } catch (Izuzetak iz) {System.out.println(iz);}
       }
       t2z1.izbaci();
       t2z1.prikaziElemente();
     }
}
```


docsity.com

T2Z2: *Довршити следећи програмски код, како би се на стандардном излазу добила следећа порука: Povrsina kruga je: 100.94*
У постојећем програму не сме се избацити или ставити под коментар ниједна линија програма. Додати наредбу на месту где су дате три тачке (...)

```
class T2Z2
{
 public static void main(String str[])
   { GeometrijskiOblik go = new ...
     System.out.println("Povrsina kruga je:" + go.Povrsina());
   }
}

interface GeometrijskiOblik
{ ...
}

class Krug ...
{ ...
  Krug(double r1){r=r1;}
  public double Povrsina() {return r*r*3.14;}
}
```

T2Z3: Довршити следећи програмски код, како би се на стандардном излазу добила следећа порука: *Povrsina kruga je: 100.94*
У постојећем програму не сме се избацити или ставити под коментар ниједна линија програма. Додати наредбу на месту где су дате три тачке (...)

```
class T2Z3
{ public static void main(String str[])
   { GeometrijskiOblik go = ...
     System.out.println("Povrsina kruga je:" + go.Povrsina());
   }
}

abstract class GeometrijskiOblik
{...
}

class Krug ...
{ …
  Krug(double r1){r=r1;}
  public double Povrsina() {return r*r*3.14;}
}
```

T2Z4: Довршити следећи програмски код, како би се на стандардном излазу добила следећа порука: *Konkretni server2 je obradio poziv*
У постојећем програму не сме се избацити или ставити под коментар ниједна линија програма. Додати наредбу на месту где су дате три тачке (...)

```
class T2Z4
{
  public static void main(String arg[])
    { ...              ks =  ...
      Klijent kl = new Klijent(ks);
      kl.obradi();
    }
}

class Klijent
{ ApstraktniServer as;
```


docsity.com

```
  Klijent(ApstraktniServer as1) {as = as1;}
  void obradi(){...                                                    }
}


interface ApstraktniServer
{ void obradi();
}

class KonkretniServer1 ...
{
  public void obradi(){...                                             }
}

class KonkretniServer2 ...
{
 public void obradi(){...                                              }
}
```


docsity.com

СП: СТРУКТУРНИ ПАТЕРНИ

Структурни патерни описују сложене структуре међусобно повезаних класа и објеката.

Постоје следећи структурни патерни:
1. Adapter патерн - Адаптер патерн конвертује интерфејс неке класе у други интерфејс који клијент очекује. Другим речима, он прилагођава некомпатибилне интерфејсе.
2. Bridge патерн - Одваја (декуплује) апстракцију од њене имплементације тако да се оне могу мењати независно.
3. Composite патерн - Објекти се састављају (компонују) у структуру стабла како би представили хијерархију целине и делова. Composite узор омогућава да се једноставни и сложени (компоновани) објекти третирају јединствено.
4. Decorator патерн - Придружује одговорност до објекта динамички. Декоратор проширује функционалност објекта динамичким додавањем функционалности других објеката.
5. Facade патерн - Обезбеђује јединствен интерфејс за скуп интерфејса неког подсистема. Facade узор дефинише интерфејс високог нивоа који омогућава да се подсистем лакше користи.
6. Flyweight патерн - Користи дељење да ефикасно подржи велики број објеката.
7. Proxy патерн - Обезбеђује посредника за приступање другом објекту како би се омогућио контролисани приступ до њега.

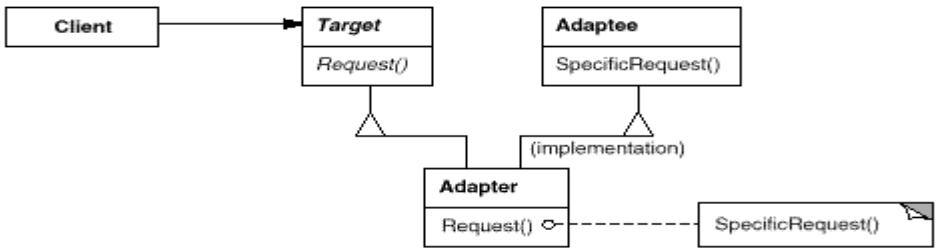
У ниже наведеном тексту даћемо детаље везане за наведене узоре.

СП1: Адаптер узор

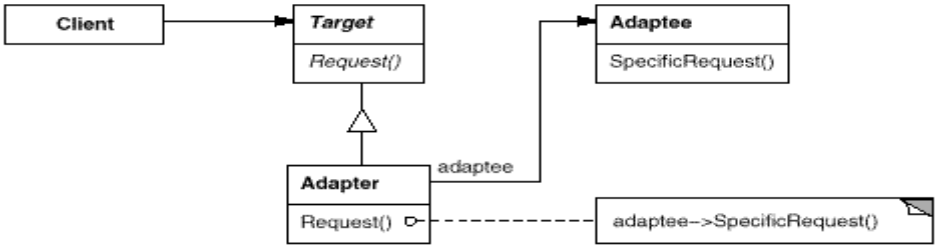
Дефиниција
Адаптер патерн конвертује интерфејс неке класе у други интерфејс који клијент очекује. Другим речима, он прилагођава некомпатибилне интерфејсе.

Појашњење ГОФ дефиниције:
Адаптер патерн конвертује интерфејс неке класе (Adaptee) у други интерфејс који клијент (Client) очекује (Target). Другим речима, он прилагођава некомпатибилне интерфејсе(Adaptee, Target) помоћу класе Adapter.

Наводимо **структуру** adapter узора, која се може јавити у два облика:
а) Класа Адаптер користи вишеструко наслеђивање код прилагођавања некомпатибилних интерфејса.



б) Класа Адаптер користи композицију код прилагођавања некомпатибилних интерфејса.



docsity.com

Учесници:

- ***Target***
  Дефинише доменски <<специфичан интерфејс>> који Client користи.
- ***Client***
  Позива операције преко жељеног интерфејса класе <<Target>>.
- ***Adaptee***
  Дефинише <<постојећи интерфејс>> који треба адаптирати.
- ***Adapter***
  Адаптира интерфејс <<Adaptee-а>> према <<Target>> интерфејсу.

Пример Adapter paterna:

*// **Кориснички захтев PAD1:** Управа Факултета је послала захтев Лабораторији за софтверско инжењерство да промени свој интерфејс, тако што ће имена операција kreirajProgramskiJezik(), kreirajSUBP(), kreirajPonudu() и vratiPonudu() променити у KrProgramskiJezik(),KrSUBP(),KrPonudu() и VrPonudu(). Тимови који праве понуде треба да прилагоде (адаптирају) стари интерфејс (SILAB) новом интерфејсу (SILABTarget), који очекује Управа Факултета без промене, старог интерфејса.*
Прилагођен је пример PBU1 (builder узор).

```
//Улога: Дефинише доменски интерфејс <<SILABTarget>> који Client користи.
interface SILABTarget    // Target
{ void KrProgramskiJezik();
  void KrSUBP();
  void KrPonudu();
  String VrPonudu();
}

//Улога: Адаптира интерфејс <<SILAB>> према <<SILABTarget>> интерфејсу.*/
class Adapter implements SILABTarget   // Adapter
{ SILAB sil;
  Adapter(SILAB sil1) {sil=sil1; }
  public void KrProgramskiJezik(){sil.kreirajProgramskiJezik();}
  public void KrSUBP(){sil.kreirajSUBP();}
  public void KrPonudu() {sil.kreirajPonudu();}
  public String VrPonudu(){return sil.vratiPonudu();}
}

// Улога: Позива операције преко жељеног интерфејса << SILABTarget >>.
class UpravaFakulteta   // Client
{
  SILABTarget silta;

  UpravaFakulteta(SILABTarget silta1){silta = silta1;}

  // Kontroliše konstrukciju <<ponude>> korišcenjem interfejsa SILABTarget.
  void Konstruisi()
    { silta.KrProgramskiJezik();
      silta.KrSUBP();
      silta.KrPonudu();
    }

   public static void main(String args[])
            {  UpravaFakulteta uf;
               SILABTarget silta;
               JavaTimPonuda jat = new JavaTimPonuda();
               silta = new Adapter(jat);
               uf = new UpravaFakulteta(silta);
               uf.Konstruisi();
               System.out.println("Ponuda java tima: " + jat.vratiPonudu());

               VBTimPonuda vbt = new VBTimPonuda();
               silta = new Adapter(vbt);
               uf = new UpravaFakulteta(silta);
               uf.Konstruisi();
               System.out.println("Ponuda VB tima: " + vbt.vratiPonudu());
    }
}
```


docsity.com

// Наведени интерфејси и класе су преузети из примера за Builder узор.
// **********
*// **Улога:** Дефинише интерфејс <<Ponuda>> који треба адаптирати.*
```
interface SILAB // Adaptee
{ void kreirajProgramskiJezik();
  void kreirajSUBP();
  void kreirajPonudu();
  String vratiPonudu();
}
```

*/*__Улоге__: а) Репрезентује сложену <<понуду>> која се конструише.*
*б) Укључује класе (интерфејсе) које дефинишу <<елементе понуде>> */*
```
class PonudaS { ProgramskiJezik pj;  SUBP subp;}
```

/ __Улоге:__ а) Конструише и групише <<елементе понуде>> имплементирајући*
<<Ponuda>>интерфејс.
б) Обезбеђује интерфејс за узимање <<понуде>>./*
```
class JavaTimPonuda implements SILAB // ConcreteBuilder1
{  // Обезбедује интерфејс за узимање <<понуде>>.
  PonudaS elpon; // elementi ponude
  String ponuda;
  JavaTimPonuda() {elpon = new PonudaS();}
  // Конструише и групише делове <<понуде>>.
  public void kreirajProgramskiJezik(){elpon.pj = new Java();}
  public void kreirajSUBP() {elpon.subp = new MySQL();}
  public void kreirajPonudu() { ponuda = "Programski jezik-" + elpon.pj.vratiProgramskiJezik() + "
                                  SUBP-" + elpon.subp.vratiSUBP();}
  public String vratiPonudu(){return ponuda;}
}

class VBTimPonuda implements SILAB // ConcreteBuilder2
{ PonudaS elpon;
  String ponuda;
  VBTimPonuda(){elpon = new PonudaS();}
  public void kreirajProgramskiJezik(){elpon.pj = new VB();}
  public void kreirajSUBP() {elpon.subp = new MSAccess();}
  public void kreirajPonudu() { ponuda = "Programski jezik-" + elpon.pj.vratiProgramskiJezik() + "
                                    SUBP-" + elpon.subp.vratiSUBP();
  public String vratiPonudu(){return ponuda;}
}
```

// Наведени интерфејси и класе су преузети из примера за Abstract Factory узор.
// **********
```
interface ProgramskiJezik
{String vratiProgramskiJezik();}

class Java implements ProgramskiJezik
{ public String vratiProgramskiJezik(){return "Java";}}

class VB implements ProgramskiJezik
{ public String vratiProgramskiJezik(){return "VB";}}
```

// **********

```
interface SUBP // AbstractProductB
{String vratiSUBP();}

class MySQL implements SUBP
{ public String vratiSUBP(){return "MySQL";}}

class MSAccess implements SUBP
{ public String vratiSUBP(){return "MS Access";}}
```

// **********

docsity.com

Коментар примера:

- Разлика између креационих, структурних и узора понашања се огледа у њиховој **одговорности**. Креациони узори су одговорни за процес креирања објеката. Структурни узори су одговорни за рад са композицијом класа или објеката. Узори понашања су одговорни за сарадњу између класа или објеката.
- У овом примеру кренули смо да надограђујемо и мењамо **Builder** узор.
- Напомена: Обично узоре почињемо да примењујемо у току одржавања и надоградње програма. У случају Адаптер патерна постојећи интерфејс (SILAB) се прилагођава новом интерфејсу (SILABTarget).
- Управа факултета је била одговорна код Builder узора за процес прављења понуде. Она је позивала методу:

```
void Konstruisi()
  { sil.kreirajProgramskiJezik();
    sil.kreirajSUBP();
    sil.kreirajPonudu();
  }
```

- Управа Факултета тражи од Лабораторије за софтверско инжењерство да прилагоди интерфејс **SILAB:**

```
interface SILAB // Adaptee
{ void kreirajProgramskiJezik();
  void kreirajSUBP();
  void kreirajPonudu();
  String vratiPonudu();
}
```

sa интерфејсом **SILABTarger**:

```
interface SILABTarget    // Target
{ void KrProgramskiJezik();
  void KrSUBP();
  void KrPonudu();
  String VrPonudu();
}
```

- То се постиже помоћу Адаптера:

```
class Adapter implements SILABTarget   //Adapter
{ SILAB sil;
  Adapter(SILAB sil1) {sil=sil1; }
  public void KrProgramskiJezik(){sil.kreirajProgramskiJezik();}
  public void KrSUBP(){sil.kreirajSUBP();}
  public void KrPonudu() {sil.kreirajPonudu();}
  public String VrPonudu(){return sil.vratiPonudu();}
}
```

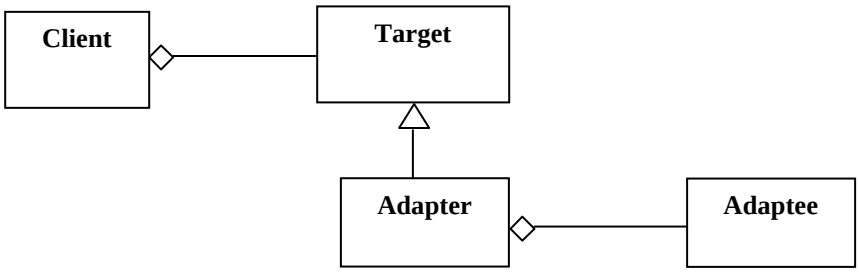
Адаптер прилагођава два различита интерфејса (SILABTarget и SILAB).

- Управа Факултета ће сада позивати *Konstruisi* методу новог интерфејса.

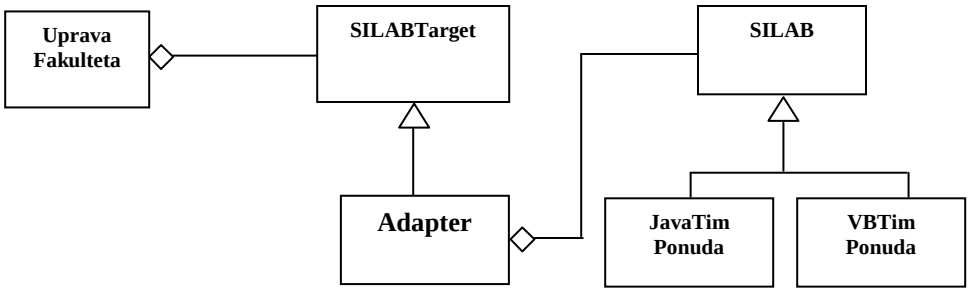
```
void Konstruisi()
   { silta.KrProgramskiJezik();
     silta.KrSUBP();
     silta.KrPonudu();
   }
```


docsity.com

- Адаптер узор има следећу структуру:



- У конкретном случају Адаптер узор има следећу структуру:



- Клијент је поставио захтев да се промени интерфејс. Треба да се постигну два циља помоћу Адаптер узора:
  а) Да се прилагоди постојећи интерфејс клијентовом.
  б) Не треба да мењамо наш постојећи интерфејс и класе које реализују интерфејс ако они могу да обезбеде жељену функционалност.

- Било би веома лоше када би смо на сваки захтев клијента мењали наш интерфејс. Интерфејс има смисла да се мења ако му се додаје нова функционалност.

- Пример Адаптер узора можемо представити у случају односа између:
  а) Наредби Јаве (Target) и Оперативног система (Adaptee) који се прилагођавају помоћу JVM (Adapter). Наредбе Јаве се пресликавају у наредбе конкретног Оперативног система.
  b) Наредби Јаве (Target) и SUBP (Adaptee) који се прилагођавају помоћу драјвера (Adapter). Наредбе Јаве се пресликавају у наредбе конкретног СУБП.

- Анализа програма:

docsity.com

```
class UpravaFakulteta   // Client
{
  SILABTarget silta;

  UpravaFakulteta(SILABTarget silta1){silta = silta1;}

   public static void main(String args[])
              {  UpravaFakulteta uf;
                 SILABTarget silta;
                 JavaTimPonuda jat = new JavaTimPonuda();
                 silta = new Adapter(jat);
                 uf = new UpravaFakulteta(silta);
                 uf.Konstruisi();
                 System.out.println("Ponuda java tima: " +
                  jat.vratiPonudu());
                 ...

