ZAVRŠNA FAZA

Poker

Jana Zdravković 15482

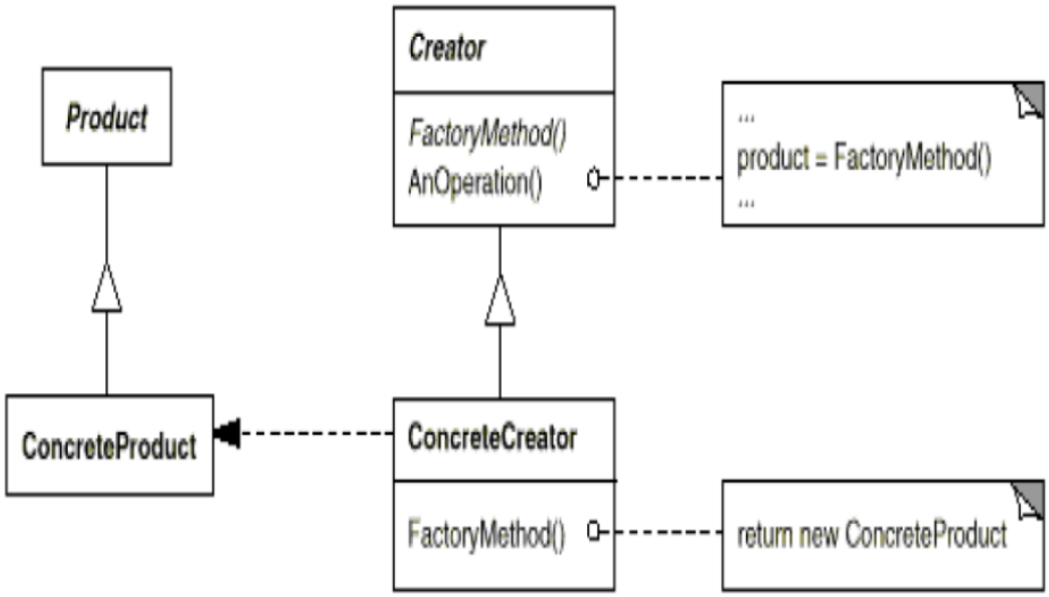
Nikola Ilić 15119

U ovom delu ćemo vam demonstrirati projektne obrasce koje smo koristili za implementaciju projekta ***POKER***.
Obrasci koje smo koristili su: ***factory***,***strategy***,***template method***,***decorator***,***proxy***,***singleton*** i ***observer***.

FACTORY OBAZAC

Definicija obrasca:

Definiše interfejs za kreiranje objekata, ali dozvoljava da izvedene klase odluče koje će klase da instanciraju. Ovaj obrazac omogućava da instanciranje objekata definiše u izvedenim klasama.
Klasa ne može predvideti klasu objekata koje mora da kreira. Klasa želi da njene podklase specificiraju objekte koje kreiraju.



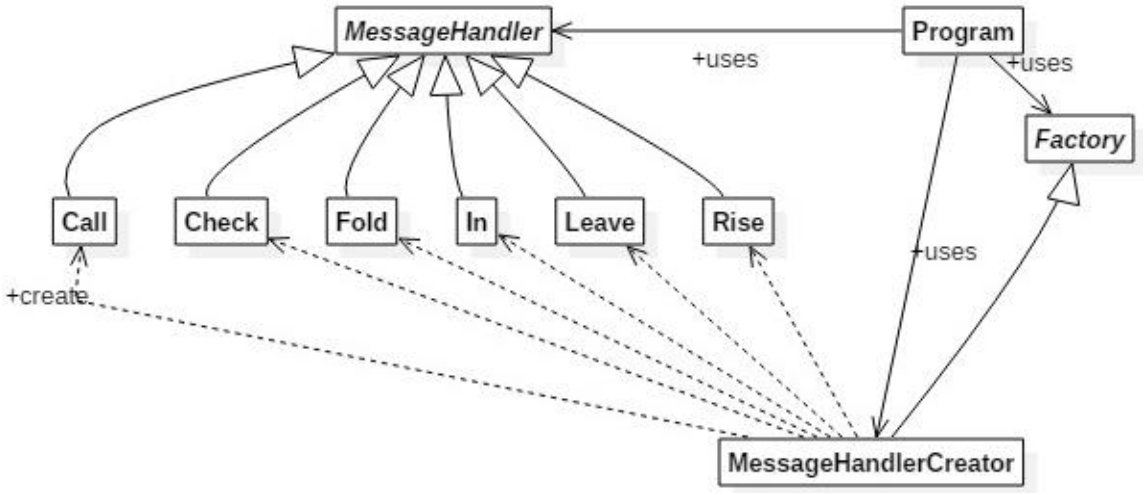
Osnovni klasin dijagram Factory obrasca prikazan je iznad.

Ovaj obrazac koristili smo kako bi smo omogućili da u serveru stola klijentskom kodu ne budu vidljive konkretne klase koje obezbeđuju specifične algoritme za obradu različitih vrsta poruka već samo super klasa tako što je factory zadužen za kreiranje konkretne klase i vraćanje iste kao super klase.Koja klasa će biti kreirana zavisi od toga koji će tip poruke FactoryMethod-u biti prosleđen.

Takođe smo ovaj obrazac koristili kako bi sakrili konkretne loggere od klijentskog koda u serveru stola omogućujući mu da vidi samo njihovu super klasu Logger. Podklasa klase LoggerFactory odlučuje koje će konkretne Loggere kreirati na osnovu konfiguracionog fajla.

Na taj način učinili smo klijentski kod servera stola fleksibilnijim i reupotrebljivijim, učinivši ga nezavisnim od kreiranja konkretnih klasa.

Dijagram klase factory-a koji kreira klase za obradu poruke:



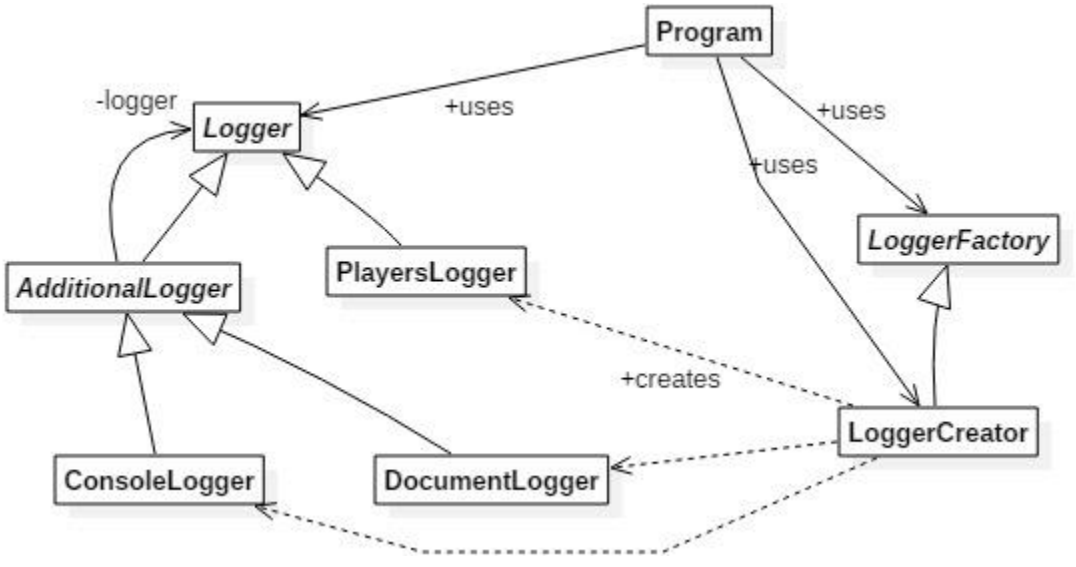
Factory-Creator

MessageHandlerCreator-ConcreteCreator

MessageHandler-Product

Call,Check,Fold...-ConcreteProduct

Dijakram klase factory-a koji kreira klase Logger-a:



