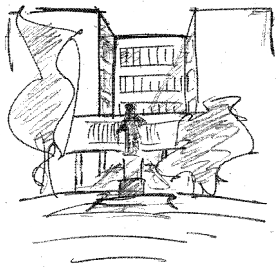
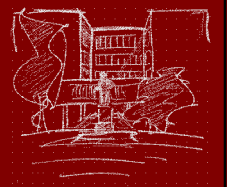
Развој софтвера 4



Саша Малков
Универзитет у Београду
Математички факултет
2023/2024

[P290]
Развој софтвера



Саша Малков

Тема 7

Обрасци за пројектовање

Поновљени проблеми

- Програмери се свакодневно срећу са поновљањем познатих проблема
- Ниво сличности може бити различит

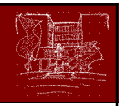
Еквивалентност проблема

- Кажемо да су проблеми *еквивалентни* ако се могу апстраховати истим моделом
  - имплементационо еквиваленти
    - ако је ниво апстракције сасвим низак, толико да се иста имплементација кода може користити за оба проблема
  - тривијално еквивалентни
    - ако је ниво апстракције толико висок да је уопштено решење неупотребљиво
  - концептуално еквивалентни
    - у осталим случајевима

Имплементациона еквивалентност

- Висока поновљива употребљивост кода
  - потпрограми
  - класе
  - библиотеке

Концептуална еквивалентност



- Нижа поновљива употребљивост кода
  - шаблони функција и класа
- Висока поновљива употребљивост концепата
  - алгоритми
  - архитектура и дизајн решења
  - ***“обрасци за пројектовање”***

Шта је образац за пројектовање?

- *"Сваки образац описује проблем који се стално понавља у нашем окружењу и затим описује суштину решења проблема тако да се то решење може употребити милион пута а да се два пута не понови на исти начин"*
  - *Christopher Alexander*
- *Christopher Alexander* је говорио о обрасцима за зидање градова, али то што је рекао важи и за обрасце објектно оријентисаног пројектовања
- Софтверска решења су изражена помоћу објеката и интерфејса уместо зидова и врата али суштина обе врсте образаца је решавање проблема у свом контексту

Термин...

- Енглески термин “***design pattern***”
- Различити преводи термина *pattern*
  - узорак
    - уобичајено, мада не сасвим тачно
  - шаблон
    - није сасвим одговарајуће, посебно не ако *шаблон* користимо за параметарски полиморфизам
  - ***образац***
    - најтачнији превод, можда мало мање уобичајен

Однос са принципима пројектовања

- Обрасци за пројектовање представљају предложена решења за неке проблеме
- Принципи пројектовања представљају апстрактна упутства за пројектовање

- Веома су тесно повезани
  - обрасци су практична примена принципа
  - принципи се илуструју обрасцима и боље се разумеју уз њих

- За курс је примереније да прво упознамо обрасце
- У књизи је обрнуто ;)

Елементи образаца (1)

- Сваки образац има четири основна елемента:
  - Име обрасца
  - Проблем
  - Решење
  - Последице

*детаљни описи образаца садрже и друге елементе, као што ћемо ускоро видети

Елементи образаца (2)

- Сваки образац има четири основна елемента:
  - **Име обрасца**
    - у неколико речи описује проблем, његова решења и његове последице
    - давањем имена обрасцу увећава се речник пројектовања
    - тако се пројектовање подиже на виши ниво апстракције
    - речник образаца омогућава размену мишљења о обрасцима, дискутовање, писање и читање
    - лакше је размишљати о пројектовању и преносити другима тај модел и његове оцене
    - проналажење добрих имена је један од најтежих послова при изради каталога.
  - Проблем
  - Решење
  - Последице

Елементи образаца (3)

- Сваки образац има четири основна елемента:
  - Име обрасца
  - **Проблем**
    - описује случај у коме се образац користи
    - описују се и проблем и његов контекст
    - могућ је опис специфичних проблема (примера)
    - проблем може описивати структуре класа или објеката чије особине наговештавају круто пројектовање
    - понекад проблем садржи списак услова потребних да би се образац применио
  - Решење
  - Последице

Елементи образаца (4)

- Сваки образац има четири основна елемента:
  - Име обрасца
  - Проблем
  - **Решење**
    - описује елементе који чине дизајн, њихове односе, одговорности и сарадњу
    - не описује одређен конкретан пројекат или имплементацију, пошто је образац као шаблон који се може применити у многим различитим ситуацијама
    - даје апстрактан опис проблема пројектовања и упутство како се он решава општим уређењем елемената (класа и објеката).
  - Последице

Елементи образаца (5)

- Сваки образац има четири основна елемента:
  - Име обрасца
  - Проблем
  - Решење
  - **Последице**
    - обухватају резултате и оцене примене обрасца
    - често се не помињу у описима одлука о пројектовању, али су веома битне за процену алтернатива и за разумевање предности и недостатака примене обрасца

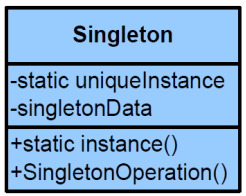
Примери

- ...

Образац Уникат (*Singleton*)

- Намена
  - Обезбеђивање да класа сме да има највише једну инстанцу
  - Пружа интерфејс до те јединствене инстанце
  - Енкапсулира њено прављење
- Решава проблем
  - Прављења јединствених инстанци
  - Редоследа прављења јединственеих инстанци
    - праве се онда када су потребне
  - Аутоматског брисања јединствених инстанци (у случају *C++*-а)
- Пример
  - Сви случајеви у којима је потребан један јединствени дељени ресурс
    - кеш
    - драјвер
    - централни каталог
    - ...

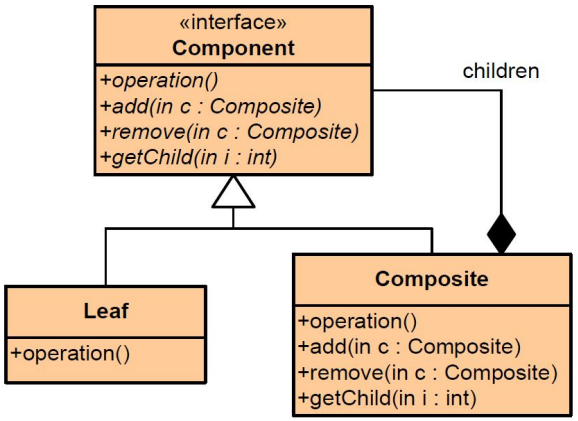
Образац Уникат (*Singleton*)



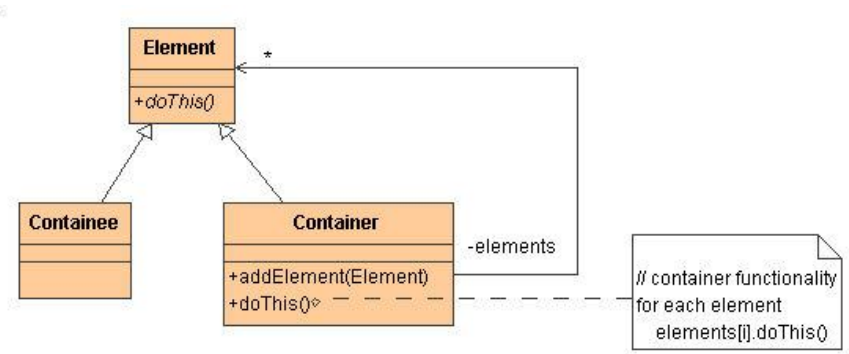
Образац Састав (*Composite*)

- Намена
  - Повезивање више објеката исте хијерархије у један сложени објекат
  - Стара се о њиховом прављењу и брисању
- Решава проблем
  - Када је потребно да у хијерархији класа постоји класа која представља сложени објекат, који је композиција више објеката исте класе
  - Често се комбинује са обрасцем Посетилац
- Пример
  - Сложен лик који се састоји од више једноставнијих ликова

Образац Састав (*Composite*)



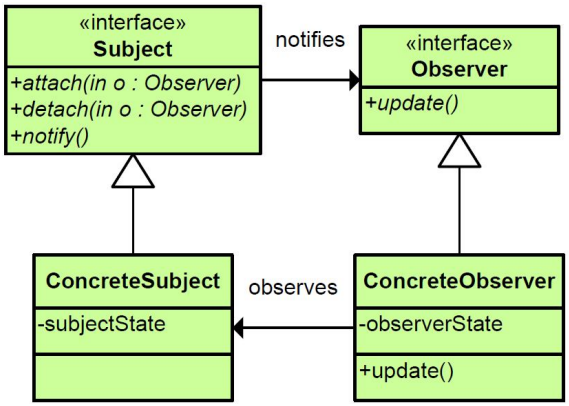
Образац Састав (*Composite*) – Пример



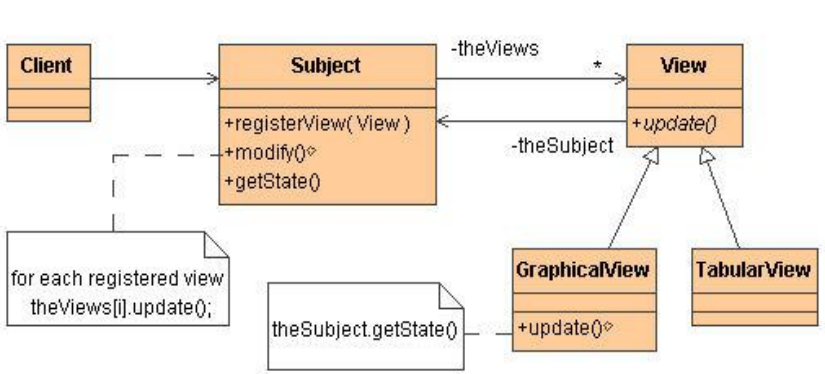
Образац Посматрач (*Observer*)

