Objektno-Orijentisano Projektovanje

Obrasci

Sadržaj

[Arhitekturni obrasci: 2](#_Toc70268765)

[Layer 2](#_Toc70268766)

[Projektni obrasci: 3](#_Toc70268767)

[Singleton 3](#_Toc70268768)

[Observer 3](#_Toc70268769)

[Factory Method 5](#_Toc70268770)

[Command 6](#_Toc70268771)

[Composite 7](#_Toc70268772)

[Bridge 9](#_Toc70268773)

[Facade 10](#_Toc70268774)

[Decorator 11](#_Toc70268775)

[Abstract Factory 13](#_Toc70268776)

[Template Method 15](#_Toc70268777)

[Strategy 16](#_Toc70268778)

[Adapter 18](#_Toc70268779)

## **Arhitekturni obrasci:**

### Layer

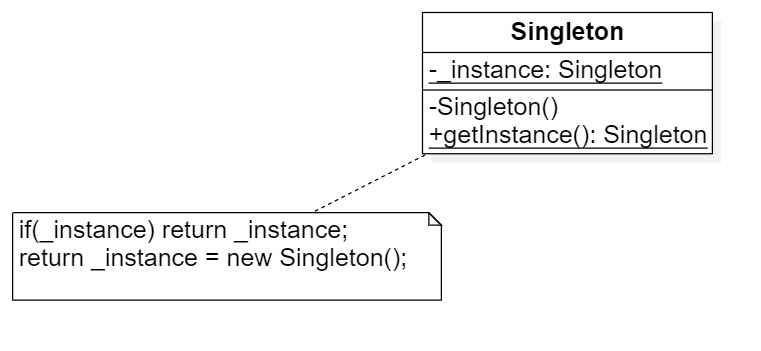
* + Organizacija sistema u nivoe omogućava struktuiranje i dekompozicuju sistema u grupe zadataka, pri čemu je svaka grupa jedan nivo apstrakcije.

## **Projektni obrasci:**

### Singleton

Obezbeđuje da klasa ima samo jednu instancu, kao i globalni pristup toj instanci.

Struktura:

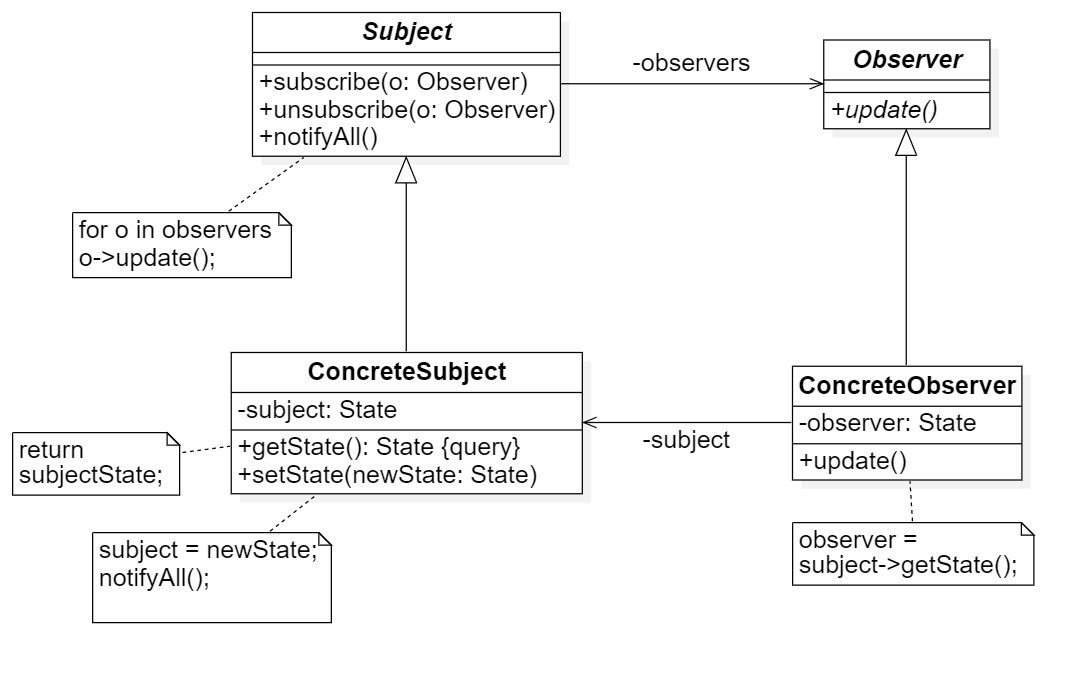


Učesnici:

* Singleton – Definiše operaciju klase getInstance() koja omogućava klijentima da pristupe jedinstvenoj isntanci klase. Može biti odgovoran za sopstveno kreiranje instance.

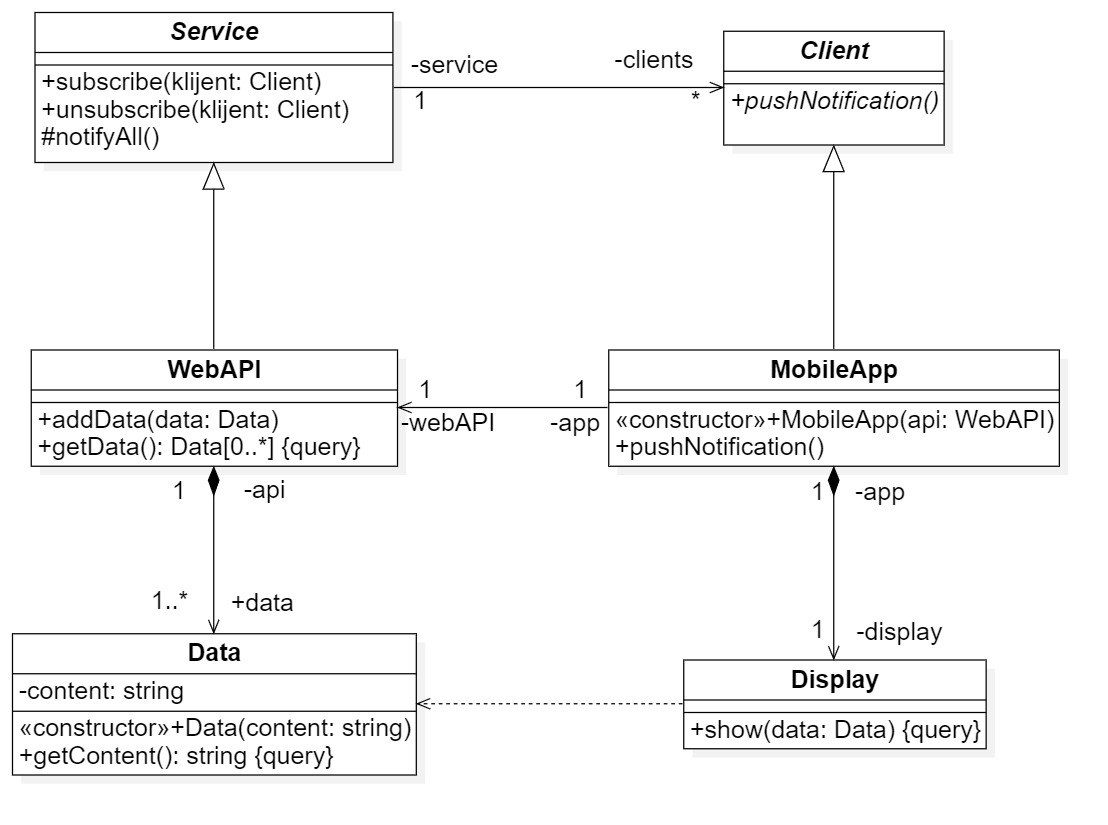
### Observer

Definiše zavisnosti tipa jedan-prema-više (one-to-many) različitim objektima i obezbeđuje da se promena stanja u jednom objektu automatski reflektuje u svim zavisnim objektima.