LoggerFactory-Creator

LoggerCreator-ConreteCreator

Logger-Product

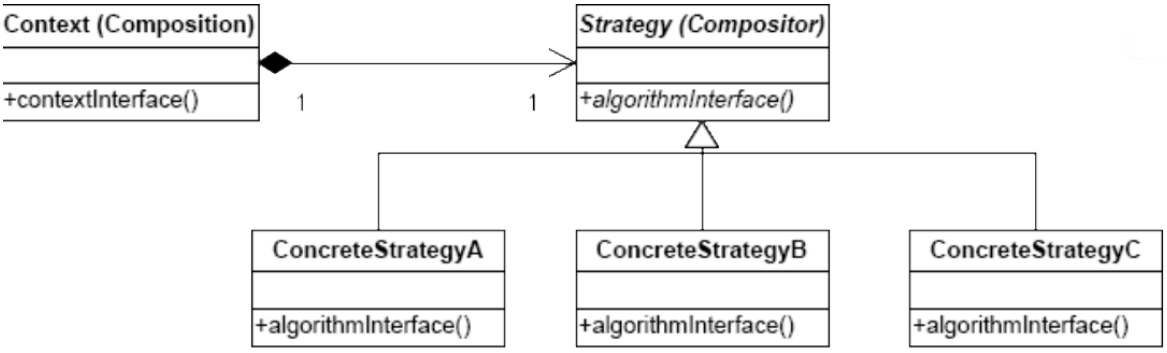
ConsoleLogger,DocumentLogger,PlayersLogger-ConreteProduct

Kao što se sa dijagrama može videti nema modifikacija u odnosu na osnovni Factory obrazac. Sam po sebi obrazac je bio prirodno rešenje za postizanje gore navedenih stavki.

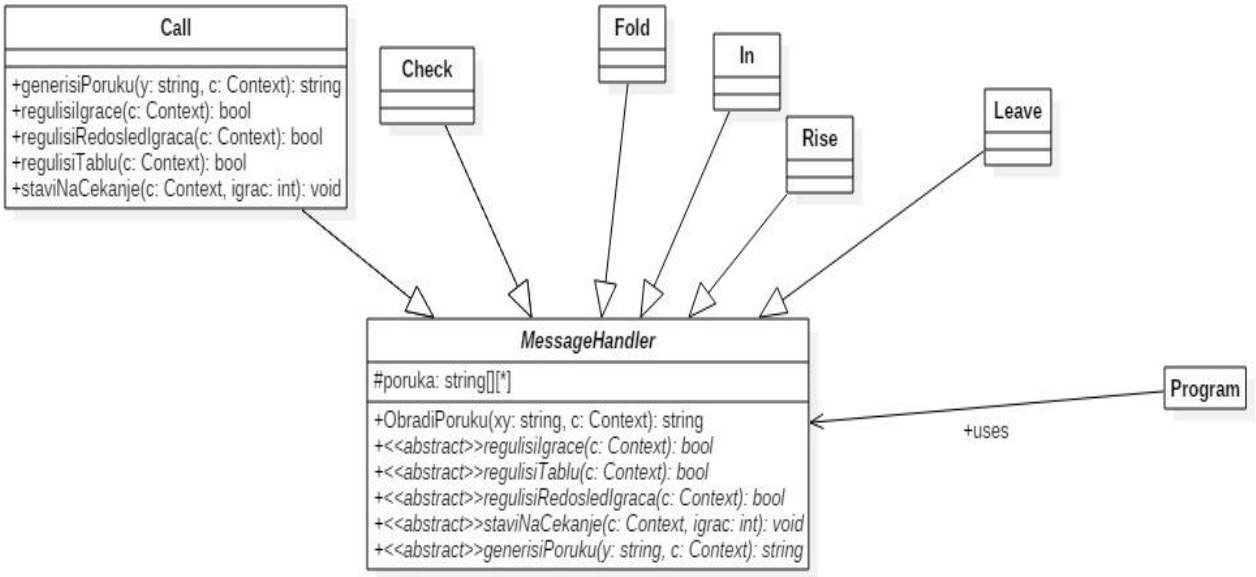
Strategy obrazac

Definiše familiju algoritama, enkapsulira ih i čini ih zamenljivim, tako da se algoritmi mogu menjati nezavisno od klijenata koji ih koriste.

Primenjiv je kada: Postoji više povezanih klasa koje se razliku samo u svom ponašanju, neophodno je postojanje više različitih verzija algoritma, kada se u okviru klase definiše više različitih ponašanja i ta ponašanja se manifestuju kao višestruke uslovne naredbe u okviru njenih operacija, tada se ta ponašanja definišu kao posebni algoritmu u okviru enkapsulirani u odgovarajuće Strategy klase, a jedinstven interfejs pokriva sve algoritme (enkapsulacije).



Koristili smo Strategy obrazac da enkapsuliramo skup algoritama za obardu različitih tipova poruka u klasu MessageHandler i kako bi smo izbegli ogroman broj naredba uslovnog grananja u serveru. Server vidi i koristi samo interfejs klase koji mu nudi MessageHandler klasa.



Dijagram2

Program-Context

Call-StrategyA

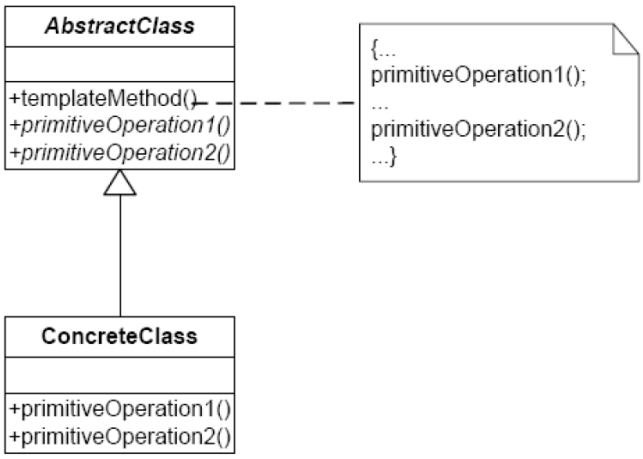
Check-StrategyB

...

Leave-StrategyF

Template Method

Definiše kostur algoritma u okviru određene operacije, ostavljajući da se određeni koraci u algoritmu definišu u podklasama. Primenljivost: Implementiranje nepromenjlivih aspekata algoritma i ostavljanje da se promenljivi delovi implementiraju u podklasama. Lokalizovanje generalnog ponašanja u okviru jedne klase da bi se povećala reupotrebljivost koda. Kontrolisanje ekstenzija koje mogu da se obave u podklasama samo kroz “hook” operacije koje se pozivaju u okviru template metoda.



Dijagram2 demonstrira i primenu TemplateMethod obrasca.

MessageHandler-AbstractClass

Call,Check,Fold,Rise...-ConcreteClass

templateMethod-ObradaPoruke

primitiveOperation1-regulisiIgrace

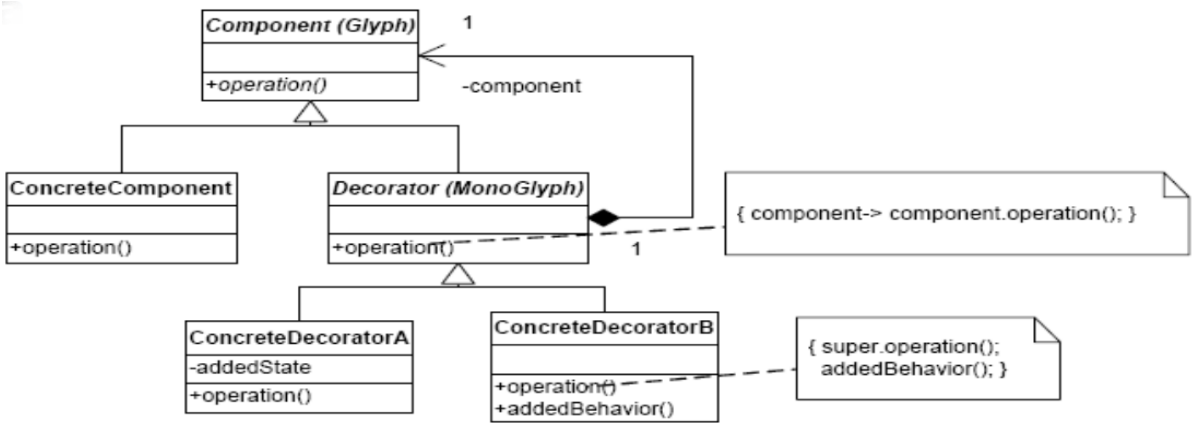
primitiveOperation2-regulisiTablu ...i za sve ostale metode važi isto

TemplateMethod smo koristili kako bi maksimalno umanjili duliranje koda po klasama koje specificiraju konkretne algoritme za obradu poruke s obzirom na to da je veliki deo svih algoritama za to isti ali ima bitnih razlika. Kostur je definisan metodom ObradaPoruke a u njoj se nalaze konkretne f-je koje su ostavljene da ih neki od konkretnih klasa-konkretan algoritam strategy-a predefinise na sebi svojstven način.