- Намена
  - Повезивање више *посматрача* са једним *посматраним* објектом
- Решава проблем
  - Дистрибуирање информација о ажурирању свим објектима којима је то потребно
- Пример
  - Раздвајање података и репрезенације
  - Једна колекција података која има више различитих визуалних репрезентација

Образац Посматрач (*Observer*)



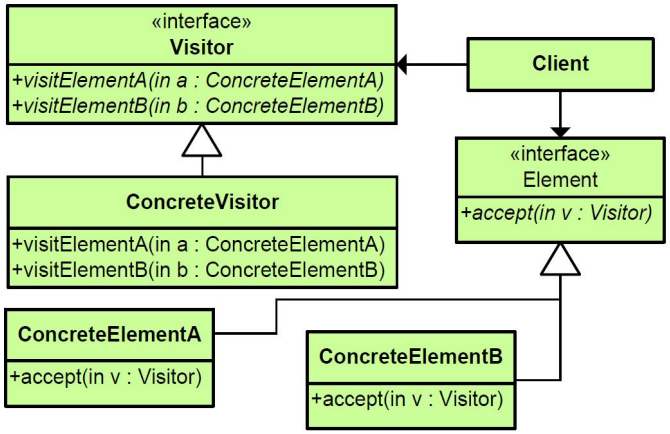
Образац Посматрач (*Observer*) – Пример

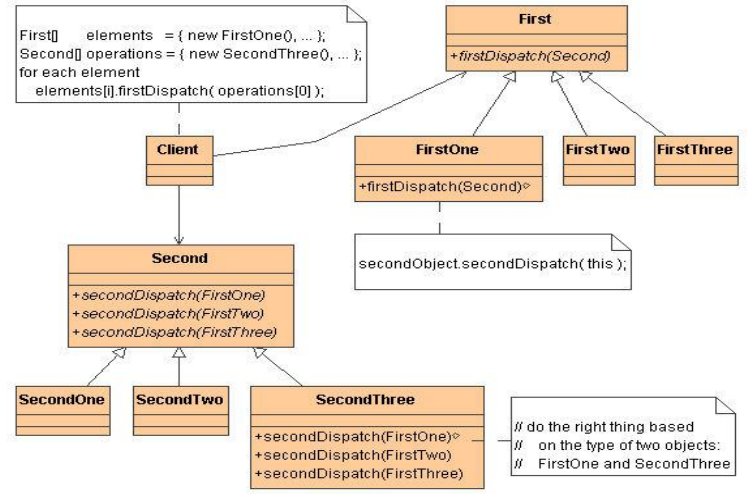


Образац Посетилац (*Visitor*)

- Намена
  - Раздвајање операције које је потребно да се извршава на елементима неке структуре од имплементације самих елемената структуре
- Решава проблем
  - Омогућава додавање и мењање операција без мењања имплементације структуре
  - Одговорности извођења операције су на посетиоцу, а (универзална) одговорност обилажења структуре је на структури
- Пример
  - Рачунање површине, обима и сл. за сложене ликове
  - Извођење операција на графовима и хијерархијама објеката

Образац Посетилац (*Visitor*)





Описивање образаца (1)

- **Име обрасца и класификација**
  - Име обрасца сажето излаже суштину обрасца.
  - Добро име је од суштинског значаја пошто оно улази у речник пројектовања.
  - Класификација обрасца изведена је по шеми која ће бити касније наведена.
- **Намена**
  - Кратак исказ који одговара на следећа питања:
    - Шта образац за пројектовање ради?
    - Какво му је образложење и намена?
    - На које конкретно питање или проблем пројектовања се образац односи?
- **Познат такође као**
  - Остала позната имена обрасца, ако постоје.

Описивање образаца (2)

- **Мотивација**
  - Сценарио који илуструје проблем пројектовања и начин на који структуре класа и објеката у обрасцу решавају тај проблем.
  - Сценарио помаже да се схвати апстрактнији опис обрасца.
- **Примењивост**
  - На које ситуације се образац може применити?
  - Који су примери лошег пројектовања које образац може да исправи?
  - Како препознати те ситуације?
- **Структура**
  - Графички приказ класа које чине образац у нотацији *UML*
    - дијаграми класа
    - дијаграми интеракције

Описивање образаца (3)

- **Учесници**
  - Класе и/или објекти који учествују у обрасцу за пројектовање као и њихове одговорности.
- **Сарадња**
  - Како учесници сарађују да би извршили своје одговорности.
- **Последице**
  - Како образац задовољава своју намену?
  - Који су недостаци и користи од коришћења обрасца?
  - Који аспект структуре система може независно да се мења?

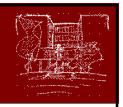
Описивање образаца (4)

- **Имплементација**
  - Којих замки, савета или техника би требало да будете свесни приликом имплементирања обрасца?
  - Има ли неких питања зависних од програмског језика?
- **Пример кода**
  - Фрагменти кода који илуструју како би се образац могао имплементирати.
- **Позната коришћења**
  - Примери обрасца који се налазе у стварним системима.
- **Повезани обрасци**
  - Који обрасци за пројектовање су у тесној вези са овим обрасцем?
  - Које су значајне разлике?
  - Уз које друге обрасце би требало користити овај образац?

Преглед образаца

- У наставку ћемо представити преглед образаца за пројектовање, пратећи књигу
- Нећемо се посветити свим обрасцима, али су укратко наведени ради комплетности

Врсте образаца



- **Градивни обрасци** (*Creational Patterns*)
- **Структурни обрасци** (*Structural Patterns*)
- **Обрасци понашања** (*Behavioral Patterns*)

Градивни обрасци (1)

- Апстрахују процес прављења објеката
  - Помоћу њих систем постаје независан од начина прављења, састављања и представљања објеката
- Разликују се по домену
  - Градивни обрасци са доменом класе
    - користе наслеђивање за мењање класе која се инстанцира
  - Градивни обрасци са доменом објекта
    - делегирају прављење неком другом објекту

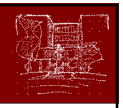
Градивни обрасци (2)

- Градивни обрасци су важни када системи више зависе од састављања објеката него од наслеђивања
  - нагласак се са фиксног кодирања фиксног скупа понашања пребацује на дефинисање мањег скупа основних понашања која се могу састављати у већи број сложенијих
  - прављење објеката са неким одређеним понашањем захтева више од пуког инстанцирања класе
- У овим обрасцима се стално понављају две теме:
  - сви ови обрасци енкапсулирају знање о томе које конкретне класе систем користи
  - крију како се праве примерци ових класа и како се састављају

Градивни обрасци (3)

- У целом систему се о објектима зна једино за њихове интерфејсе, на основу тога како су дефинисани у апстрактним класама
- Градивни обрасци пружају велику флексибилност у смислу
  - *шта* се прави,
  - *ко* то прави,
  - *како* се прави и
  - *када*.
- Омогућавају да се конфигурише систем објектима "производима" разноврсне структуре и функција.
- Конфигурисање може да буде статично (тј. одређено у време превођења) или динамичко (у време извршавања)

Градивни обрасци (4)

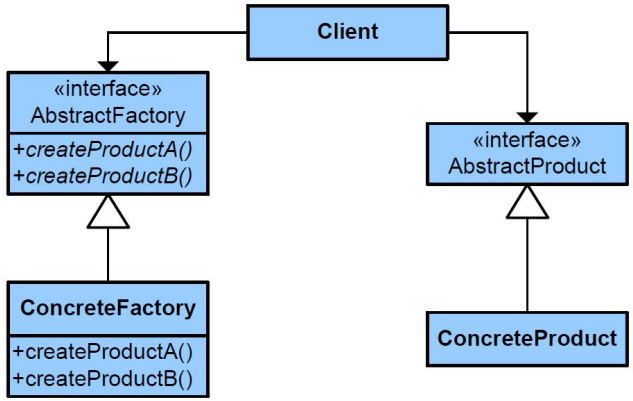


- Најважнији градивни обрасци:
  - Апстрактна фабрика (*Abstract Factory*)
  - Градитељ (*Builder*)
  - Производни метод (*Factory Method*)
  - Прототип (*Prototype*)
  - Уникат (*Singleton*)

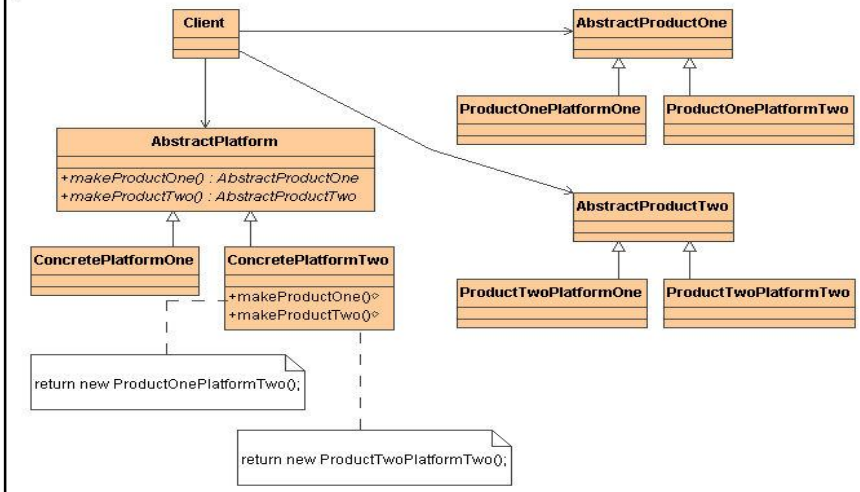
Образац Апстрактна фабрика (*Abstract Factory*)