Struktura:

Učesnici:

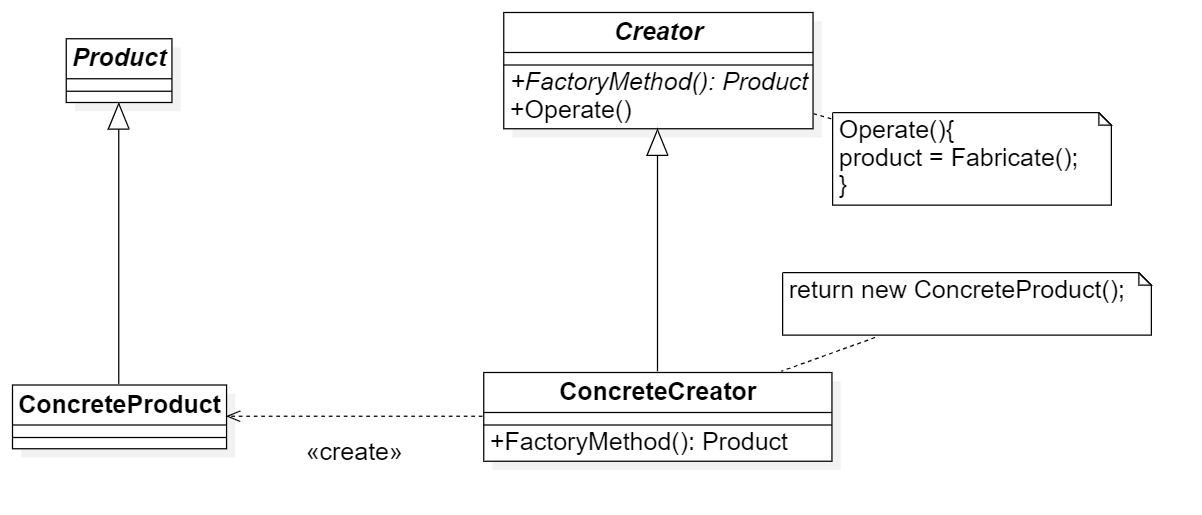
* Subject
  + Čuva referencu prema svom nadzorniku. Jedan objekat može da ima više Observer objekata.
  + Obezbeđuje interfejs za dodavanje i uklanjanje Observer objekata.
* ConcreteSubject
  + Čuva stanje koje će biti od interesa ConcreteObserver objektima
  + Šalje notifikaciju svojim nadzornicima kada promeni stanje
* Observer:
  + Definiše interfejs za ažuriranje objekata nakon što se u objektima klase Subject desi promena
* ConcreteObserver
  + Čuva referencu na ConcreteSubject objekte
  + Čuva stanje koje treba da ostane konzistentno sa stanjem roditeljske klase
  + Implementira interfejs za ažuriranje objekata koji je definisan u klasi Observer.

Primer:

Ovo je moj over the top primer, ne treba sve ovako detaljno, i ja sam se smorio nakon ovoga...

### Factory Method

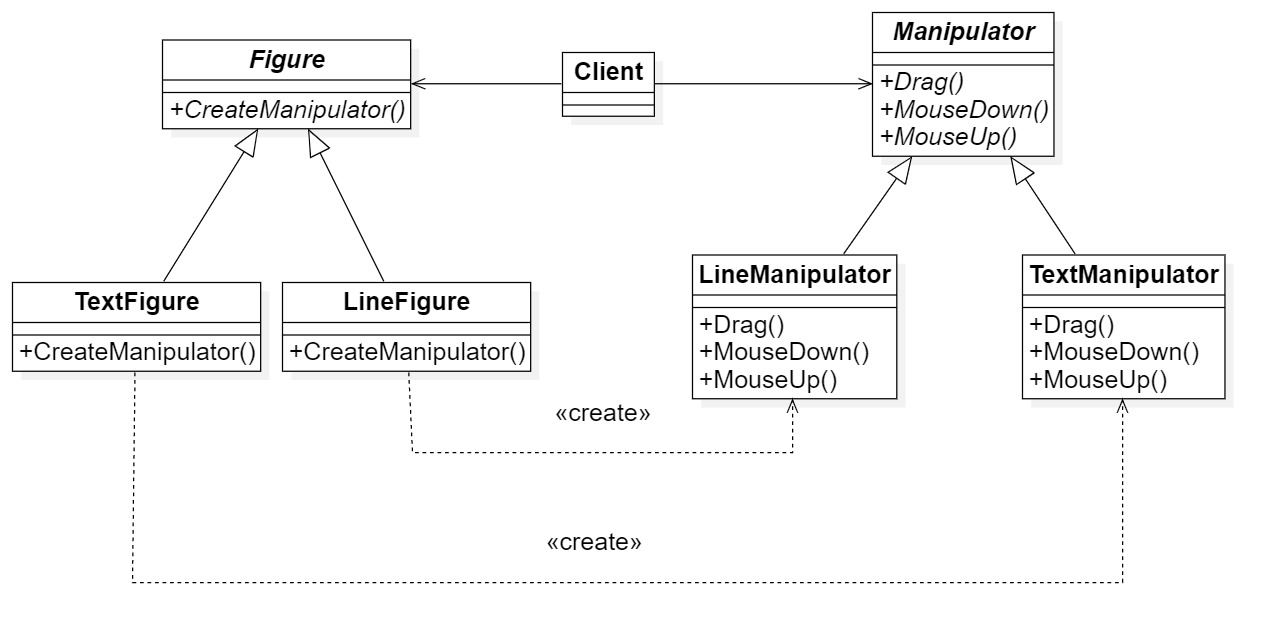
Definiše interfejs za kreiranje objekata, ali prepušta potklasama da odluče koju će klasu instancirati. Omogućava klasi da odloži instanciranje i taj posao prepusti potklasama. Poznat je i pod nazivom Virtual Constructor.

Struktura:

Učesnici:

* Product – Definiše interfejs objekata koje klasa Factory kreira.
* ConcreteProduct – Implementira Product interfejs
* Creator – Deklariše FactoryMethod(), koji vraća objekat tipa Product. Creator može da definiše i podrazumevanu implementaciju FactoryMethod()-a koja vraća podrazumevani ConcreteProduct objekat.
* ConcreteCreator – Overrride-uje FactoryMethod() kako bi se dobila konkretna instanca ConcreteProduct-a.

Primer:



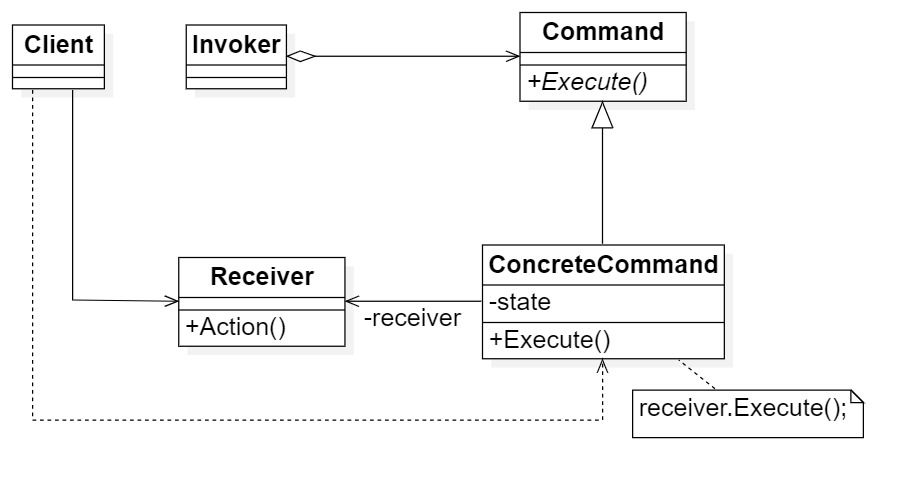
### Command

Enkapsulira zahteve u objekte i na taj način omogućava parametrizovanje klijenata sa različitim zahtevima, organizovanje reda zahteva, logovanje zahteva, omogućava realizaciju UNDO operacije itd.

Razdvaja objekte koji pozivaju operaciju od objekata koji znaju kako da izvrše operaciju.

