FAKULTET ORGANIZACIONIH NAUKA
UNIVERZITET U BEOGRADU

Seminarski rad
iz predmeta
Softverski paterni

***Tema*:**

Primena softverskih paterna na primeru Agencije za ketering usluge „Ivana Bajović“

Profesor:
Dr Siniša Vlajić

Asistenti:
Dušan Savić
Ilija Antović
Vojislav Stanojević
Miloš Milić

Student:
Ivana Bajović 40/09

Beograd, januar 2013.

Sadržaj

Design Patterns

Da li Vam se dešava da rešavate neki krajnje jednostavan problem, izgubite vreme pa Vam prijatelj kaže da je to već rešio?Ili da je možda već našao nečije prilagodljivo rešenje? Šta se dešava kada nastupe složeniji problemi? Da li ih je možda neko pre nas rešio samo u nekom drugom kontekstu pa da možemo to da iskoristimo?

„Šta su to zapravo paterni?“

Paterni, odnosno uzori projektovanja su iskustva u projektovanju objektno orijentisanih softvera. 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. Identifikuju klase i pojavljivanja, njihove uloge i saradnju i raspodelu odgovornosti. Pomažu u imenovanju i opisu generičkih rešenja koja se mogu primeniti u različitim problemskim situacijama.

Paterni su svuda oko nas, mogu se naći u prirodi, građevinama, ljubavnim romanima, ali i u softveru. 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*“. Ova definicija se može primeniti u gotovo svakoj oblasti života, pa tako i u objektno orjentisanom projektovanju softvera.

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.

Rešenje (Design) treba da bude:

**Specifično (specific),* prilagođeno problemu.

**Dovoljno opšte (generic)* da ukaže na buduće probleme

Bitne tačke u razvoju softvera:

Uvodimo Apstraktni server!

Ne pravimo ključne zavisnosti između klijenta i konkretnog servera!

Paterni su podeljeni u tri grupe:

1. *Uzori za kreiranje objekata (Kreacioni uzori)* 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.*

Kreacioni
1.AbstractFactory
2.Builder
3.FactoryMethod
4.Prototype
5.Singleton

Strukturni
1.Adapter
2.Bridge
3.Composite
4.Decorator
5.Fasade
6.Flyweight
7.Proxy

Uzori ponašanja
1.Chain Of Responsibility
2.Command
3.Mediator
4.Memento
5.Observer

Ovo je studentski rad u kom će se kroz jednu interesantnu temu, kroz 9 različitih paterna iz različitih kategorija, predstaviti njihov veliki značaj. Svaki od njih će biti detaljno prikazan kroz primar agencije za ketering usluge "Ivana Bajović".

Agencija za ketering usluge „Ivana Bajović“

Agencija za ketering usluge „Ivana Bajović“ trenutnu svojim klijentima nudi dve vrste kuhinje: italijansku kuhinju i domaću (tradicionalnu) kuhinju. Za italijansku kuhinju, odnosnu pripremu obroka italijanske kuhinje zadužen je kulinarski tim “Italiano“, dok je za usluge tradicionalne-domaće kuhinje zadužen kulinarski tim „Tradicija“.

Ketering služba vrši i izradu menija. Svaki meni se sastoji iz tri obroka:
dezerta, glavnog jela i predjela.

Trenutno u ponudi, kao predjelo, su sledeći specijaliteti:
škampi sa bademom (italijanska kuhinja)
proja (domaća kuhinja)

Trenutno u ponudi, kao glavno, jelo su sledeći specijaliteti:
testenina (italijanska kuhinja)
gulaš (domaća kuhinja)

Trenutno u ponudi, kao dezert, su sledeći specijaliteti:
sladoled (italijanska kuhinja)
orasnice (domaća kuhinja)

1. Kreacioni paterni

1.1. AbstractFactory

Apstrakcija nam pruža zanemarivanje detalja. 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 uzor 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. Uz pomoć ovog uzora sistem uvek dobija prave objekte u odgovarajućem trenutku. 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. Koristeći ovaj uzor, postaje moguće promeniti konkretnu klasu bez menjanja koda koji je koristi, čak i u rantajmu. Međutim primena ovog uzora, kao i uzora poput njega, može izazvati nepotrebnu složenost i dodatni rad u početku pisanja koda.

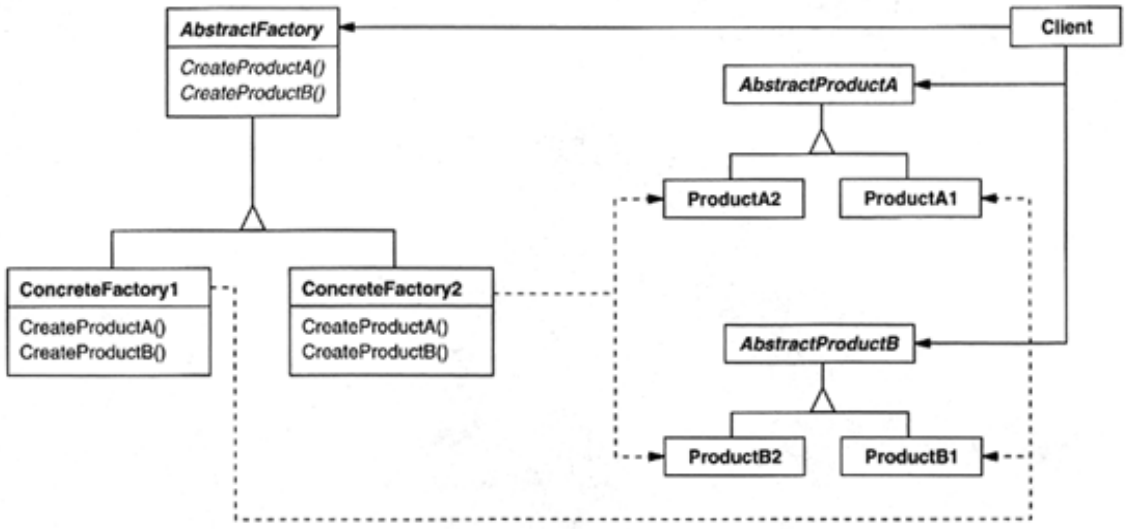
Definicija:

Obezbeđuje interfejs za kreiranje familije povezanih ili zavisnih objekata bez navođenja (specificiranja) njihovih konkretnih klasa.

Pojašnjena 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 kreiranja elemenata ponude i spaja te elemente u celini (ponudu). Za kreiranje elemenata ponude su odgovorne ConcreteFactory klase.



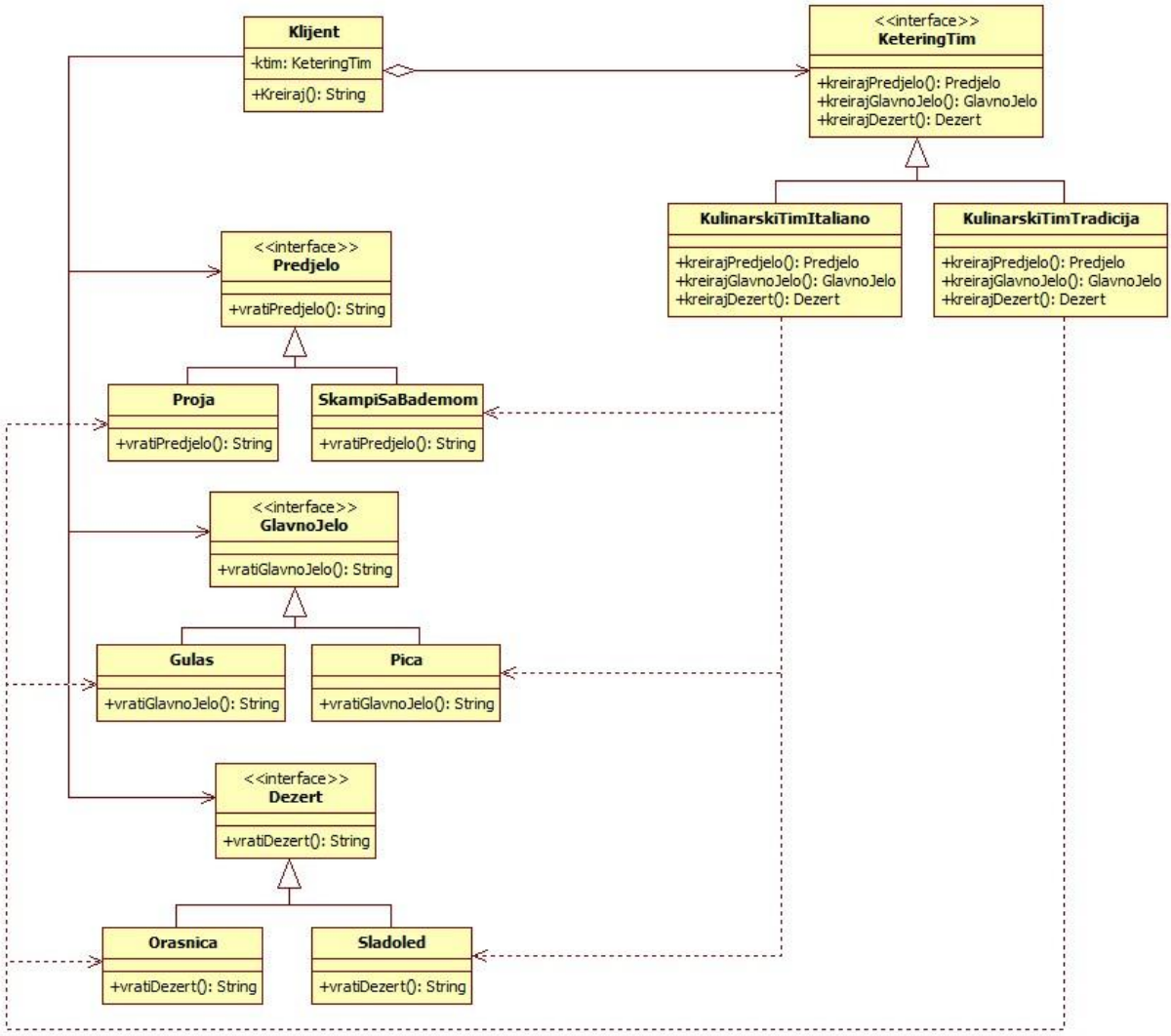
Primena Abstract Factory uzora na primeru agencije za ketering usluge:

Agencija za ketering usluge „Ivana Bajovic“ pruža usluge kreiranja menija najrazličitijim klijentima koji uvek imaju raznolike, drugačije ideje i želje.

Tako postoje klijenti koji od ketering službe zahtevaju samo da im se pošalju elementi menija odnosno da im se proslede obroci (predjelo, glavno jelo i dezert) pa će oni sami sastavljati meni po svojoj želji. Oni će takođe i vršiti kontrolu kako bi bili sigurni da se sve odvija kako treba.

Primena definicije na poslovanje agencije za ketering usluge:

Obezbeđuje interfejs **(*KeteringTim*)** za kreiranje***(kreirajPredjelo(), kreirajGlavnoJelo(), kreirajDezert())*** familije povezanih ili zavisnih objekata **(*Predjelo, GlavnoJelo, Dezert*)** bez navođenja (specificiranja) njihovih konkretnih klasa ***(*SkampiSaBademom, *Proja, Pica, Gulas, Sladoled, Orasnica)*** .


Klijent
-ktim: KeteringTim
+Kreiraj(): String
<<interface>>
KeteringTim
+kreirajPredjelo(): Predjelo
+kreirajGlavnoJelo(): GlavnoJelo
+kreirajDezert(): Dezert
KulinarskiTimItaliano
+kreirajPredjelo(): Predjelo
+kreirajGlavnoJelo(): GlavnoJelo
+kreirajDezert(): Dezert
KulinarskiTimTradicija
+kreirajPredjelo(): Predjelo
+kreirajGlavnoJelo(): GlavnoJelo
+kreirajDezert(): Dezert
<<interface>>
Predjelo
+vratiPredjelo(): String
Proja
+vratiPredjelo(): String
SkampiSaBademom
+vratiPredjelo(): String
<<interface>>
GlavnoJelo
+vratiGlavnoJelo(): String
Gulas
+vratiGlavnoJelo(): String
Pica
+vratiGlavnoJelo(): String
<<interface>>
Dezert
+vratiDezert(): String
Orasnica
+vratiDezert(): String
Sladoled
+vratiDezert(): String

U datom slučaju, primena Abstract Factory uzora je realizovana preko sledećih klasa/interfejsa:

- **KeteringTim** (AbstractFactory)

Definiše interfejs za operacije koje kreiraju proizvode odnosno predjela, glavna jela i dezerte (metode *kreirajPredjelo (), kreirajGlavnoJelo () i kreirajDezert ()*)

- **KulinarskiTimItaliano, KulinarskiTimTradicija** (ConcreteFactory)

Klase koje implementiraju operacije interfejsa KeteringTim, kojima se kreiraju konkretni prizvodi, odnosno konkretno glavno jelo, konkretno predjelo, konkretan dezert.
Klasa KulinarskiTimItaliano kreira instance klasa Pica, SkampiSaBademom i Sladoled.
Klasa KulinarskiTimItaliano kreira instance klasa Gulas, Proja i Orasnica.

- **Predjelo, GlavnoJelo i Dezert** (AbstractProduct)

Definišu interfejs za proizvode, odnosno predjelo, glavno jelo i dezert

- **Sladoled, Orasnica** (ConcreteProduct1)

Klase koje implementiraju interfejs Dezert, definišu konkretne proizvode koji će biti kreirani preko klase KulinarskiTimItaliano, odnosno klase KulinarskiTimTradicija.

- **Pica, Gulas** (ConcreteProduct2)

Klase koje implementiraju interfejs GlavnoJelo, definišu konkretne proizvode koji će biti kreirani preko klase KulinarskiTimItaliano, odnosno klase KulinarskiTimTradicija.

- **SkampiSaBademom, Proja** (ConcreteProduct3)

Klase koje implementiraju interfejs Predjelo, definišu konkretne proizvode koji će biti kreirani preko klase KulinarskiTimItaliano, odnosno klase KulinarskiTimTradicija.

- **Klijent** (Client)

Koristi klase KulinarskiTimItaliano, KulinarskiTimTradicija, Sladoled, Orasnica, Pica, Gulas, SkampiSaBademom, Proja kako bi kreirao finalni proizvod (meni).

Dakle, Klijent, koji čuva referencu na KeteringTim, od KeteringTima dobija samo kreirane pojedinačne proizvode, odnosno obroke (predjelo, glavno jelo i dezert), a zatim sam kreira meni odnosno isporučene mu obroke sklapa u celinu (metoda *Kreiraj()*). Uz to Klijent je taj koji kontroliše proces izrade elemenata menia (obroka).
KeteringTim je zadužen samo za kreiranje obroka.

Programski kod:

```
package abstractfactoryketering;

/**
 *
 * @author Ivana
 */
public interface KeteringTim  // AbstractFactory {

    Predjelo kreirajPredjelo();

    GlavnoJelo kreirajGlavnoJelo();

    Dezert kreirajDezert(); }
```

```
public class KulinarskiTimItaliano implements KeteringTim // ConcreteFactory1{

   public Predjelo kreirajPredjelo() {
      return new SkampiSaBademom();
   }
   public GlavnoJelo kreirajGlavnoJelo() {
      return new Pica();
   }
   public Dezert kreirajDezert() {
      return new Sladoled();
   }
}

public class KulinarskiTimTradicija implements KeteringTim // ConcreteFactory2 {

   public Predjelo kreirajPredjelo() {
      return new Proja();
   }
   public GlavnoJelo kreirajGlavnoJelo() {
      return new Gulas();
   }
   public Dezert kreirajDezert() {
      return new Orasnica();
   }
}

public class Klijent //Client {
        private KeteringTim ktim;

        public Klijent(KeteringTim ktim) {
            this.ktim = ktim;
        }
        public String Kreiraj() {
            Predjelo predjelo = ktim.kreirajPredjelo();
            GlavnoJelo glavnojelo = ktim.kreirajGlavnoJelo();
            Dezert dezert = ktim.kreirajDezert();
            return "Meni se sastoji iz:\n Predjelo - "
                    + predjelo.vratiPredjelo()
                    + ", glavno jelo - "
                    + glavnojelo.vratiGlavnoJelo()
                    + " i dezert -"
                    + dezert.vratiDezert();
         }
}
```

```
public interface Predjelo  // AbstractProductA {
   String vratiPredjelo ();
}

public class SkampiSaBademom implements Predjelo   //ProductA1 {
   public String vratiPredjelo () {
      return " Skampi sa bademima u umaku ";
   }
}

public class Proja implements Predjelo   //ProductA2 {
   public String vratiPredjelo () {
      return " Proja sa spanacem";
   }
}

public interface GlavnoJelo // AbstractProductB {
   String vratiGlavnoJelo ();
}

public class Gulas implements GlavnoJelo  //ProductB1 {
   public String vratiGlavnoJelo () {
      return "Gulas";
   }
}

public class Pica implements GlavnoJelo  //ProductB2 {
   public String vratiGlavnoJelo () {
      return " Pica ";
   }
}

public interface Dezert  // AbstractProductC {
   String vratiDezert ();
}

public class Orasnica implements Dezert  //ProductC1 {
   public String vratiDezert () {
      return " Orasnica ";
   }
}
```

```
public class Sladoled implements Dezert  //ProductC2 {
   public String vratiDezert () {
      return "Sladoled ";
   }
}

public class Main // Main class {
   public static void main(String[] args) {
      Klijent klijent;

      KulinarskiTimTradicija kttradicija = new KulinarskiTimTradicija(); //ConcreteFactory1
      klijent = new Klijent(kttradicija);
      System.out.println(klijent.Kreiraj());

      KulinarskiTimItaliano ktitaliano = new KulinarskiTimItaliano(); //ConcreteFactory2
      klijent = new Klijent(ktitaliano);
      System.out.println(klijent.Kreiraj());
   }
}
```

Rezultat programa:

```
Meni se sastoji iz:
 Predjelo:  Proja sa spanacem
 Glavno jelo:  Gulas
 Dezert:  Orasnica
-----------------------------------------
Meni se sastoji iz:
 Predjelo:  Skampi sa bademima u umaku
 Glavno jelo:   Pica
 Dezert: Sladoled
```

1.2. Builder uzor

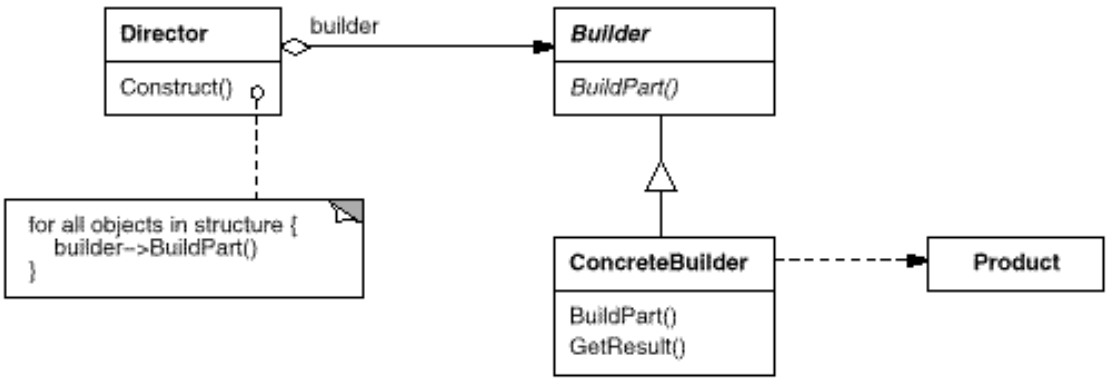
Definicija:

Builder patern razdvaja izgradnju složenog objekta od njegove reprezentacije da bi isti proces pravljenja mogao da proizvede različite reprezentacije.

Pojašnjena definicija:

Deli odgovornost za kontrolu konstrukcije **(*Director*)** složenog (kompleksnog) objekta od odgovornosti za realizaciju njegove reprezentacije-konkretne konstrukcije ***(Builder),*** tako da isti konstrukcioni proces **(*Director.Construct()*)** može da kreira različite reprezentacije konkretne konstrukcije ***(ConcreteBuilder.BuildPart())*** u zavisnosti od ConcreteBuilder-a.

Direktor (klijent) samo nadzire (kontroliše) proces kreiranja elemenata ponude, dok je odgovornost i kreiranje elemenata ponude i spajanja elemenata u celinu, odnosno ponudu, preneta na ConcreteBuilder klase.



Karakteristike Builder paterna:

1. Omogućava menjanje unutrašnje predstave proizvoda. Ukoliko želimo da promenimo unutrašnju predstavu proizvoda, treba samo dodati novu vrstu buildera, a to nam omogucava 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.

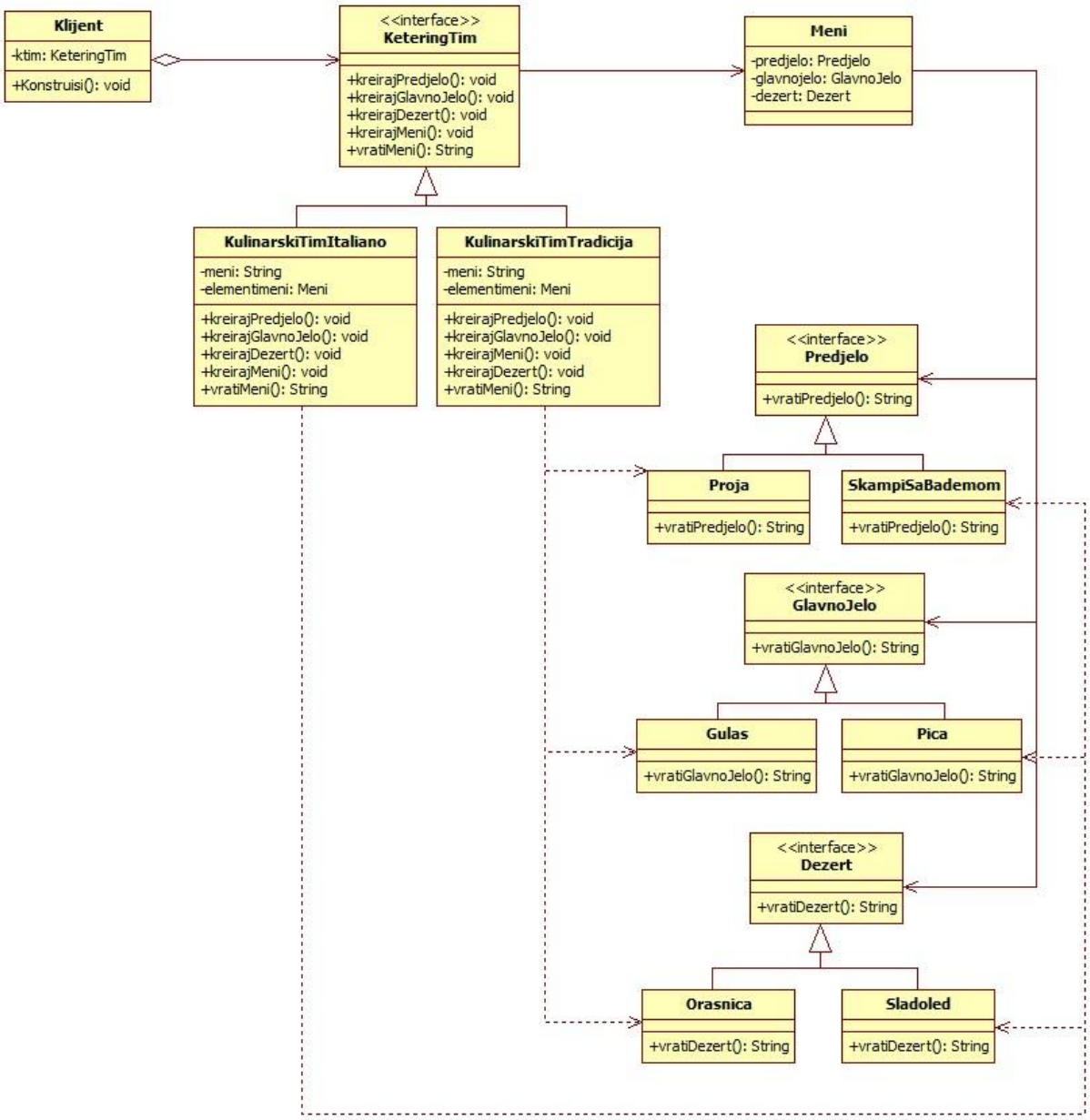
4. U interfejsu se ne pojavljuju klase koje definisu unutrašnju strukturu proizvoda, pa prema tome klijent ne mora ništa da zna o njima.

Primana Builder uzora na primeru agencije za ketering usluge:

Takođe postoje i klijenti koji od ketering službe zahtevaju da im naprave i isporuče gotove menije, dok će oni samo vršiti kontrolu kako bi bili sigurni da se sve odvija kako treba .

Primena definicije na Agenciju za ketering usluge:

Deli odgovornost za kontrolu konstrukcije **(*Klijent*)** složenog (kompleksnog) objekta od odgovornosti za realizaciju njegove reprezentacije-konkretne konstrukcije ***(KeteringTim),*** tako da isti konstrukcioni proces **(*Klijent.Konstruisi()*)** može da kreira različite reprezentacije konkretne konstrukcije ***(KulinarskiTimItaliano.kreirajPredjelo(), KulinarskiTimItaliano.kreirajGlavnoJelo(), KulinarskiTimItaliano.kreirajDezert(), KulinarskiTimTradicija.kreirajPredjelo(), KulinarskiTimTradicija. kreirajGlavnoJelo(), KulinarskiTimTradicija. kreirajDezert())*** u zavisnosti od ConcreteBuilder-a **(*KulinarskiTimItaliano, KulinarskiTimTradicija)***.



Informacije i način izrade menija su na ovaj način skriveni od klijenta. Svaki meni je nezavisan, i zato što se sastavlja korak po korak, imamo veću kontrolu.

U datom slučaju, primena Builder uzora je realizovana preko sledećih klasa/interfejsa:

- **KeteringTim** (Builder)

Definiše interfejs za operacije koje kreiraju delovi proizvoda (menia), odnosno predjela, glavna jela i dezerte, i sam proizvod - meni (metode *kreirajPredjelo (), kreirajGlavnoJelo (), kreirajDezert (), kreirajMeni() i vratiMeni()*)

- **KulinarskiTimItaliano, KulinarskiTimTradicija** (ConcreteBuilder)

Klase koje implementiraju operacije interfejsa KeteringTim, kojima se kreiraju konkretni delovi prizvoda, odnosno konkretno glavno jelo, konkretno predjelo, konkretan dezert.
Klasa KulinarskiTimItaliano kreira instance klasa Pica, SkampiSaBademom i Sladoled.
Klasa KulinarskiTimItaliano kreira instance klasa Gulas, Proja i Orasnica.
I još grupišu kreirane delove proizvoda u celinu, odnosno kreiraju meni, implementirajući metodu *kreirajMeni().*

- **Klijent** (Director)

Klasa koja čuva referencu na KeteringTim. On konstruiše proizvod, odnosno meni, i kontroliše ceo proces koristeći operacije interfejsa KetreingTim KulinarskiTimItaliano,

- **Meni** (Product)

Reprezentuje složen objekat (proizvod, odnosno meni) koji se konstruiše.
Klasa Meni uključuje interfejse Predjelo, GlavnoJelo i Dezert koji definišu delove menia (obroke) od kojih će se konstruisati sam meni.

Dakle, *Klijent* od *KeteringTima* ne dobija samo kreirane delove proizvoda, odnosno obroke (predjelo, glavno jelo i dezert), već dobija ceo meni.
KetreingTim je taj koji je zadužen za kreiranje delova proizvoda (obroka), i za sklapanje tih delova u celinu - za kreiranje menia, dok je Klijent taj koji samo kontroliše proces izrade menia.
KeteringTim je zadužen samo za kreiranje obroka.

Programski kod:

```
package builderketering;

/**
 *
 * @author Ivana
 */
public interface KeteringTim { // Builder

   void kreirajPredjelo();

   void kreirajGlavnoJelo();

    void kreirajDezert();

    void kreirajMeni();

    String vratiMeni();
}
```

```
public class Meni{
   private Predjelo predjelo;
   private GlavnoJelo glavnojelo;
   private Dezert dezert;

    public Dezert getDezert() {
       return dezert;
    }
    public void setDezert(Dezert dezert) {
       this.dezert = dezert;
    }
    public GlavnoJelo getGlavnojelo() {
       return glavnojelo;
    }
    public void setGlavnojelo(GlavnoJelo glavnojelo) {
       this.glavnojelo = glavnojelo;
    }
    public Predjelo getPredjelo() {
       return predjelo;
    }
    public void setPredjelo(Predjelo predjelo) {
       this.predjelo = predjelo;
    }
}

public class KulinarskiTimItaliano implements KeteringTim //ConcreteBuilder1{
    Meni elementimeni;
    String meni;

    public KulinarskiTimItaliano() {
       elementimeni = new Meni();
    }
    public void kreirajPredjelo() {
       elementimeni.setPredjelo(new SkampiSaBademom());
    }
    public void kreirajGlavnoJelo() {
       elementimeni.setGlavnojelo(new Pica());
    }
    public void kreirajDezert() {
       elementimeni.setDezert(new Sladoled());
    }
```

```
 public void kreirajMeni() {
       meni = "Meni se sastoji iz: \n Predjelo - "
             + elementimeni.getPredjelo().vratiPredjelo()
             + ", glavno jelo - "
             + elementimeni.getGlavnojelo().vratiGlavnoJelo()
             + " i dezert -"
             + elementimeni.getDezert().vratiDezert();
    }
    public String vratiMeni() {
       return meni;
    }
}

public class KulinarskiTimTradicija implements KeteringTim // ConcreteBuilder2  {
    Meni elementimeni;
    String meni;

    public KulinarskiTimTradicija() {
       elementimeni = new Meni();
    }
    public void kreirajPredjelo() {
       elementimeni.setPredjelo(new Proja());
    }
    public void kreirajGlavnoJelo() {
       elementimeni.setGlavnojelo(new Gulas());
    }
    public void kreirajDezert() {
       elementimeni.setDezert(new Orasnica());
    }
    public void kreirajMeni() {
       meni = "Meni se sastoji iz: \n Predjelo - "
             + elementimeni.getPredjelo().vratiPredjelo()
             + ", glavno jelo - "
             + elementimeni.getGlavnojelo().vratiGlavnoJelo()
             + " i dezert -"
             + elementimeni.getDezert().vratiDezert(); }
    public String vratiMeni() {
       return meni;
    }
}

public class Klijent // Director {
    KeteringTim ktim;
    public Klijent(KeteringTim ktim) {
       this.ktim = ktim;
    }
```

```
    void Konstruisi() {
        ktim.kreirajPredjelo();
        ktim.kreirajGlavnoJelo();
        ktim.kreirajDezert();
        ktim.kreirajMeni();
    }
}

public interface Predjelo {
    String vratiPredjelo ();
}

public class SkampiSaBademom implements Predjelo {
    public String vratiPredjelo () {
        return " Skampi sa bademima u umaku ";
    }
}

public class Proja implements Predjelo {
    public String vratiPredjelo () {
        return " Proja sa spanacem";
    }
}

public interface GlavnoJelo  {
    String vratiGlavnoJelo ();
}

public class Gulas implements GlavnoJelo{
    public String vratiGlavnoJelo () {
        return "Gulas";
    }
}

public class Pica implements GlavnoJelo {
    public String vratiGlavnoJelo () {
        return " Pica ";
    }
}

public interface Dezert {
    String vratiDezert ();
}
```

```
public class Orasnica implements Dezert{
   public String vratiDezert() {
      return " Orasnica ";
   }
}

public class Sladoled implements Dezert{
   public String vratiDezert() {
      return " Sladoled ";
   }
}

public class Main //Main class{
   public static void main(String[] args) {
      Klijent klijent;

      KulinarskiTimTradicija kttradicija = new KulinarskiTimTradicija(); //ConcreteBuilder1
      klijent = new Klijent(kttradicija);
      klijent.Konstruisi();
      System.out.println(kttradicija.vratiMeni());

      KulinarskiTimItaliano ktitaliano = new KulinarskiTimItaliano(); //ConcreteBuilder2
      klijent = new Klijent(ktitaliano);
      klijent.Konstruisi();
      System.out.println(ktitaliano.vratiMeni());
   }
}
```

Rezultat programa:

```
Meni se sastoji iz:
 Predjelo:  Proja sa spanacem
 Glavno jelo: Gulas
 Dezert: Orasnica
---------------------------------------
Meni se sastoji iz:
 Predjelo:  Skampi sa bademima u umaku
 Glavno jelo:  Pica
 Dezert:Sladoled
```

1.3. Factory method uzor

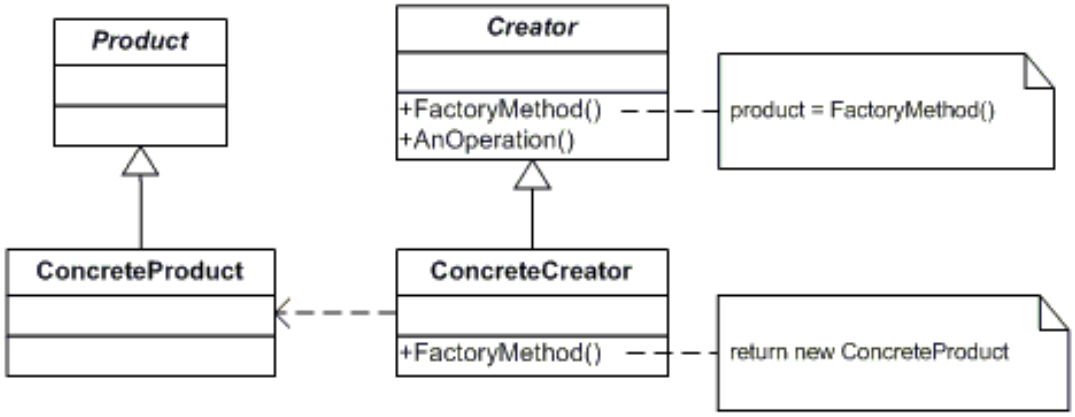
Definicija:

Factory method patern definiše interfejs za kreiranje objekata, ali omogućava podklasama da donesu odluku koju klasu će instancirati. Prenosi se nadležnost instanciranja objekata sa klase na podklase.

Pojašnjena definicija:

Definiše interfejs za kreiranje objekata **(*Creator*)**, ali omogućava podklasama ***(ConcreteCreator)*** da donesu odluku koju klasu (prozvod) će instancirati. Prenosi se nadležnost instanciranja objekata sa klase ***(Creator)*** na podklase ***(ConcreteCreator).***

Direktor (klijentska strana) nema nikakvo zaduženje, nikakvu odgovornosr, dok i kontrolu procesa kreiranja i kreiranje elemenata ponude i spajanje elemenata u celinu, odnosno ponudu, obavljaju ConcreteCreator klase (serverska strana).



ConcreteCreator nadzire proces pravljenja proizvoda i izrađuje delove i celinu proizvoda.

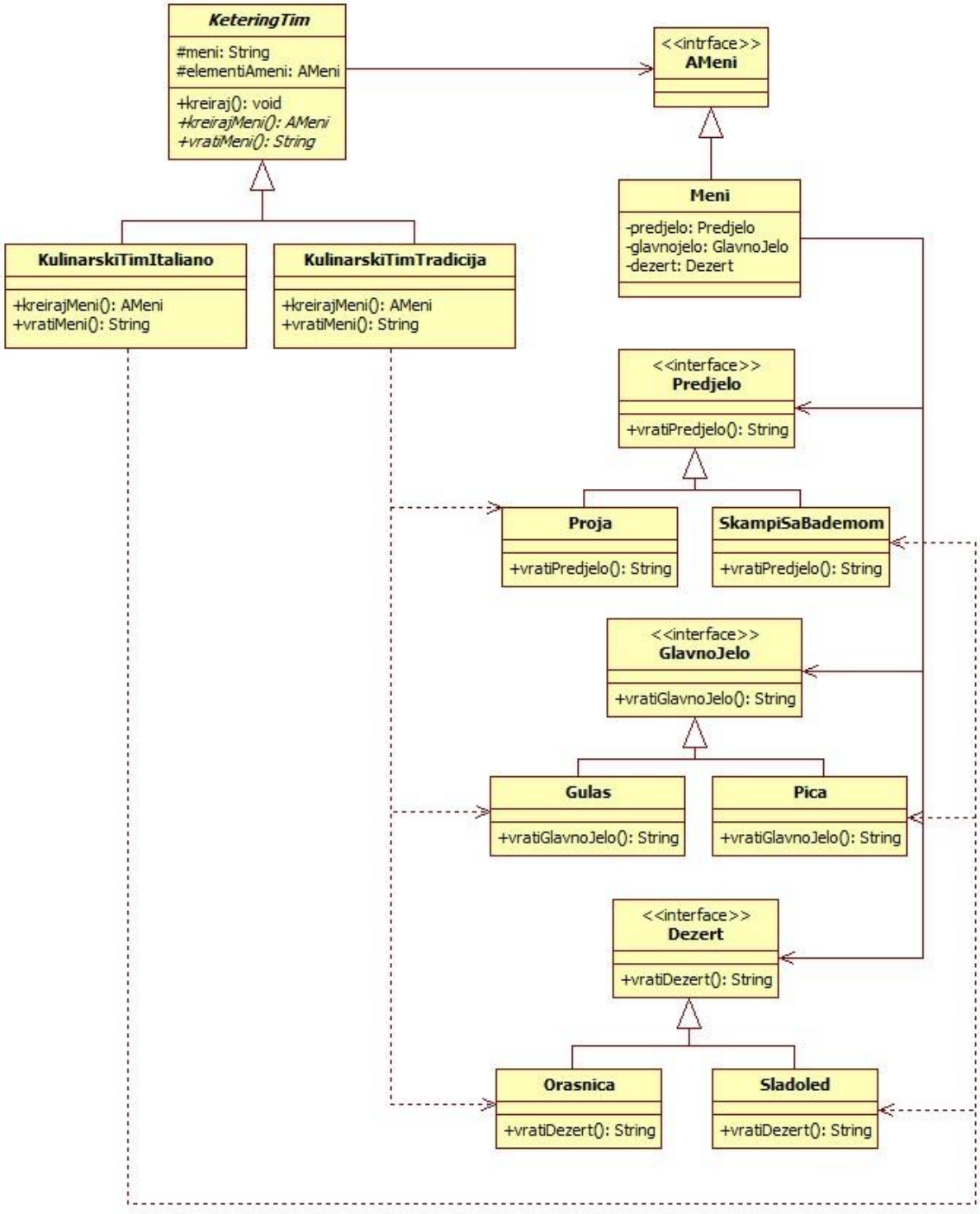
Primana Factory method uzora na primeru agencije za ketering usluge:

Treća vrsta klijenata su oni koji nemaju vremena da se bave čak ni kontrolom usluge. Oni odgovornost u potpunosti i za sve aspekte predaju ketering službi. Odnosno, zahtevaju od ketering službe da im naprave i isporuče gotove menije, stim što ketering služba vrši i kontrolu kreiranja kako bi sve prošlo u najboljem mogućem redu.

Primena definicije na Agenciju za ketering usluge:

Definiše interfejs za kreiranje objekata **(*KeteringTim*)**, ali omogućava podklasama ***(KulinarskiTimItaliano, KulinarskiTimTradicija)*** da donesu odluku koju klasu (proizvod) će

instancirati. Prenosi se nadležnost instanciranja objekata sa klase ***(KeteringTim)*** na podklase. ***KulinarskiTimItaliano, KulinarskiTimTradicija).***



U datom slučaju, primena Factory method uzora je realizovana preko sledećih klasa/interfejsa:

- **AMeni** (Product)

Definiše interfejs objekta (prozvoda) kiji se kreira od strane ketering tima

- **Meni** (ConcreteProduct)

Klas koja implementira interfejs AMeni. Reprezentuje objekat (konkretan proizvod, odnosno meni) koji se kreira. Klasa Meni uključuje interfejse Predjelo, GlavnoJelo i Dezert koji definišu delove menia (obroke)..

- **KeteringTim** (Creator)

Deklariše metodu *kreirajMeni()* koja predstavlja *FactorzMethod()* metodu. Ova metoda vraća kao rezultat objekat klase Meni. Vraća gotov proizvod, sklopljen meni.

- **KulinarskiTimItaliano, KulinarskiTimTradicija** (ConcreteBuilder)

Klase koje implementiraju interfejs KeteringTim. Implementiraju operaciju *kreirajMeni()* kojom se kreira meni (gotov proizvod). Svaka klasa implementira metodu na svoj način. Metoda klase KulinarskiTimItaliano *kreirajMeni()* vraća meni koji se sastoji iz sladoleda, skampa sa bademom i pice (implementacija metode: kreiraju se instance klasa Sladoled, Pica i SkampiSaBademom, a zatim objekat tipa Meni dobija refernce na kreirane obroke).
KulinarskiTimTradicija *kreirajMeni()* vraća meni koji se sastoji iz orasnice, proje i gulaša (implementacija metode: kreiraju se instance klasa Orasnica, Proja i Gulas, a zatim objekat tipa Meni dobija refernce na kreirane obroke).

U ovom slučaju i za kreiranje obroka (glavno jelo, predjelo i dezert), i za sklapanje tih obroka u celinu – kreiranje menia, kao i za kontrolu celog procesa zadužen je KeteringTim. Klijent nema nikakvih dodirnih tačaka sa procesom izrade. Njemu se samo predaje gotov meni dok su svi ostali koraci nepoznati za klijenta. Klijent nema nikakav uvid u stanje izrade, samo zna za postojenje KeteringTim-a i preko svoje metode *kreiraj()* on dobija od KeteringTim-a gotov meni, a KeteringTim u pozadini kreira elemente menia i sam meni.

Programski kod:

```
package factorymethodketering;

/**
 *
 * @author Ivana
 */

public class Klijent //Client {
        KeteringTim ktim; //Creator

        public Klijent(KeteringTim ktim) {
           this.ktim = ktim;
        }
        void kreiraj() {
           ktim.kreiraj();
        }
    }
```

```
interface AMeni{} // Product

public class Meni implements AMeni{ // ConcreteProduct
  public Predjelo predjelo;
  public GlavnoJelo glavnojelo;
  public Dezert dezert;

}

abstract class  KeteringTim // Creator{
   AMeni elementiAmeni;
   String meni;

   void kreiraj() {
      elementiAmeni = kreirajMeni();
   }
   abstract AMeni kreirajMeni();
   abstract String vratiMeni();
}

public class KulinarskiTimItaliano extends  KeteringTim //ConcreteCreator1{
   AMeni kreirajMeni() {
      Meni elementimeni = new Meni();
      elementimeni.predjelo = new SkampiSaBademom();
      elementimeni.glavnojelo = new Pica();
      elementimeni.dezert = new Sladoled();
      meni = "Meni se sastoji iz: \n Predjelo - "
            + elementimeni.predjelo.vratiPredjelo()
            + ", glavno jelo - "
            + elementimeni.glavnojelo.vratiGlavnoJelo()
            + " i dezert -"
            + elementimeni.dezert.vratiDezert();
      return elementimeni;
   }
   String vratiMeni() {
      return meni;
   }
}

public class KulinarskiTimTradicija implements KeteringTim //ConcreteBuilder2{
   AMeni kreirajMeni() {
      Meni elementimeni = new Meni();
      elementimeni.predjelo = new Proja();
      elementimeni.glavnojelo = new Gulas();
      elementimeni.dezert = new Orasnica();
```

```
        meni = "Meni se sastoji iz: \n Predjelo - "
                + elementimeni.predjelo.vratiPredjelo()
                + ", glavno jelo - "
                + elementimeni.glavnojelo.vratiGlavnoJelo()
                + " i dezert -"
                + elementimeni.dezert.vratiDezert();
            return elementimeni;
    }
    String vratiMeni() {
        return meni;
    }
}

public interface Predjelo {
    String vratiPredjelo ();
}

public class SkampiSaBademom implements Predjelo {
    public String vratiPredjelo () {
        return " Skampi sa bademima u umaku ";
    }
}

public class Proja implements Predjelo {
   public String vratiPredjelo () {
     return " Proja sa spanacem";
   }
}

public interface GlavnoJelo  {
    String vratiGlavnoJelo ();
}

public class Gulas implements GlavnoJelo{
    public String vratiGlavnoJelo () {
        return "Gulas";
    }
}

public class Pica implements GlavnoJelo {
    public String vratiGlavnoJelo () {
        return " Pica ";
    }
}
```

```
public interface Dezert {
   String vratiDezert ();
}

public class Orasnica implements Dezert{
   public String vratiDezert() {
      return " Orasnica ";
   }
}

public class Sladoled implements Dezert{
   public String vratiDezert() {
      return " Sladoled ";
   }
}

public class Main //Main class {
   public static void main(String[] args) {
      Klijent klijent;

      KulinarskiTimTradicija kttradicija = new KulinarskiTimTradicija(); //ConcreteCreator1
      klijent = new Klijent(kttradicija);
      klijent.kreiraj();
      System.out.println(kttradicija.vratiMeni());

      KulinarskiTimItaliano ktitaliano = new KulinarskiTimItaliano(); //ConcreteCreator2
      klijent = new Klijent(ktitaliano);
      klijent.kreiraj();
      System.out.println(kttradicija.vratiMeni());
   }
}
```

Rezultat programa:

```
Meni se sastoji iz:
 Predjelo:  Proja sa spanacem
 Glavno jelo: Gulas
 Dezert:  Orasnica
---------------------------------
Meni se sastoji iz:
 Predjelo:  Skampi sa bademima u umaku
 Glavno jelo:  Pica
 Dezert: Sladoled
```

2. Strukturni paterni

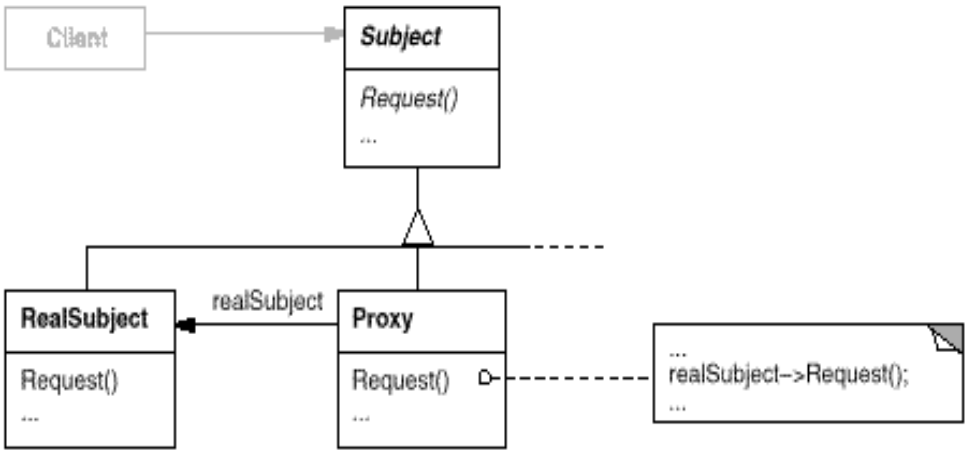
Uvod pred naredne zahteve:
Klijent Julio Pitassi organizuje novogodišnju proslavu za svoje zaposlene. Obratio se agenciji za ketering usluge "Ivana Bajović". On je poslao zahtev osoblju agencije da mu naprave ponudu odnosno meni italijanske kuhinje. Julio je poreklom iz Rima pa bi želeo svojim zaposlenima putem hrane da približi doživljaj svoje zemlje. Meni treba da obuhvati predjelo, glavno jelo i dezert. Julio će nadgledati proces kako bi sve prošlo u najboljem redu (primena ***Builder uzora – kreacioni patern,*** tako da su sledeći kod i sledeće primene drugih paterna nadograđeni na primenjen Builder uzor).

2.1. Proxy

Dobar razlog za kontrolisanje pristupa objektu jeste da se odlože puni troškovi njegovog pravljenja i inicijalizovanja do trenutka kada zaista treba da ga upotrebimo.

Definicija:
Obezbeđuje posrednika za pristupanje drugom objektu kako bi se omogućio kontrolisani pristup do njega.

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

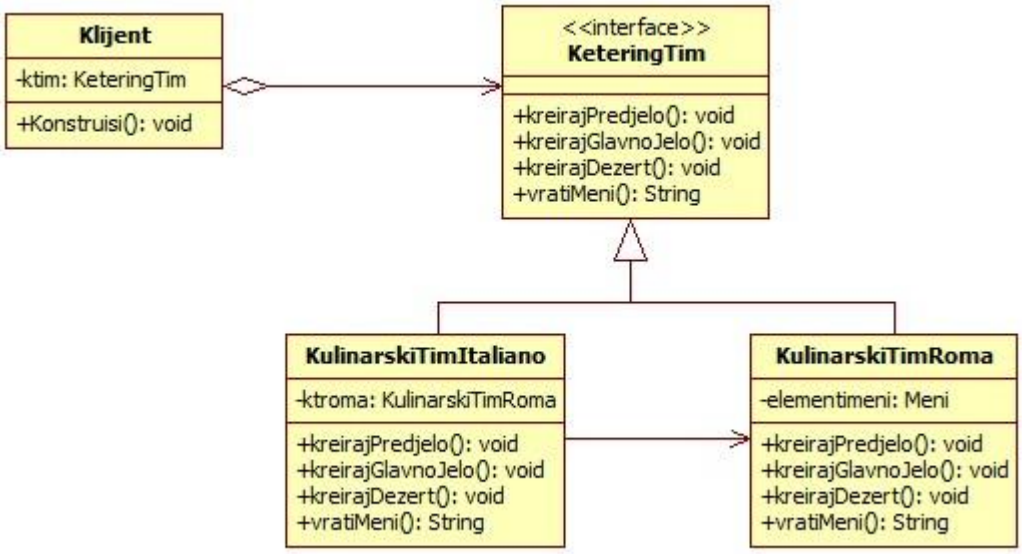


Primana Proxy uzora na primeru agencije za ketering usluge:

Gospodin Julio Pitassi je poreklom Italijan i u potpunosti je upoznat sa svim ukusima italijanske kuhinje, a pritom je i veoma kritičan i zahtevan. Očekuje besprekornu uslugu i savršeno pripremljenu hranu. Kako je u kulinarskom timu Italiano nedavno došao nov, neiskusan kuvar italijanske kuhinje odlučuju da zahtev proslede svojim kolegama iz druge agencije – kulinarskom timu Roma, kako bi im oni pripremili meni.

Primena definicije na Agenciju za ketering usluge:

Obezbeđuje posrednika ***(KulinarskiTimItaliano)*** za pristupanje drugom objektu ***(KulinarskiTimRoma)*** kako bi se omogućio kontrolisani pristup do njega.



U datom slučaju, primena Proxy uzora je realizovana preko sledećih klasa/interfejsa:

- **KeteringTim** (Subject)

Definiše zajednički interfejs za klase KulinarskiTimItaliano i KulinarskiTimRoma.

- **KulinarskiTimItaliano** (Proxy)

Klasa čuva referencu na klasu KulinarskiTimRoma. Na taj način koristi metode klase KulinarskiTimRoma.
Samo klasa KulinarskiTimItaliano zna za postojanje klase KulinarskiTimRoma pa na taj način može biti odgovorna za kreiranje njene instance, kao i za brisanje.

- **KulinarskiTimRoma** (RealSubject)

Definiše objekat koji KulinarskiTimItaliano reprezentuje. Implementira metode interfejsa KeteringTim (metode *kreirajPredjelo(), kreirajGlavnoJelo(), kreirajDezert() i vratiMeni()*), koje će KulinarskiTimItaliano pozivati u okviru svojih metoda *(), kreirajGlavnoJelo(), kreirajDezert() i vratiMeni().*

Klasa KulinarskiTimItaliano će za implementaciju svojih metoda iskoristiti implementaciju metoda klase KulinarskiTimRoma.

U datom slučaju KulinarskiTimItaliano je svoje obaveze kreiranja obroka (predjela, glavnog jela i dezerta), kao i sklapanja tih obroka u celinu, odnosno kreiranje samog menia preneo na KulinarskiTimRoma. KulinarskiTimRoma je taj koji kreira obroke – instance klasa Pica, SkapmiSaBademom i Sladoled (metode *kreirajPredjelo(), kreirajGlavnoJelo() i kreirajDezert()*) i kreira meni (metoda *vratiMeni()*). KlijentskiTimItaliano u okviru svojih metoda *kreirajPredjelo(), kreirajGlavnoJelo(), kreirajDezert() i vratiMeni()* uzima od KulinarskogTimaRoma kreirane obroke i meni.

Odgovornost se delegira sa KulinarskogTimaItaliano na KulinarskiTimRoma. Klijent ne zna za postojanje KulinarskogTimaRoma, on celu ponudu (meni) dobija od KulinarskogTimaItaliano.

Programski kod:

```
/**
 *
 * @author Ivana
 */

public class Klijent // Client {
    private KeteringTim ktim;

    public Klijent(KeteringTim ktim) {
            this.ktim = ktim;
    }
    void Konstruisi() {
            ktim.kreirajPredjelo();
            ktim.kreirajGlavnoJelo();
            ktim.kreirajDezert();
    }
}

interface KeteringTim // Subject{

    void kreirajPredjelo();

    void kreirajGlavnoJelo();

    void kreirajDezert();

    String vratiMeni();
}

public class Meni{
```

```
    private Predjelo predjelo;
    private GlavnoJelo glavnojelo;
    private Dezert dezert;

    public Dezert getDezert() {
        return dezert;
    }
    public GlavnoJelo getGlavnojelo() {
        return glavnojelo;
    }
    public Predjelo getPredjelo() {
        return predjelo;
    }
    public void setDezert(Dezert dezert) {
        this.dezert = dezert;
    }
    public void setGlavnojelo(GlavnoJelo glavnojelo) {
        this.glavnojelo = glavnojelo;
    }
    public void setPredjelo(Predjelo predjelo) {
        this.predjelo = predjelo;
    }
}

public class KulinarskiTimItaliano implements KeteringTim //Proxy {
    private KulinarskiTimRoma ktroma;

    public KulinarskiTimItaliano(KulinarskiTimRoma ktroma) {
        this.ktroma = ktroma;
    }
    public void kreirajPredjelo() {
        ktroma.kreirajPredjelo();
    }
    public void kreirajGlavnoJelo() {
        ktroma.kreirajGlavnoJelo();
    }
    public void  kreirajDezert() {
        ktroma.kreirajDezert();
    }
    public String vratiMeni() {
        return ktroma.vratiMeni();
    }
}

public class KulinarskiTimRoma implements KeteringTim // RealSubject{
    private Meni elementimeni;
```

```
    public KulinarskiTimRoma() {
        elementimeni = new Meni();
    }
    public void kreirajPredjelo() {
        elementimeni.setPredjelo(new SkampiSaBademom());
    }
    public void kreirajGlavnoJelo() {
        elementimeni.setGlavnojelo(new Pica());
    }
    public void kreirajDezert() {
        elementimeni.setDezert(new Sladoled());
    }
    public String vratiMeni() {
        return "Meni se sastoji iz: \n Predjelo:  "
                + elementimeni.getPredjelo().vratiPredjelo()
                + " \n Glavno jelo:  "
                + elementimeni.getGlavnojelo().vratiGlavnoJelo()
                + " \n Dezert: "
                + elementimeni.getDezert().vratiDezert();
     }
}

public interface Predjelo {
    String vratiPredjelo ();
}

public class SkampiSaBademom implements Predjelo {
    public String vratiPredjelo () {
        return " Skampi sa bademima u umaku ";
    }
}

public class Proja implements Predjelo {
    public String vratiPredjelo () {
        return " Proja sa spanacem";
    }
}

public interface GlavnoJelo  {
    String vratiGlavnoJelo ();
}


public class Gulas implements GlavnoJelo{
    public String vratiGlavnoJelo () {
```

```
        return "Gulas";
    }
}

public class Pica implements GlavnoJelo {
    public String vratiGlavnoJelo () {
        return " Pica ";
    }
}

public interface Dezert {
    String vratiDezert ();
}

public class Orasnica implements Dezert{
    public String vratiDezert() {
        return " Orasnica ";
    }
}

public class Sladoled implements Dezert{
    public String vratiDezert() {
        return " Sladoled ";
    }
}
```

Rezultat programa:

```
Meni se sastoji iz:
 Predjelo:   Skampi sa bademima u umaku
 Glavno jelo:  Pica
 Dezert:  Sladoled
```

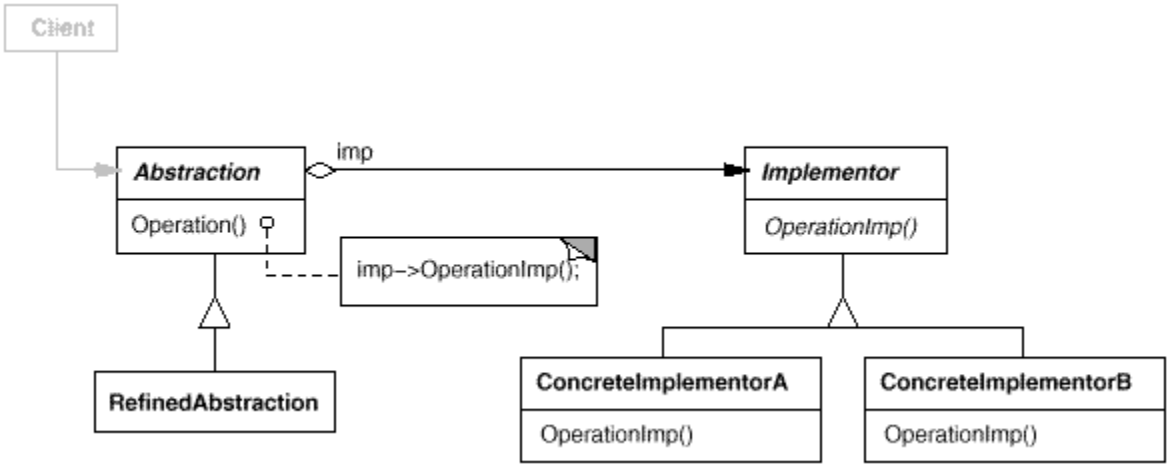
2.2. Bridge uzor

Definicija:

Odvaja (dekupluje) apstrakciju od njene implementacije tako da se one mogu menjati nezavisno.

Pojašnjena definicija:

Odvaja (dekupluje) apstrakciju ***(Abstraction)*** od njene implementacije ***(Implementor)*** tako da se one mogu menjati nezavisno.

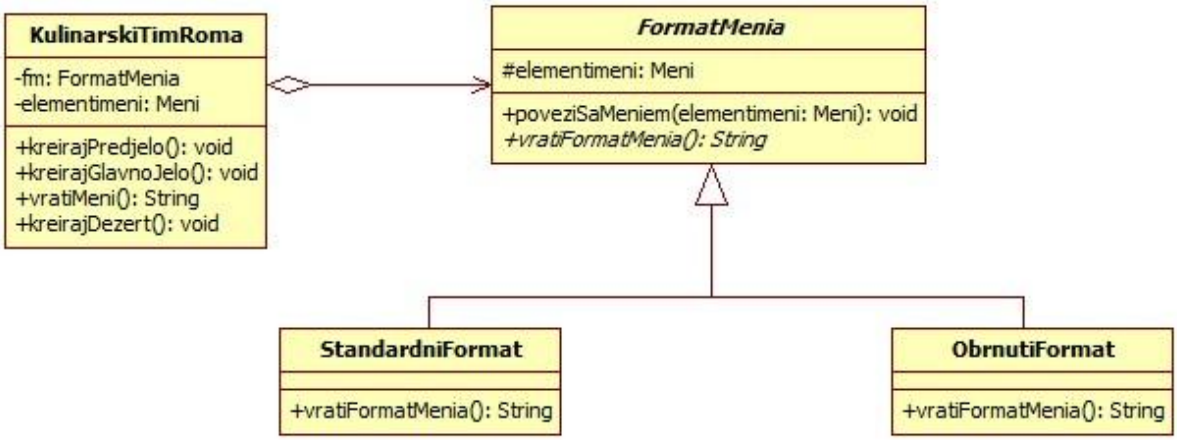


Primana Bridge uzora na primeru agencije za ketering usluge:

Klijent Julio Pitassi je saznao da mu na proslavu dolaze i saradnici iz Japana kod kojih je običaj da se na početku jede dezert a na kraju predjelo, bez obzira o kojoj je kuhinji reč. Julio Pitassi je odlučio da im udovolji i da obezbedi dva formata menija za svoju proslavu, po kojima će uslužno osoblje posluživati hranu gostima. Prvi format menija je standardan (predjelo, glavno jelo i dezert), a drugi format je dezert, glavno jelo i predjelo. Julio je postavi pred agencijom za ketering usluge ovaj zahtev (da mu se naprave dva formata menija italijanske kuhinje). Kulinarski tim Italiano je prosledio zahtev kulinarskom timu Roma.

Primena definicije na Agenciju za ketering usluge:

Odvaja (dekupluje) apstrakciju ***(KulinarskiTimRoma)*** od njene implementacije ***(FormatMenia)*** tako da se one mogu menjati nezavisno.



U datom slučaju, primena Bridge uzora je realizovana preko sledećih klasa:

- **KulinarskiTimRoma** (RefinedAbstraction)

Klasa koja čuva referencu na objekat tipa FormatMenia

- **FormatMenia** (Implementor)

Apstraktna klasa

- **StandardniFormat, ObrnutiFormat** (ConcreteImplementor)

Klase koje nasleđuju klasu FormatMenia i implementiraju njenu apstraktnu metodu *vratiFormatMenia(),* svaka klasa na svoj način.

U datom slučaju, kako je KulinarskiTimRoma taj koji kreira obroke (elemente menia) i sam meni on je taj koji ima referencu na objekat tipa FormatMeni-a (konkretno tipa StandardniFormat ili ObrnutiFormat).
KulinarskiTimRoma kreira obroke – instance klasa Pica, SkapmiSaBademom i Sladoled (metode *kreirajPredjelo(), kreirajGlavnoJelo() i kreirajDezert()*), dok se obroci sklapaju u meni u zavisnosti od toga na koji konkretan format menia (StandardniFormat ili ObrnutiFormat) KulinarskiTimRoma ima referencu. Metoda za sklapanje menia je *vratiFormatMenia()* u okviru klase FormatMenia (klase StandardniFormat i ObrnutiFormat implementiraju datu metodu svaka na svoj način), a ona se poziva u okviru metode *vratiMeni()* klase KulinarskiTimRoma.

Programski kod:

```
/**
 *
 * @author Ivana
 */
```

```
public class Klijent {
   private KeteringTim ktim;

    public Klijent(KeteringTim ktim) {
        this.ktim = ktim;
    }
     public void Konstruisi() {
        ktim.kreirajPredjelo();
        ktim.kreirajGlavnoJelo();
        ktim.kreirajDezert();
    }
}

interface KeteringTim {

    void kreirajPredjelo();
    void kreirajGlavnoJelo();
    void kreirajDezert();
    String vratiMeni();
}

public class Meni {
    private Predjelo predjelo;
    private GlavnoJelo glavnojelo;
    private Dezert dezert;

    public Dezert getDezert() {
       return dezert;
    }
    public GlavnoJelo getGlavnojelo() {
       return glavnojelo;
    }
    public Predjelo getPredjelo() {
       return predjelo;
    }
    public void setDezert(Dezert dezert) {
       this.dezert = dezert;
    }
    public void setGlavnojelo(GlavnoJelo glavnojelo) {
       this.glavnojelo = glavnojelo;
    }
    public void setPredjelo(Predjelo predjelo) {
       this.predjelo = predjelo;
    }
}
```

```
abstract class FormatMenia // Implementor{
  protected Meni elementimeni;

   public void poveziSaMeniem(Meni elementimeni) {
      this.elementimeni = elementimeni;
   }
   abstract public String vratiFormatMenia();
}

class StandardniFormat extends FormatMenia // ConcreteImplementor1{
   public String vratiFormatMenia() {
      return " Meni se sastoji iz: \n Predjelo : "
            + elementimeni.getPredjelo().vratiPredjelo()
            + " \n Glavno jelo : "
            + elementimeni.getGlavnojelo().vratiGlavnoJelo()
            + " \n Dezert : "
            + elementimeni.getDezert().vratiDezert();
   }
}

class ObrnutiFormat extends FormatMenia // ConcreteImplementor2{
   public String vratiFormatMenia() {
      return " Meni se sastoji iz: \n Dezert : "
            + elementimeni.getDezert().vratiDezert()
            + " \n Glavno jelo : "
            + elementimeni.getGlavnojelo().vratiGlavnoJelo()
            + " \n Predjelo :"
            + elementimeni.getPredjelo().vratiPredjelo();
   }
}

public class KulinarskiTimItaliano implements KeteringTim {
   private KulinarskiTimRoma ktroma;

   public KulinarskiTimItaliano(KulinarskiTimRoma ktroma) {
      this.ktroma = ktroma;
   }
   public void kreirajPredjelo() {
      ktroma.kreirajPredjelo();
   }
   public void kreirajGlavnoJelo() {
      ktroma.kreirajGlavnoJelo();
   }
   public void kreirajDezert() {
      ktroma.kreirajDezert();
   }
```

```
    public String vratiMeni() {
        return ktroma.vratiMeni();
    }
}

public class KulinarskiTimRoma implements KeteringTim // RefinedAbstraction{

    private Meni elementimeni;
    private FormatMenia fm;

    public KulinarskiTimRoma(FormatMenia fm) {
        elementimeni = new Meni();
        this.fm = fm;
        this.fm.poveziSaMeniem(elementimeni);
    }
    public void kreirajPredjelo() {
        elementimeni.setPredjelo(new SkampiSaBademom());
    }
    public void kreirajGlavnoJelo() {
        elementimeni.setGlavnojelo(new Pica());
    }
    public void kreirajDezert() {
        elementimeni.setDezert(new Sladoled());
    }
    public String vratiMeni() {
        return fm.vratiFormatMenia();
    }
}

public interface Predjelo {
    String vratiPredjelo ();
}

public class SkampiSaBademom implements Predjelo {
    public String vratiPredjelo () {
        return " Skampi sa bademima u umaku ";
    }
}

public class Proja implements Predjelo {
    public String vratiPredjelo () {
        return " Proja sa spanacem";
    }
}
```

```
public interface GlavnoJelo  {
   String vratiGlavnoJelo ();
}

public class Gulas implements GlavnoJelo{
   public String vratiGlavnoJelo () {
      return "Gulas";
   }
}

public class Pica implements GlavnoJelo {
   public String vratiGlavnoJelo () {
      return " Pica ";
   }
}

public interface Dezert {
   String vratiDezert ();
}

public class Orasnica implements Dezert{
   public String vratiDezert() {
      return " Orasnica ";
   }
}

public class Sladoled implements Dezert{
   public String vratiDezert() {
      return " Sladoled ";
   }
}

public class Main // Main class {
   public static void main(String[] args) {
      Klijent klijent;

      FormatMenia fm1 = null;
      fm1 = new StandardniFormat();

      KulinarskiTimRoma ktroma = new KulinarskiTimRoma(fm1);
      KulinarskiTimItaliano ktitaliano = new KulinarskiTimItaliano(ktroma);
      klijent = new Klijent(ktitaliano);
      klijent.Konstruisi();
      System.out.println("Standardni format menia:\n"+ktitaliano.vratiMeni());
```

```
        FormatMenia fm2 = null;
        fm2 = new ObrnutiFormat();

        KulinarskiTimRoma ktroma2 = new KulinarskiTimRoma(fm2);
        KulinarskiTimItaliano ktitaliano2 = new KulinarskiTimItaliano(ktroma2);
        klijent = new Klijent(ktitaliano2);
        klijent.Konstruisi();
        System.out.println("Obrnuti format menia za japanske goste:\n"+ktitaliano2.vratiMeni());
}
}
```

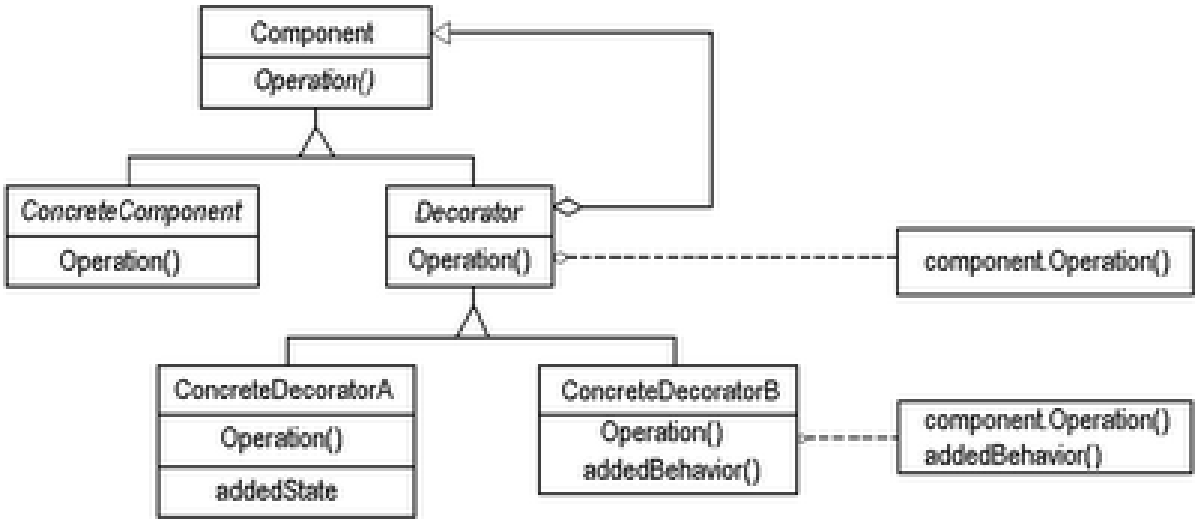
Rezultat programa:

```
Standardni format menia:
 Meni se sastoji iz:
 Predjelo :  Skampi sa bademima u umaku
 Glavno jelo :  Pica
 Dezert :Sladoled
-----------------------------------------
Obrnuti format menia za japanske goste:
 Meni se sastoji iz:
 Dezert : Sladoled
 Glavno jelo :  Pica
 Predjelo : Skampi sa bademima u umaku
```

2.3. Decorator uzor

Definicija:
Pridružuje odgovornost do objekta dinamički. Dekorator proširuje funkcionalnost objekta dinamičkim dodavanjem funkcionalnosti drugih objekata.

Pojašnjena definicija:
Pridružuje odgovornost do objekta dinamički. Dekorator proširuje funkcionalnost objekta ***(ConcreteComponent)*** dinamičkim dodavanjem funkcionalnosti drugih objekata ***(ConcreteDecoratorA, ConcreteDecoratorB)***.

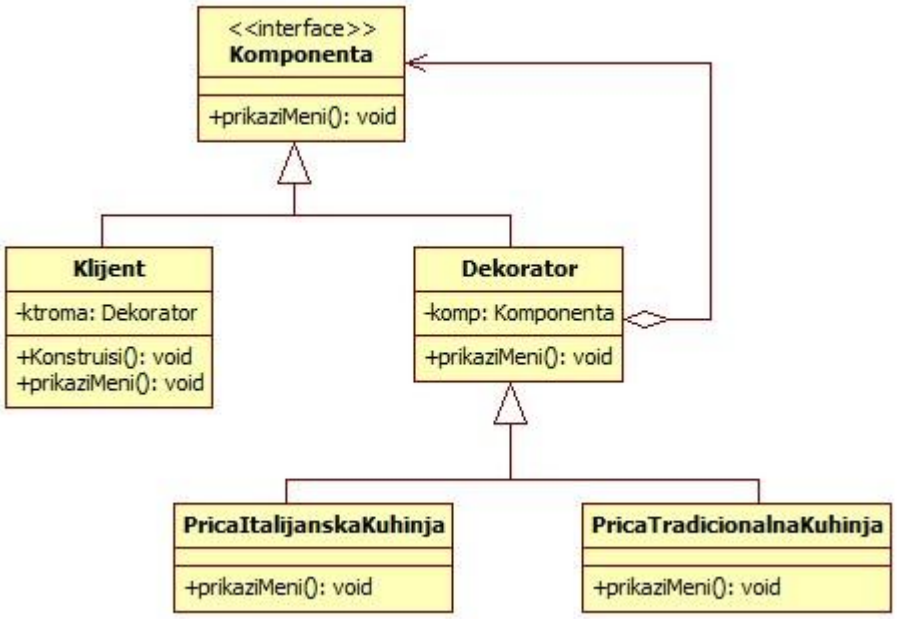


Primana Decorator uzora na primeru agencije za ketering usluge:

Klijent Julio Pitassi takođe zahteva da mu se uz meni napiše kratka priča o italijanskoj kuhinji. Agenciji za ketering službu se svideo zahtev i došli su na ideju da svojim klijentima kao uslugu nude i mogućnost pisanja kratke priče o konkretnoj kuhinji na meniju.

Primena definicije na Agenciju za ketering usluge:

Pridružuje odgovornost do objekta dinamički. Dekorator proširuje funkcionalnost objekta ***(Klijent)*** dinamičkim dodavanjem funkcionalnosti drugih objekata ***(PricaItalijanskaKuhinja, PricaTradicionalnaKuhinja)***.



U datom slučaju, primena Decorator uzora je realizovana preko sledećih klasa/interfejsa:

- **Komponenta** (Component)

Definiše interfejs za objekte kojima se odgovornost dodaje dinamički. Klase Klijent i Dekorator implementiraju interfejs Komponenta (implementiraju metodu *prikaziMeni()* svaka na svoj način).

- **Klijent** (ConcreteComponent)

Klasa koja definiše konkretan objekat kome će biti dodata odgovornost dinamički

- **Dekorator** (Decorator)

Klasa koja ima referencu na Komponentu i u okviru svoje metode *prikaziMeni()* poziva metodu *prikaziMeni()* komponente čiju referencu čuva. Na taj način se nove funkcionalnosti dodaju dinamički.

- **PricaItalijanskaKuhinja, PricaTradicionalnaKuhinja** (ConcreteDecorator)

Klase koje nasleđuju klasu Dekorator, i 'nadograđuju' implementaciju metode *prikažiMeni()* sa svojom dodatnom funkcionalnošću.

Tako, u datom slučaju, imaćemo objekat tipa Klijent koji ima svoju implementaciju metode *prikaziMeni()*.
Pozivom te metode dobićemo određenu strukturu menia, ali sada hoćemo toj strukturi da dodamo dodatnu funkcionalnost (na primer priču o italijanskoj kuhinji).

Objekat tipa PricaItalijanskaKuhinja implementira metodu prikaziMeni() tako sto prvo izvrši metodu *prikaziMeni()* komponente na koju ima referencu, čime se dobija određena struktura menia, a zatim na dobijenu strukturu menia dodaje još jedan deo (priču o italijanskoj kuhinji).

Dakle, kako je klasa Klijent komponenta (*Klijent implements Komponenta*) logično je da će objekat tipa PricaItalijanskaKuhinja imati referencu na objekat tipa Klijent.
Kako bi se dobila 'potpuna, nadograđena' struktura menia dati objekat tipa PricaItalijanskaKuhinja poziva metodu *prikaziMeni()*:

PricaItalijanskaKuhinja itprica = new PricaItalijanskaKuhinja(klijent);
itprica.prikaziMeni();

Programski kod:

```
/**
 *
 * @author Ivana
 */

interface Komponenta  // Component {
    public void prikaziMeni();
}

class Dekorator implements Komponenta{  // Decorator
    Komponenta komp;

    public Dekorator(Komponenta komp1) {
        komp = komp1;
    }

    public void prikaziMeni() {
        komp.prikaziMeni();
    }
}

class PricaItalijanskaKuhinja extends Dekorator{ // ConcreteDecorator1
    public PricaItalijanskaKuhinja(Komponenta komp1) {
        super(komp1);
    }

    public void prikaziMeni() {
        super.prikaziMeni();
        System.out.println("\nItalijanska kuhinja jeste jedna od najpopularnijih, ali i jedna od najpoznatijih kuhinja u svetu.\nItalijani vole da prilikom kuvanja koriste sezonsko voće i povrće pa se svaki ovaj segment njihove kuhinje znatno menja sa promenom godišnjeg doba.  ");
    }
}
```

```
class PricaTradicionalnaKuhinja extends Dekorator { // ConcreteDecorator2
   public PricaTradicionalnaKuhinja(Komponenta komp1) {
      super(komp1);
   }

   public void prikaziMeni() {
      super.prikaziMeni();
      System.out.println("*\n Tradicionalna domaca kuhinja je.....");
   }
}

public class Klijent  implements Komponenta //  ConcreteComponent {
   private KeteringTim ktim;

   public Klijent(KeteringTim ktim) {
      this.ktim = ktim;
   }
   public void Konstruisi() {
      ktim.kreirajPredjelo();
      ktim.kreirajGlavnoJelo();
      ktim.kreirajDezert();
   }
   public void prikaziMeni() {
      System.out.println(ktim.vratiMeni());
   }
}

interface KeteringTim {

   void kreirajPredjelo();

   void kreirajGlavnoJelo();

   void kreirajDezert();

   String vratiMeni();
}

public class Meni{
   private Predjelo predjelo;
   private GlavnoJelo glavnojelo;
   private Dezert dezert;

   public Dezert getDezert() {
      return dezert;
   }
```

```
    public GlavnoJelo getGlavnojelo() {
        return glavnojelo;
    }
    public Predjelo getPredjelo() {
        return predjelo;
    }
    public void setDezert(Dezert dezert) {
        this.dezert = dezert;
    }
    public void setGlavnojelo(GlavnoJelo glavnojelo) {
        this.glavnojelo = glavnojelo;
    }
    public void setPredjelo(Predjelo predjelo) {
        this.predjelo = predjelo;
    }
}

abstract class FormatMenia {
    Meni elementimeni;

    public void poveziSaMeniem(Meni elementimeni) {
        this.elementimeni = elementimeni;
    }
    abstract public String vratiFormatMenia();
}

class StandardniFormat extends FormatMenia {
    public String vratiFormatMenia() {
        return " Meni se sastoji iz: \n Predjelo - "
                + elementimeni.getPredjelo().vratiPredjelo()
                + " \n Glavno jelo - "
                + elementimeni.getGlavnojelo().vratiGlavnoJelo()
                + " \n Dezert -"
                + elementimeni.getDezert().vratiDezert();  }
}

class ObrnutiFormat extends FormatMenia {
     public String vratiFormatMenia() {
        return " Meni se sastoji iz: \n Predjelo - "
                + elementimeni.getDezert().vratiDezert()
                + " \n Glavno jelo - "
                + elementimeni.getGlavnojelo().vratiGlavnoJelo()
                + " \n Dezert -"
                + elementimeni.getPredjelo().vratiPredjelo();  }
}
```

```
public class KulinarskiTimItaliano implements KeteringTim {
   KulinarskiTimRoma ktroma;

   public KulinarskiTimItaliano(KulinarskiTimRoma ktroma) {
      this.ktroma = ktroma;
   }
   public void kreirajPredjelo() {
      ktroma.kreirajPredjelo();
   }
   public void kreirajGlavnoJelo() {
      ktroma.kreirajGlavnoJelo();
   }
   public void kreirajDezert() {
      ktroma.kreirajDezert();
   }
   public String vratiMeni() {
      return ktroma.vratiMeni();
   }
}

public class KulinarskiTimRoma implements KeteringTim {
   Meni elementimeni;
   FormatMenia fm;

   public KulinarskiTimRoma(FormatMenia fm) {
      elementimeni = new Meni();
      this.fm = fm;
      this.fm.poveziSaMeniem(elementimeni);
   }
   public void kreirajPredjelo() {
      elementimeni.setPredjelo(new SkampiSaBademom());
   }
   public void kreirajGlavnoJelo() {
      elementimeni.setGlavnojelo(new Pica());
   }
   public void kreirajDezert() {
      elementimeni.setDezert(new Sladoled());
   }
   public String vratiMeni() {
      return fm.vratiFormatMenia();
   }
}

public interface Predjelo {
   String vratiPredjelo ();
}
```

```
public class SkampiSaBademom implements Predjelo {
   public String vratiPredjelo () {
      return " Skampi sa bademima u umaku ";
   }
}

public class Proja implements Predjelo {
   public String vratiPredjelo () {
      return " Proja sa spanacem";
   }
}

public interface GlavnoJelo  {
   String vratiGlavnoJelo ();
}

public class Gulas implements GlavnoJelo{
   public String vratiGlavnoJelo () {
      return "Gulas";
   }
}

public class Pica implements GlavnoJelo {
   public String vratiGlavnoJelo () {
      return " Pica ";
   }
}

public interface Dezert {
   String vratiDezert ();
}

public class Orasnica implements Dezert{
   public String vratiDezert() {
      return " Orasnica ";
   }
}

public class Sladoled implements Dezert{
   public String vratiDezert() {
      return " Sladoled ";
   }
}
```

```
public class Main  // Main class {
   public static void main(String[] args) {
      Klijent klijent;

      //kreiranje manija italijanske kuhinje standardnog formata
      FormatMenia fm1 = null;
      fm1 = new StandardniFormat();
      KulinarskiTimRoma ktroma = new KulinarskiTimRoma(fm1);
      KulinarskiTimItaliano ktitaliano = new KulinarskiTimItaliano(ktroma);
      klijent = new Klijent(ktitaliano);
      klijent.Konstruisi();
      PricaItalijanskaKuhinja itprica = new PricaItalijanskaKuhinja(klijent);
      itprica.prikaziMeni();

      System.out.println("\n-------------------------------------------------------------------\n");
      //kreiranje manija italijanske kuhinje obrnutog formata za Japance
      FormatMenia fm2 = null;
      fm2 = new ObrnutiFormat();
      KulinarskiTimRoma ktroma2 = new KulinarskiTimRoma(fm2);
      KulinarskiTimItaliano ktitaliano2 = new KulinarskiTimItaliano(ktroma2);
      klijent = new Klijent(ktitaliano2);
      klijent.Konstruisi();
      PricaItalijanskaKuhinja itprica2 = new PricaItalijanskaKuhinja(klijent);
      itprica2.prikaziMeni();
   }
}
```

Rezultat programa:

```
Meni se sastoji iz:
 Predjelo -  Skampi sa bademima u umaku
 Glavno jelo -  Pica
 Dezert -Sladoled

Italijanska kuhinja jeste jedna od najpopularnijih, ali i jedna od najpoznatijih kuhinja u svetu.
Italijani vole da prilikom kuvanja koriste sezonsko voće i povrće pa se svaki ovaj segment
njihove kuhinje znatno menja sa promenom godišnjeg doba.
-------------------------------------------------------------------------------------------------------
 Meni se sastoji iz:
 Dezert - Sladoled
 Glavno jelo -  Pica
 Predjelo - Skampi sa bademima u umaku

Italijanska kuhinja jeste jedna od najpopularnijih, ali i jedna od najpoznatijih kuhinja u svetu.
Italijani vole da prilikom kuvanja koriste sezonsko voće i povrće pa se svaki ovaj segment
njihove kuhinje znatno menja sa promenom godišnjeg doba.
```

3. Paterni ponašanja

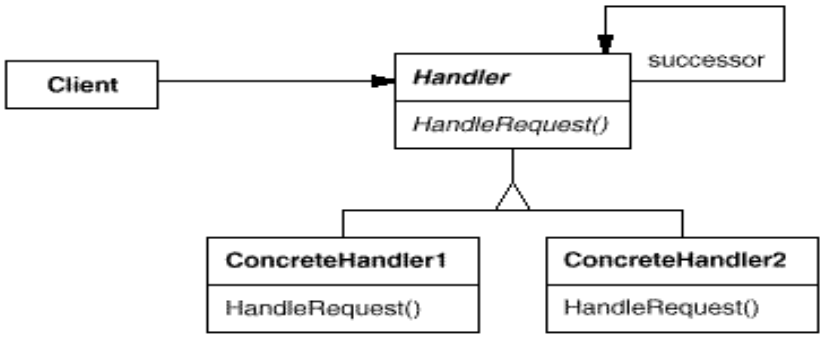
3.1. Chain of responsibility uzor

Definicija:

Izbegava čvrsto povezivanje pošiljaoca zahteva sa njegovim primaocem dajući šansu da više objekata rukuje zahtevom. Povezuje objekte primaoce i prosleđuje zahtev niz lanac dok ga neki objekat ne obradi.

Pojašnjena definicija:

Izbegava čvrsto povezivanje pošiljaoca zahteva ***(Client)*** sa njegovim primaocem ***(Handler)*** dajući šansu da više objekata rukuje zahtevom. Povezuje objekte primaoce i prosleđuje zahtev niz lanac povezanih objekata ***(ConcreteHandler1, ConcreteHandler2)*** dok ga neki objekat ne obradi.



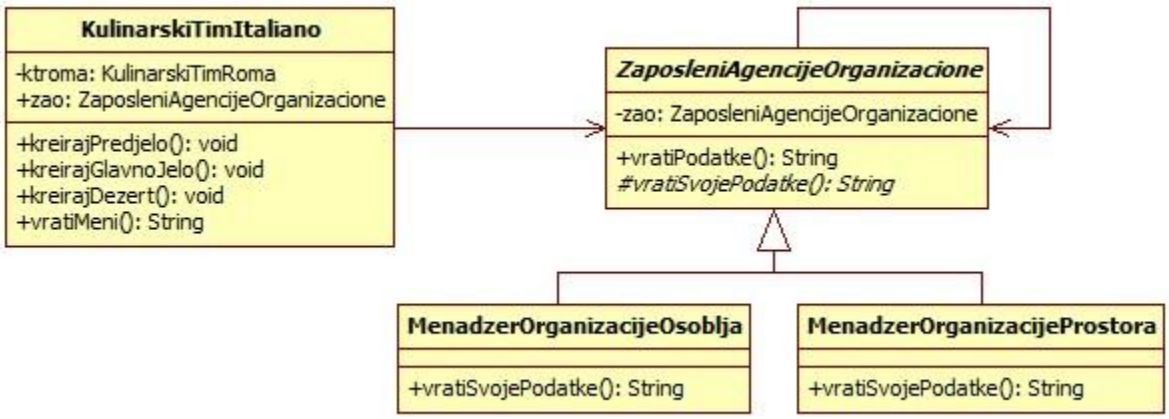
Chain of responsibility treba upotrebiti kada više objekata može da obradi zahtev, ali se unapred ne zna koji će ga obraditi. Zahtev se kreće u hijerarhiji objekata sve dok neki objekat ne uzme odgovornost da ga obradi. Na ovaj način se uprošćava međusobno povezivanje objekata jer oni imaju reference samo na svog sledbenika. Pošto on nema eksplicitnog primaoca, ne postoji garancija da će on biti obrađen, može se desiti da dođe do kraja lanca, a da ga niko ne obradi.

Primana Chain of responsability uzora na primeru agencije za ketering usluge:

Klijent Julio Pitassi se interesuje preko koga i po kojoj ceni moze da iznajmi prostor za održavanje proslave. Takođe se interesuje i preko koga i za koju cenu može da iznajmi osoblje italijanskog porekla koje će posluživati goste na proslavi. Kulinarski tim „Italiano“ sarađuje sa italijanskom agencijom „Organizacione“ koja se bavi izdavanjem prostora za proslave, organizovanjem osoblja, i ostalim poslovima vezanim za organizaciju proslava. Klijentu je ponuđeno da mu oni nabave kontakt telefone i ostale osnovne podatke osoba kojima može da se javi u vezi sa svojim zahtevima. Gospodin Julio prihvata ponudu pa se sada od menadžera agencije „Organizacione“, konkretno menadžera sektora za organizaciju prostora i menadžera sektora za organizaciju uslužnog osoblja, zahteva da im pošalju kontakt telefon i okvirnu cenu svojih usluga.

Primena definicije na Agenciju za ketering usluge:

Izbegava čvrsto povezivanje pošiljaoca zahteva ***(KulinarskiTimItaliano)*** sa njegovim primaocem ***(ZaposleniAgencijeOrganizacione)*** dajući šansu da više objekata rukuje zahtevom. Povezuje objekte primaoce i prosleđuje zahtev niz lanac povezanih objekata ***(MenadzerOrganizacijeProstora, MenadzerOrganizacijeOsoblja)*** dok ga neki objekat ne obradi.



U datom slučaju, primena Chain of responsability uzora je realizovana preko sledećih klasa:

- **ZaposleniAgencijeOrganizacione** (Handler)

Definiše interfejs za obradu zahteva.

- **MenadzerOrganizacijeProstora, MenadzerOrganizacijeOsoblja** (ConcreteHandler)

Obrađuje zahtev za koji je odgovoran.
Može pristupiti svom sledbeniku.
Ukoliko može da obradi zahtev on ga obrađuje, inače ga prosleđuje do svog sledbenika.

- **KulinarskiTimItaliano** (Client)

Inicira zahtev koji treba da se obradi.

Dakle, u datom slučaju, uvodi se apstraktna klasa ZaposleniAgencijeOrganizacione i klase: MenadzerOrganizacijeProstora i MenadzerOrganizacijeOsoblja, koje je nasleđuju, a predstavljaju konkretna pojavljivanja zaposlenih agencije "Organizacione" (konkretna pojavljivanja menadžera). Obe te klase implementiraju apstraktnu metodu *vratiSvojePodatke()* koja vraća podatke konkretnog menadžera (zaposlenog).
Klasa ZaposleniAgencijeOrganizacione sadrži referencu sama na sebe. Odnosno, postoji lanac objekata određenog tipa menadžera i svaki taj objekat ima pokazivač na sledeći objekat u nizu, a to je upravo pomenuta referenca. Preko metode *vratiPodatke()* , klase ZaposleniAgencijeOrganizacione, podaci se kreću kroz niz objekata tipa ZaposleniAgencijeOrganizacione i dopunjuju kod svakog objekta njegovim podacima. Na kraju niza je objekat čija će metoda *vratiPodatke()* vratiti njegove podatke + podatke svih prethodnih objekata niza.
Klasa KulinarskiTimItaliano čuva referencu na klasu ZaposleniAgencijeOrganizacione (konkretno na poslednji objekat, tipa ZaposleniAgencijeOrganizacione, formiranog 'niza'), i ona dopunjuje svoju ponudu menia sa podacima koje uzima preko metode *vratiPodatke*(), koju poziva preko objekta tipa ZaposleniAgencijeOrganizacione na koji ima referencu (*zao*):

```
public String vratiMeni() {
      return
            zao.vratiPodatke() + "\n" + ktroma.vratiMeni();
}
```

Programski kod:

```
/**
 *
 * @author Ivana
 */

interface Komponenta  {
          void prikaziMeni();
}

class Dekorator implements Komponenta{
  Komponenta komp;

   public Dekorator(Komponenta komp1) {
      komp = komp1;
   }
   public void prikaziMeni() {
      komp.prikaziMeni();
   }
}
```

```
class PricaItalijanskaKuhinja extends Dekorator {
   public PricaItalijanskaKuhinja(Komponenta komp1) {
      super(komp1);
   }
   public void prikaziMeni() {
      super.prikaziMeni();
      System.out.println("\nItalijanska kuhinja jeste jedna od najpopularnijih, ali i jedna od najpoznatijih kuhinja u svetu.\nItalijani vole da prilikom kuvanja koriste sezonsko voće i povrće pa se svaki ovaj segment njihove kuhinje znatno menja sa promenom godišnjeg doba.  ");
   }
}

class PricaTradicionalnaKuhinja extends Dekorator{
   public PricaTradicionalnaKuhinja(Komponenta komp1) {
      super(komp1);
   }
   public void prikaziMeni() {
      super.prikaziMeni();
      System.out.println("*\n Tradicionalna domaca kuhinja je najbolja !");
   }
}

public abstract class ZaposleniAgencijeOrganizacione { // Handler
   private ZaposleniAgencijeOrganizacione zao;

   public ZaposleniAgencijeOrganizacione(ZaposleniAgencijeOrganizacione zao) {
      this.zao = zao;
   }
   public String vratiPodatke() {
      String podaci = "";
      podaci = "\n" + vratiSvojePodatke();
      if (this.zao != null) {
         podaci = zao.vratiPodatke() + podaci;
      }
      return podaci;
   }
   protected abstract String vratiSvojePodatke();
}

public class MenadzerOrganizacijeProstora extends ZaposleniAgencijeOrganizacione {
// ConcreteHandler1

   public MenadzerOrganizacijeProstora(ZaposleniAgencijeOrganizacione zao1) {
      super(zao1);
   }
```

```
    protected String vratiSvojePodatke() {
        return " Menadzer sektora za organizaciju prostora: Milan Rakic\nKontakt telefon: 064/456
78 88\nOkvirna cena: 20.000,00 dinara";
    }
}

public class MenadzerOrganizacijeOsoblja extends ZaposleniAgencijeOrganizacione{
// ConcreteHandler2
    public MenadzerOrganizacijeOsoblja(ZaposleniAgencijeOrganizacione zao1) {
        super(zao1);
    }
    protected String vratiSvojePodatke() {
        return " Menadzer sektora za organizaciju usluznog osoblja: Marina Boljanac\nKontakt
telefon: 064/456 78 77\nOkvirna cena: 18.000,00 dinara";
    }
}

public class KulinarskiTimItaliano implements KeteringTim // Client{
    private KulinarskiTimRoma ktroma;
    private ZaposleniAgencijeOrganizacione zao;

    public KulinarskiTimItaliano(KulinarskiTimRoma ktroma, ZaposleniAgencijeOrganizacija
zao) {
        this.ktroma = ktroma;
        this.zao = zao;
    }
    public void kreirajPredjelo() {
        ktroma.kreirajPredjelo();
    }
    public void kreirajGlavnoJelo() {
        ktroma.kreirajGlavnoJelo();
    }
    public void kreirajDezert() {
        ktroma.kreirajDezert();
    }
    public String vratiMeni() {
        return
                zao.vratiPodatke() + "\n\n" + ktroma.vratiMeni();
    }
}

public class Klijent  implements Komponenta {

 KeteringTim ktim;
```

```
    public Klijent(KeteringTim ktim) {
        this.ktim = ktim;
    }

    void Konstruisi() {
        ktim.kreirajPredjelo();
        ktim.kreirajGlavnoJelo();
        ktim.kreirajDezert();
    }
    public void prikaziMeni() {
        System.out.println(ktim.vratiMeni());
    }
}

interface KeteringTim {

    void kreirajPredjelo();

    void kreirajGlavnoJelo();

    void kreirajDezert();

    String vratiMeni();
}

public class Meni{
    private Predjelo predjelo;
    private GlavnoJelo glavnojelo;
    private Dezert dezert;

    public Dezert getDezert() {
        return dezert;
    }
    public GlavnoJelo getGlavnojelo() {
        return glavnojelo;
    }
    public Predjelo getPredjelo() {
        return predjelo;
    }
    public void setDezert(Dezert dezert) {
        this.dezert = dezert;
    }
    public void setGlavnojelo(GlavnoJelo glavnojelo) {
        this.glavnojelo = glavnojelo;
    }
```

```
    public void setPredjelo(Predjelo predjelo) {
        this.predjelo = predjelo;
    }
}

abstract class FormatMenia {
    Meni elementimeni;

    public void poveziSaMeniem(Meni elementimeni) {
        this.elementimeni = elementimeni;
    }
    abstract public String vratiFormatMenia();
}

class StandardniFormat extends FormatMenia {
    public String vratiFormatMenia() {
        return " Meni se sastoji iz: \n Predjelo : "
                + elementimeni.getPredjelo().vratiPredjelo()
                + " \n Glavno jelo : "
                + elementimeni.getGlavnojelo().vratiGlavnoJelo()
                + " \n Dezert : "
                + elementimeni.getDezert().vratiDezert();
    }
}

class ObrnutiFormat extends FormatMenia {
    public String vratiFormatMenia() {
        return " Meni se sastoji iz: \n Dezert : "
                + elementimeni.getDezert().vratiDezert()
                + " \n Glavno jelo : "
                + elementimeni.getGlavnojelo().vratiGlavnoJelo()
                + " \n Predjelo : "
                + elementimeni.getPredjelo().vratiPredjelo();
    }
}

public class KulinarskiTimRoma implements KeteringTim {
    private Meni elementimeni;
    private FormatMenia fm;

    public KulinarskiTimRoma(FormatMenia fm) {
        elementimeni = new Meni();
        this.fm = fm;
        this.fm.poveziSaMeniem(elementimeni);
    }
```

```
    public void kreirajPredjelo() {
        elementimeni.setPredjelo(new SkampiSaBademom());
    }
    public void kreirajGlavnoJelo() {
        elementimeni.setGlavnojelo(new Pica());
    }
    public void kreirajDezert() {
        elementimeni.setDezert(new Sladoled());
    }
    public String vratiMeni() {
        return fm.vratiFormatMenia();
    }
}

public interface Predjelo {
    String vratiPredjelo ();
}

public class SkampiSaBademom implements Predjelo {
    public String vratiPredjelo () {
        return " Skampi sa bademima u umaku ";
    }
}

public class Proja implements Predjelo {
    public String vratiPredjelo () {
        return " Proja sa spanacem";
    }
}

public interface GlavnoJelo  {
    String vratiGlavnoJelo ();
}

public class Gulas implements GlavnoJelo{
    public String vratiGlavnoJelo () {
        return "Gulas";
    }
}

public class Pica implements GlavnoJelo {
    public String vratiGlavnoJelo () {
        return " Pica ";
    }
}
```

```
public interface Dezert {
   String vratiDezert ();
}

public class Orasnica implements Dezert{
   public String vratiDezert() {
      return " Orasnica ";
   }
}

public class Sladoled implements Dezert{
   public String vratiDezert() {
      return " Sladoled ";
   }
}

public class Main  // Main class  {
   public static void main(String[] args) {
      Klijent klijent;

      MenadzerOrganizacijeProstora mop = new MenadzerOrganizacijeProstora(null);
      MenadzerOrganizacijeOsoblja moo = new MenadzerOrganizacijeOsoblja(mop);

      //kreiranje manija italijanske kuhinje standardnog formata
      FormatMenia fm1 = null;
      fm1 = new StandardniFormat();
      KulinarskiTimRoma ktroma = new KulinarskiTimRoma(fm1);
      KulinarskiTimItaliano ktitaliano = new KulinarskiTimItaliano(ktroma, moo);
      klijent = new Klijent(ktitaliano);
      klijent.Konstruisi();
      PricaItalijanskaKuhinja itprica = new PricaItalijanskaKuhinja(klijent);
      itprica.prikaziMeni();

      System.out.println("\n-------------------------------------------------------------------\n");

      //kreiranje manija italijanske kuhinje obrnutog formata za Japance
      FormatMenia fm2 = null;
      fm2 = new ObrnutiFormat();
      KulinarskiTimRoma ktroma2 = new KulinarskiTimRoma(fm2);
      KulinarskiTimItaliano ktitaliano2 = new KulinarskiTimItaliano(ktroma2, mop);
      klijent = new Klijent(ktitaliano2);
      klijent.Konstruisi();
      PricaItalijanskaKuhinja itprica2 = new PricaItalijanskaKuhinja(klijent);
      itprica2.prikaziMeni();
   }
}
```

Rezultat programa:

****
Menadzer sektora za organizaciju prostora: Milan Rakic
Kontakt telefon: 064/456 78 88
Okvirna cena: 20.000,00 dinara
Menadzer sektora za organizaciju usluznog osoblja: Marina Boljanac
Kontakt telefon: 064/456 78 77
Okvirna cena: 18.000,00 dinara
****

Meni se sastoji iz:
Predjelo - Skampi sa bademima u umaku
Glavno jelo - Pica
Dezert -Sladoled

Italijanska kuhinja jeste jedna od najpopularnijih, ali i jedna od najpoznatijih kuhinja u svetu. Italijani vole da prilikom kuvanja koriste sezonsko voće i povrće pa se svaki ovaj segment njihove kuhinje znatno menja sa promenom godišnjeg doba.

Meni se sastoji iz:
Dezert - Sladoled
Glavno jelo - Pica
Predjelo - Skampi sa bademima u umaku

Italijanska kuhinja jeste jedna od najpopularnijih, ali i jedna od najpoznatijih kuhinja u svetu. Italijani vole da prilikom kuvanja koriste sezonsko voće i povrće pa se svaki ovaj segment njihove kuhinje znatno menja sa promenom godišnjeg doba.

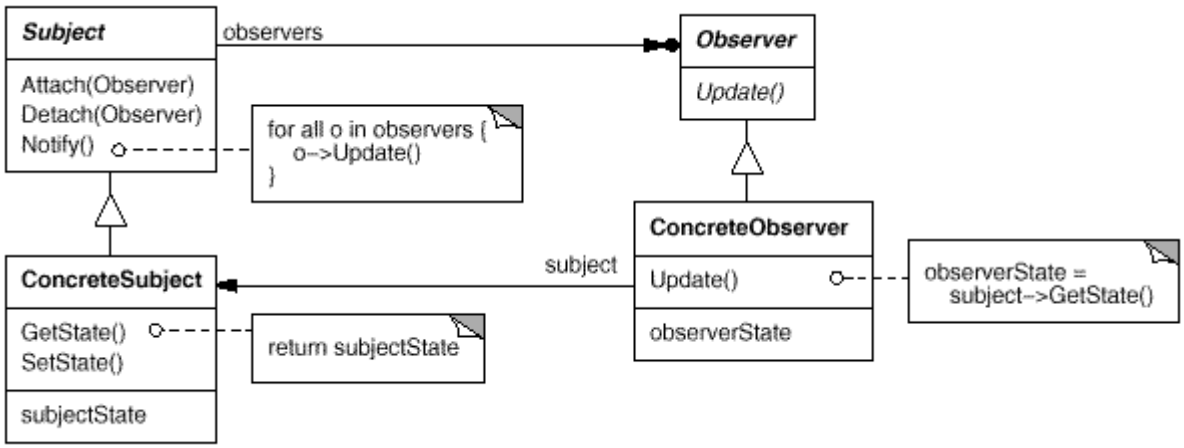
3.2. Observer uzor

Definicija:

Definiše jedan - više zavisnost između objekata , tako da promena stanja nekog objekta utiče automatski na promenu stanja svih drugih objekata koji su povezani sa njim.

Pojašnjena definicija:

Definiše jedan - više zavisnost između objekata ***(Subject -> Observer)*** tako da promena stanja nekog objekta ***(ConcreteSubject)*** utiče automatski na promenu stanja svih drugih objekata koji su povezani sa njim ***(ConcreteObserver)***.



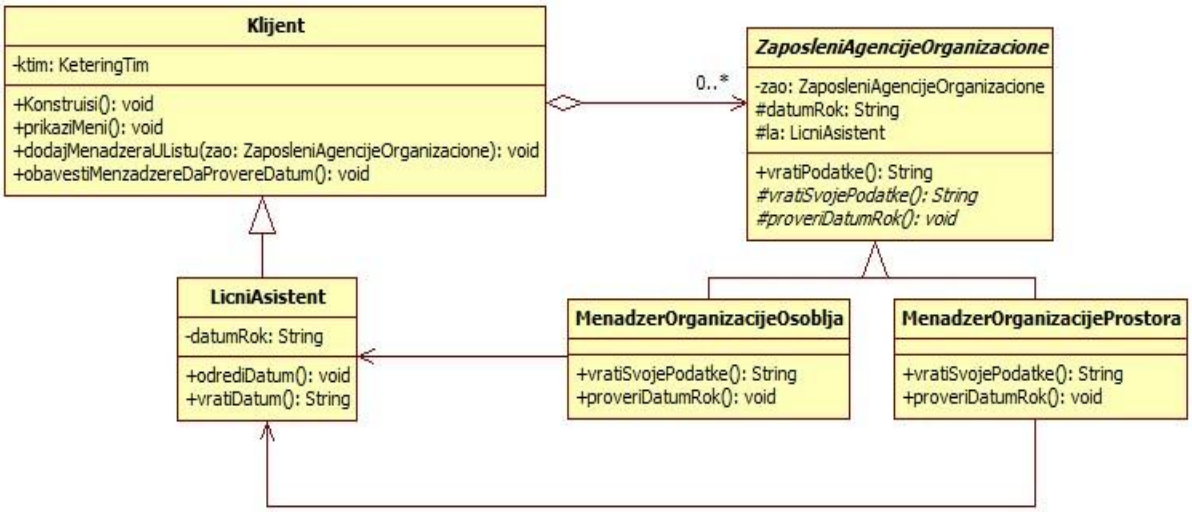
Observer obezbeđuje interface za promenu objekta na novo stanje u koje je prešao Subject objekat.Subject zna ko su njegovi observeri, a ConcreteSubject čuva stanje na koje se postavljaju observeri.ConcreteObserver čuva referencu na ConcreteSubject.

Primana Observer uzora na primeru agencije za ketering usluge:

Klijent Julio Pitassi je zadužio svog ličnog asistenta da odredi vreme do kada organizacija prostorije za proslavu i uslužnog osoblja treba da bude gotova.
Julio Pitassi obaveštava sve angažovane menadžere agencije "Organizacione" da se obrate njegovom ličnom asistentu kako bi videli do kog datuma treba da obave svoje zadatke. Svaki menadžer treba da zapamti taj datum (rok).

Primena definicije na Agenciju za ketering usluge:

Definiše jedan - više zavisnost između objekata ***(Klijent -> ZaposleniAgencijeOrganizacione)*** tako da promena stanja nekog objekta ***(LicniAsistent)*** utiče automatski na promenu stanja svih drugih objekata koji su povezani sa njim ***(MenadzerOrganizacijeOsoblja, MenadzerOrganizacijeProstora)***.



U datom slučaju, uvodimo klasu **LicniAsistent** koja preko svoje metode *odrediDatum()* postavlja vrednost svoje promenljive *datumRok* . Lični asistent je taj koji određuje datum do kada menadžeri treba da obave posao.
Klasa **Klijent** zna za postojanje svakog menadžera koga je angažovao (sadrži reference na klasu ZaposkleniAgencijeOrganizacione, konkretno na klase MenadzerOrganizacijeProstora i MenadzerOrganizacijeOsoblja).
Klijent ne obaveštava menadžere koji je to datum do kada treba da obave svoje zadatke (klasa Klijent ne zna vrednost promenljive *datumRok* niti je zanima njena vrednost), već klijent samo upućuje menadžere da saznaju taj datum od ličnog asistenta. Odnosno, preko metode *obavestiMenzadzereDaProvereDatum()* klase Klijent obaveštavaju se menadžeri da provere datum.
Klasa **ZaposkleniAgencijeOrganizacione** čuva referencu na klasu LicniAsistent, što znači da svaki menadžer (klase: MenadzerOrganizacijeProstora i MenadzerOrganizacijeOsoblja koje su konkretna pojavljivanja klase ZaposkleniAgencijeOrganizacione) znaju za postojanje ličnog asistenta.
Svaki menadžer saznaje datum putem metode *proveriDatumRok(),* klase ZaposkleniAgencijeOrganizacione, u okviru koje se uzima vrednost promenljive *datumRok* klase LicniAsistent i zatim se postavlja za vrednost promenljive datumRok klase ZaposkleniAgencijeOrganizacione:

```
public void proveriDatumRok() {
        datumRok = la.vratiDatum();
 }
```

la – objektna promenljiva tipa LicniAsistent

Programski kod:

```
/**
 *
 * @author Ivana
 */

interface Komponenta  {
        void prikaziMeni();
}

class Dekorator implements Komponenta{
  Komponenta komp;

   public Dekorator(Komponenta komp1) {
      komp = komp1;
   }
   public void prikaziMeni() {
      komp.prikaziMeni();
   }
}

class PricaItalijanskaKuhinja extends Dekorator{
   public PricaItalijanskaKuhinja(Komponenta komp1) {
      super(komp1);
   }
   public void prikaziMeni() {
      super.prikaziMeni();
      System.out.println("\nItalijanska kuhinja jeste jedna od najpopularnijih, ali i jedna od najpoznatijih kuhinja u svetu.\nItalijani vole da prilikom kuvanja koriste sezonsko voće i povrće pa se svaki ovaj segment njihove kuhinje znatno menja sa promenom godišnjeg doba.  ");
   }
}

class PricaTradicionalnaKuhinja extends Dekorator{
   public PricaTradicionalnaKuhinja(Komponenta komp1) {
      super(komp1);
   }
   public void prikaziMeni() {
      super.prikaziMeni();
      System.out.println("*\n Tradicionalna domaca kuhinja je najbolja ! ");
   }
}
```

```
public abstract class ZaposleniAgencijeOrganizacione { // Observer
   protected ZaposleniAgencijeOrganizacione zao;
   protected LicniAsistent la;
   protected String datumRok;

   public ZaposleniAgencijeOrganizacione(ZaposleniAgencijeOrganizacione zao) {
      this.zao = zao;
   }
   public String vratiPodatke() {
      String podaci = "";
      podaci = "\n" + vratiSvojePodatke();
      if (this.zao != null) {
         podaci = zao.vratiPodatke() + podaci;
      }
      return podaci;
   }
   protected abstract String vratiSvojePodatke();
   public abstract void proveriDatumRok();

   public void setLa(LicniAsistent la) {
      this.la = la;
   }
   public String getDatumRok() {
      return datumRok;
   }
}

public class MenadzerOrganizacijeProstora extends ZaposleniAgencijeOrganizacione {
// ConcreteObserver1

   public MenadzerOrganizacijeProstora(ZaposleniAgencijeOrganizacione zao1) {
      super(zao1);
   }
   String vratiSvojePodatke() {
      return " Menadzer sektora za organizaciju prostora: Milan Rakic\nKontakt telefon: 064/456
78 88\nOkvirna cena: 20.000,00 dinara";
   }
   public void proveriDatumRok() {
       datumRok = la.vratiDatum();
   }
}

public class MenadzerOrganizacijeOsoblja extends ZaposleniAgencijeOrganizacione{
// ConcreteObserver2
   public MenadzerOrganizacijeOsoblja(ZaposleniAgencijeOrganizacione zao1) {
      super(zao1);
```

```
    }
    String vratiSvojePodatke() {
        return " Menadzer sektora za organizaciju usluznog osoblja: Marina Boljanac\nKontakt telefon: 064/456 78 77\nOkvirna cena: 18.000,00 dinara";
    }
    public void proveriDatumRok() {
         datumRok = la.vratiDatum();
    }
}

public class Klijent  implements Komponenta // Subject {
    private KeteringTim ktim;
    private List <ZaposleniAgencijeOrganizacione> listaMenadzera;

    public Klijent(KeteringTim ktim) {
        this.ktim = ktim;
        listaMenadzera = new ArrayList<ZaposleniAgencijeOrganizacione>();
    }
    public void Konstruisi() {
        ktim.kreirajPredjelo();
        ktim.kreirajGlavnoJelo();
        ktim.kreirajDezert();
    }
    public void prikaziMeni() {
        System.out.println(ktim.vratiMeni());
    }
    public void dodajMenadzeraUListu(ZaposleniAgencijeOrganizacione zao){
        listaMenadzera.add(zao);
    }
    public void obavestiMenzadzereDaProvereDatum(){
        for (ZaposleniAgencijeOrganizacione zao : listaMenadzera) {
            zao.proveriDatumRok();
        }
    }
}

public class LicniAsistent extends Klijent { // ConcreteSubject
    private String datumRok;

    public LicniAsistent(){
        super(null);
    }

    public void odrediDatum(){
        datumRok = "24.12.2013.";
    }
```

```
    public String vratiDatum(){
        return datumRok;
    }
}

interface KeteringTim {

    void kreirajPredjelo();

    void kreirajGlavnoJelo();

    void kreirajDezert();

    String vratiMeni();
}

public class Meni{
    private Predjelo predjelo;
    private GlavnoJelo glavnojelo;
    private Dezert dezert;

    public Dezert getDezert() {
        return dezert;
    }
    public GlavnoJelo getGlavnojelo() {
        return glavnojelo;
    }
    public Predjelo getPredjelo() {
        return predjelo;
    }
    public void setDezert(Dezert dezert) {
        this.dezert = dezert;
    }
    public void setGlavnojelo(GlavnoJelo glavnojelo) {
        this.glavnojelo = glavnojelo;
    }
    public void setPredjelo(Predjelo predjelo) {
        this.predjelo = predjelo;
    }
}

abstract class FormatMenia  {
    Meni elementimeni;

    public void poveziSaMeniem(Meni elementimeni) {
        this.elementimeni = elementimeni;
```

```
    }
    abstract public String vratiFormatMenia();
}

class StandardniFormat extends FormatMenia {
    public String vratiFormatMenia() {
        return " Meni se sastoji iz: \n Predjelo : "
                + elementimeni.getPredjelo().vratiPredjelo()
                + " \n Glavno jelo : "
                + elementimeni.getGlavnojelo().vratiGlavnoJelo()
                + " \n Dezert : "
                + elementimeni.getDezert().vratiDezert();
    }
}

class ObrnutiFormat extends FormatMenia {
    public String vratiFormatMenia() {
        return " Meni se sastoji iz: \n Dezert : "
                + elementimeni.getDezert().vratiDezert()
                + " \n Glavno jelo : "
                + elementimeni.getGlavnojelo().vratiGlavnoJelo()
                + " \n Predjelo : "
                + elementimeni.getPredjelo().vratiPredjelo();
    }
}

public class KulinarskiTimItaliano implements KeteringTim {
    private KulinarskiTimRoma ktroma;
    private ZaposleniAgencijeOrganizacione zao;

    public KulinarskiTimItaliano(KulinarskiTimRoma ktroma, ZaposleniAgencijeOrganizacija
zao) {
        this.ktroma = ktroma;
        this.zao = zao;
    }
    public void kreirajPredjelo() {
        ktroma.kreirajPredjelo();
    }
    public void kreirajGlavnoJelo() {
        ktroma.kreirajGlavnoJelo();
    }

    public void kreirajDezert() {
        ktroma.kreirajDezert();
    }
```

```
  public String vratiMeni() {
      return
            zao.vratiPodatke() + "\n\n" + ktroma.vratiMeni();
   }
}

public class KulinarskiTimRoma implements KeteringTim {
   private Meni elementimeni;
   private FormatMenia fm;

   public KulinarskiTimRoma(FormatMenia fm) {
      elementimeni = new Meni();
      this.fm = fm;
      this.fm.poveziSaMeniem(elementimeni);
   }
   public void kreirajPredjelo() {
      elementimeni.setPredjelo(new SkampiSaBademom());
   }
   public void kreirajGlavnoJelo() {
      elementimeni.setGlavnojelo(new Pica());
   }
   public void kreirajDezert() {
      elementimeni.setDezert(new Sladoled());
   }
   public String vratiMeni() {
      return fm.vratiFormatMenia();
   }
}

public interface Predjelo {
   String vratiPredjelo ();
}

public class SkampiSaBademom implements Predjelo {
   public String vratiPredjelo () {
      return " Skampi sa bademima u umaku ";
   }
}

public class Proja implements Predjelo {
   public String vratiPredjelo () {
      return " Proja sa spanacem";
  }
}
```

```
public interface GlavnoJelo  {
   String vratiGlavnoJelo ();
}

public class Gulas implements GlavnoJelo{
   public String vratiGlavnoJelo () {
      return "Gulas";
   }
}

public class Pica implements GlavnoJelo {
   public String vratiGlavnoJelo () {
      return " Pica ";
   }
}

public interface Dezert {
   String vratiDezert ();
}

public class Orasnica implements Dezert{
   public String vratiDezert() {
      return " Orasnica ";
   }
}

public class Sladoled implements Dezert{
   public String vratiDezert() {
      return " Sladoled ";
   }
}

public class Main  // Main class  {
  public static void main(String[] args) {
      Klijent klijent;

      MenadzerOrganizacijeProstora mop = new MenadzerOrganizacijeProstora(null);
      MenadzerOrganizacijeOsoblja moo = new MenadzerOrganizacijeOsoblja(mop);

      //kreiranje manija italijanske kuhinje standardnog formata
      FormatMenia fm1 = null;
      fm1 = new StandardniFormat();
      KulinarskiTimRoma ktroma = new KulinarskiTimRoma(fm1);
      KulinarskiTimItaliano ktitaliano = new KulinarskiTimItaliano(ktroma, moo);
      klijent = new Klijent(ktitaliano);
      klijent.konstruisi();
```

```
        PricaItalijanskaKuhinja itprica = new PricaItalijanskaKuhinja(klijent);
        itprica.prikaziMeni();

        System.out.println("\n-------------------------------------------------------------------------\n");

        //kreiranje manija italijanske kuhinje obrnutog formata za Japance
        FormatMenia fm2 = null;
        fm2 = new ObrnutiFormat();
        KulinarskiTimRoma ktroma2 = new KulinarskiTimRoma(fm2);
        KulinarskiTimItaliano ktitaliano2 = new KulinarskiTimItaliano(ktroma2, null);
        klijent = new Klijent(ktitaliano2);
        klijent.konstruisi();
        PricaItalijanskaKuhinja itprica2 = new PricaItalijanskaKuhinja(klijent);
        itprica2.prikaziMeni();

        System.out.println("\n----------  Datum: ----------");
        //**** observer patern ****
        LicniAsistent asistent = new LicniAsistent();
        // asisten odredjuje datum, on za sada jedini zna koji je to datum
        asistent.odrediDatum();
        // menadzeri dobijaju referencu na asistenta
        mop.setLa(asistent);
        moo.setLa(asistent);
        // klijent sakuplja sve menadzere
        klijent.dodajMenadzeraUListu(moo);
        klijent.dodajMenadzeraUListu(mop);
        //klijent obavestava sve 'sakupljene' menadzere da pitaju asistenta za datum
        klijent.obavestiMenzadzereDaProvereDatum();

        System.out.println("Menadzer organizacije osoblja je saznao datum do kad treba da zavrsi
svoju obavezu:\n"+moo.getDatumRok());
        System.out.println("Menadzer organizacije prostora je saznao datum do kad treba da zavrsi
svoju obavezu:\n"+mop.getDatumRok());
    }
}
```

Rezultat programa:

```
*********************************************
 Menadzer sektora za organizaciju prostora: Milan Rakic
Kontakt telefon: 064/456 78 88
Okvirna cena: 20.000,00 dinara
 Menadzer sektora za organizaciju usluznog osoblja: Marina Boljanac
Kontakt telefon: 064/456 78 77
Okvirna cena: 18.000,00 dinara
*********************************************
```

Meni se sastoji iz:
Predjelo - Skampi sa bademima u umaku
Glavno jelo - Pica
Dezert -Sladoled
Italijanska kuhinja jeste jedna od najpopularnijih, ali i jedna od najpoznatijih kuhinja u svetu. Italijani vole da prilikom kuvanja koriste sezonsko voće i povrće pa se svaki ovaj segment njihove kuhinje znatno menja sa promenom godišnjeg doba.

Meni se sastoji iz:
Dezert - Sladoled
Glavno jelo - Pica
Predjelo - Skampi sa bademima u umaku
Italijanska kuhinja jeste jedna od najpopularnijih, ali i jedna od najpoznatijih kuhinja u svetu. Italijani vole da prilikom kuvanja koriste sezonsko voće i povrće pa se svaki ovaj segment njihove kuhinje znatno menja sa promenom godišnjeg doba.

---------- Datum: ----------
Menadzer organizacije osoblja je saznao datum do kad treba da zavrsi svoju obavezu:
24.12.2013.
Menadzer organizacije prostora je saznao datum do kad treba da zavrsi svoju obavezu:
24.12.2013.

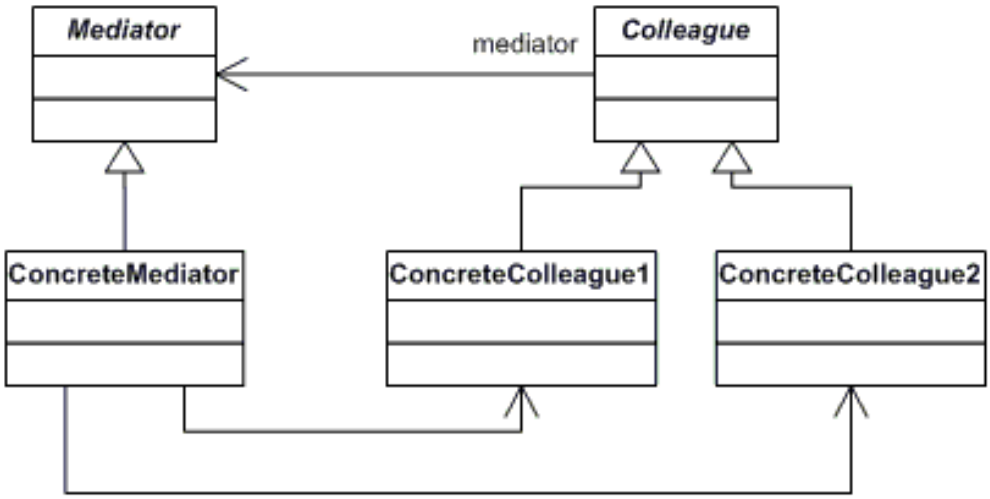
3.3. Mediator uzor

Definicija:

Definiše objekat koji sadrži skup objekata koji su u međusobnoj interakciji. Interakcija između dva objekta može se nezavisno menjati u odnosu na druge interakcije. Pomoću mediatora uspostavlja se slaba veza između objekata čime se povećava održivost i modularnost sistema.

Pojašnjena definicija:

Definiše objekat **(Mediator)** koji sadrži skup objekata **(ConcreteColleague1, ConcreteColleague2)** koji su u međusobnoj interakciji. Interakcija između dva objekta može se nezavisno menjati u odnosu na druge interakcije. Pomoću mediatora uspostavlja se slaba veza između objekata **(ConcreteColleague1, ConcreteColleague2)** čime se povećava održivost i modularnost sistema.



Učesnici u Mediator uzoru su:

- **Mediator**
- **ConcreteMediator**
- **Colleague classes**

Primana Mediator uzora na primeru agencije za ketering usluge:

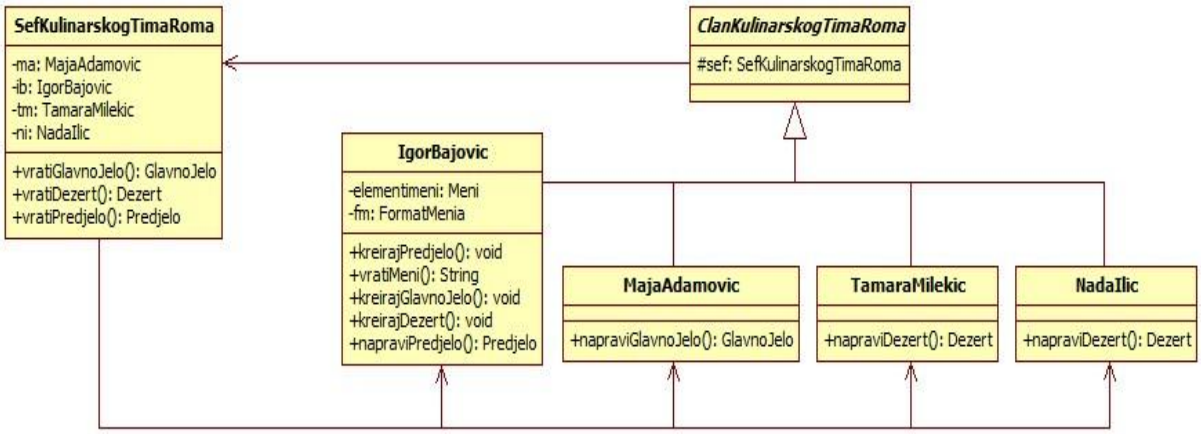
Šef kulinarskog tima Roma zadužio je Igora Bajovića da kordinira procesom pripreme menia i njegovih delova (obroka): predjela, glavnog jela i dezerta.
Igor Bajovic postavlja zahtev šefu laboratorije u procesu pripreme menia da pronađe članove tima koji su specijalizovani za pravljenje glavnog jela i dezerta, dok je on sam specijalizovan za pravljenje predjela. On sam nema vremena da određuje ko će pored njega učestvovati u izradi menia.

Sef kulinarskog tima Roma pronašao je sledeće članove tima: Nada Ilić i Tamara Milekić su specijalizovani za dezert. Maja Adamović je specijalizovana za glavno jelo.

Napomena :
Igor Bajović taj koji preuzima na sebe obavezu sklapanja menia (u zavisnosti od datog formata menia). Njemu ustvari kulinarski tim Italiano delegira svoj zahtev za izradom menia. Zbog toga u implementaciji, odnosnoi nadogradnji dosadašnje implementacije mediator paternom, *klasa KulinarskiTimRoma* postaje *klasa IgorBajovic*!

Primena definicije na Agenciju za ketering usluge:

Definiše objekat **(SefKulinarskogTimaRoma)** koji sadrži skup objekata **(NadaIlic, MajaAdamovic, IgorBajovic, TamaraMilekic)** koji su u međusobnoj interakciji. Interakcija između dva objekta može se nezavisno menjati u odnosu na druge interakcije. Pomoću mediatora uspostavlja se slaba veza između **objekata (NadaIlic, MajaAdamovic, IgorBajovic, TamaraMilekic)** čime se povećava održivost i modularnost sistema.



Učesnici u Mediator uzoru su:

- **SefKulinarskogTimaRoma** (ConcreteMediator)
- **NadaIlic, MajaAdamovic, IgorBajovic, TamaraMilekic** (Colleague classes)

U datom slučaju, uvodimo apstraktnu klasu **ClanKulinarskogTimaRoma**, i klase **Igor Bajović, Nada Ilić, Tamara Milekić i Maja Adamović** koje nasleđuju klasu ClanKulinarskogTimaRoma. Uvodi se i klasa **SefKulinarskogTimaRoma.**

Igor Bajović, Nada Ilić, Tamara Milekić i Maja Adamović su članovi kulinarskog tima Roma, ali u datom slučaju niko od njih ne zna:

1. za postojanje svojih kolega
2. koja je čija obaveza, uloga u timu
3. ko učestvuje u kreiranju menia a ko ne

Ali znaju ko im je šef i preko njega članovi kulinarskog tima međusobno komuniciraju.

Šef kulinarskog tima Roma zna za postojanje svih članova tima, on je taj koji određuje ko će praviti dezert, ko glavno jelo a ko predjelo.
Klasa SefKulinarskogTimaRoma definiše interfejs za komunikaciju sa klasama Igor Bajović, Nada Ilić, Tamara Milekić i Maja Adamović.

Igor Bajović preko svog šefa uzima glavno jelo i predjelo (ne znajući ko ih je napravio) i sklapa meni. On nema nikakav kontakt sa ostalim članovima tima (takvu želju je i izrazio u zahtevu).

Programski kod:

```
/**
 *
 * @author Ivana
 */

interface Komponenta  {
        void prikaziMeni();
}

class Dekorator implements Komponenta {
   Komponenta komp;

   public Dekorator(Komponenta komp1) {
      komp = komp1;
   }
   public void prikaziMeni() {
      komp.prikaziMeni();
   }
}

class PricaItalijanskaKuhinja extends Dekorator {
   public PricaItalijanskaKuhinja(Komponenta komp1) {
      super(komp1);
   }
   public void prikaziMeni() {
      super.prikaziMeni();
      System.out.println("\nItalijanska kuhinja jeste jedna od najpopularnijih, ali i jedna od najpoznatijih kuhinja u svetu.\nItalijani vole da prilikom kuvanja koriste sezonsko voće i povrće pa se svaki ovaj segment njihove kuhinje znatno menja sa promenom godišnjeg doba.  ");
   }
}
```

```
class PricaTradicionalnaKuhinja extends Dekorator {
   public PricaTradicionalnaKuhinja(Komponenta komp1) {
      super(komp1);
   }
   public void prikaziMeni() {
      super.prikaziMeni();
      System.out.println("\n Tradicionalna domaca kuhinja je najbolja !");
   }
}

public abstract class ZaposleniAgencijeOrganizacione {
   protected ZaposleniAgencijeOrganizacione zao;
   protected LicniAsistent la;
   protected String datumRok;

   public ZaposleniAgencijeOrganizacione(ZaposleniAgencijeOrganizacione zao) {
      this.zao = zao;
   }
   public String vratiPodatke() {
      String podaci = "";
      podaci = "\n" + vratiSvojePodatke();
      if (this.zao != null) {
         podaci = zao.vratiPodatke() + podaci;
      }
      return podaci;
   }
   abstract String vratiSvojePodatke();
   abstract void proveriDatumRok();

   public void setLa(LicniAsistent la) {
      this.la = la;
   }
   public String getDatumRok() {
      return datumRok;
   }
}

public class MenadzerOrganizacijeProstora extends ZaposleniAgencijeOrganizacione {

public MenadzerOrganizacijeProstora(ZaposleniAgencijeOrganizacione zao1) {
      super(zao1);
   }
   String vratiSvojePodatke() {
      return " Menadzer sektora za organizaciju prostora: Milan Rakic\nKontakt telefon: 064/456
78 88\nOkvirna cena: 20.000,00 dinara";
   }
```

```
    public void proveriDatumRok() {
        datumRok = la.vratiDatum();
    }
}

public class MenadzerOrganizacijeOsoblja extends ZaposleniAgencijeOrganizacione{

    public MenadzerOrganizacijeOsoblja(ZaposleniAgencijeOrganizacione zao1) {
        super(zao1);
    }

    String vratiSvojePodatke() {
        return " Menadzer sektora za organizaciju usluznog osoblja: Marina Boljanac\nKontakt
telefon: 064/456 78 77\nOkvirna cena: 18.000,00 dinara";
    }
    public void proveriDatumRok() {
        datumRok = la.vratiDatum();
    }
}

public class Klijent  implements Komponenta {
    KeteringTim ktim;
    List <ZaposleniAgencijeOrganizacione> listaMenadzera;

    public Klijent(KeteringTim ktim) {
        this.ktim = ktim;
        listaMenadzera = new ArrayList<ZaposleniAgencijeOrganizacione>();
    }

    void Konstruisi() {
        ktim.kreirajPredjelo();
        ktim.kreirajGlavnoJelo();
        ktim.kreirajDezert();
    }
    public void prikaziMeni() {
        System.out.println(ktim.vratiMeni());
    }
    public void dodajMenadzeraUListu(ZaposleniAgencijeOrganizacione zao){
        listaMenadzera.add(zao);
    }
    public void obavestiMenzadzereDaProverеDatum(){
        for (ZaposleniAgencijeOrganizacione zao : listaMenadzera) {
            zao.proveriDatumRok();
        }
    }
}
```

```
public class LicniAsistent extends Klijent {
   private String datumRok;

   public LicniAsistent(){
      super(null);
   }
   public void odrediDatum(){
      datumRok = "24.12.2013.";
   }

   public String vratiDatum(){
      return datumRok;
   }
}

interface KeteringTim  {

   void kreirajPredjelo();

   void kreirajGlavnoJelo();

   void kreirajDezert();

   String vratiMeni();
}

public class KulinarskiTimItaliano implements KeteringTim {
   private KulinarskiTimRoma ktroma;
   private ZaposleniAgencijeOrganizacione zao;

   public KulinarskiTimItaliano(KulinarskiTimRoma ktroma, ZaposleniAgencijeOrganizacija
zao) {
      this.ktroma = ktroma;
      this.zao = zao;
   }
   public void kreirajPredjelo() {
      ktroma.kreirajPredjelo();
   }
   public void kreirajGlavnoJelo() {
      ktroma.kreirajGlavnoJelo();
   }
   public void kreirajDezert() {
      ktroma.kreirajDezert();
   }
```

```
    public String vratiMeni() {
        return
                zao.vratiPodatke() + "\n\n" + ktroma.vratiMeni();
    }
}

public abstract class ClanKulinarskogTimaRoma { // Colleague
    protected SefKulinarskogTimaRoma sef;

    public ClanKulinarskogTimaRoma(SefKulinarskogTimaRoma sef) {
        this.sef = sef;
    }
}

public class IgorBajovic extends ClanKulinarskogTimaRoma implements KeteringTim {
// ConcreteColleague1
    private Meni elementimeni;
    private FormatMenia fm;

    // zna samo ko mu je sef, ne zna ko ce biti zaduzen za pravljenje obroka pored njega
    public IgorBajovic(SefKulinarskogTimaRoma sef1) {
        super(sef1);
        elementimeni = new Meni();
    }

    // on je specijalizovan za pravljenje predjela
    public void kreirajPredjelo() {
        elementimeni.setPredjelo(napraviPredjelo());
    }

    // on ne zna ko ce kreirati glavno jelo vec samo od sefa uzima to glavno jelo
    public void kreirajGlavnoJelo() {
        elementimeni.setGlavnojelo(sef.vratiGlavnoJelo());
    }

    // on ne zna ko ce kreirati dezert vec samo od sefa uzima taj dezert
    public void kreirajDezert() {
        elementimeni.setDezert(sef.vratiDezert());
    }

    // on koordinira ceo proces izrade menia i on ga spaja u celinu i vraca
    public String vratiMeni() {
        return fm.vratiFormatMenia();
    }
```

```
    // on sam pravi predjelo
    public Predjelo napraviPredjelo() {
        return new SkampiSaBademom();
    }

    public void postaviFmIPoveziSaMeniem (FormatMenia fm) {
        this.fm = fm;
        this.fm.poveziSaMeniem(elementimeni);
    }
}

public class MajaAdamovic extends ClanKulinarskogTimaRoma{ // ConcreteColleague2

    public MajaAdamovic(SefKulinarskogTimaRoma sef1){
        super(sef1);
    }
    public GlavnoJelo napraviGlavnoJelo (){
        return new Pica();
    }
}

public class NadaIlic extends ClanKulinarskogTimaRoma{ // ConcreteColleague3

     public NadaIlic(SefKulinarskogTimaRoma sef1){
        super(sef1);
    }
    public Dezert napraviDezert (){
        return new Sladoled();
    }
}

public class TamaraMilekic extends ClanKulinarskogTimaRoma{ // ConcreteColleague4

    public TamaraMilekic(SefKulinarskogTimaRoma sef1){
        super(sef1);
    }
    public Dezert napraviDezert (){
        return new Sladoled();
    }
}

public class SefKulinarskogTimaRoma { // Mediator
    private MajaAdamovic ma;
    private IgorBajovic ib;
    private TamaraMilekic tm;
    private NadaIlic ni;
```

```
    public SefKulinarskogTimaRoma(){
        ma = new MajaAdamovic(this);
        ib = new IgorBajovic(this);
        tm = new TamaraMilekic(this);
        ni = new NadaIlic(this);
    }

    public IgorBajovic getIb() {
        return ib;
    }
    public GlavnoJelo vratiGlavnoJelo() {
        return ma.napraviGlavnoJelo();
    }
    public Dezert vratiDezert() {
        return ni.napraviDezert();
    }
    public Predjelo vratiPredjelo() {
        return ib.napraviPredjelo();
    }
}

public class Meni{
    private Predjelo predjelo;
    private GlavnoJelo glavnojelo;
    private Dezert dezert;

    public Dezert getDezert() {
        return dezert;
    }
    public GlavnoJelo getGlavnojelo() {
        return glavnojelo;
    }
    public Predjelo getPredjelo() {
        return predjelo;
    }
    public void setDezert(Dezert dezert) {
        this.dezert = dezert;
    }
    public void setGlavnojelo(GlavnoJelo glavnojelo) {
        this.glavnojelo = glavnojelo;
    }
    public void setPredjelo(Predjelo predjelo) {
        this.predjelo = predjelo;
    }
}
```

```
abstract class FormatMenia  {
   Meni elementimeni

   public void poveziSaMeniem(Meni elementimeni) {
      this.elementimeni = elementimeni;
   }
   abstract public String vratiFormatMenia();
}

class StandardniFormat extends FormatMenia {
   public String vratiFormatMenia() {
      return " Meni se sastoji iz: \n Predjelo : "
            + elementimeni.getPredjelo().vratiPredjelo()
            + " \n Glavno jelo : "
            + elementimeni.getGlavnojelo().vratiGlavnoJelo()
            + " \n Dezert :"
            + elementimeni.getDezert().vratiDezert();
   }
}

class ObrnutiFormat extends FormatMenia {
   public String vratiFormatMenia() {
      return " Meni se sastoji iz: \n Dezert :  "
            + elementimeni.getDezert().vratiDezert()
            + " \n Glavno jelo : "
            + elementimeni.getGlavnojelo().vratiGlavnoJelo()
            + " \n Predjelo : "
            + elementimeni.getPredjelo().vratiPredjelo();
   }
}

public interface Predjelo {
   String vratiPredjelo ();
}

public class SkampiSaBademom implements Predjelo {
   public String vratiPredjelo () {
      return " Skampi sa bademima u umaku ";
   }
}

public class Proja implements Predjelo {
   public String vratiPredjelo () {
      return " Proja sa spanacem";
  }
}
```

```
public interface GlavnoJelo  {
   String vratiGlavnoJelo ();
}

public class Gulas implements GlavnoJelo{
   public String vratiGlavnoJelo () {
      return "Gulas";
   }
}

public class Pica implements GlavnoJelo {
   public String vratiGlavnoJelo () {
      return " Pica ";
   }
}

public interface Dezert {
   String vratiDezert ();
}

public class Orasnica implements Dezert{
   public String vratiDezert() {
      return " Orasnica ";
   }
}

public class Sladoled implements Dezert{
   public String vratiDezert() {
      return " Sladoled ";
   }
}

public class Main  // Main class  {
  public static void main(String[] args) {
      Klijent klijent;

      MenadzerOrganizacijeProstora mop = new MenadzerOrganizacijeProstora(null);
      MenadzerOrganizacijeOsoblja moo = new MenadzerOrganizacijeOsoblja(mop);

      SefKulinarskogTimaRoma sef = new SefKulinarskogTimaRoma();

      //kreiranje manija italijanske kuhinje standardnog formata
      FormatMenia fm1 = null;
      fm1 = new StandardniFormat();
      sef.getIb().postaviFmIPoveziSaMeniem(fm1);
```

```
        KulinarskiTimItaliano ktitaliano = new KulinarskiTimItaliano(sef.getIgorBajovic(), moo);
        klijent = new Klijent(ktitaliano);
        klijent.konstruisi();
        PricaItalijanskaKuhinja itprica = new PricaItalijanskaKuhinja(klijent);
        itprica.prikaziMeni();

        System.out.println("\n------------------------------------------------------------\n");

        //kreiranje manija italijanske kuhinje obrnutog formata za Japance
        FormatMenia fm2 = null;
        fm2 = new ObrnutiFormat();
        sef.getIb().postaviFmIPoveziSaMeniem(fm2);
        KulinarskiTimItaliano ktitaliano2 = new KulinarskiTimItaliano(sef.getIgorBajovic(), null);
        klijent = new Klijent(ktitaliano2);
        klijent.konstruisi();
        PricaItalijanskaKuhinja itprica2 = new PricaItalijanskaKuhinja(klijent);
        itprica2.prikaziMeni();

        System.out.println("\n----------  Datum: ----------");
        //**** observer patern ****
        LicniAsistent asistent = new LicniAsistent();
        // asisten odredjuje datum, on za sada jedini zna koji je to datum
        asistent.odrediDatum();
        // menadzeri dobijaju referencu na asistenta
        mop.setLa(asistent);
        moo.setLa(asistent);
        // klijent sakuplja sve menadzere
        klijent.dodajMenadzeraUListu(moo);
        klijent.dodajMenadzeraUListu(mop);
        //klijent obavestava sve 'sakupljene' menadzere da pitaju asistenta za datum
        klijent.obavestiMenzadzereDaProvereDatum();

        System.out.println("Menadzer organizacije osoblja je saznao datum do kad treba da zavrsi
svoju obavezu:\n"+moo.getDatumRok());
        System.out.println("Menadzer organizacije prostora je saznao datum do kad treba da zavrsi
svoju obavezu:\n"+mop.getDatumRok());
    }
}
```

Rezultat programa:

****
Menadzer sektora za organizaciju prostora: Milan Rakic
Kontakt telefon: 064/456 78 88
Okvirna cena: 20.000,00 dinara
Menadzer sektora za organizaciju usluznog osoblja: Marina Boljanac
Kontakt telefon: 064/456 78 77
Okvirna cena: 18.000,00 dinara
****

Standardni meni se sastoji iz:
Predjelo : Skampi sa bademima u umaku
Glavno jelo : Pica
Dezert : Sladoled

Dodatak:
Italijanska kuhinja jeste jedna od najpopularnijih, ali i jedna od najpoznatijih kuhinja u svetu. Italijani vole da prilikom kuvanja koriste sezonsko voće i povrće pa se svaki ovaj segment njihove kuhinje znatno menja sa promenom godišnjeg doba.

Meni za japanske goste se sastoji iz:
Dezert : Sladoled
Glavno jelo : Pica
Predjelo : Skampi sa bademima u umaku

Dodatak:
Italijanska kuhinja jeste jedna od najpopularnijih, ali i jedna od najpoznatijih kuhinja u svetu. Italijani vole da prilikom kuvanja koriste sezonsko voće i povrće pa se svaki ovaj segment njihove kuhinje znatno menja sa promenom godišnjeg doba.

---------- Datum: ----------
Menadzer organizacije osoblja je saznao datum do kad treba da zavrsi svoju obavezu:
24.12.2013.
Menadzer organizacije prostora je saznao datum do kad treba da zavrsi svoju obavezu:
24.12.2013.

Zaključak

Ako se dešava da se pri razvoju nekog softverskog sistema, u jednom trenutku, susretnete sa problemom koji ste već nekada rešavali ali ne možete da se setite tačnog rešenja, pre svega se zapitajte da li na dobar način pristupate svom radu. Kako biste izbegli gubljenje mnogo sati, možda i dana ponovnog rešavanja problema, počnite da zapisujete svoja rešenja, opišite ih jasno i precizno. Iz ovakve tačke gledanja su se i razvili paterni – već dokazana rešenja iskusnih programera. Možda ste i vi sledeći iskusni programer koji će ostalim kolegama obezbediti još jedan novi patern, i na taj način indirektno poboljšati njihov programski kod. Uživajte u suštinskom razumevanju opštih osobina paterna i nikada vas neće izneveriti.

Literatura

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