**УНИВЕРЗИТЕТ У БЕОГРАДУ ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА**

Катедра за софтверско инжењерство

Лабораторија за софтверско инжењерство(**СИЛАБ**)

Семинарски рад из предмета:

СОФТВЕРСКИ ПАТЕРНИ

|  |  |  |
| --- | --- | --- |
| Професор: | Студент: | Бројиндекса: |
| **Проф. др Синиша Влајић** | **Maрина Јовановић** | **0311/2018** |

Београд - 2022.

Contents

[1. СОФТВЕРСКИ ПАТЕРНИ 5](#_Toc95410781)

[1.1. Основне дефиниције патерна 5](#_Toc95410782)

[1.2. Општи облик GOF патерна пројектовања 7](#_Toc95410783)

[1.3. Примењивост патерна у животу 10](#_Toc95410784)

[2. ПРИМЕНА СОФТВЕРСКИХ ПАТЕРНА У РАЗВОЈУ СОФТВЕРСКОГ СИСТЕМ 11](#_Toc95410785)

[2.1. Опис софтверског система 11](#_Toc95410786)

[2.2. Патерни за креирање објеката 12](#_Toc95410787)

[ПК1: Abstract Factory патерн 12](#_Toc95410788)

[ПК2: Builder патерн 15](#_Toc95410789)

[ПК3: Factory method патерн 18](#_Toc95410791)

[ПК4: Prototype патерна 21](#_Toc95410793)

[ПК5: Singleton патерн 24](#_Toc95410794)

[2.3. Структурни патерни 26](#_Toc95410797)

[СП1: Adapter патерн 26](#_Toc95410798)

[СП2: Bridge патерн 29](#_Toc95410800)

[СП3. Composite патерн 32](#_Toc95410801)

[СП4: Decorator патерн 36](#_Toc95410803)

[СП5: Facade патерн 39](#_Toc95410805)

[СП6: Flyweight патерн 42](#_Toc95410806)

[СП7: Proxy патерн 45](#_Toc95410807)

[2.4. Патерни понашања 48](#_Toc95410809)

[ПП1. Chain of responsibility патерн 48](#_Toc95410810)

[ПП2. Command патерн 51](#_Toc95410812)

[ПП3. Interpreter патерн 54](#_Toc95410814)

[ПП4. Memento патерн 57](#_Toc95410815)

[ПП5. Observer патерн 59](#_Toc95410816)

[ПП6. State патерн 62](#_Toc95410818)

[ПП7. Strategy патерн 65](#_Toc95410819)

[ПП8. Template method патерн 68](#_Toc95410821)

[3. ЗАКЉУЧАК 71](#_Toc95410822)

[4. ЛИТЕРАТУРА 72](#_Toc95410823)

УВОД

Овај семинрски рад настао је у циљу бољег разумевања и практичне примене софтверских патерна који се изучавају на истоименом предмету. На почетку рада, дат је теоријски увод у основе софтверских патерна, као и различите дефиниције. У теоријском делу објашњено је шта су софтверски патерни и која је њихова улога у развоју софтверског система. Анализиран је општи облик GOF патерна пројектовања, где је патерн представљен истовремено као структура и као процес. Такође, дати су и општи облици структуре проблема и структуре решења, којима се може описати чак 20 од 23 GOF патерна пројектовања, или неки њихов део.

У овом раду обрађено је 20 од укупно 23 GOF патерна пројектовања и за сваки је дата спецификација као и документација конкретног примера на који је патерн примењен. За сваки патерн дата је дефиниција са појашњењем, одговарајући UML дијаграм класа опште структуре патерна, као и дијаграм класа конкретне примене патерна на примеру софтверског система за пријаве студената за праксу.

# СОФТВЕРСКИ ПАТЕРНИ

## Основне дефиниције патерна

Основе патерна налазе се у радовима америчког архитекте Кристофера Александера (Christopher Alexander). Александер је дао неке од првих дефиниција патерна као појма које није везивао директно за пројектовање софтвера, већ их је дефинисао на општијем нивоу. Његова дефиниција гласи: ***„Сваки патерн је троделно правило, које успоставља релацију између неког проблема, његовог решења и њиховог контекста. Патерн је у исто време и ствар, која се дешава у стварности, и правило које говори када и како се креира наведена ствар“.*** Иако је Александер ову дефиницију извео на основу структура градова и њихових грађевина, односно на основу пројектовања архитектуре у грађевинартву, патерни су веома битни и у пројектовању софтвера.

Бек (Kent Back) и Канингхам започели су 1987. године експерименте са идејом примене патерна у програмирању. Након тога и други значајни аутори у овој области почињу да истражују и развијају софтверске патерне. Софтверски патерни доживљају пуну афирмацију 1994. године са појавом књиге *Design Patterns: Elements of Reusable Object-Oriented Software*, коју су написали Erich Gamma, Richard Helm, Ralph Johnson и John Vlissides. Габриел (Richard Gabriel) је дао дефиницију софтверских патерна која је заснована на дефиницији коју је дао Александер: „***Сваки патерн је троделно правило, које успоставља релацију између неког контекста, неког система сила који се понављају у том контексту (проблем) и софтверске конфигурације која омогућава тим силама да успоставе одговарајуће односе (решење)”.*** Термин контекст указује на неки софтверски систем са дефинисаним ограничењима, док се термин сила користи да укаже на елементе тог софтверског системе и њихове међусобне односе. Термин софтверска конфигурација указује на структуру софтверског система у којој су односи између елемената софтверског система такви да омогућавају поновну употребу те структуре у различитим проблемским ситуацијама.

На могућност поновне употребе патерна указују и Рихл (Dirk Riehle) и Цулиговен (Heinz Zullighoven) који кажу да патерн представља апстракцију конкретног облика која се може поново користити у специфичним контекстима. Коплин (Jim Coplien) за патерн каже Патерн је правило за грађење ствари, али је оно истовремено и сама ствар.

Ако се фокусирамо на улогу патерна у развоју софтвера, можемо рећи да су патерни пројектовања оппште, поново употребљиво, решење неких проблема који се свакодневно јављају у пројектовању софтвера. Они нам доносе доказана и тестирана решења која сигурно могу доста убрзати и олакшати развој софтвера. Међутим, наравно да је сваки софтверски систем је сам по себи специфичан, односно везан за неки домен и проблем који решава. Наш задатак је да уградимо те специфичности у неке од патерна пројектовања, и препознамо који је одговарајући за наш проблем.

Такође, када пројектујемо софтвер јако је битно да узмемо у обзир све могуће проблеме и могуће надоградње које се могу десити у будућности. Неки проблеми који нису видљиви у раној фази развоја могу бити од великог значаја у каснијим фазама. Са великим бројем таквих проблема се неко већ суочио и није му било лако, па се одлучио да помогне будућим генерацијама и тако нешто документује и то у виду софтверских патерна.

Дакле, можемо закључити да патерни који се користе у развоја софтверских система, имају улогу да : олакшају развој нових софтверских система и помогну у одржавању и надоградњу постојећих софтверских система.

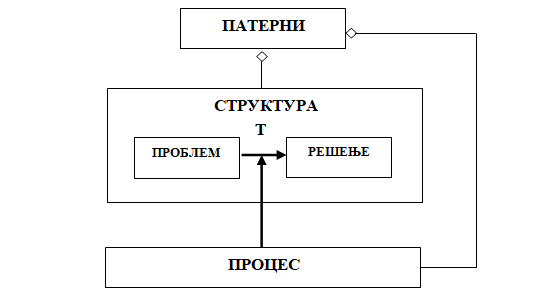
Када се у току имплементације софтверског система пронађе решење проблема, треба препознати шта је у решењу опште, а шта специфично, односно треба идентификовати који су делови програмског кода променљиви, а који непроменљиви. Променљива места се сматрају опасним местима програмског кода с обзиром да се код на тим местима мора мењати приликом имплементације нових корисничких захтева. Наш задатак је да препознамо та места и ту убацимо патерне. Такође, битно је нагласити да се не треба залетати и постављати патерне тек тако, већ треба прво добро размислити. У неким ситуацијама патерни могу беспотребно повећати сложеност система увођењем новог нивоа апстракције.

У сваком систему различитости имају тенденцију да се остваре. Уколико се све различитости (до најситнијих различитости) у некој појави остваре десиће се *„апсолутни хаос”*. Патерни имају механизам који не дозвољава да систем уђе у ово стање. Они 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, тако да се деси *„апсолутни ред”*, јер би тада имали *„тоталитарни”* систем који не прихвата различитости. Не треба заустављати почетак настанка неке различитости јер се тиме спречава и успорава развој система. Хаос и ред се непрекидно смењују и то је нормалан процес у развоју било ког софтверског система.

Патерни држе хаос и ред у непрекидној равнотежи и не дозвољавају да било ко од њих постане апсолутан. Патерни омогућавају да производ реда и хаоса буде увек константан.[С. Влајић – „Софтверски патерни“]

## Општи облик GOF патерна пројектовања

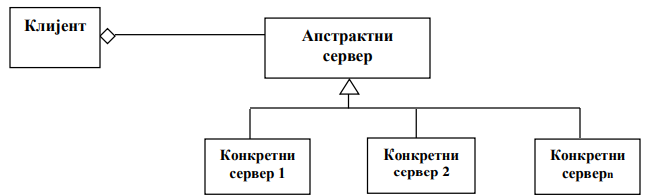
Како бисмо на прави начин схватили општи облик патерна и његов кључни механизам потребно је да извршимо малу анализу постојећих дефиниција патерна. Александер је рекао: *„Патерн је у исто време ствар која се дешава у стварности и правило које говори када и како се креира та ствар“*. Jim Coplien је напоменуо сличну ствар, он каже: *„Патерн је правило за грађење ствари, али је оно истовремено и сама ствар“.* Дакле, можемо приметити да се сваки патерн састоји из два важна дела: **проблем** и **решење**. На основу наведеног може се закључити да је патерн у исто време и структура (ствар) и процес:



Слика 1: Патерни као структура и процес

GOF патерни пројектовања садрже структуру проблема и структуру решења. Патерн такође представља процес који објашњава када и како се трансформацијом креира структура решења из структуре проблема.