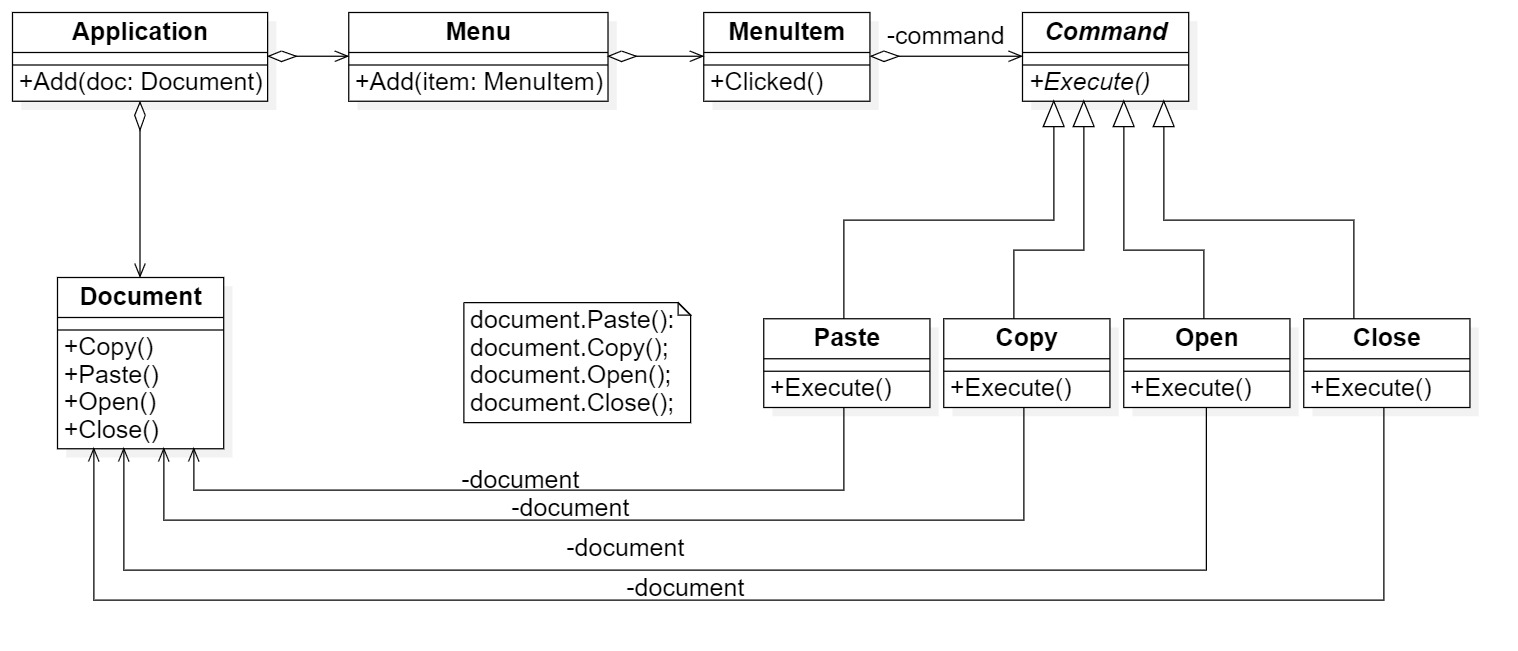
Komande su objekti. Mogu da se prošite kao i bilo koji drugi objekat.

Lako je dodavanje novih komandi, jer ne moraju da se menjaju postojeće klase.

Struktura:

Učesnici:

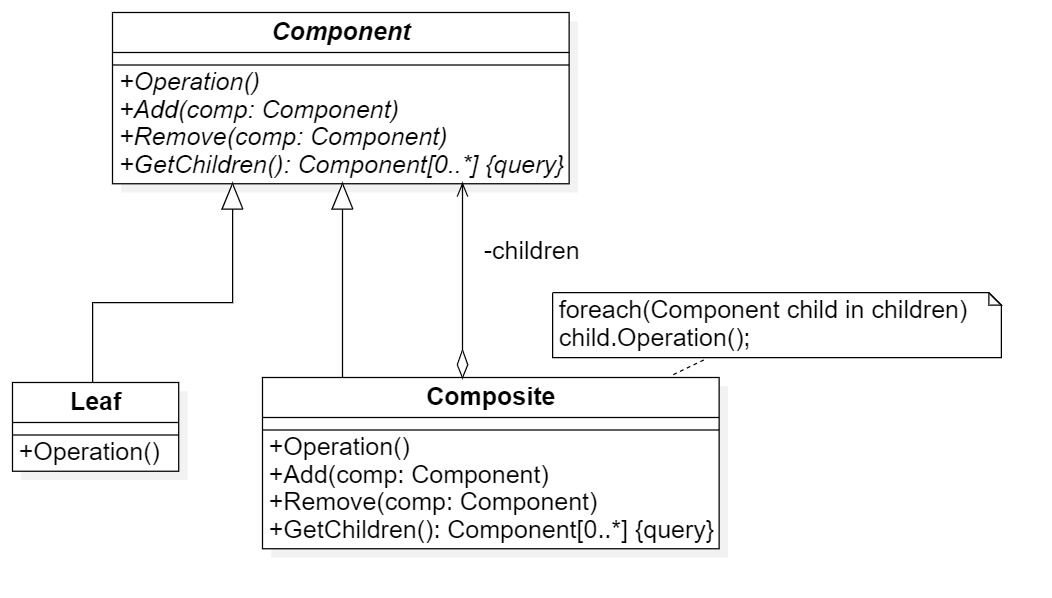
* Command – Deklariše interfejs za izvršavanje operacije
* ConcreteCommand – Definiše vezu između objekta tipa Receiver i konkretne akcije. Takođe implementira Execute() pobuđivanjem odgovarajuće operacije Receiver-a.
* Client – kreira objekat tipa ConcreteCommand i postavlja njegov Receiver.
* Invoker – Traži od Command-e da izvrši zahtev
* Receiver – Zna kako da izvrši operacije u vezi sa zahtevom Invokera. Bilo koja klasa može imati ovu ulogu.

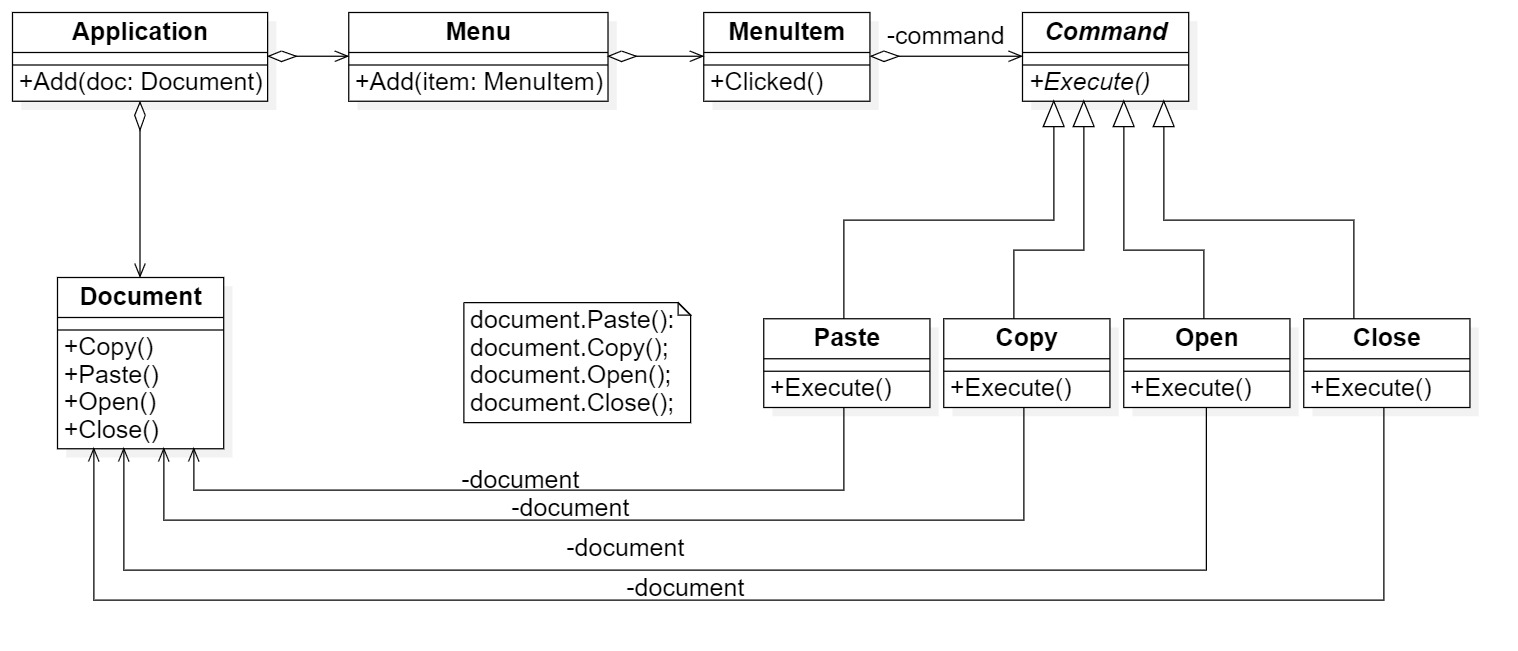
Primer:

### Composite

Uređuje objekte u hijerarhijsku strukturu stabla kako bi predstavio hijerarhiju celina-deo tipa.

