FAKULTET ORGANIZACIONIH NAUKA
UNIVERZITET U BEOGRADU

Seminarski rad iz predmeta
Softverski paterni

Tema:

Primena softverskih paterna na primeru turističke agencije „World travell“

Profesor: Dr Siniša Vlajić

Student: Ivana Joković 84/09

Asistenti: Dušan Savić
Vojislav Stanojević

Beograd, januar 2013.

SADRŽAJ

1. UVOD

Paterni su svuda oko nas. Mogu se naći u prirodi, građevinama, ljubavnim romanima, ali i u softveru. Ako posmatramo život uočavamo da se on sastoji od mnoštva procesa. Da bismo izbegli haos, treba da upravljamo svim tim procesa. U tome nam pomažu paterni.

Kristofer Aleksandar kaže:

„*Svaki patern opisuje problem koji se stalno ponavlja u našem okruženju i zatim opisuje suštinu rešenja problema tako što se to rešenje može upotrebiti milion puta, a da se dva puta ne ponovi na isti način*“.

U ovom seminarskom radu će se na 6 različitih paterna, iz različitih kategorija (kreacioni, strukturni i paterni ponašanja), objasniti osnovna zamisao i svrha paterna, kao i njihov veliki značaj u programiranju uopšte. Svaki od paterna će biti detaljno predstavljen, a kroz primer turističke agencije i klijenta koji vrši kupovinu odnosno rezervaciju aranžmana pokazaće se i konkretan kod napisan u programskom jeziku Java.

2. OSNOVE O PATERNIMA

Svrha paterna?

Paterni odnosno uzori imaju cilj da nam pomognu u održavanju i nadogradnji softverskih sistema.

Šta je patern?

Rešenje nekog problema, u nekom kontekstu, koje se može ponovi iskoristiti za rešavanje nekih novih problema.

Iz toga sledi da u definisanju paterna učestvuju tri elementa:

- *Problem*
- *Rešenje*
- *Kontekst*

Paterni, odnosno uzori projektovanja su **iskustva** u projektovanju objektno orijentisanih softvera.

Jedno od osnovnih svojstava paterna jeste njegova mogućnost da se može primeniti u rešavanju različitih problema.

To su *gotova rešenja* koja se primenjuju u projektovanju softvera. Oni predstavljaju opise komunikacija između objekata,klasa, koji su prilagođeni da reše generalni problem u posebnom kontekstu. Kada se desi neki problem i nađe nejgovo rešenje, potrebno je uložiti još dodatnog napora i vremena i od specifičnog rešenja napraviti generičko rešenje koje se može primeniti na različite probleme koji se dešavaju u životu. Paterni pomažu u imenovanju i opisu generičkih rešenja koja se mogu primeniti u različitim problemskim situacijama. Pored toga, paterni identifikuju klase i pojavljivanja, njihove uloge i saradnju i raspodelu odgovornosti.

Najbolji način da koristimo softverske paterne jeste da se što bolje upoznamo sa njima, da počnemo da prepoznajemo mesta u našim aplikacijama gde možemo da ih upotrebimo. Umesto da se mučimo ili kopiramo kod , uz pomoć paterna mi kopiramo tuđa iskustva.

Za uspešno razmatranje i rešavanje bilo kakvog problema, neophodno je prvo razumeti njegovu suštinu, pa tek onda preći na projektovanje i implementaciju rešenja.Ako se želi fleksibilnost i mogućnost dinamičke izmene funkcionalnosti programa, paterni su idealni, ali oni pak povećavaju složenost sistema.

Svaki patern ima*:* ime, problem, rešenje i posledice.

- ***Ime*** paterna se koristi da bi se u nekoliko reči opisao problem, njegova rešenja i njegove posledice.

- ***Problem*** opisuje situaciju u kojoj koristimo patern. Opisuje se problem i njegov kontekst.

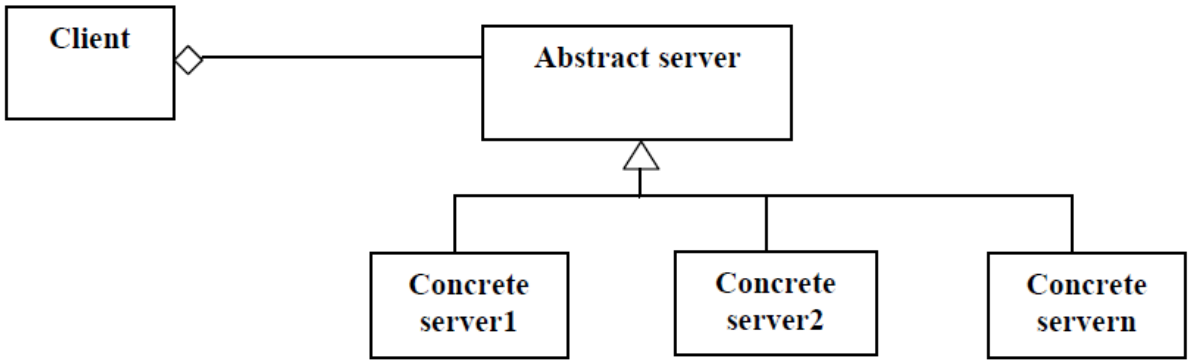
- ***Rešenje*** opisuje elemente koji čine dizajn, njihove odnose, raspodelu odgovornosti kao i pravila saradnje. Ono ne opisuje određen konkretan projekat ili implementaciju, pošto je patern šablon koji se može primeniti u mnogim različitim situacijama. Dakle, patern daje apstraktan opis problema projektovanja kao i smernice kako se on rešava opštim uređenjem elemenata.

- ***Posledice*** su rezultati i ocene primene uzorka koje su bitne za procenu alternativa i za razumevanje gubitaka i dobitaka od primene uzorka.

Opšti oblik paterna

Struktura koja omogućava da softverski system bude održiv, i koja postoji kod 20 od 23 GOF(Gang Of Four) paterna projektovanja, je ključni mehanizam ili svojstvo paterna projektovanja i ona omogućava lako održavanje programa, njegovu nadogradnju i prilagođavanje korisničkim zahtevima

.



Klijent nije vezan za konkretni server, već se uvodi apstraktni server. Neodrživa struktura se prevodi u održivu. Klijent je vezan za apstraktni server, iz koga su nasleđivanjem, odnosno implementacijom, izvedeni konkretni server. Apstraktni server može biti apstraktna klasa(abstract class) ili interfejs(interface). Klijent se u vreme kompajliranja vezuje za apstraktni server, a tek u vreme izvršavanja zna koji konkretni server će da realizuje njegov zahtev. Upravo ovo kasno povezivanje (late binding) u vreme izvršavanja programa, a ne u vreme kompajliranja, daje samom programu fleksibilnost, da jedan klijentski zahtev može biti realizovan na različite načine, preko različitih konkretnih server. Dodavanje novog konkretnog server ne zahteva promenu klijenta, što je još jedna velika prednost ove strukture.

Paterni su podeljeni u tri grupe:

1.Uzori za kreiranje objekata apstrahuju proces instancijalizacije, tj. kreiranja objekata. Daju veliku prilagodljivost u tome šta će biti kreirano, ko će to kreirati, kako i kada će biti kreirano. Oni pomažu da se izgradi sistem nezavisno od toga kako su objekti kreirani, komponovani ili reprezentovani.

2.Strukturni uzori opisuju složene strukture međusobno povezanih klasa i objekata.

3.Uzori ponašanja opisuju način na koji klase ili objekti sarađuju i raspoređuju odgovornosti.

Konkretno, u ovom primeru turističke agencije, biće prikazano 8 paterna, a to su: AbstractFactory, Builder, Adapter, Decorator, Proxy, Observer, Template, Singletone.

1.Kreacioni	2. Strukturni	3.Uzori ponašanja
AbstractFactory	Adapter	Chain Of Responsibility
Builder	Bridge	Command
FactoryMethod	Composite	Mediator
Prototype	Decorator	Memento
Singleton	Fasade	Observer
	Flyweight	State
	Proxy	Template

3.PROBLEM

Klijent dolazi u turističku agenciju „World travell“ da izabere aranžman za svoj naredni odmor. Turistička agencija ima u ponudi i letovanja i zimovanja. Klijent na početku bira gde će biti smešten. * Agencija u ponudi ima hotele i vile. Klijentu se takođe daje izbor da li želi da putuje avionom ili autobusom. Nakon što klijent sastavi sam svoju želju, saopštava je zaposlenom u agenciji, i na osnovu njegove želje traže se moguće turističke ponude odnosno aranžmani koji odgovaraju tim zahtevima.

Klijent dolazi u turističku agenciju „World travell“ da izabere aranžman za svoj naredni odmor. Turistička agencija ima u ponudi i letovanja i zimovanja. Osnovni kriterijum po kome stranka bira aranžman je smeštaj. Agencija u ponudi ima hotele i vile. Klijentu se takođe daje izbor da li želi da putuje avionom ili autobusom. Stranka u agenciji naglašava da su joj to bitne karakteristike (vreme, smestaj i prevoz) i na osnovu njih dobija turističku ponudu odnosno aranžman koju zaposleni u agenciji sastavlja.

Nakon što je pogledao sve ponude, klijent se odlučio da tekuće godine ide na letovanje. Ipak, odlazi da još malo razmisli, ali napominje da će se vratiti da rezerviše a nakon toga i uplati novac za ponudu koja mu se najviše dopala. Međutim, po povratku saznaje da su svi kapaciteti popunjeni, i da nema vise slobodnih mesta u tom turističkom aranžmanu. Agencija “World travel” uspešno sarađuje sa drugom agencijom “Happy”, i obe vode turiste na iste lokacije i u isto vreme. Zaposleni u agenciji objašnjava identičnu ponudu agencije “Happy", dok klijent kod sebe ima katalog svih aranžmana agencije “World travell” (Subject). Klijent gleda u katalogu opis ponude letovanja koja mu se dopala (Proxy), dok zaposleni u agenciji poziva drugu agenciju I traži da se rezerviše ponuda za letovanje za njegovog klijenta (Real Subject).

Svaka stranka prilikom rezervacije putovanja može da izabere još neke dodatne komponente kojima bi proširila svoju turističku ponudu. Agencija stranke vodi na razne izlete, u posete najpoznatijim muzejima država u koje se putuje, a pored toga nudi turistima da uplate dodatni novac i za korišćenje usluga Spa centra. Ako se turista odluči za izlet, treba da doplati još 4000 dinara, za posetu muzeju 2000 dinara, dok za neograničeno korišćenje Spa centra plaća dodatno još 1500 dinara.

Stranka dolazi u turističku agenciju ne bi li rezervisala ponudu za svoje putovanje. Međutim, i dalje je u dilemi da li želi tekuće godine da ide na letovanje ili na zimovanje. Stranka sa sobom ima katalog koji je dobila prošli put kada je dolazila u agenciju, i sada traži od zaposlenog u agenciji da joj detaljnije objasni aranžmane koji su u ponudi, i na taj način joj pomogne da izabere aranžman. Zaposleni pored toga što u katalogu piše da se na letovanje putuje avionom, mušteriji objašnjava ko je prevoznik, a za zimovanje pored šturog opisa u katalogu da se putuje autobusom, navodi takođe ko je prevoznik, da su autobusi klimatizovani i na dva sprata. Stranku zanima i gde će biti smeštena, te joj zaposleni daje detaljnije informacije o vilama, odnosno hotelima, kao što su: koliko kreveta u sobi ima, da li terasa ima pogled na more, koliko je vila udaljena od plaže, sa koliko zvezdica je hotel itd. Pored toga, zaposleni objašnjava stranci da aranžman može da plati na rate i koja je u tom slučaju visina mesečne rate.

Turistička agencija pored ponuda za fizička lica, ima i specijalne ponude, odnosno turističke aranžmane za državne ustanove, kao što su fakulteti, škole i vrtići. Kataloge sa turističkim ponudama zaposleni u agenciji redovno šalje ustanovama, ne bi li one na vreme bile obaveštene koje su trenutno aktuelne ponude, koje su cene, da bi imale vremena da odluče na koju lokaciju da vode decu na rekreativnu nastavu, odnosno đake i studente na ekskurzije. Svaki put kada se u katalog unese nova turistička ponuda, ili ako se neka ponuda izbaci za narednu sezonu, sve ustanove o tome bivaju obaveštene. Ukoliko se škola, fakultet ili vrtić odluči za neki od aranžmana agencije, dobiće odgovarajući popust na određen broj dece, đaka, odnosno studenata. Učenici škola mogu svoju ekskurziju da plate na 10 rata, i ukoliko ih se prijavi više od 200, imaće popust od 10 % po učeniku. Studenti ekskurziju plaćaju na 7 rata, i ukoliko ih je više od 100 prijavljenih, dobijaju popust od po 5%. Roditelji rekreativnu nastavu za decu mogu da plate na 7 rata, i ukoliko bude više od 300 prijavljene dece, dobiće popust u iznosu od 15%.