- Намена
  - Служи за прављење фамилија сличних објеката чији избор може да зависи од околности, платформе и сл.
  - Користи се уместо оператора *new* за прављење објеката
- Решава проблем
  - Преносивости програма
    - различите верзије фабрике праве различите врсте објеката, зависно од платформе или других околности
- Пример
  - Прављење елемената корисничког интерфејса за различите оперативне системе или визуална окружења

Образац Апстрактна фабрика (*Abstract Factory*)



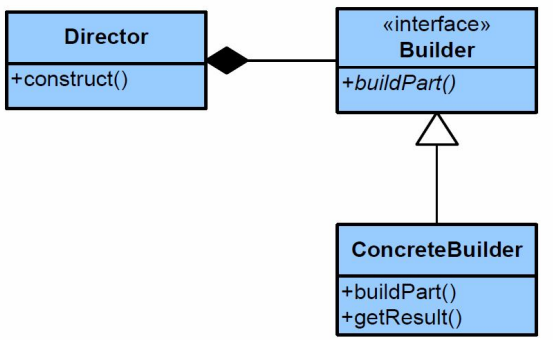
Образац Апстрактна фабрика (*Abstract Factory*) – Пример

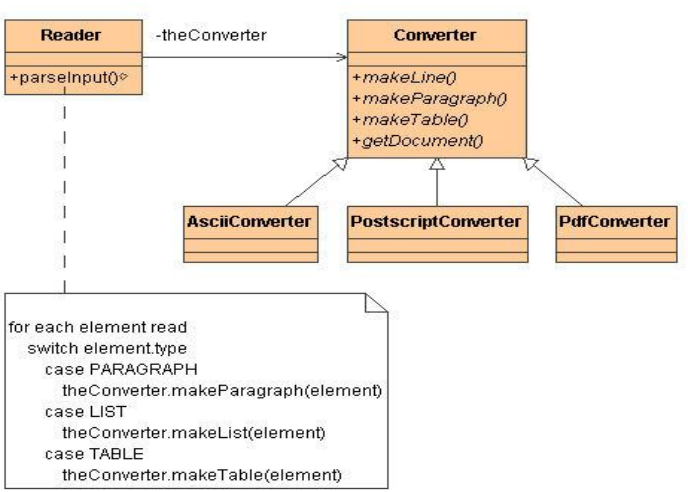


Образац Градитељ (*Builder*)

- Намена
  - Раздваја конструкцију сложених објеката од конструкције њихових делова (тј. од његове репрезентације)
  - Омогућава прављење различитих репрезентација за различите потребе
- Решава проблем
  - Прављење различитих сложених објеката са сличном апстрактном структуром а различитом имплементацијом
- Пример
  - Читање докумената записаних у различитим форматима

Образац Градитељ (*Builder*)

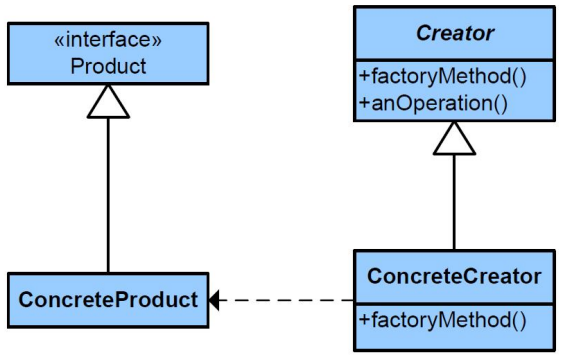




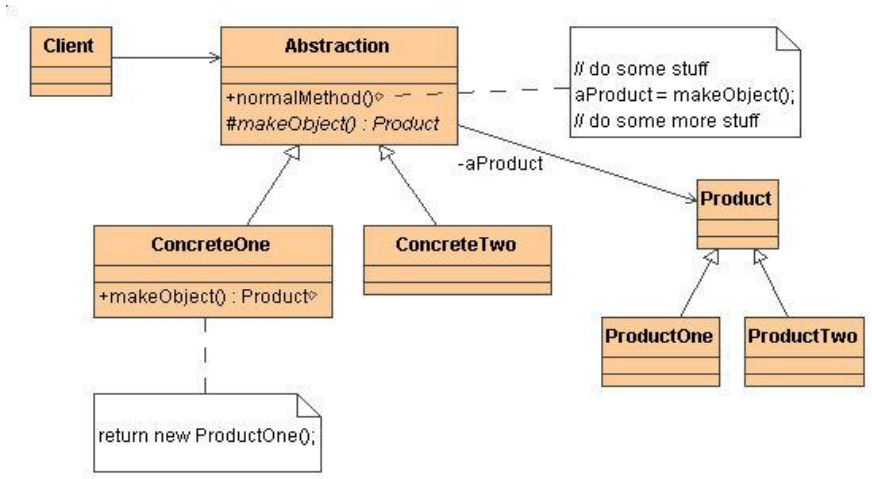
Образац Производни метод (*Factory Method*)

- Намена
  - Дефинише интерфејс за прављење објеката, али препушта потклasama да одлуче које ће објекте да направе
  - Користи се уместо оператора *new* за прављење објеката
- Решава проблем
  - Прављења различитих објеката у различитим околностима
  - Апстрактна фабрика је посвећена само прављењу објеката, а овде се у хијерархији класа која има други фокус прави један метод за прављење објеката
    - производни метод може да делегира прављење фабрици ако је потребно
- Пример
  - На пример, апстраховани геометријски лик може да прави одговарајући објекат који се црта на екрану

Образац Производни метод (*Factory Method*)



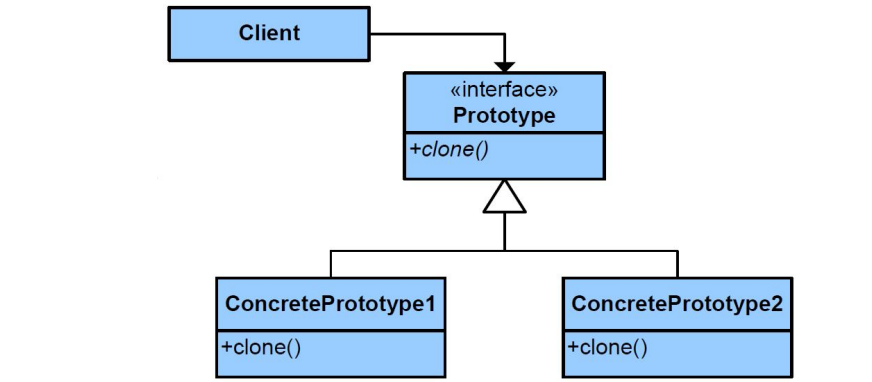
Образац Производни метод (*Factory Method*) – Пример



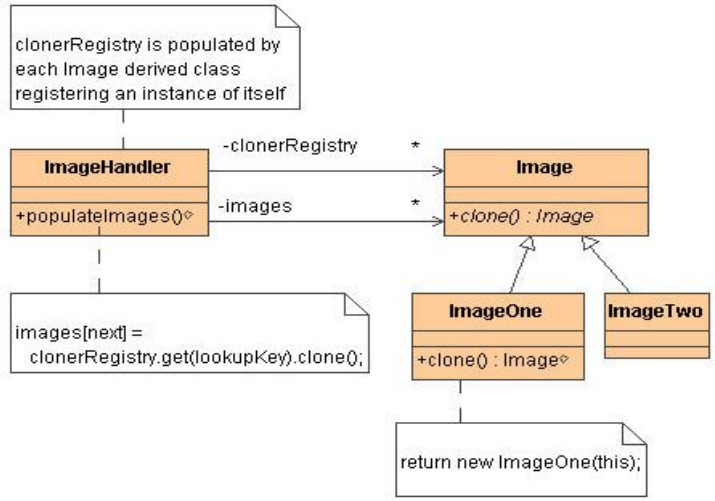
Образац Прототип (*Prototype*)

- Намена
  - Прављење нових објеката као копија задатих прототипова
  - Користи се уместо оператора *new* за прављење објеката
  - Може да се користи и у комбинацији са обрасцем Мува
- Решава проблем
  - Објекти који се праве могу бити инстанце различитих класа хијерархије, али и различите инстанце истих класа
  - Изабрани прототипови обично имају неке *канонске* облике који омогућавају накнадно прилагођавање
  - Алтернатива апстрактној фабрици, омогућава динамичко прављење ка
- Пример
  - Типични елементи корисничког интерфејса могу да се додају у каталог при иницијализацији

Образац Прототип (*Prototype*)



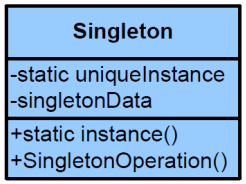
Образац Прототип (*Prototype*) – Пример



Образац Уникат (*Singleton*)

- Намена
  - Обезбеђивање да класа сме да има највише једну инстанцу
  - Пружа интерфејс до те јединствене инстанце
  - Енкапсулира њено прављење
- Решава проблем
  - Прављења јединствених инстанци
  - Редоследа прављења јединствнеих инстанци
    - праве се онда када су потребне
  - Аутоматског брисања јединствених инстанци (у случају *C++*-а)
- Пример
  - Сви случајеви у којима је потребан један јединствени дељени ресурс
    - кеш
    - драјвер
    - ...

