

# Softverski obrasci i komponente

## Strukturalni obrasci

Prof. dr Igor Dejanović ([igord@uns.ac.rs](mailto:igord@uns.ac.rs))

Kreirano 2022-11-01 Tue 23:52, pritisni ESC za mapu, m za meni, Ctrl+Shift+F za pretragu

# Sadržaj

1. Strukturalni softverski obrasci
2. Adapter
3. Bridge
4. Composite
5. Decorator
6. Proxy
7. Literatura

# Strukturalni softverski obrasci

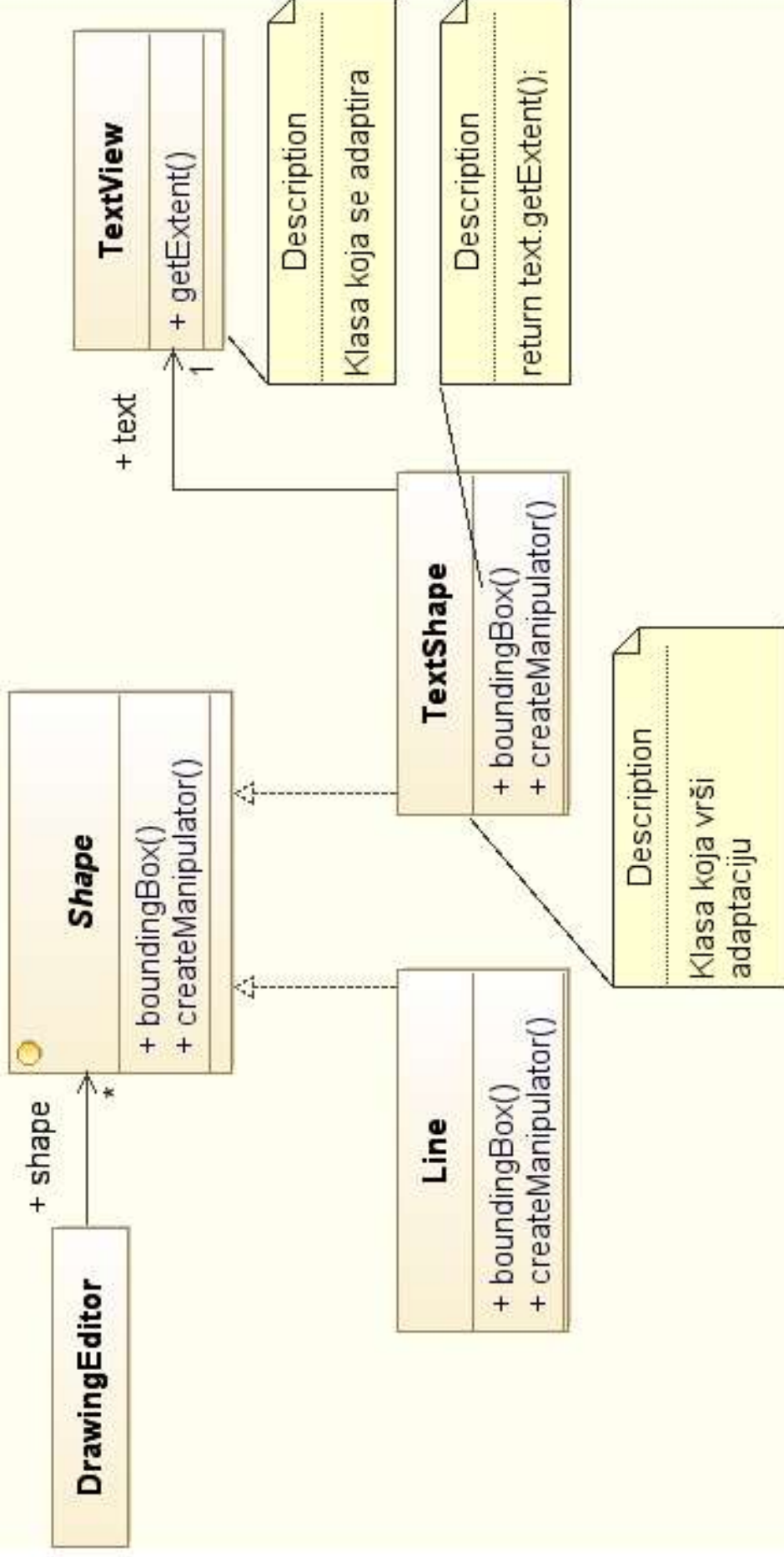
Bave se udruživanjem objekata i klasa, korišćenjem nasleđivanja, kompozicije i delegacije, u cilju formiranja složenijih struktura.