Svaki aranžman turističke agencije ima tačno definisanu rutu putovanja. Ruta putovanja započinje polaskom, zatim sledi dolazak na željenu destinaciju, i na kraju povratak u Beograd. Agencija organizuje putovanja tako što na početku odredi vremenski okvir, kad se kreće, za koliko se stiže na određenu lokaciju, i kada je planiran povratak u Beograd. Svaka ponuda se po tome razlikuje, odnosno svaka ponuda ima drugačije vreme polaska i povratka. Ukoliko agencija poseduje ponudu za letovanje u Italiji, onda samim tim ima i 3 različite rute putovanja, u zavisnosti od toga kojim se prevoznim sredstvom ide na putovanje. Ukoliko se putuje autobusom kreće se ranije, i vraća kasnije, jer je putovanje znatno duže nego avionom. Pored toga, agencija turiste može da vodi na putovanja i minibusom, za koji takođe definiše posebnu rutu putovanja, u skladu sa vremenom putovanja.

4. KREACIONI PATERNI

4.1. ABSTRACT FACTORY

Apstrakcija nam pruža zanemarivanje detalja, odnosno kontrolisano uključivanje detalja (ignorisanje svih svojstava subjekta koja nisu relevantna za tekuću svrhu sa ciljem koncentrisanja samo na ona koja to jesu). Olakšava nam da u bitnim trenucima brinemo o celini, a da brigu o detaljima ostavimo za kasnije. Upravo, Abstract Factory nam omogućava klase koje će se pobrinuti o kreiranju delova željenog objekta umesto nas. Ovaj interfejs prenosi odgovornost za kreiranje objekata do njegovih ConcreteFactory podklasa. Abstract Factory patern treba koristiti kada se upravlja stvaranjem grupe međusobno zavisnih objekata. Treba ga koristiti kada sistem ne sme da zavisi od toga kako se njegovi objekti prave, sastavljaju i predstavljaju. Klijent ne zna, odnosno ne vodi brigu o tome koji proizvod je iz koje od ovih klasa, posto radi sa njihovim apstraktnim predstavama. Klijent kreira ponudu preko apstraktnog interfejsa i ne vidi kako konkretne klase kreiraju objekte.

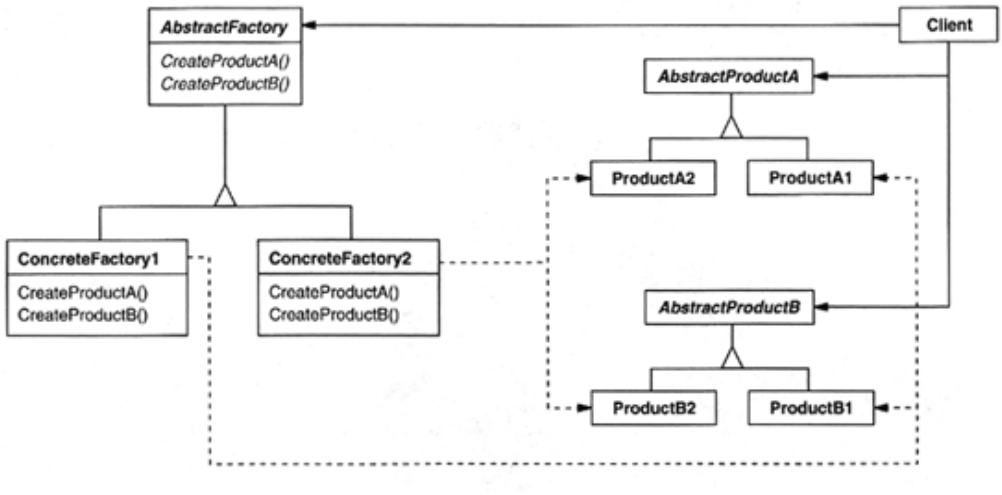
4.1.1.Definicija

Obezbeđuje interfejs **(*AbstractFactory*)** za kreiranje ***(CreateProductA(), CreateProductB())*** familije povezanih ili zavisnih objekata **(*AbstractProductA, AbstractProductB*)** bez navođenja (specificiranja) njihovih konkretnih klasa ***(ProductA1, ProductA2, ProductB1, ProductB2).***

Klijent nadzire proces pravljenja i izrađuje proizvod u celini. Izrada delova proizvoda se prenose na ConcreteFactory klase.

1. Klijent- stranka je odgovoran za kontrolu kreiranja ponude.
2. Server je zadužen za pravljenje elemenata ponude
3. Klijent je odgovoran za pravljenje proizvoda u celini

4.1.2. Struktura



Učesnici:

- **Abstractfactory**

Deklariše interfejs operacije koje kreiraju proizvode

- **ConcreteFactory**

Implementira operacije AbstractFactory interfejsa, kojima se kreiraju konkretni proizvodi

- **AbstractProduct**

Deklariše interfejs za proizode

- **ConcreteProduct**

Definiše proizvode koji će biti kreirani preko ConcreteFactory klasa

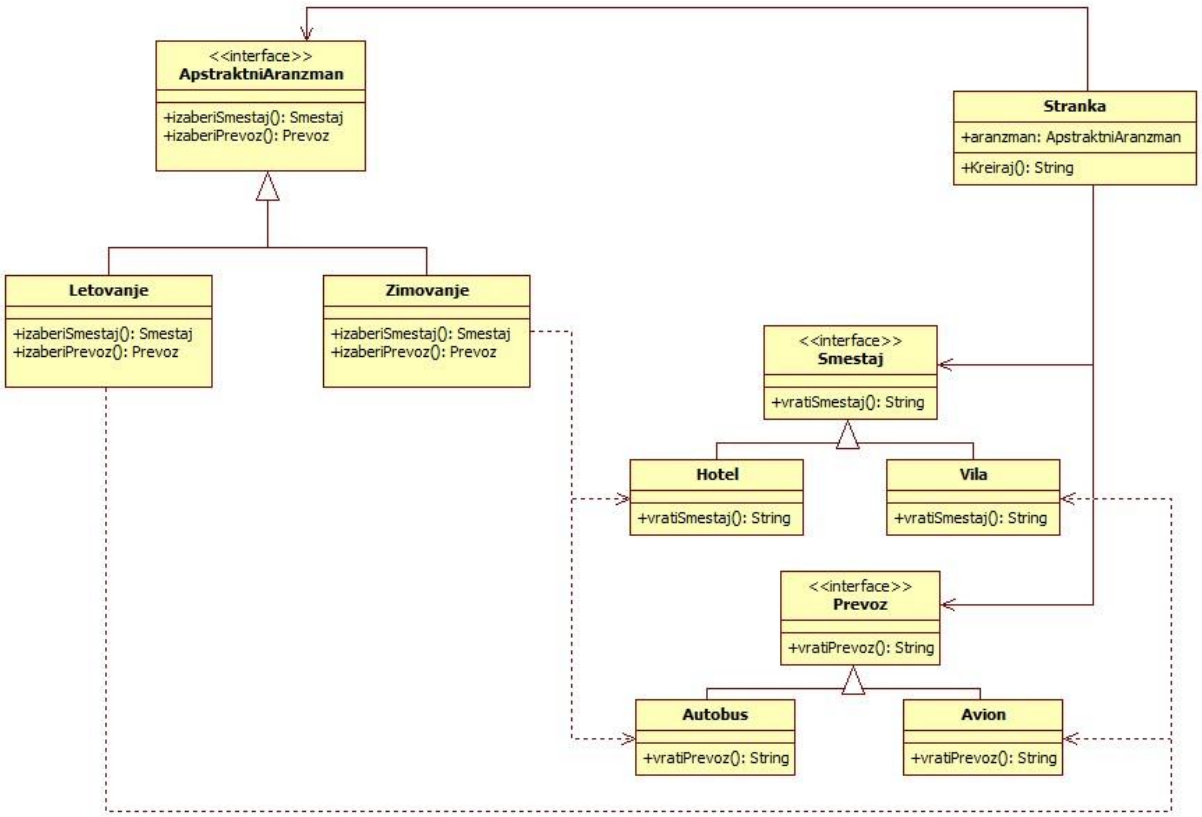
- **Client**

Koristi AbstractFactory i AbstractProduct interfejse za kreiranje finalnog proizvoda

4.1.3. Primer turističke agencije

Klijent dolazi u turističku agenciju „World travell“ da izabere aranžman za svoj naredni odmor. Turistička agencija ima u ponudi i letovanja i zimovanja. Klijent na početku bira gde će biti smešten. * Agencija u ponudi ima hotele i vile. Klijentu se takođe daje izbor da li želi da putuje avionom ili autobusom. Nakon što klijent sastavi sam svoju želju, saopštava je zaposlenom u agenciji, i na osnovu njegove želje traže se moguće turističke ponude odnosno aranžmani koji odgovaraju tim zahtevima.

Za rešenje ovog problema koristi se Abstract Factory patern. On se koristi kada stranka želi da rezerviše turistički aranžman određenih karakteristika. Prilikom izbora aranžmana, stranka bira iz jedne od postojeće dve vrste prevoza i dve vrste smeštaja. Stranka direktno nagleda proces kreiranja ponude, a zaposleni na njen zahtev kreira dve ponude: letovanje i zimovanje. Abstract Factory obezbeđuje interfejs **(*ApstraktniAranžman*)** za kreiranje ***(izaberiSmestaj(), izaberiPrevoz())*** familije povezanih ili zavisnih objekata **(*Smeštaj, Prevoz*)** bez navođenja (specificiranja) njihovih konkretnih klasa ***(Hotel , Vila, Autobus, Avion)***.



** Prilikom izbora turističkog aranžmana postoje razni kriterijumi, kao što su vreme putovanja, cena, destinacija, a ovde su uzeta samo dva radi jednostavnosti primera.*

4.1.4. Programski kod

Interfejs ApstraktniAranžman (AbstractFactory):

```
public interface ApstraktniAranzman {

    Smestaj izaberiSmestaj();

    Prevoz izaberiPrevoz();


}
```

Klasa Letovanje (Concrete Factory 1):

```
public class Letovanje implements ApstraktniAranzman {

    @Override
    public Smestaj izaberiSmestaj() {
        return new Vila();
    }

    @Override
    public Prevoz izaberiPrevoz() {
        return new Avion();
    }
}
```

Klasa Zimovanje (Concrete Factory 2):

```
public class Zimovanje implements ApstraktniAranzman {

    @Override
    public Smestaj izaberiSmestaj() {
      return new Hotel();
    }

    @Override
    public Prevoz izaberiPrevoz() {
        return new Autobus();
    }
}
```

Klasa Stranka (Client):

```
public class Stranka {

    ApstraktniAranzman aranzman;

    public Stranka(ApstraktniAranzman aranzman) {
        this.aranzman = aranzman;
    }

    String Kreiraj() {
        Smestaj smestaj = aranzman.izaberiSmestaj();
        Prevoz prevoz = aranzman.izaberiPrevoz();
        return "Smestaj za izabrani aranzman je: " + smestaj.vratiSmestaj() +
                " " + "i putuje se prevoznim sredstvom: "
                + prevoz.vratiPrevoz() + ".";
    }
}
```

Interface Smestaj (AbstractProductB):

```
public interface Smestaj {
    String vratiSmestaj();
}
```

Klasa Vila (ConcreteProductB1):

```
public class Vila implements Smestaj{

    @Override
    public String vratiSmestaj() {
        return "Vila";
    }

}
```

Klasa Hotel (ConcreteProductB2):

```
public class Hotel implements Smestaj{

    @Override
    public String vratiSmestaj() {
        return "Hotel";
    }

}
```

Interface Prevoz (AbstractProductC):

```
public interface Prevoz {
    String vratiPrevoz();
}
```

Klasa Autobus (ConreteProductC1):

```
public class Autobus implements Prevoz {

    @Override
    public String vratiPrevoz() {
        return "Autobus";
    }

}
```

Klasa Avion (ConreteProductC2):

```
public class Avion implements Prevoz{

    @Override
    public String vratiPrevoz() {
        return "Avion";
    }

}
```

Glavna (main) klasa:

```
public class Glavna {

    public static void main(String[] args) {
        Stranka stranka1;
        Letovanje letovanje = new Letovanje();
        stranka1 = new Stranka(letovanje);
        System.out.println("Prva stranka:");
        System.out.println(stranka1.Kreiraj());

        Stranka stranka2;
        Zimovanje zimovanje = new Zimovanje();
        stranka2 = new Stranka(zimovanje);
        System.out.println("Druga stranka:");
        System.out.println(stranka2.Kreiraj());

    }
}
```

Rezultat izvršavanja programa:

```
run:
Prva stranka:
Smestaj za izabrani aranzman je: Vila i putuje se prevoznim sredstvom: Avion.
Druga stranka:
Smestaj za izabrani aranzman je: Hotel i putuje se prevoznim sredstvom: Autobus.
BUILD SUCCESSFUL (total time: 0 seconds)
```

4.2. BUILDER PATERN

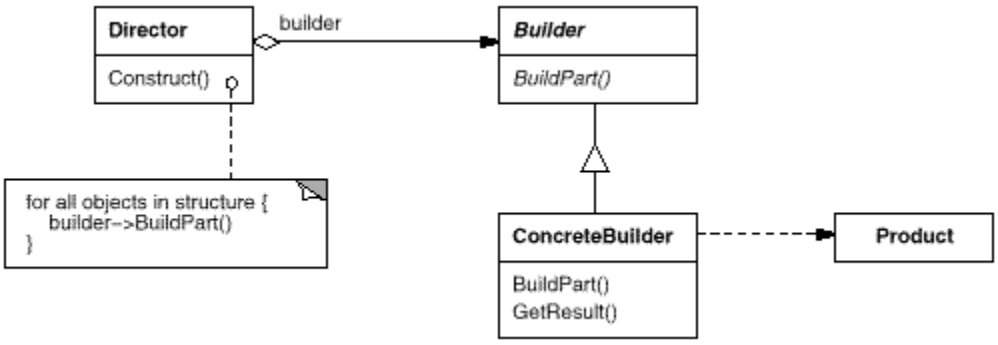
4.2.1. Definicija

Razdvaja izgradnju složenog objekta od njegove reprezentacije da bi isti proces pravljenja mogao da proizvede različite reprezentacije. On deli odgovornost za kontrolu konstrukcije (**Director**) složenog objekta od odgovornosti za realizaciju njegove reprezentacije-konkretne konstrukcije (**Builder**), tako da isti konstrukcioni proces (**Director.Construct()**) može da kreira različite reprezentacije u zavisnosti od ConcreteBuildera.