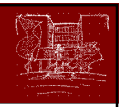
Образац Уникат (*Singleton*)



Структурни обрасци

- Баве се начином на који се класе и објекти састављају у веће структуре
- Деле се на
  - Структурне обрасце са доменом *класа* и
  - Структурне обрасце са доменом *објеката*

Структурни обрасци са доменом класа



- Структурни обрсци са доменом *класа* користе наслеђивање за састављање интерфејса или имплементација
  - Једноставан пример је вишеструко наслеђивање којим се две или више класа комбинују у једну
    - Резултат је класа у којој су комбинована својства родитељских класа
    - Овај образац је посебно користан за комбиновано коришћење независно развијених библиотека класа
  - Други пример је класни облик обрасца Адаптер
    - Адаптер прилагођава један интерфејс другом интерфејсу тако што прави уједначену апстракцију различитих интерфејса
    - Класни адаптер то постиже приватним наслеђивањем класе коју прилагођава
    - Адаптер затим изражава свој интерфејс појмовима класе коју прилагођава

Структурни обрасци са доменом објеката

- Структурни обрасци са доменом *објеката* описују начине за постизање нове функционалности комбиновањем објеката уместо састављањем интерфејса или имплементација
  - Додатна флексибилност састављања објеката потиче од могућности да се састав мења у време извршавања што је немогуће код статичког састављања класа

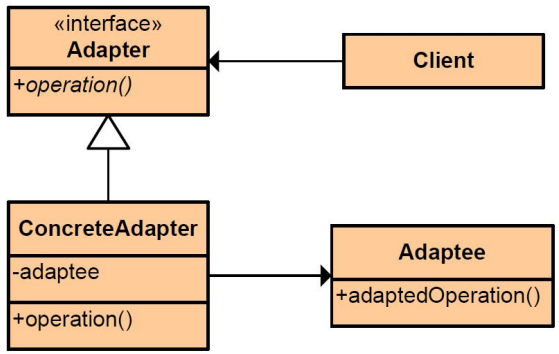
Структурни обрасци

- Најважнији структурни обрасци су:
  - Адаптер (*Adapter*)
  - Мост (*Bridge*)
  - Састав (*Composite*)
  - Декоратер (*Decorator*)
  - Фасада (*Facade*)
  - Мува (*Flyweight*)
  - Прокси (*Proxy*)

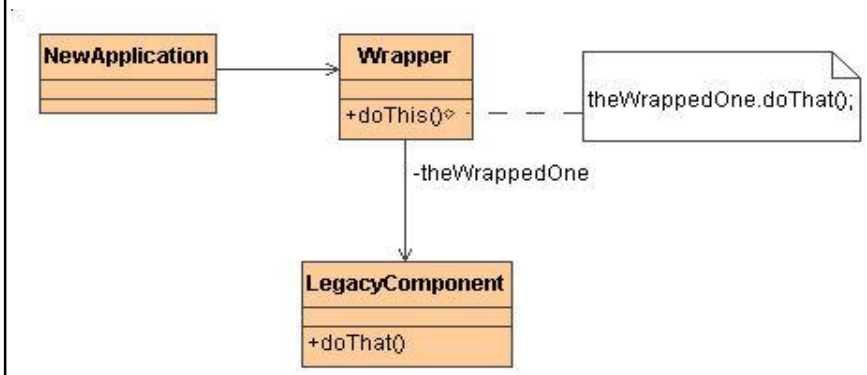
Образац Адаптер (*Adapter*)

- Намена
  - Прилагођава интерфејс објекта другачијим потребама клијента
  - Енкапсулира објекат
  - Назива се и Омотач (*Wrapper*)
- Решава проблем
  - Када постоји класа која ради оно што нам је потребно али са другачијим интерфејсом, омогућава њену употребу на начин који нам одговара
- Пример
  - Постоји класа која ради са подацима у империјалном систему мера а потребно нам је да ради у метричком систему мера

Образац Адаптер (*Adapter*)



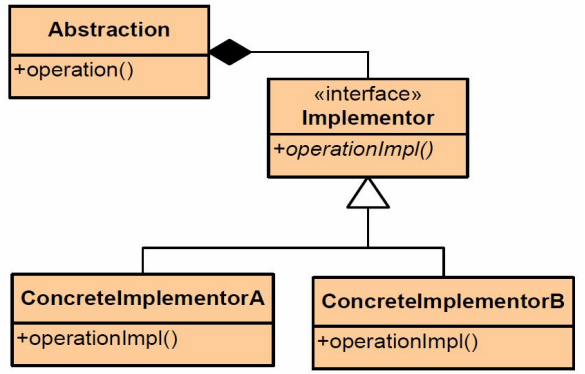
Образац Адаптер (*Adapter*) – Пример



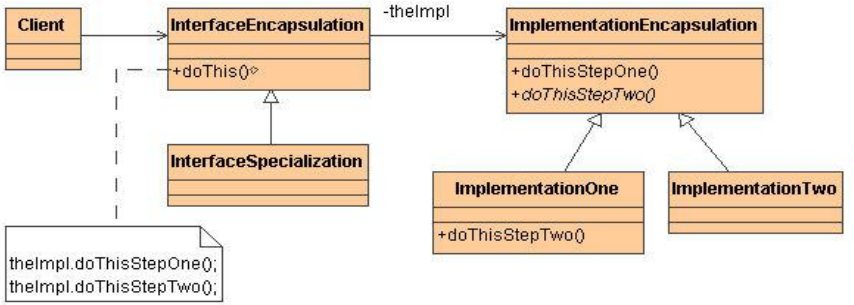
Образац Мост (*Bridge*)

- Намена
  - Раздвајање одговорности апстракције и имплементације, тако да могу да се одвојено мењају
  - Мост се прави као енкапсулација објекта који је имплементација
- Решава проблем
  - Када за један посао може да постоји више различитих имплементација, а избор имплементације желимо да мењамо динамички
- Пример
  - Постоји класа која ради са подацима у империјалном систему мера а потребно нам је да ради у метричком систему мера
  - Мост не мора ни да буде свестан сложености понашања имплементације ни онога што клијент од ње очекује, на њему је да на апстрактан начин успостави везу клијента и имплементације

Образац Мост (*Bridge*)



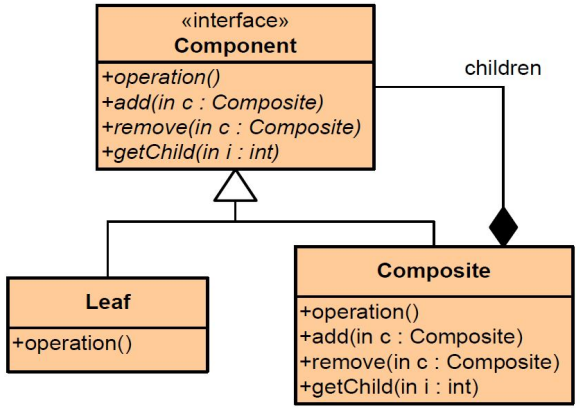
Образац Мост (*Bridge*) – Пример



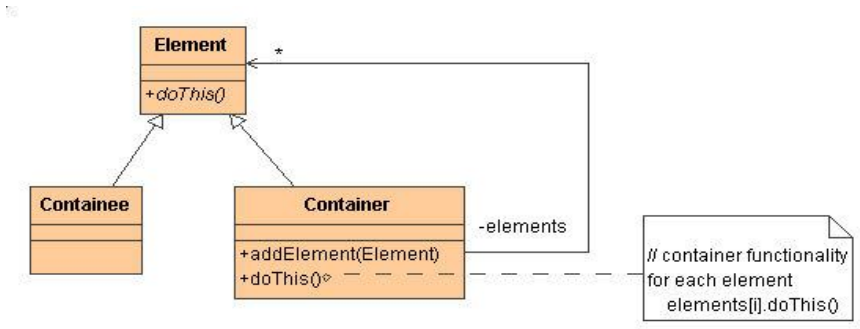
Образац Састав (*Composite*)

- Намена
  - Повезивање више објеката исте хијерархије у један сложени објекат
  - Стара се о њиховом прављењу и брисању
- Решава проблем
  - Када је потребно да у хијерархији класа постоји класа која представља сложени објекат, који је композиција више објеката исте класе
  - Често се комбинује са обрасцем Посетилац
- Пример
  - Сложен лик који се састоји од више једноставнијих ликова

Образац Састав (*Composite*)



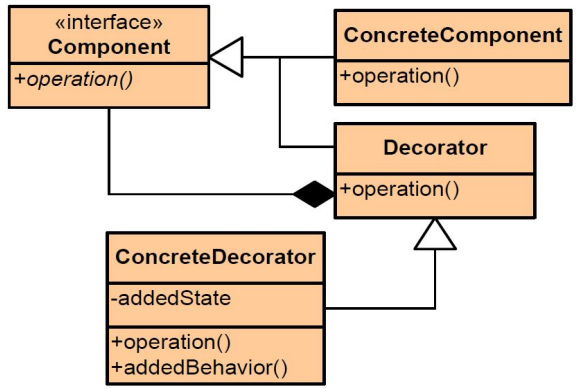
Образац Састав (*Composite*) – Пример



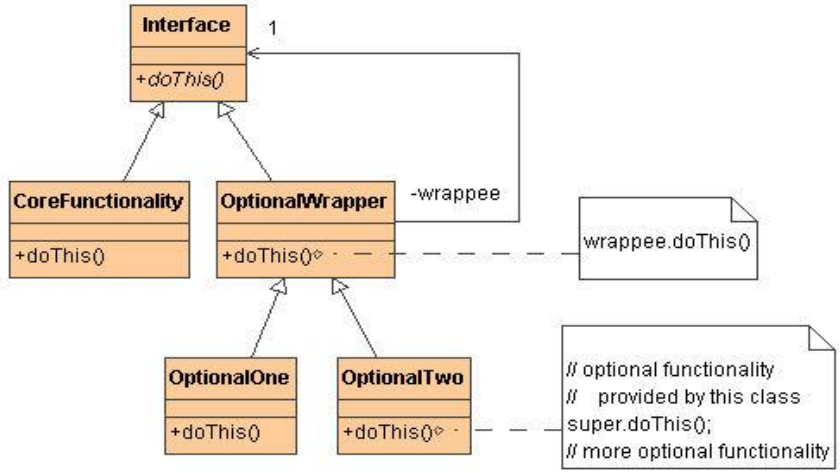
Образац Декоратер (*Decorator*)