# Adapter

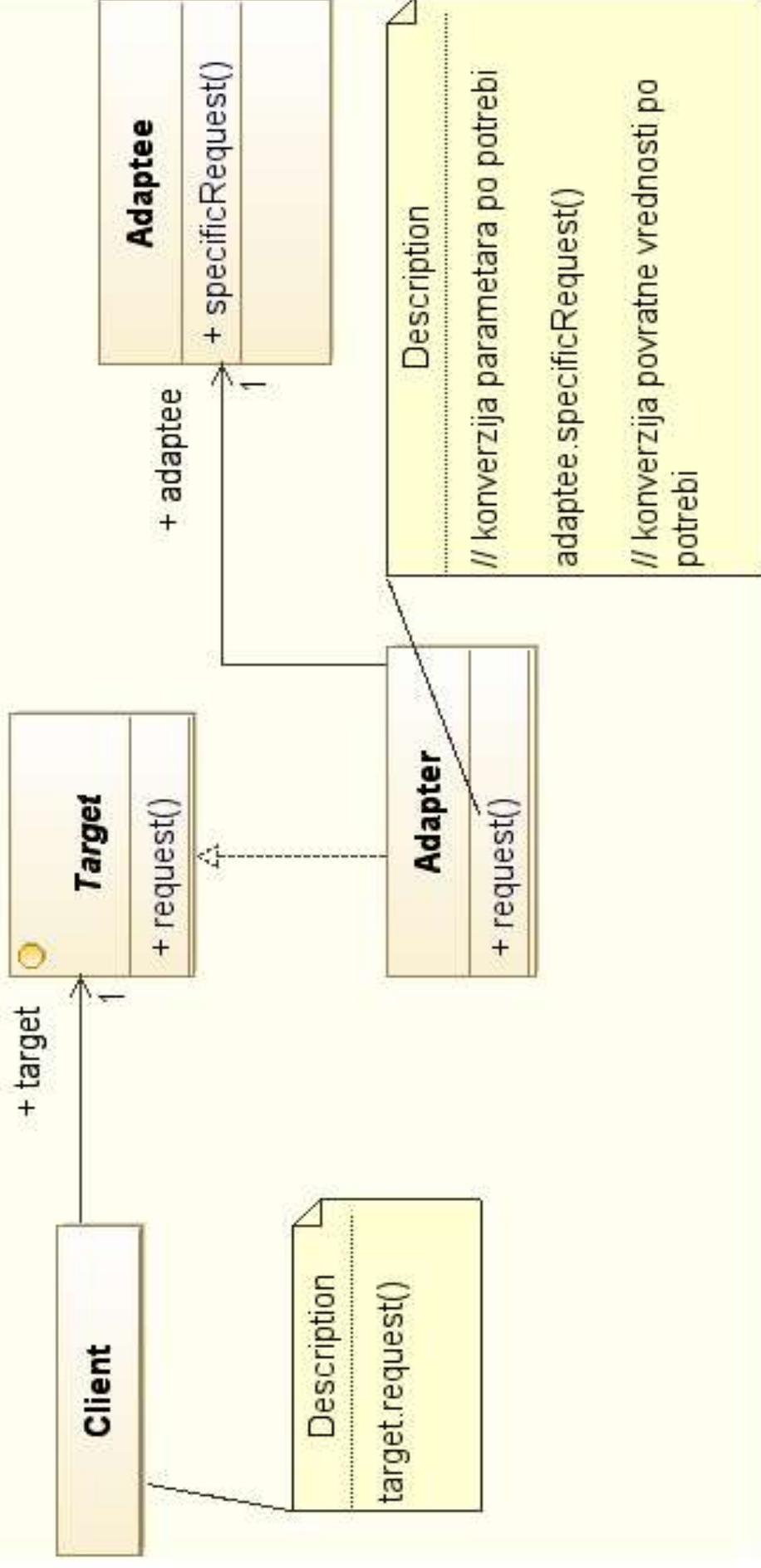
# Adapter

- Prilagođavamo komponentu čija nam je funkcionalnost potrebna ali njen interfejs ne odgovara našim zahtevima.
- Npr. prilagođavanje COTS (*Commercial Of-The-Shelf*) komponenti našem sistemu ili prilagođavanje starih komponenti zahtevima novog sistema.
- Kreiranje komponente tako da je moguće koristiti u različitim, unapred nepoznatim, scenarijima.
- Poznat još i kao *Wrapper*.

# Primer



# Struktura



## Dve verzije

- *Baziran na klasama i nasleđivanju* – Klasa koja se adaptira se nasleđuje i klasa naslednica implementira odgovarajući interfejs.
- *Baziran na objektima i delegaciji* – Objektu koji se adaptira se delegiraju pozivi od strane adapter objekta.

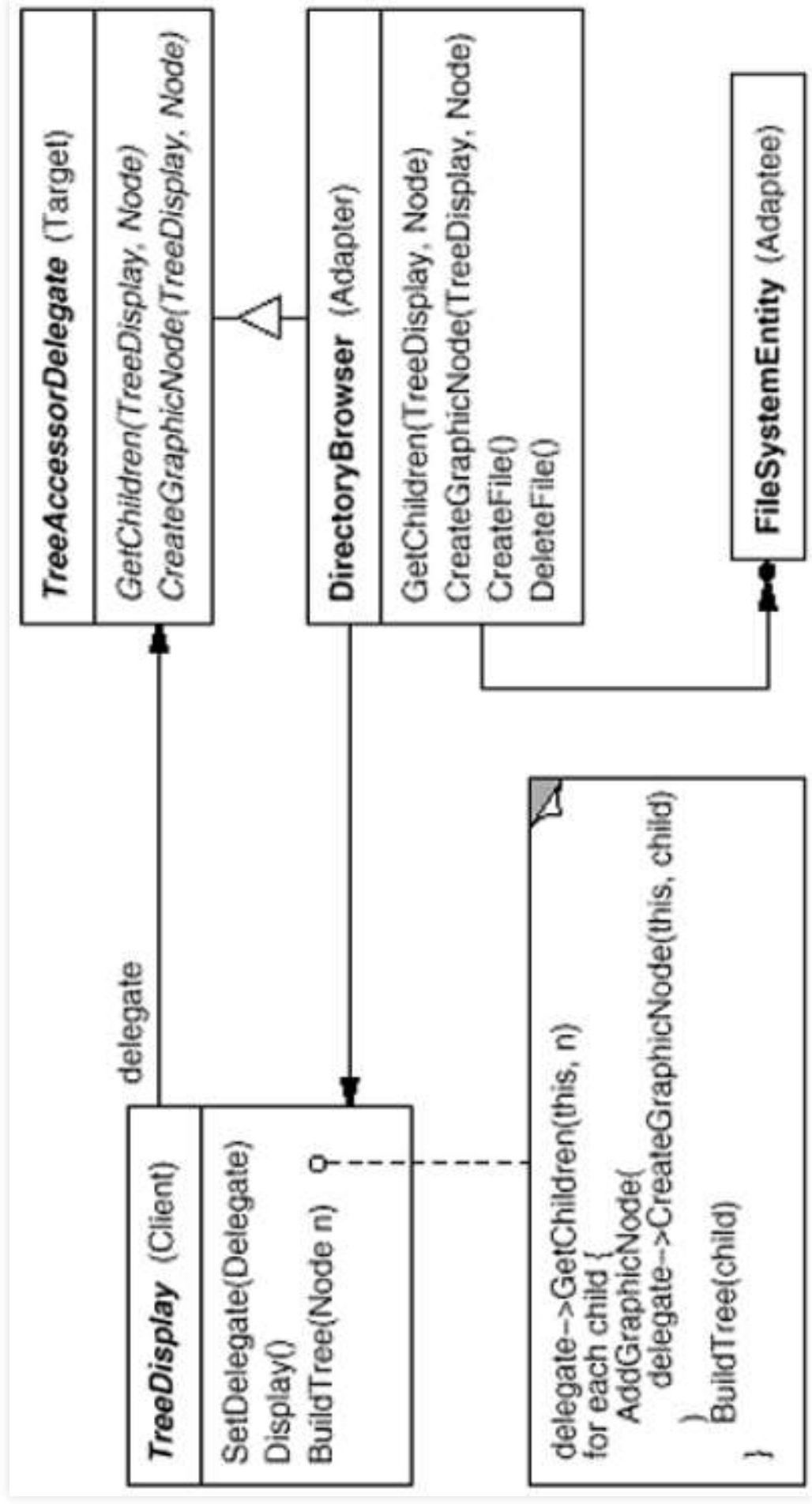


# Napomene

- Količina posla koju adapter radi varira od obične konverzije parametara i naziva metoda do podrške za potpuno drugačiji set operacija.
- Izmenjivi adapteri (*Pluggable adapters*) – Kreiranje komponente tako da se unapred predvidi adaptacija objekata sa kojima treba da sarađuje iako unapred objekti i njihove klase nisu poznati. Primer *TreeDisplay*.<sup>1</sup>

• E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, 1994

# Primer - TreeDisplay



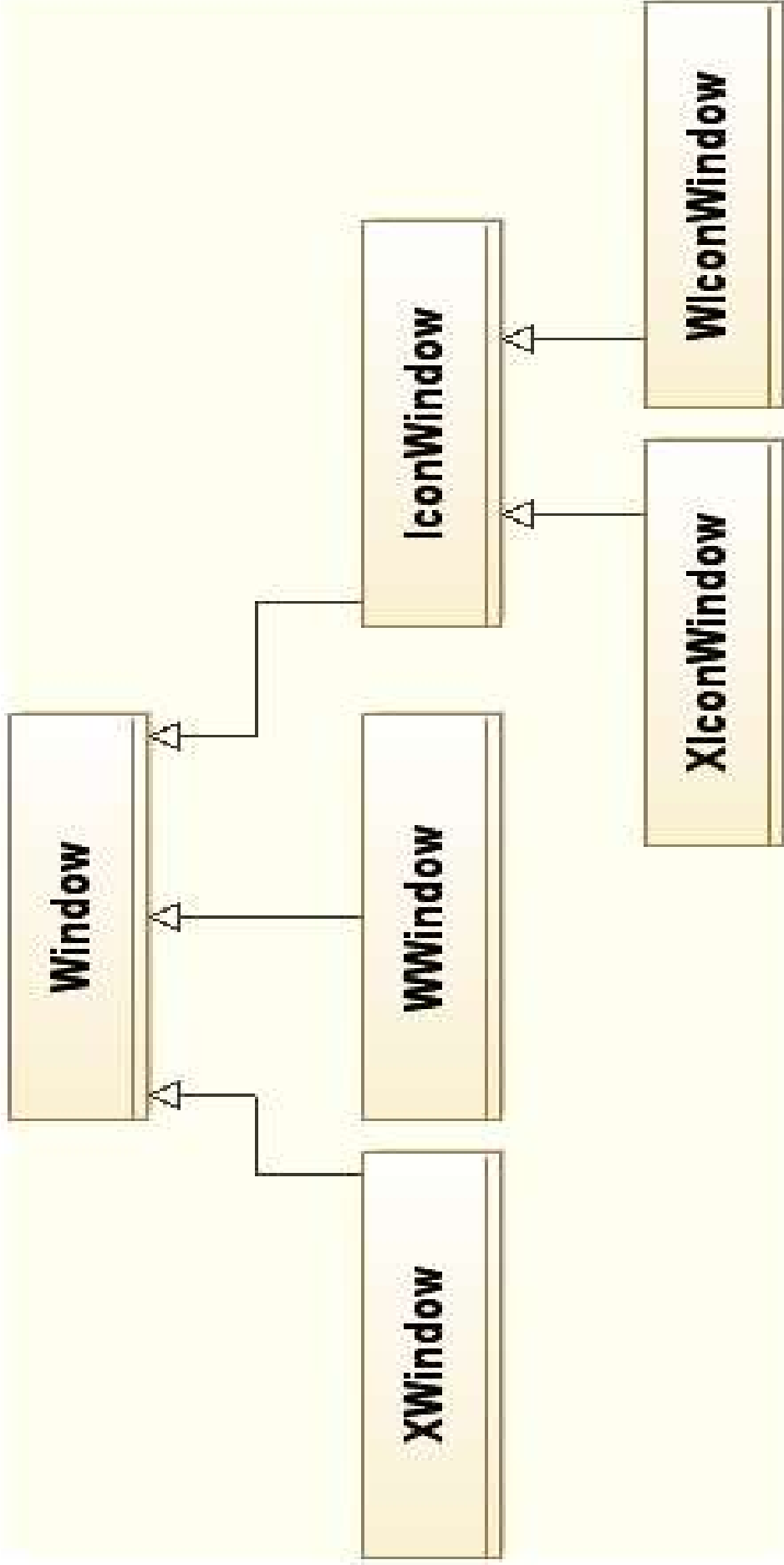
- E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, 1994

# Bridge

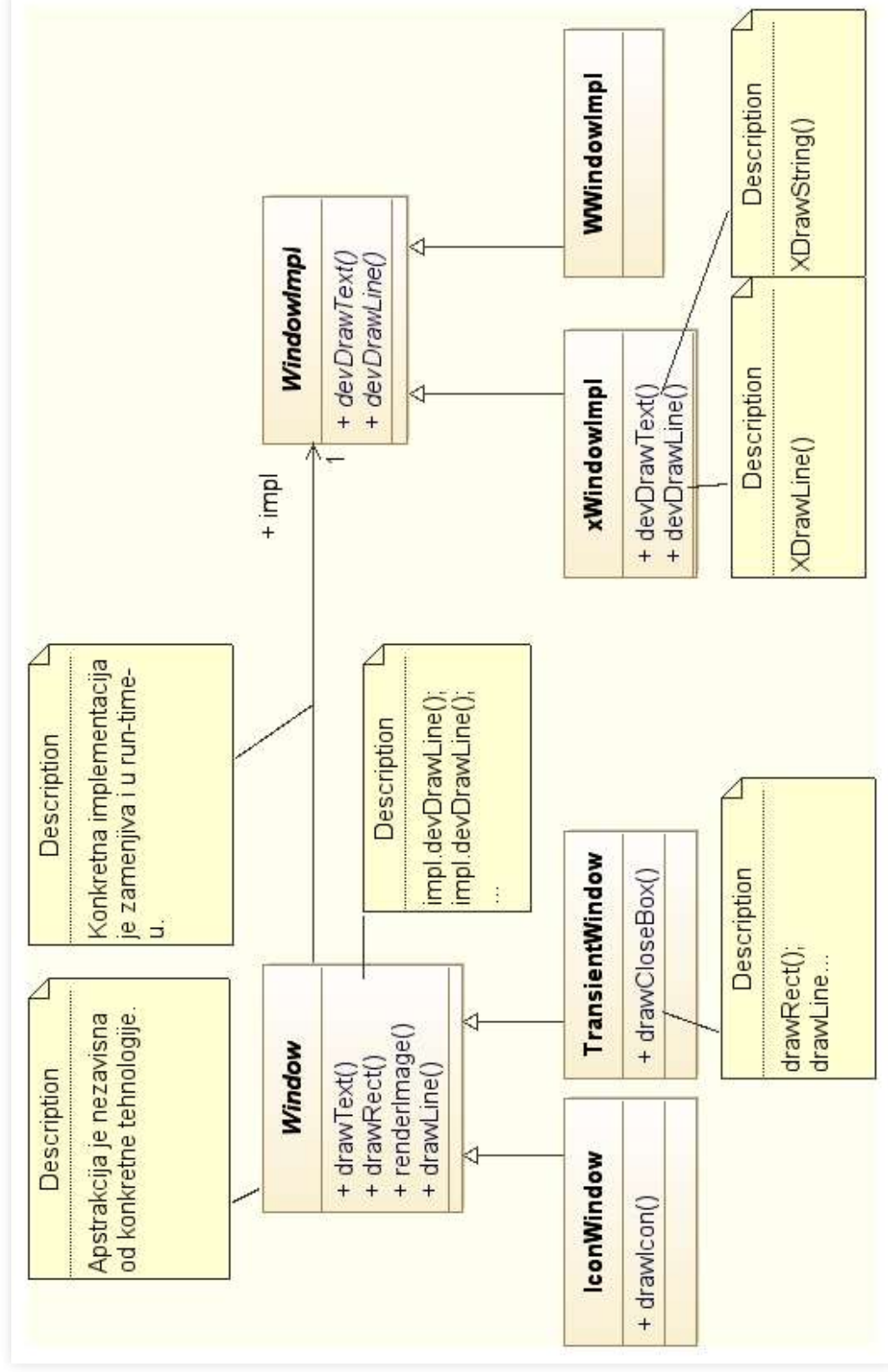
# Bridge

Razdvajanje abstrakcije od implementacije tako da se mogu nezavisno menjati.

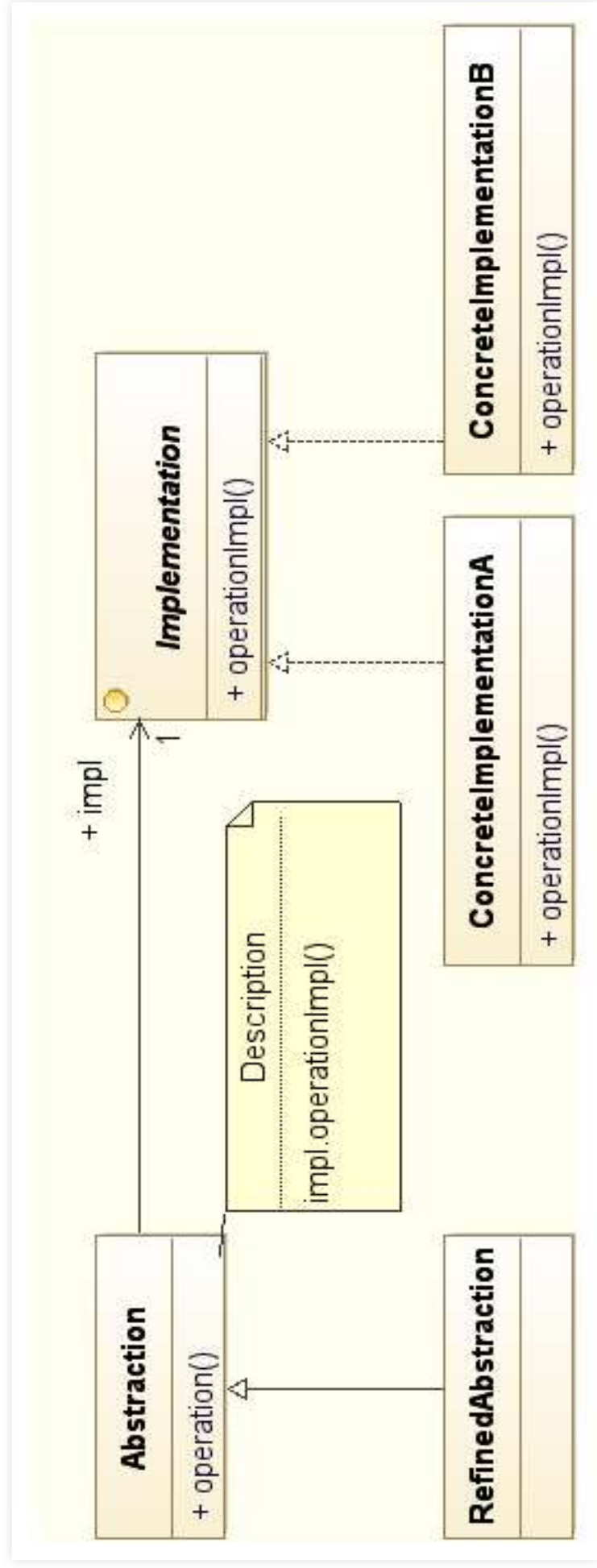
# Primer problema



# Primena *Bridge* obrasca



# Struktura



## Kada koristiti

- Kada želimo izbeći trajno vezivanje apstrakcije i implementacije. Na primer, ako je potrebna izmena implementacije u vreme izvršavanja aplikacije.
- Kada želimo da omogućimo nezavisno proširenje apstrakcije i implementacije kroz nasleđivanje.
- Kada imamo složene hijerarhije klasa kao što je pokazano u primeru.



# Instanciranje konkretne implementacije

Odluka o instanciranju konkretne implementacije:

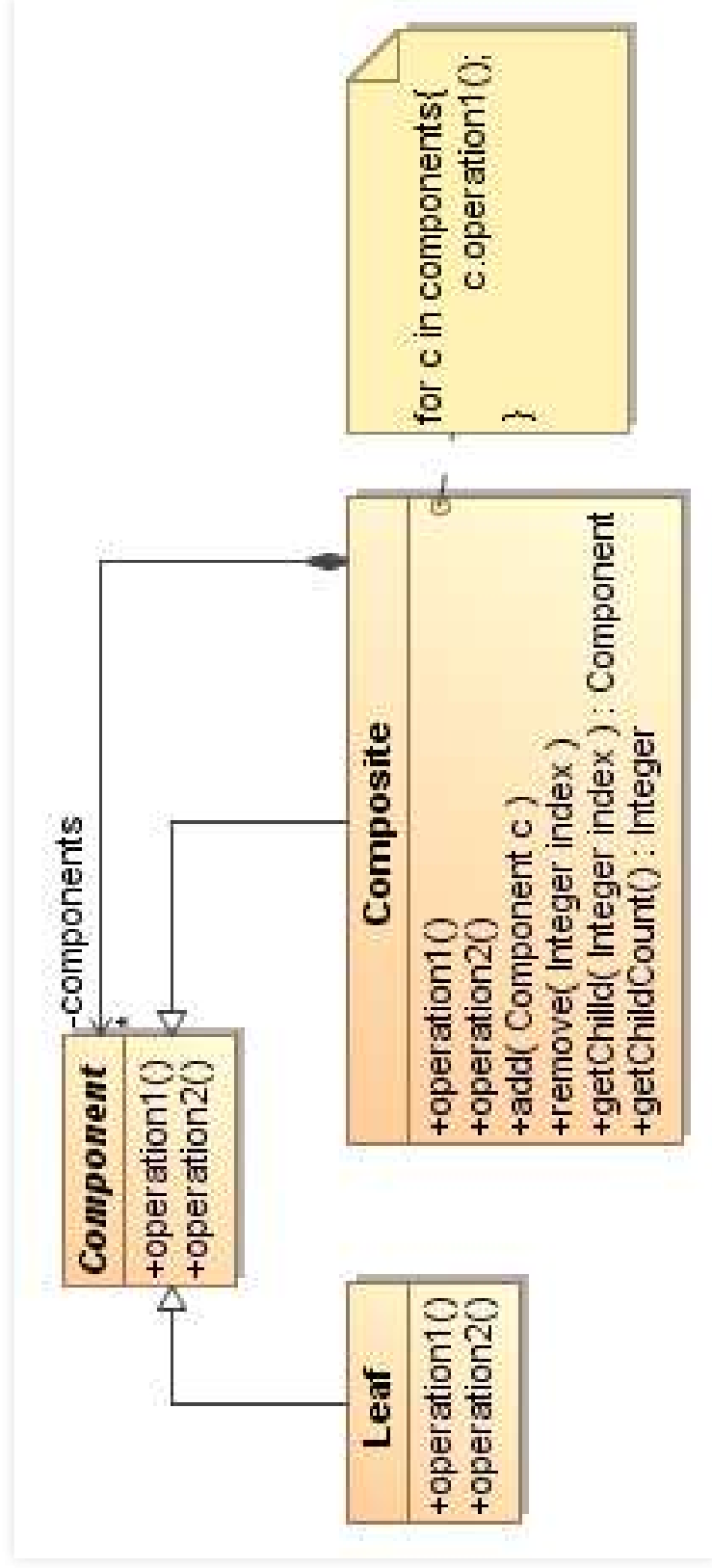
- Konstruktor apstrakcije putem parametra (ako zna za sve implementacije).
- Delegiranje nadležnosti drugom objektu – videti *Factory Method*.

# Composite

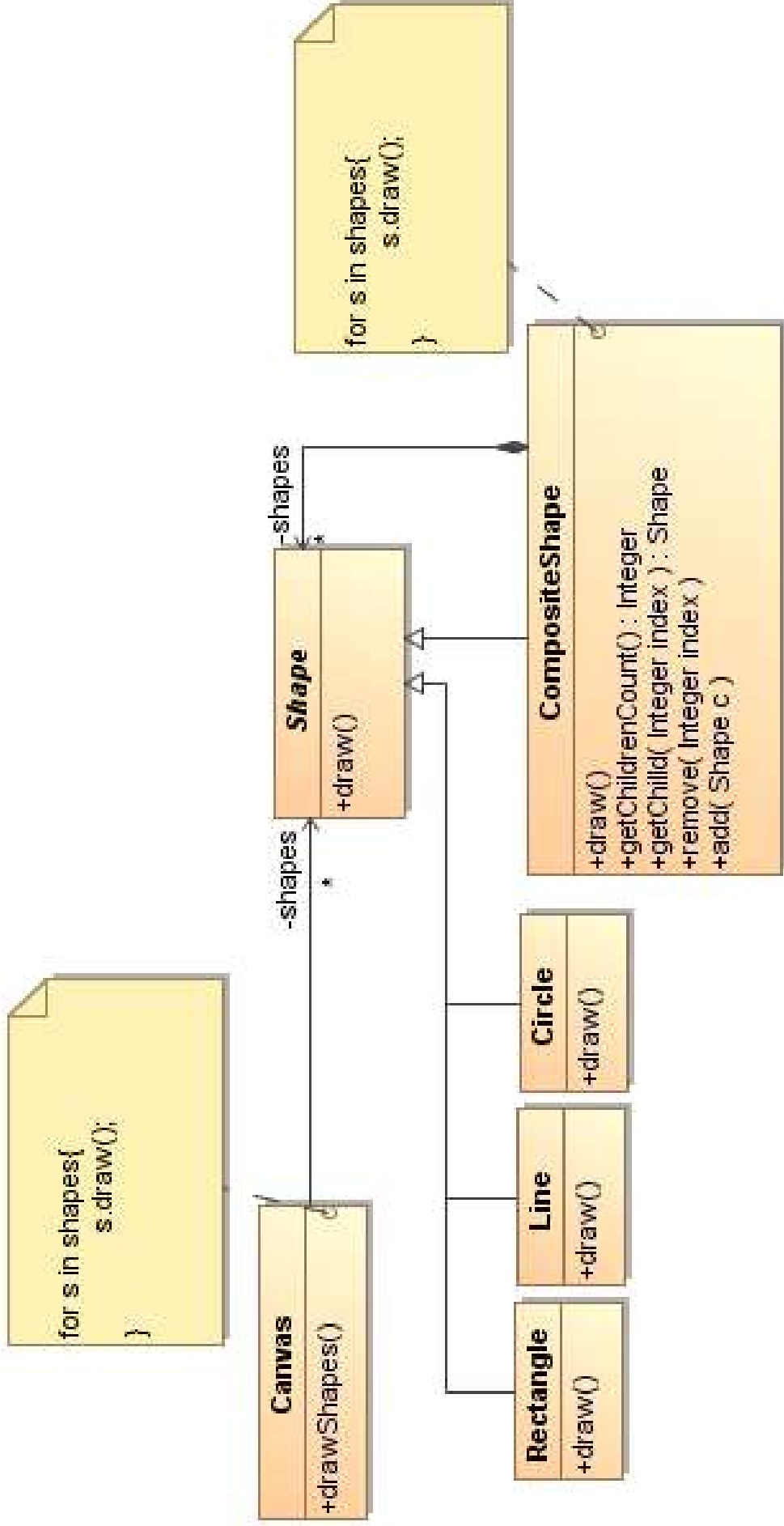
# Composite

- Kompozicija objekata u strukture oblika stabla.
- Uniformno tretiranje prostih i složenih objekata.

# Struktura obrasca



# Primer



# Napomena

GOF knjiga – metode za manipulaciju child elementima u `Component` abstraktnoj klasi.

## Kada koristiti?

- Ako želite da kreirate hijerarhije objekata tipa celina-deo.
- Želite da klijent može da ignoriše razlike između kompozitnih i prostih objekata i tretira ih na uniforman način.

# Demonstracija obrasca *Composite* na primeru u Javi

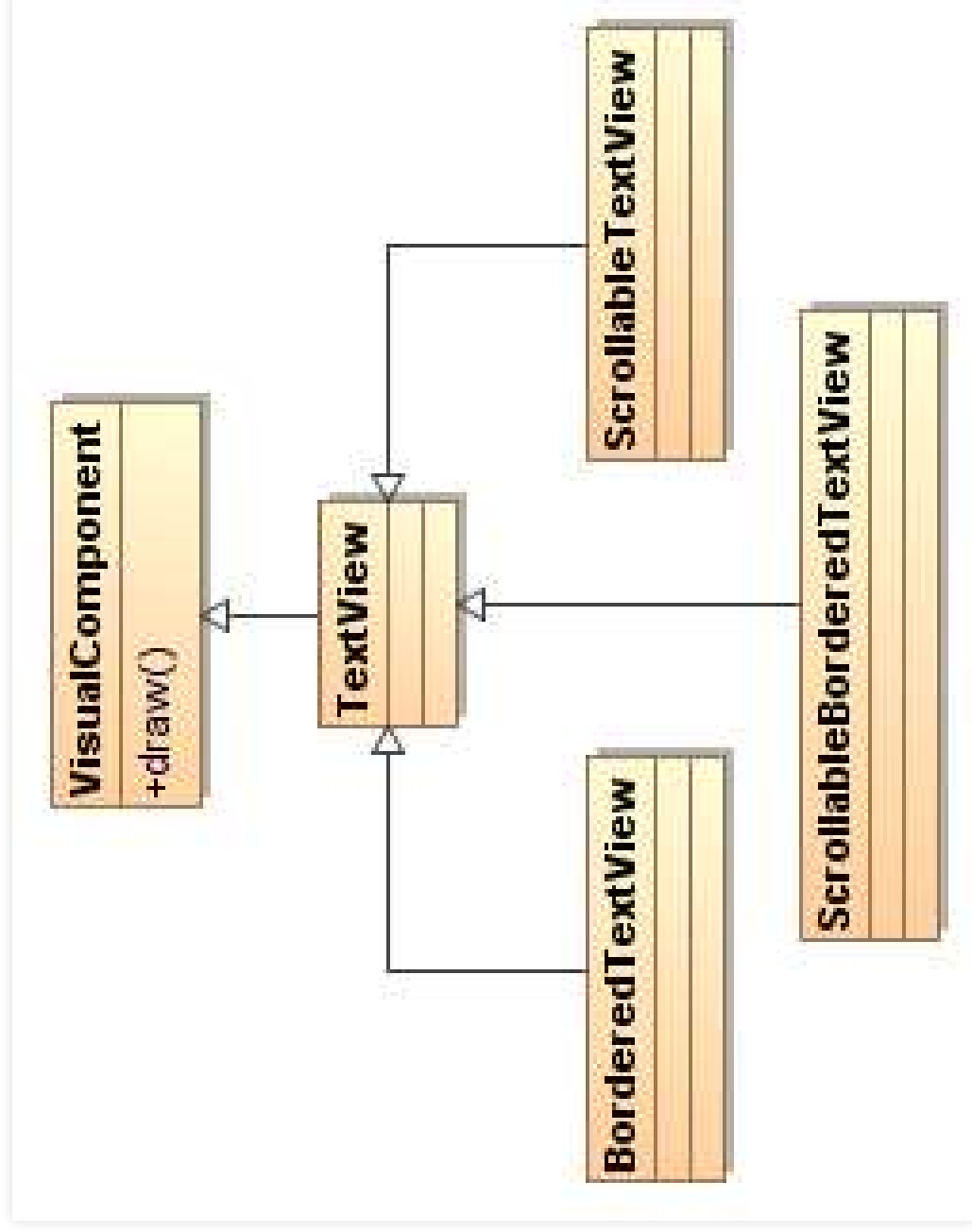


# Decorator

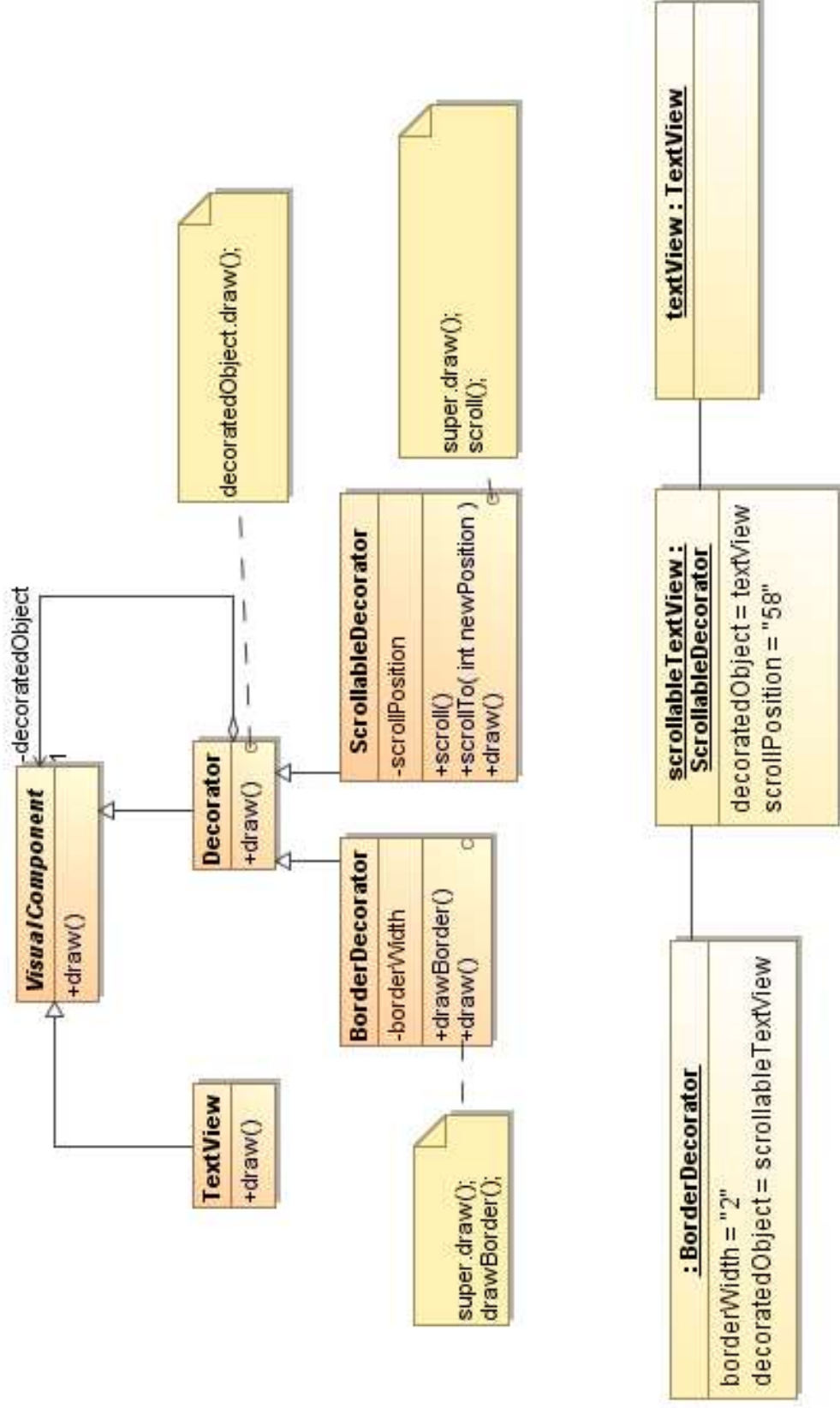
# Decorator

- Dodavanje odgovornosti objektu dinamički.
- Fleksibilna alternativa za nasleđivanje.
- Po strukturi sličan *Adapter* obrascu ali ima drugu namenu.

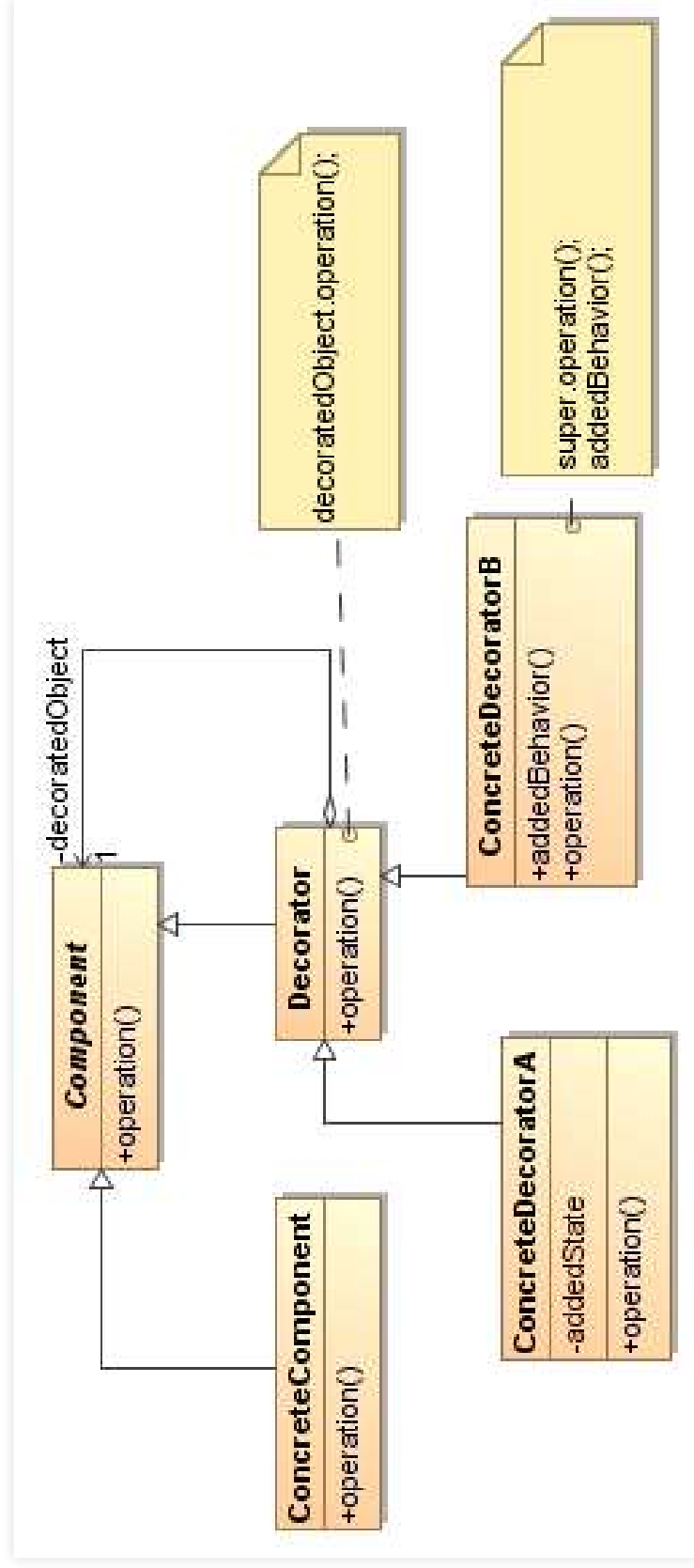
# Motivacija



# Primer



# Struktura obrasca

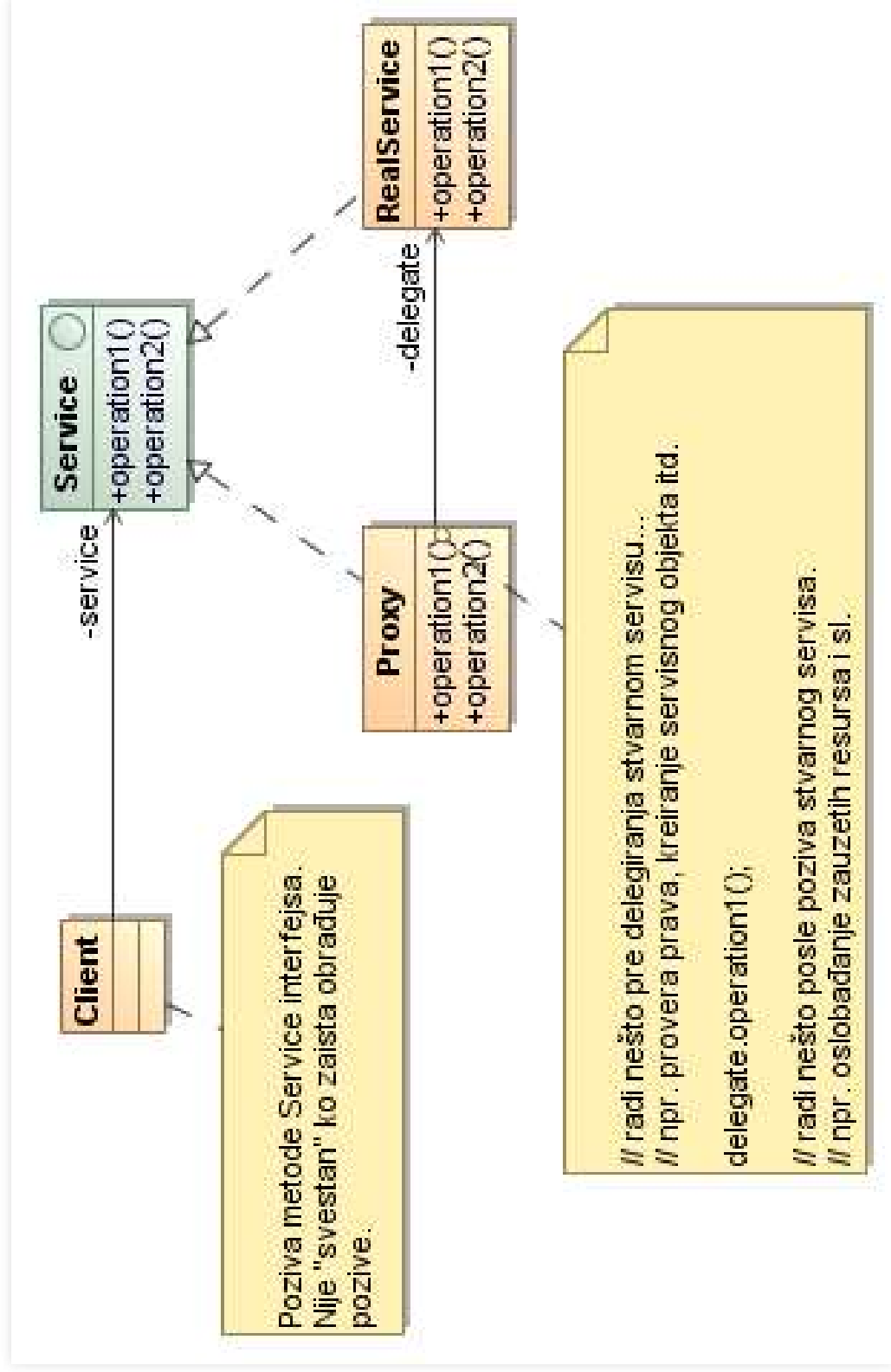


# Proxy

# Proxy

- Posredovanje između objekta koji pruža servis i objekta koji koristi servis u cilju implementacije kontrole pristupa, transparentnog pristupa udaljenim objektima, AOP tehnika i sl.
- Implementacija najčešće delegira pozive metoda objektu koji pruža
- Klijent nije svestan da postoji posrednik između njega i objekta koji obezbeđuje servis.
- Poznat i pod nazivom *Surrogate*.

# Struktura obrasca

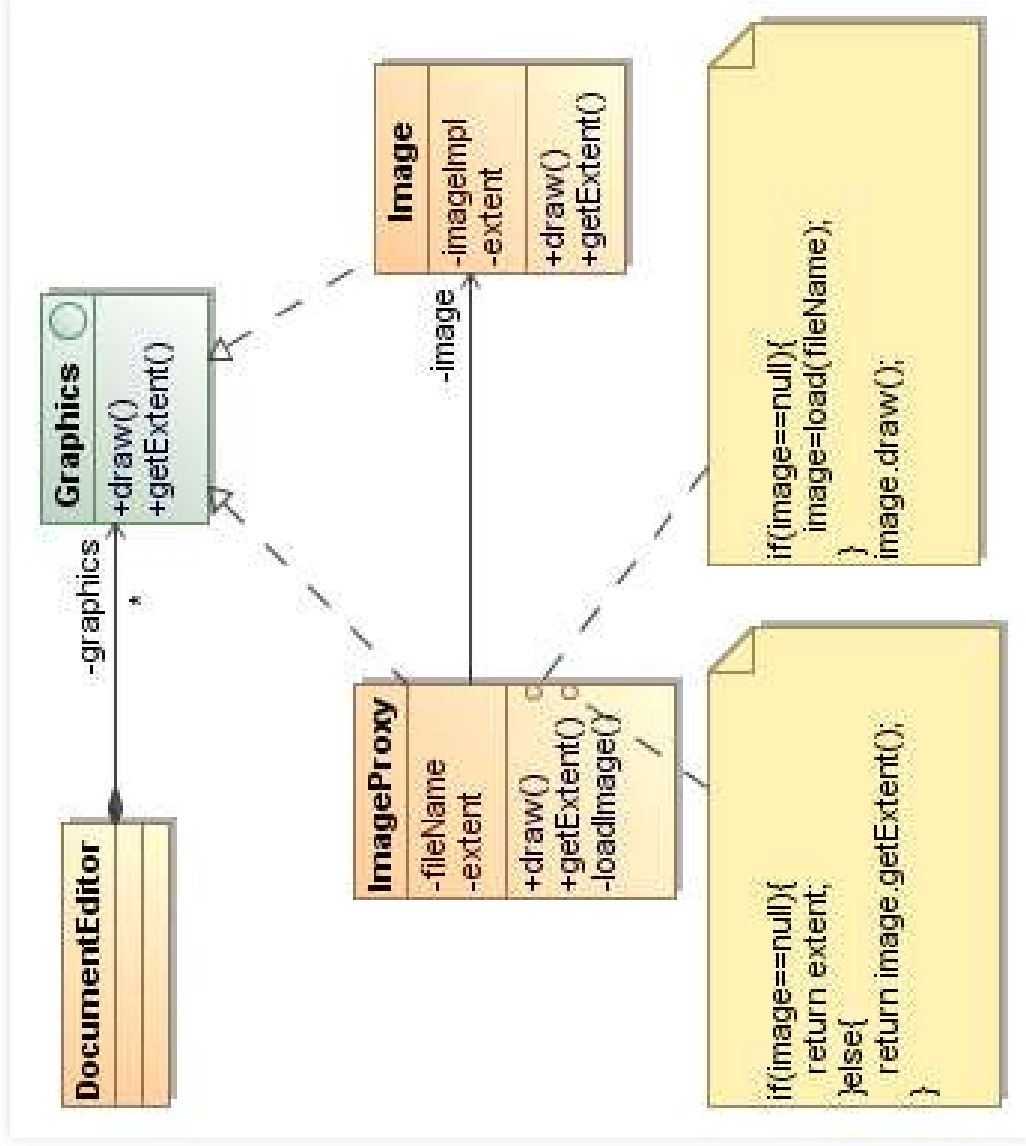




## Kada se koristi?

- Odlaganje skupe inicijalizacije objekta – virtualni proksi (*Virtual Proxy*) ili lenji proxy (*Lazy Proxy*).
- Pristup udaljenom objektu kao da je lokalni (upotrebom tehnologija za distribuirane objekte - CORBA, DCOM, RMI i sl.) – *Remote Proxy*.
- Provera prava pristupa objektu – *Protective Proxy*.
- Sinhronizacija pristupa (proveravanje da li je objekat zaključan prilikom pristupa), brojanje referenci (*smart pointer*), implementaciju AOP tehnika – *Smart Proxy*.

# Primer - *Lazy Proxy*



## *Copy-On-Write* optimizacija

- Posebna vrsta lenjeg ili virtuelnog proksija.
- Odlaganje kopiranja složenog objekta (videti *Prototype* obrazac) za trenutak kada se zatraži operacija modifikacije.
- Potrebno brojanje referenci: svako kreiranje novog proxy-ja uvećava brojč referenci, svako stvarno kopiranje ciljnog objekta umanjuje brojč referenci. Kada brojč referenci postane 0 objekat se može ukloniti.

# Demonstracija obrasca *Proxy* na primeru u Javi

# Literatura

- M. Grand, *Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML*, John Wiley & Sons, Inc., vol. 1, 2002
- E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, 1994