Director nadzire proces pravljenja proizvoda. Izrada delova i celine se prenosi na ConcreteBuilder klase.

1. Klijent zadužen za kontrolu sastavljanja proizvoda.
2. Server zadužen za pravljenje elemenata ponude.
3. Server zadužen za pravljenje proizvoda u celini.

4.2.2. Struktura



Učesnici:

- **Builder**

Specificira apstraktni interfejs za kreiranje <<delova proizvoda27>>.

- **ConcreteBuilder**

Konstruiše i grupiše <<delove proizvoda>> implementirajući <<Builder>> interfejs. Obezbeđuje interfejs za uzimanje <<proizvoda>>.

- **Director**

Konstruiše objekat (<<proizvod>>) korišćenjem <<Builder>> interfejsa. Kontroliše konstrukciju <<proizvoda>> korišćenjem <<Builder>> interfejsa.

- **Product**

Reprezentuje složen (kompleksan) objekat (<<proizvod>>) koji se konstruiše. Uključuje klase (interfejse) koji definišu <<delove proizvoda>>, uključujući interfejse za grupisanje delova u finalni rezultat (<<proizvod>>).

Karakteristike Buildera

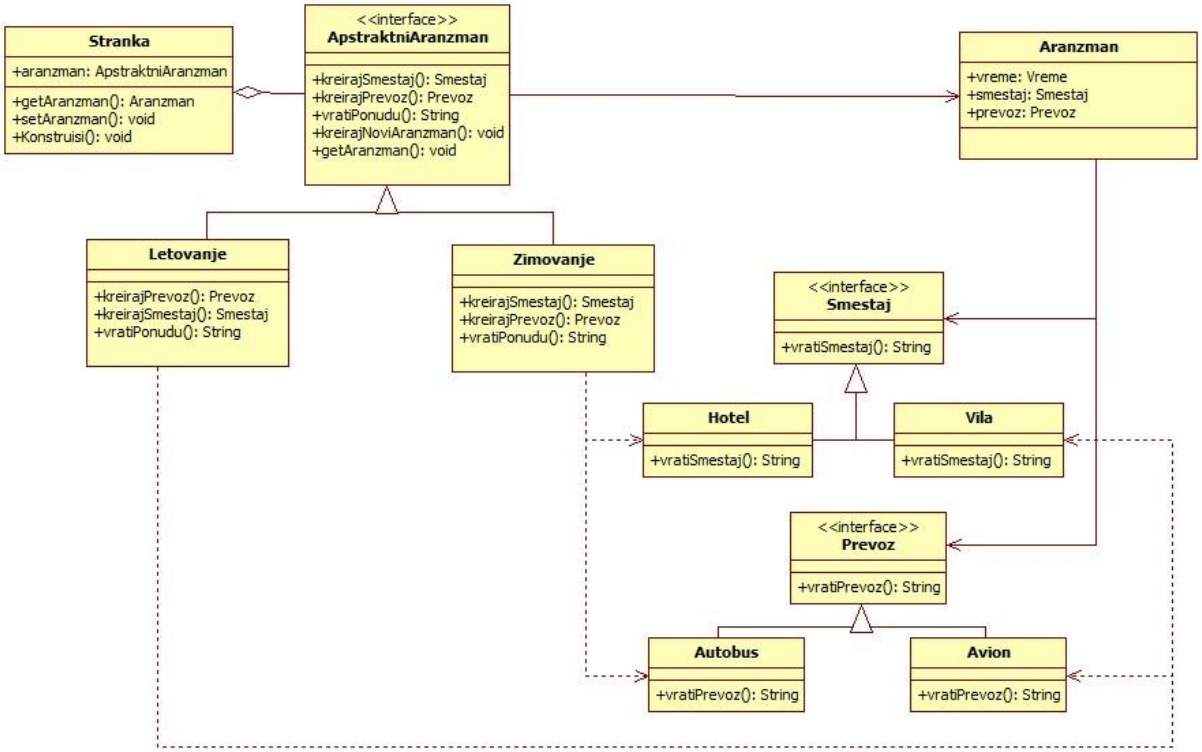
1. Omogućava menjanje unutrašnje predstave proizvoda. Ukoliko želimo da promenimo unutrašnju predstavu proizvoda, treba samo dodati novu vrstu buildera, a to nam omogućava apstraktni interfejs. On krije i način sklapanja proizvoda, i predstavu i unutrašnju strukturu proizvoda.
2. Bilder gradi proizvod korak po korak po kontrolom direktora, a ne odjednom, i tek kad se proizvod ceo završi, direktor ga preuzima.
3. Različiti direktori mogu koristiti iste linije koda.
4. U interfejsu se ne pojavljuju klase koje definišu unutrašnju strukturu proizvoda, pa prema tome klijent ne mora ništa da zna o njima.

4.2.3. Primer turističke agencije

Klijent dolazi u turističku agenciju „World travell“ da izabere aranžman za svoj naredni odmor. Turistička agencija ima u ponudi i letovanja i zimovanja. Osnovni kriterijum po kome stranka bira aranžman je smeštaj. Agencija u ponudi ima hotele i vile. Klijentu se takođe daje izbor da li želi da putuje avionom ili autobusom. Stranka u agenciji naglašava da su joj to bitne karakteristike (vreme, smestaj i prevoz) i na osnovu njih dobija turističku ponudu odnosno aranžman koju zaposleni u agenciji sastavlja.

Za rešenje ovog problema koristi se Builder patern. On se koristi kada u procesu kreiranja ponude, stranka samo nadgleda proces, odnosno kontroliše, dok zaposleni u agenciji sam kreira elemente ponude, odnosno smeštaj i prevoz, a na osnovu toga što je klijent naglasio koje su mu bitne karakteristike. Tačnije, klijent naglašava da su mu kod turističkog aranžmana jako bitni smeštaj i prevoz, a prepušta zaposlenom da pronađe odgovarajuće elemente ponude. Nakon toga, zaposleni u agenciji, odnosno server, sastavlja ponudu u celini.

Builder patern razdvaja izgradnju složenog objekta od njegove reprezentacije da bi isti proces pravljenja mogao da proizvede različite reprezentacije. On deli odgovornost za kontrolu konstrukcije (**Stranka**) složenog objekta od odgovornosti za realizaciju njegove reprezentacije- konkretne konstrukcije (**ApstraktniAranzman**), tako da isti konstrukcioni proces (**Director.Construct()**) može da kreira različite reprezentacije u zavisnosti od ConcreteBuildera (**Letovanje, Zimovanje**). Za opis ovog paterna se koriste apstraktne metode kreirajSmestaj(), kreirajPrevoz() i vratiPonudu() koja vraća String. Kao što je već navedeno, stranka nadgleda te aktivnosti. U klasi Stranka postoji metoda Konstruisi(), koja je analogna metodi Construct(). Preko te metode stranka nadgleda kreiranje ponude.



4.2.4. Programski kod

Apstraktna klasa Aranzman (Builder):

```
public abstract class ApstraktniAranzman {

    protected Aranzman aranzman;

    public Aranzman getAranzman() {
        return aranzman;
    }

    public void kreirajNoviAranzman() {
        aranzman = new Aranzman();
    }

    public abstract void kreirajSmestaj();

    public abstract void kreirajPrevoz();

    public abstract String vratiPonudu();
}
```

Klasa Aranzman:

```
package builder;
/*
 * Navedene klase klase i interfejsi su preuzeti
 * odnosno importovani iz primera za Abstract Factory uzor
 */

import abstractfactory.Prevoz;
import abstractfactory.Smestaj;

public class Aranzman {

    private Smestaj smestaj;
    private Prevoz prevoz;

    public Prevoz getPrevoz() {
        return prevoz;
    }

    public void setPrevoz(Prevoz prevoz) {
        this.prevoz = prevoz;
    }

    public Smestaj getSmestaj() {
        return smestaj;
    }

    public void setSmestaj(Smestaj smestaj) {
        this.smestaj = smestaj;
    }
}
```

Klasa Letovanje (ConcreteBuilder1):

```
public class Letovanje extends ApstraktniAranzman {

    @Override
    public void kreirajSmestaj() {
        aranzman.setSmestaj(new Vila());
    }

    @Override
    public void kreirajPrevoz() {
        aranzman.setPrevoz(new Avion());
    }

    @Override
    public String vratiPonudu() {
        System.out.println("Izabrani aranzman--- Smestaj je: "
                + aranzman.getSmestaj().vratiSmestaj()
                + " i prevoz je: " + aranzman.getPrevoz().vratiPrevoz() + ".");
        return "";

    }
}
```

Klasa Zimovanje (ConcreteBuilder2):

```
public class Zimovanje extends ApstraktniAranzman {

    @Override
    public void kreirajSmestaj() {
        aranzman.setSmestaj(new Hotel());
    }

    @Override
    public void kreirajPrevoz() {
        aranzman.setPrevoz(new Autobus());
    }

    @Override
    public String vratiPonudu() {
        System.out.println("Izabrani aranzman--- Smestaj je: "
                + aranzman.getSmestaj().vratiSmestaj()
                + " i prevoz je: " + aranzman.getPrevoz().vratiPrevoz() + ".");
        return "";
    }
}
```

Klasa Stranka (Director):

```
public class Stranka {

    private ApstraktniAranzman aranzman;

    public Aranzman getAranzman() {
        return aranzman.getAranzman();
    }

    public void setAranzman(ApstraktniAranzman aranzman) {
        this.aranzman = aranzman;
    }

    public void Konstruisi() {
        aranzman.kreirajNoviAranzman();
        aranzman.kreirajSmestaj();
        aranzman.kreirajPrevoz();
    }
}
```

Glavna (main) klasa:

```
public class Glavna {

    public static void main(String[] args) {
        Stranka stranka3 = new Stranka();
        ApstraktniAranzman letovanje = new Letovanje();
        stranka3.setAranzman(letovanje);
        stranka3.Konstruisi();
        System.out.println(letovanje.vratiPonudu());

        Stranka stranka4 = new Stranka();
        ApstraktniAranzman zimovanje = new Zimovanje();
        stranka4.setAranzman(zimovanje);
        stranka4.Konstruisi();
        System.out.println(zimovanje.vratiPonudu());
    }
}
```

Rezultat izvršavanja programa:

```
run:
Izabrani aranzman--- Smestaj je: Vila i prevoz je: Avion.

Izabrani aranzman--- Smestaj je: Hotel i prevoz je: Autobus.

BUILD SUCCESSFUL (total time: 1 second)
```

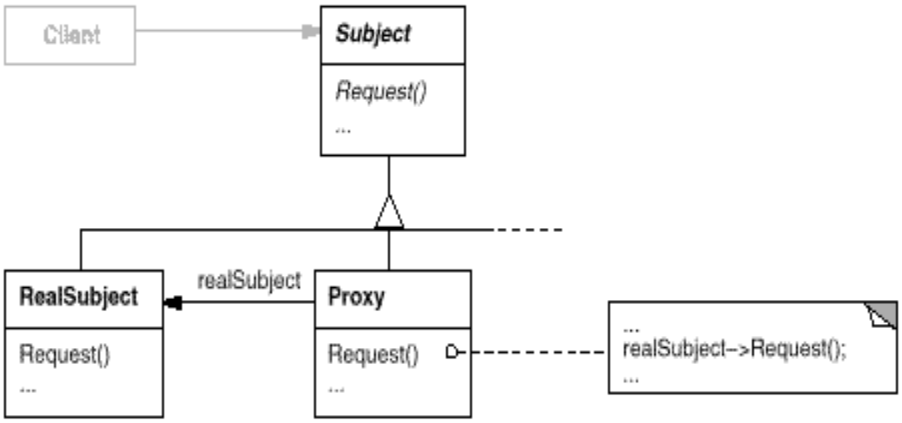
5. STRUKTURNI UZORI

5.1. PROXY

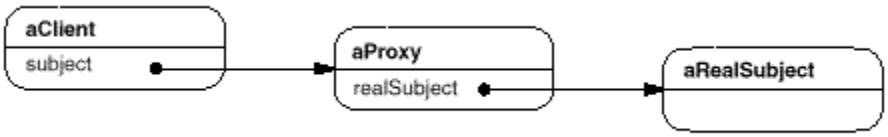
5.1.1. Definicija

Obezbeđuje posrednika (Proxy) za pristupanje drugom objektu(RealSubject) kako bi se omogućio kontrolisani pristup do njega.

5.1.2. Struktura



Mogući objektni dijagram proxy strukture u realnom vremenu:



Učesnici:

- **Proxy**

Sadrži reference koje omogućavaju <<Proxy>> objektu pristup do <<realnog subjekta>>. Obezbeđuje interfejs identičan sa interfejsom <<Subject >> tako da <<proxy objekat>> može zameniti <<RealSubject objekat>>. Kontroliše pristup do <<RealSubject>> objekta i može biti odgovoran za njegovo kreiranje i brisanje.

- **Subject**

Definiše zajednički interfejs za <<RealSubject>> i <<Proxy klase>> tako da <<Proxy objekat>> može zameniti <<RealSubject objekat>>.

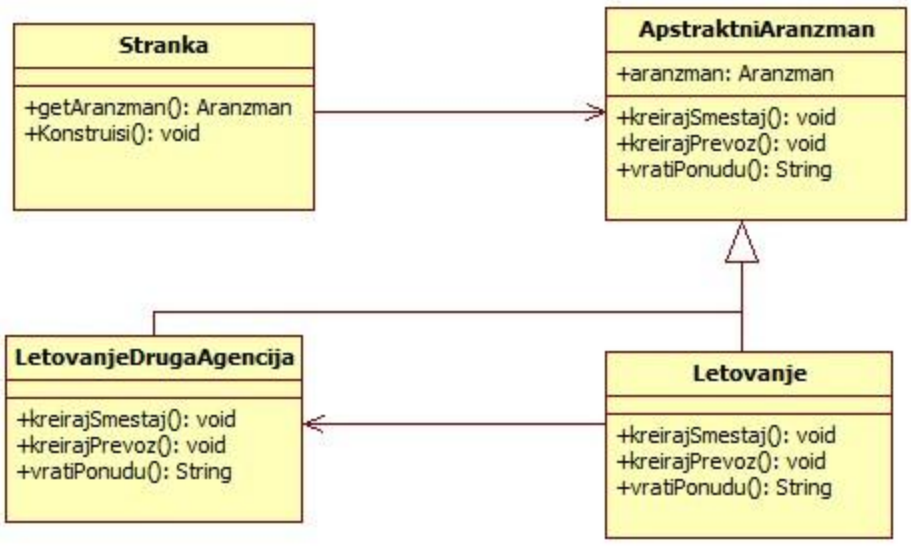
- **RealSubject**

Definiše <<RealSubject>> objekat koji <<Proxy>> objekat reprezentuje**.**

5.1.3. Primer turističke agencije

Nakon što je pogledao sve ponude, klijent se odlučio da tekuće godine ide na letovanje. Ipak, odlazi da još malo razmisli, ali napominje da će se vratiti da rezerviše a nakon toga i uplati novac za ponudu koja mu se najviše dopala. Međutim, po povratku saznaje da su svi kapaciteti popunjeni, i da nema vise slobodnih mesta u tom turističkom aranžmanu. Agencija “World travel” uspešno sarađuje sa drugom agencijom “Happy”, i obe vode turiste na iste lokacije i u isto vreme. Zaposleni u agenciji objašnjava identičnu ponudu agencije “Happy", dok klijent kod sebe ima katalog svih aranžmana agencije “World travell” (Subject). Klijent gleda u katalogu opis ponude letovanja koja mu se dopala (Proxy), dok zaposleni u agenciji poziva drugu agenciju I traži da se rezerviše ponuda za letovanje za njegovog klijenta (Real Subject).

Proxy patern obezbeđuje posrednika (**Proxy**) koji je u ovom primeru aranžman za letovanje za pristupanje drugom objektu (**Real Subject**) koji je u ovom primeru ponuda za letovanje druge agencije, kako bi se omogućio kontrolisani pristup do njega. Odnosno, letovanje agencije “World travel” je posrednik, jer stranka sada pristupa letovanju druge agencije.



5.1.4. Programski kod

Klasa Stranka (Client):

```
public class Stranka {

    private ApstraktniAranzman aranzman;

    public Aranzman getAranzman() {
        return aranzman.getAranzman();
    }

    public Stranka(ApstraktniAranzman aranzman) {
        this.aranzman = aranzman;
    }

    public void Konstruisi() {
        aranzman.kreirajNoviAranzman();
        aranzman.kreirajSmestaj();
        aranzman.kreirajPrevoz();
    }
}
```

Klasa Letovanje (Proxy):

```
public class Letovanje extends ApstraktniAranzman {

    LetovanjeDrugaAgencija letovanjeDA;

    public Letovanje(LetovanjeDrugaAgencija letovanjeDA) {
        this.letovanjeDA = letovanjeDA;
    }

    @Override
    public void kreirajSmestaj() {
        letovanjeDA.kreirajSmestaj();
    }

    @Override
    public void kreirajPrevoz() {
        letovanjeDA.kreirajPrevoz();
    }

    @Override
    public String vratiPonudu() {
        return letovanjeDA.vratiPonudu();

    }
}
```

Klasa LetovanjeDrugaAgencija (RealSubject):

```
public class LetovanjeDrugaAgencija extends ApstraktniAranzman {

    Aranzman aranzman;

    public LetovanjeDrugaAgencija() {
        aranzman = new Aranzman();
    }

    @Override
    public void kreirajSmestaj() {
        aranzman.setSmestaj(new Vila());
    }

    @Override
    public void kreirajPrevoz() {
        aranzman.setPrevoz(new Avion());
    }

    @Override
    public String vratiPonudu() {
        System.out.println("Agencija Happy:" + "\n"
                + "Izabrani aranzman--- Smestaj je: "
                + aranzman.getSmestaj().vratiSmestaj()
                + " i prevoz je: " + aranzman.getPrevoz().vratiPrevoz() + " .");
        return "";
    }
}
```

Glavna (main) klasa:

```
public class Glavna {

    public static void main(String[] args) {
        Stranka stranka;
        LetovanjeDrugaAgencija letovanjeDrugaAgencija = new LetovanjeDrugaAgencija();
        Letovanje letovanje = new Letovanje(letovanjeDrugaAgencija);
        stranka = new Stranka(letovanje);
        stranka.Konstruisi();
        System.out.println(letovanje.vratiPonudu());
    }
}
```

Rezultat izvršavanja programa:

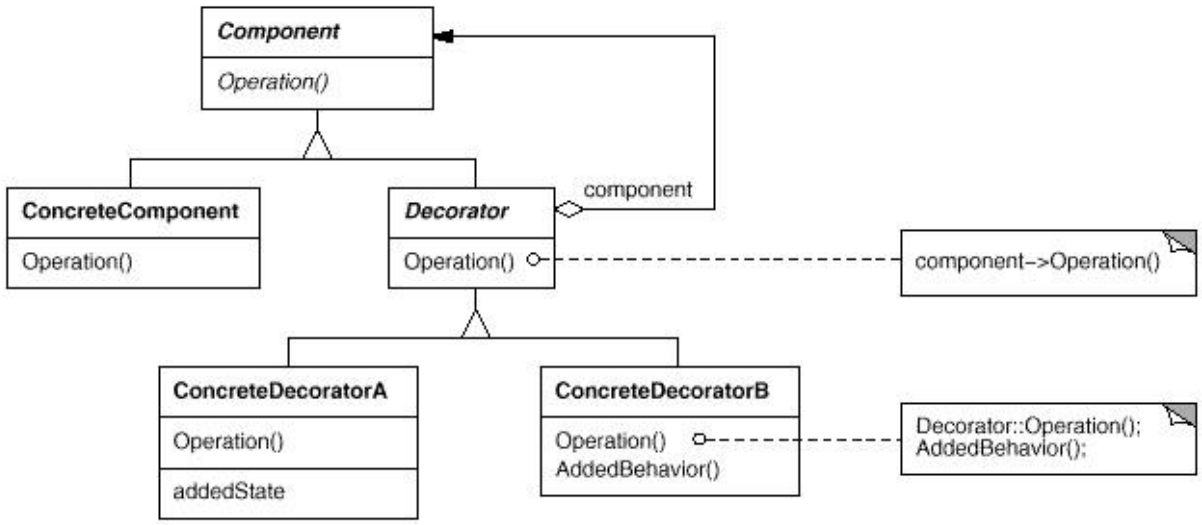
```
run:
Agencija Happy:
Izabrani aranzman--- Smestaj je: Vila i prevoz je: Avion.

BUILD SUCCESSFUL (total time: 0 seconds)
```

5.2. DECORATOR

5.2.1. Defininija

Decorator patern pridružuje odgovornost do objekta dinamički. Dekorator proširuje funkcionalnost objekta dinamičkim dodavanjem funkcionalnosti drugih objekata.

5.2.2. Struktura



Učesnici:

- **Component**

Definiše interfejs za objekte kojima se odgovornost dodaje dinamički.

- **ConcreteComponent**

Definiše konkretan objekat kome će biti dodata odgovornost dinamički

- **Decorator**

Čuva referencu na komponentu i poziva operaciju komponenti

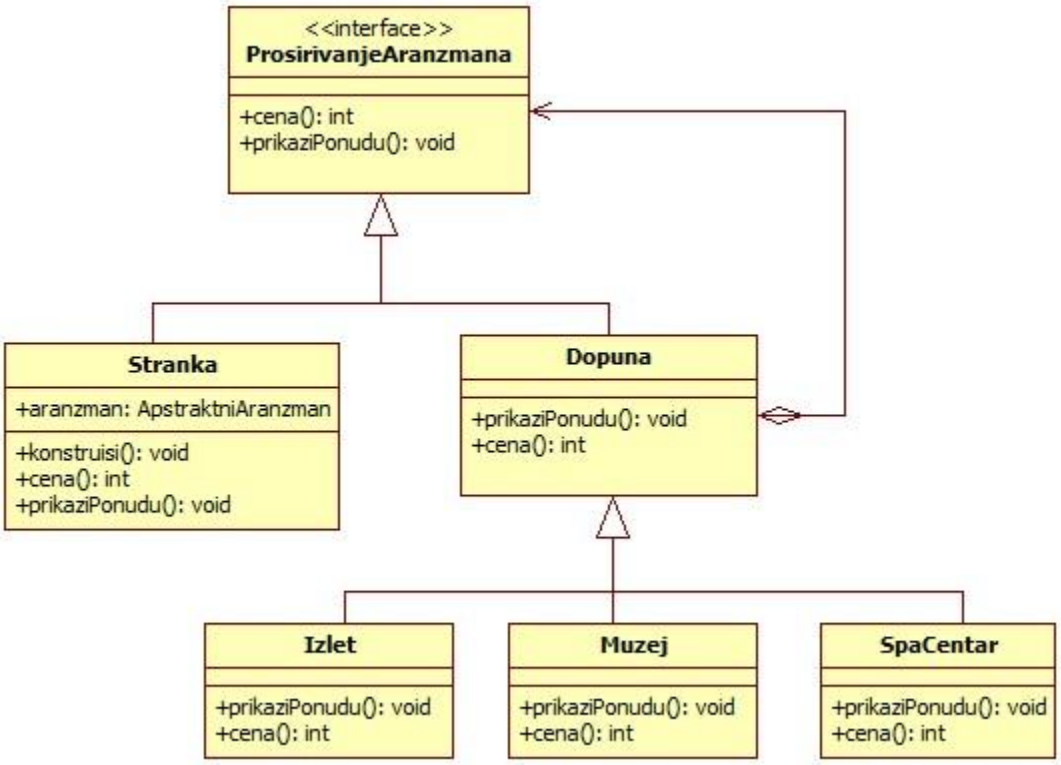
- **ConcreteDecorator**

Dodaje odgovornost do komponente

5.2.3. Primer turističke agencije

Svaka stranka prilikom rezervacije putovanja može da izabere još neke dodatne komponente kojima bi proširila svoju turističku ponudu. Agencija stranke vodi na razne izlete, u posete najpoznatijim muzejima država u koje se putuje, a pored toga nudi turistima da uplate dodatni novac i za korišćenje usluga Spa centra. Ako se turista odluči za izlet, treba da doplati još 4000 dinara, za posetu muzeju 2000 dinara, dok za neograničeno korišćenje Spa centra plaća dodatno još 1500 dinara.