Од укупно 23 GOF патерна пројектовања у књизи Design Patterns (креациони, патерни структуре и патерни понашања), 20 патерна или неки њихов део се може описати једном структуром. Структура дата на следећој слици је кључни механизам GOF патерна пројектовања:



Слика 2: СтруктурарешењаGOFпатернапројектовања

Можемо приметити да структуру решења чини уређена тројка (Клијент, Апстрактни сервер, Конкретни сервер), где може бити произвољан број (n) конкретних сервера.

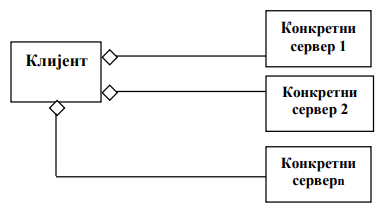
**Клијент** је елемент структуре који користи функционалностиапстрактног и конкретног сервера како би могао да обави сопствену функционалност.

**Апстрактни сервер** је елемент структуре патерна који клијенту даје (обезбеђује) апстрактну функционалност, која се може реализовати са више конкретних функционалности.

**Конкретни сервер** је елемент структуре патерна који клијенту даје конкретну функционалност, која представља реализацију апстрактне функционалности.

Између клијента и апстрактног сервера може бити веза агрегације или зависности. Између апстрактног сервера и конкректног сервера је веза наслеђивања или реализације. Између клијента и конректног сервера не постоји директна веза, већ је она индиректна преко апстрактног сервера. Предност овакве структуре јесте да додавањем новог конкретног сервера не мењамо клијента. Клијент се повезује са конкретним сервером посредно преко апстрактног сервера, и то у време извршења програма, што даје програму флексибилност.

Структуру решења проблема добили смо на основу структуре проблема GOF патерна пројектовања, код које су конкретни сервери директно везани за клијента као што је приказано на слици:



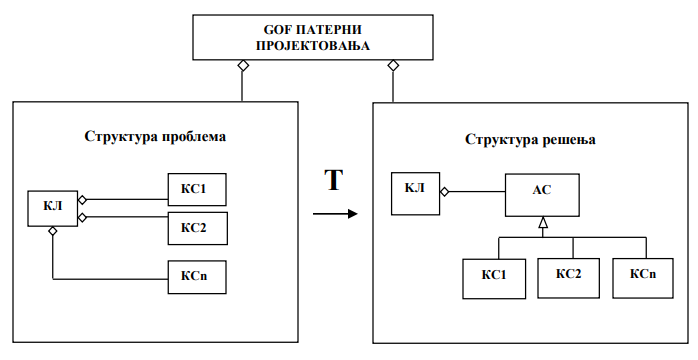
Слика 3: Структура проблемаGOFпатернапројектовања

**Структура проблема** GOF патерна пројектовања је уређена двојка (Клијент, Конкретни сервер), где може бити произвољан број конкретних сервера.

**Клијент** је елемент структуре патерна који користи функционалности конкретног сервера како би могао да обави сопстевену функционалност.

**Конкретни сервер** је елемент структуре патерна који клијенту даје конкретну функционалност. Између клијента и конкретног сервера може бити веза агрегације или зависности . Наведена структура је тешка за одржавање јер се при додавању новог конкретног сервера мења клијент. Клијент се у време компајлирања програма везује за један конкретни сервер, што онемогућава флексибилност програма у току његовог извршавања.

Пошто смо се упознали и са структуром проблема и са структуром решења можемо представити **општи облик GOF патерна пројектовања** на следећи начи:



Слика 4: Општи облик GOF патерна пројектовања

Општи облик GOF патерна садржи: структуру проблема, структуру решења и трансформацију (Т) структуре проблема у структуру решења.

До трансформације Т долази када се структура проблема често мења, са појавом нових корисничких захтева, тј. појавом нових конкретних сервера, који утичу на честу промену клијента. Програм који има структуру проблема, која се често мења, је тежак за одржавање и надоградњу. Односно,таква структура програма је “неодржива”. Са друге стране, код структуре решења, број конкретних сервера не утиче на клијента, што чини овакву структуру “одрживом”. Ово значи да патерни, трансформацијом Т обезбеђује механизам, којим се неодржива структура трансформише у одрживу.

## Примењивост патерна у животу

Поред тога што нам патерни могу помоћи када развијамо неки софтверски систем, њихов значај и примену можемо приметити и у свакодневном животу. Ако уочимо да се неки сличан проблем понавља, логично је да ћемо покушати да пронађемо решење које ћемо примењивати у таквим ситуацијама и које ће нам уштедети време у будућности. Тај пар, проблем и решење нас опет доводи до патерна.

Ситуација која се студентима често дешава јесте узимање потврде о статусу редовног студент која нам је потребна у различите сврхе. Уместо да сваки пут одлазимо у студентску службу и чекамо у дугачком реду, можемо је извадити на почетку године и копирати у неколико примерака, а затим искористити копију (у ситуацијама у којима је то могуће). На овом примеру видимо значај Prototype патерна. Дакле, када креирање новог објекта одузима пуно времена и када већ имамо сличну инстанцу објекта, уместо да креирамо нови комплексни објекат, само ћемо копирати постојећи.

Пример на коме можемо видети примену Аdapter патерна је читач картице који се понаша као адаптер између меморијске картице и лаптопа. Да бисмо меморијску картицу очитали на лаптопу, морамо је убацити у читач који је посредник између ова два ентитета.

Још један патерн који бих издвојила, чију примену видимо на примеру наручивање робе преко интернета је Facade патерн. Уколико често наручујемо робу преко неког сајта, уместо да сваки пут попуњавамо личне податке, податке о картици и адреси испоруке, ове податке можемо сачувати у оквиру сајта и тиме убрзати процес наручивања.

Коришћењем Тemplate method патерна можемо приказати примену патерна у процесу кувања кафе. Сваки процес кувања кафе састоји се из основних делова(припремити воду, сипати кафу, додати додатке), али постоје и специфични делови у зависности од врсте кафе. Закључујемо да помоћу поменутог патерна ми дефинишемо скелет неког процеса, а имплементацију препуштамо подкласи.

Поменути примери су само ситуације у којима сам препознала патерне који су обрађени у овом семинарском раду. Патерни се могу применити у заиста великом броју ситуација у животу. Из тог разлога битно је да када приметимо да нам се понављају слични проблеми, одвојимо мало времена и да анализирамо решења. Од тих решења треба направити једно опште, генеричко решење које ће бити применљиво на било који од проблема те класе.

Када се упознате са патернима и њиховом применом у свакодневном животу, схватате да је решавање проблема без коришћења патерна тежи начин који одузима много времена и углавном проузрокоје много фрустрације и нервирања.

# ПРИМЕНА СОФТВЕРСКИХ ПАТЕРНА У РАЗВОЈУ СОФТВЕРСКОГ СИСТЕМ

## Опис софтверског система

У наредним поглављима биће дат кратак опис 20 од укупно 23 GOF патерна пројектовања са применом сваког на конкретан софтверски систем. Софтверски систем који се обрађује омогућава студентима пријаву за праксу, као и обраду основних информација о пријавама (памћење, измену и брисање). Приликом развоја коришћена је тронивојска архитектура. На екранској форми софтверског система налазе се поља за унос и приказ података. Контролер повезује екранску форму и базу података, односно он прихвата податке и захтеве од екранске форме и прослеђује их брокеру базе података. Брокер базе података обезбеђује памћење, промену, брисање и приказ података пријава за праксу. Решења система дата су коришћењем патерна пројектовања.

## Патерни за креирање објеката

### ПК1: Abstract Factory патерн

**Дефиниција:**

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

**Појашњење дефиниције:**

Обезбеђује интерфејс (*AbstractFactory*) за креирање *(CreateProductA(), CreateProductB())* фамилије повезаних или зависних производа (*AbstractProductA, AbstractProductB*), без навођења њихових конкретних производа *(ProductA1, ProductA2,ProductB1,ProductB2)*.

**Структура Abstract Factory патерна:**

Client

<<interface>> AbstractFactory

CreateProductA()

CreateProductB()

ConcreteFactory1

CreateProductA()

CreateProductB()

ConcreteFactory2

CreateProductA()

CreateProductB()

<<interface>>

AbstractProductA

ProductA2

ProductA1

<<interface>>

AbstractProductB

ProductB2

ProductB1

**Учесници:**

* **Client** - Користи *AbstractFactory* и *AbstractProduct* интерфејсе и из њих изведене класе за креирање сложеног производа.
* **AbstractFactory** - Декларише интерфејс за операције *(CreateProductA(), CreateProductB())* које креирају производе.
* **ConcreteFactory** - Имплементира операције *(CreateProductA(), CreateProductB())* интерфејса *AbstractFactory*, којима се креирају производи *(ProductA1, ProductA2,ProductB1,ProductB2)*.
* **AbstractProduct** - Декларише интерфејс (*AbstractProductA, AbstractProductB*) за производе.
* **ConcreteProduct** - Дефинише производе *(ProductA1, ProductA2, ProductB1, ProductB2)* који ће бити креирани преко *ConcreteFactory* класа. Имплементира операције *AbstractProduct* интерфејса.

**Кориснички захтев *АbstractFactory*:** Шеф развоја софтверског система (за обраду пријаве за праксу) је послао захтев пројектантима софтверског система да креирају елементе софтверског система:

А) Екранску форму (view) са пољима за прихват и приказ података.

Б) Контролер за размену података између екранске форме и базе података.

Ц) Брокер базе података помоћу кога се омогућава памћење, промена, брисање и приказ података.

Након прихватања елемената софтверског система шеф ће креирати и покренути софтверски систем за обраду пријава студената. Шеф надзире процес израде софтверског система.

**Дијаграм класа примера *АbstractFactory*:**

Sef

<<interface>>

Projektant

EkranskaForma kreirajEkranskuFormu()

BrokerBazePodataka kreirajBrokerBazePodataka ()

Kontroler kreirajKontroler (EkranskaForma ef,BrokerBazePodataka dbbr)

B

Panel1

A

A

Panel1

Panel2

Panel1

SoftverskiSistem

EkranskaForma1

<<abstract>>

EkranskaForma

EkranskaForma2

Panel2

<<abstract>>

Panel

Panel1

BrokerBazePodataka2

<<abstract>>

BrokerBazePodataka

BrokerBazePodataka1

Kontroler2

<<abstract>>

Kontroler

Kontroler1

B

DKPrijavaZaPraksu

Projektant2

Projektant3

Projektant4

Projektant1

***Веза између елемената структуре решења Abstract Factory патерна и структуре решења примера AbstractFactory:***

|  |  |
| --- | --- |
| **Елементи AbstractFactory патерна** | **Елементиструктурепримера AbstractFactory** |
| Client | Sef |
| AbstractFactory | Projektant |
| ConcreteFactory | Projektant1,Projektant2,Projektant3,Projektant4 |
| AbstractProductA | EkranskaForma |
| AbstractProductB | BrokerBazePodataka |
| AbstractProductC | Kontroler |
| ProductA | EkranskaForma1,EkranskaForma2 |
| ProductB | BrokerBazePodataka1,BrokerBazePodataka2 |
| ProductC | Kontroler1,Kontroler2 |

### ПК2: Builder патерн

**Дефиниција:**

Дели конструкцију сложеног објекта (производа) од његове репрезентације, тако да исти конструкциони процес може да креира различите репрезентације.

**Појашњење дефиниције:**

Дели одговорност за контролу конструкције (*Director*) сложеног производа од одговорности за реализацију његове репрезентације (*Builder*), тако да исти конструкциони процес (*Direktor.Construct()*) може да креира различите репрезентације (сложене производе).

**Структура Builder патерна:**

## 

<<interface>>

Builder

BuildPart()

ConcreteBuilder

BuildPart()

GetResult()

Product

Director

Construct()

*for all objects in structure*

*{builder.BuildPart()}*

**Учесници:**

* **Director -** Контролише процес конструкције сложеног производа коришћењем *Builder* интерфејса.
* **Builder** - Специфицира интерфејс за креирање сложеног производа.
* **ConcreteBuilder -** Конструише и групише производе у сложени производ имплементирајући Builder интерфејс.
* **Product -** Репрезентује сложени производ који се конструише.

**Кориснички захтев *Builder*:** Шеф развоја софтверског система (за обраду пријаве за праксу) је послао захтев пројектантима софтверског система да креирају софтверски систем и елементе софтверског система:

А) Екранску форму (view) са пољима за прихват и приказ података.

Б) Контролер за размену података између екранске форме и базе података.

Ц) Брокер базе података помоћу кога се омогућава памћење, промена, брисање и приказ података.

Шеф ће надзире процес израде софтверског система.

**Дијаграм класа примера *Builder:***

Sef

<<abstract>>

Projektant

EkranskaForma kreirajEkranskuFormu()

BrokerBazePodataka kreirajBrokerBazePodataka ()

Kontroler kreirajKontroler (EkranskaForma ef,BrokerBazePodataka dbbr)

B

Panel1

A

A

Panel1

Panel2

Panel1

SoftverskiSistem

EkranskaForma1

<<abstract>>

EkranskaForma

EkranskaForma2

Panel2

<<abstract>>

Panel

Panel1

BrokerBazePodataka2

<<abstract>>

BrokerBazePodataka

BrokerBazePodataka1

Kontroler2

<<abstract>>

Kontroler

Kontroler1

B

DKPrijavaZaPraksu

Projektant2

Projektant3

Projektant4

Projektant1

***Веза између елемената структуре решења Builder патерна и структуре решења примера Builder***

|  |  |
| --- | --- |
| **Елементи Builder патерна** | **Елементи структуре примера Builder** |
| Director | Sef |
| Builder | Projektant |
| ConcreteBuilder | Projektant1,Projektant2,Projektant3,Projektant4 |
| AbstractProductA | EkranskaForma |
| AbstractProductB | BrokerBazePodataka |
| AbstractProductC | Kontroler |
| ProductA | EkranskaForma1,EkranskaForma2 |
| ProductB | BrokerBazePodataka1,BrokerBazePodataka2 |
| ProductC | Kontroler1,Kontroler2 |

### ПК3: Factory method патерн

**Дефиниција:**

Дефинише интерфејс за креирање објекта (производа), али преноси на подкласе одлуку коју ће класу истанцирати. Factory method преноси надлежност инстанцирања са класе на подкласе.

**Појашњење дефиниције:**

Дефинише интерфејс *(Creator)* за креирање објекта (производа), али преноси на подкласе *(ConcreteCreator)* одлуку коју ће класу (*ConcreteProduct*) истанцирати. Factory method преноси надлежност инстанцирања са класе *(Creator)* на подкласе*(ConcreteCreator)*.

**Структура Factory method патерна:**

## 

<<abstract >>

Creator

FactoryMethod()

AnOperation()

ConcreteCreator

FactoryMethod()

<<interface>>

Product

ConcreteProduct

*product = FactoryMethod()*

**Учесници:**

* **Creator** - Дефинише интерфејс за креирање производа.
* **ConcreteCreator** - Креира производ (*ConcreteProduct*) помоћу *FactoryMethod* методе.
* **Product** - Дефинише интерфејс за производе који ће бити креирани.
* **ConcreteProduct** - Дефинише производ који ће бити креиран и имплементира *Product* интерфејс.

**Кориснички захтев *FactoryMethod*:** Шеф развоја софтверског система (за обраду пријаве студената) је послао захтев пројектантима софтверског система да надзиру процес израде софтверског система, креирају софтверски систем и елементе софтверског система:

А) Екранску форму (view) са пољима за прихват и приказ података.

Б) Контролер за размену података између екранске форме и базе података.

Ц) Брокер базе података помоћу кога се омогућава памћење, промена, брисање и приказ података.

Шеф неће имати никакве надлежности у процесу израде софтверског система.

**Дијаграм класа примера *FactoryMethod:***

Sef

<<abstract>>

Projektant

EkranskaForma kreirajEkranskuFormu()

BrokerBazePodataka kreirajBrokerBazePodataka ()

Kontroler kreirajKontroler (EkranskaForma ef,BrokerBazePodataka dbbr)

B

Panel1

A

A

Panel1

Panel2

Panel1

SoftverskiSistem1

EkranskaForma1

<<abstract>>

EkranskaForma

EkranskaForma2

Panel2

<<abstract>>

Panel

Panel1

BrokerBazePodataka2

<<abstract>>

BrokerBazePodataka

BrokerBazePodataka1

Kontroler2

<<abstract>>

Kontroler

Kontroler1

B

DKPrijavaZaPraksu

Projektant2

Projektant3

Projektant4

Projektant1

SoftverskiSistem

***Веза између елемената структуре решења Factory Method патерна и структуре решења примера FactoryMethod***

|  |  |
| --- | --- |
| **Елементи Factory Method патерна** | **Елементиструктурепримера FactoryMethod** |
| Creator | Projektant |
| ConcreteCreator | Projektant1,Projektant2,Projektant3,Projektant4 |
| Product | SoftverskiSistem |
| Concrete Product | SoftverskiSistem1 |

### ПК4: Prototype патерна

**Дефиниција:**

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

**Појашњење дефиниције:**

Одређује (специфицира) врсте објеката (*ConcretePrototype1, ConcretePrototype2)*  које ће бити креиране коришћењем прототипског појављивања (*prototype*) и креира нове објекте (*p*) копирањем тог прототипа (*prototype.Clone()*).

**Структура Prototype патерна:**

<<interface >>

Prototype

Clone()

Client

Operation()

p = prototype.Clone()

ConcretePrototype1

Clone()

ConcretePrototype2

Clone()

return copy of self

return copy of self

**Учесници:**

* **Prototype -** Декларише интерфејс за сопствено клонирање.
* **ConcretePrototype -** Имплементира операцију за сопствено клонирање.
* **Client -** Захтева од прототипа да се клонира.

**Кориснички захтев *Prototype*:** Шеф развоја софтверског система (за обраду пријаве за праксу) је послао захтев пројектантима софтверског система да након креирања софтверског система, направе копију тог софтверског система.

Преузет је пример *FactoryMethod* из Factory method патерна.

**Дијаграм класа примера *Prototype:***

Sef

**<<abstract>>**

**Projektant**

**…**

**Public void Kreiraj() { … SoftverskiSistem ss1 = ss.Clone(); …}**

B

Panel1

A

A

Panel1

Panel2

Panel1

**SoftverskiSistem1**

**public SoftverskiSistem Clone() { return new SoftverskiSistem1(this);}**

EkranskaForma1

<<abstract>>

EkranskaForma

EkranskaForma2

Panel2

<<abstract>>

Panel

Panel1

BrokerBazePodataka2

<<abstract>>

BrokerBazePodataka

BrokerBazePodataka1

Kontroler2

<<abstract>>

Kontroler

Kontroler1

B

DKPrijavaZaPraksu

Projektant2

Projektant3

Projektant4

Projektant1

**SoftverskiSistem**

**SoftverskiSistem Clone();**

***Веза између елемената структуре решења Prototype патерна и структуре решења примера Prototype***

|  |  |
| --- | --- |
| **Елементи Prototype патерна** | **Елементи структуре примера Prototype** |
| Client | Sef |
| | Prototype | SoftverskiSistem |
| ConcretePrototype | SoftverskiSistem1 |

### ПК5: Singleton патерн

**Дефиниција:**

Обезбеђује класи само једно појављивање и глобални приступ до ње.

**Структура Singleton патерна:**

return uniqueInstance

Singleton

static Instance()

SingletonOperation()

GetSingletonData()

static uniqueInstance

singletonData

## 

## 

**Учесници:**

* **Singleton** **–** дефинишеInstance()операцију која омогућава клијентима приступ до њеног јединственог појављивања.

**Кориснички захтев *Singleton*:** Онемогућити да се креира више од једног софтверског система.

Овај пример је направљен на основу примера (*AbstractFactory*) Abstract Factory патерна.

**Дијаграм класа примера *Singleton:***

Sef

<<interface>>

Projektant

EkranskaForma kreirajEkranskuFormu()

BrokerBazePodataka kreirajBrokerBazePodataka ()

Kontroler kreirajKontroler (EkranskaForma ef,BrokerBazePodataka dbbr)

B

Panel1

A

A

Panel1

Panel2

Panel1

SoftverskiSistem

EkranskaForma1

<<abstract>>

EkranskaForma

EkranskaForma2

Panel2

<<abstract>>

Panel

Panel1

BrokerBazePodataka2

<<abstract>>

BrokerBazePodataka

BrokerBazePodataka1

Kontroler2

<<abstract>>

Kontroler

Kontroler1

B

DKIspitnaPrijava

Projektant2

Projektant3

Projektant4

Projektant1

**SoftverskiSistemSingleton**

**static Sef.SoftverskiSistem Instance(...) {... return ss;}**

**static Sef.SoftverskiSistem ss;**

***Веза између елемената структуре решења Singleton патерна и структуре решења примера Singleton***

|  |  |
| --- | --- |
| **Елементи Singleton патерна** | **Елементи структуре примера Singleton** |
| Singleton | Sef.SoftverskiSistemSingleton |

## Структурни патерни

### СП1: Adapter патерн

**Дефиниција:**

Конвертује интерфејс неке класе у други интерфејс који клијент очекује. Адаптер патерн омогућава заједнички рад класа које имају некомбатибилне интерфејсе.

**Појашњење дефиниције:**

Конвертује интерфејс неке класе (Adaptee) у други интерфејс (Target) који клијент (Client) очекује. Адаптер патерн омогућава заједнички рад класа (Adaptee, Target) које имају некомбатибилне интерфејсе .

**Структура** Adapter патерна се може јавити у два облика:

**а)** Класа *Adapter* користи вишеструко наслеђивање код прилагођавања некомпатибилних интерфејса.

Client

<<interface>>

Target

Request()

Adapter

Request()

Adaptee

SpecificRequest()

SpecificRequest()

**б)** Класа *Adapter* користи композицију код прилагођавања некомпатибилних интерфејса.

## 

Client

<<interface>>

Target

Request()

Adapter

Request()

Adaptee adaptee

Adaptee

SpecificRequest()

adaptee.SpecificRequest()

**Учесници:**

* **Client** *-* Сарађује са интерфејсом Adaptee преко интерфејса Target.
* **Target *-*** Дефинише доменски-специфичан интерфејс који класа Client користи.
* **Adapter *-*** Адаптира (прилагођава) интерфејс Adaptee интерфејсу Target.
* **Adaptee *-*** Дефинише постојећи интерфејс који треба адаптирати.

**Кориснички захтев *Adapter*:** Шеф развоја софтверског система (за обраду пријаве за праксу) је послао захтев пројектантима софтверског система да креирају софтверски систем и елементе софтверског система:

А) Екранску форму (view) са пољима за прихват и приказ података.

Б) Контролер за размену података између екранске форме и базе података.

Ц) Брокер базе података помоћу кога се омогућава памћење, промена, брисање и приказ података.

Шеф ће надзире процес израде софтверског система.

Пројектанти имају дефинисану апстрактну класу Projektant која има апстрактне методе:

*void kreirajEkranskuFormu()*

*void kreirajBrokerBazePodataka ()*

*void kreirajKontroler ()*

*void kreirajSoftverskiSistem()*

*void prikaziEkranskuFormu()*

Шеф жели да комуницира са пројектантима преко преводиоцa. Преводилац је представљен преко апстрактне Prevodilac која има следеће апстрактне методе:

*void createScreenForm()*

*void createDatabaseBroker()*

*void createController ()*

*void createSoftwareSystem()*

*void showScreenForm()*

Веза између апстрактних метода апстрактне класе Prevodilac и апстрактних метода апстрактне класе Projektant:

|  |  |
| --- | --- |
| *void kreirajEkranskuFormu()* | *void createScreenForm()* |
| *void kreirajBrokerBazePodataka ()* | *void createDatabaseBroker()* |
| *void kreirajKontroler ()* | *void createController ()* |
| *void kreirajSoftverskiSistem()* | *void createSoftwareSystem()* |
| *void prikaziEkranskuFormu()* | *void showScreenForm()* |

**Дијаграм класа:**

**pr.kreirajBrokerBazePodataka()**

**pr.kreirajEkranskuFormu()**

**Sef**

**<<abstract>>**

**Projektant**

**void kreirajEkranskuFormu()**

**void kreirajBrokerBazePodataka ()**

**…**

B

Panel1

A

A

Panel1

Panel2

Panel1

SoftverskiSistem

EkranskaForma1

<<abstract>>

EkranskaForma

EkranskaForma2

Panel2

<<abstract>>

Panel

Panel1

BrokerBazePodataka2

<<abstract>>

BrokerBazePodataka

BrokerBazePodataka1

Kontroler2

<<abstract>>

Kontroler

Kontroler1

B

DKPrijavaZaPraksu

Projektant2

Projektant3

Projektant4

Projektant1

**<<abstract>>**

**Prevodilac**

**void createScreenForm()**

**void createDatabaseBroker()**

**…**

**Prevodilac1**

**void createScreenForm()**

**void createDatabaseBroker()**

**…**

**Projektant pr**

***Веза између елемената структуре решења Adapter патерна и структуре решења примера Adapter***

|  |  |
| --- | --- |
| **Елементи Adapter патерна** | **Елементи структуре примера Adapter** |
| Client | Sef |
| Target | Prevodilac |
| Adapter | Prevodilac1 |
| Adaptee | Projektant |

***Напоменa за пример Adapter:*** Овај пример је заснован на примеру *Builder*

### СП2: Bridge патерн

**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.

**Структура Bridge патерна:**

Implementor

OperationImp()

Abstraction

Operation()

Implementor imp

imp.OperationImp()

ConcreteImplementorA

OperationImp()

ConcreteImplementorB

OperationImp()

Client

RefinedAbstraction

**Учесници:**

* **Abstraction -** Дефинише интерфејс апстракције.Чува референцу на објекат типа *Implementor*.
* **RefinedAbstraction -** Проширује интерфејс *Abstraction*.
* **Implementor -** Дефинише интерфејс за имплементационе класе (*ConcreteImplementorA, ConcreteImplementorB*). Овај интерфејс не мора да одговара интерфејсу *Abstraction* и они могу бити веома различит. Обично *Implementor* интерфејс обезбеђује само примитивне операције док интефејс *Abstraction* дефинише операције високог нивоа које су засноване на наведеним примитивним операцијама.
* **ConcreteImplementor -** Имплементира интерфејс *Implementor*.

**Кориснички захтев *Bridge*:** Шеф развоја софтверског система (за обраду пријаве за праксу) је послао захтев пројектантима софтверског система да креирају елементе софтверског система:

А) Екранску форму (view) са пољима за прихват и приказ података.

Б) Контролер за размену података између екранске форме и базе података.

Ц) Брокер базе података помоћу кога се омогућава памћење, промена, брисање и приказ података. Брокер базе података треба да буде генерички, што значи да он треба да садржи скуп генеричких метода преко којих ће се моћи обрадити подаци различитих доменских класа.

Након прихватања елемената софтверског система шеф ће креирати и покренути софтверски систем за обраду пријава студената. Шеф надзире процес израде софтверског система.

**Дијаграм класа:**

Sef

<<interface>>

Projektant

EkranskaForma kreirajEkranskuFormu()

BrokerBazePodataka kreirajBrokerBazePodataka ()

Kontroler kreirajKontroler (EkranskaForma ef,BrokerBazePodataka dbbr)

B

Panel1

A

A

Panel1

Panel2

Panel1

SoftverskiSistem

EkranskaForma1

<<abstract>>

EkranskaForma

EkranskaForma2

Panel2

<<abstract>>

Panel

Panel1

BrokerBazePodataka2

<<abstract>>

BrokerBazePodataka

**BrokerBazePodataka1**

Kontroler2

<<abstract>>

Kontroler

Kontroler1

B

**Projektant2**

Projektant3

Projektant4

Projektant1

**BrokerBazePodataka11**

**GeneralDObject**

**DKPrijavaZaPraksu**

***Веза између елемената структуре решења Bridge патерна и структуре решења примера Bridge***

|  |  |
| --- | --- |
| **Елементи Bridge патерна** | **Елементи структуре пример Bridge** |
| Abstraction | BrokerBazePodataka1 |
| Refined Abstraction | BrokerBazePodataka11 |
| Implementor | GeneralDObject |
| Concrete Implementor | DKPrijavaZaPraksu |

***Напомеaе за пример Bridge:*** Овај пример је заснован на примеру AbstractFactory

### СП3. Composite патерн

**Дефиниција:**

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

**Појашњење дефиниције:**

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

**Структура Composite патерна:**

## 

<<interface >>

Component

Operation()

Add (Component)

Remove (Component)

GetChild(int)

Client

\*

*forall g in children g.Operation()*

children

Leaf

Operation()

Composition

Operation()

Add (Component)

Remove (Component)

GetChild(int)

Типична структура *Composite* објекта има следећи изглед:

: Composite

: Composite

: Leaf

: Leaf

: Leaf

: Leaf

: Leaf

: Leaf

**Учесници:**

* **Client -** Манипулише објектима (компонентама) у структури помоћу *Component* интерфејса.
* **Component -** Декларише интерфејс за објекте који ће да образују структуру. Декларише интерфејс за приступање и управљање објектима структуре.
* **Leaf** - Представља просте објекте (*Leaf*) у структури и дефинише њихово понашање. Прости објекти немају децу-објекте.
* **Composite** - Дефинише понашање за сложене објекте (*Composition*) који имају децу-објекте. Чува децу-објекте. Имплементира операције интерфејса *Component.*

**Кориснички захтев *Composite*:** За изабраног пројектанта направити и приказати хијерархију класа које чине софтверски систем.

**Дијаграм класа:**

G

D

**Sef**

<<interface>>

Projektant

EkranskaForma kreirajEkranskuFormu()

BrokerBazePodataka kreirajBrokerBazePodataka ()

Kontroler kreirajKontroler (EkranskaForma ef,BrokerBazePodataka dbbr)

B

Panel1

A

A

Panel1

Panel1

SoftverskiSistem

<<abstract>>

EkranskaForma

EkranskaForma2

Panel2

<<abstract>>

Panel

Panel1

BrokerBazePodataka2

<<abstract>>

BrokerBazePodataka

BrokerBazePodataka1

Kontroler2

<<abstract>>

Kontroler

Kontroler1

B

DKPrijavaZaPraksuu

Projektant2

Projektant3

Projektant4

Projektant1

**PanelStablo**

**Komponenta**

**Kompozicija**

**List**

**CEkranskaForma**

**CKontroler**

**CPanel**

**CProjektant**

**CBroker**

**BazPodataka**

C

D

G

C

***Веза између елемената структуре решења Composite патерна и структуре решења примера Composite***

|  |  |
| --- | --- |
| **Елементи Composite патерна** | **Елементи структуре примера Composite** |
| Client | Sef |
| Component | Komponenta |
| Leaf | List, CPanel,CBrokerBazePodataka |
| Composite | Kompozicija, CProjektant, CKontroler, CEkranskaForma |

***Напоменa за пример Composite:*** Овај пример је заснован на примеру AbstractFactory

### СП4: Decorator патерн

**Дефиниција:**

Придружује додатне одговорности (функционалности) до објекта динамички. Decorator патерн обезбеђује флексибилност у избору подкласа које проширују функционалност.

**Појашњење дефиниције:**

Придружује додатне одговорности (функционалности) до објекта (*ConcreteComponent*) динамички. Декоратор обезбеђује флексибилност у избору подкласа *(ConcreteDecoratorA, ConcreteDecoratorB)* које проширују функционалност *ConcreteComponent* објекта.

**Структура Decorator патерна:**

## 

component

Component

Operation()

component.Operation()

ConcreteComponent

Operation()

Decorator

Operation()

ConcreteDecoratorA

Operation()

addedState

ConcretDecoratorB

Operation()

AddedBehavior()

Decorator::Operation()

AddedBehavior()

**Учесници:**

* **Component -** Дефинише интерфејс за *ConcreteComponent* објекте којима се одговорност додаје динамички.
* **ConcreteComponent** - Дефинише објекат коме ће бити додата одговорност динамички.
* **Decorator** - Чува референцу на *Component* објекат.Дефинише интерфејс који је у складу са интерфејсом *Component*.
* **ConcreteDecorator** - Додаје одговорност до *ConcreteComponent* објектa.

**Кориснички захтев *Decorator*:** Потребно је променити постојећи Panel1 коришћењем декоратор патерна. Потребно је при отварању панела Panel1:

а) поставити да дугме Nadji не буде видљиво.

б) поставити фокус програма на поље BrojIndeksa.

Када се кликне дугме Kreiraj, након креирања нове испитне пријаве у бази података и њеног приказивања на панелу Panel1 поставити фокус програма на поље BrojIndeksa.

Када се кликну дугмад Kreiraj, Nadji, Promeni и Obrisi омогућити да 5 секунди након клика буде приказана одговарајућа слика са пратећим звуком. Свако дугме је везано за посебну слику са пратећим звуком.

**Дијаграм класа:**

Sef

<<interface>>

Projektant

EkranskaForma kreirajEkranskuFormu()

BrokerBazePodataka kreirajBrokerBazePodataka ()

Kontroler kreirajKontroler (EkranskaForma ef,BrokerBazePodataka dbbr)

B

Panel1

A

A

Panel1

Panel2

Panel1

SoftverskiSistem

EkranskaForma1

<<abstract>>

EkranskaForma

EkranskaForma2

Panel2

<<abstract>>

Panel

Panel1

BrokerBazePodataka2

<<abstract>>

BrokerBazePodataka

BrokerBazePodataka1

Kontroler2

<<abstract>>

Kontroler

Kontroler1

B

DKPrijavaZaPraksu

Projektant2

Projektant3

Projektant4

Projektant1

Decorator

KonkretniDekoratorKontroleraB

KonkretniDekoratorKontroleraA

***Веза између елемената структуре решења Decorator патерна и структуре решења примера Decorator***

|  |  |
| --- | --- |
| **ЕлементиDecorator патерна** | **ЕлементиструктурепримераSP4\_Decorator** |
| Component | Kontroler |
| Concrete Component | Kontroler1 |
| Decorator | DekoratorKontroler |
| Concrete Decorator | KonkretniDekoratorKontroleraA, KonkretniDekoratorKontroleraB |

***Напоменa за пример Decorator:*** Овај пример је заснован на примеру AbstractFactory

### СП5: Facade патерн

**Дефиниција:**

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

**Појашњење дефиниције:**

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

**Структура Facade патерна:**

Facade

subsystem classes

**Учесници:**

* **Façade -** Зна које класе подсистема (*Sybsystem classes*) су одговорне за послате захтеве од клијента. Преноси одговорност за извршење клијентских захтева до објеката подистема.
* **Sybsystem classes** - Имплементирају подсистемске функционалности. Обрађују захтеве које су добили од *facade* објекта. Не знају ко је *facade* објекат, јер не чувају референцу на њега.

**Кориснички захтев *Facade*:** Омогућити да пословни партнер управља процесом развоја софтверског система. Шеф треба да омогући пословном партнеру све захтеване функционалности.

**Дијаграм класа:**

PoslovniPartner

Sef

<<abstract>>

Projektant

EkranskaForma kreirajEkranskuFormu()

BrokerBazePodataka kreirajBrokerBazePodataka ()

Kontroler kreirajKontroler (EkranskaForma ef,BrokerBazePodataka dbbr)

B

Panel1

A

A

Panel1

Panel2

Panel1

SoftverskiSistem

EkranskaForma1

<<abstract>>

EkranskaForma

EkranskaForma2

Panel2

<<abstract>>

Panel

Panel1

BrokerBazePodataka2

<<abstract>>

BrokerBazePodataka

BrokerBazePodataka1

Kontroler2

<<abstract>>

Kontroler

Kontroler1

B

DKPrijavaZaPraksu

Projektant2

Projektant3

Projektant4

Projektant1

***Веза између елемената структуре решења Facade патерна и структуре решења примера Facade***

|  |  |
| --- | --- |
| **Елементи Facade патерна** | **Елементи структуре примера Facade** |
| Facade | Sef |
| Sybsystem classes | Projektant1,Projektant2,Projektant3,Projektant4 |
| Client | PoslovniPartner |

***Напоменa за пример Facede:*** Овај пример је заснован на примеру Builder

### СП6: Flyweight патерн

**Дефиниција:**

Користи дељење да ефикасно подржи велики број ситних објеката.

**Појашњење ГОФ дефиниције:**

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

**Структура Flyweight патерна:**

\*

flyweights

FlyweightFactory

getFlyweight(key)

<interface>

Flyweight

Operation(extrinsicState)

ConcreteFlyweight

Operation(extrinsicState)

intrinsicState

UnsharedConcreteFlyweight

Operation(extrinsicState)

allState

Client

if (flyweight(key) exists { return existing flyweight;}

else {

create new flyweight;

add it to pool of flyweights;

return the new flyweight;}

**Учесници:**

* **Flyweight –** декларише интерфејс преко кога *Flyweight* објекти могу да прихвате и делују на спољашње стање.
* **ConcreteFlyweight** – имплементира *Flyweight* интерфејс и додаје простор за унутрашње стање (*intrinsicState*) ако постоји. *ConcreteFlyweight* објекат мора бити дељив. Стање које се чува мора бити унутрашње; т.ј., не сме зависити од контекста у коме се налази *ConcreteFlyweight*.
* **UnsharedConcreteFlyweight** –пореддељивих *Flyweight* подкласa постоје и *Flyweight* подкласе које нису дељиве (*UnsharedConcreteFlyweight*). *Flyweight* интерфејс омогућава дељење али га не намеће. Заједничко за недељиве *Flyweight* подкласе је да имају, на неком нивоу, *ConcreteFlyweight* објекте као децу у објектној структури.
* **FlyweightFactory** –креира и управља *Flyweight* објектима. Он омогућава дељивост *Flyweight* објектима. Када клијент захтева *Flyweight* објекат, *FlyweightFactory* објекат враћа постојећи *Flyweight* објекат или креира нови ако исти не постоји.
* **Client** – садржи референце на *Flyweight* објекте. Израчунава или чува спољашња стања *Flyweight* објеката.

**Кориснички захтев *Flyweigt*:** Омогућити преко Flyweigt патерна да се чува лог датотека која ће чувати податке о:

а) називу операције која се извршила,

б) доменском објекту над којим је извршена операција и

ц) поруци о успешности операције.

Дељени подаци ће бити назив операције и порука, док ће недељени податак бити доменски објекат.

**Дијаграм класа:**

\*

Sef

<<interface>>

Projektant

EkranskaForma kreirajEkranskuFormu()

BrokerBazePodataka kreirajBrokerBazePodataka ()

Kontroler kreirajKontroler (EkranskaForma ef,BrokerBazePodataka dbbr)

B

Panel1

A

A

Panel1

Panel2

Panel1

SoftverskiSistem

EkranskaForma1

<<abstract>>

EkranskaForma

EkranskaForma2

Panel2

<<abstract>>

Panel

Panel1

BrokerBazePodataka2

<<abstract>>

BrokerBazePodataka

BrokerBazePodataka1

Kontroler2

<<abstract>>

Kontroler

Kontroler1

B

DKPrijavaZaPraksu

Projektant2

Projektant3

Projektant4

Projektant1

Log

LogDeljeni

LogNedeljeni

LogFactory

***Веза између елемената структуре решења Flyweight патерна и структуре решења примера Flyweight***

|  |  |
| --- | --- |
| **Елементи Flyweight патерна** | **Елементи структуре примера Flyweight** |
| Flyweight | Log |
| ConcreteFlyweight | LogDeljeni |
| UnsharedConcreteFlyweight | LogNedeljeni |
| Flyweight Factory | LogFactory |
| Client | Sef |

***Напоменa за пример Flyweight:*** Овај пример је заснован на примеру AbstractFactory

### СП7: Proxy патерн

**Дефиниција:**

Обезбеђује посредника за приступање другом објектукако би се омогућио контролисани приступ до њега.

**Појашњење** **дефиниције:**

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

**Структура Proxy патерна:**

<<abstract >>

Subject

Request()

Proxy

Request()

realSubject

Client

RealSubject

Request()

*realSubject.Request()*

Приказујемо могући објектни дијаграм proxy структуре у реалном времену.

## 

: Client

subject

: Proxy

realsubject

: RealSubject

**Учесници:**

* **Proxy**
  + Садржи референцу које омогућава *Proxy* објекту приступ до *RealSubject* објекта.
  + 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 *RealSubjec*t и *Proxy* класе тако да се *Proxy* објекат може користити свуда где се очекује RealSubject објекат.
* **RealSubject**
  + Дефинише *RealSubject* објекат који репрезентује *Proxy* објекат.

**Кориснички захтев *Proxy*:** Коришћењем proxy патерна онемогућити операцију брисања испитне пријаве преко Panel1 панела.

**Дијаграм класа:**

Sef

<<interface>>

Projektant

EkranskaForma kreirajEkranskuFormu()

BrokerBazePodataka kreirajBrokerBazePodataka ()

Kontroler kreirajKontroler (EkranskaForma ef,BrokerBazePodataka dbbr)

B

Panel1

A

A

Panel1

Panel2

Panel1

SoftverskiSistem

EkranskaForma1

<<abstract>>

EkranskaForma

EkranskaForma2

Panel2

<<abstract>>

Panel

Panel1

BrokerBazePodataka2

<<abstract>>

BrokerBazePodataka

BrokerBazePodataka1

Kontroler1

<<abstract>>

Kontroler

Kontroler2

B

DKPrijavaZaPraksu

Projektant2

Projektant3

Projektant4

Projektant1

ProxyKontroler

OsluskivacX

***Веза између елемената структуре решења Proxy патерна и структуре решења примера Proxy***

|  |  |
| --- | --- |
| **Елементи Proxy патерна** | **Елементи структуре примера Proxy** |
| Proxy | ProxyKontroler |
| Subject | Kontroler |
| RealSubject | Kontroler1 |
| Client | Sef |

***Напоменa за пример Proxy:*** Овај пример је заснован на примеру AbstractFactory

## Патерни понашања

### 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 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.

**Структура Chain of responsibility патерна:**

## 

successor

<<interface >>

Handler

HandleRequest()

Handler successor

Client

ConcreteHandler1

HandleRequest()

ConcreteHandler2

HandleRequest()

**Учесници:**

* **Handler -** Дефинише интерфејс за обраду захтева. Садржи линк (*successor*) ка следећем *Handler* објекту.
* **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.

**Кориснички захтев *ChainOfResponsibility*:** Софтверски систем треба да омогући проверу:

а) вредносног ограничења над типом атрибута sifraKompanije објектa пријаве за праксу и

б) структурног органичења између објеката пријаве за праксу и компаније. Објекат пријаве за праксу се не може сачувати у бази података ако је нарушен референцијални интегритет:

0,\*

1,1

PrijavaZaPraksu

Kompanija

**Дијаграм класа примера *ChainOfResponsibility:***

Sef

<<interface>>

Projektant

EkranskaForma kreirajEkranskuFormu()

BrokerBazePodataka kreirajBrokerBazePodataka ()

Kontroler kreirajKontroler (EkranskaForma ef,BrokerBazePodataka dbbr)

B

Panel1

A

A

Panel1

Panel2

Panel1

SoftverskiSistem

EkranskaForma1

<<abstract>>

EkranskaForma

EkranskaForma2

Panel2

<<abstract>>

Panel

Panel1

BrokerBazePodataka2

<<abstract>>

BrokerBazePodataka

BrokerBazePodataka1

Kontroler1

<<abstract>>

Kontroler

Kontroler2

B

DKPrijavaZaPraksu

Projektant2

Projektant3

Projektant4

Projektant1

KontrolerVrOg

OsluskivacX

KontrolerStOg

***Веза између елемената структуре решења Chain of responsibility патерна и структуре решења примера ChainOfResponsibility***

|  |  |
| --- | --- |
| **Елементи *Chain of responsibility* патерна** | **Елементи структуре примера *ChainOfResponsibility*** |
| Handler | Kontroler |
| ConcreteHandler | KontrolerVrOg,KontrolerStOg |
| Client | Sef |

***Напоменa за пример ChainOfResponsibility:***Овај пример је заснован на примеру AbstractFactory

### **ПП2. Command патерн**

**Дефиниција:**

Захтев се учаурује као објекат, што омогућава клијентима да параметризују различите захтеве, редове или дневнике захтева, и подржава повратне (undoable) операције чији се ефекат може поништити.

**Појашњење дефиниције:**

Захтев (*Execute()*) се учаурује као објекат (*ConcreteCommand*), што омогућава клијентима (*Client*) да параметризују различите захтеве (*ConcreteCommand.execute(),Receiver.Action()*), редове или дневнике захтева, и подржава повратне (undoable) операције чији се ефекат може поништити.

**Структура Command патерна:**

## 

receiver

<<interface >>

Command

Execute()

ConcreteCommand

Execute()

state

Invoker

Receiver

Action()

*receiver.Action()*

Client

**Учесници:**

* **Invoker** - Позива *ConcreteCommand* објекат да изврши постављени захтев (*Execute()*).
* **Command** - Декларише интерфејс за извршење операције (*Execute()*).
* **ConcreteCommand** - Дефинише везу између *Receiver* објекта и aкције (*Action()*). Имплементира *Execute()* методу позивајући методу *Action() Receiver* објекта.
* **Client** - Креира *ConcreteCommand* објекат и дефинише његов *Receiver* објекат .
* **Receiver** - Извршава методу (*Action()*) која је придружена постављеном захтеву(*Execute()*). Било класа може да буде *Receiver* класа.

**Кориснички захтев *Command*:** Направити пројектанта софтверског система који повезује (помоћу класе Invoker) елементе панела са класама које су на општем нивоу одговорне за извршење системских операција (ConcreteCommand). Омогућити да контролер (Client) софтверског система повеже класе које су одговорне за извршење системских операција на општем нивоу (ConcreteCommand) са класама које су одговорне за извршење системских операција на конкретном нивоу (Receiver).

**Дијаграм класа примера *Command:***

F

G

F

G

A1 B1 C1 D1 E1

A B C D E

D1

E1

C1

B1

A1

D

E

C

B

A

Sef

<<abstract>>

Projektant

EkranskaForma kreirajEkranskuFormu()

BrokerBazePodataka kreirajBrokerBazePodataka ()

Kontroler kreirajKontroler (EkranskaForma ef,BrokerBazePodataka dbbr)

Panel1

SoftverskiSistem

<<abstract>>

EkranskaForma

EkranskaForma2

<<abstract>>

Panel

Panel1

<<abstract>>

BrokerBazePodataka

BrokerBazePodataka1

Kontroler1

<<abstract>>

Kontroler

DKPrijavaZaPraksu

Projektant1

SistemskaOperacija

Invoker1

SOKreiraj

SONadji

SOObrisi

SOPromeni

SOZapamti

KreirajDomenski

Objekat

NadjiDomenski

Objekat

ObrisiDomenski

Objekat

PromenijDomenski

Objekat

ZapamtiDomenski

Objekat

Invoker

SO

***Веза између елемената структуре решења Command патерна и структуре решења примера Command***

|  |  |
| --- | --- |
| **Елементи *Command* патерна** | **Елементи структуре примера *Comman*** |
| Invoker | Invoker,Invoker1 |
| Command | SistemskaOperacija |
| ConcreteCommand | SOKreiraj,SONadji,SOObrisi,SOPromeni,SOZapamti |
| Client | Kontroler1 |
| Receiver | KreirajDomenskiObjekat, NadjiDomenskiObjekat, ObrisiDomenskiObjekat, PromeniDomenskiObjekat, ZapamtiDomenskiObjekat |

### ПП3. Interpreter патерн

**Дефиниција:**

Задати језик, дефинише репрезентацију граматике језика заједно са интерпретером, који користи ту репрезентацију да интерпретира (тумачи) реченице у језику.

**Појашњење дефиниције:**

Задати језик (*Context*), дефинише репрезентацију граматике језика заједно са интерпретером (*AbstractExpression, TerminalExpression, NonterminalExpression*). Interpreter користи ту репрезентацију да интерпретира, помоћу операције *Interpret(Context)* реченице у језику.

**Структура Interpreter патерна:**

\*

<<interface>>

AbstractExpression

Interpret(Context)

NonterminalExpression

Interpret(Context)

Client

Context

TerminalExpression

Interpret(Context)

**Учесници:**

* **Client -** Гради стабло апстрактне синтаксе, репрезентујући (представљајући) реченице језика које су дефинисане граматиком. Стабло апстрактне синтаксе је састављено од *TerminalExpression* и *NonterminalExpression* класа.
* **Context -** Садржи информације које су глобалне за интерпретер.
* **AbstractExpression -** Декларише апстрактну *Interpret(Context)* операцију која је заједничка за све чворове стабла апстрактне синтаксе.
* **TerminalExpression -** Имплементира *Interpret(Context)* операцију која је придружена *TerminalExpression* симболу (објекту) граматике. Један објекат се захтева за сваки *TerminalExpression* симбол у реченици.
* **NonterminalExpression -** Једна класа се захтева за свако правило у граматици. Чува референцу на низ *AbstractExpression* објеката[[1]](#footnote-1). Имплементира *Interpret(Context)* операцију за *NonterminalExpression* симболе (објекте) у граматици.

**Кориснички захтев *Interpreter*:** Пре него што се запамти објекат испитне пријаве проверити да ли је вредност атрибута BrojIndeksa у формату (CCCC-CC), при чему је C ознака за цифру (бројеви од 0 до 9). Прва четири знака формата означавају број индекса и они могу добити вредност између 1 и 1200. Последња два знака формата означавају годину уписа која не може бити већа од 17. Наведени проблем решити коришћењем Interpreter патерна.

**Дијаграм класа примера *Interpreter:***

Projektant2

\*

Sef

<<abstract>>

Projektant

EkranskaForma kreirajEkranskuFormu()

BrokerBazePodataka kreirajBrokerBazePodataka ()

Kontroler kreirajKontroler (EkranskaForma ef,BrokerBazePodataka dbbr)

Panel1

SoftverskiSistem

<<abstract>>

EkranskaForma

EkranskaForma1

<<abstract>>

Panel

Panel2

<<abstract>>

BrokerBazePodataka

BrokerBazePodataka1

Kontroler1

<<abstract>>

Kontroler

DKPrijavaZaPraksu

ApstraktniIzraz

TerminalniIzraz

NeterminalniIzraz

Cifra

Znak

SekBI

SekBIPravilo1

***Веза између елемената структуре решења Interpreter патерна и структуре решења примера Interpreter***

|  |  |
| --- | --- |
| **Елементи *Interpreter* патерна** | **Елементи структуре примерa *Interpreter*** |
| Context | DKPrijavaZaPraksu |
| AbstractExpression | ApstraktniIzraz |
| TerminalExpression | TerminalIzraz |
| NonterminalExpression | NeterminalIzraz |
| Client | Sef, Kontroler1 |

### ПП4. Memento патерн

**Дефиниција:**

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

**Појашњење дефиниције:**

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

**Структура Memento патерна:**

memento

Memento

GetState()

SetState()

state

Originator

setMemento (Memento m)

CreateMemento()

state

Caretaker

*return new Memento(state)*

*state = m.getState()*

**Учесници:**

* **Memento -** Чува интерно стање *Originator* објекта.
* **Originator** - Креира *Memento* објекат који чува његово интерно стање (*Memento.state = Originator.state*).Користи *Memento* објекат да поврати запамћено стање.
* **Caretaker** - Одговоран је за памћења *Memento* објекта. Ништа не ради са садржајем *Memento* објекта.

**Кориснички захтев *Memento*:** Омогући поништење последње операције (undo) која мења објекат помоћу Memento патерна.

**Дијаграм класа примера *Memento:***

<<abstract>>

Projektant

EkranskaForma kreirajEkranskuFormu()

BrokerBazePodataka kreirajBrokerBazePodataka ()

Kontroler kreirajKontroler (EkranskaForma ef,BrokerBazePodataka dbbr)

Sef

DKPrijavaZaPraksu

<<abstract>>

Kontroler

Memento

Originator

Projektant1

OsluskivacX

OsluskivacPromeni

Kontroler1

BrokerBazePodataka1

<<abstract>>

BrokerBazePodataka

Panel1

<<abstract>>

Panel

EkranskaForma2

<<abstract>>

EkranskaForma

SoftverskiSistem

Panel1

***Веза између елемената структуре решења Memento патерна и структуре решења примера Memento***

|  |  |
| --- | --- |
| **Елементи *Memento* патерна** | **Елементи труктуре примера *Memento*** |
| Memento | DKPrijavaZaPraksu |
| Originator | Originator |
| Caretaker | Kontroler1 |

### ПП5. 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 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*).

**Структура Observer патерна:**

## 

observers

\*

subject

<<interface>>

Observer

Update ()

ConcreteObserver

Update()

observerState

ConcreteSubject

GetState()

SetState()

subjectState

*observerState = subject.GetState()*

Subject

Attach(Observer)

Detach(Observer)

Notify()

*for all o in observers {o.Update()}*

*return subjectState*

**Учесници:**

* **Subject -** Зна ко су његови *Observer* објекти. Обезбеђује интерфејс за повезивање и развезивање Оbserver објеката.
* **ConcreteSubject (CS)** - стање на које се постављају *ConcreteObserver* објекти. Шаље обавештење до његових *Observer* објеката када се промени његово стање (*subjectState*).
* **Observer** - Дефинише интерфејс за промену објеката (*Update()*) који треба да буду обавештени када се промени *Subject* објекат.
* **ConcreteObserver (CO)** -Чува референцу на *ConcreteSubject* објекат.Чува стање које треба да остане конзистентно са стањем *ConcreteSubject* објекта. Имплементира *Observer* интерфејс како би сачувао његово стање конзистентно са стањем *ConcreteSubject* објекта.

**Кориснички захтев *Observer*:** Покренути три различите екранске форме за обраду испитне пријаве. Уколико се промени или обрише нека од испитних пријава, преко једне (текуће) од наведених форми, треба обавестити остале форме да се то десило како би освежили податке. Остале форме ће освежити податке уколико обрађују исту пријаву (она треба да буде активна на форми), као што је пријава текуће екранске форме. Захтев урадити помоћу Observer патерна.

**Дијаграм класа примера *Observer:***

Projektant1

\*

Sef

<<abstract>>

Projektant

EkranskaForma kreirajEkranskuFormu()

BrokerBazePodataka kreirajBrokerBazePodataka ()

Kontroler kreirajKontroler (EkranskaForma ef,BrokerBazePodataka dbbr)

Panel1

SoftverskiSistem

<<abstract>>

EkranskaForma

EkranskaForma2

<<abstract>>

Panel

Panel1

<<abstract>>

BrokerBazePodataka

BrokerBazePodataka1

Kontroler1

<<abstract>>

Kontroler

DKPrijavaZaPraksu

OsluskivacNadji1

Subject

Observer

ConcreteObserver

OsluskivacX

***Веза између елемената структуре решења Observer патерна и структуре решења примера Observer***

|  |  |
| --- | --- |
| **Елементи *Observer* патерна** | **Елементи структуре примера *Observer*** |
| Subject | Subject |
| ConcreteSubject | BrokerBazePodataka1 |
| Observer | Observer |
| Concrete Observer | ConcreteObserver |

### ПП6. State патерн

**Дефиниција:**

Допушта објекту да промени понашање када се мења његовo интернo стање.

**Појашњење дефиниције:**

Допушта објекту (*Context*) да промени понашање (*ConcreteStateA, ConcreteStateB*) када се мења његово интерно стање (*state*).

**Структура State патерна:**

Context

Request()

<<interface>>

State

Handle()

state

*state.Handle()*

*...*

ConcreteStateB

Handle()

ConcreteStateA

Handle()

**Учесници:**

* **Context** - Дефинише интерфејс за клијента.Садржи појављивање *State* подкласе која дефинише текуће стање *Context* објекта.
* **State** - Дефинише интерфејс за *Context* објекат, односно понашање које се мења у зависности од промене стања *Context* објекта.
* **ConcreteState** - Свака *ConcreteState* подкласа имплементира одређено понашање у зависности од стања Context објекта.

**Кориснички захтев PST2:** Омогућити да пријава за праксу (уколико постоји) буде у три могућа стања: Необрађен, Обрађен и Сторниран. Уколико је пријава необрађена она се може променити, обрадити и сторнирати. Уколико је пријава обрађена она се не може променити и обрадити али се може сторнирати. Уколико је пријава сторнирана она се не може променити, обрадити и сторнирати. Уколико пријава не постоји, она се налази у Null стању. Null стање онемогућава да се изводе операције промени, обради и сторнирај над непостојећом пријавом. Захтев урадити коришћењем State патерна.

**Дијаграм класа примера PST2**

Projektant1

Sef

<<abstract>>

Projektant

EkranskaForma kreirajEkranskuFormu()

BrokerBazePodataka kreirajBrokerBazePodataka ()

Kontroler kreirajKontroler (EkranskaForma ef,BrokerBazePodataka dbbr)

Panel1

SoftverskiSistem

<<abstract>>

EkranskaForma

EkranskaForma2

<<abstract>>

Panel

Panel1

<<abstract>>

BrokerBazePodataka

BrokerBazePodataka1

Kontroler1

<<abstract>>

Kontroler

DKPrijavaZaPraksu

Stanje

Neobradjen

Obradjen

Storniran

NullStanje

OsluskivacX

***Веза између елемената структуре решења State патерна и структуре решења примера State***

|  |  |
| --- | --- |
| **Елементи *State* патерна** | **Елементи структуре примера *State*** |
| Context | Kontroler1 |
| Stete | Neobradjen, NullStanje, Obradjen, Storniran |
| Concrete State | Stanje |

### ПП7. Strategy патерн

**Дефиниција:**

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

**Појашњење 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 клијенaта (*Context*) који га користи.

**Структура Strategy патерна:**

## 

strategy

<<interface >>

Strategy

AlgorithmInterface ()

Context

ContextInterface()

Strategy strategy

ConcreteStrategyA

AlgorithmInterface ()

ConcreteStrategyB

AlgorithmInterface ()

**Учесници:**

* **Context** - Он садржи референцу на *Strategy* интерфејс.*Context* објекат је конфигурисан са *ConcretyStrategy* објектом. Може да дефинише интерфејс, који омогућава *Strategy* објекту приступ до његових података.
* **Strategy** - Декларише интерфејс који је заједнички за све подржане алгоритме. *Context* објекат користи овај интерфејс да позове алгоритам који је дефинисан преко *ConcreteStrategy* објекта.
* **ConcreteStrategy** - Имплементира алгоритам коришћењем *Strategy* интерфејса.

**Кориснички захтев *Strategy*:** Омогућити различите стратегије при извршењу следећих системских операција: KreirajDomenskiObjekat, NadjiDomenskiObjekat, ObrisiDomenskiObjekat и PromeniDomenskiObjekat. Захтев урадити помоћу Strategy патерна.

**Дијаграм класа примера *Strategy:***

D

C

B

A

A B C D

Projektant1

Sef

<<abstract>>

Projektant

EkranskaForma kreirajEkranskuFormu()

BrokerBazePodataka kreirajBrokerBazePodataka ()

Kontroler kreirajKontroler (EkranskaForma ef,BrokerBazePodataka dbbr)

Panel1

SoftverskiSistem

<<abstract>>

EkranskaForma

EkranskaForma2

<<abstract>>

Panel

Panel1

<<abstract>>

BrokerBazePodataka

BrokerBazePodataka1

Kontroler1

<<abstract>>

Kontroler

DKPrijavaZaPraksu

SistemskaOperacija

NadjiDomenski

Objekat

ObrisiDomenski

Objekat

PromeniDomenski

Objekat

ZapamtiDomenskiObjekat

Dugme

KreirajDomenskiObjekat

TekstPolje

***Веза између елемената структуре решења Strategy патерна и структуре решења примера Strategy***

|  |  |
| --- | --- |
| **Елементи S*trategy* атерна** | **Елементи структуре примера *Strategy*** |
| Context | Dugme |
| Strategy | SistemskaOperacija |
| Concrete Strategy | KreirajDomenskiObjekat, NadjiDomenskiObjekat, ObrisiDomenskiObjekat, PromeniDomenskiObjekat, ZapamtiDomenskiObjekat |

### ПП8. Template method патерн

**Дефиниција:**

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

**Појашњење дефиниције:**

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

**Структура Template method патерна:**

<<abstract >>

AbstractClass

TemplateMethod()

PrimitiveOperation1()

PrimitiveOperation2()

ConcreteClass

PrimitiveOperation1()

PrimitiveOperation2()

*...*

*PrimitiveOperation1()*

*...*

*PrimitiveOperation2*()

...

**Учесници:**

* **AbstractClass -**Дефинише апстрактне примитивне операције (*PrimitiveOperation1(), PrimitiveOperation2()*) које *ConcreteClass* подкласа имплементира. Имплементира *TemplateMethod()* операцију дефинисањем скелета алгоритма. *TemplateMethod()* операција позива примитивне операције (*PrimitiveOperation1(), PrimitiveOperation2()*) које су дефинисане у класи *AbstractClass.*
* **ConcreteClass -** Имплементира примитивне операције које описују специфична понашања подкласа.

**Кориснички захтев *TemplateMethod*:** Направити генеричку (општу) класу у којој се налази метода (TemplateMethod) којa je одговорнa за извршење било које од системских операција на општем нивоу. Такође направити конкретне класе које су одговорне за извршење системских операција на детаљном нивоу. Наведене системске операција обезбеђују извођење основних операција над базом података. Захтев урадити коришћењем Template method патерна.

**Дијаграм класа примера *TemplateMethod:***

Projektant1

A B C D E

D

E

C

B

A

Sef

<<abstract>>

Projektant

EkranskaForma kreirajEkranskuFormu()

BrokerBazePodataka kreirajBrokerBazePodataka ()

Kontroler kreirajKontroler (EkranskaForma ef,BrokerBazePodataka dbbr)

Panel1

SoftverskiSistem

<<abstract>>

EkranskaForma

EkranskaForma2

<<abstract>>

Panel

Panel1

<<abstract>>

BrokerBazePodataka

BrokerBazePodataka1

Kontroler1

<<abstract>>

Kontroler

DKPrijavaZaPraksu

OpstaDBSO

TekstPolje

KreirajDomenskiObjekat

NadjiDomenski

Objekat

ObrisiDomenski

OBjekat

PromeniDomenski

Objekat

ZapamtiDomenskiObjekat

Dugme

***Веза између елемената структуре решења Tamplete Methodпатерна и структуре решења примера TamplateMethod***

|  |  |
| --- | --- |
| **Елементи Tamplate Methodатерна** | **Елементи структуре примера *TamplateMethod*** |
| AbstractClass | OpstaDBSO |
| ConcreteClass | KreirajDomenskiObjekat, NadjiDomenskiObjekat, ObrisiDomenskiObjekat, PromeniDomenskiObjekat, ZapamtiDomenskiObjekat |

# ЗАКЉУЧАК

Након изучавања кључних дефиниција и општих теоријских концепата који се односе на софтверске патерне, за потребе овог рада, успела сам да стечено знање применим у пракси и реализујем 20 показних примера примене GOF патерна пројектовања на примеру софтверског система за пријаве за праксу. Сваки од ових примера урађен је у програмском језику Јава. Примењујући софтверске патерне на свом студијском примеру, схватила сам да они представљају веома важан концепт у пројектовању софтверских система и да ће ми знање стечено на овом предмету много олакшати даље учење и рад.

За примену правог патерна на правом месту потребно је много искуства и потпуно разумевање сваког од патерна. Из тог разлога, мислим да је веома важно што смо на овом предмету научили да уочимо критична места у програму и да, уз помоћ општег облика патерна, самостално дођемо до решења.

Изучавање патерна отворило ми је нове видике и подстакло ме је да на све проблеме гледам мало другачије. Схватила сам да није важно само решити проблем, већ сагледати све околности и спречити његово поновно појављивање. Оваквим приступом штедимо време које бисмо константно трошили на решавање истих проблема који се враћају.

Израда овог семинарског рада је био веома занимљив процес кроз који сам много напредовала и зато се радујем даљем изучавању и коришћењу софтверских патерна.

# ЛИТЕРАТУРА

1. GammaE, HelmR, JohnsonR,VlissidesJ: Design Patterns. Addison-Wesley: ReadingMA, 1995
2. SinišaVlajić,VojislavStanojević,DušanSavić,MilošMilić,IlijaAntović,SašaLazarević, *The General Form of GoF Design Patterns*, The World of Computer Science andInformation Technology Journal (WSCIT), Volume 6, Issue 2. pp. 12-20, 2016, ISSN: 2221-0741.
3. СинишаВлајић:*Софтверскипатерни*,ИздавачЗлатнипресек,ISBN:978-86-86887-30-6,

Београд, 2014.

1. Када кажемо *AbstractExpression* објекат, мислимо на објекте класа које су изведене из интерфејса *AbstractExpression*. [↑](#footnote-ref-1)