- Намена
  - Додавање одговорности и карактеристика објекта динамички
  - Декорација може да само обави део посла који уме, а остало препушта објекту који декорише
- Решава проблем
  - Експлозије величине хијерархије ако се моделира велики број ортогоналних опционих карактеристика
    - Уместо да се праве разлиите класе хијерархије за све комбинације опционих карактеристика, оне се имплементирају као декорације
- Пример
  - Уместо свих врста прозора са менијем, скролом, дугмићима и сл., сви ти додаци се праве као декорације

Образац Декоратер (*Decorator*)



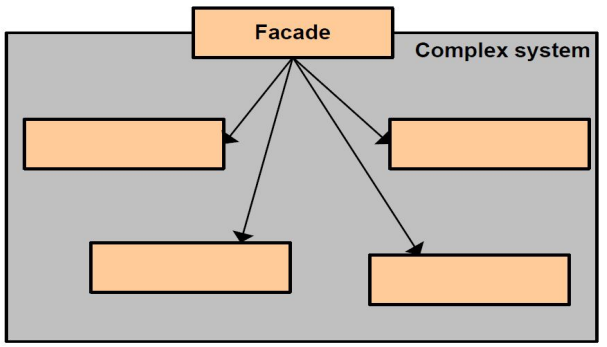
Образац Декоратер (*Decorator*) – Пример



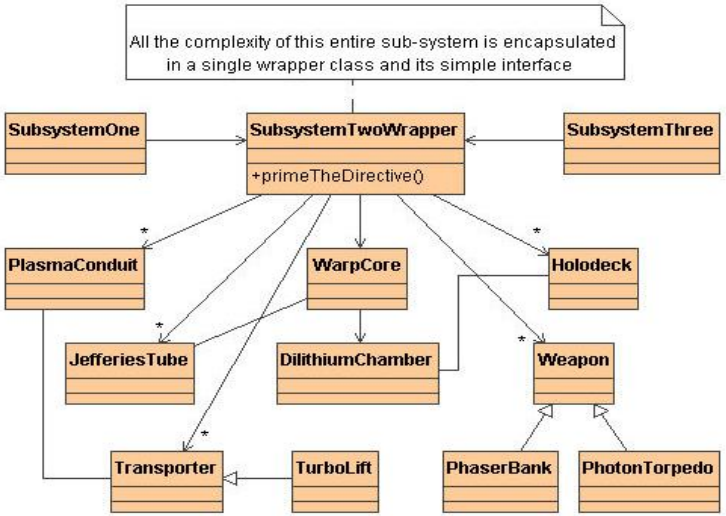
Образац Фасада (*Facade*)

- Намена
  - Представља интерфејс подсистема
  - Омогућава енкапсулирање осталих класа подсистема
- Решава проблем
  - Употребе сложених библиотека или компоненти
    - Сва функционалност се постиже кроз фасаду
- Пример
  - Типична примена је за интерфејс компоненти

Образац Фасада (*Facade*)



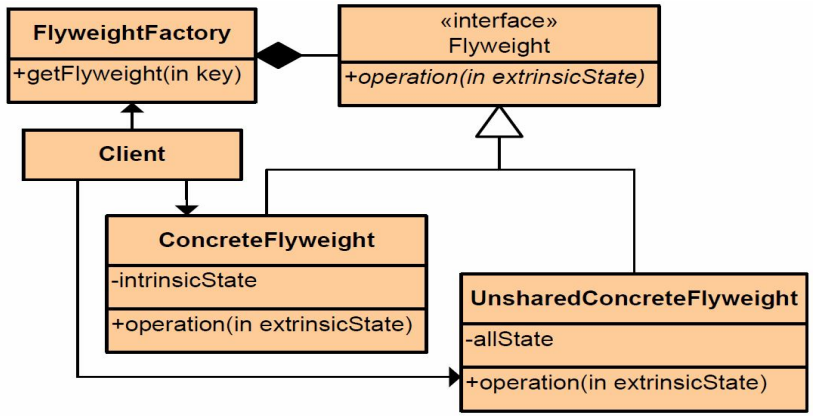
Образац Фасада (*Facade*) – Пример



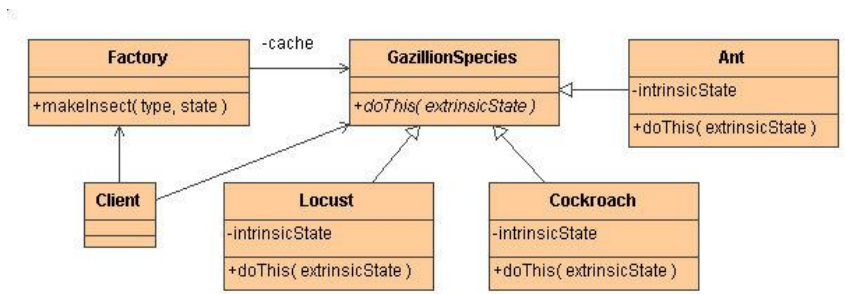
Образац Мува (*Flyweight*)

- Намена
  - Примењује дељење објеката ради остваривања бољих перформанси
- Решава проблем
  - Користи се уместо прављења великог броја идентичних инстанци неких објеката
- Пример
  - Имплементација слова у процесору текста
  - Имплементација честих објеката у видео играма

Образац Мува (*Flyweight*)



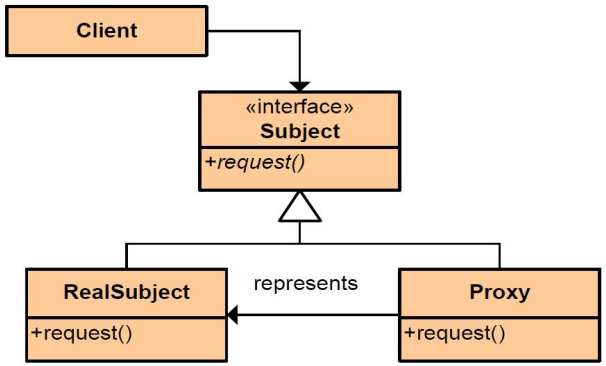
Образац Мува (*Flyweight*) – Пример

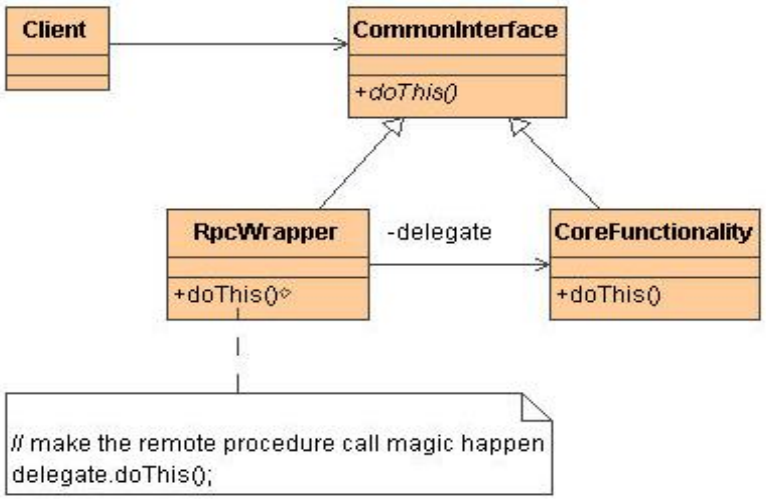


Образац Прокси (*Proxy*)

- Намена
  - Увођење додатне апстракције између клијента и имплементације
  - Структура је слична адаптеру, али понашање није – адаптер имплементира другачији интерфејс а прокси имплементира *исти интерфејс*
- Решава проблем
  - Када је потребно динамички успостављати однос клијента са имплементацијом
  - Може да има сличности са декоратером
- Пример
  - Када од клијента желимо да сакријемо сложеност имплементације
    - било да је у питању сложеност прављења или одржавања
    - не и сложености интерфејса, јер очекујемо да је исти, или бар исти као онај део који нам је потребан

Образац Прокси (*Proxy*)





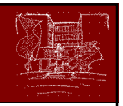
Обрасци понашања (1)

- Баве се алгоритмима и расподелом одговорности међу објектима
  - Не описују само обрасце структуре објеката или класа већ и обрасце њихове међусобне комуникације
  - Описују природу сложеног тока контроле који се тешко прати у време извршавања
  - Пажња прелази са самог тока контроле на начин међусобног повезивања објеката
- Деле се на
  - класне обрасце понашања и
  - објектне обрасце понашања

Класни обрасци понашања

- Користе наслеђивање за дистрибуирање понашања међу класама
  - Примери су
    - Шаблонски метод и
    - Интерпретатор

Објектни обрасци понашања



- Користе састављање објеката а не наслеђивање
- Неки од њих описују како група равноправних објеката сарађује на задатку који ниједан од њих не може сам да обави
  - Овде је важно питање како равноправни објекти сазнају један за другога
  - Равноправни објекти би могли да чувају међусобне експлицитне референце али би то појачало њихово везивање
    - У крајњем случају би сваки објекат знао за све остале
- Примери:
  - Посредник

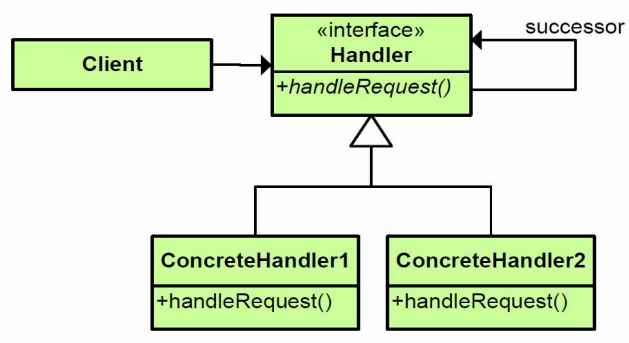
Обрасци понашања

- Најважнији обрасци понашања су:
  - Ланац одговорности (*Chain of Responsibility*)
  - Команда (*Command*)
  - Интерпретатор (*Interpreter*)
  - Итератор (*Iterator*)
  - Посредник (*Mediator*)
  - Подсетник (*Memento*)
  - Посматрач (*Observer*)
  - Стање (*State*)
  - Стратегија (*Strategy*)
  - Шаблонски метод (*Template Method*)
  - Посетилац (*Visitor*)

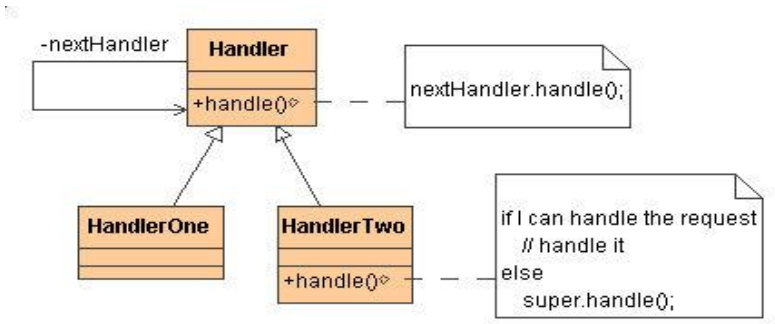
Образац Ланац одговорности (*Chain of Responsibility*)

- Намена
  - Спуштање нивоа спрегнутости између клијента и примаоца објекта
  - Обрада објекта (пакета) се препушта низу објеката, сваки може да уради део посла и да проследи објекат даље
  - Клијент се ослобађа потребе старања о прослеђивању објекта различитим процесорима
- Решава проблем
  - Када на неки догађај (примљену информацију) може или мора да се одреагује на потенцијално више места
- Пример
  - Низ управљача који се старају о појединачним нитима
    - сваки може да преузме посао ако је нит слободна или да га пусти низ ланац ако није
  - Као производна трака у фабрици, сваки елемент ланца може да уради одговарајући део посла и да га препусти даље ако је све у реду

Образац Ланац одговорности (*Chain of Responsibility*)



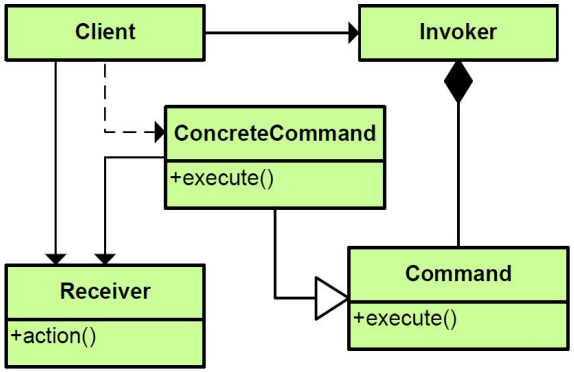
Образац Ланац одговорности (*Chain of Responsibility*) – Пример



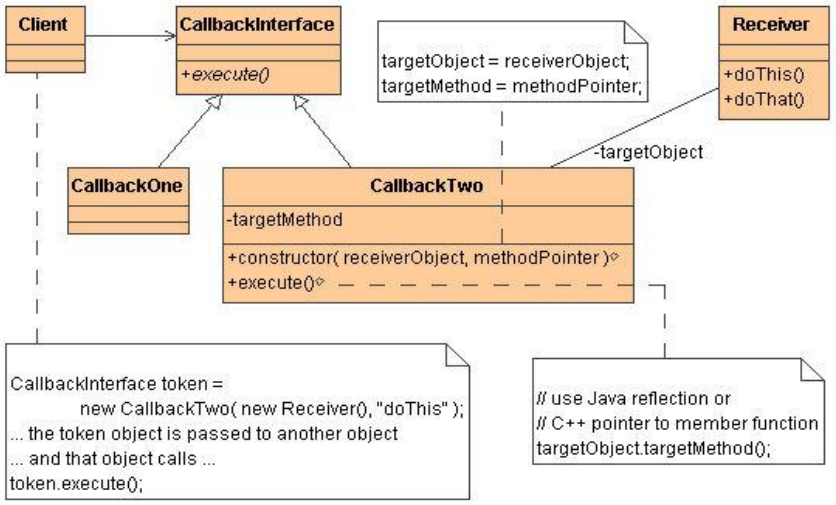
Образац Команда (*Command*)

- Намена
  - Преводи (енкапсулира) захтев у облику посла који је потребно да се уради
  - Омогућава рад са операцијама као са објектима
    - прављење секвенци (макроа)
    - поништавање корака
    - прављење дневника активности
- Решава проблем
  - Слање команди у облику објеката
- Пример
  - Раздвајање одговорности корисничког интерфејса и процесора
    - кориснички интерфејс припрема команде и шаље их процесору

Образац Команда (*Command*)



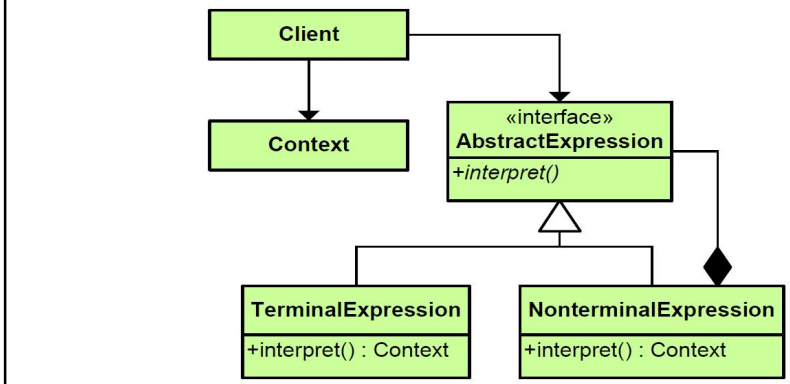
Образац Команда (*Command*) – Пример



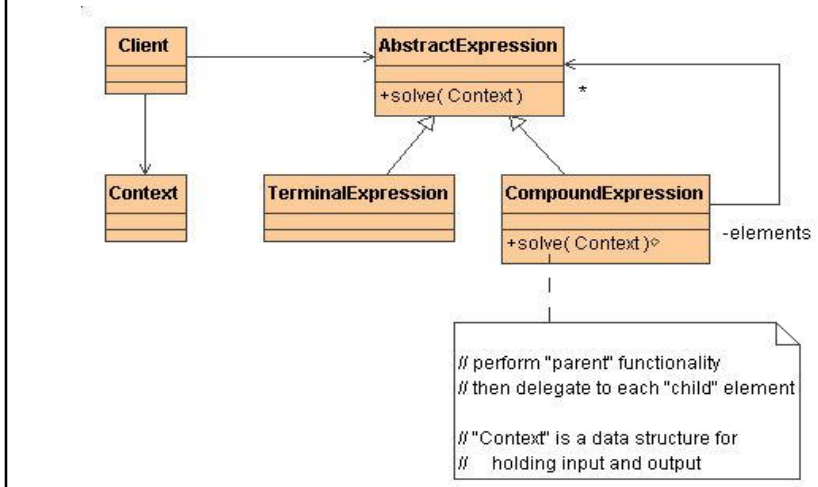
Образац Интерпретатор (*Interpreter*)

- Намена
  - Интерпретација датог језика
  - Пресликава запис на језику у одговарајуће апстрактне структуре
- Решава проблем
  - Читање сложеног записа на неком дефинисаном језику
- Пример
  - Читање података у формату *JSON*

Образац Интерпретатор (*Interpreter*)



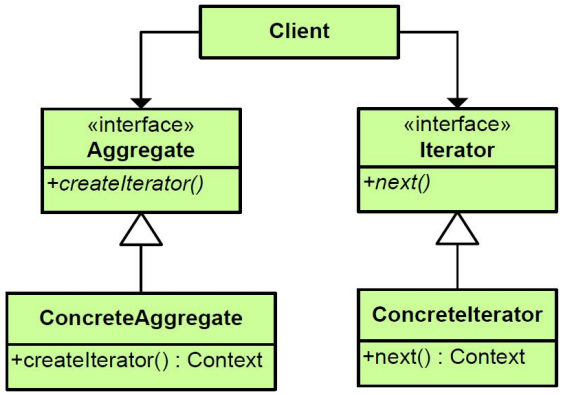
Образац Интерпретатор (*Interpreter*) – Пример



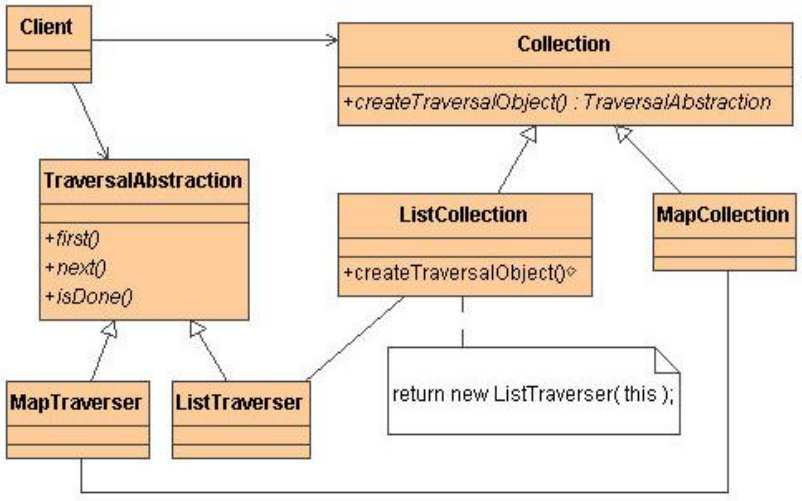
Образац Итератор (*Iterator*)

- Намена
  - Апстраховање обиласка елемената колекције
- Решава проблем
  - Обиласка колекције са сложеном унутрашњом структуром
- Пример
  - Итератори стандардне библиотеке *C++*

Образац Итератор (*Iterator*)



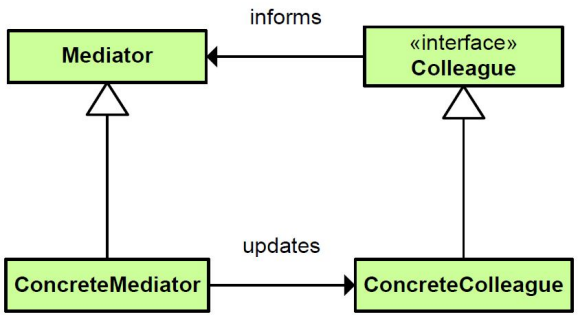
Образац Итератор (*Iterator*) – Пример

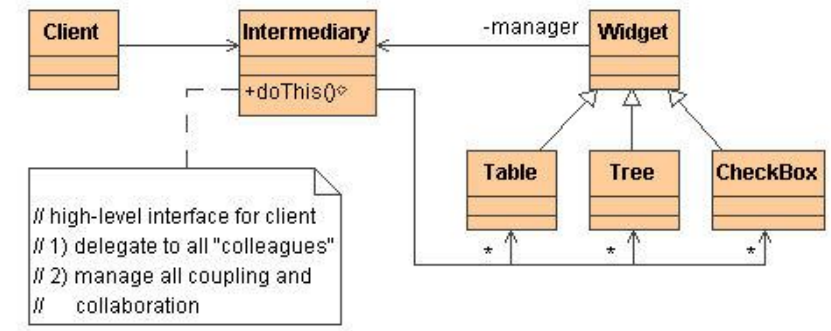


Образац Посредник (*Mediator*)

- Намена
  - Енкапсулирање сложене интеракције између више објеката
  - Омогућава имплементирање функционалних односа више-више
  - Има сличну улогу као Фасада али не и исти циљ
    - корисници Посредника нису спољни клијенти него управо објекти чије повезивање он олакшава
- Решава проблем
  - Проналажења потребних објеката
  - Успостављања комуникације са потребним објектима
  - Обављања комуникације са њима
- Пример
  - Ако чворови графа морају да комуницирају, често је једноставније да то раде преко посредника

Образац Посредник (*Mediator*)

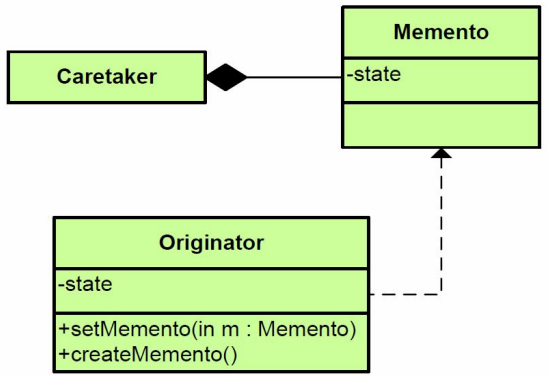




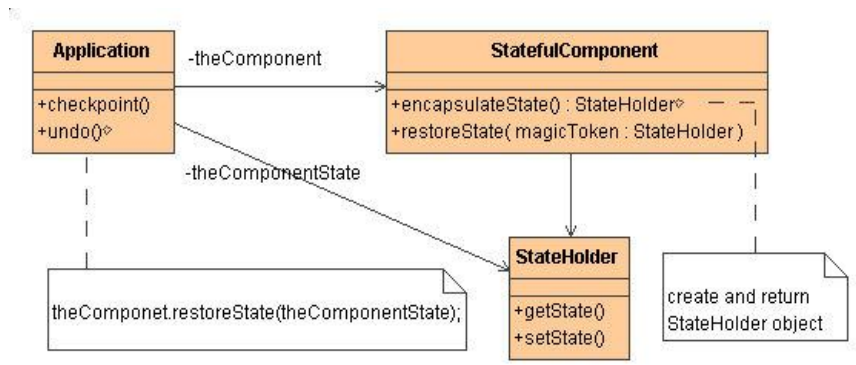
Образац Подсетник (*Memento*)

- Намена
  - Чување стања неког објекта да би он касније могао да се врати у то стање, ако буде потребно
  - На тај начин може да се омогућава поништавање промена или чување стабилног стања
  - Обично се користи у комбинацији са обрасцем Стање
- Решава проблем
  - Враћања објекта у претходно стање
- Пример
  - Чување стања слике пре њеног мењања

Образац Подсетник (*Memento*)



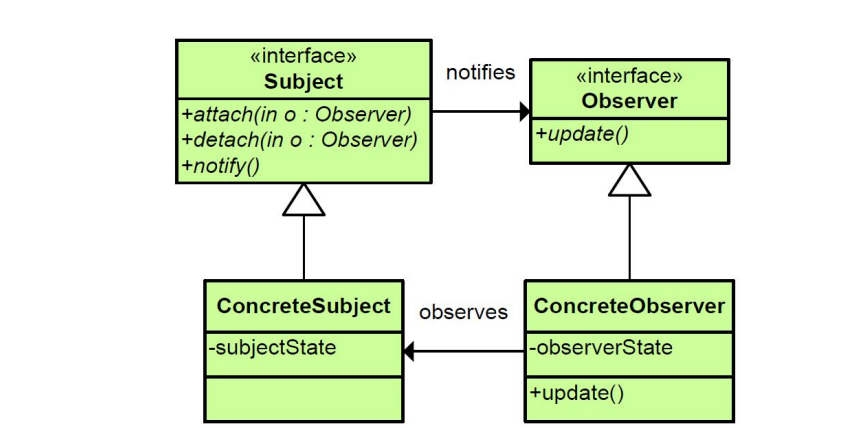
Образац Подсетник (*Memento*) – Пример



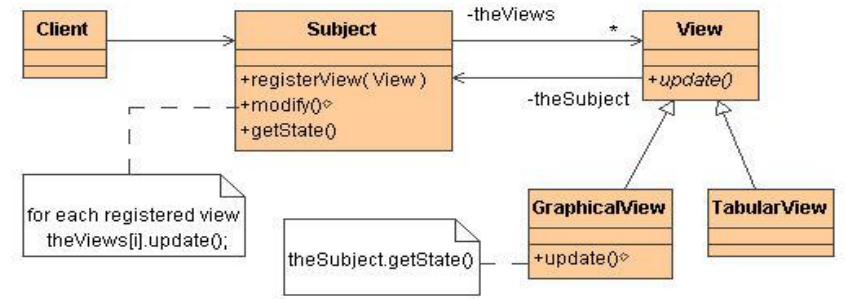
Образац Посматрач (*Observer*)

- Намена
  - Повезивање више *посматрача* са једним *посматраним* објектом
- Решава проблем
  - Дистрибуирање информација о ажурирању свим објектима којима је то потребно
- Пример
  - Раздвајање података и репрезенације
  - Једна колекција података која има више различитих визуалних репрезентација

Образац Посматрач (*Observer*)



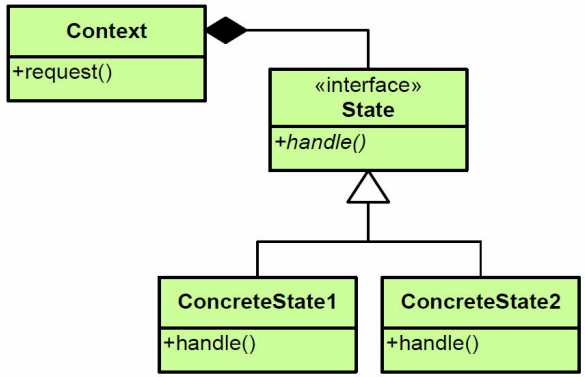
Образац Посматрач (*Observer*) – Пример



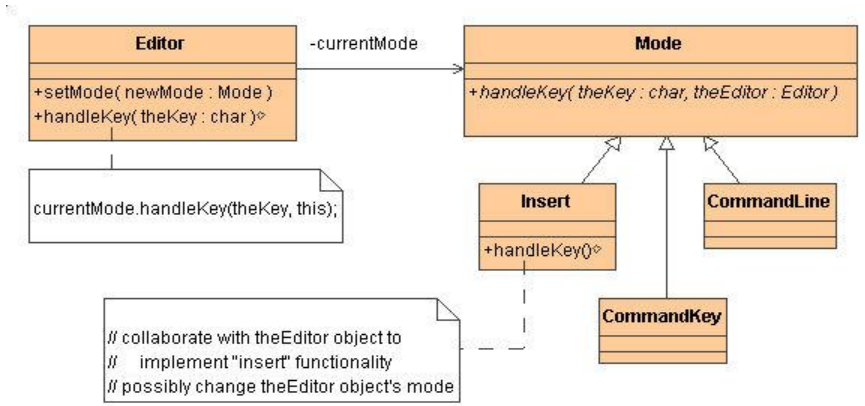
Образац Стање (*State*)

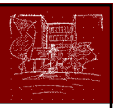
- Намена
  - Омогућава објекту да промени понашање када се промени његово интерно стање
  - Може да представља имплементацију коначног аутомата
- Решава проблем
  - Уместо да се при различитим операцијама проверава стање објекта, заправо различити објекти стања имплементирају различито понашање
- Пример
  - Програм за цртање при промени алата може да постави активан алат као објекат
    - Све акције миша се затим шаљу активном алату, уместо да се проверава стање и бира акција

Образац Стање (*State*)



Образац Стање (*State*) – Пример

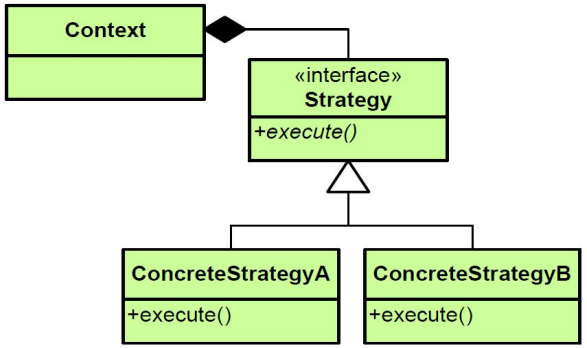




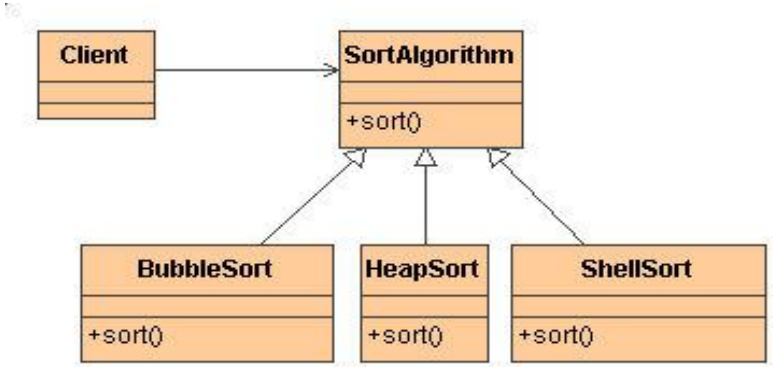
Образац Стратегија (*Strategy*)

- Намена
  - Дефинисање апстрактне класе алгоритама, чији кораци нису фиксни
  - Оквирни поступак је дефинисан, а кораци се препуштају конкретним инстанцама
- Решава проблем
  - Када је потребно да се примењују слични или исти алгоритми али са различитим конкретним корацима
  - Може да омогућава динамички избор алгоритма
  - Представља алтернативу сложеним проверама стања
- Пример
  - На пример, можемо да дефинишемо стратегију која ради независно од типа колекције података:
    - додајемо један по један елемент, затим сортирамо и исписујемо
  - У зависности од колекције, ове операције се имплементирају различито

Образац Стратегија (*Strategy*)



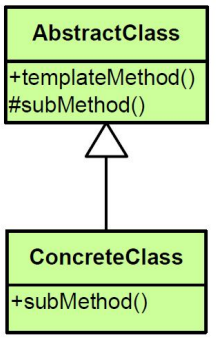
Образац Стратегија (*Strategy*) – Пример



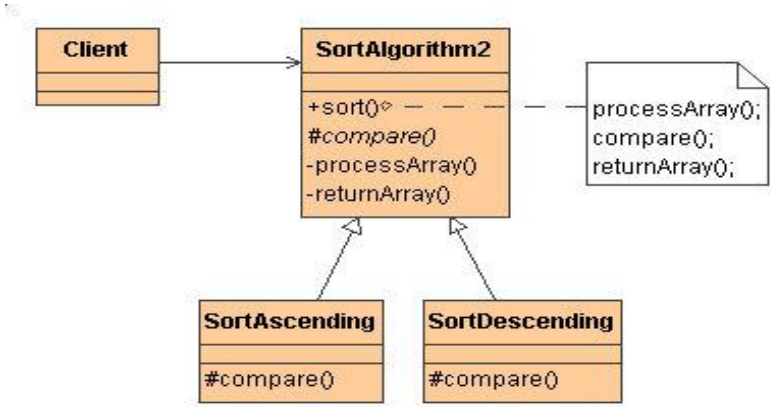
Образац Шаблонски метод (*Template Method*)

- Намена
  - Имплементира оквир понашања а поткласама препушта имплементације корака
  - Слично Стратегији али
    - Стратегија делегира кораке другим класама а Шаблонски метод их имплементира у хијерархији
    - Стратегија мења појединачне објекте а Шаблонски метод мења класе
- Решава проблем
  - Када је потребно да се примењују слични или исти алгоритми али са различитим конкретним корацима
  - Представља алтернативу сложеним проверама стања
- Пример
  - На пример, можемо да дефинишемо стратегију која ради независно од типа колекције података:
    - додајемо један по један елемент, затим сортирамо и исписујемо
  - У зависности од колекције, ове операције се имплементирају различито

Образац Шаблонски метод (*Template Method*)



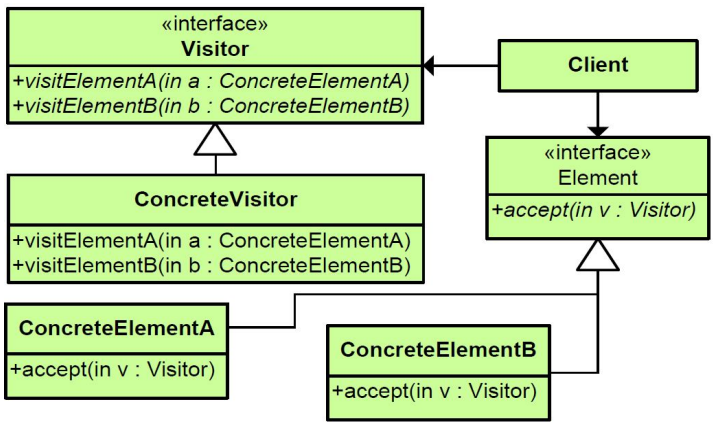
Образац Шаблонски метод (*Template Method*) – Пример



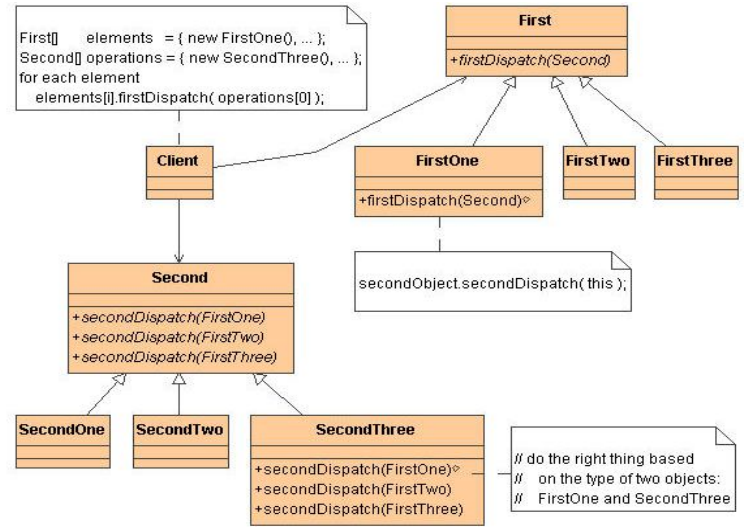
Образац Посетилац (*Visitor*)

- Намена
  - Раздвајање операције које је потребно да се извршава на елементима неке структуре од имплементације самих елемената структуре
- Решава проблем
  - Омогућава додавање и мењање операција без мењања имплементације структуре
  - Одговорности извођења операције су на посетиоцу, а (универзална) одговорност обилажења структуре је на структури
- Пример
  - Рачунање површине, обима и сл. за сложене ликове
  - Извођење операција на графовима и хијерархијама објеката

Образац Посетилац (*Visitor*)



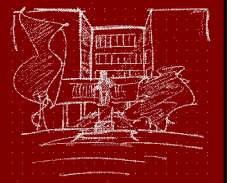
Образац Посетилац (*Visitor*) – Пример



Литература за тему

- *Gamma, Helm, Johnson, Vlissides*,
  Design Patterns – Elements of Reusable OO Software,
  Addison Wesley, 1995
  - ***Готова решења***, *CET, 2002.*
- *Vince Huston*, ***Huston Design Patterns***
  - *http://www.vincehuston.org/dp/*
  - садржи кратак опис и примере примене на језицима *C++* и *Java*
    - постоје и сложенији примери са више образаца
  - дијаграми у овој презентацији су преузети одатле
- **Refactoring Guru**
  - *https://refactoring.guru/*
- *Christopher Alexander, Sara Ishikawa, Murray Silverstein*,
  A Pattern Language: Towns, Buildings, Construction,
  Oxford University Press, 1977.

Хвала на пажњи!



МАТФ
Универзитет у Београду
Математички факултет