Dekorator patern po definiciji služi za dodavanje novih funkcija objektu, odnosno on pridružuje odgovornost do objekta dinamički. U ovom primeru to znači da se jedan turistički aranžman može proširiti dodatnim funkcijama, odnosno strankama se nudi da uplate dodatan novac za posetu muzeju, odlazak na izlet ili za korišćenje spa centra. **Decorator** koji je u ovom primeru klasa Dopuna proširuje funkcionalnost objekta Stranka (**Concrete Component**) dinamičkim dodavanjem funkcionalnosti drugih objekata: Izlet (**ConcreteDecoratorA**), Muzej (**ConcreteDecoratorB**) i SpaCentar (**ConcreteDecoratorC**).



5.2.4. Programski kod

Interface ProsirivanjeAranzmana (Component):

```
public interface ProsirivanjeAranzmana {
    public abstract int cena();
    void prikaziPonudu();
}
```

Klasa Dopuna (Decorator):

```
public abstract class Dopuna implements ProsirivanjeAranzmana {

    ProsirivanjeAranzmana komp;

    public Dopuna(ProsirivanjeAranzmana komp) {
        this.komp = komp;
    }

    @Override
    public void prikaziPonudu() {
        komp.prikaziPonudu();
    }

    public abstract int cena();
```

Klasa Izlet (ConcreteDecoratorA):

```
public class Izlet extends Dopuna {

    public Izlet(ProsirivanjeAranzmana komp) {
        super(komp);
    }

    @Override
    public void prikaziPonudu() {
        super.prikaziPonudu();
        System.out.println("\nStranka je izabrala da na odmoru takodje "
                + "ode i na izlet koji agencija nudi,"
                + " i za to je uplatila dodatni novac."
                + " Cena izleta je 4000 dinara.");
    }

    @Override
    public int cena() {
        return komp.cena() + 4000;
    }
}
```

Klasa Muzej (ConcreteDecoratorB):

```
public class Muzej extends Dopuna{

    public Muzej(ProsirivanjeAranzmana komp) {
        super(komp);
    }

    @Override
    public void prikaziPonudu() {
        super.prikaziPonudu();
        System.out.println("\nAranzman je dopunjen posetom muzeju."
                + " Poseta muzeju koji je stranka izabrala kosta 2000 dinara.");
    }
    @Override
    public int cena() {
     return  komp.cena()+ 2000;
    }

}
```

Klasa SpaCentar (ConcreteDecoratorC):

```
public class Spacentar extends Dopuna {

    public Spacentar(ProsirivanjeAranzmana komp) {
        super(komp);
    }

    @Override
    public void prikaziPonudu() {
        super.prikaziPonudu();
        System.out.println("\nStranka je uplatila dodatni novac"
                + " da u sklopu ponude ima i koriscenje spa centra."
                + " Koriscenje spa centra se naplacuje 2000 dinara.");
    }

    @Override
    public int cena() {
        return komp.cena() + 1500;
    }
}
```

Klasa Stranka (ConcreteComponent):

```
public class Stranka implements ProsirivanjeAranzmana {

    private int cena;
    ApstraktniAranzman aranzman;

    public Stranka(ApstraktniAranzman aranzman) {
        this.aranzman = aranzman;
    }

    public void Konstruisi() {
        aranzman.kreirajNoviAranzman();
        aranzman.kreirajSmestaj();
        aranzman.kreirajPrevoz();
    }

    @Override
    public void prikaziPonudu() {
        System.out.println("\nDopunjena ponuda: " + aranzman.vratiPonudu());
    }

    @Override
    public int cena() {
        return cena;
    }
}
```

Rezultat izvršavanja programa:

```
run:
Izabrani aranzman--- Smestaj je: Vila i prevoz je: Avion.

Dopunjena ponuda:

Stranka je izabrala da na odmoru takodje ode i na izlet koji agencija nudi, i za to je uplatila dodatni novac. Cena izleta je 4000 dinara.

Aranzman je dopunjen posetom muzeju. Poseta muzeju koji je stranka izabrala kosta 2000 dinara.

Stranka je uplatila dodatni novac da u sklopu ponude ima i koriscenje spa centra. Koriscenje spa centra se naplacuje 2000 dinara.
Ukupna cena svih dopuna aranzmana koje je stranka izabrala: 7500
BUILD SUCCESSFUL (total time: 0 seconds)
```

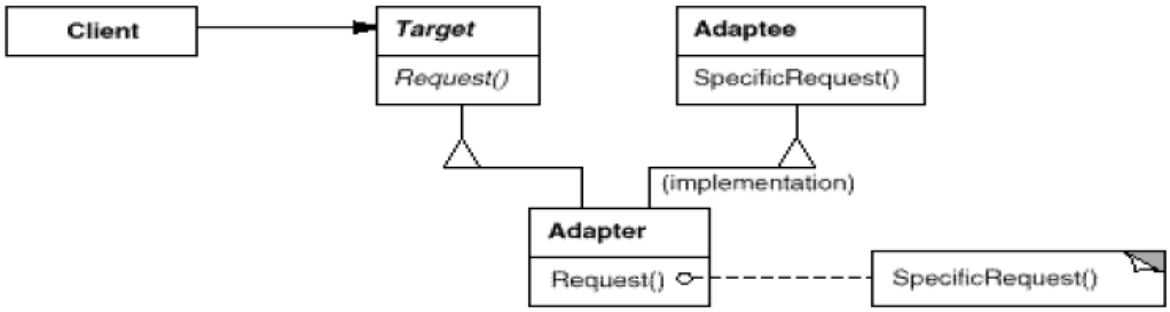
5.3. ADAPTER

5.3.1. Definicija

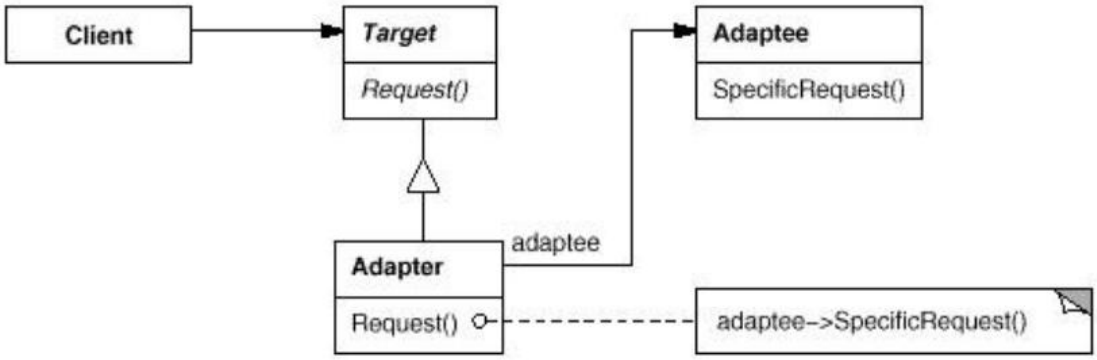
Adapter patern konvertuje interfejs neke klase u drugi interfejs koji klijent očekuje. Drugim rečima, on prilagođava nekompatibilne interfejse.

Ponekad postoji potreba da se postojeća klasa upotrebi pošto nam pruža neke ili sve mogućnosti koje su nam potrebne, ali njen interfejs nije usklađen sa interfejsom koji nam je potreban. Tada je prilika da se upotrebi Adapter patern koji konvertuje interfejs postojeće klase u drugi interfejs koji klijenti očekuju. Adapter omogućava saradnju klasa koje inače ne bi mogle da sarađuju zbog nekompatibilnosti njihovih interfejsa.

5.3.2. Struktura



Klasa Adapter koristi višestruko nasleđivanje kod prolagođavanja nekompatibilnih interfejsa.



Klasa Adapter koristi kompoziciju kod prilagođavanja nekompatibilnih interfejsa

Učesnici:

- **Target**

Definiše domenski, specifičan interfejs, koji Client koristi

- **Client**

Poziva operacije preko željenog interfejsa klase Target

- **Adaptee**

Definiše interfejs koji treba adaptirati

- **Adapter**

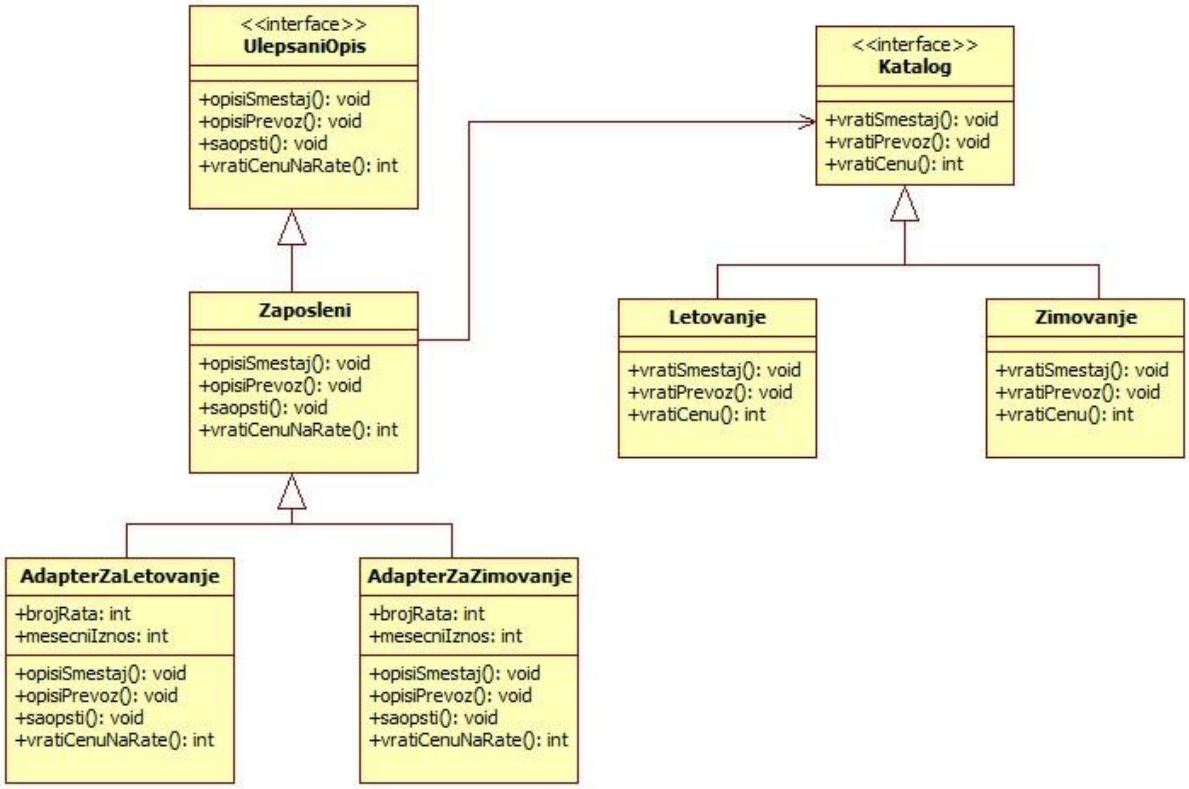
Adaptira interfejs Adaptee prema Target interfejsu

5.3.3. Primer turističke agencije

Postoje neke činjenice koje je ponekad potrebno saopštiti turistima na drugačiji, lepši način, detaljnije, ne bi li ih ubedili da na putovanje idu upravo preko naše agencije. Adapter patern upravo je upotrebljen za prilagođavanje, odnosno detaljnije objašnjavanje karakteristika turističkih aranžmana zainteresovanim strankama.

Stranka dolazi u turističku agenciju ne bi li rezervisala ponudu za svoje putovanje. Međutim, i dalje je u dilemi da li želi tekuće godine da ide na letovanje ili na zimovanje. Stranka sa sobom ima katalog koji je dobila prošli put kada je dolazila u agenciju, i sada traži od zaposlenog u agenciji da joj detaljnije objasni aranžmane koji su u ponudi, i na taj način joj pomogne da izabere aranžman. Zaposleni pored toga što u katalogu piše da se na letovanje putuje avionom, mušteriji objašnjava ko je prevoznik, a za zimovanje pored šturog opisa u katalogu da se putuje autobusom, navodi takođe ko je prevoznik, da su autobusi klimatizovani i na dva sprata. Stranku zanima i gde će biti smeštena, te joj zaposleni daje detaljnije informacije o vilama, odnosno hotelima, kao što su: koliko kreveta u sobi ima, da li terasa ima pogled na more, koliko je vila udaljena od plaže, sa koliko zvezdica je hotel itd. Pored toga, zaposleni objašnjava stranci da aranžman može da plati na rate i koja je u tom slučaju visina mesečne rate.

Za rešenje ovog problema korišćen je Adapter patern. On konvertuje interfejs klase Katalog (**Adaptee**) u drugi interfejs UlepsaniOpis (**Target**) koji klijent očekuje.Drugim rečima, on prilagođava nekompatibilne interfejse Katalog i UlepsaniOpis pomoću klase Zaposleni (**Adapter**). U ovom primeru se javljaju dva turistička aranžmana- letovanje i zimovanje, i samim tim i različita objašnjenja za svaki od njih. Zbog toga ovde postoji klasa Zaposleni koja je apstraktni adapter, a nju nasleđuju klase AdapterZaLetovanje i AdapterZaZimovanje koje su konkretni adapteri. Interfejs Katalog je samim tim AbstractAdaptee, a njega implementiraju konkretne klase Letovanje i Zimovanje koje su ConcreteAdaptee.



5.3.4. Programski kod

Interface Katalog (Adaptee):

```
public interface Katalog {
    String vratiSmestaj();
    String vratiPrevoz();
    int vratiCenu();

}
```

Klasa Letovanje:

```
public class Letovanje implements Katalog {

    @Override
    public String vratiSmestaj() {
        return "Vila";
    }

    @Override
    public String vratiPrevoz() {
        return "Avion";
    }

    @Override
    public int vratiCenu() {
        return 15000;
    }
}
```

Klasa Zimovanje:

```
public class Zimovanje implements Katalog {

    @Override
    public String vratiSmestaj() {
        return "Hotel";
    }

    @Override
    public String vratiPrevoz() {
        return "Autobus";
    }

    @Override
    public int vratiCenu() {
        return 10000;
    }
}
```

Interface UlepsaniOpis (Target):

```
public interface UlepsaniOpis {

    void opisiSmestaj();

    void opisiPrevoz();

    int vratiCenuNaRate();

    void saopsti();
}
```

Klasa Zaposleni (Apstraktni Adapter):

```
public abstract class Zaposleni implements UlepsaniOpis {

    Katalog katalog;

    @Override
    public void saopsti() {
        opisiSmestaj();
        opisiPrevoz();
        vratiCenuNaRate();
    }

    @Override
    public void opisiSmestaj() {
        katalog.vratiSmestaj();
    }

    @Override
    public void opisiPrevoz() {
        katalog.vratiPrevoz();
    }

    @Override
    public int vratiCenuNaRate() {
        return katalog.vratiCenu();
    }
}
```

Klasa AdapterZaLetovanje (Konkretni Adapter):

```
public class AdapterZaLetovanje extends Zaposleni {

    int brojRata;
    private int mesecniIznos;

    public AdapterZaLetovanje(Katalog katalog, int brojRata) {
        this.katalog = katalog;
        this.brojRata = brojRata;
    }

    @Override
    public void opisiSmestaj() {
        System.out.println(katalog.vratiSmestaj() + " je zgrada u "
                + "kojoj cete biti smesteni."
                + " To je prelepa vila sa pogledom na more."
                + " Sobe su dvokrevetne, imate kuhinju i kupatilo."
                + "Vila je udaljena svega 100 metara od plaze.");
    }

    @Override
    public void opisiPrevoz() {
        System.out.println("Na letovanje cete ici prevoznim sredstvom "
                + katalog.vratiPrevoz() + "om."
                + " Avioni su prevoznika JatArways.");
    }

    @Override
    public int vratiCenuNaRate() {
        mesecniIznos = katalog.vratiCenu() / brojRata;
        System.out.println("Agencija vam omogucava da letovanje platite na rate"
                + ", pri cemu ce mesecna rata iznositi: " + mesecniIznos
                + " dinara.");
        return 0;


    }
}
```

Klasa AdapterZaZimovanje (Konkretni Adapter):

```
public class AdapterZaZimovanje extends Zaposleni {

    int brojRata;
    private int mesecniIznos;

    public AdapterZaZimovanje(Katalog katalog, int brojRata) {
        this.katalog = katalog;
        this.brojRata = brojRata;
    }

    @Override
    public void saopsti() {
        super.saopsti();
    }

    @Override
    public void opisiPrevoz() {
        System.out.println("Prevoz kojim cete ici na zimovanje je: " +
                katalog.vratiPrevoz()
                + ". Autobusi su klimatizovani, na sprat,"
                + " a prevoznik je Gaga Tours.");

    }

    @Override
    public void opisiSmestaj() {
        System.out.println("Na zimovanju cete biti smesteni u " +
                katalog.vratiSmestaj() + "u " + "sa 4 zvezdice.");
    }

    @Override
    public int vratiCenuNaRate() {
        mesecniIznos = katalog.vratiCenu() / brojRata;
        System.out.println("Agencija vam omogucava da zimovanje platite na rate"
                + ", pri cemu ce mesecna rata iznositi: "
                + mesecniIznos + " dinara.");
        return 0;

    }
}
```

Glavna (main) klasa:

```
public class Glavna {

    public static void main(String[] args) {
        Katalog letovanje = new Letovanje();
        UlepsaniOpis opis = new AdapterZaLetovanje(letovanje, 6);
        opis.saopsti();
        System.out.println("_________________");
        Katalog zimovanje = new Zimovanje();
        UlepsaniOpis opis1 = new AdapterZaZimovanje(zimovanje, 7);
        opis1.saopsti();
        System.out.println("_________________");
    }
}
```

Rezultat izvršavanja programa:

```
run:
Vila je zgrada u kojoj cete biti smesteni. To je prelepa vila sa pogledom na more. Sobe su dvokrevetne, imate kuhinju i kupatilo.
Na letovanje cete ici prevoznim sredstvom Avionom. Avioni su prevoznika JatArways.
Agencija vam omogucava da letovanje platite na rate, pri cemu ce mesecna rata iznositi: 2500 dinara.
_________________
Na zimovanju cete biti smesteni u Hotelu sa 4 zvezdice.
Prevoz kojim cete ici na zimovanje je: Autobus. Autobusi su klimatizovani, na sprat, a prevoznik je Gaga Tours.
Agencija vam omogucava da zimovanje platite na rate, pri cemu ce mesecna rata iznositi: 1428 dinara.
_________________
BUILD SUCCESSFUL (total time: 1 second)
```

6. PATERNI PONAŠANJA

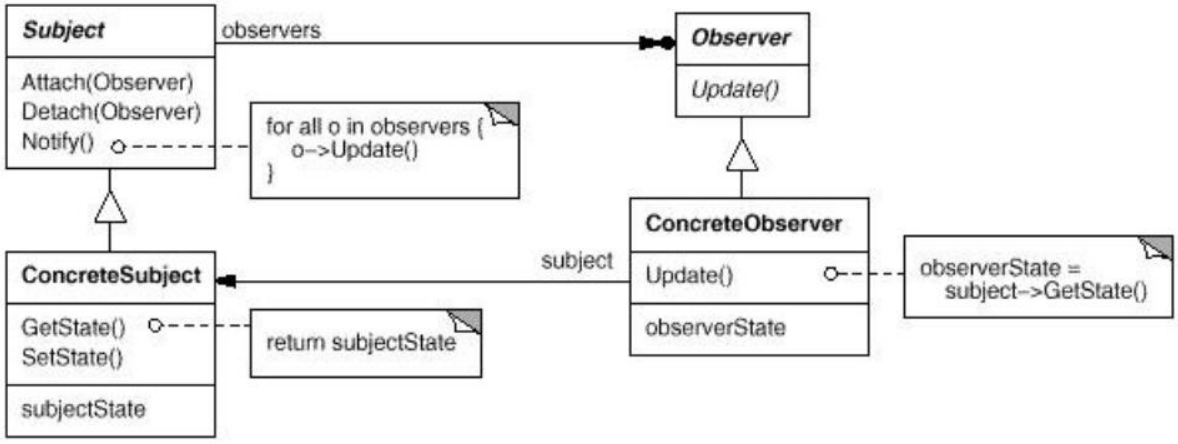
6.1. OBSERVER

6.1.1. Definicija

Observer patern definiše jedan-više zavisnost između objekata (Subject -> Observer), tako da promena stanja nekog objekta (ConcreteSubject) utiče automatski na promenu stanja svih objekata koji su povezani sa njim (Concrete Observer).

Observer obezbeđuje interface za promenu objekta na novo stanje u koje je prešao Subject objekat. Subject zna ko su njegovi observeri, a ConcreteSubject čuva stanje na koje se postavljaju observeri. ConcreteObserver ima referencu na ConcreteSubject.

6.1.2. Struktura



Učesnici:

- **Subject**

Zna ko su njegovi observeri
Obezbeđuje interfejs za vezivanje i razvezivanje observer

- **Observer**

Obezbeđuje interfejs za promenu objekta na novo stanje u koje je prešao Subject objekat

- **ConcreteSubject**

Čuva stanje na koje se postavljaju observer

- **ConcreteObserver**

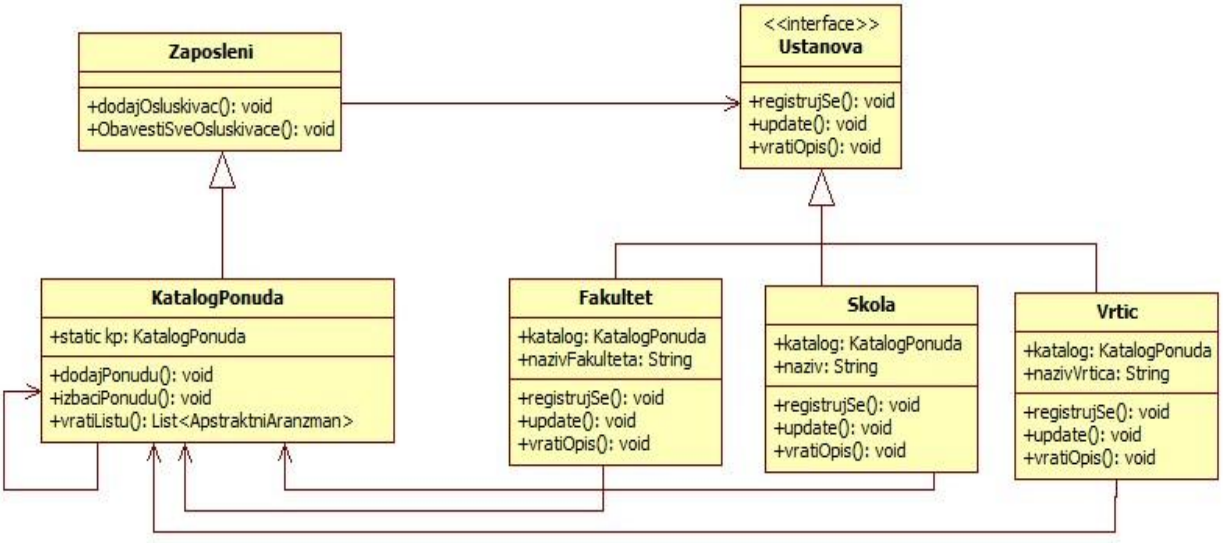
Čuva referencu na Concrete Subject objekat

Čuva stanje koje je konzistentno sa stanjem ConcreteSubject objekta
Implementira interfejs kojim se objekat postavlja novo stanje u koje je prešao Subject objekat

6.1.3. Primer turističke agencije

Turistička agencija pored ponuda za fizička lica, ima i specijalne ponude, odnosno turističke aranžmane za državne ustanove, kao što su fakulteti, škole i vrtići. Kataloge sa turističkim ponudama zaposleni u agenciji redovno šalje ustanovama, ne bi li one na vreme bile obaveštene koje su trenutno aktuelne ponude, koje su cene, da bi imale vremena da odluče na koju lokaciju da vode decu na rekreativnu nastavu, odnosno đake i studente na ekskurzije. Svaki put kada se u katalog unese nova turistička ponuda, ili ako se neka ponuda izbaci za narednu sezonu, sve ustanove o tome bivaju obaveštene. Ukoliko se škola, fakultet ili vrtić odluči za neki od aranžmana agencije, dobiće odgovarajući popust na određen broj dece, đaka, odnosno studenata. Učenici škola mogu svoju ekskurziju da plate na 10 rata, i ukoliko ih se prijavi više od 200, imaće popust od 10 % po učeniku. Studenti ekskurziju plaćaju na 7 rata, i ukoliko ih je više od 100 prijavljenih, dobijaju popust od po 5%. Roditelji rekreativnu nastavu za decu mogu da plate na 7 rata, i ukoliko bude više od 300 prijavljene dece, dobiće popust u iznosu od 15%.

Za rešenje ovog problema koristi se Observer patern. On definiše jedan-više zavisnost između objekata (Subject → Observer), koji su u ovom primeru klasa Zaposleni (**Subjec**t) i interfejs Ustanova (**Observer**), tako da promena stanja objekta KatalogPonuda (**ConcreteSubject**) utiče automatski na promenu stanja svih drugih objekata: Fakultet (**ConcreteObserverA**), Skola (**ConcreteObserverB**) i Vrtic (**ConcreteObserverC**) koji su povezani sa njim.



6.1.4. Programski kod

Klasa Zaposleni (Subject):

```
public abstract class Zaposleni {

    private List<Ustanova> listaOsluskivaca;

    public Zaposleni() {
        listaOsluskivaca = new ArrayList();
    }

    public void dodajOsluskivac(Ustanova kupac) {
        listaOsluskivaca.add(kupac);
    }

    public void ObavestiSveOsluskivace() {

        for (Ustanova k : listaOsluskivaca) {
            k.update();
        }
    }
}
```

Interface Ustanova (Observer):

```
public interface Ustanova {

    public void update();

    public void registrujSe(KatalogPonuda katalog);

    public void vratiOpis();
}
```

Klasa KatalogPonuda (ConcreteSubject):

```
public class KatalogPonuda extends Zaposleni {

    private List<ApstraktniAranzman> aranzmani;
    private static KatalogPonuda kp;

    public static KatalogPonuda vratiKatalog() {
        if (kp == null) {
            kp = new KatalogPonuda();
        }
        return kp;
    }

    public KatalogPonuda() {
        aranzmani = new ArrayList<ApstraktniAranzman>();
    }

    public void dodajPonudu(ApstraktniAranzman aranzman) {
        aranzmani.add(aranzman);
        ObavestiSveOsluskivace();
    }

    public void izbaciPonudu(int i) {
        aranzmani.remove(i);
        ObavestiSveOsluskivace();
    }

    public List<ApstraktniAranzman> vratiListu() {
        return aranzmani;
    }
}
```

Klasa Fakultet (ConcreteObserverA):

```
public class Fakultet implements Ustanova {

    String nazivFakulteta;
    KatalogPonuda katalog;
    private List<ApstraktniAranzman> aranzmani;

    public String getNazivFakulteta() {
        return nazivFakulteta;
    }

    public Fakultet(String nazivFakulteta) {
        this.nazivFakulteta = nazivFakulteta;
        aranzmani = new ArrayList<ApstraktniAranzman>();
    }

    @Override
    public void update() {
        aranzmani.clear();
        aranzmani.addAll(katalog.vratiListu());
    }

    @Override
    public void registrujSe(KatalogPonuda katalog) {
        this.katalog = katalog;
    }

   @Override
   public void vratiOpis() {
       System.out.println(getNazivFakulteta());
       for (ApstraktniAranzman ar : aranzmani) {
           System.out.println(ar.opisAranzmana());
           System.out.println("Cena turistickog aranzmana za Fakultete: "
                   + ar.cenaAranzmana()
                   + ". U ponudi za fakultete su najludja party putovanja, "
                   + "apsolventske ekskurzije"
                   + " u Italiju, Spaniju i Francusku."
                   + " Placanje je moguce na 7 rata."
                   + "Ukoliko broj prijavljenih studenata bude veci od 100,"
                   + " dobijate popust od 5% po studentu"
                   + " i tada ce cena aranzmana iznositi " +
                   ar.cenaAranzmana() * 0.95);
       }
   }
}
```

Klasa Skola (ConcreteObserverB):

```
public class Skola implements Ustanova {

    String naziv;
    KatalogPonuda katalog;
    private List<ApstraktniAranzman> aranzmani;

    public Skola(String naziv) {
        this.naziv = naziv;
        aranzmani = new ArrayList<ApstraktniAranzman>();
    }

    public String getNaziv() {
        return naziv;
    }

    @Override
    public void update() {
        aranzmani.clear();
        aranzmani.addAll(katalog.vratiListu());
    }

     @Override
     public void registrujSe(KatalogPonuda katalog) {
         this.katalog = katalog;
     }

     @Override
     public void vratiOpis() {
         System.out.println(getNaziv());
         for (ApstraktniAranzman ar : aranzmani) {
             System.out.println(ar.opisAranzmana());
             System.out.println("Cena turistickog aranzmana za skole: " +
                     ar.cenaAranzmana()
                     + ". U ponudi za ekskurzije su zimovanja "
                     + "i letovanja u gradovima Srbije"
                     + ", kao i maturantske ekskurzije u Italiju, "
                     + "Grcku i Spaniju."
                     + " Placanje je moguce na 10 rata."
                     + "Ukoliko broj prijavljenih ucenika bude veci od 200, "
                     + "dobijate popust od 10% po uceniku"
                     + " i tada ce cena aranzmana iznositi " +
                     ar.cenaAranzmana() * 0.9);
         }
     }
}
```

Klasa Vrtic (ConcreteObserverC):

```
public class Vrtic implements Ustanova {

    String nazivVrtica;
    KatalogPonuda katalog;
    private List<ApstraktniAranzman> aranzmani;

    public String getNazivVrtica() {
        return nazivVrtica;
    }

    public Vrtic(String nazivVrtica) {
        this.nazivVrtica = nazivVrtica;
        aranzmani = new ArrayList<ApstraktniAranzman>();
    }

    @Override
    public void update() {
        aranzmani.clear();
        aranzmani.addAll(katalog.vratiListu());
    }

    @Override
    public void registrujSe(KatalogPonuda katalog) {
        this.katalog = katalog;
    }

    @Override
    public void vratiOpis() {
        System.out.println(getNazivVrtica());
        for (ApstraktniAranzman ar : aranzmani) {
            System.out.println(ar.opisAranzmana());
            System.out.println("Cena turistickog aranzmana za vrtice: "
                    + ar.cenaAranzmana()
                    + ". Posto vodite na rekreativnu nastavu "
                    + "decu mladju od 7 godina"
                    + " u ponudi za Vasu ustanovu su aranzmani na"
                    + " obliznjim planinama za zimovanje, a letovanje u CG"
                    + ", placanje je moguce na 7 rata. "
                    + "Ukoliko broj prijavljene dece bude veci od 300,"
                    + " dobijate popust od 15% po detetu"
                    + " i tada ce cena aranzmana iznositi " +
                    ar.cenaAranzmana() * 0.85);
        }
    }
}
```

Dodatne klase korišćene u ovom primeru:

```
public abstract class ApstraktniAranzman {

    public abstract String opisAranzmana();
    public abstract int cenaAranzmana();
}

public class Letovanje extends ApstraktniAranzman {

    @Override
    public String opisAranzmana() {
        return "\nTuristicka agencija World travell ima neverovatne ponude"
                + " za letovanja. "
                + "Godinama vodimo turiste na najlepse destinacije"
                + ", uz nisku cenu, i proveren kvalitet."
                + " Ponude su za mesec jul, smestaj je u vilama, "
                + "a prevoz avionom.";
    }

    @Override
    public int cenaAranzmana() {
        return 20000;
    }
}
public class Zimovanje extends ApstraktniAranzman {

    @Override
    public String opisAranzmana() {
        return "\nTuristicka agencija World travell ima neverovatne ponude "
                + "za zimovanja. "
                + "Za ovih par godina koliko uspesno poslujemo"
                + " vodimo turiste na planine"
                + ", uz nisku cenu, i proveren kvalitet."
                + " Ponude su za mesec januar, smestaj je u hotelima,"
                + " a prevoz autobusom.";
    }

    @Override
    public int cenaAranzmana() {
        return 15000;
    }
}
```

Glavna (main) klasa:

```
public class Glavna {

    public static void main(String[] args) {
        ApstraktniAranzman letovanje = new Letovanje();
        ApstraktniAranzman zimovanje = new Zimovanje();
        KatalogPonuda aranzmani = KatalogPonuda.vratiKatalog();

        aranzmani.dodajPonudu(letovanje);

        Skola skola = new Skola("Emilija Ostojic");
        Fakultet fakultet = new Fakultet("FON");
        Fakultet fakultet1 = new Fakultet("FPN");

        skola.registrujSe(aranzmani);
        fakultet.registrujSe(aranzmani);
        fakultet1.registrujSe(aranzmani);

        aranzmani.dodajOsluskivac(skola);
        aranzmani.dodajOsluskivac(fakultet);
        aranzmani.dodajOsluskivac(fakultet1);

        aranzmani.dodajPonudu(zimovanje);

         System.out.println("Skole: \n");
         skola.vratiOpis();
         System.out.println("_____");
         System.out.println("Fakulteti: \n");
         fakultet.vratiOpis();
         fakultet1.vratiOpis();
         System.out.println("_____");
         System.out.println("");
    }
}
```

Rezultat izvršavanja programa:

```
Skole:

Emilija Ostojic

Turisticka agencija World travell ima neverovatne ponude za letovanja. Godinama vodimo
turiste na najlepse destinacije, uz nisku cenu, i proveren kvalitet. Ponude su za mesec jul,
smestaj je u vilama, a prevoz avionom.
Cena turistickog aranzmana za skole: 20000. U ponudi za ekskurzije su zimovanja i letovanja u
gradovima Srbije, kao i maturantske ekskurzije u Italiju, Grcku i Spaniju. Placanje je moguce na
10 rata.Ukoliko broj prijavljenih ucenika bude veci od 200, dobijate popust od 10% po uceniku i
tada ce cena aranzmana iznositi 18000.0
```

Turisticka agencija World travell ima neverovatne ponude za zimovanja. Za ovih par godina koliko uspesno poslujemo vodimo turiste na planine, uz nisku cenu, i proveren kvalitet. Ponude su za mesec januar, smestaj je u hotelima, a prevoz autobusom.
Cena turistickog aranzmana za skole: 15000. U ponudi za ekskurzije su zimovanja i letovanja u gradovima Srbije, kao i maturantske ekskurzije u Italiju, Grcku i Spaniju. Placanje je moguce na 10 rata.Ukoliko broj prijavljenih ucenika bude veci od 200, dobijate popust od 10% po uceniku i tada ce cena aranzmana iznositi 13500.0

Fakulteti:

FON

Turisticka agencija World travell ima neverovatne ponude za letovanja. Godinama vodimo turiste na najlepse destinacije, uz nisku cenu, i proveren kvalitet. Ponude su za mesec jul, smestaj je u vilama, a prevoz avionom.
Cena turistickog aranzmana za Fakultete: 20000. U ponudi za fakultete su najludja party putovanja, apsolventske ekskurzije u Italiju, Spaniju i Francusku. Placanje je moguce na 7 rata.Ukoliko broj prijavljenih studenata bude veci od 100, dobijate popust od 5% po studentu i tada ce cena aranzmana iznositi 19000.0

Turisticka agencija World travell ima neverovatne ponude za zimovanja. Za ovih par godina koliko uspesno poslujemo vodimo turiste na planine, uz nisku cenu, i proveren kvalitet. Ponude su za mesec januar, smestaj je u hotelima, a prevoz autobusom.
Cena turistickog aranzmana za Fakultete: 15000. U ponudi za fakultete su najludja party putovanja, apsolventske ekskurzije u Italiju, Spaniju i Francusku. Placanje je moguce na 7 rata.Ukoliko broj prijavljenih studenata bude veci od 100, dobijate popust od 5% po studentu i tada ce cena aranzmana iznositi 14250.0
FPN

Turisticka agencija World travell ima neverovatne ponude za letovanja. Godinama vodimo turiste na najlepse destinacije, uz nisku cenu, i proveren kvalitet. Ponude su za mesec jul, smestaj je u vilama, a prevoz avionom.
Cena turistickog aranzmana za Fakultete: 20000. U ponudi za fakultete su najludja party putovanja, apsolventske ekskurzije u Italiju, Spaniju i Francusku. Placanje je moguce na 7 rata.Ukoliko broj prijavljenih studenata bude veci od 100, dobijate popust od 5% po studentu i tada ce cena aranzmana iznositi 19000.0

Turisticka agencija World travell ima neverovatne ponude za zimovanja. Za ovih par godina koliko uspesno poslujemo vodimo turiste na planine, uz nisku cenu, i proveren kvalitet. Ponude su za mesec januar, smestaj je u hotelima, a prevoz autobusom.
Cena turistickog aranzmana za Fakultete: 15000. U ponudi za fakultete su najludja party putovanja, apsolventske ekskurzije u Italiju, Spaniju i Francusku. Placanje je moguce na 7 rata.Ukoliko broj prijavljenih studenata bude veci od 100, dobijate popust od 5% po studentu i tada ce cena aranzmana iznositi 14250.0

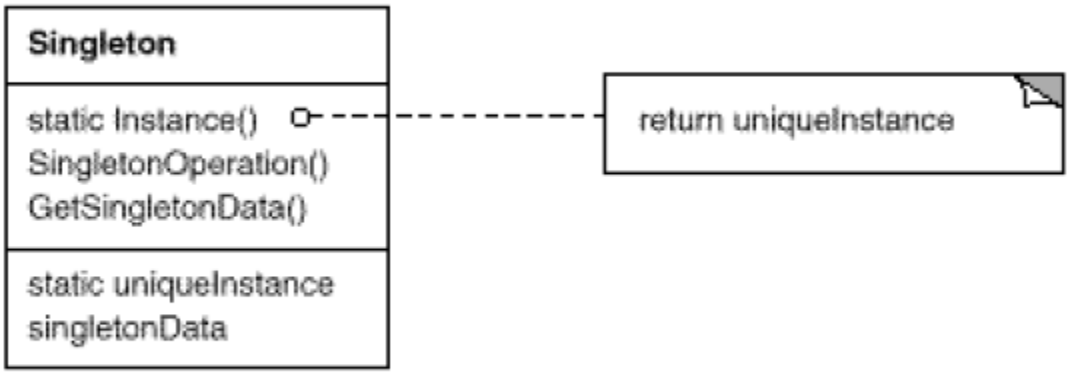
BUILD SUCCESSFUL (total time: 0 seconds)

6.2. SINGLETON

6.2.1. Definicija

Singleton patern obezbeđuje klasi samo jedno pojavljivanje i globalni pristup do nje.