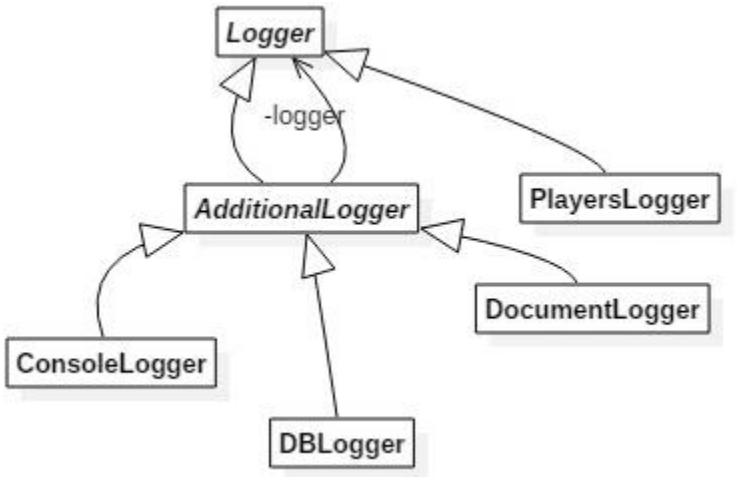
Iz dijagrama klase se vidi da nema modifikacija u odnosu na osnovne definicije obrazaca s tim što smo implementirali templateMethod nad Strategy obrascem.

Decorator

Dinamičko dodavanje novih obaveza (funkcionalnosti) objektima. Primenljivost: Kada je nepraktično proširenje funkcionalnosti primenom nasleđivanja klasa, dinamičko i transparentno dodavanje novih funkcionalnosti objektima bez uticaja na druge objekte.



Dijagram klase dekoratora iz našeg programa:



Logger-Component

AdditionalLogger-Decorator

PlayersLogger-ConcreteComponent

ConcreteDecoratorA-ConsoleLogger

ConcreteDecoratortB-DocumentLogger

ConcreteDecoratorC-DBLogger

PlayersLogger predstavljaju bazičnu funkcionalnost odnosno omogućavaju da se potezi preko mreže šalju svim igračima na stolu omogućujući na taj način da stanje na stolu svih igrača bude konzistentno,bez toga igra ne bi imala smisla. Međutim funckionalnosti kao što su upisivanje poteza u

bazu,dokument ili ispisivanje na konzolu radi debagiranja nisu i one su dodatne i na osnovu konfiguracionog fajla mogu da se isključe ili uključe-nisu obavezne. TO omogućava koa što se vidi sa dijagrama dekorator obrazac. Koristili smo ga kako bi izbegli eksploziju podklasa klase Logger-sve kombinacije i kako bi obezbedili dinamičku promenu funkcionalnosti.

Sa dijagrama se vidi da nema modifikacija u odnosu na definiciju obrasca.

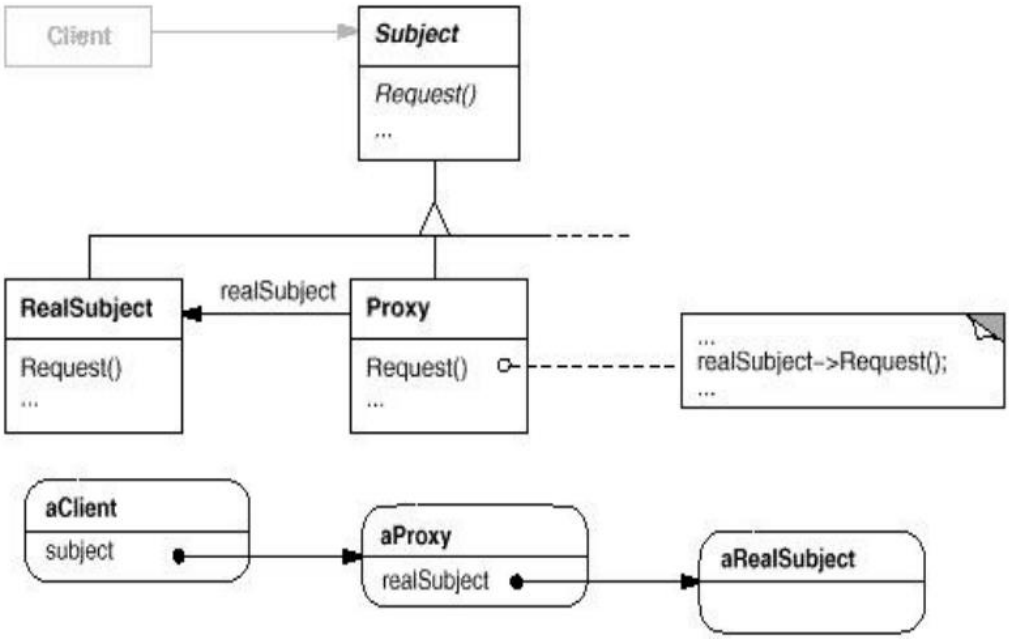
Proxy obrazac

Vodi računa o referenci koja proxy dozvoljava pristup ka RealSubject-u. Obezbeđuje interfejs identičan interfejsu Subject-a. Vodi računa o pristupu RealSubject-u i može biti odgovoran za kreiranje i brisanje istog. Ostale odgovornosti zavise od vrste proxy-a:

• Remote proxy obezbeđuje lokalnu reprezentaciju objekta koji se nalazi u drugom adresnom prostoru. Odgovoran je za kodiranje zahteva i njegovih argumenata, kao i za slanje ovako kodiranog zahteva realnim objektima (RealSubjects) koji se nalaze u različitim adresnim prostorima. Ovo je kod nas slučaj pošto bi nam ulogu Proxy-a imao Master server koji vodi brigu o ServerSto-lu koji se nalazi u drugom adresnom prostoru ali se ne prenose poruke preko njega već se diretno razmenjuju između klijenta i stoservera međutim samo između onih klijenata kojiima master server omogući to i to bi bila modifikacija.

• Virtual proxy kreira “skupe” objekte na zahtev; čuva dodatne informacije o realnim objektima tako da se može smanjiti učestanost obraćanja. Kod nas Master server kreira ServerSto ukoliko do tada nije bilo igrača na njemu i gasi ga ukoliko više nema istih na njemu kako bi čuvao resurse.

• Protection proxy kontroliše pristup originalnom objektu; vodi računa o tome da li objekat koji ga referencira ima dovoljan nivo privilegija da bi se prosledio zahtev. Kod nas Master server ne dozvoljava uvek igračima da pristupe stolu već samo ukoliko ima dovoljno slobodnih mesta na njemu.



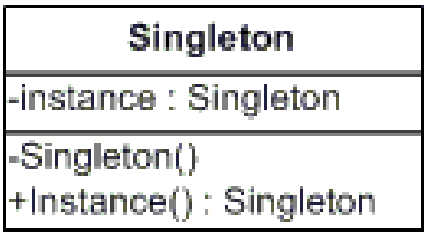
Client-Klijenti koji igraju poker (klijentski kod)

RealSubject-StoServer

Proxy-MasterServer

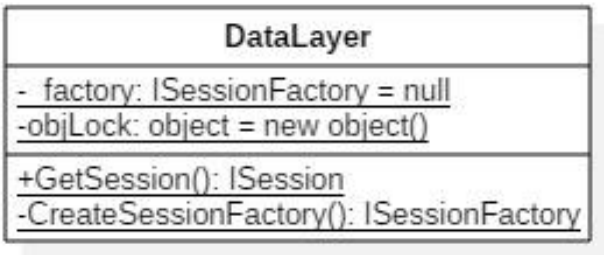
Singleton

Obezbeđuje da određena klasa ima maksimalno jedan aktivni objekat, i obezbeđuje da on bude dostupan.



Definiše operaciju Instance koja obezbeđuje korisniku pristup jedinom kreiranom objektu .

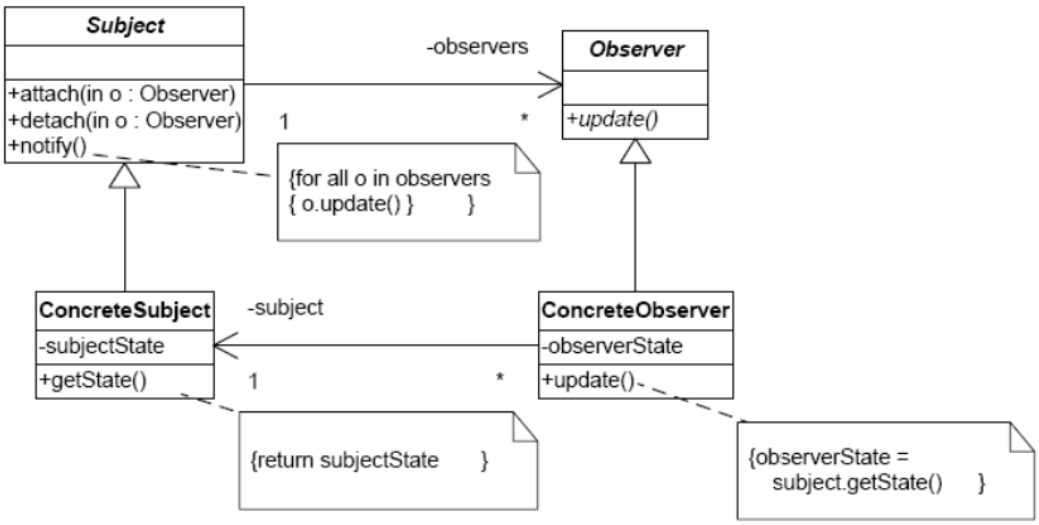
Definiše konstruktor koji je odgovoran za kreiranje i održavanje svog jedinog objekta .

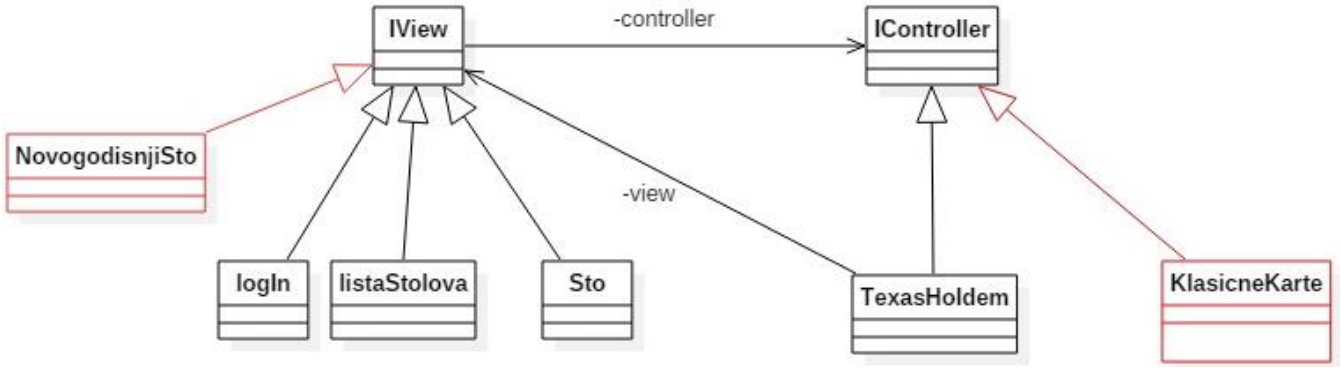


Vidimo sa dijagrama da bi _factory privatni clan odgovarao Singleton promenljivoj kojoj ograničavamo broj instanci na jednu. Funkcija GetSession bi odgovarao funkciji Instance i ona prilikom poziva proverava da li je referenca _factory null i ukoliko jeste poziva CreateSessionFactory privatnu funkciju kojoj samo ona može da pristupi i kreira instancu tipa SessionFactory i omogući da _factory pokazuje na nju a zatim vrati _factory. CreateSessionFactory odgovara funkcii Singleton. Ako _factory nije null on ga samo vrati i na ovaj način ne dozvoljava ponovno kreiranje Session-a jer je to skupa operacija.

Observer

Definiše jedan-na-više zavisnost između objekata tako da kada jedan objekat promeni svoje stanje, svi objekti koji zavise od njega se obaveštavaju o tome i automatski ažuriraju.Služi za održavanje konzistentnosti između povezanih objekata.





Observer smo implementirali na gore predstavaljen način.

Iview-Subject

ConcreteSubject-logIn,listaStolova,Sto...

Icontroller-Observer

ConcreteObserver-TexasHoldem,KlasicneKarte...

Modifikacije na osnovu definicije Observera bi bile to što naš subject može da ima samo jednog observera ali ima smisla ovako implementirati ukoliko hoćemo kasnije da menjamo controllere (korisnici traže neke nove tipove pokera-Klasičan poker na primer) i to bi se postiglo jednostavnim implementiranjem novog kontrolera sa novim načinom računanja poena i sl i priključio bi se na view. View bi takođe mogo lako da se menja nezavisno od observera samo je potrebno da se na njega taj observver priključi. Na taj način možemo da pravimo i nadograđujemo korisnički interfejs (npr. Novogodišnji izgled stola) bez većih promena u postojećem kodu.