          }
```

```
  void Konstruisi()
    {
 silta.KrProgramskiJezik();
       silta.KrSUBP();
       silta.KrPonudu();
    }

 }
```

```
class JavaTimPonuda implements SILAB //
ConcreteBuilder1
{
  PonudaS elpon; // elementi ponude
  String ponuda;
  JavaTimPonuda() {elpon = new PonudaS();}
   public void kreirajProgramskiJezik()
         {elpon.pj = new Java();}
  public void kreirajSUBP()
         {elpon.subp = new MySQL();}
  public void kreirajPonudu()
         { ponuda = "Programski jezik-" +
           elpon.pj.vratiProgramskiJezik() + "
           SUBP-" + elpon.subp.vratiSUBP();}
  public String vratiPonudu(){return ponuda;}
}
```

```
class Adapter implements SILABTarget   // Adapter
{ SILAB sil;
  Adapter(SILAB sil1) {sil=sil1; }
  public void

KrProgramskiJezik(){sil.kreirajProgramskiJezik();}
  public void KrSUBP(){sil.kreirajSUBP();}
  public void KrPonudu() {sil.kreirajPonudu();}
  public String VrPonudu(){return sil.vratiPonudu();}
}
```

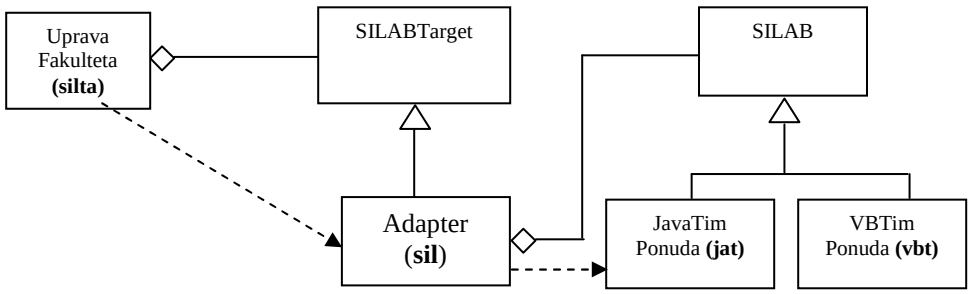
Када се повезују објекти узора то се ради на следећи начин:

a) Прво се креира објекат који од никога не зависе (**jat**).

b) Затим се креира адаптер објекат (**sil**), који се преко конструктора повезује са конкретним објектом који се адаптира (**jat**).

c) Затим се креира клијентски објекат (**uf**), који се преко конструктора повезује са конкретним адаптер објектом (**sil**).

- Креирају се објекти из десне у леву страну.



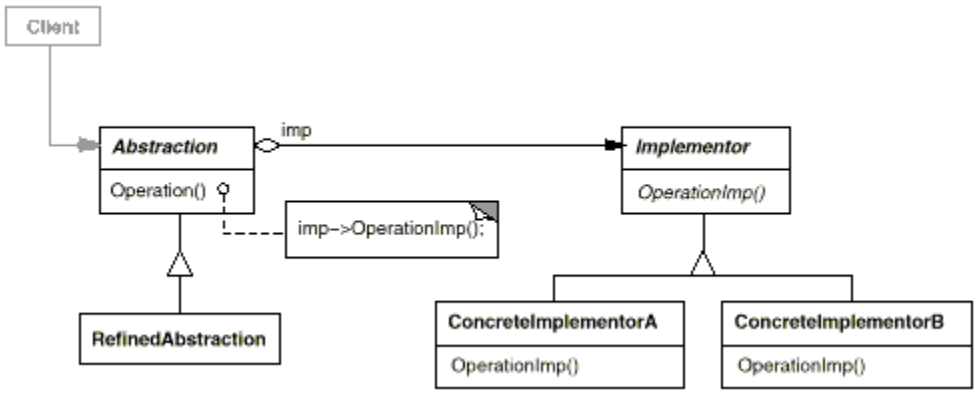
docsity.com

СП2: Bridge узор

Дефиниција
Одваја (декуплује) апстракцију од њене имплементације тако да се оне могу мењати независно.

Појашњење ГОФ дефиниције:
Одваја (декуплује) апстракцију (Abstraction) од њене имплементације (Implementor) тако да се оне могу мењати независно.

*Наводимо **структуру** bridge узора:*



Учесници:

- **Abstraction**

Дефинише интерфејс <<апстракције>>. Чува референцу на објекат типа <<Implementor>>.

- **RefinedAbstraction**

Проширује интерфејс дефинисан класом <<Abstraction>>.

- **Implementor**

Дефинише интерфејс за имплементационе класе (<<ConcreteImplementorA>>, <<ConcreteImplementorB>>). Овај интерфејс не мора да одговара интерфејсу класе <<Abstraction>>. Обично <<Implementor>> интерфејс обезбеђује само примитивне операције а класа <<Abstraction>> дефинише операције високог нивоа које су засноване на наведеним примитивним операцијама.

- **ConcreteImplementor**

Имплементира интерфејс класе <<Implementor >>.

Пример Bridge узора:

Кориснички захтев PBR1: *Управа Факултета је тражила од тимова да промене формат Понуде тако што ће се прво навести СУБП па тек онда програмски језик.*
Java тим у понуди треба да наведе ко су аутори Понуде. Управа Факултета је поставила захтев да и стари формат понуде остану расположиви.

Надограђен је пример за builder узор (PBU1), реализован преко апстрактне класе.

```
class UpravaFakulteta   // Client
{
  SILAB sil;

  UpravaFakulteta(SILAB sil1){sil= sil1;}

  void Konstruisi()
    { sil.kreirajProgramskiJezik();
      sil.kreirajSUBP();
    }
```


docsity.com

```
  public static void main(String args[])
          {  UpravaFakulteta uf;

          FormatPonude fp = null;
            if (args[0].equals("1")) fp = new FormatPonude1();
          if (args[0].equals("2")) fp = new FormatPonude2();

            JavaTimPonuda jat = new JavaTimPonuda(fp);
            uf = new UpravaFakulteta(jat);
            uf.Konstruisi();
          System.out.println("Ponuda java tima: " + jat.vratiPonudu());

            VBTimPonuda vbt = new VBTimPonuda(fp);
          uf = new UpravaFakulteta(vbt);
          uf.Konstruisi();
          System.out.println("Ponuda VB tima: " + vbt.vratiPonudu());
  }
}

class PonudaS { ProgramskiJezik pj;  SUBP subp;}

// Uloga: Definiše interfejs <<SILAB>>. Cuva referencu na objekat tipa <<FormatPonude>>.
abstract class SILAB  // Abstraction
{ PonudaS elpon;
  String ponuda;
  FormatPonude fp;
  SILAB(FormatPonude fp1) { elpon = new PonudaS(); fp = fp1; fp.poveziSaPonudom(elpon);}
  abstract void kreirajProgramskiJezik();
  abstract void kreirajSUBP();
  public String vratiPonudu(){return fp.vratiFormatPonude();}
}


class JavaTimPonuda extends SILAB // RefinedAbstraction1
{
  JavaTimPonuda(FormatPonude fp1) {super(fp1); }
  public void kreirajProgramskiJezik(){elpon.pj = new Java();}
  public void kreirajSUBP() {elpon.subp = new MySQL();}

  public String vratiPonudu(){return "Autor: Lab.za soft. inz. " + fp.vratiFormatPonude();}
}

class VBTimPonuda extends SILAB // RefinedAbstraction2
{
  VBTimPonuda(FormatPonude fp1){super(fp1);}
  public void kreirajProgramskiJezik(){elpon.pj = new VB();}
  public void kreirajSUBP() {elpon.subp = new MSAccess();}
}


// Uloga: Definiše interfejs za implementacione klase (FormatPonude1, FormatPonude2).
abstract class FormatPonude   // Implementor
{  PonudaS pon;
   void poveziSaPonudom(PonudaS pon1){ pon = pon1;}
   abstract String vratiFormatPonude();
}

// Uloga: Implementira interfejs klase <<FormatPonude >>.
class FormatPonude1 extends FormatPonude   // Concrete Implementor A
{
  String vratiFormatPonude()
{  return  "Programski jezik-" + pon.pj.vratiProgramskiJezik() + "  SUBP-" + pon.subp.vratiSUBP();}
}


class FormatPonude2 extends FormatPonude // Concrete Implementor B
{
String vratiFormatPonude()
{ return  "SUBP-" + pon.subp.vratiSUBP() + " Programski jezik-" + pon.pj.vratiProgramskiJezik();}
}
```

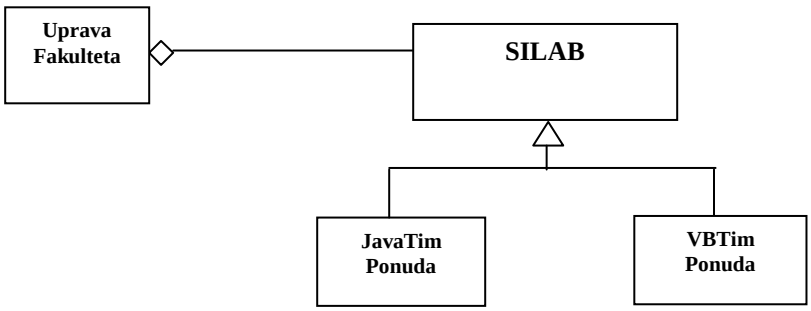

docsity.com

```
// Navedeni interfejsi i klase su preuzeti iz primera za Abstract Factory uzor.
// ************************************************************
interface ProgramskiJezik
{String vratiProgramskiJezik();}

class Java implements ProgramskiJezik
{ public String vratiProgramskiJezik(){return "Java";}}

class VB implements ProgramskiJezik
{ public String vratiProgramskiJezik(){return "VB";}}

// ************************************************************


interface SUBP
{String vratiSUBP();}

class MySQL implements SUBP
{ public String vratiSUBP(){return "MySQL";}}

class MSAccess implements SUBP
{ public String vratiSUBP(){return "MS Access";}}

// ************************************************************
```

Прво се изабере конкретна имплементација (FormatPonude). Затим се имплементација повеже са апстракцијом (SUBP). На крају се клијент (UpravaFakulteta) повезује са апстракцијом (SUBP).
Овај патерн подсећа на top down методу помоћу које се коришћењем концепата апстракције и декомпозиције поједностављује сложеност једног проблема.

Коментар примера:

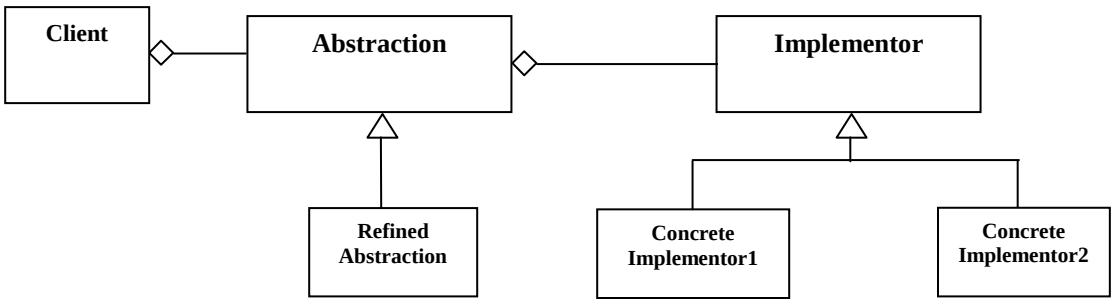
- У овом примеру кренули смо да надограђујемо и мењамо **Builder** узор.



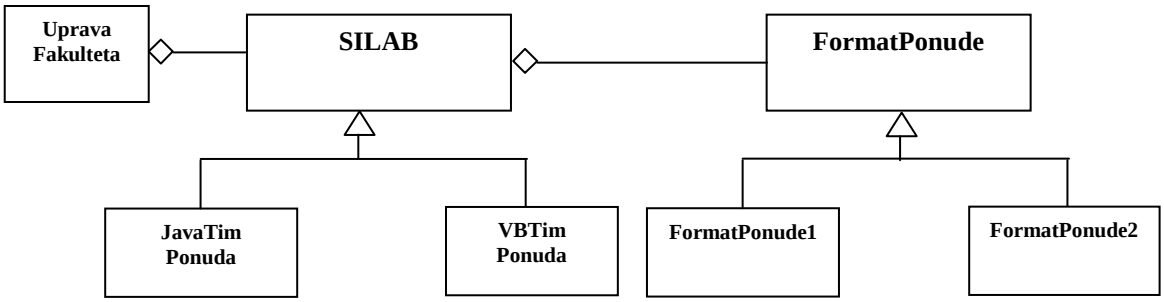
- Поставља се захтев у нашем примеру да се промени:
  а) формат понуде, тако што ће се прво навести СУБП па тек онда програмски језик.
  б) Јава тим треба да наведе ко су аутори понуде.

docsity.com

- Bridge узор има следећу структуру:



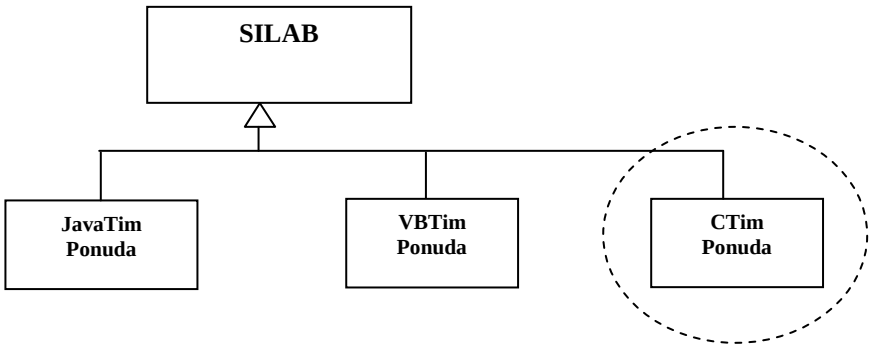
- У конкретном случају Bridge узор има следећу структуру:



- У наведеном случају проблем је био у томе што је сваки тим враћао исти формат понуде. Сада је могуће да тимови буду повезани и да враћају различите формате понуде.

- У класи JavaTimPonuda се обезбеђује захтев да Јава тим у понуди наводи ко су аутори понуде.

- Шта значи да се независно раздвајају апстракција од имплементације у Bridge узору? Уколико уведемо нову класу CTimPonuda (RefinedAbstraction), код Bridge узора, неће се морати мењати ништа што се односи на формат понуде(Implementor).



docsity.com

- Формати понуда се не мењају у свом генералном облику:
  а) **"Programski jezik-" + pon.pj.vratiProgramskiJezik() + " SUBP-" + pon.subp.vratiSUBP()**

  б)**"SUBP-" + pon.subp.vratiSUBP() + " Programski jezik-" + pon.pj.vratiProgramskiJezik();**

Формат понуде има променљиве и непроменљиве делове. На променљиве делове утичу тимови са својим специјализованим понудама.

- Преко командне линије се бира формат понуде. Управа Факултета поставља захтев у ком ће формату бити понуда:
  if (args[0].equals("1")) fp = new FormatPonude1();
  if (args[0].equals("2")) fp = new FormatPonude2();

- Важно правило: **У односу између клијента и сервера прво се креира сервер па тек онда клијент.**

```
class Klijent
{ Server s;
 Klijent(Server s1){s=s1;}
}

 class Server
{…}
 ...

public static void main(String args[])
    { Server s = new Server();
      Klijent k = new Klijent(s);
      …
   }
```

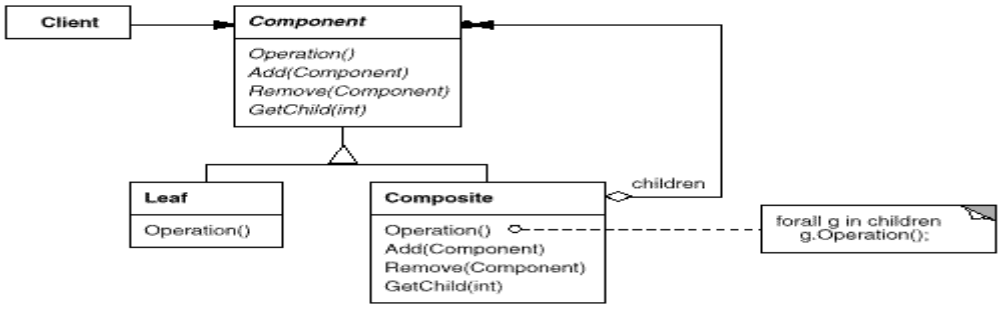

docsity.com

СП3. Composite узор

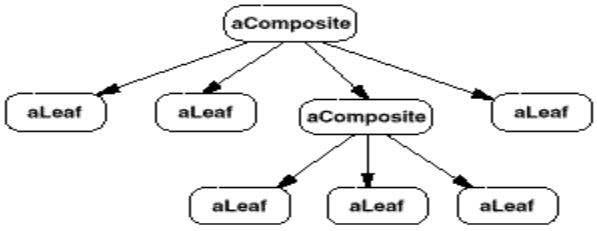
Дефиниција
Објекти се састављају (компонују) у структуру стабла како би представили хијерархију целине и делова. Composite узор омогућава да се једноставни и сложени (компоновани) објекти третирају јединствено.

Појашњење ГОФ дефиниције:
Објекти се састављају (компонују) у структуру стабла како би представили хијерархију целине и делова. Composite узор омогућава да се једноставни (Leaf) и сложени (Composite) објекти третирају јединствено. Једноставни и сложени објекти су компоненте (Component).

*Наводимо **структуру** Composite узора:*



Типична структура Composite објекта има следећи изглед:



*Као и **учеснике** у узору:*

- **Component**
  Декларише интерфејс за <<објекте >> који ће да образују састав (композицију).
  Декларише интерфејс за приступање и управљање његовим <<деца-компонентама>>.
- **Leaf**
  Представља <<просте објекте (листове)>> у саставу (композицији) и дефинише њихово понашање.
  Лист нема деца-компоненте.
- **Composite**
  Дефинише понашање за <<компоненте>> које имају децу.
  Чува <<децу-компоненте>>.
- **Client**
  Манипулише <<објектима у саставу (композицији) >> кроз <<Component >> интерфејс.

Пример Composite узора:

docsity.com

Кориснички захтев ПБР1: *Управа Факултета је тражила да тим који је у понуди навео Јаву као програмски језик детаљније образложи неке од најважнијих Јава технологија које ће се користити за израду софтверског система ПДС-а на ФОН-у, ту се посебно мисли на J2EE технологију.*

Надограђен је пример за bridge узор (PBR1).

```
class UpravaFakulteta   // Client
{
  SILAB sil;

  UpravaFakulteta(SILAB sil1){sil= sil1;}

  void Konstruisi()
    { sil.kreirajProgramskiJezik();
      sil.kreirajSUBP();
    }

   public static void main(String args[])
            {  UpravaFakulteta uf;

                FormatPonude fp = null;
                if (args[0].equals("1")) fp = new FormatPonude1();
                if (args[0].equals("2")) fp = new FormatPonude2();

               JavaTimPonuda jat = new JavaTimPonuda(fp);
               uf = new UpravaFakulteta(jat);
               uf.Konstruisi();
               System.out.println("Ponuda java tima: " + jat.vratiPonudu());

               VBTimPonuda vbt = new VBTimPonuda(fp);
                uf = new UpravaFakulteta(vbt);
                uf.Konstruisi();
                System.out.println("Ponuda VB tima: " + vbt.vratiPonudu());
    }
}


class PonudaS { ProgramskiJezik pj;  SUBP subp;}

// Uloga: Definiše interfejs <<SILAB>>. Cuva referencu na objekat tipa <<FormatPonude>>.
abstract class SILAB  // Abstraction
{ PonudaS elpon;
  String ponuda;
  FormatPonude fp;
  SILAB(FormatPonude fp1) { elpon = new PonudaS(); fp = fp1; fp.poveziSaPonudom(elpon);}
  abstract void kreirajProgramskiJezik();
  abstract void kreirajSUBP();
  public String vratiPonudu(){return fp.vratiFormatPonude();}
}


class JavaTimPonuda extends SILAB // RefinedAbstraction1
{
  JavaTimPonuda(FormatPonude fp1) {super(fp1); }
  public void kreirajProgramskiJezik(){elpon.pj = new Java();}
  public void kreirajSUBP() {elpon.subp = new MySQL();}

  public String vratiPonudu(){return "Autor: Lab.za soft. inz. " + fp.vratiFormatPonude();}
}

class VBTimPonuda extends SILAB // RefinedAbstraction2
{
  VBTimPonuda(FormatPonude fp1){super(fp1);}
  public void kreirajProgramskiJezik(){elpon.pj = new VB();}
  public void kreirajSUBP() {elpon.subp = new MSAccess();}
}
```


docsity.com

```
// Uloga: Definiše interfejs za implementacione klase (FormatPonude1, FormatPonude2).
abstract class FormatPonude   // Implementor
{  PonudaS pon;
   void poveziSaPonudom(PonudaS pon1){ pon = pon1;}
   abstract String vratiFormatPonude();
}

// Uloga: Implementira interfejs klase <<FormatPonude >>.
class FormatPonude1 extends FormatPonude   // Concrete Implementor A
{
  String vratiFormatPonude()
{  return  "Programski jezik-" + pon.pj.vratiProgramskiJezik() + "  SUBP-" + pon.subp.vratiSUBP();}
}


class FormatPonude2 extends FormatPonude // Concrete Implementor B
{
String vratiFormatPonude()
{ return  "SUBP-" + pon.subp.vratiSUBP() + " Programski jezik-" + pon.pj.vratiProgramskiJezik();}
}
```

// Navedeni interfejsi i klase su preuzeti iz primera za Abstract Factory uzor.

```
// ***************************************************************
interface ProgramskiJezik
{String vratiProgramskiJezik();}
```

/* **Uloga:** Manipuliše objektima u sastavu (kompoziciji) <<Java,J2SE,J2EE,...,EntityBean i SessionBean>> kroz <<KComponent >> interfejs.*/

```
class Java implements ProgramskiJezik // Client
{ public String vratiProgramskiJezik(){
            EntityBean eb = new EntityBean();
            SessionBean sb = new SessionBean();
            EJB ejb = new EJB();
            ejb.povezi(eb);
            ejb.povezi(sb);
            Servlet sr = new Servlet();
            JSP jsp = new JSP();
            J2EE j2ee = new J2EE();
            j2ee.povezi(ejb);
            j2ee.povezi(sr);
            j2ee.povezi(jsp);
            J2SE j2se = new J2SE();
            JavaPlatforma j = new JavaPlatforma();
            j.povezi(j2se);
            j.povezi(j2ee);
         return j.vratiTehnologiju(); }
}


class VB implements ProgramskiJezik
{ public String vratiProgramskiJezik(){return "VB";}}

// ***************************************************************
```

/* **Uloge:**
a) Deklariše interfejs za <<Java,J2SE,J2EE,...,EntityBean i SessionBean >> komponente koje ce da obrazuju sastav (kompoziciju).
b) Deklariše operaciju VratiTehnologiju() za pristupanje i upravljanje deca komponentama <<J2SE,J2EE,JSP,...,EntityBean i SessionBean>>.*/

```
abstract class KComponent    // Component
{ static int nivo =0;   abstract String vratiTehnologiju();}
```


docsity.com

```
/* Uloge:
   a) Definiše ponašanje za komponente <<Java,J2EE i EJB>> koje imaju decu.
   b) Cuva decu-komponente <<J2SE,J2EE,JSP,Servlet,EJB,EntityBean i SessionBean>>.
*/
class Composite extends KComponent   // Composite
{   KComponent com[];
    int brojKomponenti;
     Composite() { com = new KComponent[5]; brojKomponenti = 0; }
    String vratiTehnologiju()
      {  String teh = " se sastoji od:";
         nivo ++;
         for(int i=0;i<brojKomponenti;i++)
            {  teh = teh + "\n" + vratiPomeraj(nivo) + (i+1) + ":" + com[i].vratiTehnologiju(); }
         nivo --;
         return teh;
      }
    void povezi(KComponent com1)
     { if (brojKomponenti < 5 )
               { com[brojKomponenti] = com1;
                 brojKomponenti++;
               }
       else
               { System.out.println("Ne moze se povezati element sa kompozicijom. Maksimalni broj
                                     elemenata kompozicije je 5");
               }
      }

 String vratiPomeraj (int nivo)
    { String pomeraj = "";
      for(int i=0;i<nivo*3;i++) {pomeraj = pomeraj + "-";}
               return pomeraj;
    }
}
```

/* **Uloge** (Leaf):
a) Predstavlja proste objekte (listove)<< J2SE, JSP,Servlete,EntityBean i SessionBean>> u sastavu (kompoziciji) i
definiše njihovo ponašanje.
b) List nema deca-komponente.
*/

```
class EntityBean extends KComponent   // Leaf
{ String vratiTehnologiju(){return "EntityBean";}}

class SessionBean extends KComponent  // Leaf
{ String vratiTehnologiju(){return "SessionBean";}}

/* Uloge (Composite): Manipuliše objektima u sastavu (kompoziciji)
<<J2SE i J2EE su u sastavu Java; JSP,Servlet i EJB su u sastavu J2EE;
EntityBean i SessionBean su u sastavu EJB>>
kroz  <<KComponent >>  interfejs.*/

class EJB extends Composite // Composite
{ String vratiTehnologiju(){ return "EJB" + super.vratiTehnologiju();}}

class Servlet extends KComponent // Leaf
{ String vratiTehnologiju(){return "Servlet";}}

class JSP extends KComponent  // Leaf
{ String vratiTehnologiju(){return "JSP";}}

class J2EE extends Composite // Composite
{ String vratiTehnologiju(){ return "J2EE" + super.vratiTehnologiju();}}

class J2SE extends KComponent // Leaf
{ String vratiTehnologiju(){return "J2SE";}}
```


docsity.com

```
class JavaPlatforma extends Composite // Composite
{ String vratiTehnologiju(){ return "Java\nJava" + super.vratiTehnologiju();}}

//*****************************************************************

interface SUBP
{String vratiSUBP();}

class MySQL implements SUBP
{ public String vratiSUBP(){return "MySQL";}}

class MSAccess implements SUBP
{ public String vratiSUBP(){return "MS Access";}}

// ***************************************************************
```

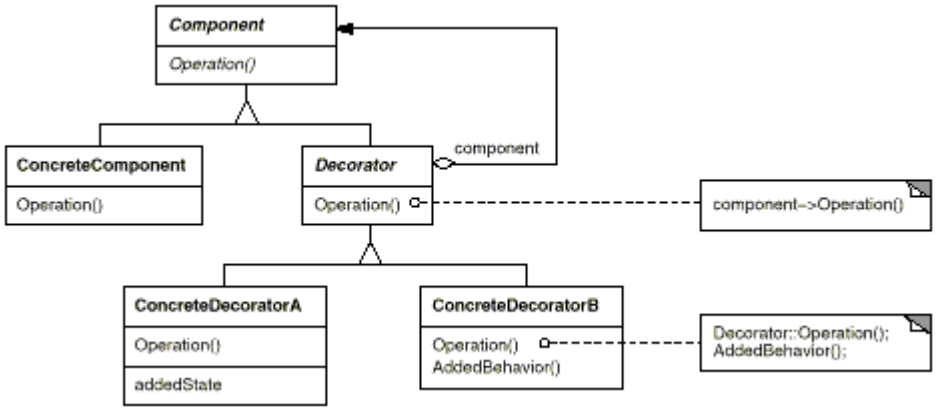

docsity.com

СП4: Decorator узор

Дефиниција
Придружује одговорност до објекта динамички. Декоратор проширује функционалност објекта динамичким додавањем функционалности других објеката.

Појашњење ГОФ дефиниције:
Придружује одговорност до објекта динамички. Декоратор проширује функционалност објекта (ConcreteComponent) динамичким додавањем функционалности других објеката (ConcreteDecoratorA, ConcreteDecoratorB).

*Наводимо **структуру** Decorator узора:*



Учесници:

- **Component**

Дефинише интерфејс за објекте којима се одговорност додаје динамички.

- **ConcreteComponent**

Дефинише << конкретан објекат>> коме ће бити додата одговорност динамички.

- **Decorator**

Чува референцу на << компоненту>> и позива <<операцију>> компоненти.

- **ConcreteDecorator**

Додаје одговорност до компоненте.

Пример Decorator узора:

***Корисницки захтев ПБР1**: Управа Факултета треба да прошири понуду Јава тима са датумом када је издата понуда. Након тога понуду треба просирити са местом где је издата понуда.*

Надограђен је пример за Builder узор (PBU1), реализован преко апстрактне класе.

```
/* Uloga: Definiše interfejs za objekte kojima se odgovornost dodaje dinamički.*/
interface Komponent   // Component
{ void prikaziPonudu();}

/* Uloga: : Čuva referencu na  <<komponentu>> i poziva operaciju <<PrikaziPonudu()>>
komponente.*/
class Decorator implements Komponent // Decorator
{   Komponent komp;
```


docsity.com

```
   Decorator(Komponent komp1) {komp = komp1;}
   public void prikaziPonudu(){komp.prikaziPonudu();}
}

/* Uloga: Dodaje odgovornost do komponente. */
class Datum extends Decorator  // Concrete Decorator 1
{   Datum(Komponent komp1) {super(komp1);}
            public void prikaziPonudu(){super.prikaziPonudu(); System.out.println("Datum:");}
}

class Mesto extends Decorator  // Concrete Decorator 2
{   Mesto(Komponent komp1) {super(komp1);}
            public void prikaziPonudu(){super.prikaziPonudu(); System.out.println("Mesto:");}
}

/* Uloga: Definiše  <<upravu fakulteta>> kojoj ce biti dodata odgovornost dinamički. */
class UpravaFakulteta implements Komponent  // ConcreteComponent
{
  SILAB sil;
  UpravaFakulteta(SILAB sil1){sil = sil1;}

    void Konstruisi()
    { sil.kreirajProgramskiJezik();
      sil.kreirajSUBP();
    }

   public static void main(String args[])
            {  UpravaFakulteta uf;
               FormatPonude fp = null;
               if (args[0].equals("1")) fp = new FormatPonude1();
             if (args[0].equals("2")) fp = new FormatPonude2();
               JavaTimPonuda jat = new JavaTimPonuda(fp);
               uf = new UpravaFakulteta(jat);
               uf.Konstruisi();

               Datum dat = new Datum(uf);
               Mesto mes = new Mesto(dat);

              // Moglo je i ovako da se napise:
            // Mesto mes = new Mesto(new Datum(uf));

               mes.prikaziPonudu();

              // u ovom primeru se prvo pojavljuje mesto pa datum na ponudi
              // Mesto mes = new Mesto(uf);
              // Datum dat = new Datum(mes);
              // dat.prikaziPonudu();

   }

 public void prikaziPonudu(){System.out.println("Ponuda java tima: " + sil.vratiPonudu());}
}

class PonudaS { ProgramskiJezik pj;  SUBP subp;}

// Uloga: Definiše interfejs <<SILAB>>. Cuva referencu na objekat tipa <<FormatPonude>>.
abstract class SILAB  // Abstraction
{ PonudaS elpon;
  String ponuda;
  FormatPonude fp;
  SILAB(FormatPonude fp1) { elpon = new PonudaS(); fp = fp1; fp.poveziSaPonudom(elpon);}
  abstract void kreirajProgramskiJezik();
  abstract void kreirajSUBP();
  public String vratiPonudu(){return fp.vratiFormatPonude();}
}
```


docsity.com

```
class JavaTimPonuda extends SILAB // RefinedAbstraction1
{
  JavaTimPonuda(FormatPonude fp1) {super(fp1); }
  public void kreirajProgramskiJezik(){elpon.pj = new Java();}
  public void kreirajSUBP() {elpon.subp = new MySQL();}

  public String vratiPonudu(){return "Autor: Lab.za soft. inz. " + fp.vratiFormatPonude();}
}

class VBTimPonuda extends SILAB // RefinedAbstraction2
{
  VBTimPonuda(FormatPonude fp1){super(fp1);}
  public void kreirajProgramskiJezik(){elpon.pj = new VB();}
  public void kreirajSUBP() {elpon.subp = new MSAccess();}
}


// Uloga: Definiše interfejs za implementacione klase (FormatPonude1, FormatPonude2).
abstract class FormatPonude   // Implementor
{  PonudaS pon;
   void poveziSaPonudom(PonudaS pon1){ pon = pon1;}
   abstract String vratiFormatPonude();
}

// Uloga: Implementira interfejs klase <<FormatPonude >>.
class FormatPonude1 extends FormatPonude   // Concrete Implementor A
{
  String vratiFormatPonude()
{  return  "Programski jezik-" + pon.pj.vratiProgramskiJezik() + "  SUBP-" + pon.subp.vratiSUBP();}
}


class FormatPonude2 extends FormatPonude // Concrete Implementor B
{
String vratiFormatPonude()
{ return  "SUBP-" + pon.subp.vratiSUBP() + " Programski jezik-" + pon.pj.vratiProgramskiJezik();}
}

// Navedeni interfejsi i klase su preuzeti iz primera za Abstract Factory uzor.
// ************************************************************
interface ProgramskiJezik
{String vratiProgramskiJezik();}

class Java implements ProgramskiJezik
{ public String vratiProgramskiJezik(){return "Java";}}

class VB implements ProgramskiJezik
{ public String vratiProgramskiJezik(){return "VB";}}

// ************************************************************

interface SUBP
{String vratiSUBP();}

class MySQL implements SUBP
{ public String vratiSUBP(){return "MySQL";}}

class MSAccess implements SUBP
{ public String vratiSUBP(){return "MS Access";}}

// ************************************************************
```

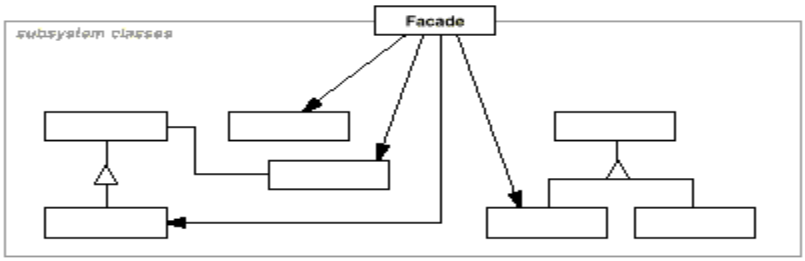

docsity.com

СП5: Facade узор

Дефиниција
Обезбеђује јединствен интерфејс за скуп интерфејса неког подсистема. Facade узор дефинише интерфејс високог нивоа који омогућава да се подсистем лакше користи.

Појашњење ГОФ дефиниције:
Обезбеђује јединствен интерфејс (Facade) за скуп интерфејса (Sybsystem classes) неког подсистема. Facade узор дефинише интерфејс високог нивоа који омогућава да се подсистем лакше користи.

*Наводимо **структуру** Facade узора:*



*Као и **учеснике** у узору:*

- **Facade**
  Зна које <<подсистемске класе>> су одговорне за сваки од захтева који му се прослеђује.
  Преноси одговорност за извршење клијентских захтева до одговарајућих <<подсистемских објеката >>.
- **Sybsystem classes**
  Имплементирају <<подсистемске функционалности>>.
  Не знају ко је facade класа.

Пример Facade узора:

Корисницки захтев PFA1: *Управа Факултета треба да управља процесом прављења понуде тако сто ће на високом нивоу да издаје задатке Комисији за понуде која треба оперативно да реализује сваки од постављених задатака.*

Коришћен је пример за bridge узор (PBR1).

```
class UpravaFakulteta // client
{
  static KomisijaZaPonude kzp;
  UpravaFakulteta(){kzp = new KomisijaZaPonude();}

  public static void main(String args[])
            {  UpravaFakulteta uf = new UpravaFakulteta(); // a
               kzp.odrediFormatPonude(args[0]); // b
               kzp.kreirajPonuduJavaTima(); // c
               kzp.Konstruisi(); // d
               kzp.prikaziPonudu(); // e
    }
}
```

/ **Uloge**:*
a) Zna koje podsistemske klase <<FormatPonude, JavaTimPonuda>> su odgovorne za svaki od zahteva koji mu se prosleduje.

docsity.com

b) Prenosi odgovornost za izvršenje klijentskih zahteva do odgovarajucih podsistemskih objekata <<fp,pon>>.
*/

```
class KomisijaZaPonude  // Facade
{   FormatPonude fp;
    SILAB sil;
    void odrediFormatPonude(String arg)
                { if (arg.equals("1")) fp = new FormatPonude1();

        if (arg.equals("2")) fp = new FormatPonude2();
      }
    void kreirajPonuduJavaTima(){sil = new JavaTimPonuda(fp);}
    void Konstruisi(){
              sil.kreirajProgramskiJezik();  // d1
              sil.kreirajSUBP(); // d2
}
    void prikaziPonudu(){System.out.println("Ponuda java tima: " + sil.vratiPonudu());} // e1
}
```

/* ***Uloga:*** *Implementira podsistemsku funkcionalnost, kreira format ponude <<FormatPonude1,FormatPonude2>>.* */

```
abstract class FormatPonude   // sybsystem class
{  PonudaS pon;
   void poveziSaPonudom(PonudaS pon1){ pon = pon1;}
   abstract String vratiFormatPonude();
}


class FormatPonude1 extends FormatPonude
{
String vratiFormatPonude() { return "Programski jezik-" + pon.pj.vratiProgramskiJezik() + "  SUBP-" +
pon.subp.vratiSUBP();}
}

class FormatPonude2 extends FormatPonude
{
String vratiFormatPonude() { return "SUBP-" + pon.subp.vratiSUBP() + " Programski jezik-" +
pon.pj.vratiProgramskiJezik();}
}



class PonudaS { ProgramskiJezik pj;  SUBP subp;}
```

/* **Uloga:** Implementira podsistemsku funkcionalnost <<vratiPonudu()>>.*/

```
abstract class SILAB  // sybsystem class
{ PonudaS pon;
  FormatPonude fp;
  SILAB(FormatPonude fp1) {
                pon = new PonudaS();  // c1
                fp = fp1;  // c2
                fp.poveziSaPonudom(pon); // c3
}
  abstract void kreirajProgramskiJezik();
  abstract void kreirajSUBP();
  public String vratiPonudu(){return fp.vratiFormatPonude();}
}


class JavaTimPonuda extends SILAB
{ JavaTimPonuda(FormatPonude fp1) {super(fp1);}
  public void kreirajProgramskiJezik(){pon.pj = new Java();}
  public void kreirajSUBP() {pon.subp = new MySQL();}
  public String vratiPonudu(){return "Autor: Lab.za soft. inz. " + fp.vratiFormatPonude();}
}
```


docsity.com

```
class VBTimPonuda extends SILAB
{ VBTimPonuda(FormatPonude fp1) {super(fp1);}
  public void kreirajProgramskiJezik(){pon.pj = new VB();}
  public void kreirajSUBP() {pon.subp = new MSAccess();}
}


// ************************************************************
interface ProgramskiJezik
{String vratiProgramskiJezik();}

class Java implements ProgramskiJezik
{ public String vratiProgramskiJezik(){return "Java";}}

class VB implements ProgramskiJezik
{ public String vratiProgramskiJezik(){return "VB";}}

// ************************************************************

interface SUBP
{String vratiSUBP();}

class MySQL implements SUBP
{ public String vratiSUBP(){return "MySQL";}}

class MSAccess implements SUBP
{ public String vratiSUBP(){return "MS Access";}}

// ************************************************************
```


docsity.com

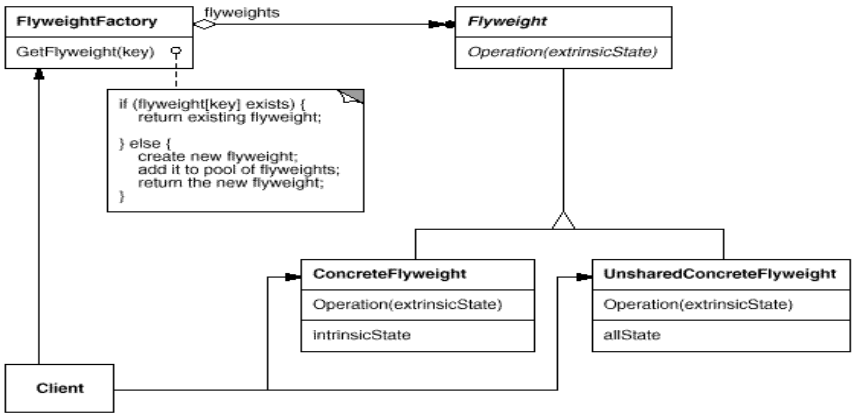
СП6: Flyweight узор

Дефиниција
Користи дељење да ефикасно подржи велики број објеката.

Појашњење ГОФ дефиниције:

Користи дељење (ConcreteFlyweight) да ефикасно подржи велики број објеката.

*Наводимо **структуру** Flyweight узора:*



*Као и **учеснике** у узору:*

- **Flyweight –** декларише интерфејс преко кога <<flyweight>> може да прихвати и да делује на спољашње стање.
- **ConcreteFlyweight** – имплементира <<flyweight>> интерфејс и додаје простор за унутрашње стање ако је то потребно. <<ConcreteFlyweight>> објекат мора бити дељив. Стање које се чува у <<ConcreteFlyweight-у>> мора бити унутрашње (не сме зависити од контекста у коме се налази ConcreteFlyweight).
- **UnsharedConcreteFlyweight** – Поред дељивих <<Flyweight>> подкласе постоје и Flyweight подкласе које нису дељиве. Заједничко за недељиве <<Flyweight>> подкласе је да имају <<ConcreteFlyweight>> објекте као децу у flyweight објектној структури.
- **FlyweightFactory** – креира и управља <,flyweight>> објектима. Он омогућава да <<flyweight>> објекти имају особину дељивости. Када клијент захтева <<flyweight>>, <<FlyweightFactory>> објекат враћа постојеће појављивање или креира ново ако исто не постоји.
- **Client** – садржи референце на <<flyweight-ове>>. Израчунава или чува спољашња стања <<flyweight-ова>>.

Пример Flyweight узора:

Кориснички захтев PFW1: *Управа Факултета је тражила од Јава тима да припреми своју понуду у следеца три различита облика:*
а) Понуда Јава тима.
Аутор: Лаб. за софтверско инжењерство.
Програмски језик: Java.
СУБП: MySQL.

б) Понуда Јава тима. Аутор: Лаб. за софтверско инзењерство.
Програмски језик: Java. СУБП: MySQL.

docsity.com

ц) Понуда Јава тима.
Програмски језик: Јава.
СУБП: MySQL.

Аутор: Лаб. за софтверско инзењерство.

Шеф Лабораторије за софтверско инжењерство је тражио од Јава тима да елементе понуде чува на једном месту, како се не би десила редуданса истих елемената понуде у различитим захтеваним облицима понуде.

```
class UpravaFakulteta
{ static Ponuda pon;
  public static void main(String args[])
            {  pon = new JavaTimPonuda();
               pon.kreirajProgramskiJezik();
               pon.kreirajSUBP();
               pon.dodajElementePonude();
               pon.dodajOblikPonude1();
               pon.prikaziPonudu();
               pon.dodajOblikPonude2();
               pon.prikaziPonudu();
               pon.dodajOblikPonude3();
               pon.prikaziPonudu();
            }
}

class PonudaS { ProgramskiJezik pj;  SUBP subp;}

abstract class Ponuda
{ PonudaS pon;
  abstract void kreirajProgramskiJezik();
  abstract void kreirajSUBP();
  abstract void dodajElementePonude();
  abstract void dodajOblikPonude1();
  abstract void dodajOblikPonude2();
  abstract void dodajOblikPonude3();
  abstract void prikaziPonudu();
}
```

/* **Uloga:** Sadrzi reference na <<flyweight-ove>>. Izracunava ili cuva spoljašnja stanja <<flyweight-ova>>.*/

```
class JavaTimPonuda extends Ponuda  // Client
{ FlyweightFactory ff;
  UnsharedConcreteFlyweight ucf[];
  FlyWeight cf;

  JavaTimPonuda(){ff = new FlyweightFactory(); pon = new PonudaS();
                  ucf=new UnsharedConcreteFlyweight[4];}
  void kreirajProgramskiJezik(){pon.pj = new Java();}
  void kreirajSUBP() {pon.subp = new MySQL();}
  void dodajElementePonude(){
              ff.dodajFlyWeight("Ponuda Java tima.");
              ff.dodajFlyWeight("Autor: Lab. za softversko inzenjerstvo.");
            ff.dodajFlyWeight("Programski jezik: " + pon.pj.vratiProgramskiJezik() + ".");
            ff.dodajFlyWeight("SUBP:" + pon.subp.vratiSUBP() + ".");
              }
  void dodajOblikPonude1()
    {       cf = ff.dodajFlyWeight("Ponuda Java tima.");
            ucf[0] = new UnsharedConcreteFlyweight(cf,"");

              cf = ff.dodajFlyWeight("Autor: Lab. za softversko inzenjerstvo.");
              ucf[1] = new UnsharedConcreteFlyweight(cf,"\n");

              cf = ff.dodajFlyWeight("Programski jezik: " + pon.pj.vratiProgramskiJezik() + ".");
              ucf[2] = new UnsharedConcreteFlyweight(cf,"\n");
```


docsity.com

```
            cf = ff.dodajFlyWeight("SUBP:" + pon.subp.vratiSUBP() + ".");
            ucf[3] = new UnsharedConcreteFlyweight(cf,"\n");

   }

   void dodajOblikPonude2()
    {       cf = ff.dodajFlyWeight("Ponuda Java tima.");
            ucf[0] = new UnsharedConcreteFlyweight(cf,"");

              cf = ff.dodajFlyWeight("Autor: Lab. za softversko inzenjerstvo.");
              ucf[1] = new UnsharedConcreteFlyweight(cf,"\t");

              cf = ff.dodajFlyWeight("Programski jezik: " + pon.pj.vratiProgramskiJezik() + ".");
              ucf[2] = new UnsharedConcreteFlyweight(cf,"\n");

            cf = ff.dodajFlyWeight("SUBP:" + pon.subp.vratiSUBP() + ".");
            ucf[3] = new UnsharedConcreteFlyweight(cf,"");
   }

void dodajOblikPonude3()
    {       cf = ff.dodajFlyWeight("Ponuda Java tima.");
            ucf[0] = new UnsharedConcreteFlyweight(cf,"");

              cf = ff.dodajFlyWeight("Programski jezik: " + pon.pj.vratiProgramskiJezik() + ".");
              ucf[1] = new UnsharedConcreteFlyweight(cf,"\n\t\t");

            cf = ff.dodajFlyWeight("SUBP:" + pon.subp.vratiSUBP() + ".");
            ucf[2] = new UnsharedConcreteFlyweight(cf,"\n\t\t");

           cf = ff.dodajFlyWeight("Autor: Lab. za softversko inzenjerstvo.");
             ucf[3] = new UnsharedConcreteFlyweight(cf,"\n");
    }

void prikaziPonudu()
  { String ponuda = "";

    for(int i=0;i<4;i++)  { ponuda = ponuda + ucf[i].vratiStanje(); }
    System.out.println(ponuda);
    System.out.println("*******************************");
  }
 }


// ***************************************************************
interface ProgramskiJezik
{String vratiProgramskiJezik();}

class Java implements ProgramskiJezik
{ public String vratiProgramskiJezik(){return "Java";}}

// ***************************************************************

interface SUBP
{String vratiSUBP();}

class MySQL implements SUBP
{ public String vratiSUBP(){return "MySQL";}}

// ***************************************************************
```

/* Uloga: Kreira i upravlja <flyweight>> objektima. On omogucava da <<flyweight>> objekti imaju osobinu deljivosti. Kada klijent zahteva <<flyweight>>, <<FlyweightFactory>> objekat vraca postojece pojavljivanje ili kreira novo ako isto ne postoji.*/

```
class FlyweightFactory
{
            FlyWeight ff[];
            int brojFlyweights;
```


docsity.com

```
    FlyweightFactory(){ff = new FlyWeight[4];brojFlyweights=0;}
  FlyWeight dodajFlyWeight(String flyweight)
    {  for(int i = 0; i<brojFlyweights;i++)
          { if (ff[i].vratiStanje().equals(flyweight))
                { return ff[i]; }
          }

       ff[brojFlyweights++] = new ConcreteFlyweight(flyweight);
       return ff[brojFlyweights-1];
     }
}
```

/* Uloga: Deklariše interfejs preko koga <<flyweight>> može da prihvati i
da deluje na spoljašnje stanje.*/

```
abstract class FlyWeight
{
   abstract String vratiStanje();
   String vratiStanje(String spoljasnjeStanje){return "";}
}
```

/* Uloga: Implementira <<flyweight>> interfejs i dodaje prostor za
unutrašnje stanje ako je to potrebno. <<ConcreteFlyweight>> objekat
mora biti deljiv. Stanje koje se cuva u <<ConcreteFlyweight-u>> mora
biti unutrašnje (ne sme zavisiti od konteksta u kome se nalazi ConcreteFlyweight). */

```
class ConcreteFlyweight extends FlyWeight
{
  String unutrasnjeStanje;

  ConcreteFlyweight(String unutrasnjeStanje1){unutrasnjeStanje = unutrasnjeStanje1;}
  public String vratiStanje(String spoljasnjeStanje){
                          //System.out.print(spoljasnjeStanje + " " + unutrasnjeStanje);
                          return spoljasnjeStanje + " " + unutrasnjeStanje;}
  public String vratiStanje(){return unutrasnjeStanje;}
}
```

/* Uloga: Pored deljivih <<Flyweight>> podklase postoje i Flyweight podklase
koje nisu deljive. Zajednicko za nedeljive <<Flyweight>> podklase je da
imaju <<ConcreteFlyweight>> objekte kao decu u flyweight objektnoj strukturi.*/

```
class UnsharedConcreteFlyweight  extends FlyWeight
{ String spoljasnjeStanje;
  FlyWeight cf;
  UnsharedConcreteFlyweight(FlyWeight cf1,String spoljasnjeStanje1)
        {cf = cf1; spoljasnjeStanje = spoljasnjeStanje1;}
  public String vratiStanje(){return cf.vratiStanje(spoljasnjeStanje);}
}
```

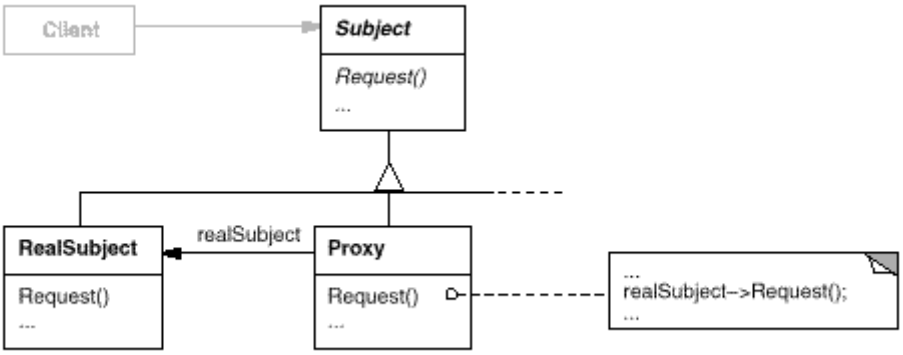

docsity.com

СП7: Proxy узор

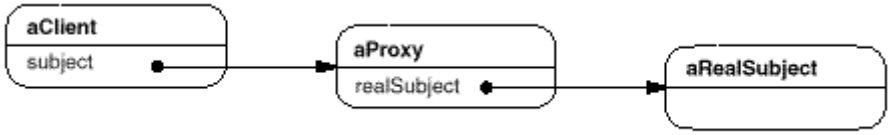
Дефиниција
Обезбеђује посредника за приступање другом објекту како би се омогућио контролисани приступ до њега.

Појашњење ГОФ дефиниције:
Обезбеђује посредника (Proxy) за приступање другом објекту **(**RealSubject**)** како би се омогућио контролисани приступ до њега.

Наводимо **структуру** Proxy узора:



Приказујемо могући објектни дијаграм proxy структуре у реалном времену.



Као и** учеснике **у узору:

- **Proxy**
  - Садржи референце које омогућавају <<Proxy>> објекту приступ до <<реалног субјекта>>.
  - Обезбеђује интерфејс идентичан са интерфејсом <<Subject >> тако да <<proxy објекат>> може заменити <<RealSubject објекат>>.
  - Контролише приступ до <<RealSubject>> објекта и може бити одговоран за његово креирање и брисање.
- **Subject**
  - Дефинише заједнички интерфејс за <<RealSubject>> и <<Proxy класе>> тако да <<Proxy објекат>> може заменити <<RealSubject објекат>>.
- **RealSubject**
  - Дефинише <<RealSubject>> објекат који <<Proxy>> објекат репрезентује.

Пример Proxy узора:

docsity.com

Кориснички захтев ППХ1: *Управа Факултета је послала захтев Јава тиму Лабораторије за софтверско инжењерство да направи (састави) понуду за израду софтверског система последипломских студија ФОН-а. У понуди треба се наведе:*
а) Програмски језик у коме ће се развијати програм.
б) Систем за управљање базом података у коме ће се чувати подаци.
Управа Факултета ће надзирати (контролисати) израду понуда.
Јава тим са ФОН-а је послао захтев Јава тиму фирме JavaGroup да направи наведену понуду.

```
class UpravaFakulteta   // Client
{
  SILAB sil;
  UpravaFakulteta(SILAB sil1){sil = sil1;}

  void Konstruisi()
    { sil.kreirajProgramskiJezik();
      sil.kreirajSUBP();
    }

   public static void main(String args[])
            {  UpravaFakulteta uf;
            JavaTimGroupPonuda jtg = new JavaTimGroupPonuda();
               JavaTimPonuda jat = new JavaTimPonuda(jtg);
               uf = new UpravaFakulteta(jat);
               uf.Konstruisi();
            System.out.println("Ponuda java tima: " + jat.vratiPonudu());
  }
}
```

```
/* Uloga: Definiše zajednicki interfejs za <<JavaTimGroupPonuda>> i <<JavaTimPonuda>> tako da
<<JavaTimPonuda objekat>> moze zameniti <<JavaTimGroupPonuda objekat>>.*/
interface SILAB // Subject
{ void kreirajProgramskiJezik();
  void kreirajSUBP();
  String vratiPonudu();
}
```

```
class PonudaS { ProgramskiJezik pj;  SUBP subp;}
```

```
/* Uloge:
a) Sadrzi reference koje omogucavaju <<JavaTimPonuda>> objektu pristup do
<<JavaTimGroupPonuda>>.
b) Obezbeduje interfejs identican sa interfejsom <<Ponuda>> tako da <<JavaTimPonuda
objekat>> moze zameniti << JavaTimGroupPonuda objekat>>.
c) Kontroliše pristup do <<JavaTimGroupPonuda>> objekta i moze biti odgovoran za njegovo
kreiranje i brisanje.
*/
class JavaTimPonuda implements SILAB // Proxy
{ JavaTimGroupPonuda jtg;
  JavaTimPonuda(JavaTimGroupPonuda jtg1) {jtg = jtg1;}
  public void kreirajProgramskiJezik(){jtg.kreirajProgramskiJezik();}
  public void kreirajSUBP() {jtg.kreirajSUBP();;}
  public String vratiPonudu(){return jtg.vratiPonudu();}
}
```

```
/* Uloga:  Definise <<JavaTimGroupPonuda>> objekat koji <<JavaTimPonuda>> objekat reprezentuje.
*/
class JavaTimGroupPonuda implements SILAB // RealSubject
{
  PonudaS pon;
  JavaTimGroupPonuda() {pon = new PonudaS();}
  public void kreirajProgramskiJezik(){pon.pj = new Java();}
```


docsity.com

```
  public void kreirajSUBP() {pon.subp = new MySQL();}
  public String vratiPonudu(){return "Programski jezik-" + pon.pj.vratiProgramskiJezik() + "  SUBP-" +
pon.subp.vratiSUBP();}
}
```

// Navedeni interfejsi i klase su preuzeti iz primera za Abstract Factory uzor.

```
// ***************************************************************
interface ProgramskiJezik
{String vratiProgramskiJezik();}

class Java implements ProgramskiJezik
{ public String vratiProgramskiJezik(){return "Java";}}

class VB implements ProgramskiJezik
{ public String vratiProgramskiJezik(){return "VB";}}

// ***************************************************************

interface SUBP
{String vratiSUBP();}

class MySQL implements SUBP
{ public String vratiSUBP(){return "MySQL";}}

class MSAccess implements SUBP
{ public String vratiSUBP(){return "MS Access";}}

// ***************************************************************
```


docsity.com

ПП: Патерни понашања

Патерни понашања описују начин на који класе или објекти сарађују и распоређују одговорности.

Постоје следећи патерни понашања:
1. Chain of responsability - Избегава чврсто повезивање између пошиљаоца захтева и његовог примаоца, обезбеђујући ланац повезаних објеката, који ће да обрађују захтев све док се он не обради.
2. Command - Захтев се учаурује као објекат, што омогућава клијентима да параметризују различите захтеве. Наведеним приступом подржава се извршење повратних(undoable) операција као и опоравак података услед насилног прекида програма.
3. Interpreter - За дати језик, дефинише се репрезентација граматике језика заједно са интерпретером који користи ту репрезентацију да интерпретира реченице у језику.
4. Iterator – Обезбеђује начин да приступи елементима агрегатног објекта секвенцијално без излагања његове унутрашње репрезентације.
5. **Mediator** - Дефинише објекат који садржи скуп објеката који су у међусобној интеракцији. Интеракција између објеката може се независно мењати у односу на друге интеракције. Помоћу медијатора се успоставља слаба веза између објеката.
6. Memento - Без нарушавања учаурења memento патерн чува интерно стање објекта тако да објекат може бити враћен у то стање касније.
7. Observer - Дефинише један-више зависност између објеката, тако да промена стања неког објекта утиче аутоматски на промену стања свих других објеката који су повезани са њим.
8. State – Допушта објекту да промени понашање када се мења његова интерна структура.
9. Strategy - Дефинише фамилију алгоритама и обезбеђује њихову међузависнот. Стратегу узор омогућава промену алгоритма независно од клијента који га користи.
10. Template method – Дефинише скелет алгоритма у операцији, препуштајући извршење неких корака операција подкласама. Template method омогућава подкласама да редефинишу неке од корака алгоритама без промене алгоритамске структуре.
11. Visitor – Представља операцију која се извршава на елементима објектне структуре. Visitor омогућава да се дефинише нова операција без промене класа или елемената над којима она (операција) оперише.

docsity.com

ПП1. Chain of responsibility патерн

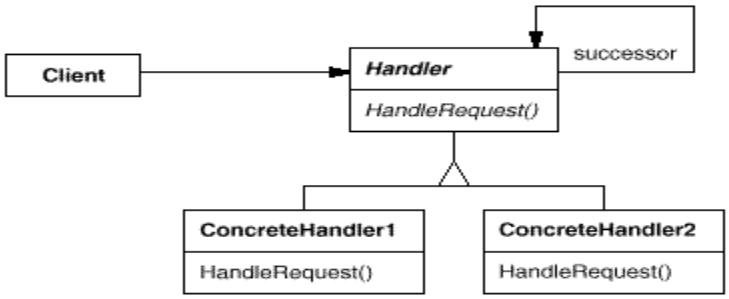
Дефиниција

Избегава чврсто повезивање између пошиљаоца захтева и његовог примаоца, обезбеђујући ланац повезаних објеката, који ће да обрађују захтев све док се он не обради.

Појашњење ГОФ дефиниције:

Избегава чврсто повезивање између пошиљаоца захтева (Client) и његовог примаоца (Handler), обезбеђујући ланац повезаних објеката(ConcreteHandler1, ConcreteHandler2), који ће да обрађују захтев све док се он не обради.

Наводимо структуру Chain of responsibility узора:



Као и учеснике у узору:

- **Handler**
  Дефинише <<интерфејс>> за обраду захтева
- **ConcreteHandler**
  - Обрађује <<захтев>> за који је одговоран.
  - Може приступити његовом <<следбенику>>.
  - Уколико може да <<обради захтев>> он га обрађује, иначе га прослеђује до <<следбеника>>.
- **Client**
  Иницира <<захтев>> који треба да се обради.

Примери Chain of responsibility узора:

```
/*Кориснички захтев COR1: Шеф Лабораторије за СИ је послао захтев
члановима Јава тима да свако од њих прикаже утрошено време у писању понуде.

Варијанта 1: Када више Хандлер-а може да обради захтев.

*/


class UpravaFakulteta
{
  SILAB sil;
  UpravaFakulteta(SILAB sil1){sil = sil1;}
```


docsity.com

```
  void kreirajPonudu(){sil.kreirajProgramskiJezik();
                       sil.kreirajSUBP();
                      sil.kreirajPonudu();}
  public static void main(String args[])
            {  UpravaFakulteta uf;
               DusanSavic ds = new DusanSavic(null,true);
            IlijaAntovic ia = new IlijaAntovic(ds,true);
            VojislavStanojevic vs = new VojislavStanojevic(ia,true);
            MilosMilic mm = new MilosMilic(vs,true);
               JavaTimPonuda jat = new JavaTimPonuda(mm);
               uf = new UpravaFakulteta(jat);
               uf.kreirajPonudu();
               System.out.println(jat.vratiPonudu());

    }
}

abstract class SILAB
{
  PonudaS elpon;
  String ponuda;
  SILAB() {elpon = new PonudaS();}
  abstract void kreirajProgramskiJezik();
  abstract void kreirajSUBP();
  abstract String utrosenoVreme();
  public void kreirajPonudu(){ponuda = "Programski jezik-" + elpon.pj.vratiProgramskiJezik() + "  SUBP-" +
elpon.subp.vratiSUBP() + " " + utrosenoVreme();}
  String vratiPonudu() {return ponuda;}
}



class PonudaS { ProgramskiJezik pj;  SUBP subp;}


/* Uloga: Inicira <<zahtev za utroseno vreme>> koji treba da se obradi.
*/
class JavaTimPonuda extends SILAB     // Client
{ ClanJavaTima cjt;
  JavaTimPonuda(ClanJavaTima cjt1){cjt = cjt1;}
  public void kreirajProgramskiJezik(){elpon.pj = new Java();}
  public void kreirajSUBP() {elpon.subp = new MySQL();}
  String utrosenoVreme(){return cjt.utrosenoVreme();} }

/* Uloga: Definiše <<apstraktnu klasu ClanJavaTima>> za obradu zahteva
*/
abstract class ClanJavaTima //Handler
{  ClanJavaTima cjt;
   boolean radioNaPonudi;
   ClanJavaTima(ClanJavaTima cjt1,boolean radioNaPonudi1){cjt = cjt1;
                     radioNaPonudi=radioNaPonudi1;}

   String utrosenoVreme(){ String uvreme = "";
               if (radioNaPonudi == true) uvreme = "\nUtroseno vreme:" + vratiVreme() + " ";
               if (cjt!=null ) uvreme = uvreme + cjt.utrosenoVreme(); // Lifo lista
            // uvreme = cjt.utrosenoVreme() + uvreme; - Fifo lista
               return uvreme;
   }

   abstract String vratiVreme();
}

/* Uloge:
o          Obraduje <<zahtev za utrosenoVreme>> za koji je odgovoran.
o          Moze pristupiti njegovom <<sledbeniku - cjt>>.
o          Ukoliko moze da <<obradi zahtev za utrosenoVreme>> on ga obraduje,
i prosleduje ga do <<sledbenika - cjt >>.

*/
class DusanSavic extends ClanJavaTima  //ConcreteHandler1
```


docsity.com

```
{ DusanSavic(ClanJavaTima cjt1,boolean radioNaPonudi1){super(cjt1,radioNaPonudi1);}
  String vratiVreme(){return " Dusan Savic - 2h";}}

class IlijaAntovic extends ClanJavaTima  //ConcreteHandler2
{ IlijaAntovic(ClanJavaTima cjt1,boolean radioNaPonudi1){super(cjt1,radioNaPonudi1);}
  String vratiVreme(){return " Ilija Antovic - 1h 30'";}}

class VojislavStanojevic extends ClanJavaTima  //ConcreteHandler3
{ VojislavStanojevic(ClanJavaTima cjt1,boolean radioNaPonudi1){super(cjt1,radioNaPonudi1);}
  String vratiVreme(){return " Vojislav Stanojevic - 1 25'";}}

class MilosMilic extends ClanJavaTima  //ConcreteHandler4
{MilosMilic(ClanJavaTima cjt1,boolean radioNaPonudi1){super(cjt1,radioNaPonudi1);}
 String vratiVreme(){return " Milos Milic - 1h 20'";}}

interface ProgramskiJezik
{String vratiProgramskiJezik();}

class Java implements ProgramskiJezik
{ public String vratiProgramskiJezik(){return "Java";}}

class VB implements ProgramskiJezik
{ public String vratiProgramskiJezik(){return "VB";}}

interface SUBP
{String vratiSUBP();}

class MySQL implements SUBP
{ public String vratiSUBP(){return "MySQL";}}

class MSAccess implements SUBP
{ public String vratiSUBP(){return "MS Access";}}
```


docsity.com

*/*__Кориснички захтев COR2:__ Шеф Лабораторије за СИ је послао захтев члановима Јава тима да се у понуди наведе члан који је био кординатор писања понуде.*

Варијанта 2: Када само један Хандлер може да обради захтев.

**/*

```
class UpravaFakulteta1
{
  SILAB sil;
  UpravaFakulteta1(SILAB sil1){sil = sil1;}

  void kreirajPonudu(){sil.kreirajProgramskiJezik();
                       sil.kreirajSUBP();
                      sil.kreirajPonudu();}

  public static void main(String args[])
            {  UpravaFakulteta1 uf;
             DusanSavic ds = new DusanSavic(null,false);
            IlijaAntovic ia = new IlijaAntovic(ds,true);
            VojislavStanojevic vs = new VojislavStanojevic(ia,false);
            MilosMilic mm = new MilosMilic(vs,false);
               JavaTimPonuda jat = new JavaTimPonuda(mm);
               uf = new UpravaFakulteta1(jat);
               uf.kreirajPonudu();
               System.out.println(jat.vratiPonudu());
    }
}

abstract class SILAB
{
  PonudaS elpon;
  String ponuda;
  SILAB() {elpon = new PonudaS();}
  abstract void kreirajProgramskiJezik();
  abstract void kreirajSUBP();
  abstract String kordinatorPonude();
  public void kreirajPonudu(){ponuda = "Programski jezik-" + elpon.pj.vratiProgramskiJezik() + "
  SUBP-" + elpon.subp.vratiSUBP() + " " + kordinatorPonude();}
  String vratiPonudu() {return ponuda;}
}


class PonudaS { ProgramskiJezik pj;  SUBP subp;}


/* Uloga: Inicira <<zahtev za utroseno vreme>> koji treba da se obradi.
*/
class JavaTimPonuda extends SILAB     // Client
{ ClanJavaTima cjt;
  JavaTimPonuda(ClanJavaTima cjt1){cjt = cjt1;}
  public void kreirajProgramskiJezik(){elpon.pj = new Java();}
  public void kreirajSUBP() {elpon.subp = new MySQL();}
  String kordinatorPonude(){return cjt.vratiKordinatorPonude();}
}

/* Uloga: Definiše <<apstraktnu klasu ClanJavaTima>> za obradu zahteva
*/
abstract class ClanJavaTima //Handler
{  ClanJavaTima cjt;
   boolean kordinatorPonude;
   ClanJavaTima(ClanJavaTima cjt1,boolean kordinatorPonude1)
     {cjt =  cjt1;kordinatorPonude=kordinatorPonude1;}
```


docsity.com

```
   String vratiKordinatorPonude(){ String uvreme ="";
              if (kordinatorPonude == true)
                { uvreme = "\nKordinator ponude:" + vratiKordinatora();
                  return uvreme; // ovde  ce se zaustaviti kretanje kroz listu.
                 }
     // Pokazace prvog na koga naidje preko lifo liste koji ima  true vrednost
     // atributa kordinatorPonude
              if (cjt!=null )uvreme = cjt.vratiKordinatorPonude();
     return uvreme;
   }


   abstract String vratiKordinatora();
}


class DusanSavic extends ClanJavaTima  //ConcreteHandler1
{ DusanSavic(ClanJavaTima cjt1,boolean kordinatorPonude1){super(cjt1,kordinatorPonude1);}
  String vratiKordinatora(){return " Dusan Savic";}}

class IlijaAntovic extends ClanJavaTima  //ConcreteHandler2
{ IlijaAntovic(ClanJavaTima cjt1,boolean kordinatorPonude1){super(cjt1,kordinatorPonude1);}
  String vratiKordinatora(){return " Ilija Antovic";}}

class VojislavStanojevic extends ClanJavaTima  //ConcreteHandler3
{ VojislavStanojevic(ClanJavaTima cjt1,boolean kordinatorPonude1){super(cjt1,kordinatorPonude1);}
  String vratiKordinatora(){return " Vojislav Stanojevic";}}

class MilosMilic extends ClanJavaTima  //ConcreteHandler4
{MilosMilic(ClanJavaTima cjt1,boolean kordinatorPonude1){super(cjt1,kordinatorPonude1);}
 String vratiKordinatora(){return " Milos Milic";}}


interface ProgramskiJezik
{String vratiProgramskiJezik();}

class Java implements ProgramskiJezik
{ public String vratiProgramskiJezik(){return "Java";}}

class VB implements ProgramskiJezik
{ public String vratiProgramskiJezik(){return "VB";}}


interface SUBP
{String vratiSUBP();}

class MySQL implements SUBP
{ public String vratiSUBP(){return "MySQL";}}

class MSAccess implements SUBP
{ public String vratiSUBP(){return "MS Access";}}
```


docsity.com

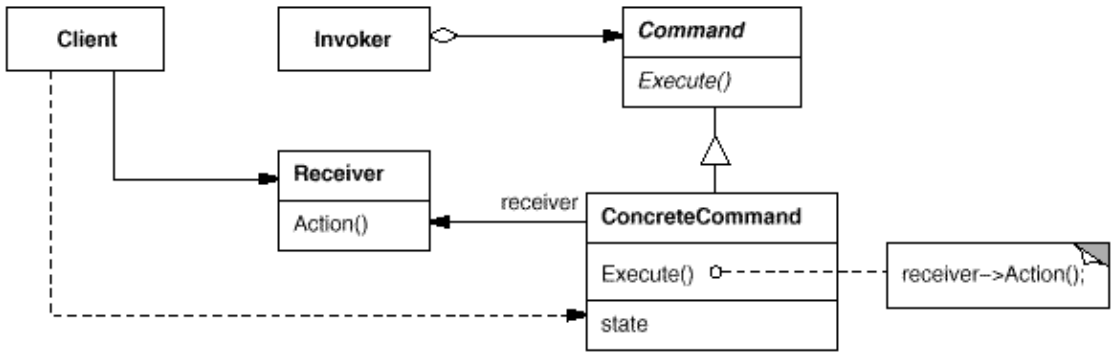
ПП2. Command патерн

Дефиниција

Захтев се учаурује као објекат, што омогућава клијентима да параметризују различите захтеве. Наведеним приступом подржава се извршење повратних(undoable) операција као и опоравак података услед насилног прекида програма

Појашњење ГОФ дефиниције:

Наводимо структуру Command узора:



*Као и **учеснике** у узору:*

- **Command**

  Декларише <<интерфејс>> за извршење операције.

- **ConcreteCommand**

  Дефинише повезивање између <<Receiver>> објекта и акције.

  Имплементира <<Изврши (Execute)>> методу позивајући одговарајућу операцију <<Receiver>> објекта.

- **Client**

  Креира <<ConcreteCommand>> објекат и дефинише његов <<Receiver>> објекат .

- **Invoker**

  Позива <<ConcreteCommand>> објекат да изврши постављени захтев.

- **Receiver**

  Извршава <<операцију>> која је придружена постављеном <<захтеву>>.

docsity.com

Пример Command узора:

*/*__Кориснички захтев Comm1:__ Шеф лабораторије одређје да је јава тим задужен да врати Управи Факултета време потребно за израду понуде. Шеф Лабораторије је одредио Душана Савића да процени колико је времена потребно да се направи понуда. Сваки пут када Управа Факултета позове Јава тим да врати време потребно да се направи понуда, Јава тим позива Душана Савића да он да процену о потребном времену. Наведену процену Јава тим шаље до Управе Факултета. */*

```
/* Uloga: Kreira <<JavaTimPonuda>> objekat i definiše njegov <<DusanSavic>> objekat .*/
class SefLaboratorije   // Klient
{ // Odredio je ko je receiver (Dusan Savic) i sta je concrete command (JavaTimPonuda).
  DusanSavic ds;
  SILAB sil;
  SefLaboratorije(){ds = new DusanSavic();sil = new JavaTimPonuda(ds);}
  SILAB vratiTimPonuda(){return sil;}
}

/* Uloga: Poziva <<JavaTimPonuda>> objekat da izvrši postavljeni zahtev. */
class UpravaFakulteta  //Invoker
{
  SILAB sil;
  UpravaFakulteta(SILAB sil1){sil = sil1;}

     public static void main(String args[])
              {  UpravaFakulteta uf;
              SefLaboratorije sl = new  SefLaboratorije();
              uf = new UpravaFakulteta(sl.vratiTimPonuda());
                 System.out.println(uf.vratiVremeZaIzraduPonude());
             }

  String vratiVremeZaIzraduPonude(){return sil.vratiVremeZaIzraduPonude();}
}


/* Uloga: Deklariše <<apstraktnu klasu Ponuda>> za izvršenje operacije.*/
abstract class SILAB // Command
{
  abstract String vratiVremeZaIzraduPonude();
}


/* Uloge: a) Definiše povezivanje izmedu <<DusanSavic>> objekta i akcije <<proceniVremeZaPonudu>>.
b) Implementira <<vratiVremeZaIzraduPonude>> metodu pozivajuci odgovarajucu operaciju <<proceniVremeZaPonudu
DusanSavic>> objekta.
*/
class JavaTimPonuda extends SILAB   // ConcreteCommand
{ ClanJavaTima cjt;
  JavaTimPonuda(ClanJavaTima cjt1){cjt=cjt1;}
  String vratiVremeZaIzraduPonude(){return cjt.proceniVremeZaPonudu();}
}


abstract class ClanJavaTima
{abstract String proceniVremeZaPonudu();
}

/* Uloga: Izvršava <<operaciju proceniVremeZaPonudu>> koja je pridruzena postavljenom
<<zahtevu vratiVremeZaIzraduPonude>>. */
class DusanSavic  extends ClanJavaTima// Receiver
{ String proceniVremeZaPonudu(){return "Potrebno je 2 dana da se napravi ponuda";}
}
```


docsity.com

ППЗ. Interpreter

Дефиниција

За дати језик, дефинише се репрезентација граматике језика заједно са интерпретером који користи ту репрезентацију да интерпретира реченице у језику.

ПП4. Iterator патерн

Дефиниција

Обезбеђује начин да приступи елементима агрегатног објекта секвенцијално без излагања његове унутрашње репрезентације.

ПП5. Mediator патерн

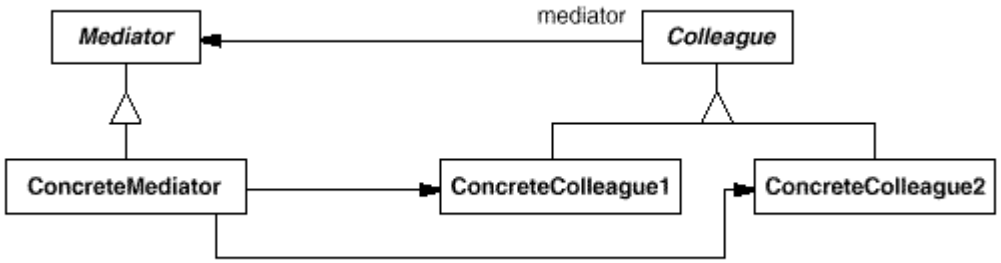
Дефиниција

Дефинише објекат који садржи скуп објеката који су у међусобној интеракцији. Интеракција између објеката може се независно мењати у односу на друге интеракције. Помоћу медијатора се успоставља слаба веза између објеката.

Појашњење ГОФ дефиниције:

Дефинише објекат (Mediator) који садржи скуп објеката (ConcreteColleague1, ConcreteColleague2) који су у међусобној интеракцији. Интеракција између објеката може се независно мењати у односу на друге интеракције. Помоћу медијатора се успоставља слаба веза између објеката(ConcreteColleague1, ConcreteColleague2).

*Наводимо **структуру** Mediator узора:*



*Као и **учеснике** у узору:*

- **Mediator**

  Дефинише <<интерфејс>> за комуникацију са <<сарадничким класама>>.

- **ConcreteMediator**

  Имплементира интеракцију између <<сарадничких класа>>.

  Медијатор зна све сараднике између којих настаје интеракција.

- **Colleague classes**

docsity.com

Сваки сарадник зна ко му је медијатор.

Сарадник комуницира са другим сарадником преко медијатора.

Пример Mediator узора:

*/*__Korisnicki zahtev Med1:__ Sef Laboratorije za SI je zaduzio Dusana Savica da kordinira procesom pripreme ponudu za tronivojsku aplikaciju koja ima: GUI, aplikacionu logiku i bazu podataka.Dusan Savic postavlja zahtev sefu laboratorije u procesu pripreme ponude da pronadje clanove tima koji su specijalizovani za GUI i aplikacionu logiku.Dusan Savic je specijalizovan za baze podataka. Sef Laboratorije je pronasao sledece clanove tima:Ilija Antovic i Milos Milic su specijalizovani za GUI.Vojislav Stanojevic je specijalizovan za alikacionu logiku.*
/

```
/* Uloga: Definiše <<interfejs Sef>> za komunikaciju sa <<saradnickim klasama>>.
*/
abstract class Sef   // Mediator
{
abstract  String pripremiPonuduBP();
abstract  String pripremiPonuduGUI();
abstract  String pripremiPonuduAL();
}


/* Uloge:
   a) Implementira interakciju izmedu <<saradnickih klasa>>.
   b) Medijator zna sve saradnike <<Dusan Savic, Ilija Antovic, Vojislav Stanojevic,
   Milos Milic>> izmedu kojih nastaje interakcija.
*/
class SefLaboratorije extends Sef    //ConcreteMediator
{ DusanSavic ds;
  IlijaAntovic ia;
  VojislavStanojevic vs;
  MilosMilic mm;
  SefLaboratorije()
    {
     ds = new DusanSavic(this);
     ia = new IlijaAntovic(this);
     vs = new VojislavStanojevic(this);
     mm = new MilosMilic(this);
    }
  String pripremiPonuduBP(){return ds.pripremiPonuduBP();}
  String pripremiPonuduGUI(){return mm.pripremiPonuduGUI();}
  String pripremiPonuduAL(){return vs.pripremiPonuduAL();}

  public static void main(String arg[])
   {
     SefLaboratorije sf = new SefLaboratorije();
     System.out.println(sf.ds.kordinirajProcesPonude());
   }
}



abstract class ClanJavaTima //Colleage
{  Sef sef;
   ClanJavaTima(Sef sef1) {sef = sef1;}

}


/* Uloge:
   a) Svaki saradnik zna ko mu je medijator.
   b) Saradnik komunicira sa drugim saradnikom preko medijatora.
*/
class DusanSavic extends ClanJavaTima  //ConcreteColleague1
```


docsity.com

```
{ DusanSavic(Sef sef1) {super(sef1);}
  String kordinirajProcesPonude(){ return sef.pripremiPonuduGUI() + " " +
     sef.pripremiPonuduAL() + " " + pripremiPonuduBP();}
  String pripremiPonuduBP(){return "Dusan Savic: Ponuda - Baza podataka";}}

class IlijaAntovic extends ClanJavaTima  //ConcreteColleague2
{ IlijaAntovic(Sef sef1) {super(sef1);}
  String pripremiPonuduGUI(){return "Ilija Antovic: Ponuda - GUI";}}

class VojislavStanojevic extends ClanJavaTima  //ConcreteColleague3
{ VojislavStanojevic(Sef sef1) {super(sef1);}
  String pripremiPonuduAL(){return " Vojislav Stanojevic: Ponuda - Aplikaciona logika";}}

class MilosMilic extends ClanJavaTima  //ConcreteColleague4
{ MilosMilic(Sef sef1) {super(sef1);}
  String pripremiPonuduGUI(){return " Milos Milic:  Ponuda GUI";}}
```

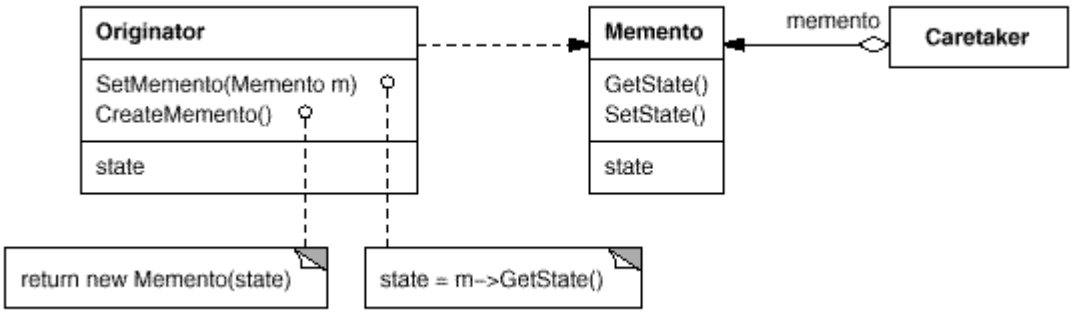

docsity.com

ППб. Memento патерн

Дефиниција
Без нарушавања учаурења memento патерн чува интерно стање објекта тако да објекат може бити враћен у то стање касније.

Појашњење ГОФ дефиниције:
Без нарушавања учаурења memento патерн чува интерно стање (Caretaker) објекта тако да објекат може бити враћен у то стање касније.

Наводимо структуру Memento патерна:



Као и учеснике у патерну:

- **Memento**
  Чува интерно стање <<Originator>> објекта.
  Штити приступ до запамћеног стања свим објектима осим << Originator >> објекту.
- **Originator**
  Креира <<memento>> објекат који чува његово интерно стање.
  Користи <<memento>> објекат да поврати запамћено стање.
- **Caretaker**
  Кординира процесом памћења стања <<Originator>> објекта и његовог каснијег враћања у то стање.

Пример Memento узора:

*/****Korisnicki zahtev MEM1:*** *Uprava Fakulteta je trazila od Java tima da napravi ponudu. Nakon posmatranja prve ponude Uprava Fakulteta je trazila da se ponuda promeni u delu koji se odnosi na procenjeno vreme izrade projekta(procenjeno vreme je 6 meseci). Uprava je trazila da se smanji vreme izrade projekta. Uprava Fakulteta je poslala prvu ponudu do Komisije za nabavke koja je zaduzena da cuva tekucu aktivnu ponudu (tekuca ponuda = prva ponuda).*
Uprava Fakulteta je trazila od Komisije da joj omogući da po potrebi može preuzeti prvu ponude.

docsity.com

Java tim je promenio procenjeno vreme na 3 meseca i poslao je drugu ponudu.
Uprava Fakulteta je poslala drugu ponudu do Komisije za nabavke
(tekuca ponuda = druga ponuda). Nakon analize druge ponude (zbog novonastalih okolnosti - Java tim je u medjuvremenu zaduzen da uradi jos jedan projekat) Uprava Fakulteta je odlucila da izabere prvu ponudu koju je preuzela od Komisije za nabavku.
(tekuca ponuda = prva ponuda).
*/

```
/* Uloge:
a) Cuva interno stanje <<KomisijaZaNabavke>> objekta.
b) Štiti pristup do zapamcenog stanja svim objektima osim
                <<KomisijaZaNabavke>> objektu.
*/
class Memento  // Memento
{ SILAB sil;
  void postaviStanje(SILAB sil1) { sil=sil1;}
  SILAB uzmiStanje() { return sil;}
}

/* Uloge:
a) Kreira <<memento>> objekat koji cuva njegovo interno stanje.
b) Koristi <<memento>> objekat da povrati zapamceno stanje.
*/
class KomisijaZaNabavke // Originator
{SILAB sil;
 KomisijaZaNabavke() {}
 void poveziSaPonudom(SILAB sil1) {sil=sil1;}
 void postaviMemento(Memento mem) { sil = mem.uzmiStanje();}
 Memento kreirajMemento(){
            Memento mem = new Memento();
            mem.postaviStanje(sil);
            return mem;
            }
}

/* Uloga: Kordinira procesom pamcenja stanja <<KomisijaZaNabavke>> objekta i njegovog
kasnijeg vracanja u to stanje.*/
class UpravaFakulteta  // CareTaker
{ static Memento mem;

  public static void main(String args[])
          {   UpravaFakulteta uf = new UpravaFakulteta();
              /* Prva ponuda */
              JavaTimPonuda jat = new JavaTimPonuda("normalno");
              KomisijaZaNabavke kzn = new KomisijaZaNabavke();
              kzn.poveziSaPonudom(jat);
              mem = kzn.kreirajMemento();
              System.out.println("Ponuda java tima: " + jat.vratiPonudu());

              /* Druga ponuda */
              jat = new JavaTimPonuda("skraceno");
              kzn.poveziSaPonudom(jat);
               System.out.println("Ponuda java tima: " + jat.vratiPonudu());

              /* Vracanje na prvu ponudu */
              kzn.postaviMemento(mem);
              System.out.println("Ponuda java tima: " + kzn.sil.vratiPonudu());
  }
}


interface SILAB
{ void kreirajProgramskiJezik();
  void kreirajSUBP();
  void kreirajProcenjenoVreme(String duz);
  String vratiPonudu();
}
```


docsity.com

```
class PonudaS { ProgramskiJezik pj;  SUBP subp; String pv;}

class JavaTimPonuda implements SILAB
{ PonudaS pon;
  JavaTimPonuda(String duz) {pon = new PonudaS(); kreirajProgramskiJezik(); kreirajSUBP();
kreirajProcenjenoVreme(duz); }

  public void kreirajProgramskiJezik(){pon.pj = new Java();}
  public void kreirajSUBP() {pon.subp = new MySQL();}
  public void kreirajProcenjenoVreme(String duz) {pon.pv = procVreme(duz);}
  String procVreme(String duz) { if (duz.equals("skraceno")== true) return "3 meseca";
                                 if (duz.equals("normalno")== true) return "6 meseci";
                                 return "";}
  public String vratiPonudu(){return "Programski jezik-" + pon.pj.vratiProgramskiJezik() + "  SUBP-" +
pon.subp.vratiSUBP() + " procenjeno vreme projekta: " + pon.pv;}
}


interface ProgramskiJezik
{String vratiProgramskiJezik();}

class Java implements ProgramskiJezik
{ public String vratiProgramskiJezik(){return "Java";}}


interface SUBP
{String vratiSUBP();}

class MySQL implements SUBP
{ public String vratiSUBP(){return "MySQL";}}
```


docsity.com

ПП7. Observer патерн

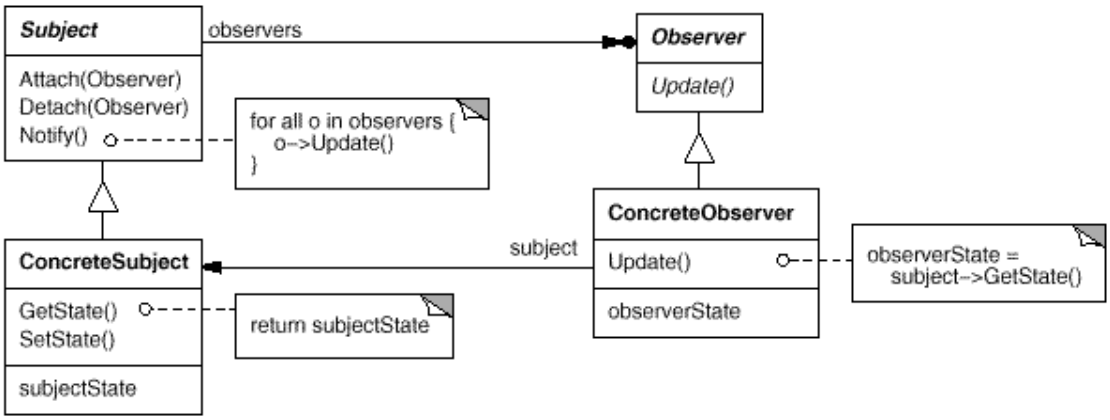
Дефиниција

Дефинише један-више зависност између објеката, тако да промена стања неког објекта утиче аутоматски на промену стања свих других објеката који су повезани са њим.

Појашњење ГОФ дефиниције:

Дефинише један-више зависност између објеката (Subject--→Observer) тако да промена стања неког објекта (ConcreteSubject) утиче аутоматски на промену стања свих других објеката који су повезани са њим(ConcreteObserver).

Наводимо структуру Observer узора:



Као и учеснике у узору:

- **Subject**
  Зна ко су његови <<observeri>>.
  Обезбеђује интерфејс за везивање и развезивање <<observera>>.
- **Observer**
  Обезбеђује интерфејс за промену објекта на ново стање у које је прешао <<Subject>> објекат.
- **ConcreteSubject (CS)**
  Чува стање на које се постављају <<observeri>>.
- **ConcreteObserver (CO)**
  Чува референцу на <<ConcreteSubject>> објекат.
  Чува стање које је конзистентно са стањем <<ConcreteSubject>> објекта.
  Имплементира интерфејс којим се објекат поставља на ново стање у које је прешао <<Subject>> објекат.

docsity.com

Пример узора:

*/*__Korisnicki zahtev Obs1:__ Uprava Fakulteta je zaduzila Komisiju za nabavke da odredi vreme do kada treba da se posalju ponude.*
Uprava Fakulteta obavestava sve timove da se obrati Komisiji za nabavke kako bi videli do kog datuma treba poslati ponudu. Timovi treba da zapamte datum do kada treba poslati ponudu.
**/*

```
/* Uloge:
a) Zna ko su njegovi <<timovi>>.
b) Obezbeduje interfejs za vezivanje i razvezivanje <<timova>>.
*/
class UpravaFakulteta   // Subject
{
  SILAB sil[]; // ConcreteSubjects
  int brObs; // broj observera

  UpravaFakulteta(){sil = new SILAB[2]; brObs = 0;}

  void Dodaj(SILAB sil1) {sil[brObs++] = sil1;}
  void Izbaci(){}

  void obavestiTimove()
   { for(int i=0; i<brObs;i++)
       sil[i].proveriDatum();
   }

   public static void main(String args[])
              { UpravaFakulteta uf = new UpravaFakulteta();
             KomisijaZaNabavke kzn = new KomisijaZaNabavke();

              kzn.odrediDatum();

                 JavaTimPonuda jat = new JavaTimPonuda(kzn);  // ConcreteObserver1
                 uf.Dodaj(jat);

                 VBTimPonuda vbt = new VBTimPonuda(kzn); // ConcreteObserver2
              uf.Dodaj(vbt);

              uf.obavestiTimove();
           }
}

/* Uloge: Cuva stanje na koje se postavljaju <<timovi>>. */
class KomisijaZaNabavke extends UpravaFakulteta  // ConcreteSubject
{ String datumPonude;
  void odrediDatum(){datumPonude = "22-dec-10";}
  String vratiDatum(){return datumPonude;}
}


/* Uloga: Obezbeduje interfejs za promenu objekta na novo stanje u koje je prešao
<<KomisijaZaNabavke>> objekat. */
abstract class SILAB // Observer
{ KomisijaZaNabavke kzn;
  String datumPonude;
  SILAB(KomisijaZaNabavke kzn1) {kzn = kzn1;}
  abstract void proveriDatum();
}

/* Uloge:
a) Cuva referencu na <<KomisijaZaNabavke>> objekat.
b) Cuva stanje koje je konzistentno sa stanjem <<KomisijaZaNabavke>> objektom.
*/
class JavaTimPonuda extends SILAB // ConcreteObserver1
{ JavaTimPonuda(KomisijaZaNabavke kzn1){super(kzn1);}
  void proveriDatum() { datumPonude = kzn.vratiDatum();
```


docsity.com

```
                System.out.println("Java tim - Datum do kada treba poslati ponudu: " + datumPonude);}
}

class VBTimPonuda extends SILAB // ConcreteObserver2
{ VBTimPonuda(KomisijaZaNabavke kzn1){super(kzn1);}
  void proveriDatum() { datumPonude = kzn.vratiDatum();
                System.out.println("VB tim - Datum do kada treba poslati ponudu: " + datumPonude);}
}
```

ППВ8. State патерн

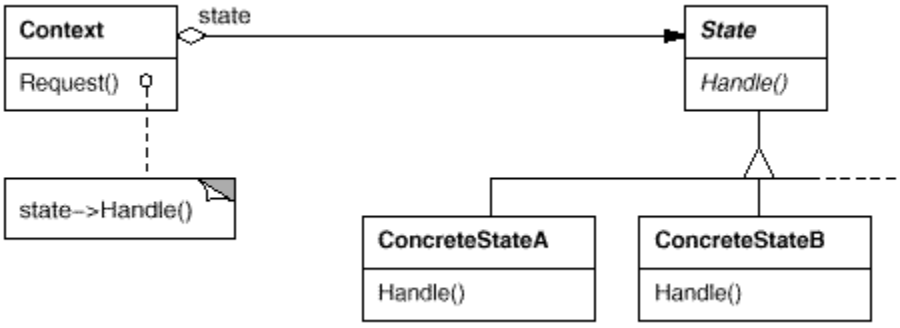
Дефиниција

Допушта објекту да промени понашање када се мења његова интерна структура.

Појашњење ГОФ дефиниције:

Допушта објекту (Context) да промени понашање (ConcreteStateA, ConcreteStateB) када се мења његова интерна структура (Request).

Наводимо структуру State патерна:



Као и учеснике у State патерну:

- **Context**
  - Дефинише интерфејс за клијента.
  - Садржи појављивање ConcreteState подкласе која дефинише текуће стање.
- **State**
  - Дефинише интерфејс за Context објекат, односно садржи понашање које се мења у зависности од промене стања контекст објекта.
- **ConcreteState подкласа** (ConcreteStateA,ConcreteStateB)
  - Свака подкласа (која представља стања контекст објекта) имплементира понашање State класе.

docsity.com

Пример патерна:

Кориснички захтев: *Управа Факултета ће у зависности од расположивих средстава (контекста) да одреди да ли ће прихватити понуду Java или VB тима. Уколико су расположива средства већа од 150000 пројекат ће радити Java тим, иначе ће пројекат радити VB тим.*

```
class UpravaFakulteta   // Context
{
  static SILAB sil; // State
  static double raspSredstva;

  public static void main(String args[])
              {  raspSredstva = Double.parseDouble(args[0]);
                 if (raspSredstva > 150000)
                    { sil = new JavaTimPonuda();  // ConcreteState1
                    }
                 else
                    { sil = new VBTimPonuda(); // ConcreteState2
                    }

       System.out.println(sil.izabranTim());

   }
}

abstract class SILAB// State
{ abstract String izabranTim();
}

class JavaTimPonuda extends SILAB // ConcreteState1
{ String izabranTim() {return "Java tim!";}
}

class VBTimPonuda extends SILAB // ConcreteState2
{ String izabranTim() {return "VB tim!";}
}
```


docsity.com

ПП9. Strategy патерн

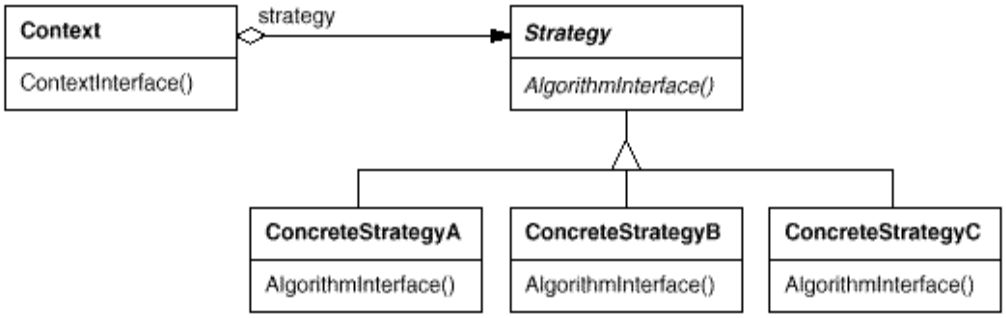
Дефиниција

Дефинише фамилију алгоритама, учаурује сваки од њих и обезбеђује да они могу бити замењиви. Strategy патерн омогућава промену алгоритма независно од клијента који га користи.

Појашњење GOF дефиниције:

Дефинише фамилију алгоритама (ConcreteStrategyA, ConcreteStrategyB, ...), учаурује сваки од њих и обезбеђује да они могу бити замењиви (Strategy). Strategy патерн омогућава промену алгоритма (ConcreteStrategyA, ConcreteStrategyB, ...) независно од клијента (Context) који га користи.

Наводимо структуру Strategy патерна:



Као и учеснике у Strategy патерну:

- **Strategy**
  - Декларише interfejs koji je zajednički za sve algoritme koji ga podržavaju. Context koristi ovaj interfejs da pozove algoritam definisan sa ConcreteStrategy klasom.

- **ConcreteStrategy**
  - Implementira algoritam koji koristi Strategy interfejs.

- **Context**
  - Context објекат је конфигурисана са ConcretyStrategy објектом.
  - Он садржи референцу на Strategy објекат.
  - Може да дефинише интерфејс, који омогућава Strategy објекту приступ, до његових података.

docsity.com

Пример патерна:

Кориснички захтев: *Управа Факултета захтева од Java и VB тима да предложе најважније кораке у развоју софтверског система. Управа Факултета одређује тим који ће развијати софтверски систем.*

```
class UpravaFakulteta   // Context
{
  static SILAB sil; // State

  public static void main(String args[])
            {   sil = new
                System.out.println(sil.strategijaRazvoja());
            }
}
abstract class SILAB// Strategy
{ abstract String strategijaRazvoja ();
}

class JavaTimPonuda extends SILAB // ConcreteStrategy1
{ String strategijaRazvoja() { String pom = "Use case driven strategija.";
                               pom = pom +   " Larmanova metoda.";
                               pom = pom +  " Iterativno-inkrementalni model.";
                               return pom;}
}

class VBTimPonuda extends SILAB // ConcreteStrategy 2
{ String strategijaRazvoja() { String pom =  "Test driven strategija.";
                               pom = pom +  " Extreme programming .";
                               pom = pom +  " Iterativno-inkrementalni model.";
                               return pom;}
}
```


docsity.com

ППП10. Template method

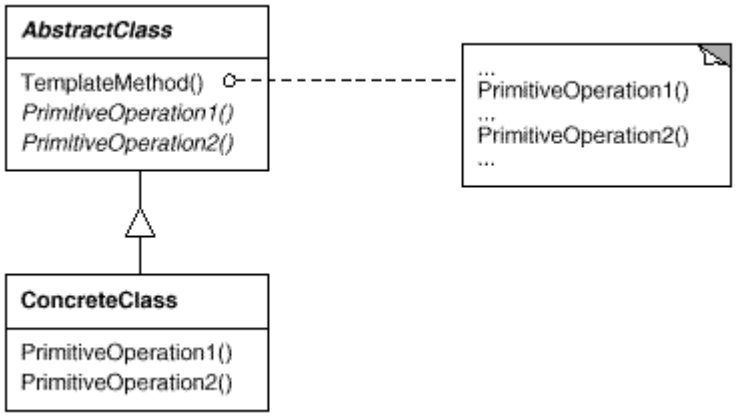
Дефиниција

Дефинише скелет алгоритма у операцији, препуштајући извршење неких корака операција подкласама. Template method омогућава подкласама да редефинишу неке од корака алгоритама без промене алгоритамске структуре.

Појашњење ГОФ дефиниције:

Дефинише скелет алгоритма у операцији (TemplateMethod), препуштајући извршење неких корака операција подкласама (ConcreteClass). Template method омогућава подкласама да редефинишу неке од корака алгоритама (PrimitiveOperation1, PrimitiveOperation2) без промене алгоритамске структуре (TemplateMethod).

Наводимо структуру Template method патерна:



Као и учеснике у Template method патерну:

- **AbstractClass**

  Дефинише апстрактне примитивне операције које конкретна подкласа имплементира.

  Имплементира алгоритам template method-а. Template method позива примитивне операције.

- **ConcreteClass**

  Имплементира примитивне операције које описују специфична понашања подкласа.

docsity.com

***Пример патерна**:*

***Кориснички захтев:** Управа Факултета захтева од Лабораторије за софтверско инжењерство да предложи најважније кораке у развоју софтверског система. Java и VB тим ће предложити конкретне кораке који ће се извшити у развоју софтверског система. Управа Факултета одређује тим који ће развијати софтверски систем.*

```
class UpravaFakulteta
{
  static SILAB sil; // AbstractClass
  public static void main(String args[])
              {   sil = new  JavaTimPonuda();
                  System.out.println(sil.koraciRazvoja());
             }
}
abstract class SILAB //  AbstractClass
{ String koraciRazvoja ()
      { String pom = IzborStrategije();
        pom = pom + IzborMetode();
        pom = pom + IzborModela();
        return pom;
       }
   abstract String IzborStrategije();
   abstract String IzborMetode();
   abstract String IzborModela();

}

class JavaTimPonuda extends SILAB // ConcreteClass1
{  String IzborStrategije() {return "Use case driven strategija.";}
   String IzborMetode() {return " Larmanova metoda.";}
   String IzborModela() { return " Iterativno-inkrementalni model.";}
}

class VBTimPonuda extends SILAB // ConcreteClass2
{  String IzborStrategije() {return "Test driven strategija.";}
   String IzborMetode() {return " Extreme programming . ";}
   String IzborModela() { return " Iterativno-inkrementalni model.";}
}
```


docsity.com

ППП11. Visitor

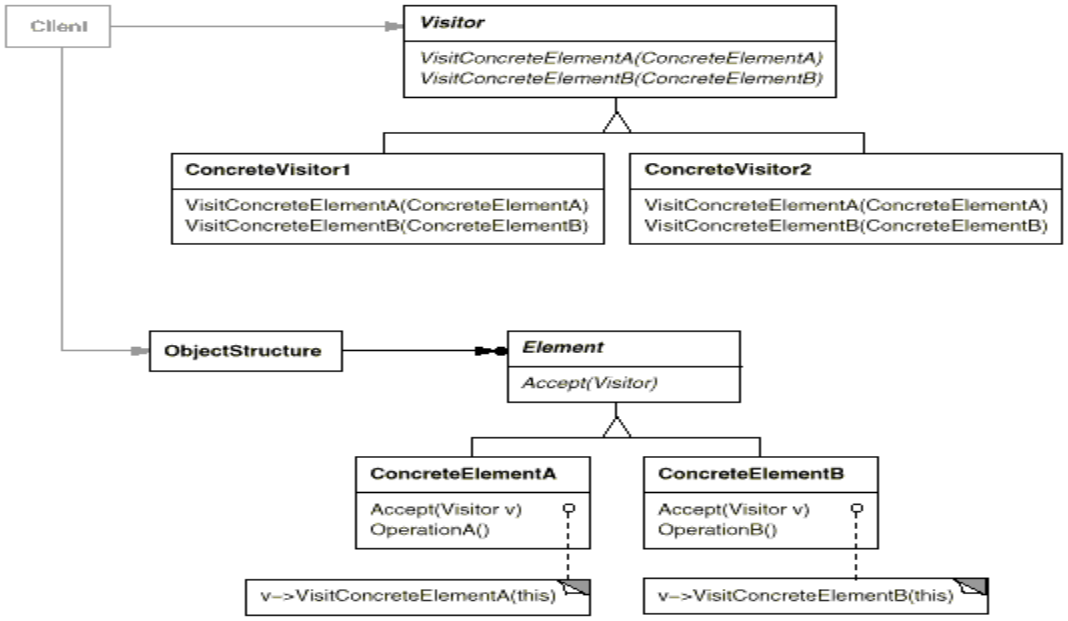
Дефиниција

Представља операцију која се извршава на елементима објектне структуре. Visitor омогућава да се дефинише нова операција без промене класа или елемената над којима она (операција) оперише.

Појашњење ГОФ дефиниције:

Представља операцију (ObjectStructure) која се извршава на елементима објектне структуре (ConcreteElementA, ConcreteElementB). Visitor омогућава да се дефинише нова операција (VisitConcreteElementA, VisitConcreteElementB) без промене класа или елемената (ConcreteElementA, ConcreteElementB) над којима она (операција) оперише.

Наводимо структуру Visitor патерна:



Као и учеснике у Visitor патерну:

- **Visitor –** декларише операцију Visit() за сваку ConcreteElement класу објектне структуре. Наведена операција као аргумент садржи објекат ConcreteElement класе коме ће да приступи преко интерфејса његове класе.
- **ConcreteVisitor** – имплементира сваку операцију декларисану преко Visitora. Свака операција имплементира део генералног алгоритма који је дефинисан или код клијента или код класе ObjectStructure. ConcreteVisitor обезбеђује контекст алгоритма и чува његово локално стање. Ово стање често акумулира резултате у току кретања кроз структуру.
- **Element** – дефинише операцију Accept() која прихвата Visitor као аргумент.
- **ConcreteElement** - имплементира операцију Accept().
- **ObjectStructure** - чува елементе објектне структуре и обезбеђује интерфејс који допушта Visitor-у да види наведене елементе. Може бити или композиција или колекција као што су листа или скуп.

docsity.com

Пример патерна:

Кориснички захтев: *Управа Факултета одредјује да ће понуду стратегије и методе развоја софтвера урадити Јава тим. Управа Факултета позива сефа лабораторије за СИ да одреди чланове Јава тима који су одговорни за одредјивање стратегије и методе развоја софтвера.*/*

```
class UpravaFakulteta
 {
   public static void main(String[] arg)
   { JavaTimPonuda jtp = new JavaTimPonuda();
              SefLaboratorije sl = new SefLaboratorije();
              sl.odrediOdgovornost(jtp);

     // Ukoliko bi uprava izabrala VB tim da pripremi ponudu
     VBTimPonuda vtp = new VBTimPonuda();
              sl.odrediOdgovornost(vtp);
            }

 }

class SefLaboratorije  // ObjectStructure
{ ElementPonude el[];

  SefLaboratorije()
   {  el = new ElementPonude[2];
      el[0] = new Strategija();
      el[1] = new Metoda();
   }

  void odrediOdgovornost(SILAB s)
   {  el[0].PrihvatiTim(s);
      el[1].PrihvatiTim(s);
      System.out.println("Odgovoran za strategiju: " + el[0].vratiOdgovoran());
      System.out.println("Odgovoran za metodu: " + el[1].vratiOdgovoran());
    }
}


abstract class ElementPonude   // Element
{ String odgovoran;
  abstract void PrihvatiTim (SILAB s);  // Accept()
  abstract String vratiOdgovoran();
}


class Strategija extends ElementPonude  // ConcreteElement1
{
  void PrihvatiTim (SILAB s) {s.OdgovoranZaStrategiju(this);}
  void odgovoranZaStrategiju(String odgovoran1) {odgovoran = odgovoran1;}
  String vratiOdgovoran() { return odgovoran;}
}

class Metoda extends ElementPonude // ConcreteElement2
{
  void PrihvatiTim (SILAB s) {s.OdgovoranZaMetodu(this);}
  void odgovoranZaMetodu(String odgovoran1) {odgovoran = odgovoran1;}
  String vratiOdgovoran() { return odgovoran;}
}


abstract class SILAB //Visitor
{
  abstract void OdgovoranZaStrategiju(Strategija s);
  abstract void OdgovoranZaMetodu(Metoda m);
}
```


docsity.com

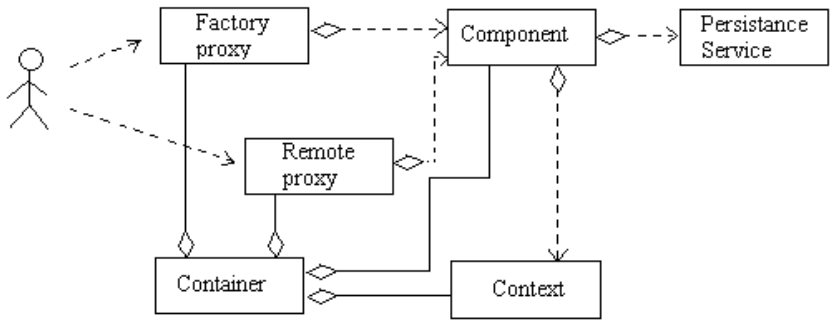
```
class JavaTimPonuda extends SILAB // ConcreteVisitor1
{
  void OdgovoranZaStrategiju(Strategija s){s.odgovoranZaStrategiju("Vojislav Stanojevic");}
  void OdgovoranZaMetodu(Metoda m){m.odgovoranZaMetodu("Milos Milic");}
}


class VBTimPonuda extends SILAB // ConcreteVisitor2
{
  void OdgovoranZaStrategiju(Strategija s){s.odgovoranZaStrategiju("Dusan Savic");}
  void OdgovoranZaMetodu(Metoda m){m.odgovoranZaMetodu("Ilija Antovic");}
}
```


docsity.com

4.5 МАКРО АРХИТЕКТУРА

ECF (Enterprise Component Framework) ПАТЕРН

ECF је макроархитектурни патерн за развој сложених (Enterprise) дистрибуираних апликација које су засноване на софтверским компонентама (Component), које се могу поново користити у новим проблемским ситуацијама.
ECF (Слика ECF) садржи следеће елементе: Client, FactoryProxy, RemoteProxy, Context, Component, Container i PersistenceService.
Наведени елементи имају следеће улоге:
Client поставља захтев за извршење операције над Component елементом.
Proxy елемент, пресреће клијентски захтев и и извршава операцију над Component елементом. FactoryProxy је задужен да обезбеди следеће операсије: create(), find() и remove(), док је RemoteProxy задужен да обезбеди остале операције које позива клијент.
Context елемент постоји за сваки component елемент и он чува његов контекст, као што је: стање трансакције, перзистентност, сигурност,…
Container елемент обухвата (агрегира) све елементе ECF узора осим Client и PersistenceService елемената. Он обезбеђује run-time окружење на коме се извршавају разни сервиси дистрибуиране обраде апликација, као што су: међупроцесна комуникација, сигурност, перзистентност и трансакције. Component елемент, када жели да обезбеди своју перзистентност позива PersistanceService елемент који је за то задужен.



Slika ECF: ECF патерн

Из ECF патерна су изведене EJB (Enterprise JavaBeans) и COM+ архитектуре.

docsity.com

MVC (Model-View-Controller) ПАТЕРН

MVC (Слика MVC) је макроархитектурни патерн, који дели софтверски систем у три дела:
а) **view** - обезбеђује кориснику интерфејс (екранску форму) помоћу које ће корисник да уноси податке и позива одговарајуће операције које треба да се изврше над model-ом. View приказује кориснику стање модела.
b) **controller** – ослушкује и прихвата захтев од клијента за извршење операције. Након тога позива операцију која је дефинисана у моделу. Уколико model промени стање controller обавештава view да је промењено стање.
c) **model** – представља стање система. Стање могу мењати неке од операција model-а.

Правило 1: Контролер прати догађаје који се извршавају над view и на одговарајући начин реагује на њих.

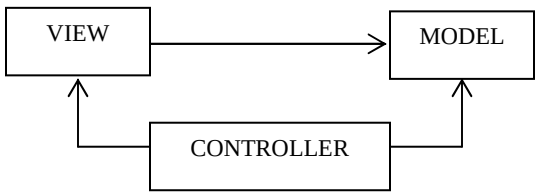
Правило 2: Контролер је диспечер који прима захтев од view-а и преусмерава га до model-а.

Правило 3: View мора да рефлектује стање модела. Сваки пут када се промени стање model-а, view треба да буде обавештен о томе. Нпр. код Observer патерна ConcreteSubject елемент је **Model** док је ConcreteObserver елемент **View**. Када се промени стање од ConcreteSubject елемента тада Subject (који јр **Controller)** обавештава ConcreteObserver да је промењено стање и тада ConcreteObserver чита ново стање од ConcreteSubject-а.

Правило 4: Model не мора да зна ко је View и Controller.

Connelly Barnes је рекао: " Најлакши начин да разумете MVC је: model је податак, view је екранска форма, controller је лепак између modela и view-а.

У књизи **Design Patterns** MVC се објашњава на следећи начин: Model је апликациони објекат, View је екранска презентација а Controller дефинише како кориснички интерфејс реагује на корисничке улазе. MVC је објашњен у контексту прављења корисничког интерфејса код SmallTalk програмског језика.



Slika MVC: MVC патерн

Пример MVC1:

/* Кориснички захтев: Направити екранску форму која ће имати: а) поље за прихват бројева и
б) поље за приказ збира свих до тада унетих бројева. Збир се рачуна кликом на дугме. */

```
// View.java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;


public class View extends JFrame
{ Model mod;
  // Labele koja sadrzi naziv ekranske forme koja se otvara.
  private JLabel LNazivForme;
```


docsity.com

```
  // Polja preko kojih se obradjuju podaci.
  private JFormattedTextField PBroj;
  private JFormattedTextField PZbir;

  // Labele koje opisuju polja za obradu podataka.
  private JLabel LBroj;
  private JLabel LZbir;

  // Dugme preko koga se poziva sistemska operacija.
  public JButton BZbir;

// 1. Konstruktor ekranske forme
public View (Model mod1)
  { KreirajKomponenteEkranskeForme();          // 1.1
    PokreniMenadzeraRasporedaKomponeti(); // 1.2
    PostaviImeForme();                          // 1.3
    PostaviPoljeZaPrihvatBrojeva();          // 1.4
    PostaviPoljeZaZbir();                       // 1.5
    PostaviLabeluZaPrihvatBrojeva();         //  1.6
    PostaviLabeluZaZbir();                      // 1.7
    PostaviDugmeZbir();                          // 1.8
    mod = mod1;
    postaviVrednost();
    pack();
    show();
    }

// 1.1 Kreiranje i inicijalizacija komponenti ekranske forme
void KreirajKomponenteEkranskeForme()
  { LNazivForme = new JLabel();
    PBroj = new JFormattedTextField();
    PZbir = new JFormattedTextField();
    LBroj= new JLabel();
    LZbir = new JLabel();
    BZbir = new JButton();
 }

// 1.2 Kreiranje menadjera rasporeda komponenti i njegovo dodeljivanje do kontejnera okvira(JFrame komponente).
void PokreniMenadzeraRasporedaKomponeti()
  { getContentPane().setLayout(new AbsoluteLayout());}

// 1.3 Odredivanje naslovnog teksta i njegovo dodeljivanje do kontejnera okvira.
void PostaviImeForme()
{  LNazivForme.setFont(new Font("Times New Roman", 1, 12));
   LNazivForme.setText("SABIRANJE NIZA BROJEVA");
   getContentPane().add(LNazivForme, new AbsoluteConstraints(20, 10, -1, -1));
}

// 1.4
void PostaviPoljeZaPrihvatBrojeva()
 { // Dodeljivanje pocetne vrednosti i formata polja.
   PBroj.setValue(new String("0"));
   // Polje se dodaje kontejneru okvira (JFrame).
    getContentPane().add(PBroj, new AbsoluteConstraints(50, 70, 70, -1));
}

// 1.5
void PostaviPoljeZaZbir()
 {  // Dodeljivanje pocetne vrednosti i formata polja.
    PZbir.setValue(new String("0"));
    // Vrednost polja ne moze da se menja.
   PZbir.setEditable(false);
    // Polje se dodaje kontejneru okvira (JFrame)
    getContentPane().add(PZbir, new AbsoluteConstraints(50, 100, 90, -1));
 }

// 1.6
void PostaviLabeluZaPrihvatBrojeva()
{ LBroj.setText("Broj");
  getContentPane().add(LBroj, new AbsoluteConstraints(20, 70, -1, -1));}
```


docsity.com

```
// 1.7
void  PostaviLabeluZaZbir()
 { LZbir.setText("Zbir");
   getContentPane().add(LZbir, new AbsoluteConstraints(20, 100, -1, -1));
 }

//*********************************************************************************
// 1.8
void PostaviDugmeZbir()
 { BZbir.setText("Zbir");
   getContentPane().add(BZbir, new AbsoluteConstraints(160, 60, -1, -1));
}

public int uzmiVrednost(){
      return Integer.parseInt((String)PBroj.getValue()); }

public void postaviVrednost()  { PZbir.setValue(String.valueOf(mod.uzmiBroj())); }

}
```

// Controller.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Controller
{  Model mod;
   View view;
   Controller(View view1,Model mod1)    {mod = mod1; view = view1; OsluskujDugmeZbir(); }

    void OsluskujDugmeZbir()
     { view.BZbir.addActionListener(new mojOsluskivac(this));}

}

class mojOsluskivac implements ActionListener
  {  Controller c;

     mojOsluskivac (Controller c1)  {c=c1;}

     public void actionPerformed(ActionEvent evt)
                 {  int pom = c.view.uzmiVrednost();
                    c.mod.sistemskaOperacija(pom);
                    c.view.postaviVrednost();
                 }
 }
```

// Model.java

```
public class Model
{ int broj;
  Model() {broj = 0;}
public void sistemskaOperacija(int broj1)  {  broj = broj + broj1; }
public int uzmiBroj () { return broj;}
}
```

// MVC1.java – glavni program

```
public class MVC1
{
public static void main(String args[])
 {  Model mod = new Model();
     View view = new View(mod);
    Controller con = new Controller(view, mod);
    }
}
```


docsity.com

Пример MVC2: /* Кориснички захтев: Направити три екранске форме истог типа које ће имати: а) поље за прихват бројева и б) поље за приказ збира свих до тада унетих бројева. Збир се рачуна кликом на дугме форме. Сваки пут када се промени збир унетих бројева та промена треба да се види на свакој од наведених форми. То значи да када се промени стање модела контролер треба да јави екранским формама да је промењено стање.*/

```
// View.java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class View extends JFrame
{  // Isto kao u MVC1
...
// 1. Konstruktor ekranske forme
public View (Model mod1, String Naziv)
  { KreirajKomponenteEkranskeForme(Naziv);     // 1.1
    PokreniMenadzeraRasporedaKomponeti();     // 1.2
    PostaviImeForme();                         // 1.3
    PostaviPoljeZaPrihvatBrojeva();            // 1.4
    PostaviPoljeZaZbir();                      // 1.5
    PostaviLabeluZaPrihvatBrojeva();           // 1.6
    PostaviLabeluZaZbir();                     // 1.7
    PostaviDugmeZbir();                        // 1.8
    mod = mod1;
    postaviVrednost();
    pack();
    show();
    }

// 1.1 Kreiranje i inicijalizacija komponenti ekranske forme
void KreirajKomponenteEkranskeForme(String Naziv)
  { LNazivForme = new JLabel();
    PBroj = new JFormattedTextField();
    PZbir = new JFormattedTextField();
    LBroj= new JLabel();
    LZbir = new JLabel();
    BZbir = new JButton();
    Container con = getContentPane();
    con.setName(Naziv);
 }


// Isto kao u MVC1
...
}

// Controller.java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;


public class Controller
{ Model mod;
   View view[];
    Controller(View[] view1,Model mod1)
        { mod = mod1; view = view1;
          for(int i=0;i<view.length;i++ ) OsluskujDugmeZbir(i); }

    void OsluskujDugmeZbir(int i)
      {   view[i].BZbir.addActionListener(new mojOsluskivac(this)); }
}
```


docsity.com

```
class mojOsluskivac implements ActionListener
 {    Controller c;

      mojOsluskivac (Controller c1)
       {c=c1;}
      public void actionPerformed(ActionEvent evt)
         {   int pom = 0;
             Object ob = evt.getSource();
             JButton b = (JButton) ob;
             Container con = b.getParent();
             for(int i1=0;i1<c.view.length;i1++ )
                 {  if (con.getName().equals(c.view[i1].getContentPane().getName()))
                          pom = c.view[i1].uzmiVrednost();
                  }

             c.mod.sistemskaOperacija(pom);

             for(int i1=0;i1<c.view.length;i1++ )
                {  c.view[i1].postaviVrednost();
                 }
         }
 }


// Model.java
public class Model
{ int broj;
  Model() {broj = 0;}
  public void sistemskaOperacija(int broj1)   {  broj = broj + broj1; }
  public int uzmiBroj () { return broj;}
}


// MVC2.java
public class MVC2
{
// Glavni program
public static void main(String args[])
 {  Model mod = new Model();
    View[] view = new View[3];
    view[0] = new View(mod,"1");
    view[1] = new View(mod,"2");
    view[2] = new View(mod,"3");
    Controller con = new Controller(view, mod);
 }
}
```


docsity.com

Пример MVC3: /* Кориснички захтев: Направити три екранске форме истог типа које ће имати: а) поље за прихват бројева и б) поље за приказ збира свих до тада унетих бројева. Збир се рачуна кликом на дугме форме. Сваки пут када се промени збир унетих бројева та промена треба да се види на свакој од наведених форми. Овај задатак треба урадити преко аспекта. То значи да када се промени стање модела, аспект треба да обавести контролер да јави екранским формама да је промењено стање. */

```
// View.java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class View extends JFrame
{  // Isto kao u primeru MVC2

}


// Controller.java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;


public class Controller
{ Model mod;
  View view[];

   Controller(View[] view1,Model mod1)
                {mod = mod1; view = view1;
                 for(int i=0;i<view.length;i++ ) OsluskujDugmeZbir(i); }

    void OsluskujDugmeZbir(int i)
     {   view[i].BZbir.addActionListener(new mojOsluskivac(this));
     }

    void procitajStanje()
     {  for(int i1=0;i1<view.length;i1++ )
              {   view[i1].postaviVrednost();   }
     }

 }


class mojOsluskivac implements ActionListener
 {     Controller c;
       mojOsluskivac (Controller c1)     {c=c1;}
       public void actionPerformed(ActionEvent evt)
         {       int pom = 0;
                 Object ob = evt.getSource();
                 JButton b = (JButton) ob;
                 Container con = b.getParent();
                 for(int i1=0;i1<c.view.length;i1++ )
                    {  if (con.getName().equals(c.view[i1].getContentPane().getName()))
                              pom = c.view[i1].uzmiVrednost();
                    }
                 c.mod.sistemskaOperacija(pom);
          }
}
```


docsity.com

// Model.java

```
public class Model
{ int broj;
  Model() {broj = 0;}
  public void sistemskaOperacija(int broj1)   {  broj = broj + broj1; }
   public int uzmiBroj () { return broj;}
}
```

// **MVC3.java**

```
public class MVC3
{  static Controller con;
   static View[] view;
   static Model mod;
// Glavni program
public static void main(String args[])
   {   mod = new Model();
       view = new View[3];
       view[0] = new View(mod,"1");
       view[1] = new View(mod,"2");
       view[2] = new View(mod,"3");
       con = new Controller(view, mod);
   }
}
```

// **Asp1.aj**

```
public aspect Asp1
 {
     pointcut SO() : execution(public void Model.sistemskaOperacija(int ));

     after() returning : SO() { MVC3.con.procitajStanje();}

}
```


docsity.com

ИМПЛЕМЕНТАЦИОНИ ПАТЕРНИ – ПРОГРАМСКИ ИДИОМИ

Програмски идиоми су патерни који се користе у конкретном програмском језику како би поједноставили програмирање. Навешћу неколико примера у Јави:

1. Коришћење for петље код приказа елемената колекције

Почетна верзија програма

```
import java.util.*;

class Idiom1D
 {
            public static void main(String[] arg)
             {   List list = new ArrayList();
                 String a = "Pera";
                 String b = "Laza";
                 list.add(a);
                 list.add(b);
                 String s;
                 for(Iterator i = list.iterator(); i.hasNext();)
                   { s = (String) i.next(); System.out.print(s + " ");}
             }
}
```

Поједностављена верзија програма

```
import java.util.*;

class Idiom1P
 {
            public static void main(String[] arg)
             {   List<String> list  = new ArrayList <String>();
                 String a = "Pera";
                 String b = "Laza";
                 list.add(a);
                 list.add(b);
                 for(String s : list)  System.out.print(s + " ");
             }
}
```

2. Скраћивање наредбе System.out.println

Почетна верзија програма

```
class Idiom2D
 {
            public static void main(String[] arg)
             {
                System.out.println("Danas je lep dan!");

             }
}
```


docsity.com

Поједностављена верзија програма

```
import static java.lang.System.out;

class Idiom2P
 {
            public static void main(String[] arg)
             {  out.println("Danas je lep dan!");
             }
}
```

3. Коришћење Scanner класе уместо BafferedReader класе

Почетна верзија програма

```
import java.io.*;

class Idiom3D
 {
            public static void main  (String[] arg) throws Exception
             {
                BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
                System.out.println("Unesi rec:");
                String rec = br.readLine();
                 System.out.println("Uneta rec je : " + rec);

             }
}
```

Поједностављена верзија програма

```
import java.util.Scanner;
import static java.lang.System.out;

class Idiom3P
 {
      public static void main  (String[] arg)
             {  Scanner sc = new Scanner(System.in);
                out.println("Unesi rec:");
                String rec = sc.nextLine();
                out.println("Uneta rec je : " + rec);

             }
 }
```


docsity.com

4. Штампање садржаја низа

Почетна верзија програма

```
import java.util.Arrays;
import java.util.List;
import java.util.Iterator;


class Idiom4D
 {
            public static void main  (String[] arg) throws Exception
             { String[] strArray = new String[]{"Pera","Mika","Laza"};
               List list = Arrays.asList(strArray);
               Iterator itr = list.iterator();
                while(itr.hasNext())  { System.out.print(itr.next() + " ");}
             }
}
```

Поједностављена верзија програма

```
import java.util.Arrays;

class Idiom4P {
        public static void main  (String[] arg) throws Exception
                {  String[] imena = new String[]{"Pera","Mika","Laza"};
                   System.out.print(Arrays.asList( imena ));
                }
}
```


docsity.com