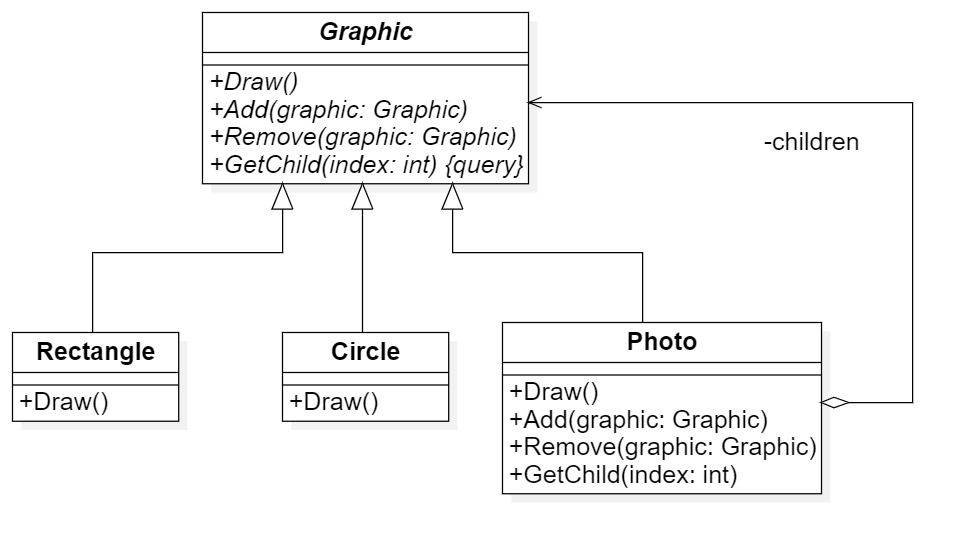
Ovaj obrazac omogućava uniformno tretiranje individualnih objekata i kompozitnih objekata.

Struktura:



Učesnici:

* Component:
  + Deklariše interfejs za objekte koji mogu da čine kompoziciju
  + Implementira osnovno ponašanje za interfejs zajedički svim klasama
  + Deklariše interfejs za pristup i manipulaciju njegovim podelementima (child components)
  + (opciono) definiše i implementira interfejs za pristup roditeljskom elementu u rekurzivnim strukturama.
* Leaf:
  + Predstavlja objekat koji učestvuje u kompoziciji. Ovako se predstavljaju takozvani listovi (elemnenti koji nemaju objekte u kompoziciji)
  + Definiše ponašanje za primitivne objekte u kompoziciji
* Composite:
  + Definiše ponašanje elemenata koji mogu da imaju podselemente
  + Čuva podelemente
  + Implementira operacije koje se odnose na podelemente u intrefejsu klase Component.
* Client:
  + Manipuliše objektima kompozicvije kroz interfejs koji obezbeđuje klasa Component.

Primer:

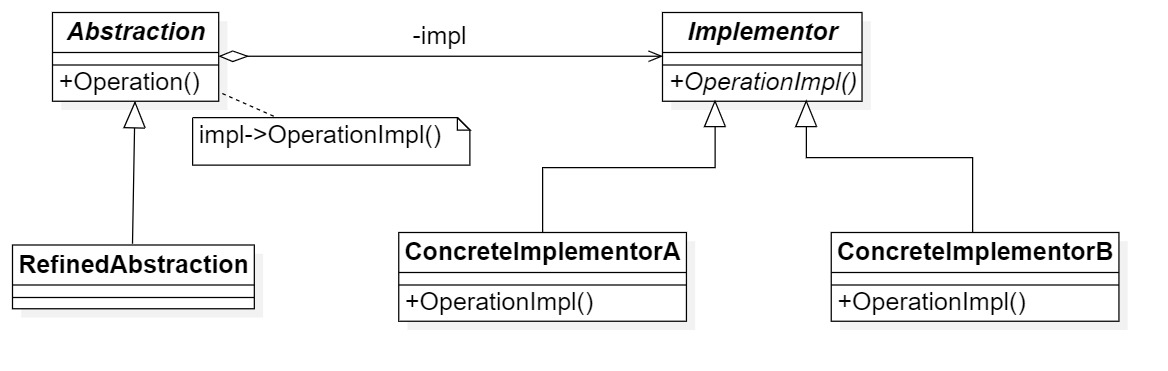
Ovaj primer iz stvarnog sveta prikazuje ulogu obrasca Composite u kreirtanju grafičke strukture stabla koja je sačinjena od primitivnih elemenata: pravougaonika i krugova i složenijih čvorova (grupe elemenata za crtanje od kojih se prave složeniji elementi).

### Bridge

Povezuje razdvojene hijerarhije konceptualnih (apstraktnih) klasa i njihovih konkretnih implementacija.

Promene u implementacionom delu ne zahtevaju promenu koda na klijentskoj strani.

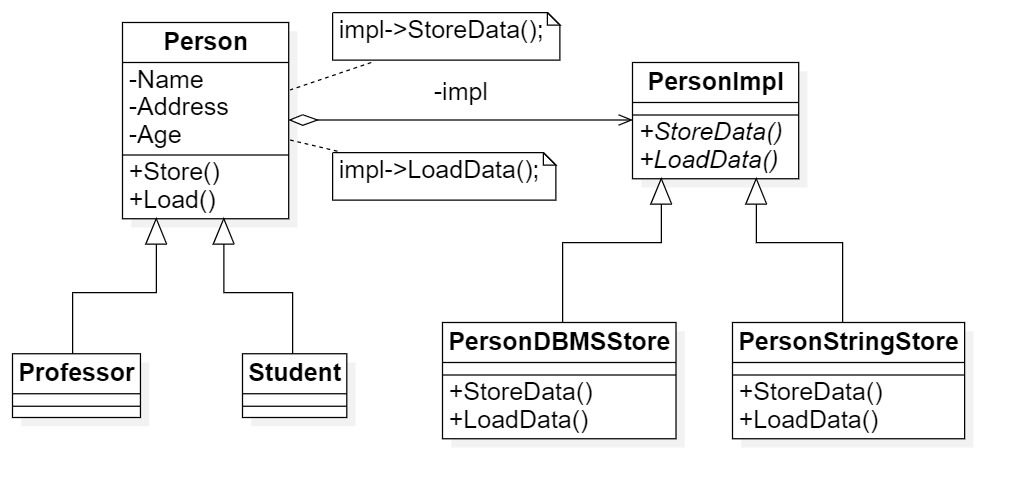
Implementacija je kompletno sakrivena od klijenta.

Struktura:

Učesnici:

* Abstraction – Definiše intrefejs apstrakcije. Održava referencu na objekat tipa Implementor
* RefinedAbstraction – Proširuje interfejs definisan Abstraction klasom
* Implementor – Definiše interfejs za implementacione klase. Ovaj interfejs ne mora da odgovara interfejsu Abstraction, u suštini ovi interfejsi su dosta različiti. Obično je Implementor interfejs koji omogućava samo primitivne operacije, dok Abstraction definiše operacije višeg nivoa, zasnovane na tim primitivnim operacijama.
* ConcreteImplementor – Implementira interfejs Implementor i definiše konkretne primitivne operacije.