6.2.2. Struktura



Učesnici:

- **Singleton**- definiše Instance() operaciju koja omogućava klijentima pristup do njenog jedninstvenog pojavljivanja

6.2.3. Primer turističke agencije

U primeru zaobserver patern, obezbeđeno je da se instanca katalogPonuda pojavljuje samo jedan put, odnosno da ima jedinstveno pojavljivanje (Singleton). Prilikom pojavljivanja nove ponude, ili brisanja postojeće, ne treba da se pravi nova lista, već se sve promene vrše nad jednom, postojećom listom.

6.2.4. Programski kod

Deo klase KatalogPonuda preuzete iz primera za Observer patern:

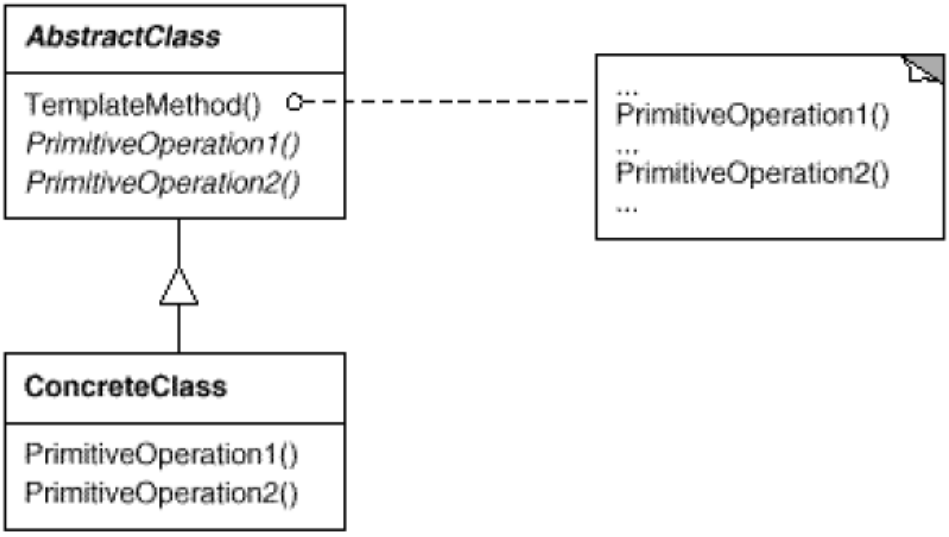
```
public class KatalogPonuda extends Zaposleni {

    private List<ApstraktniAranzman> aranzmani;
    private static KatalogPonuda kp;

    public static KatalogPonuda vratiKatalog() {
        if (kp == null) {
            kp = new KatalogPonuda();
        }
        return kp;
    }

    public KatalogPonuda() {
        aranzmani = new ArrayList<ApstraktniAranzman>();
    }
```

6.3. TEMPLATE METHOD

6.3.1. Definicija

Definiše skelet algoritma u operaciji prepuštajući izvršenje nekih koraka operacija podklasama. Template method omogućava podklasama da redefinišu neke od koraka algoritama bez promene algoritamske strukture.

6.3.2. Struktura



Učesnici:

- **AbstractClass**

Definiše apstraktne primitivne operacije koje konkretna podklasa implementira.
Implementira algoritam template method-a. Template method poziva primitivne operacije.

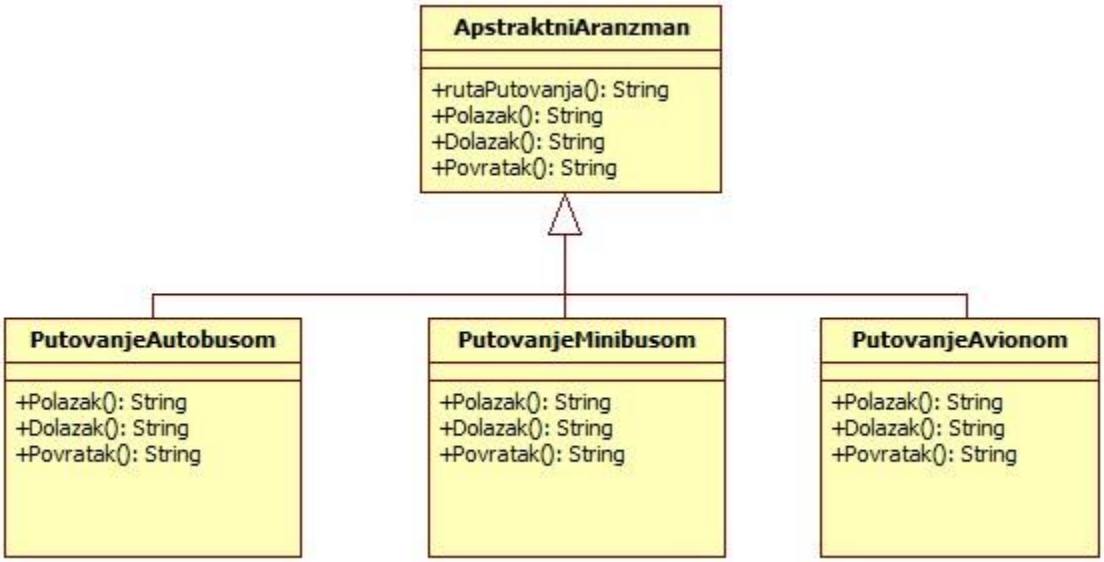
- **ConcreteClass**

Implementira primitivne operacije koje opisuju specifična ponašanja podklasa.

6.3.3. Primer turističke agencije

Svaki aranžman turističke agencije ima tačno definisanu rutu putovanja. Ruta putovanja započinje polaskom, zatim sledi dolazak na željenu destinaciju, i na kraju povratak u Beograd. Agencija organizuje putovanja tako što na početku odredi vremenski okvir, kad se kreće, za koliko se stiže na određenu lokaciju, i kada je planiran povratak u Beograd. Svaka ponuda se po tome razlikuje, odnosno svaka ponuda ima drugačije vreme polaska i povratka. Ukoliko agencija poseduje ponudu za letovanje u Italiji, onda samim tim ima i 3 različite rute putovanja, u zavisnosti od toga kojim se prevoznim sredstvom ide na putovanje. Ukoliko se putuje autobusom kreće se ranije, i vraća kasnije, jer je putovanje znatno duže nego avionom. Pored toga, agencija turiste može da vodi na putovanja i minibusom, za koji takođe definiše posebnu rutu putovanja, u skladu sa vremenom putovanja.

Za rešenje ovog problema korišćen je template patern zbog toga što on definiše skelet algoritma u operaciji prepuštajući izvršenje nekih koraka operacija podklasama. Odnosno, definiše skelet metode String rutaPutovanja(), pri čemu korake tog algoritma polazak(), povratak () i dolazak () redefinišu podklase PutovanjeAutobusom, PutovanjeAvionom i PutovanjeMiniBusom.



5.2.4. Programski kod

Klasa ApstraktniAranzman (AbstractClass)

```
public abstract class ApstraktniAranzman {

    String rutaPutovanja() {
        System.out.println("Ruta putovanja:");
        String pom = polazak();
        pom = pom + polazak();
        pom = pom + povratak();
        return pom;
    }

    abstract String polazak();

    abstract String dolazak();

    abstract String povratak();
```

Klasa PutovanjeAutobusom (ConcreteClassA):

```
public class PutovanjeAutobusom extends ApstraktniAranzman {

    @Override
    String polazak() {
        return "Polazak za putnike koji na putovanje idu autobusom"
                + " je ispred agencije 25.7.2013. u 21 h.\n";
    }

    @Override
    String dolazak() {
        return "Na zeljenu lokaciju stizemo ujutru, 26.7.2013.\n";
    }

    @Override
    String povratak() {
        return "Polazak za Beograd je planiran za 11.8.2013.\n";
    }
}
```

Klasa PutovanjeMinibusom (ConcreteClassB):

```
public class PutovanjeMiniBusom extends ApstraktniAranzman {

    @Override
    String polazak() {
        return "Polazak za putnike koji putuju minibusom"
                + " je 24.7.2013. u 19h.\n";
    }

    @Override
    String dolazak() {
        return "Dolazak na zeljenu lokaciju je planiran za 23:30 casova.\n";
    }

    @Override
    String povratak() {
        return "Za Beograd se krece 12.8.2013.\n";
    }
}
```

Klasa PutovanjeAvionom (ConcreteClassC):

```
public class PutovanjeAvionom extends ApstraktniAranzman {

    @Override
    String polazak() {
        return "Polazak za putnike koji putuju avionom je 27.7.2013.godine.u 6 casova.\n";
    }

    @Override
    String dolazak() {
        return "Na zeljenu lokaciju stizemo za 6 sati.\n";
    }

    @Override
    String povratak() {
        return "Povratak za Beograd je 10.8.2013.\n";
    }
}
```

Pomoćna klasa korišćena u ovom primeru:

```
public class Stranka {
    static ApstraktniAranzman aranzman;
}
```

Glavna (main) klasa:

```
public class Glavna {

    public static void main(String[] args) {
        Stranka stranka1 = new Stranka();
        ApstraktniAranzman aranzman1 = new PutovanjeAutobusom();
        System.out.println(aranzman1.rutaPutovanja());
        System.out.println("__________________");

        Stranka stranka2 = new Stranka();
        ApstraktniAranzman aranzman2 = new PutovanjeAvionom();
        System.out.println(aranzman2.rutaPutovanja());
        System.out.println("__________________");

        Stranka stranka3 = new Stranka();
        ApstraktniAranzman aranzman3 = new PutovanjeMiniBusom();
        System.out.println(aranzman3.rutaPutovanja());
        System.out.println("__________________");
    }
}
```

Rezultat izvršavanja programa:

```
run:
Ruta putovanja:
Polazak za putnike koji na putovanje idu autobusom je ispred agencije 25.7.2013. u 21 h.
Na zeljenu lokaciju stizemo ujutru, 26.7.2013.
Polazak za Beograd je planiran za 11.8.2013.


__________________
Ruta putovanja:
Polazak za putnike koji putuju avionom je 27.7.2013.godine.u 6 casova.
Na zeljenu lokaciju stizemo za 6 sati.
Povratak za Beograd je 10.8.2013.


__________________
Ruta putovanja:
Polazak za putnike koji putuju minibusom je 24.7.2013. u 19h.
Dolazak na zeljenu lokaciju je planiran za 23:30 casova.
Za Beograd se krece 12.8.2013.


__________________
BUILD SUCCESSFUL (total time: 1 second)
```

7. ZAKLJUČAK

Najbolji način da koristimo softverske paterne jeste da se što bolje upoznamo sa njima, da počnemo da prepoznajemo mesta u našim aplikacijama gde možemo da ih upotrebimo. Umesto da se mučimo ili kopiramo kod , uz pomoć paterna mi kopiramo tuđa iskustva.

U ovom seminarskom radu objašnjeno je 8 softverskih paterna na primeru turističke agencije. Susreli smo se sa raznim problemima, korisničkim zahtevima, koje smo uspešno rešili primenom nekog paterna. Patern je upravo rešenje tog nekog problema, koje se može ponovo iskoristiti. Ukoliko se ponovo susretnemo sa nekim sličnim problemom znaćemo kako da ga rešimo. U ovom seminarskom radu je za svaki patern naveden problem, a zatim i konkretno rešenje tog problema.

Pokazali smo kako dodavanje funkcionalnosti u postojeći programski kod može da se kontroliše, i spreči nastanak totalnog haosa usled postojanja špageti koda. Ideja je da napravimo takav softverski system koji će se lako prilagoditi svakom novom korisničkom zahtevu.

8. LITERATURA

Gotova rešenja - elementi objektno orijentisanog softvera, Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides