УНИВЕРЗИТЕТ У БЕОГРАДУ

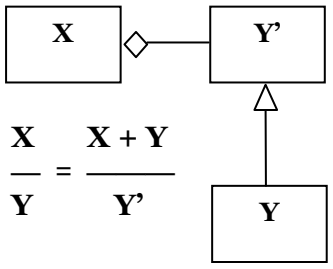
ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

Катедра за софтверско инжењерство

СОФТВЕРСКИ ПАТЕРНИ

Аутор:

Др Синиша Влајић ванр.проф.



Београд - 2014.

СП: Структурни патерни

Структурни патерни *описују сложене структуре међусобно повезаних класа и објеката.*

Постоје следећи структурни патерни:

1. Adapter патерн - Конвертује интерфејс неке класе у други интерфејс који клијент очекује. Адаптер патерн омогућава заједнички рад класа које имају некомбатибилне интерфејсе .

2. Bridge патерн - Одваја (декуплује) апстракцију од њене имплементације тако да се оне могу мењати независно.

3. Composite патерн - Објекти се састављају (компонују) у структуру стабла како би представили хијерархију целине и делова. Composite патерн омогућава да се једноставни и сложени објекти третирају јединствено.

4. Decorator патерн - Придружује додатне одговорности (функционалности) до објекта динамички. Decorator патерн обезбеђује флексибилност у избору подкласа које проширују функционалност.

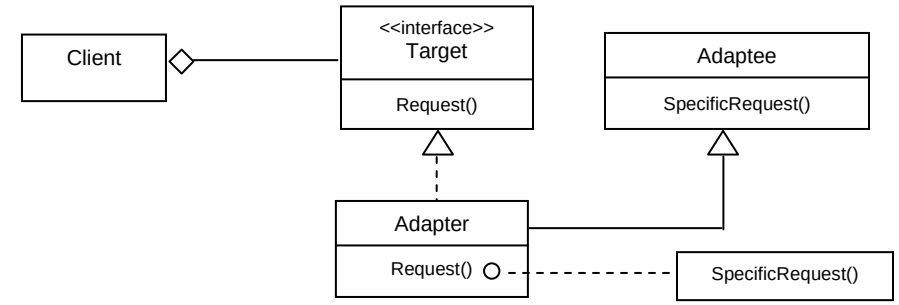
5. Facade патерн - Обезбеђује јединствен интерфејс за скуп интерфејса неког подсистема. Facade патерн дефинише интерфејс високог нивоа који омогућава да се подсистем лакше користи.

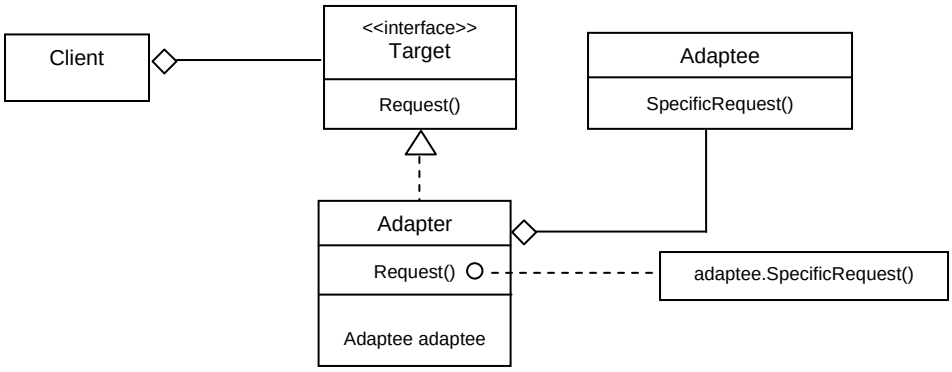
6. Flyweight патерн - Користи дељење да ефикасно подржи велики број ситних објеката.

7. Proxy патерн - Обезбеђује посредника за приступање другом објекту како би се омогућио контролисани приступ до њега.

СП1: Adapter патерн
Дефиниција
Конвертује интерфејс неке класе у други интерфејс који клијент очекује. Адаптер патерн омогућава заједнички рад класа које имају некомбатибилне интерфејсе.

Структура Adapter патерна се може јавити у два облика:





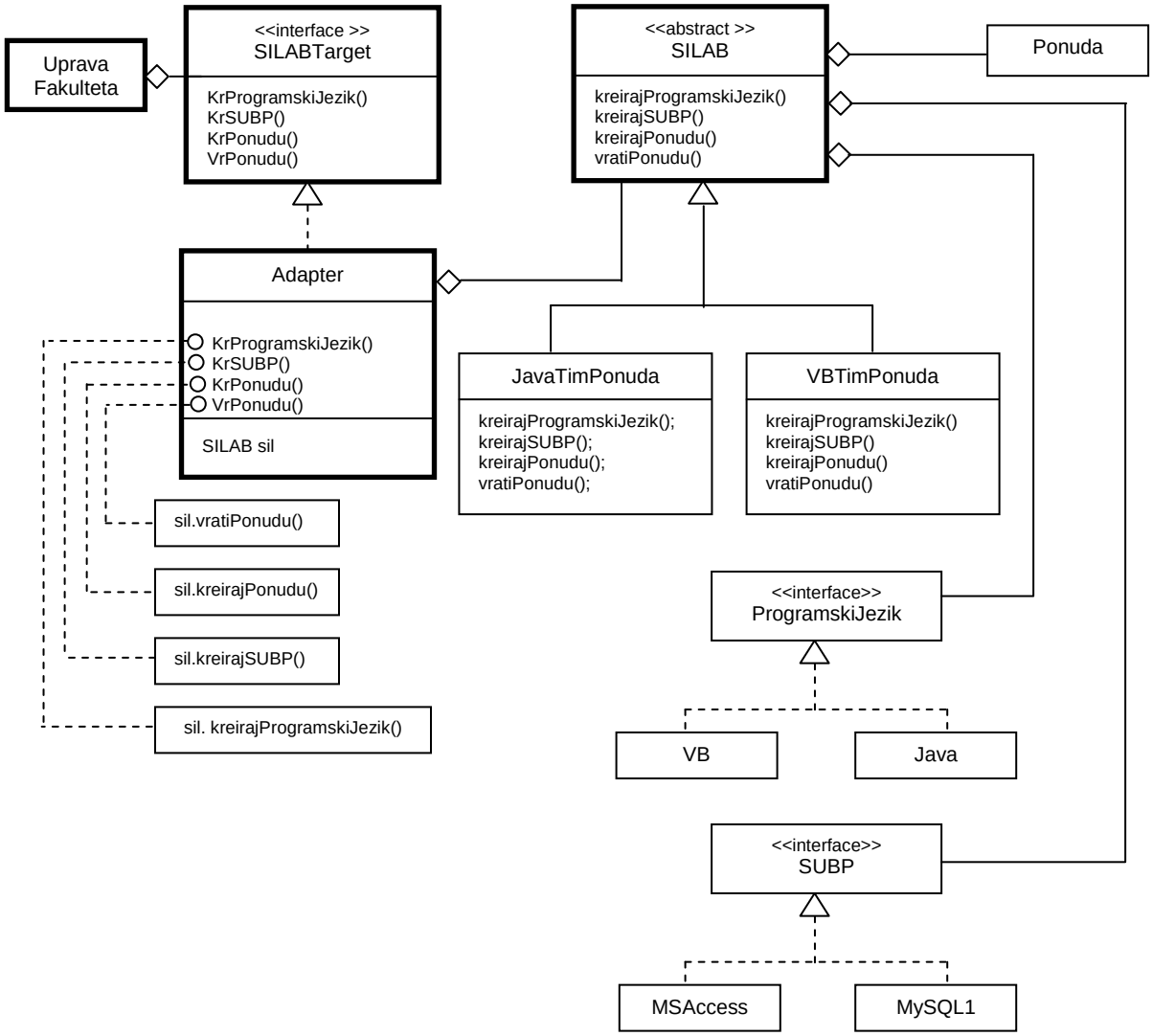
Пример Adapter paterna

Кориснички захтев PAD1: *Управа Факултета је послала захтев Лабораторији за софтверско инжењерство да промени свој интерфејс, тако што ће имена операција kreirajProgramskiJezik(), kreirajSUBP(), kreirajPonudu() и vratiPonudu() променити у KrProgramskiJezik(), KrSUBP(), KrPonudu() и VrPonudu(). Тимови који праве понуде треба да прилагоде (адаптирају) стари интерфејс (SILAB) новом интерфејсу (SILABTarget), који очекује Управа Факултета без промене, старог интерфејса.*

```
//Улога: Дефинише доменски-специфичан интерфејс који класа UpravaFakulteta користи.
interface SILABTarget    // Target
{ void KrProgramskiJezik();
  void KrSUBP();
  void KrPonudu();
  String VrPonudu();
}

//Улога: Адаптира (прилагођава) интерфејс SILAB интерфејсу SilabTarget.
class Adapter implements SILABTarget   // Adapter
{ SILAB sil;
  Adapter(SILAB sil1) {sil=sil1; }
  public void KrProgramskiJezik(){sil.kreirajProgramskiJezik();}
  public void KrSUBP(){sil.kreirajSUBP();}
  public void KrPonudu() {sil.kreirajPonudu();}
  public String VrPonudu(){return sil.vratiPonudu();}
}

// Улога: Сарађује са интерфејсом SILAB преко интерфејса SILABTarget.
class UpravaFakulteta   // Client
{
  SILABTarget silta;

  UpravaFakulteta(SILABTarget silta1){silta = silta1;}

  // Контролише конструкцију понуде коришћењем интерфејса SILABTarget.
  void Konstruisi()
    { silta.KrProgramskiJezik();
      silta.KrSUBP();
      silta.KrPonudu();
    }

   public static void main(String args[])
              {  UpravaFakulteta uf;
                 SILABTarget silta;
                 JavaTimPonuda jat = new JavaTimPonuda();
                 silta = new Adapter(jat);
                 uf = new UpravaFakulteta(silta);
                 uf.Konstruisi();
                 System.out.println("Ponuda java tima: " + jat.vratiPonudu());

                 VBTimPonuda vbt = new VBTimPonuda();
                 silta = new Adapter(vbt);
                 uf = new UpravaFakulteta(silta);
                 uf.Konstruisi();
                 System.out.println("Ponuda VB tima: " + vbt.vratiPonudu());
   }
}

// Улога: Дефинише постојећи интерфејс који треба адаптирати.
abstract class  SILAB // Adaptee
{  ProgramskiJezik  pj;
    SUBP subp;
    Ponuda pon;

   abstract void kreirajProgramskiJezik();
   abstract void kreirajSUBP();
   abstract void kreirajPonudu();
   abstract String vratiPonudu();
 }

class Ponuda {String ponuda;}
```

```
class JavaTimPonuda extends SILAB
{ JavaTimPonuda() {pon = new Ponuda();}
  public void kreirajProgramskiJezik(){pj = new Java();}
  public void kreirajSUBP() { subp = new MySQL();}
  public void kreirajPonudu() { pon.ponuda = "Programski jezik-" + pj.vratiProgramskiJezik() + " SUBP-" +
                                    subp.vratiSUBP();}
  public String vratiPonudu(){return pon.ponuda;}
}


class VBTimPonuda extends SILAB
{
  VBTimPonuda(){pon = new Ponuda();}
  public void kreirajProgramskiJezik(){pj = new VB();}
  public void kreirajSUBP() {subp = new MSAccess();}
  public void kreirajPonudu() { pon.ponuda = "Programski jezik-" + pj.vratiProgramskiJezik() + " SUBP-" +
                                    subp.vratiSUBP();}
  public String vratiPonudu(){return pon.ponuda;}
}
```

Дијаграм класа примера PAD1:



Објашњење примера:

Коментар примера:

- Разлика између креационих, структурних и узора понашања се огледа у њиховој **одговорности**. Креациони узори су одговорни за процес креирања објеката. Структурни узори су одговорни за рад са композицијом класа или објеката. Узори понашања су одговорни за сарадњу између класа или објеката.
- У овом примеру кренули смо да надограђујемо и мењамо **Builder** узор.
- Напомена: Обично узоре почињемо да примењујемо у току одржавања и надоградње програма. У случају Адаптер патерна постојећи интерфејс (SILAB) се прилагођава новом интерфејсу (SILABTarget).
- Управа факултета је била одговорна код Builder узора за процес прављења понуде. Она је позивала методу:

```
void Konstruisi()
  { sil.kreirajProgramskiJezik();
    sil.kreirajSUBP();
    sil.kreirajPonudu();
  }
```

- Управа Факултета тражи од Лабораторије за софтверско инжењерство да прилагоди интерфејс **SILAB:**

```
interface SILAB // Adaptee
{ void kreirajProgramskiJezik();
  void kreirajSUBP();
  void kreirajPonudu();
  String vratiPonudu();
}
```

sa интерфејсом **SILABTarger**:

```
interface SILABTarget    // Target
{ void KrProgramskiJezik();
  void KrSUBP();
  void KrPonudu();
  String VrPonudu();
}
```

- То се постиже помоћу Адаптера:

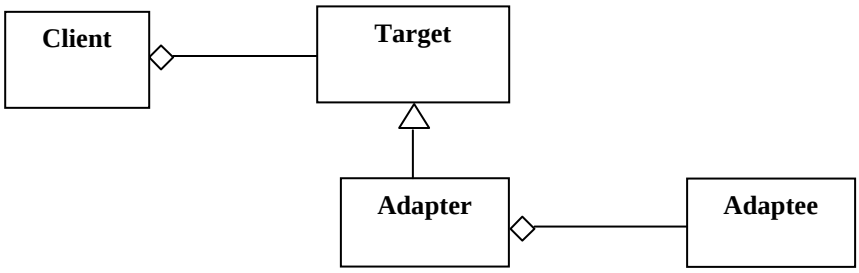
```
class Adapter implements SILABTarget   //Adapter
{ SILAB sil;
  Adapter(SILAB sil1) {sil=sil1; }
  public void KrProgramskiJezik(){sil.kreirajProgramskiJezik();}
  public void KrSUBP(){sil.kreirajSUBP();}
  public void KrPonudu() {sil.kreirajPonudu();}
  public String VrPonudu(){return sil.vratiPonudu();}
}
```

Адаптер прилагођава два различита интерфејса (SILABTarget и SILAB).

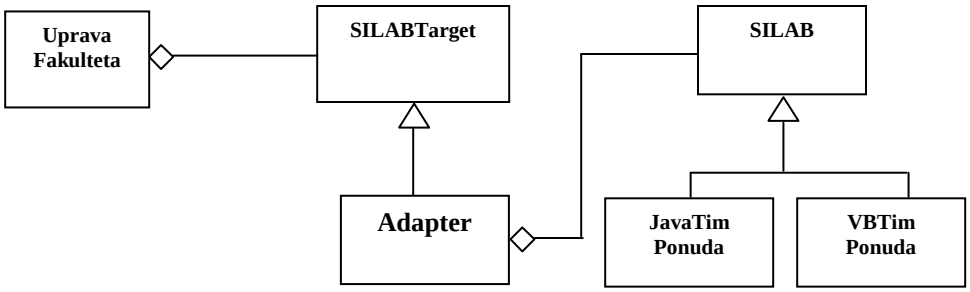
- Управа Факултета ће сада позивати *Konstruisi* методу новог интерфејса.

```
void Konstruisi()
   { silta.KrProgramskiJezik();
     silta.KrSUBP();
     silta.KrPonudu();
   }
```

- Адаптер узор има следећу структуру:



- У конкретном случају Адаптер узор има следећу структуру:

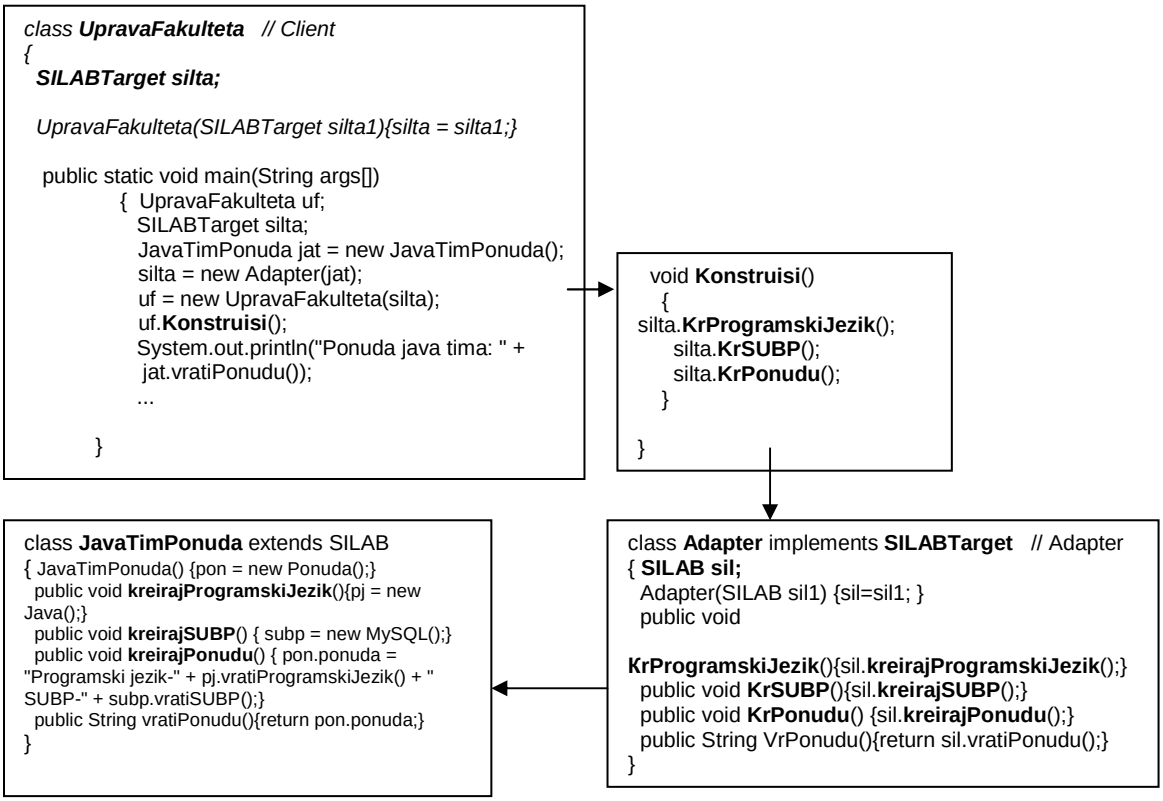


- Клијент је поставио захтев да се промени интерфејс. Треба да се постигну два циља помоћу Адаптер узора:
  а) Да се прилагоди постојећи интерфејс клијентовом.
  б) Не треба да мењамо наш постојећи интерфејс и класе које реализују интерфејс ако они могу да обезбеде жељену функционалност.

- Било би веома лоше када би смо на сваки захтев клијента мењали наш интерфејс. Интерфејс има смисла да се мења ако му се додаје нова функционалност.

- Пример Адаптер узора можемо представити у случају односа између:
  а) Наредби Јаве (Target) и Оперативног система (Adaptee) који се прилагођавају помоћу JVM (Adapter). Наредбе Јаве се пресликавају у наредбе конкретног Оперативног система.
  b) Наредби Јаве (Target) и SUBP (Adaptee) који се прилагођавају помоћу драјвера (Adapter). Наредбе Јаве се пресликавају у наредбе конкретног СУБП.

- Анализа програма:


class UpravaFakulteta // Client
{
SILABTarget silta;
UpravaFakulteta(SILABTarget silta1){silta = silta1;}
public static void main(String args[])
{ UpravaFakulteta uf;
SILABTarget silta;
JavaTimPonuda jat = new JavaTimPonuda();
silta = new Adapter(jat);
uf = new UpravaFakulteta(silta);
uf.Konstruisi();
System.out.println("Ponuda java tima: " +
jat.vratiPonudu());
...
}
void Konstruisi()
{
silta.KrProgramskiJezik();
silta.KrSUBP();
silta.KrPonudu();
}
}
class Adapter implements SILABTarget // Adapter
{ SILAB sil;
Adapter(SILAB sil1) {sil=sil1; }
public void
KrProgramskiJezik(){sil.kreirajProgramskiJezik();}
public void KrSUBP(){sil.kreirajSUBP();}
public void KrPonudu() {sil.kreirajPonudu();}
public String VrPonudu(){return sil.vratiPonudu();}
}
class JavaTimPonuda extends SILAB
{ JavaTimPonuda() {pon = new Ponuda();}
public void kreirajProgramskiJezik(){pj = new
Java();}
public void kreirajSUBP() { subp = new MySQL();}
public void kreirajPonudu() { pon.ponuda =
"Programski jezik-" + pj.vratiProgramskiJezik() + "
SUBP-" + subp.vratiSUBP();}
public String vratiPonudu(){return pon.ponuda;}
}

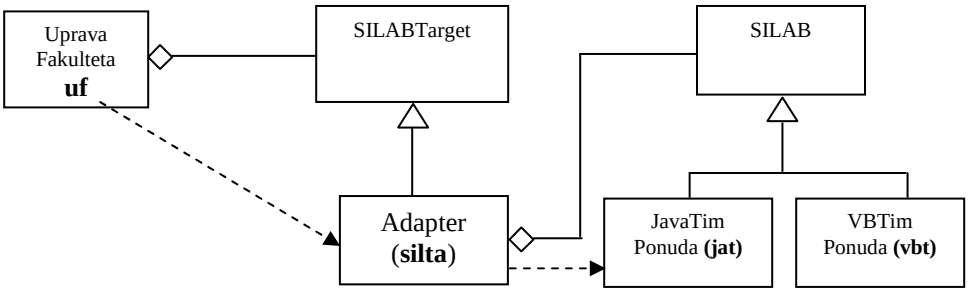
Када се повезују објекти узора то се ради на следећи начин:

a) Прво се креира објекат који од никога не зависе (**jat**).

b) Затим се креира адаптер објекат (**silta**), који се преко конструктора повезује са конкретним објектом који се адаптира (**jat**).

c) Затим се креира клијентски објекат (**uf**), који се преко конструктора повезује са конкретним адаптер објектом (**silta**).

- Креирају се објекти из десне у леву страну.



Важно правило: *Уколико имамо два класе, при чему нека класа X користи класу Y (класа Y је независна у односу на класу X), за класу X можемо да кажемо да је клијент класа док је класа Y сервер класа. При креирању клијентске и серверске класе, увек се прво формира серверска па тек онда клијентска класа.*

```
class X // Клијент класа
{Y y;

 X(Y y1){y=y1;}
}
class Y // Сервер класа
  {…}

public static void main(String args[])
    { Y y = new Y();
      X x = new X(y);     …
   }
```

Задатак Z-AD1:

1. Прилагодити клијенту методу **PrikaziStara()** са методом **PrikaziNova()**. Додати програм на месту где су дате три тачке.

```
interface Target
{ …
}

class Adapter …
{ Adaptee adaptee;
  …
}

class  Adaptee
{
 void prikaziStara(){System.out.println("Danas je lep dan.");}
}


class Client
{  …
   Client(...){tar=tar1;}

   public static void main(String args[])
          {  …
             ad.prikaziNova();
           }
}
```

Решење Z-AD1:

```
interface Target
{ void prikaziNova();
}

class Adapter implements Target
{ Adaptee adaptee;
   Adapter(Adaptee adaptee1){adaptee = adaptee1;}
   public void prikaziNova() {adaptee.prikaziStara();}
}


class  Adaptee
{ void prikaziStara(){System.out.println("Danas je lep dan.");}}


class Client
{ Target tar;
   Client(Target tar1){tar=tar1;}

   public static void main(String args[])
          {  Adaptee adaptee = new Adaptee();
             Adapter ad = new Adapter(adaptee);
             ad.prikaziNova();
          }
}
```

Задатак Z-AD2: Додати програм на месту три тачке како би се добила порука: Danas je lep dan.

```
class Server
{
  void Prikazi(...) {c.PrikaziPoruku();}
}


class Client1
{
   Server s;
   Client1(Server s1) {...}

   public static void main(String args[])
           {
             ...
             Client1 c = new Client1(s);
             c.Prikazi();
           }

  void Prikazi(){s.Prikazi(...);}

  void PrikaziPoruku(){System.out.println("Danas je lep dan.");}
}
```

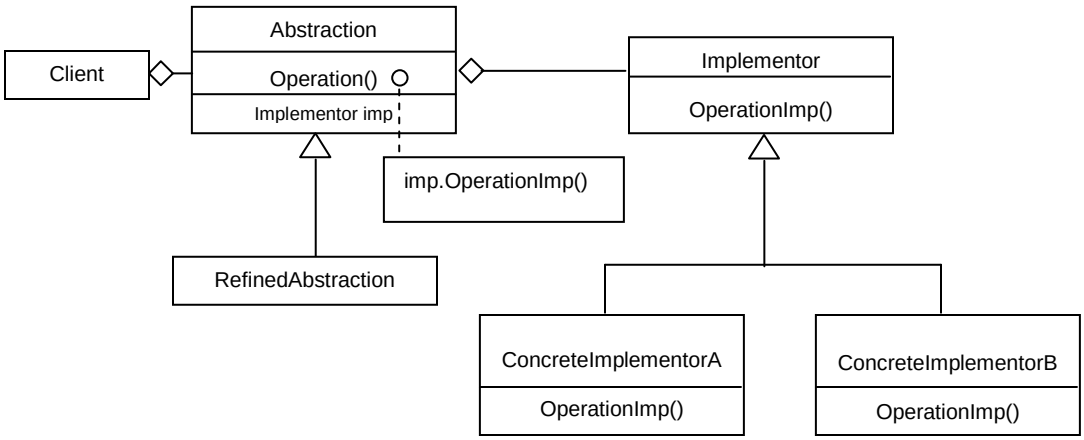
Решење Z-AD2:

```
class Server
{  void Prikazi(Client1 c) {c.PrikaziPoruku();}
}


class Client1
{  Server s;
   Client1(Server s1) {s=s1;}

   public static void main(String args[])
           {
             Server s = new Server();
             Client1 c = new Client1(s);
             c.Prikazi();
           }

  void Prikazi(){s.Prikazi(this);}

  void PrikaziPoruku(){System.out.println("Danas je lep dan.");}
}
```

СП2: Bridge патерн

Дефиниција
Одваја (декуплује) апстракцију од њене имплементације тако да се оне могу мењати независно.

Структура Bridge патерна



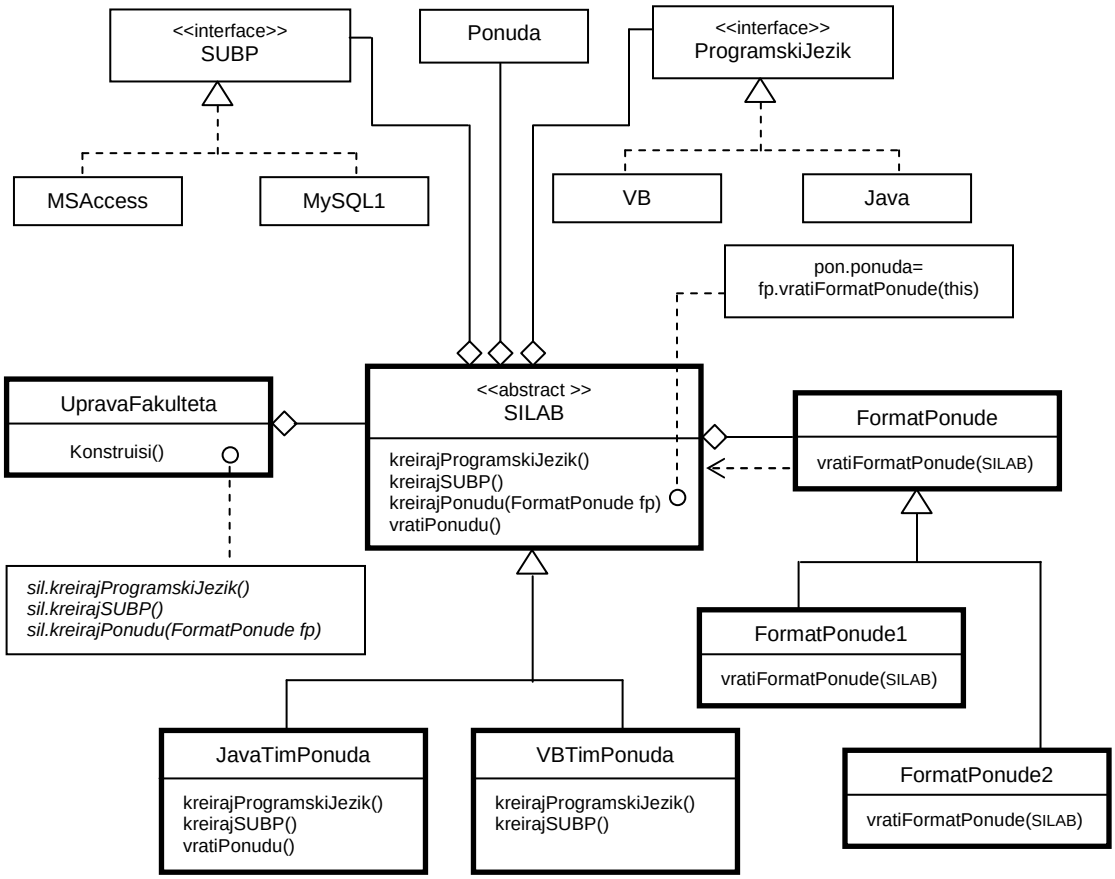
Пример Bridge узора

Кориснички захтев PBR1: *Управа Факултета је тражила од тимова да промене формат Понуде тако што ће се прво навести СУБП па тек онда програмски језик.*
Java тим у понуди треба да наведе ко су аутори Понуде. Управа Факултета је поставила захтев да и стари формат понуде остану расположив.

```
class UpravaFakulteta   // Client
{
  SILAB sil;
  UpravaFakulteta(SILAB sil1){sil= sil1;}
  void Konstruisi(FormatPonude fp)
    { sil.kreirajProgramskiJezik();
      sil.kreirajSUBP();
      sil.kreirajPonudu(fp);
    }

   public static void main(String args[])
              {
                UpravaFakulteta uf;
                FormatPonude fp = null;
                if (args[0].equals("1")) fp = new FormatPonude1();
                if (args[0].equals("2")) fp = new FormatPonude2();

                JavaTimPonuda jat = new JavaTimPonuda();
                uf = new UpravaFakulteta(jat);
                uf.Konstruisi(fp);
                System.out.println("Ponuda java tima: " + jat.vratiPonudu());

               VBTimPonuda vbt = new VBTimPonuda();
               uf = new UpravaFakulteta(vbt);
               uf.Konstruisi(fp);
               System.out.println("Ponuda VB tima: " + vbt.vratiPonudu());
              }
 }

// Улога: Дефинише интерфејс апстракције.
abstract class  SILAB  // Abstracion
{  ProgramskiJezik  pj;
   SUBP subp;
   Ponuda pon;

   abstract void kreirajProgramskiJezik();
   abstract void kreirajSUBP();

   public String vratiPonudu(){return pon.ponuda;}
   public void kreirajPonudu(FormatPonude fp) { pon.ponuda = fp.vratiFormatPonude(this);}
 }

class JavaTimPonuda extends SILAB // RefinedAbstraction1
{ JavaTimPonuda() {pon = new Ponuda();}
  public void kreirajProgramskiJezik(){pj = new Java();}
  public void kreirajSUBP() { subp = new MySQL();}
  public String vratiPonudu(){return "Autor: Lab za soft. inzenjerstvo: " + pon.ponuda;}
}

class VBTimPonuda extends SILAB // RefinedAbstraction2
{ VBTimPonuda(){pon = new Ponuda();}
  public void kreirajProgramskiJezik(){pj = new VB();}
  public void kreirajSUBP() {subp = new MSAccess();}
}

class Ponuda {String ponuda;}

// Улога: Дефинише интерфејс за имплементационе класе  (FormatPonude1, FormatPonude2).
abstract class FormatPonude   // Implementor
{  abstract String vratiFormatPonude(SILAB sil);
}
```

```
// Улога: Имплементира интерфејсFormatPonude.
class FormatPonude1 extends FormatPonude   // Concrete Implementor A
{  String vratiFormatPonude(SILAB sil)
     {  return  "Programski jezik-" + sil.pj.vratiProgramskiJezik() + "  SUBP-" + sil.subp.vratiSUBP();}
}

class FormatPonude2 extends FormatPonude // Concrete Implementor B
{
String vratiFormatPonude(SILAB sil)
   { return  "SUBP-" + sil.subp.vratiSUBP() + " Programski jezik-" + sil.pj.vratiProgramskiJezik();}
}

interface ProgramskiJezik {String vratiProgramskiJezik();}

class Java implements ProgramskiJezik { public String vratiProgramskiJezik(){return "Java";}}

class VB implements ProgramskiJezik { public String vratiProgramskiJezik(){return "VB";}}

interface SUBP {String vratiSUBP();}

class MySQL implements SUB P{ public String vratiSUBP(){return "MySQL";}}

class MSAccess implements SUBP { public String vratiSUBP(){return "MS Access";}}
```

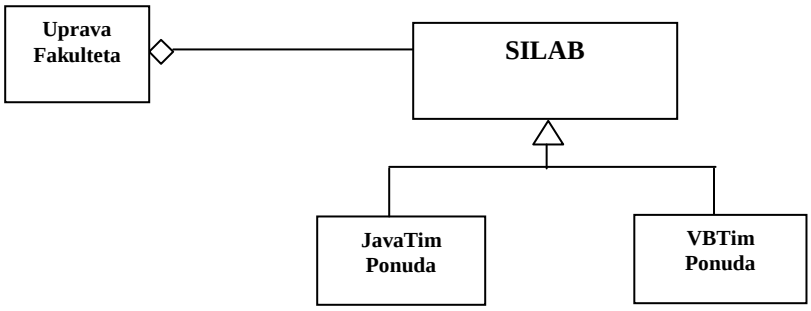
Дијаграм класа примера PBR1:



Прво се изабере конкретна имплементација (FormatPonude). Затим се имплементација повеже са апстракцијом (SUBP). На крају се клијент (UpravaFakulteta) повезује са апстракцијом (SUBP).
Овај патерн подсећа на top down методу помоћу које се коришћењем концепата апстракције и декомпозиције поједностављује сложеност једног проблема.

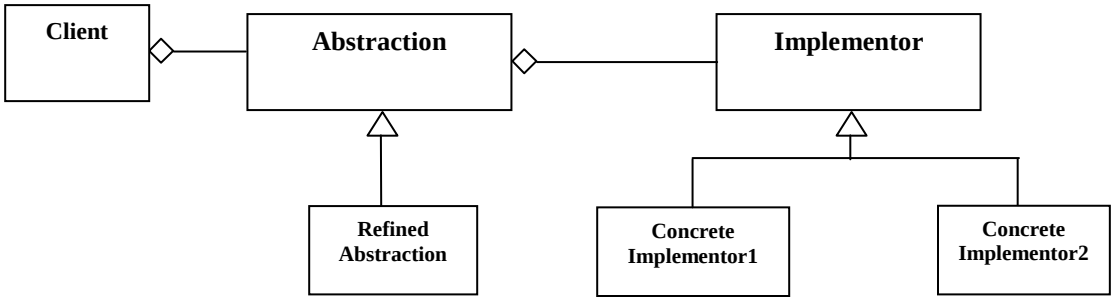
Коментар примера:

- У овом примеру кренули смо да надограђујемо и мењамо **Builder** узор.

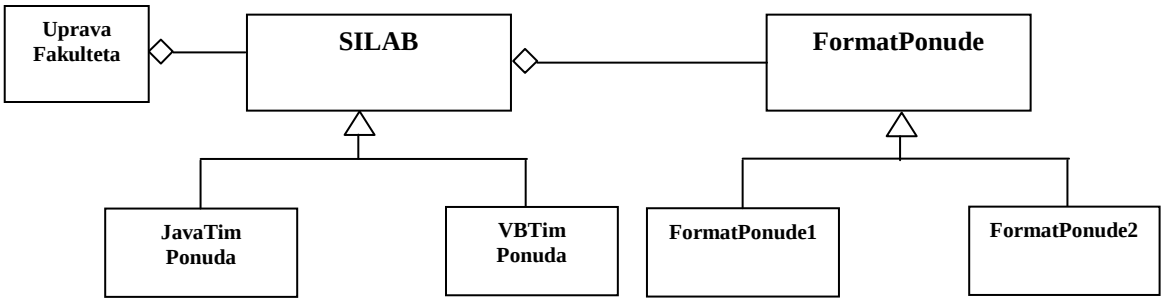


- Поставља се захтев у нашем примеру да се промени:
  а) формат понуде, тако што ће се прво навести СУБП па тек онда програмски језик.
  б) Јава тим треба да наведе ко су аутори понуде.

- Bridge узор има следећу структуру:



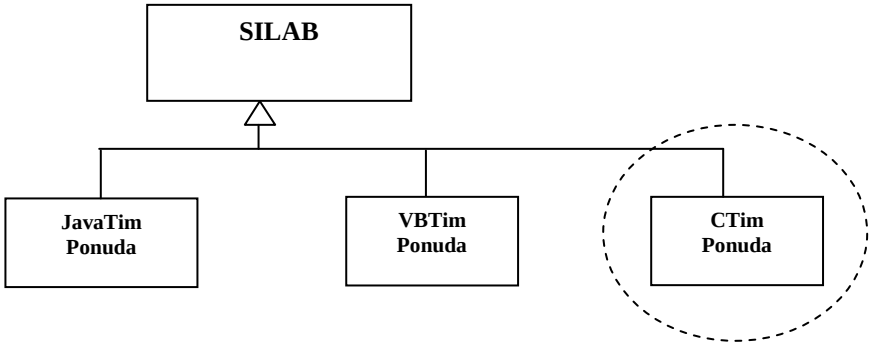
- У конкретном случају Bridge узор има следећу структуру:



- У наведеном случају проблем је био у томе што је сваки тим враћао исти формат понуде. Сада је могуће да тимови буду повезани и да враћају различите формате понуде.

- У класи JavaTimPonuda се обезбеђује захтев да Јава тим у понуди наводи ко су аутори понуде.

- Шта значи да се независно раздвајају апстракција од имплементације у Bridge узору? Уколико уведемо нову класу CTimPonuda (RefinedAbstraction), код Bridge узора, неће се морати мењати ништа што се односи на формат понуде(Implementor).



- Формати понуда се не мењају у свом генералном облику:
  а) **"Programski jezik-" + pon.pj.vratiProgramskiJezik() + " SUBP-" + pon.subp.vratiSUBP()**

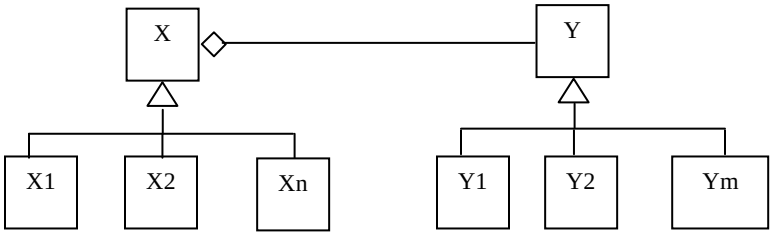
  б)**"SUBP-" + pon.subp.vratiSUBP() + " Programski jezik-" + pon.pj.vratiProgramskiJezik();**

Формат понуде има променљиве и непроменљиве делове. На променљиве делове утичу тимови са својим специјализованим понудама.

- Преко командне линије се бира формат понуде. Управа Факултета поставља захтев у ком ће формату бити понуда:

```
if (args[0].equals("1")) fp = new FormatPonude1();
if (args[0].equals("2")) fp = new FormatPonude2();
```

У наведеном примеру је омогућено да додавање нових тимова Лабораторије за софтверско инжењерство (нпр. *CTimPonuda*) и нових формата понуде (нпр. *FormatPonude3*) буде независно. То значи да било који тим Лабораторије за софтверско инжењерство и било који формат понуде могу бити повезани. На основу наведеног изводи се следећи закључак:
Уколико имамо класе или интерфејсе X и Y, где X садржи Y и где су из X изведене класе X1,X2,...,Xn, док су из класе Y изведене класе Y1,Y2,...,Ym, могуће је да се повеже било која класа која је изведена из X са класом која је изведена из Y.



То значи да било које Xi, i = (1,..,n) може бити повезано са било које Yj, j = (1,..,m).

Задатак Z-BR1: Додати програм на месту три тачке како би се добиле поруке:
Sutra ce biti jos lepsi dan.
Danas je lep dan.

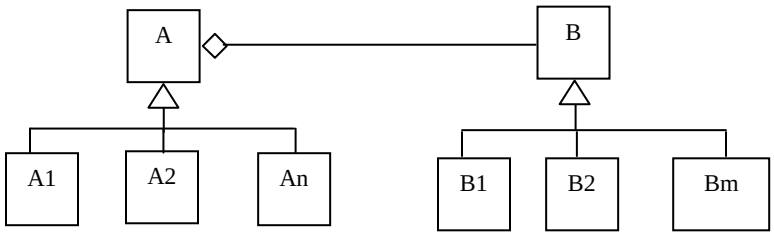
```
class Client2
  {
    Abstraction a;
    Client2 (Abstraction a1) {a=a1;}

     public static void main(String args[])
            {   ConcreteImplementorA ciA = new ConcreteImplementorA();
                ConcreteImplementorB ciB = new ConcreteImplementorB();
                RefinedAbstraction1 ra = new RefinedAbstraction1();
                Client2 c = new Client2(...);
                c.Prikazi(...); c.Prikazi(...);  }

     void Prikazi(Implementor i){a.Prikazi(i);}
   }

  abstract class  Abstraction { ... }

  class RefinedAbstraction1 extends Abstraction
  { void Prikazi(...){i.Prikazi();}
  }

  abstract class Implementor { ...}

  class ConcreteImplementorA extends Implementor
  { void Prikazi(){System.out.println("Danas je lep dan.");}
  }

  class ConcreteImplementorB extends Implementor
  { void Prikazi(){System.out.println("Sutra ce biti jos lepsi dan.");}
  }
```

Решење Z-BR1:

```
class Client2
{ Abstraction a;
  Client2 (Abstraction a1) {a=a1;}
   public static void main(String args[])
   {   ConcreteImplementorA ciA = new ConcreteImplementorA();
       ConcreteImplementorB ciB = new ConcreteImplementorB();
       RefinedAbstraction1 ra = new RefinedAbstraction1();
       Client2 c = new Client2(ra);
       c.Prikazi(ciB); c.Prikazi(ciA);

    }

  void Prikazi(Implementor i){a.Prikazi(i);}
 }


abstract class  Abstraction { abstract void Prikazi(Implementor i);}

class RefinedAbstraction1 extends Abstraction
   { void Prikazi(Implementor i){i.Prikazi();}}


abstract class Implementor { abstract void Prikazi();}

class ConcreteImplementorA extends Implementor
 { void Prikazi(){System.out.println("Danas je lep dan.");}}

class ConcreteImplementorB extends Implementor
{ void Prikazi(){System.out.println("Sutra ce biti jos lepsi dan.");}}
```

Задатак Z-BR2: Направити структуру класа где се било која класа из неког скупа класа А може повезати са било којом класом из неког скупа класа В.

Решење Z-BR2:

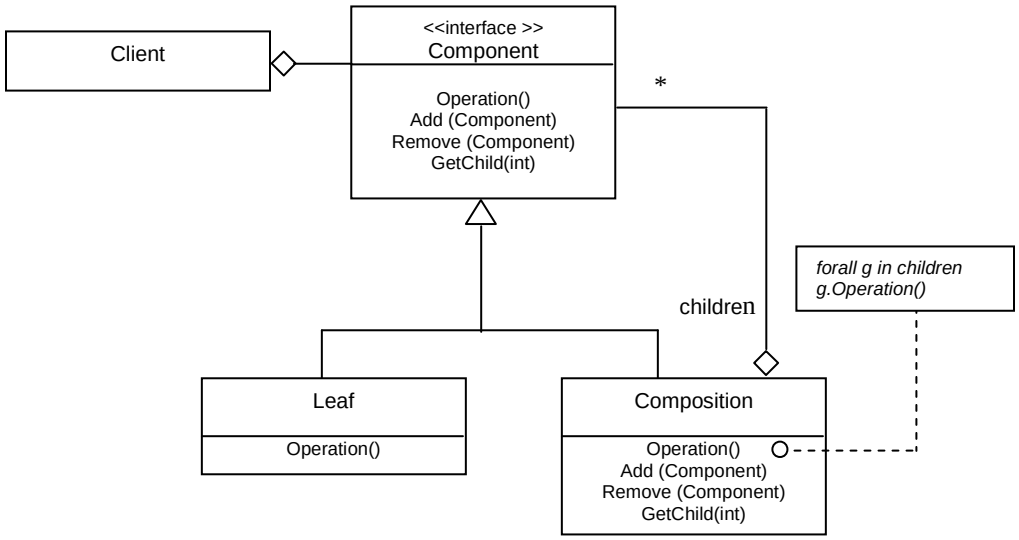


То значи да било које Ai, i = (1,..,n) може бити повезано са било које Bj, j = (1,..,m).

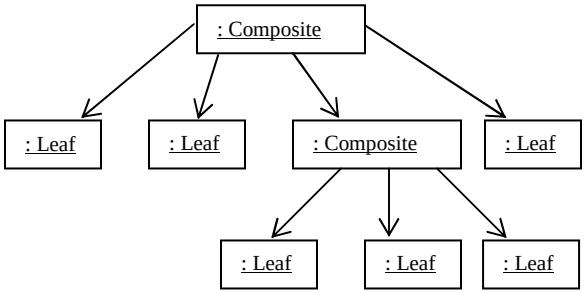
СП3. Composite патерн

Дефиниција
Објекти се састављају (компонују) у структуру стабла како би представили хијерархију целине и делова. *Composite* патерн омогућава да се једноставни и сложени објекти третирају јединствено.

Структура Composite патерна



Типична структура *Composite* објекта има следећи изглед:



Пример Composite патерна

Кориснички захтев PCO1*: Управа Факултета је тражила да тим који је у понуди навео Јаву као програмски језик детаљније образложи неке од најважнијих Јава технологија које ће се користити за израду софтверског система ПДС-а на ФОН-у, ту се посебно мисли на J2EE технологију.*

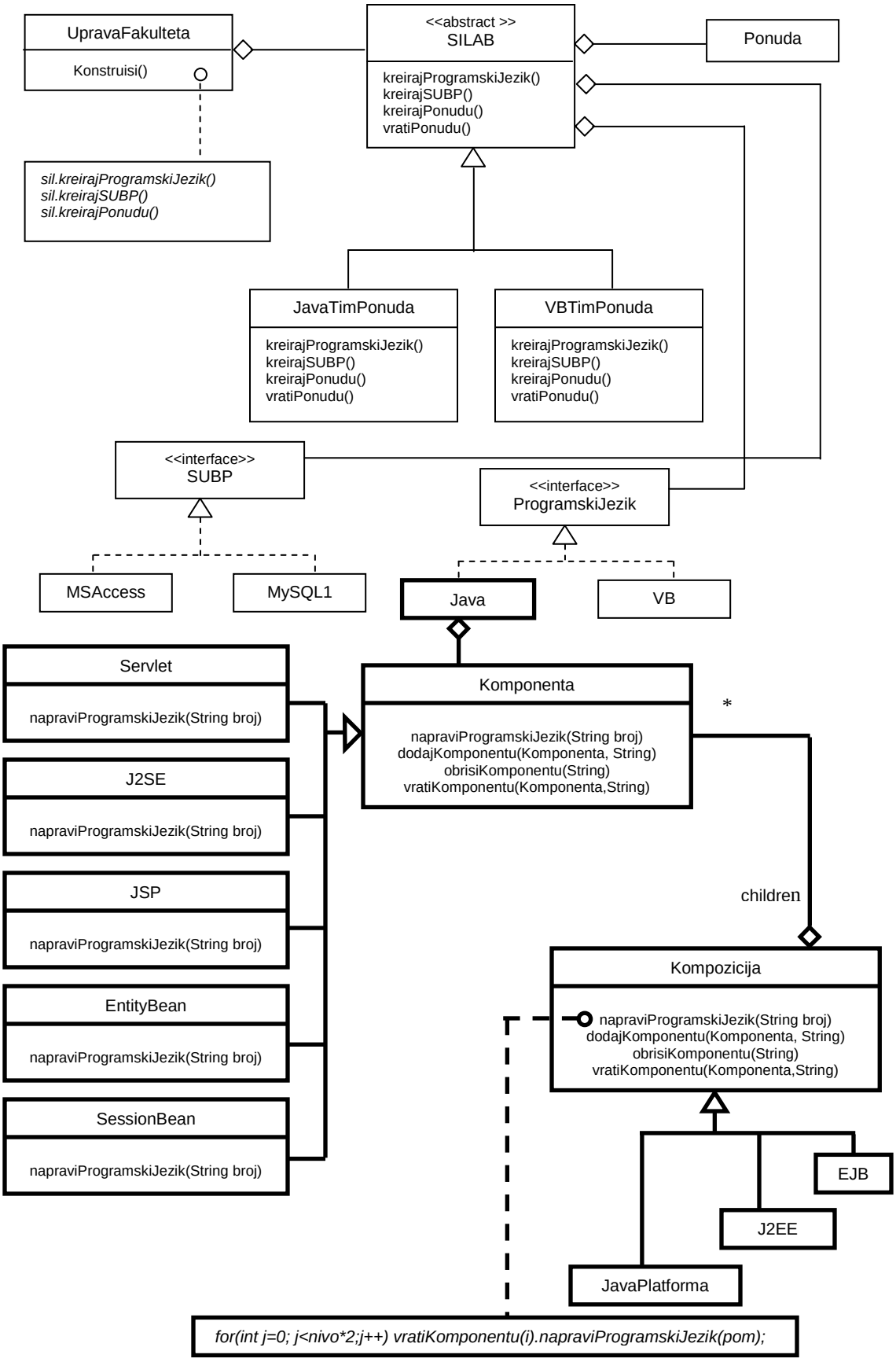
```
class UpravaFakulteta
{ SILAB sil;
  UpravaFakulteta(SILAB sil1){sil = sil1;}
  void Konstruisi()
    { sil.kreirajProgramskiJezik();
      sil.kreirajSUBP();
      sil.kreirajPonudu();
    }
   public static void main(String args[])
              {  UpravaFakulteta uf;
                 JavaTimPonuda jat = new JavaTimPonuda();
                 uf = new UpravaFakulteta(jat);
                 uf.Konstruisi();
                 System.out.println("Ponuda java tima: " + jat.vratiPonuda());
                 VBTimPonuda vbt = new VBTimPonuda();
                  uf = new UpravaFakulteta(vbt);
                  uf.Konstruisi();
                  System.out.println("Ponuda VB tima: " + vbt.vratiPonuda());
              }
}
abstract class  SILAB
{  ProgramskiJezik  pj;
    SUBP subp;
    Ponuda pon;
   abstract void kreirajProgramskiJezik();
   abstract void kreirajSUBP();
   abstract void kreirajPonudu();
   abstract String vratiPonudu();
 }
class Ponuda {String ponuda;}
class JavaTimPonuda extends SILAB
{ JavaTimPonuda() {pon = new Ponuda();}
  public void kreirajProgramskiJezik(){pj = new Java();}
  public void kreirajSUBP() { subp = new MySQL();}
  public void kreirajPonudu() { pon.ponuda = "\n Programski jezik:" + pj.vratiProgramskiJezik() + " \n SUBP:" + subp.vratiSUBP();}
  public String vratiPonudu(){return pon.ponuda;}
}
class VBTimPonuda extends SILAB
{ VBTimPonuda(){pon = new Ponuda();}
  public void kreirajProgramskiJezik(){pj = new VB();}
  public void kreirajSUBP() {subp = new MSAccess();}
  public void kreirajPonudu() { pon.ponuda = "\n Programski jezik:" + pj.vratiProgramskiJezik() + "  \n SUBP:" + subp.vratiSUBP();}
  public String vratiPonudu(){return pon.ponuda;}
}
interface ProgramskiJezik  {String vratiProgramskiJezik();}
class VB implements ProgramskiJezik { public String vratiProgramskiJezik(){return "VB";}}
interface SUBP {String vratiSUBP();}
class MySQL implements SUBP { public String vratiSUBP(){return "MySQL";}}
class MSAccess implements SUBP  { public String vratiSUBP(){return "MS Access";}}
// Улога: Манипулише објектима (компонентама) у структури помоћу Komponenta  интерфејса.
class Java implements ProgramskiJezik  // Client
{  Komponenta j;
   static String ponudaJava;
   Java()
    {  ponudaJava="\n";
       j = new JavaPlatforma("JavaPlatforma");
      J2SE j2se = new J2SE("J2SE");
      j.dodajKomponentu(j2se,"JavaPlatforma");
      J2EE j2ee = new J2EE("J2EE");
      j.dodajKomponentu(j2ee,"JavaPlatforma");
      EJB ejb = new EJB("EJB");
      j.dodajKomponentu(ejb,"J2EE");
      Servlet sr = new Servlet("Servlet");
      j.dodajKomponentu(sr,"J2EE");
      JSP jsp = new JSP("JSP");
       j.dodajKomponentu(jsp,"J2EE");
```

```
    EntityBean eb = new EntityBean("EntityBean");
    j.dodajKomponentu(eb,"EJB");
    SessionBean sb = new SessionBean("SessionBean");
    j.dodajKomponentu(sb,"EJB");
  }
  public String vratiProgramskiJezik()
  { j.napraviProgramskiJezik("1");
    return Java.ponudaJava;
  }
}
// Uloga: Deklarise interfejs za objekte koji ce da obrazuju strukturu. Deklarise interfejs za
// pristupanje i upravljanje objektima strukture.
class Komponenta  // Component
{ String sifraKomponente;
  static int nivo = 1;
  Komponenta(String sifraKomponente1){sifraKomponente = new String(sifraKomponente1);}
  String vratiSifruKomponente() {return sifraKomponente;}
  Komponenta vratiKomponentu(int i) {return null;}
  Komponenta vratiKomponentu(Komponenta Komponenta,String sifraKomponente1){return null;}
  void napraviProgramskiJezik(String broj) {}
  void dodajKomponentu(int i, Komponenta deteKomponenta) {}
  void dodajKomponentu(Komponenta deteKomponenta, String sifraRoditelja) {}
  void obrisiKomponentu(String sifraKomponente){}
  String opisTehnologije(){return "";}
}
// Uloga: Definise ponasanje za slozene objekte koji imaju decu-objekte. Cuva  decu-objekte. Implementira operacije
// interfejsa Komponenta.
class Kompozicija extends Komponenta // Composite
{ Komponenta kom[];
  Kompozicija(String sifraKomponente1){super(sifraKomponente1); kom = new Komponenta[5]; }
  Komponenta vratiKomponentu(int i) {return kom[i];}
  void dodajKomponentu(int i, Komponenta deteKomponenta) {kom[i]=deteKomponenta;}
  public void napraviProgramskiJezik(String broj) // parametar broj oznacava broj koji stoji ispred Java tehnologija
     {   // formatiranje Java ponude
         Java.ponudaJava = Java.ponudaJava + " " + broj + ":" + vratiSifruKomponente() + opisTehnologije();
         //**************************
         for (int i=0; i<5; i++)
          {   if (vratiKomponentu(i) != null)
               { nivo++;
                 // formatiranje Java ponude
                Java.ponudaJava = Java.ponudaJava + "\n";
                 for(int j=0; j<nivo*2;j++) Java.ponudaJava = Java.ponudaJava + " " ;
                  String pom = broj + "." + (i+1);
                 //**************************
                 vratiKomponentu(i).napraviProgramskiJezik(pom);
                 nivo--;
                }
          }
     }
 public void dodajKomponentu(Komponenta deteKomponenta, String sifraRoditelja)
 { int i;
   Komponenta roditeljKomponenta = vratiKomponentu(this,sifraRoditelja);   // ako nema roditelja vraca null
   if (roditeljKomponenta != null)
        { for(i=0; i<5;i++)
             if (roditeljKomponenta.vratiKomponentu(i) == null)
                { roditeljKomponenta.dodajKomponentu(i, deteKomponenta);
                  break;
                }
             if (i==5)
                { System.out.println("Ne moze da se unese dete-komponenta, jer je njena komponenta-roditelj popunila svu
                                      decu-komponente (maksimalno 5 dece):");
                }
          }
       else
         {  System.out.println("Roditelj sa sifrom:" + sifraRoditelja + " ne postoji: ");
         }
   }
 Komponenta vratiKomponentu(Komponenta Komponenta,String sifraKomponente1)
 {  if (Komponenta.vratiSifruKomponente().equals(sifraKomponente1))
```

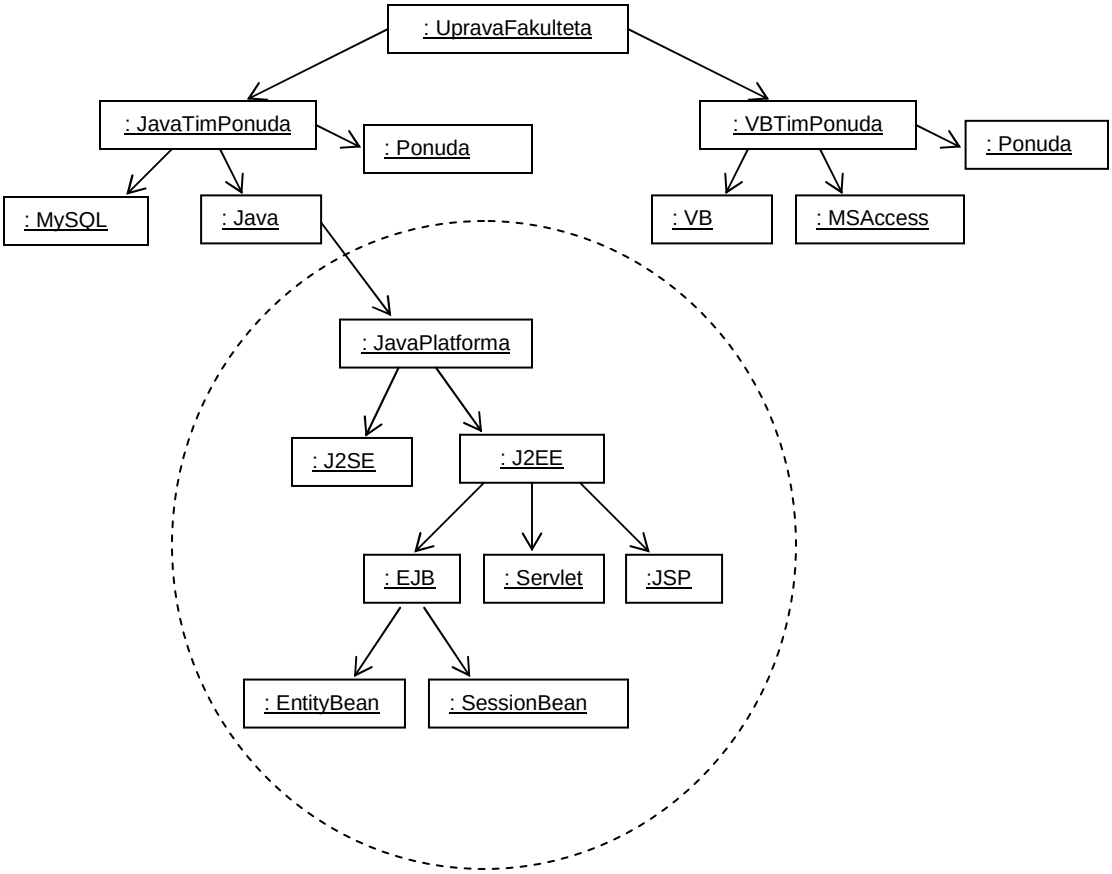
```
        return Komponenta;
      for(int i=0; i<5;i++)
        {  if(Komponenta.vratiKomponentu(i) != null)
              { if (Komponenta.vratiKomponentu(i).vratiSifruKomponente().equals(sifraKomponente1))
                   return Komponenta.vratiKomponentu(i);
                else
                   { Komponenta pom = vratiKomponentu(Komponenta.vratiKomponentu(i),sifraKomponente1);
                     if (pom!=null)
                          return pom;
                    }
              }
        }
     return null;
   }
}

// ********************************************************************************
class EJB extends Kompozicija // Composite
{ EJB(String sifraKomponente1) {super(sifraKomponente1);}
  String opisTehnologije() {return " - Java serverska tehnologija";}
}
class J2EE extends Kompozicija // Composite
{ J2EE(String sifraKomponente1) {super(sifraKomponente1);}
  String opisTehnologije() {return " - Java tehnologija za razvoj slozenih aplikacija";}
}
class JavaPlatforma extends Kompozicija // Composite
{ JavaPlatforma(String sifraKomponente1) {super(sifraKomponente1);}
}
// ********************************************************************************
// Улога: Представља просте објекте (Leaf) у  структури и дефинише њихово понашање. Прости
// објекти немају децу-објекте.
class EntityBean extends Komponenta   // Leaf
{ EntityBean(String sifraKomponente1) {super(sifraKomponente1);}
  public void napraviProgramskiJezik(String broj)
   { Java.ponudaJava = Java.ponudaJava + " " + broj + ":" + vratiSifruKomponente() + " - perzistentni podaci";}
}
class SessionBean extends Komponenta  // Leaf
{ SessionBean(String sifraKomponente1) {super(sifraKomponente1);}
public void napraviProgramskiJezik(String broj)
   {Java.ponudaJava = Java.ponudaJava + " "  + broj + ":" + vratiSifruKomponente() + " - poslovna logika";}
}
class Servlet extends Komponenta // Leaf
{ Servlet(String sifraKomponente1) {super(sifraKomponente1);}
 public void napraviProgramskiJezik(String broj)
 {Java.ponudaJava = Java.ponudaJava + " " + broj + ":" +  vratiSifruKomponente() + " - Web tehnologija (korisnici interfejs
  + poslovna logika)";}
}

class JSP extends Komponenta  // Leaf
{ JSP(String sifraKomponente1) {super(sifraKomponente1);}
public void napraviProgramskiJezik(String broj)
 { Java.ponudaJava = Java.ponudaJava + " " + broj + ":" + vratiSifruKomponente() + " - Web tehnologija (kor. interfejs)";}
}
class J2SE extends Komponenta // Leaf
{ J2SE(String sifraKomponente1) {super(sifraKomponente1);}
  public void napraviProgramskiJezik(String broj)
  { Java.ponudaJava = Java.ponudaJava + " " + broj + ":" + vratiSifruKomponente() + " - Standardna Java tehnologija";}
}
// ********************************************************************************
```

ZCO1: *Урадити за пример PCO1 методу obrisiKomponentu(String sifraKomponente), где ће се на основу параметра sifraKomponente наћи објекат са том шифром, након чега ће објекат бити обрисан из структуре.*

Дијаграм класа примера PCO1

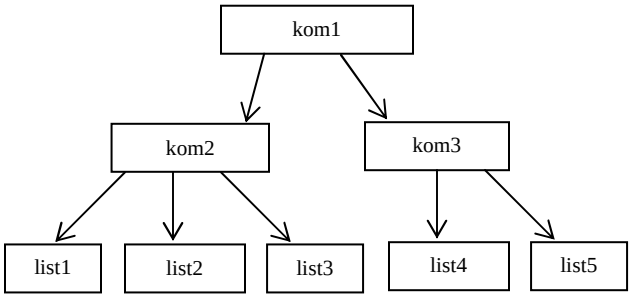


Објектни дијаграм примера PCO1



Задатак Z – CO1: Додати програм на месту три тачке како би се на стандардном излазу добило: Треба да се добије оваква структура објеката:

Kompozicija kom1 ima decu:
1.Kompozicija kom2 ima decu:
1.List1
2.List2
3. List3
2.Kompozicija kom3 ima decu:
1.List4
2.List5




```
class Client3
{
  public static void main(String arg[])
    {
      Kompozicija kom1 = new Kompozicija("kom1");  Kompozicija kom2 = new Kompozicija("kom2");
      Kompozicija kom3 = new Kompozicija("kom3");
      kom1.Dodaj(kom2);

      ...
      List list1 = new List("list1");  List list2 = new List("list2");  List list3 = new List("list3");

      ...
      List list4 = new List("list4"); List list5 = new List("list5");

      ...
      kom1.Prikazi();
    }
}
```

```
abstract class Komponenta
{ String Naziv;

  ...
  abstract void Prikazi();
  void Dodaj(Komponenta kom){}
}
```

```
class List extends Komponenta
{  List(String Naziv) {super(Naziv);}

   void Prikazi(){... }
}
```

```
class Kompozicija extends Komponenta
{ Komponenta deca[];   int brojDece;   static int  nivo = 0;

  Kompozicija(String Naziv) {super(Naziv); deca = new Komponenta[5];brojDece=0;}

  void Prikazi()
    {  System.out.println("Kompozicija " + Naziv + " ima decu:");

       for(int i=0;i<brojDece; i++)   {  nivo ++;  Pomeraj(i);     ...     nivo--;     }
    }

  void Dodaj(Komponenta kom){deca[brojDece]=kom; brojDece++;}
  void Pomeraj(int i) { for (int j=0;j<nivo;j++)  System.out.print("\t");   System.out.print((i + 1) + ".");  }
}
```

Решење Z-CO1:

```
class Client3
{
   public static void main(String arg[])
     {
       Kompozicija kom1 = new Kompozicija("kom1");  Kompozicija kom2 = new Kompozicija("kom2");
       Kompozicija kom3 = new Kompozicija("kom3");
       kom1.Dodaj(kom2);
       kom1.Dodaj(kom3);
       List list1 = new List("list1");  List list2 = new List("list2");  List list3 = new List("list3");
       kom2.Dodaj(list1);
       kom2.Dodaj(list2);
       kom2.Dodaj(list3);
       List list4 = new List("list4"); List list5 = new List("list5");
       kom3.Dodaj(list4);
       kom3.Dodaj(list5);
       kom1.Prikazi();
     }
}

abstract class Komponenta
{ String Naziv;
  Komponenta(String Naziv1){Naziv = Naziv1;}
  abstract void Prikazi();
  void Dodaj(Komponenta kom){}
}

class Kompozicija extends Komponenta
{ Komponenta deca[];   int brojDece;   static int  nivo = 0;

  Kompozicija(String Naziv) {super(Naziv); deca = new Komponenta[5];brojDece=0;}

  void Prikazi()
     {  System.out.println("Kompozicija " + Naziv + " ima decu:");
        for(int i=0;i<brojDece; i++)   {  nivo ++;  Pomeraj(i);      deca[i].Prikazi();     nivo--;     }
     }

  void Dodaj(Komponenta kom){deca[brojDece]=kom; brojDece++;}
  void Pomeraj(int i) { for (int j=0;j<nivo;j++)   System.out.print("\t");  System.out.print((i + 1) + ".");  }
}

class List extends Komponenta
{  List(String Naziv) {super(Naziv);}
   void Prikazi(){System.out.println(Naziv);  }}
```

Задатак Z – CO2: Додати програм на месту три тачке како би програм могао да претражи компоненте композитне структуре преко назива компоненте и да прикаже поруку о томе.

```
class Client4
{
   public static void main(String arg[])
     {
       Kompozicija kom1 = new Kompozicija("kom1");    Kompozicija kom2 = new Kompozicija("kom2");
       Kompozicija kom3 = new Kompozicija("kom3");     kom1.Dodaj(kom2);     kom1.Dodaj(kom3);
       List list1 = new List("list1");  List list2 = new List("list2");   List list3 = new List("list3");
       kom2.Dodaj(list1);  kom2.Dodaj(list2);    kom2.Dodaj(list3);   List list4 = new List("list4");  List list5 = new List("list5");
       kom3.Dodaj(list4);  kom3.Dodaj(list5);

       System.out.println("Unesi naziv komponente:");
       Scanner in = new Scanner(System.in);
       String nazivKomp = in.next();

       if (kom1.Nadji(...))
           System.out.println("Komponenta:" + nazivKomp + " postoji!");
       else
           System.out.println("Komponenta:" + nazivKomp + " ne postoji!");
        }
}
abstract class Komponenta
{ String Naziv;
  Komponenta(String Naziv1){Naziv = Naziv1;}
  void Dodaj(Komponenta kom){}
  boolean Nadji(String Naziv1){return Naziv.equals(Naziv1);}
}

class List extends Komponenta
     { List(String Naziv) {super(Naziv);}}

class Kompozicija extends Komponenta
{ Komponenta deca[];
  int brojDece;

  Kompozicija(String Naziv) {super(Naziv); deca = new Komponenta[5];brojDece=0;}
  void Dodaj(Komponenta kom){deca[brojDece]=kom; brojDece++;}

  boolean Nadji(String Naziv1)

    { if (...) return true;
      for(int i=0;i<brojDece; i++)

        {  if (deca[i].Nadji(Naziv1))       ...      }

      ...
    }

  }
```

Решење Z – CO2:

```
class Client4
{
   public static void main(String arg[])
     {
       Kompozicija kom1 = new Kompozicija("kom1");    Kompozicija kom2 = new Kompozicija("kom2");
       Kompozicija kom3 = new Kompozicija("kom3");
       kom1.Dodaj(kom2);     kom1.Dodaj(kom3);
       List list1 = new List("list1");  List list2 = new List("list2");   List list3 = new List("list3");
       kom2.Dodaj(list1);  kom2.Dodaj(list2);    kom2.Dodaj(list3);
       List list4 = new List("list4");  List list5 = new List("list5");
       kom3.Dodaj(list4);  kom3.Dodaj(list5);
       System.out.println("Unesi naziv komponente:");

       Scanner in = new Scanner(System.in);
       String nazivKomp = in.next();

       if (kom1.Nadji(nazivKomp))
           System.out.println("Komponenta:" + nazivKomp + " postoji!");
       else
           System.out.println("Komponenta:" + nazivKomp + " ne postoji!");
        }
}

abstract class Komponenta
{ String Naziv;
  Komponenta(String Naziv1){Naziv = Naziv1;}
  void Dodaj(Komponenta kom){}
  boolean Nadji(String Naziv1){return Naziv.equals(Naziv1);}
}

class Kompozicija extends Komponenta
{ Komponenta deca[];
  int brojDece;

  Kompozicija(String Naziv) {super(Naziv); deca = new Komponenta[5];brojDece=0;}
  void Dodaj(Komponenta kom){deca[brojDece]=kom; brojDece++;}

  boolean Nadji(String Naziv1)
    { if (super.Nadji(Naziv1)) return true;
      for(int i=0;i<brojDece; i++)
        {  if (deca[i].Nadji(Naziv1))
              return true;
        }
      return false;
    }

  }


class List extends Komponenta { List(String Naziv) {super(Naziv);}}
```

Задатак Z – CO3: Додати програм на месту три тачке како би програм могао да претражи компоненте композитне структуре преко назива компоненте и да прикаже сву децу тражене компоненте уколико она постоји.

```
import java.util.Scanner;

class Client5
{
   public static void main(String arg[])
     {
      Kompozicija kom1 = new Kompozicija("kom1");  Kompozicija kom2 = new Kompozicija("kom2");
      Kompozicija kom3 = new Kompozicija("kom3");
      kom1.Dodaj(kom2);  kom1.Dodaj(kom3);
      List list1 = new List("list1"); List list2 = new List("list2");   List list3 = new List("list3");
      kom2.Dodaj(list1); kom2.Dodaj(list2);  kom2.Dodaj(list3);
      List list4 = new List("list4");  List list5 = new List("list5");
      kom3.Dodaj(list4);   kom3.Dodaj(list5);
      System.out.println("Unesi naziv komponente:");
      Scanner in = new Scanner(System.in);
      String nazivKomp = in.next();

     Komponenta pom;
     pom = kom1.Nadji(nazivKomp);

     if (...)
        pom.Prikazi();
     else
        System.out.println("Ne postoji trazena komponenta!");
     }
 }

abstract class Komponenta
{ String Naziv;
  Komponenta(String Naziv1){Naziv = Naziv1;}
  void Dodaj(Komponenta kom){}
  Komponenta Nadji(String Naziv1){if (Naziv.equals(Naziv1)) return ...; return null;}
  String vratiNaziv() {return Naziv;}
  abstract void Prikazi();
}

class List extends Komponenta
 {  List(String Naziv) {super(Naziv);}
    void Prikazi(){System.out.println("Nema dece.");}
 }

class Kompozicija extends Komponenta
{ Komponenta deca[];
  int brojDece;
  Kompozicija(String Naziv) {super(Naziv); deca = new Komponenta[5];brojDece=0;}
  void Dodaj(Komponenta kom){deca[brojDece]=kom; brojDece++;}

Komponenta Nadji(String Naziv1)
   {   Komponenta pom = super.Nadji(Naziv1);

      if (pom!=null) ...;

      for(int i=0;i<brojDece; i++)    {   ...    }
      return null;
   }

  void Prikazi()
    { System.out.println("Deca od komponente "+ this.Naziv + " su:") ;
      for (int i=0;i<brojDece;i++)  {  System.out.println((i+1) + ". dete " + deca[i].vratiNaziv()); } }
}
```

Решење Z – CO3:

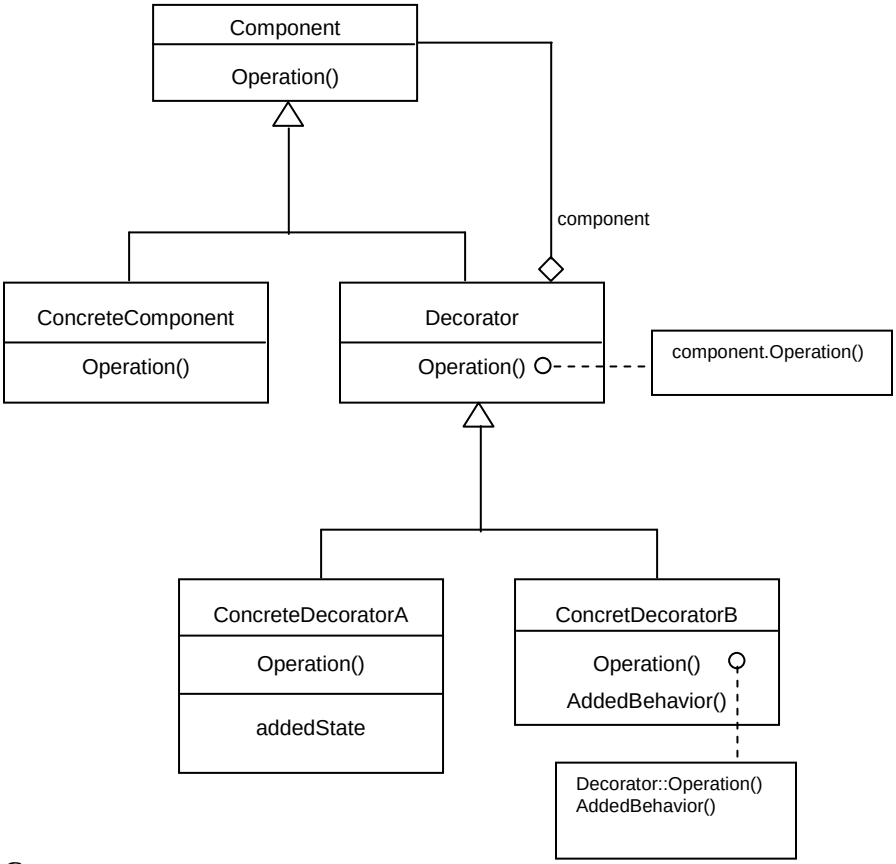
```
import java.util.Scanner;

class Client5
{
   public static void main(String arg[])
     {
       Kompozicija kom1 = new Kompozicija("kom1");  Kompozicija kom2 = new Kompozicija("kom2");
       Kompozicija kom3 = new Kompozicija("kom3");
       kom1.Dodaj(kom2);  kom1.Dodaj(kom3);
       List list1 = new List("list1"); List list2 = new List("list2");   List list3 = new List("list3");
       kom2.Dodaj(list1); kom2.Dodaj(list2);  kom2.Dodaj(list3);
       List list4 = new List("list4");  List list5 = new List("list5");
       kom3.Dodaj(list4);   kom3.Dodaj(list5);
       System.out.println("Unesi naziv komponente:");

       Scanner in = new Scanner(System.in);
       String nazivKomp = in.next();

      Komponenta pom;
      pom = kom1.Nadji(nazivKomp);
      if (pom!=null)
          pom.Prikazi();
      else
          System.out.println("Ne postoji trazena komponenta!");
      }
  }

class List extends Komponenta
 {  List(String Naziv) {super(Naziv);}
    void Prikazi(){System.out.println("Nema dece.");}
 }

abstract class Komponenta
{ String Naziv;
  Komponenta(String Naziv1){Naziv = Naziv1;}
  void Dodaj(Komponenta kom){}
  Komponenta Nadji(String Naziv1){if (Naziv.equals(Naziv1)) return this; return null;}
  String vratiNaziv() {return Naziv;}
  abstract void Prikazi();
}

class Kompozicija extends Komponenta
{ Komponenta deca[];
  int brojDece;
  Kompozicija(String Naziv) {super(Naziv); deca = new Komponenta[5];brojDece=0;}
  void Dodaj(Komponenta kom){deca[brojDece]=kom; brojDece++;}

Komponenta Nadji(String Naziv1)
    {   Komponenta pom = super.Nadji(Naziv1);
       if (pom!=null) return pom;
       for(int i=0;i<brojDece; i++)
        {  pom = deca[i].Nadji(Naziv1);
             if (pom!=null)   return pom;
         }
       return null;
    }

  void Prikazi()
    { System.out.println("Deca od komponente "+ this.Naziv + " su:") ;
      for (int i=0;i<brojDece;i++)  {  System.out.println((i+1) + ". dete " + deca[i].vratiNaziv()); }
    }
}
```

СП4: Decorator патерн

Дефиниција

Придружује додатне одговорности (функционалности) до објекта динамички. Decorator патерн обезбеђује флексибилност у избору подкласа које проширују функционалност.

Структура Decorator патерна



Учесници

- **Component**

Дефинише интерфејс за *ConcreteComponent* објекте којима се одговорност додаје динамички.

- **ConcreteComponent**

Дефинише објекат коме ће бити додата одговорност динамички.

- **Decorator**

Чува референцу на *Component* објекат.Дефинише интерфејс који је у складу са интерфејсом *Component*.

- **ConcreteDecorator**

Додаје одговорност до *ConcreteComponent* објекта.

Пример Decorator патерна

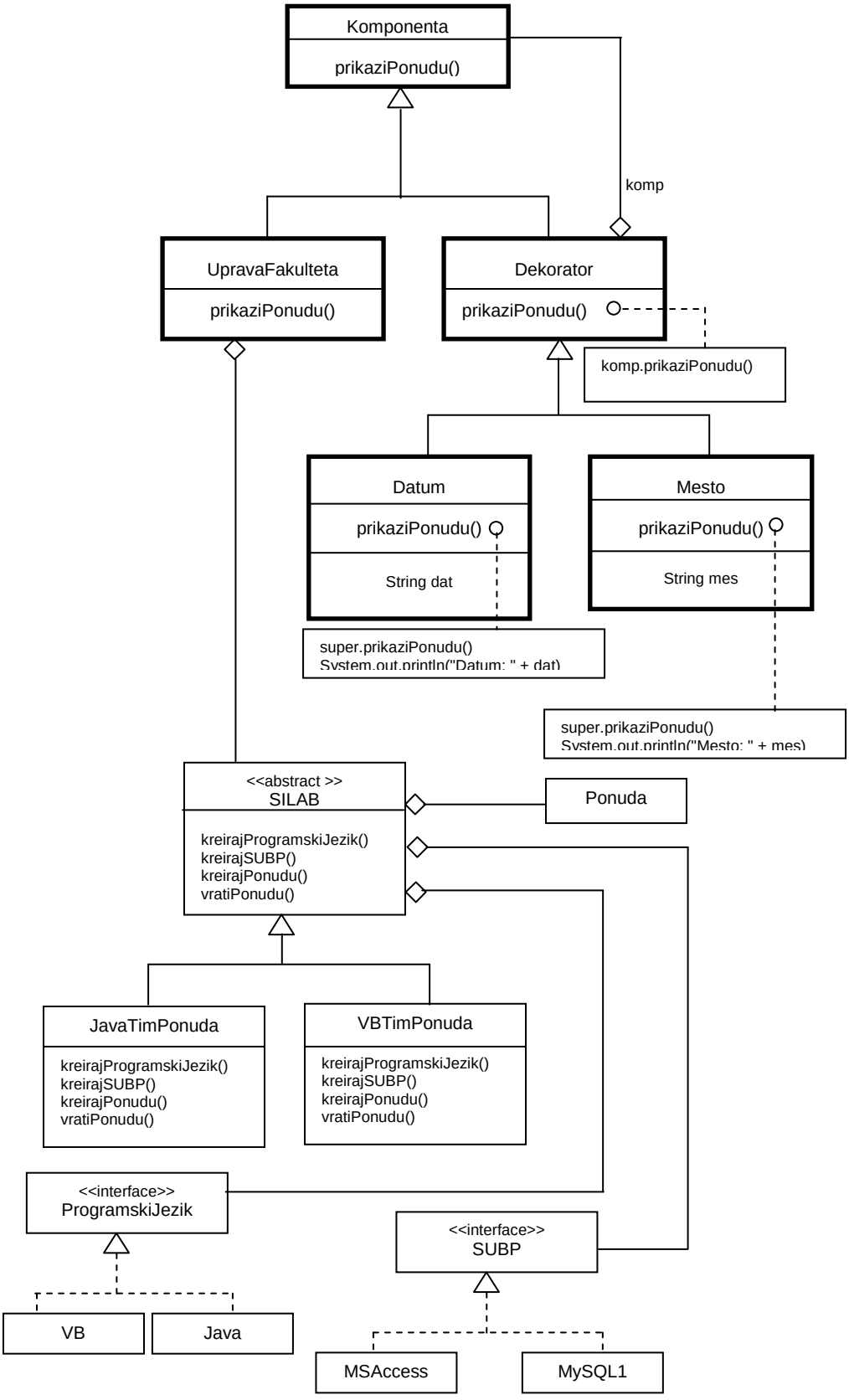
Кориснички захтев PDR1[1]: *Управа Факултета треба да прошири понуду Јава тима са датумом када је издата понуда. Након тога понуду треба проширити са местом где је издата понуда.*

```
// Улога: Дефинише интерфејс за UpravaFakulteta објекат коте се одговорност додаје динамички.

interface Komponenta   // Component
{ void prikaziPonudu();}
// Улога: Чува референцу на Komponenta објекат. Дефинише интерфејс који је у складу са интерфејсом
/ Komponenta.
class  Dekorator implements Komponenta // Decorator
{   Komponenta komp;
    Dekorator(Komponenta komp1) {komp = komp1;}
    public void prikaziPonudu(){komp.prikaziPonudu();}
}
// Улога: Додаје одговорност до UpravaFakulteta објекта.
class Datum extends Dekorator  // Concrete Decorator 1
{   String dat;
    Datum(Komponenta komp1) {super(komp1); dat = new String (”28.08.2014”);}
    public void prikaziPonudu(){super.prikaziPonudu(); System.out.println("Datum: " + dat);}
}
class Mesto extends Dekorator  // Concrete Decorator 2
{   String mes;
    Mesto(Komponenta komp1) {super(komp1);mes=new String(“Beograd”);}
    public void prikaziPonudu(){super.prikaziPonudu(); System.out.println("Mesto: " + mes);}
}
// Улога: Дефинише  објекат коме ће бити додата одговорност  динамички.
class UpravaFakulteta implements Komponenta  // ConcreteComponent
{ SILAB sil; // Builder
  UpravaFakulteta(SILAB sil1){sil = sil1;}
  void Konstruisi()
    { sil.kreirajProgramskiJezik();
      sil.kreirajSUBP();
      sil.kreirajPonudu();
    }
   public static void main(String args[])
     {  UpravaFakulteta uf;
        JavaTimPonuda jat = new JavaTimPonuda();  // ConcreteBuilder1
        uf = new UpravaFakulteta(jat);
        uf.Konstruisi();
        Datum dat = new Datum(uf);
        Mesto mes = new Mesto(dat);
         // Moglo je i ovako da se napise: Mesto mes = new Mesto(new Datum(uf));
         mes.prikaziPonudu();
         // u ovom primeru se prvo pojavljuje mesto pa datum na ponudi
         //  mes = new Mesto(uf);dat = new Datum(mes); dat.prikaziPonudu();
      }
   public void prikaziPonudu(){System.out.println("Ponuda java tima: \n" + sil.vratiPonudu());}
}
abstract class  SILAB
{  ProgramskiJezik  pj;  SUBP subp;
    Ponuda pon;
   abstract void kreirajProgramskiJezik();
   abstract void kreirajSUBP();
   abstract void kreirajPonudu();
   abstract String vratiPonudu();
 }
class Ponuda {String ponuda;}
class JavaTimPonuda extends SILAB
{ JavaTimPonuda() {pon = new Ponuda();}
  public void kreirajProgramskiJezik(){pj = new Java();}
  public void kreirajSUBP() { subp = new MySQL();}
  public void kreirajPonudu() { pon.ponuda = "Programski jezik-" + pj.vratiProgramskiJezik() + "
                                SUBP-" + subp.vratiSUBP();}
  public String vratiPonudu(){return pon.ponuda;}
}
class VBTimPonuda extends SILAB {
  VBTimPonuda(){pon = new Ponuda();}
  public void kreirajProgramskiJezik(){pj = new VB();}
```

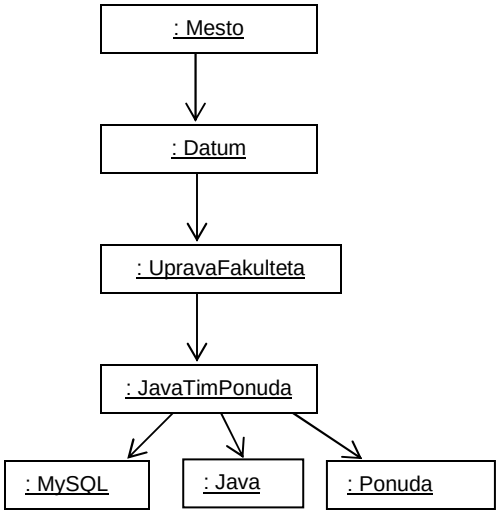
[1] Наведени пример представља проширење примера који је урађен код Builder патерна.

```
  public void kreirajSUBP() {subp = new MSAccess();}
  public void kreirajPonudu() { pon.ponuda = "Programski jezik-" + pj.vratiProgramskiJezik() + "
                                                SUBP-" + subp.vratiSUBP();}
  public String vratiPonudu(){return pon.ponuda;}
}

interface ProgramskiJezik  {String vratiProgramskiJezik();}
class Java implements ProgramskiJezik { public String vratiProgramskiJezik(){return "Java";}}

class VB implements ProgramskiJezik { public String vratiProgramskiJezik(){return "VB";}}
interface SUBP {String vratiSUBP();}
class MySQL implements SUBP { public String vratiSUBP(){return "MySQL";}}
class MSAccess implements SUBP  { public String vratiSUBP(){return "MS Access";}}
```

Дијаграм класа примера PDR1



Објектни дијаграм примера PDR1



Задатак Z – DE1: Додати програм на месту три тачке како би се добиле поруке:
Juce je bilo oblacno vreme.
Danas je lep dan.
Sutra ce biti jos lepsi dan.

```
interface Komponenta
{ void prikazi();}

class  Dekorator implements Komponenta

{  ...

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

class KonkretniDekoratorA extends Dekorator
{  KonkretniDekoratorA(Komponenta komp1) {super(komp1); }
   public void prikazi(){super.prikazi(); System.out.println("Danas je lep dan.");}
}
class KonkretniDekoratorB  extends Dekorator
{  KonkretniDekoratorB(Komponenta komp1) {super(komp1);}

   public void prikazi(){super.prikazi(); ...  }
}

class KonkretnaKomponenta implements Komponenta

{ public void prikazi() { ...  }
}

class Client6
{
    public static void main(String args[])
    {   KonkretnaKomponenta kk = new KonkretnaKomponenta();

        ...

        kdb.prikazi();
    }
}
```

Решење Z – DE1:

```
interface Komponenta
{ void prikazi();}

class  Dekorator implements Komponenta
{  Komponenta komp;
   Dekorator(Komponenta komp1) {komp = komp1;}
   public void prikazi(){komp.prikazi();}
}

class KonkretniDekoratorA extends Dekorator
{  KonkretniDekoratorA(Komponenta komp1) {super(komp1); }
   public void prikazi(){super.prikazi(); System.out.println("Danas je lep dan.");}
}
class KonkretniDekoratorB  extends Dekorator
{  KonkretniDekoratorB(Komponenta komp1) {super(komp1);}
   public void prikazi(){super.prikazi(); System.out.println("Sutra ce biti jos lepsi dan");}
}

class KonkretnaKomponenta implements Komponenta
{ public void prikazi() { System.out.println("Juce je bilo oblacno vreme.");}
}

class Client6
{
    public static void main(String args[])
    {   KonkretnaKomponenta kk = new KonkretnaKomponenta();
        KonkretniDekoratorA kda = new KonkretniDekoratorA(kk);
        KonkretniDekoratorB kdb = new KonkretniDekoratorB(kda);
        kdb.prikazi();
    }

}
```