Primer:



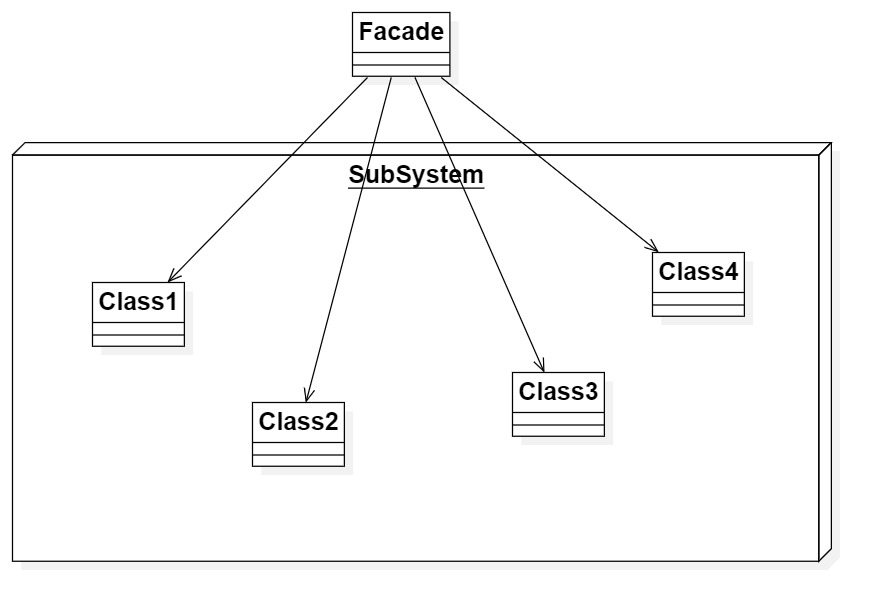
### Facade

Razdvaja podsistem od klijenta koji koriste usluge podsistema.

Obezbeđuje jednostavan interfejs ka celom podsistemu.

Implementacija je kompletno sakrivena od klijenta.

Omogućava jednostavniju organizaciju sistema po slojevima, što olakšava upotrebu podsistema.

Struktura:

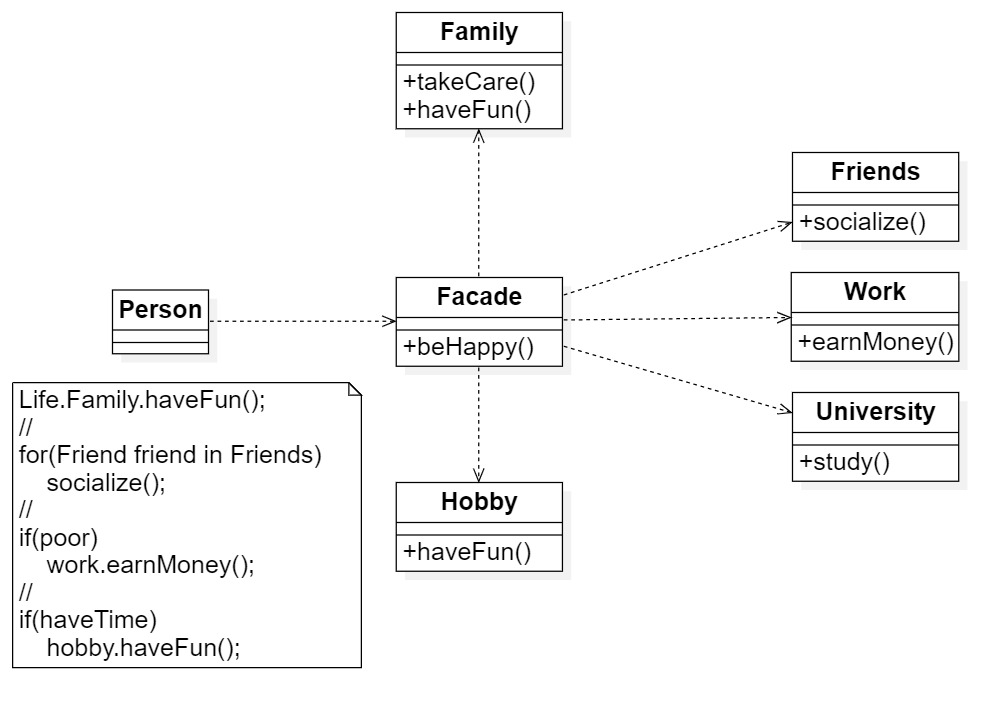
Facade

* Zna koje klase podsistema treba da prihvate spoljni zahtev
* Delegira klijentski zahtev odgovarajućem objektu iz podsistema

Subsystem classes

* Implementiraju funkcionalnost podsistema
* Obrađuju zahteve koji im dolaze od objekta klase Facade
* Nemaju pojma o fasadi i ne čuvaju nikakvu referencu na nju.

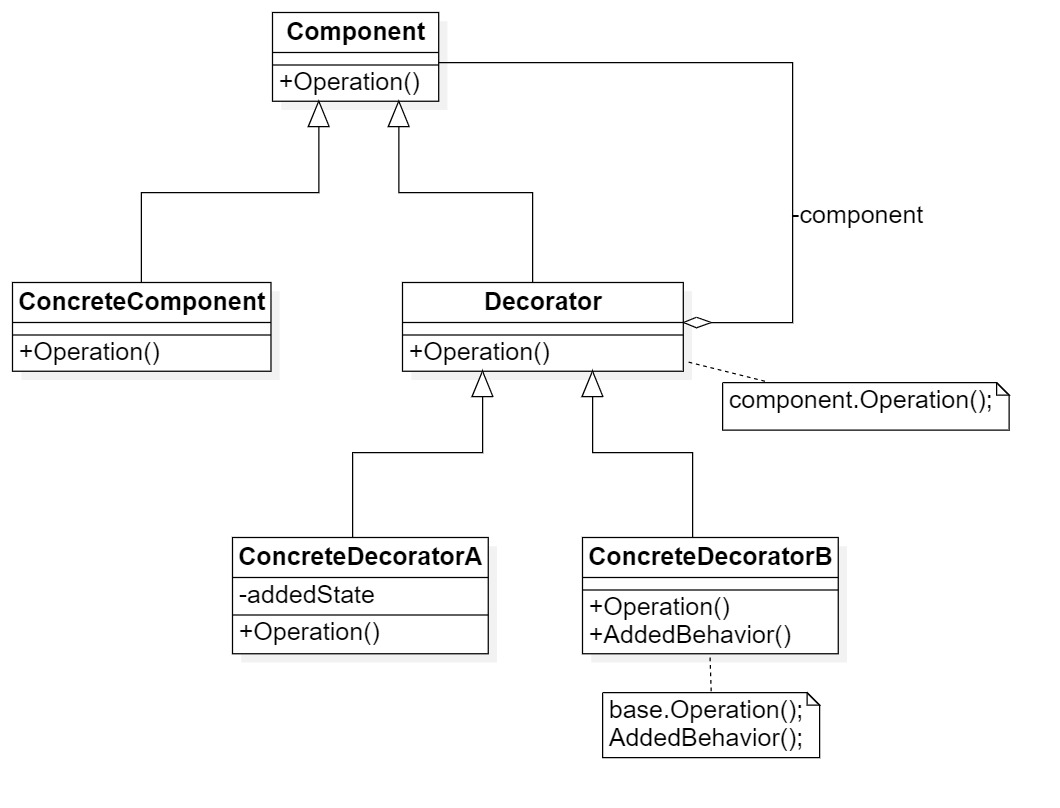
Primer:



### Decorator

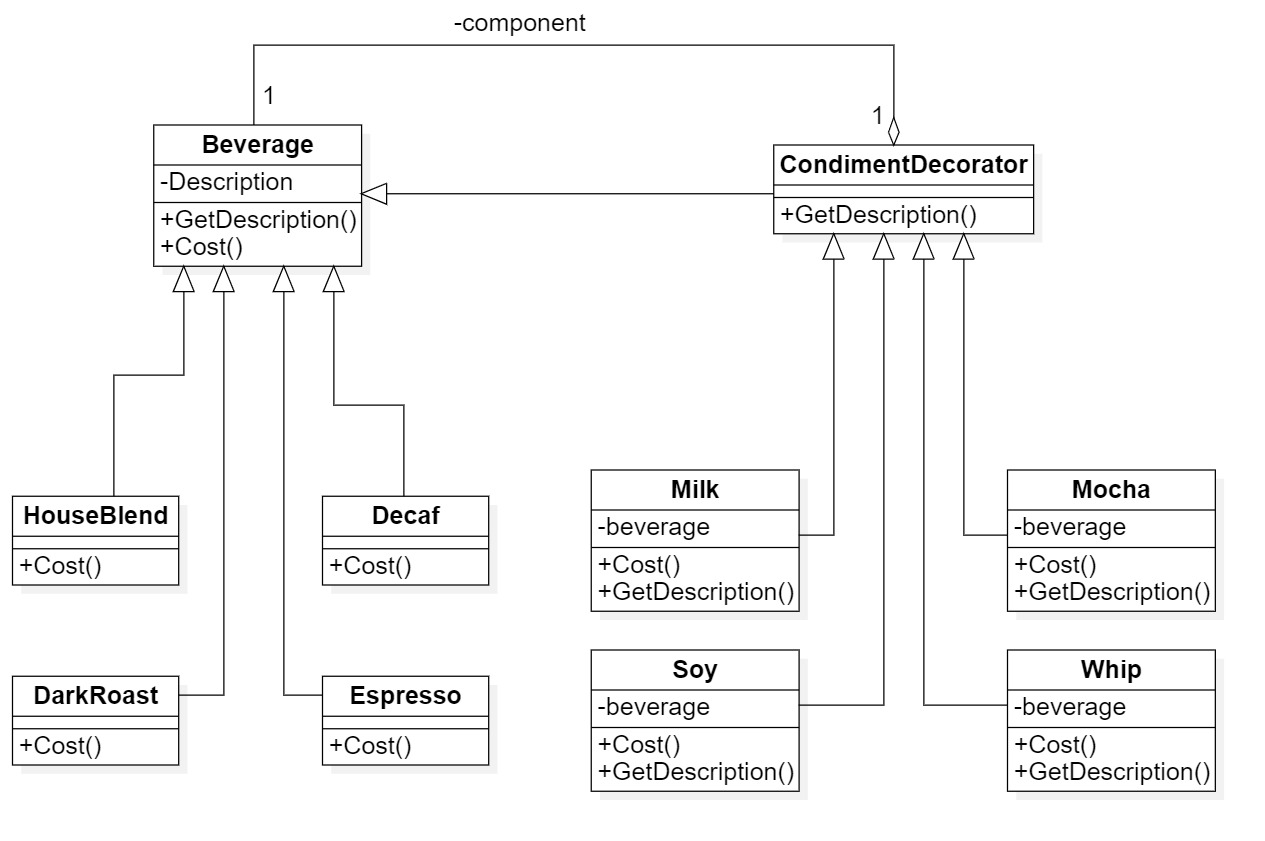
Služi za dinamičko dodavanje novih funkcija nekom objektu. Obrazac Decorator takođe omogućava fleksibilan način za unapređenje funkcionalnosti već dodatih potklasa.

Jednom dodat, ne može se ukloniti, odnosno uništenje se postiže uništenjem same instance koju dekoriše.

Struktura:

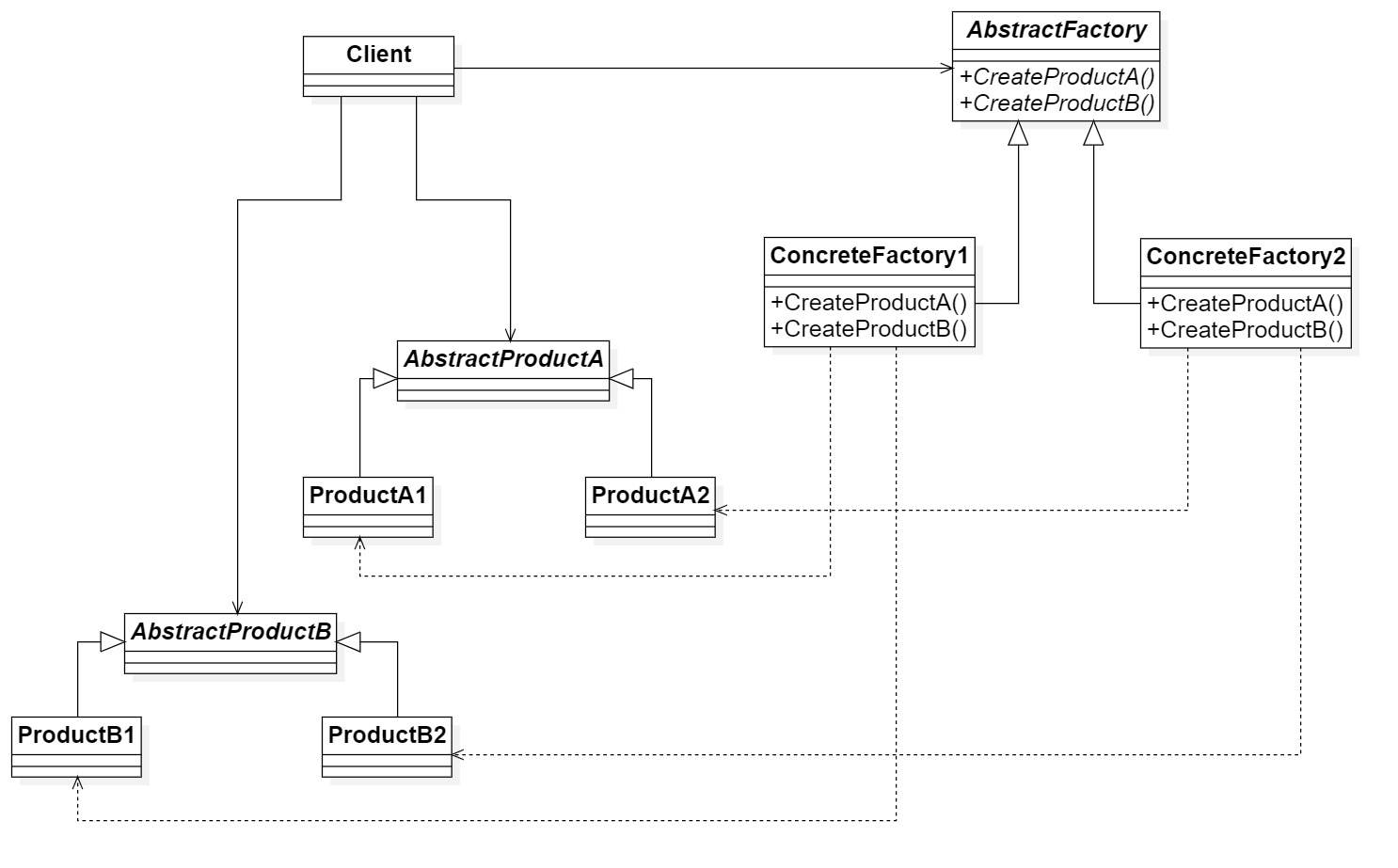
Učesnici:

* Component – Definiše interfejs za objekte kojima se dinamički mogu dodavati nove funkcije
* ConcreteComponent – Predstavlja klasu čijim se objektima mogu dinamički dodavati funkcije
* Decorator – Čuva referencu na objekat klase Componenet i definiše interfejs koji se slaže sa interfejsom klase Component
* ConcreteDecorator – Dodaje nove funkcije komponenti

Primer:

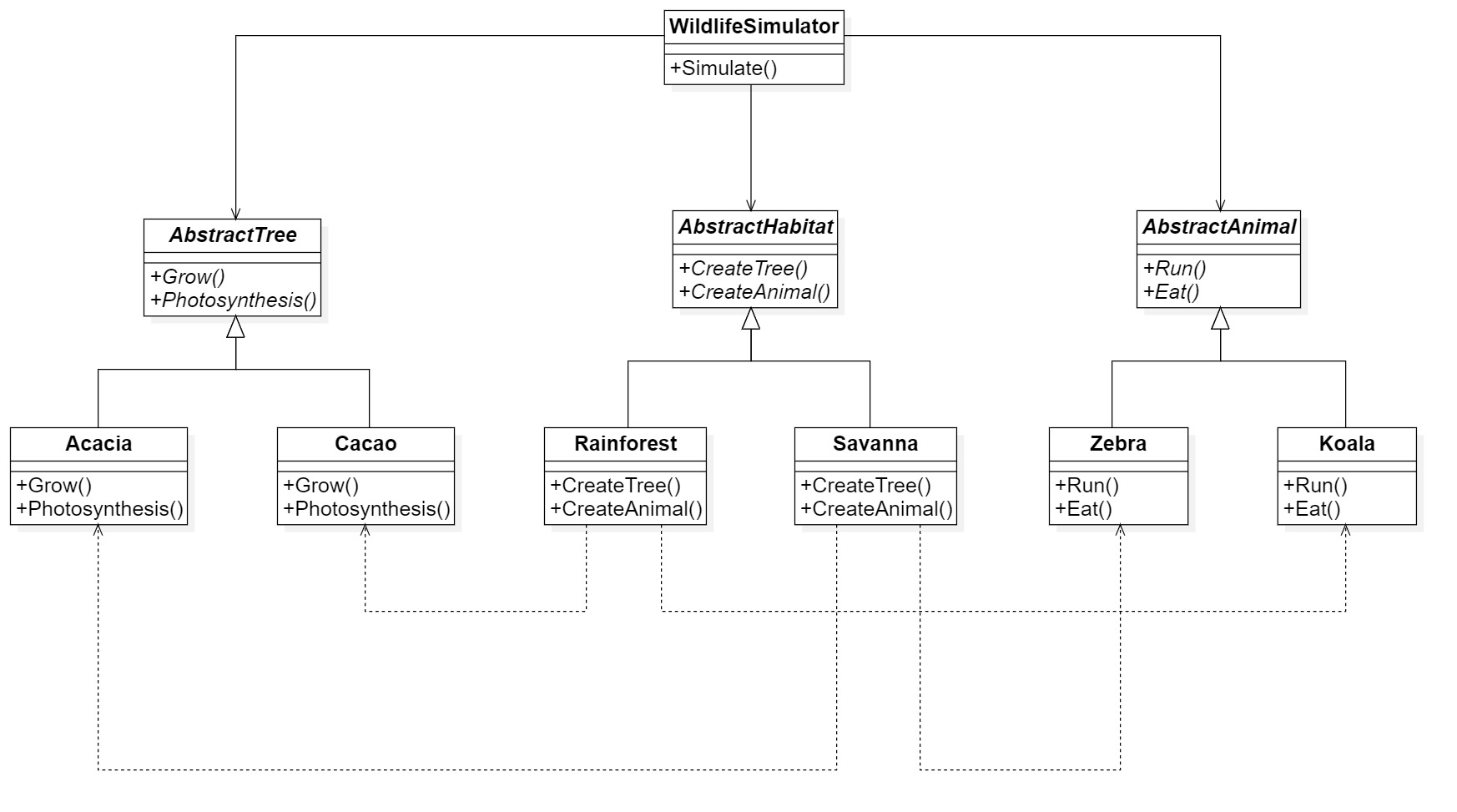
### Abstract Factory

Obezbeđuje interfejs za kreiranje familije povezanih i međusobno zavisnih objekata bez specificiranja njihovih konkretnih klasa.

Struktura:

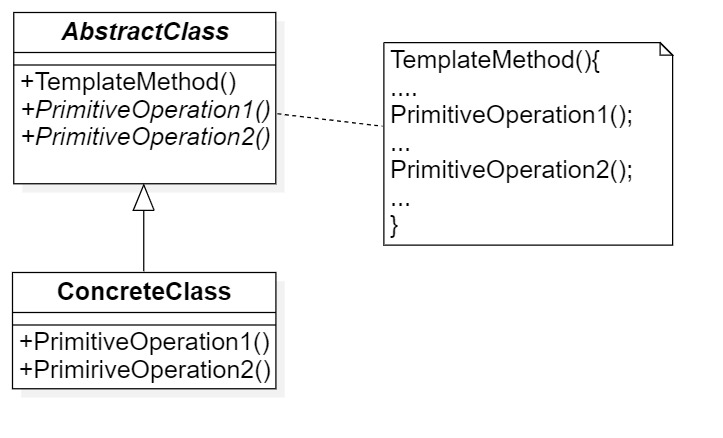
Učesnici:

* AbstractFactory – Deklariše intrefejs za operacije koje kreiraju određene apstraktne objekte
* ConcreteFactory – Implementira operacije za kreiranje određene vrste konkretnih objekata
* AbstractProduct – Deklariše interfejs za određenu vrstu objekata
* Product
  + Definiše objekat koji će biti kreiran u skladu sa odgovarajućom ConcreteFactory klasom
  + Implementira interfejs koji definiše klasa AbstractProduct
* Client – Korisnički interfejs ka klasama AbstractFactory i AbstractProduct

Primer:

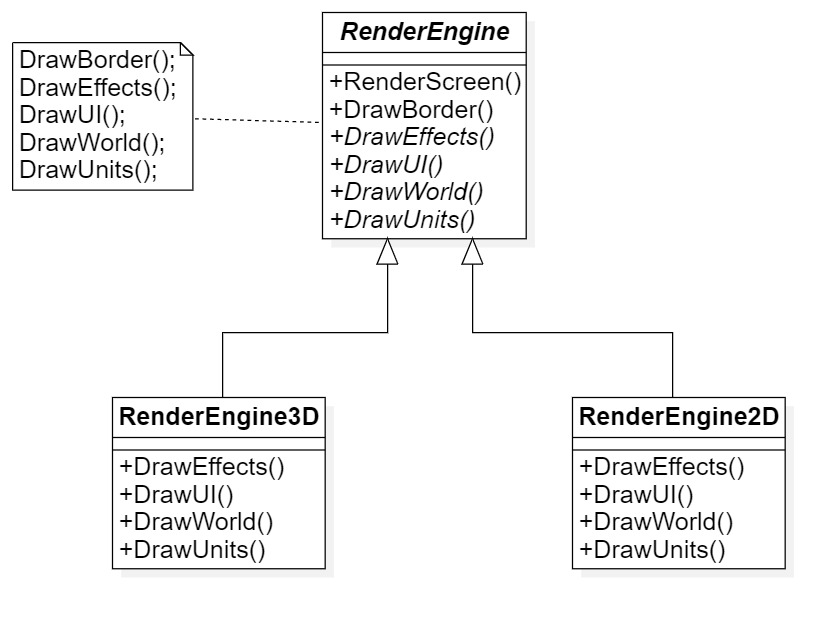
### Template Method

Definiše osnovu algoritma u nekoj operaciji tako što pojedine korake razdvaja u podklase. Ovaj obrazac dozvoljava podklasama da predefinišu određene korake u alrotmu i to bez menjanja strukture algoritma.

Struktura:

Učesnici:

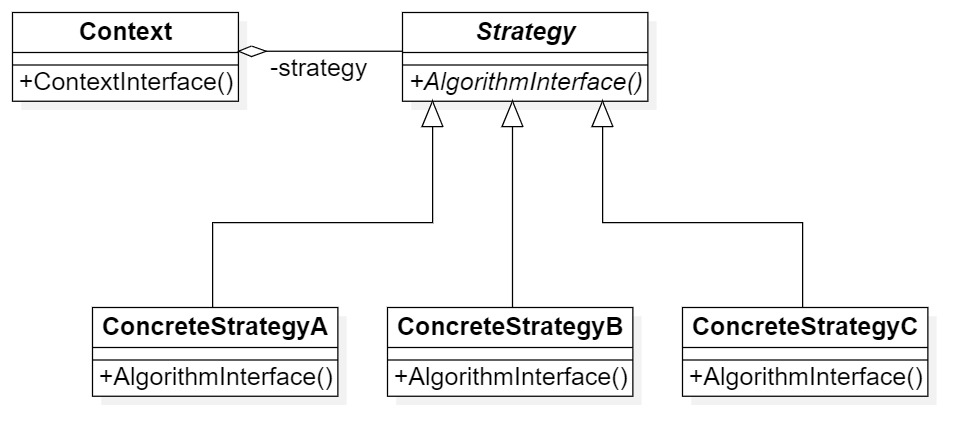
* AbstractClass:
  + Definiše apstraktne primitivne operacije koje nkonkretne podklase koriste za implementaciju algoritama
  + Implementira šablonski metod definišući skelet algoritma. Šablonski metod će pozivati kako primitivne operacije tako i operacije definisane u apstraktnim klasama drugih familija objekata.
* ConcreteClass:
  + Implementira primitivne operacije i vodi računa o specifičnim koracima u algoritmu koji unose podklase.

Primer:

Strategy

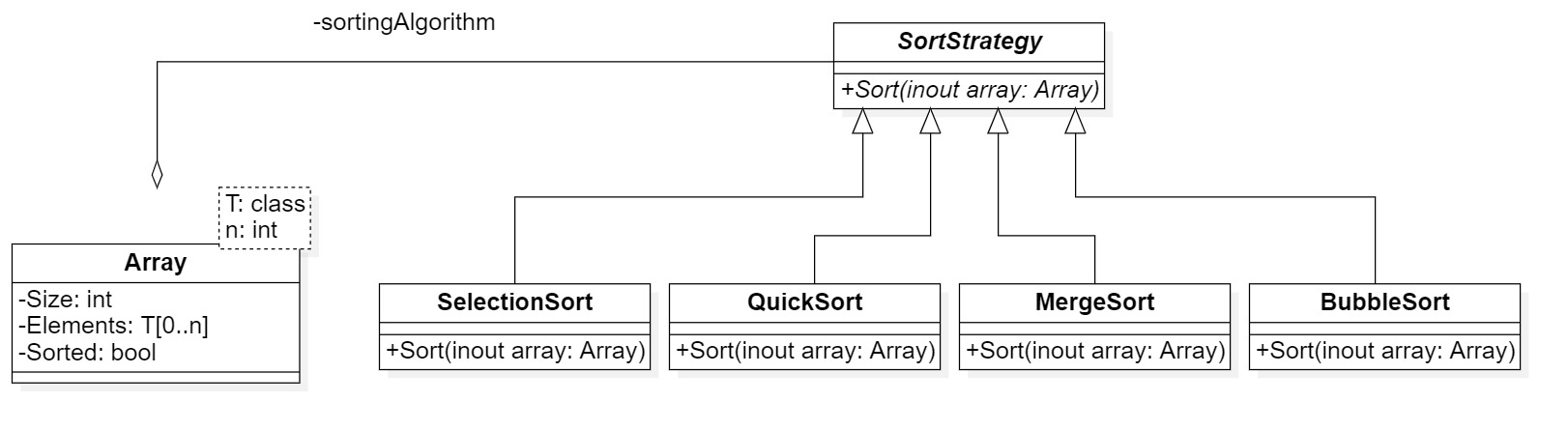
Definiše familiju algoritama, enkapsulacija svakih od njih, čini ih međusobno dostupnim. Strategija dozvoljava algoritmu da bude promenljiv, nezavisno od klijenta koji ga koriste.

Struktura:



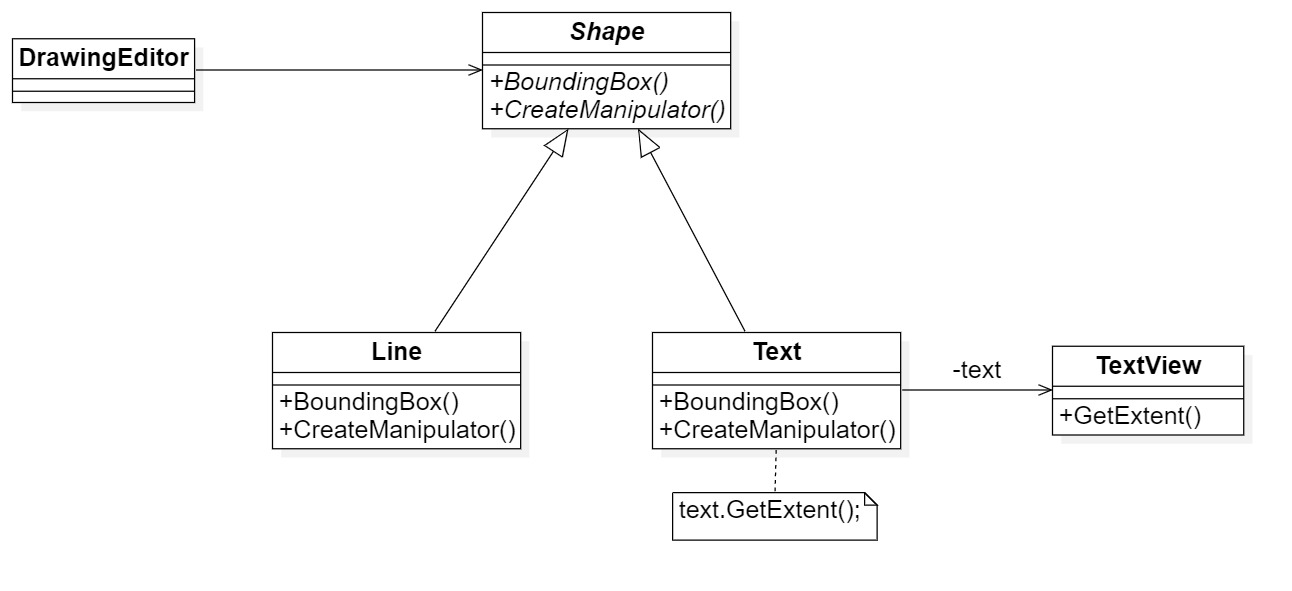
Učesnici:

* Strategy – Deklariše zajednički interfejs za sve podržane algoritme. Objekti klase kontekst koriste ovaj interfejs da pozovu algoritme definisane ConcreteStrategy familijom klasa.
* ConcreteStrategy – Implementira algoritam koristeći interfejs klase Strategy
* Context:
  + Konfiguriše se pomoću ConcreteStrategy objekata
  + Čuva referencu na objekat klase Strategy
  + Može da definiše interfejs koji objektu klase Strategy dozvoljava pristup podacima

Primer:

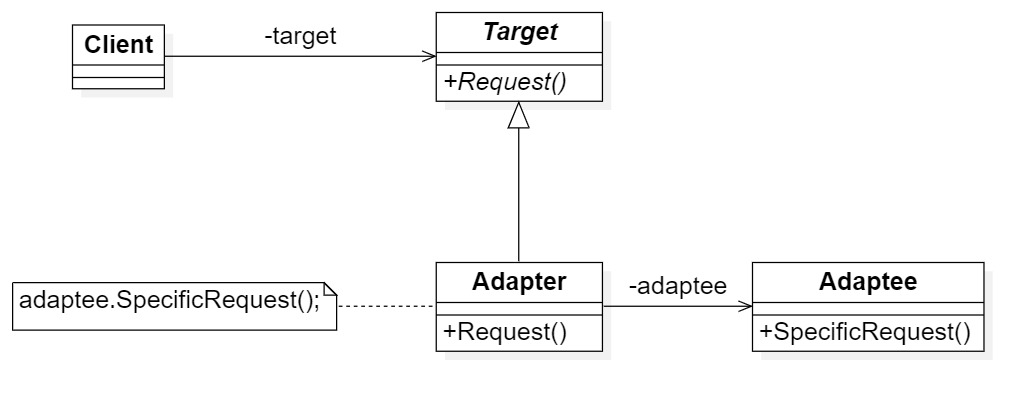
### Adapter

Konvertuje interfejs jedne klase u interfejs kakav očekuje druga klasa. Obrazac adapter omogućava da klase, koje inače ne bi mogle zbog različitih interfejsa, da funkcionišu zajedno. Implementacija ovog obrasca može da bude dosta kompleksna, ukoliko pokušavamo da uklopimo dosta različite klase.

Struktura:

Učesnici:

* Target – Definiše specifični interfejs koji koristi klasa Client.
* Adapter – Adaptira interfejs klase Adaptee prema interfejsu klase
* Adaptee – Definiše postojeći interfejs koji treba adaptirati
* Client – Vodi računa o objektima koji zahtevaju interfejs klase Target.

Primer: