OBLIKOVNI OBRASCI U PROGRAMIRANJU
SIMPTOMI PROGRAMA KOJI PROPADA, URUŠAVA SE, ILI JE NAPROSTO NEPRIKLADNO ORGANIZIRAN
→ KRUTOST: TEŠKO NADOGRAĐIVANJE, PROMJENE REZULTIRAJU DOMINO EFEKTOM ← APSTRAKCIJA ENKAPSULACIJA
→ KRHKOST: LAKO UNOŠENJE SUPTILNIH POGREŠAKA (USLIJED PONAVLJANJA)
→ NEPOKRETNOST: TEŠKO VIŠESTRUKO KORIŠTENJE (PRETJERANA MEĐUOVISNOST)
→ VISKOZNOST: SKLONOST K SLABLJENJU INTEGRITETA PROGRAMA (USLIJED PRETJERANE SLOŽENOSTI, PONAVLJANJA, ...)
• VISKOZNOST PROGRAMSKE ORGANIZACIJE: NADOGRADNJE KOJE ČUVAJU INTEGRITET ZAHTIJEVAJU PUNO MANUALNOG RADA ILI NISU OČITE
• VISKOZNOST RAZVOJNOG PROCESA: SPORA, NEEFIKASNA RAZVOJNA OKOLINA

"SPAGHETTI CODE" ← PRETJERANA MEĐUOVISNOST

"SWISS ARMY KNIFE" ← PRETJERANA SLOŽENOST

"REINVENT THE WHEEL" ← PONAVLJANJE UMJESTO PONOVNOG KORIŠTENJA

LOGIČKO OBLIKOVANJE - ELEMENTI PROGRAMSKOG JEZIKA (MODULI: RAZREDI I FUNKCIJE)
FIZIČKO OBLIKOVANJE - RASPODJELA FUNKCIONALNOSTI PO DATOTEKAMA (KOMPONENTE: SUČELJA I IMPLEMENTACIJE)
DINAMIČKI POLIMORFIZAM - ODREDIŠTE POSTAJE POZNATO TEK U TRENUTKU IZVOĐENJA (JAVA, C++, PYTHON)
STATIČKI POLIMORFIZAM - ISPITNA KOMPONENTA PARAMETRIZIRA PREDLOŽAK
GENERIČKO PROGRAMIRANJE - PROŠIRENA GRAMATIKA OMOGUĆAVA PARAMETRIZIRANJE FUNKCIJA I RAZREDA
ORTOGONALNOST - NEMA MEĐUOVISNOSTI - ALGORITAMA I SPREMNIKA (STL)

GENERICI VS. VIRTUALNE FUNKCIJE
+ MANJA VREMENSKA SLOŽENOST
+ MANJA PROSTORNA SLOŽENOST
- VEĆI IZVRŠNI KOD
- NAKON PREVOĐENJA FLEKSIBILNOST NESTAJE

OBLIKOVANJE TEMELJENO NA UGOVORU (DBC)
→ PREDUVJETI ZA PRIMJENJIVOST KOMPONENTE (PREDUVJETI REGULIRAJU OBAVEZE KLIJENTA PREMA PRUŽATELJU)
→ INVARIJANTE ILI INTERNI POKAZATELJI INTEGRITETA KOMPONENTE
→ POSTUVJETI ILI GARANCIJA ISPRAVNOSTI REZULTATA (POSTUVJETI REGULIRAJU OBAVEZE PRUŽATELJA PREMA KLIJENTU)

LOGIČKA NAČELA:
→ NAČELO NADOGRADNJE BEZ PROMJENE (NBP)
• DOBIVAMO KOMPONENTE KOJE SU OTVORENE ZA NADOGRADNJU, ALI ZATVORENE ZA PROMJENE (OPEN-CLOSED PRINCIPLE)
• STARI KOD RADI S NOVIM KODOM
• PRIMJENOM NASLJEĐIVANJA:
- NASLJEĐIVANJE IMPLEMENTACIJE
+ NOVI RAZREDI POZIVAJU TEMELJNU IMPLEMENTACIJU NASLJEĐIVANJEM STAROG RAZREDA (POSTOJEĆI KLIJENTI NEMOGU DOĆI DO NOVE FUNKCIONALNOSTI)
+ STARI KOD POZIVA NOVI
- NASLJEĐIVANJE SUČELJA
+ KLIJENTI TRANSPARENTNO PRISTUPAJU NOVOJ IMPLEMENTACIJI POLIMORFNIM POZIVOM PREKO STAROG SUČELJA
+ STARI KOD POZIVA NOVI KOD
• GENERIČKIM PROGRAMIRANJEM
• KONCEPTI KOJI POSPJEŠUJU NBP: ENKAPSULACIJA, VIRTUALNE FUNKCIJE, APSTRAKTNI RAZREDI, GENERIČKI PROGRAMI
→ NAČELO NADOMJESTIVOSTI OSNOVNIH RAZREDA (LNS) (LISKOVINO NAČELO SUPSTITUCIJE)
• NASLJEĐIVANJE MODELIRA RELACIJU JE-VRSTA
• IZVEDENI RAZREDI TREBAJU POŠTIVATI UGOVORE OSNOVNOG RAZREDA:
- PREDUVJETI IZVEDENIH METODA (NA PARAMETRE ILI NA STANJE OBJEKTA) MORAJU BITI JEDNAKI ONIMA U OSNOVNOM RAZREDU ILI OSLABLJENI
- POSTUVJETI IZVEDENIH METODA MORAJU BITI ISTI ILI POSTROŽENI
- IZVEDENI RAZRED KRŠI LNS AKO NEKI KLIJENT KOJI KONKRETNO RADI S OSNOVNIM RAZREDOM NE MOŽE RADITI S IZVEDENIM RAZREDOM
• KRŠENJE LNS MOŽE SE POPRAVITI NA 3 NAČINA:
1. SMANJITI ODGOVORNOST OSNOVNOG RAZREDA (POJAČATI PREDUVJETE, OSLABITI POSTUVJETE)
2. POVEĆATI ODGOVORNOST IZVEDENOG RAZREDA (OSLABITI PREDUVJETE, POJAČATI POSTUVJETE)
3. ODUSTATI OD IZRAVNOG RODITELJSKOG ODNOSA DVAJU RAZREDA
• ZNAČAJKE JEZIKA S IMPLICITNIM TIPIZIRANJEM (DUCK TYPING)
- TIP ARGUMENATA FUNKCIJA I ATRIBUTA RAZ[illegible]
- NADOMJESTIVE TIPOVE MOŽEMO GRADITI I BEZ NASLJEĐIVANJA
• LNS PREDSTAVLJA RECEPT ZA KORIŠTENJE NASLJEĐIVANJA

→ NAČELO INVERZIJE OVISNOSTI (NIO)
- NADOGRADIVOST SE MOŽE OSTVARITI USMJERAVANJEM OVISNOSTI PREMA APSTRAKTNIM SUČELJIMA
  - OVISNOSTI IDU PREMA APSTRAKCIJAMA KOJE IMAJU MALO RAZLOGA ZA PROMJENU JER NEMAJU IMPLEMENTACIJU
  - NE OVISIMO VIŠE O NEPOSTOJANIM MODULIMA S DETALJNIM IMPLEMENTACIJAMA : KAŽEMO DA JE TA OVISNOST INVERTIRANA
  - PUTEVI OVISNOSTI SU KRAĆI
  - PROBLEM: STVARANJE OBJEKATA KONKRETNIH RAZREDA ← INJEKCIJA OVISNOSTI, OBRAZAC TVORNICE
- INJEKCIJA OVISNOSTI : UMJESTO HARDKODIRANOG KOMPLICIRANOG KONKRETNOG ČLANA UVODI SE KONFIGURIRANJE PREKO REFERENCE NA OSNOVNI RAZRED
  - OVISNOST USLIJED STVARANJA IZVLAČI SE U ZASEBNU KOMPONENTU
  - MOGUĆNOST NEOVISNOG ISPITIVANJA

→ NAČELO JEDINSTVENE ODGOVORNOSTI (NJO)
- PROGRAMSKI MODULI MORAJU IMATI SAMO JEDNU ODGOVORNOST
  - AKO MODUL IMA VIŠE ODGOVORNOSTI, MEĐU NJIMA SE JAVLJAJU NEPRIRODNE MEĐUOVISNOSTI (KRHKOST, NEPOKRETNOST)
- OBLIKOVATI ORTOGONALAN SUSTAV U KOJEM RAZDIOBA POSLOVA ODGOVARA INTRINSIČNOJ STRUKTURI PROBLEMA (SVAKA PORODICA KOMPONENATA MODELIRA SVOJU OS PROMJENE)
- ODGOVORNOST JE KVANT FUNKCIONALNOSTI IZ DOMENE APLIKACIJE (SVAKA ODGOVORNOST JE UJEDNO I RAZLOG ZA PROMJENU ~~BEBA~~ MODULA)
- KADA SMO SIGURNI DA MODULI NEMAJU JEDINSTVENU PODSUSTAV TREBA PREKROJITI
- ORTOGONALNOST → MODULI RAZLIČITIH PORODICA MOGU SE PROIZVOLJNO KOMBINIRATI (JEDINSTVENE ODGOVORNOSTI ↔ ORTOGONALNI SUSTAVI)
  - AKO SE ISTA FUNKCIONALNOST NALAZI NA VIŠE MJESTA, SUSTAV NIJE ORTOGONALAN
- SMANJIVANJE NEPOTREBNE MEĐUOVISNOSTI TE POTICANJE ODRŽIVE EVOLUCIJE

→ NAČELO IZDVAJANJA SUČELJA (NIS)
- NAČELO SUBERIRA DA NEKOHERENTNIM KONCEPTIMA (AKO IH BAŠ MORAMO IMATI) PRISTUPATI PREKO IZDVOJENIH SUČELJA
  (NIJE DOBRO PRIMORAVATI KLIJENTE NA OVISNOST O SUČELJU KOJEG NE KORISTE) KLIJENTI TREBAJU
- IZDVOJENA SUČELJA PRIMJENJUJEMO KOD SLOŽENIH KONCEPATA KOJI SE KORISTE NA VIŠE ORTOGONALNIH NAČINA (RECEPT ZA VIŠESTRUKO NASLJEĐIVANJE)

ELEMENTARNA NAČELA :

→ KEEP IT SIMPLE STUPID (KISS)
(YOU AIN'T GONNA NEED IT – YAGNI)

→ DO NOT REPEAT YOURSELF (DRY)
(THE REFACTOR RULE OF THREE)

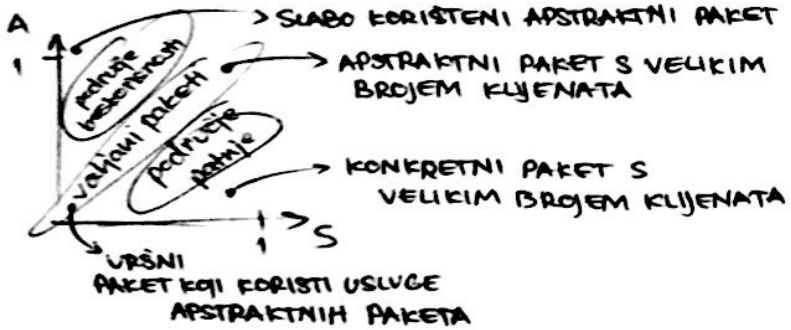
NADOGRADIVOST, RAZUMLJIVOST I FLEKSIBILNOST POSPJEŠUJU SE LOGIČKIM NAČELIMA

FIZIČKA NAČELA
- LAKO ISPITIVANJE
  - NAJZGODNIJE PROVODITI NAD DATOTEKAMA IZVORNOG KODA
  - OBIČNO TEŽIMO TESTIRATI N KOMPONENTI ZASEBNO, A NE SVE MOGUĆE INTERAKCIJE : INKREMENTALNO VS. SVEOBUHVATNO (BIG BANG) TESTIRANJE
  - POSEBNA POGODNOST KOD PROGRAMSKOG OBLIKOVANJA : TESTIRANJE U IZOLACIJI
  - U KONTEKSTU TESTIRANJA I FIZIČKE ORGANIZACIJE, KLJUČNA RELACIJA JE OVISNOST
  - AKO JE OVISNOST ACIKLIČKA → KOMPONENTAMA MOŽEMO DODIJELITI RAZINE (RAZINE DEFINIRAJU REDOSLIJED INKREMENTALNOG TESTIRANJA KOMPONENATA)
- POVOLJNO : PLITKA I NEPOVEZANA STRUKTURA OVISNOSTI
- NEPOVOLJNO : DUBOKA ILI MONOLITNA STRUKTURA OVISNOSTI TE CIKLUSI
- CIKLUSI OMETAJU ISPITIVANJE, VIŠESTRUKO KORIŠTENJE I RAZUMIJEVANJE
  - EKSTREM: KOMPONENTA NA DNU HIJERARHIJE OVISI O VRŠNOJ KOMPONENTI
  - RJEŠENJE ZA PREKIDANJE CIKLUSA UVIJEK POSTOJI, ALI OPĆENITOG RECEPTA NEMA
- POTREBA ZA VEĆIM OBLIKOVNIM JEDINICAMA KOJE SE ZAJEDNO RAZVIJAJU I KORISTE : TAKVE JEDINICE NAZIVAMO PAKETIMA

- KAKO GRUPIRATI KOMPONENTE U PAKETE (NAČELA KOHERENTNOSTI)
  + GRUPIRANJE PREMA KORELACIJI KORIŠTENJA
  + GRUPIRANJE PREMA ZAJEDNIČKOM IZDAVANJU
    * GRUPIRAMO KOMPONENTE KOJE SE ZAJEDNO IZDAJU I ODRŽAVAJU
    * INTERAKCIJA IZMEĐU OBLIKOVNIH (KORELACIJA KORISTENJA) I POSLOVNIH KRITERIJA (IZDAVANJE)
  + GRUPIRANJE PREMA ODGOVORNOSTI (OSJETLJIVOST NA PROMJENE)
    * KOMPONENTE OSJETLJIVE NA PROMJENE IZ ISTOG SKUPA
    * SAMO JEDAN RAZLOG ZA IZDANJE NOVOG PAKETA
  + NIJE DOBRO DA KOMPONENTE MODELIRAJU JEDINSTVENU OS PROMJENE SLOŽENOG SUSTAVA (TJ. DA IMAJU JEDINSTVENU ODGOVORNOST)
- KAKO VALJA UREDITI ODNOSE MEĐU PAKETIMA (NAČELA STABILNOSTI)
  + RAZDIOBA RAZREDA PO PAKETIMA MORA PRIGUŠIVATI PROMJENE
  + NAČELO ACIKLIČKE OVISNOSTI
  + NAČELO STABILNE OVISNOSTI – USMJERAVANJE OVISNOSTI PREMA NESTALNIJIM PAKETIMA
  + NAČELO PRIMJERENE APSTRAKCIJE – PAKET TREBA [illegible]
- STABILNOST = OTPORNOST NA PROMJENE

+ POZITIVNE I NEGATIVNE KONOTACIJE : POUZDANOST VS. INERTNOST
+ INERTNOST JE ODREĐENA MEĐUOVISNOSTIMA
+ METRIKA STABILNOSTI PAKETA $S = \frac{O_U}{O_U + O_I} \in \langle 0,1 \rangle$
  * O_U ← ODGOVORNOST : BROJ VANJSKIH KOMPONENTI KOJE OVISE O PAKETU
  * O_I ← NEPOSTOJANOST : BROJ VANJSKIH KOMPONENTI O KOJIMA PAKET OVISI
+ STABILNOST JE U KONTRADIKCIJI S IZLAŽNIM OVISNOSTIMA : STABILNI PAKETI MOGU BITI ILI NADOGRADIVI BEZ PROMJENE ILI RELATIVNO JEDNOSTAVNI

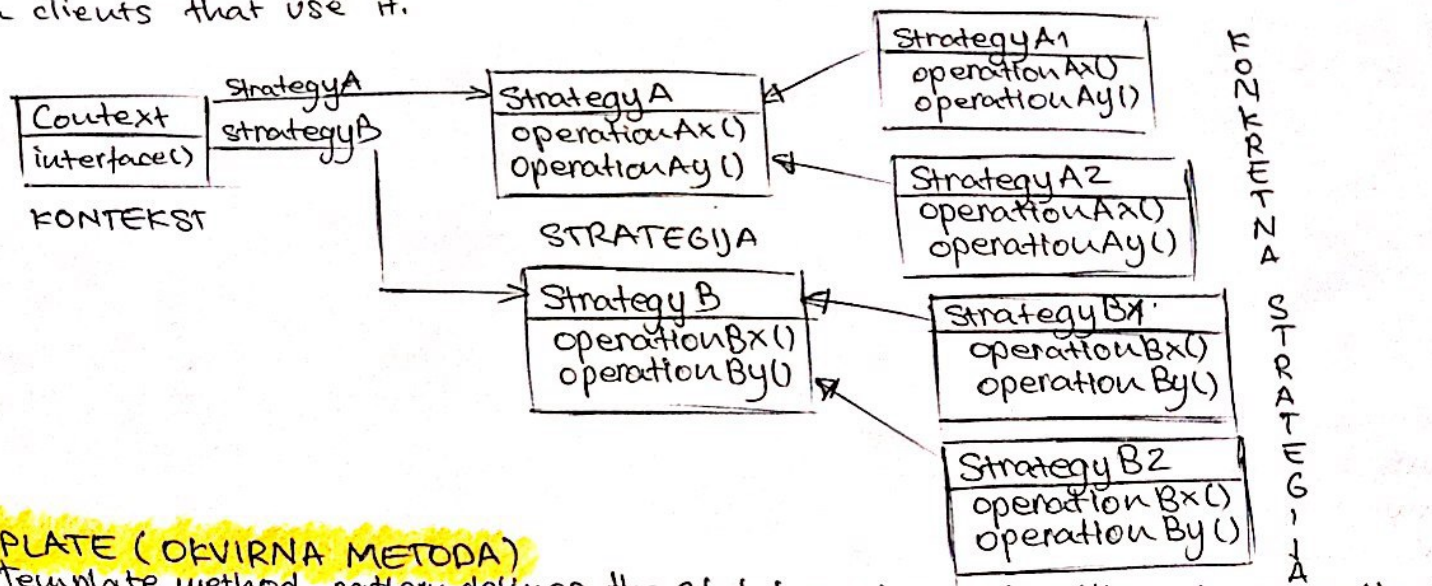
- METRIKA APSTRAKTNOSTI PAKETA $A = \frac{N_A}{N_C} \in \langle 0,1 \rangle$
  + N_A ← BROJ APSTRAKTNIH RAZREDA PAKETA
  + N_C ← UKUPNI BROJ RAZREDA PAKETA
  + RAČUNAJU SE SAMO RAZREDI O KOJIMA OVISE VANJSKI PAKETI
- S >> A ... PODRUČJE BESKORISNOSTI (MOŽE BITI OK, AKO JE PAKET ZREO)
  A >> S ... PODRUČJE PATNJE (SIMPTOM PRETJERANOG OBLIKOVANJA)
  A = S ... DOBRO OBLIKOVANI PAKETI (IDEALNO S=A=0 ILI S=A=1)



LOGIČKI I FIZIČKI CILJEVI OBLIKOVANJA SU KOMPATIBILNI

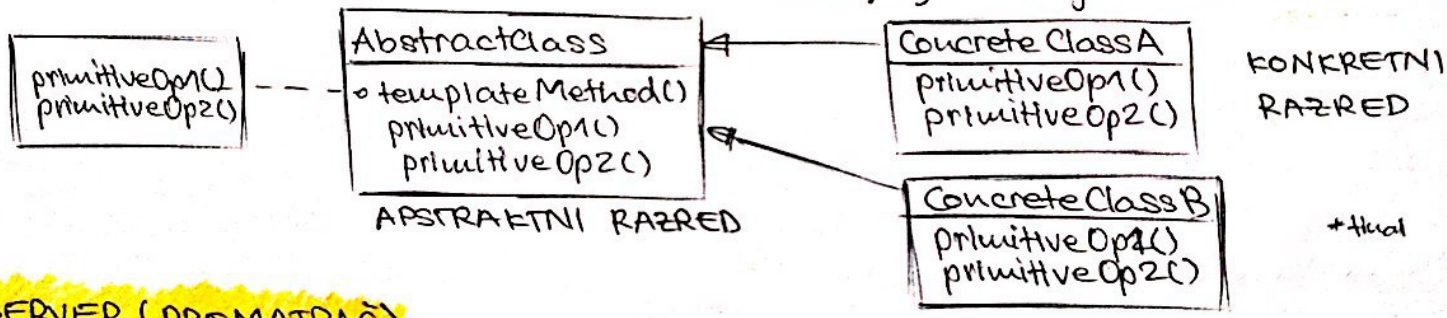
STRATEGY (STRATEGIJA)

The strategy pattern defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it.



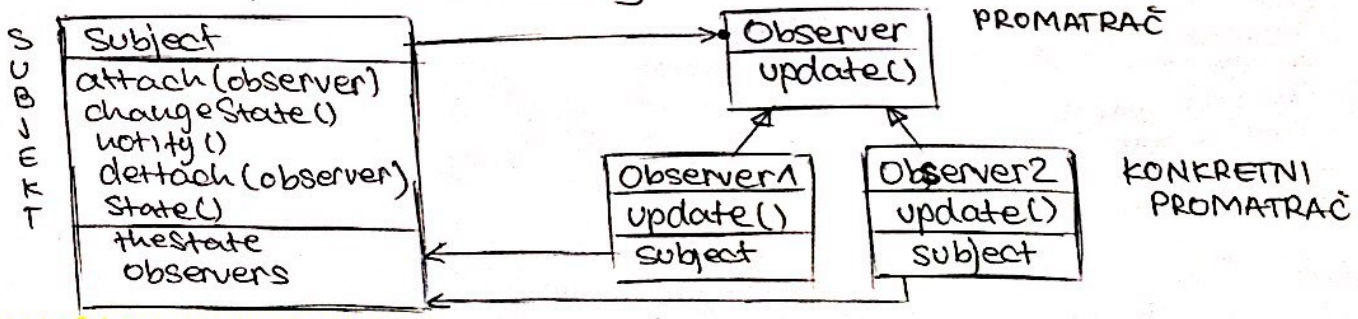
TEMPLATE (OKVIRNA METODA)

The Template method pattern defines the skeleton of an algorithm in a method, deferring some steps to subclasses. Template method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.



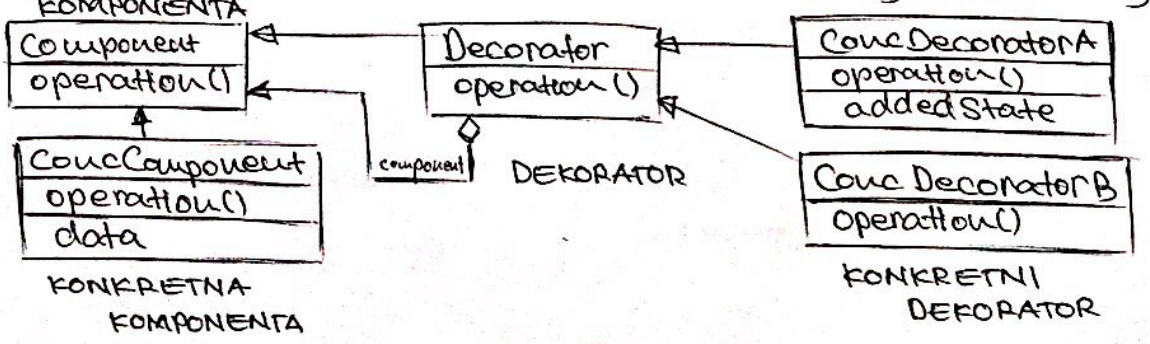
OBSERVER (PROMATRAČ)

The Observer pattern defines a one-to-many ~~relationship~~ dependency between objects so that when one object state changes, all of its dependents are notified and updated automatically.



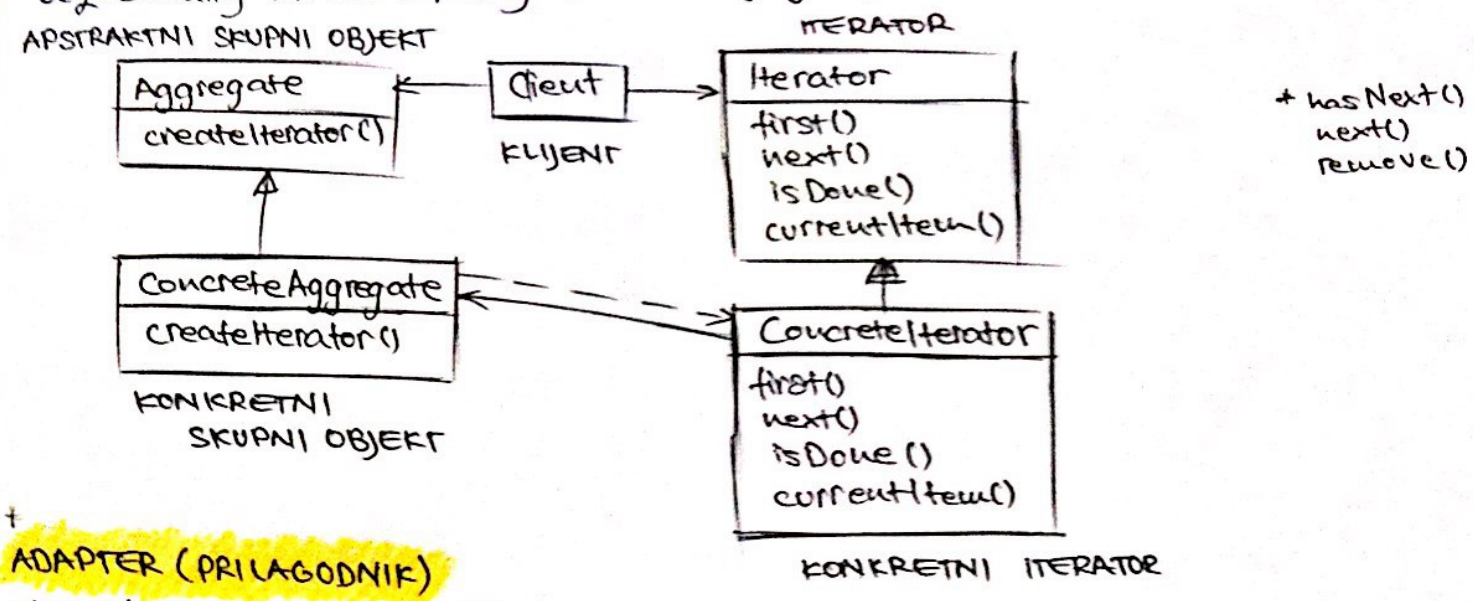
DECORATOR (DEKORATOR)

The Decorator pattern attaches additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.



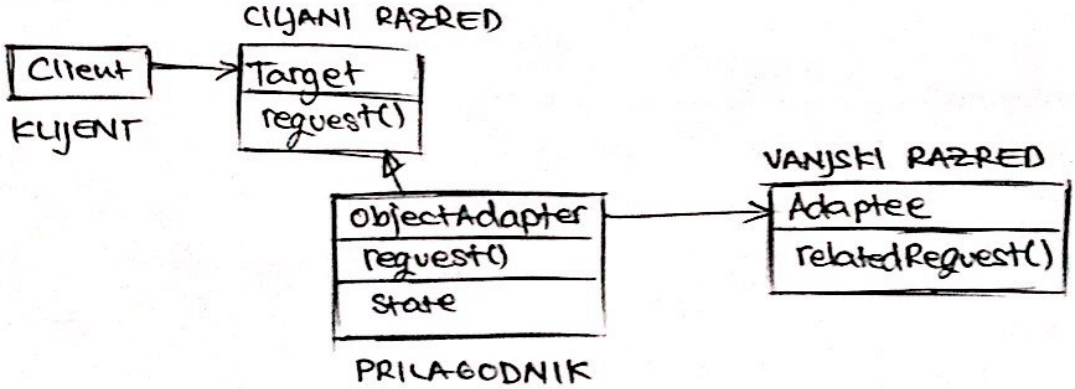
ITERATOR (ITERATOR)

The iterator pattern provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation.



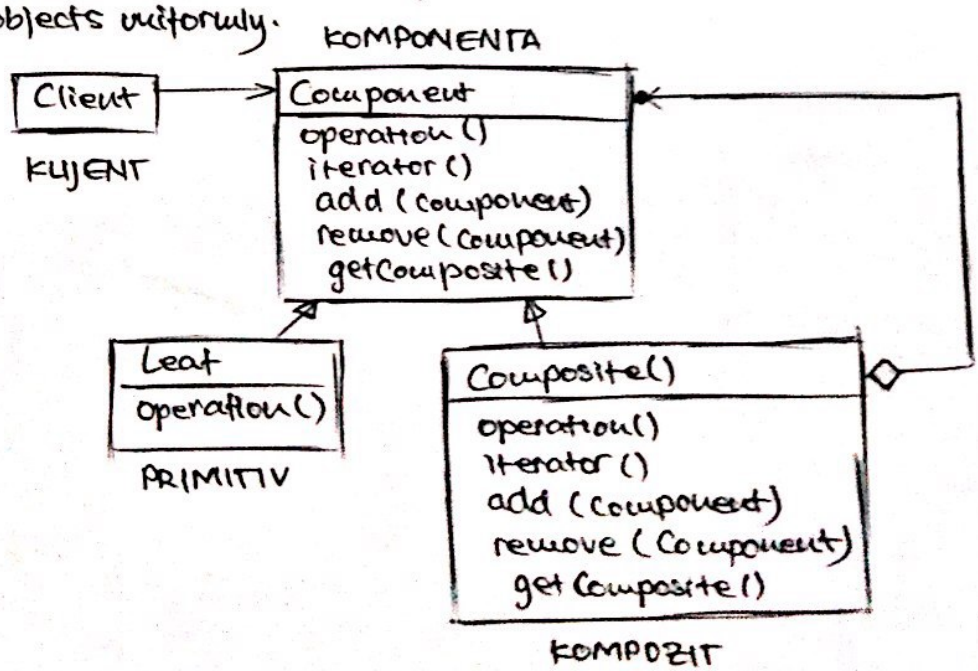
ADAPTER (PRILAGODNIK)

The adapter pattern converts the interface of a class into another interface the client expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.



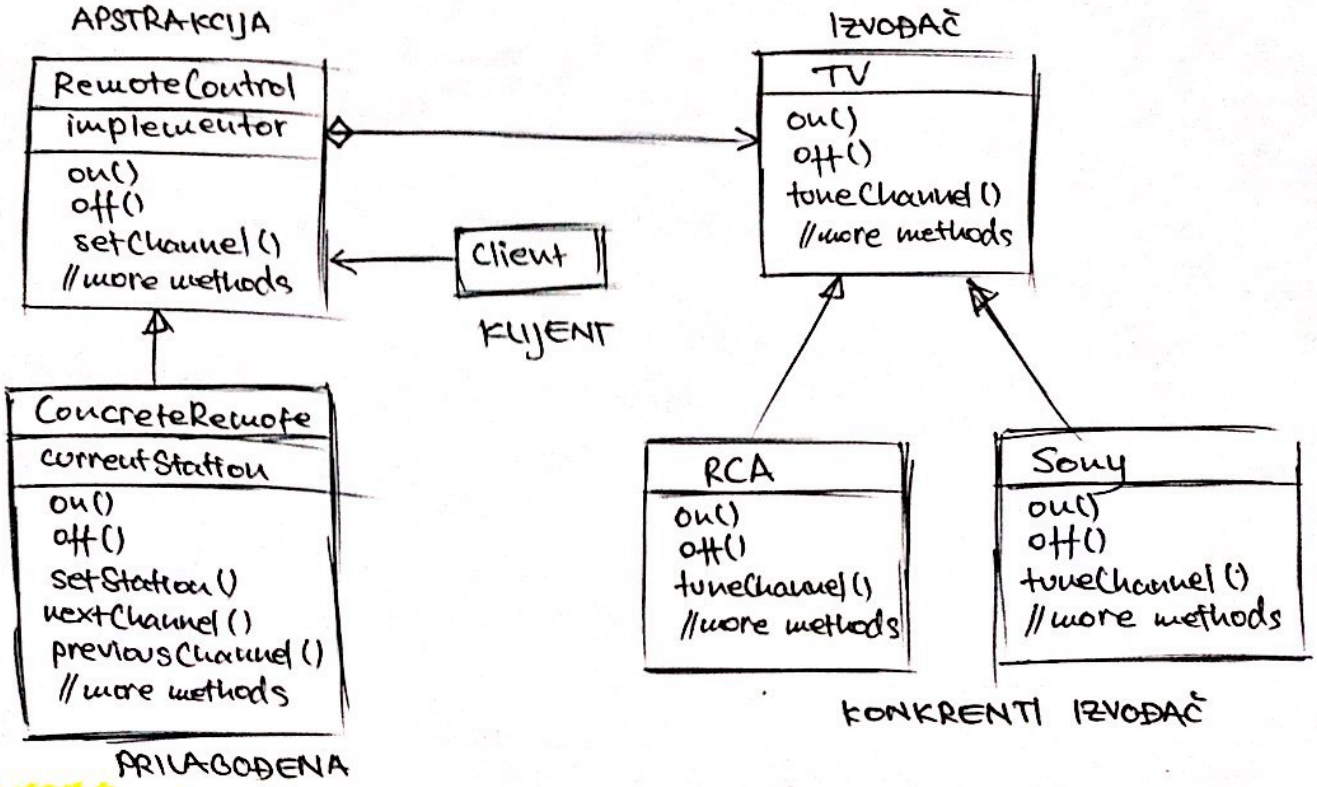
COMPOSITE (KOMPOZIT)

The composite pattern allows you to compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.



BRIDGE (MOST)

Use the bridge pattern to vary not only your implementations, but also your abstraction.

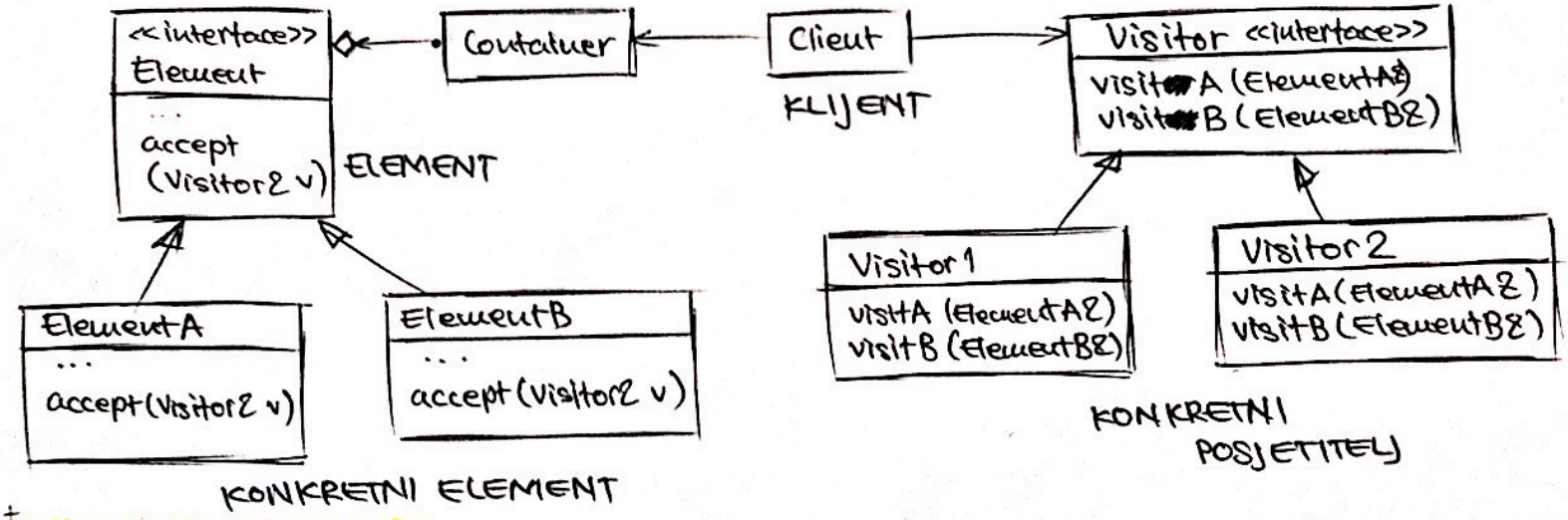


VISITOR (POSJETITELJ)

APSTRAKCIJA

Use the visitor pattern when you want to add capabilities to a composite of objects and encapsulation is not important.

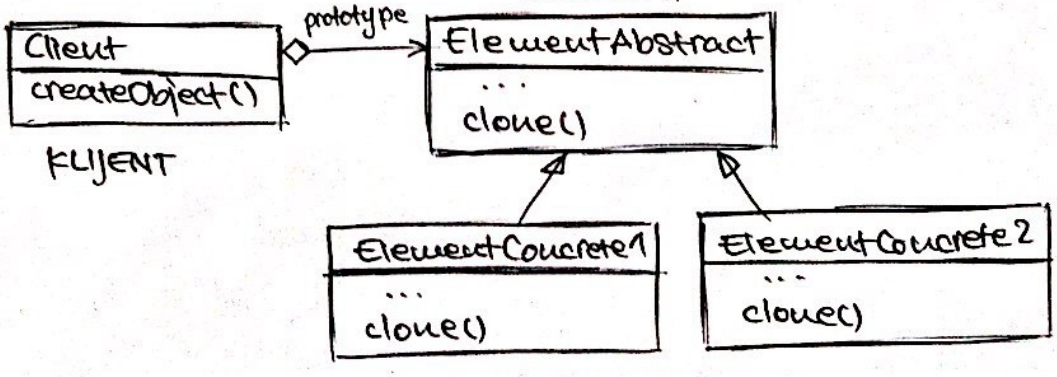
POSJETITELJ



PROTOTYPE (PROTOTIP)

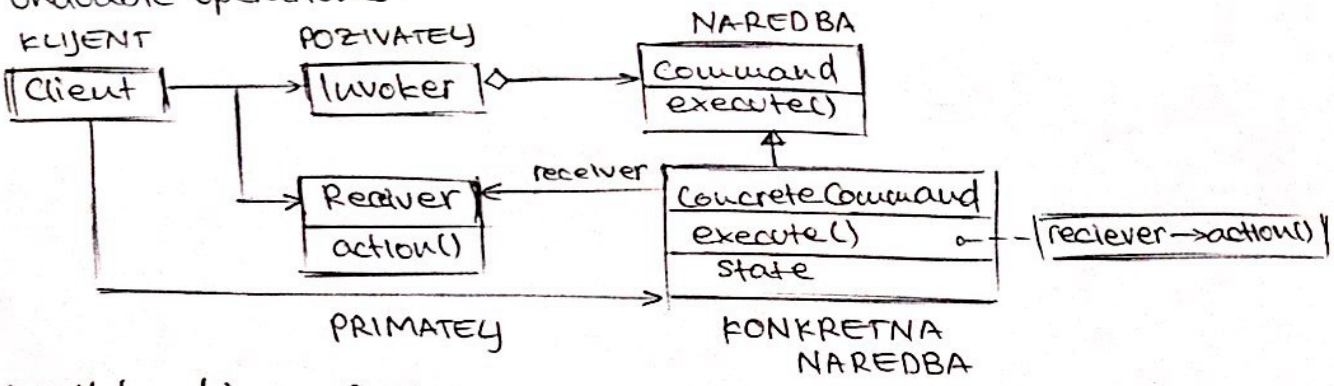
Use the prototype pattern when creating an instance of a given class is either expensive or complicated.

APSTRAKTNI ELEMENT

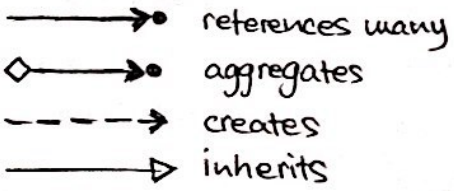


COMMAND (NAREDBA)

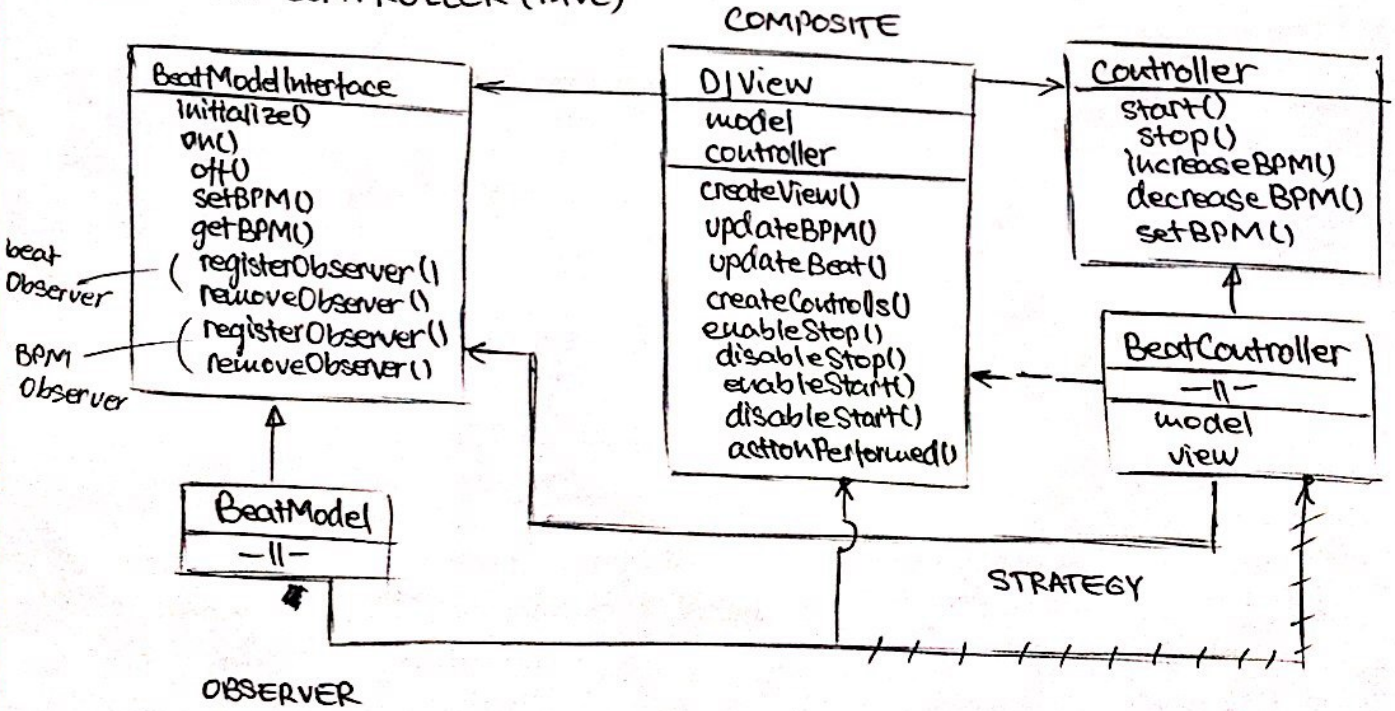
The command pattern encapsulates a request as an object, thereby letting you parameterize other objects with different requests, queue, or log requests and support undoable operations.



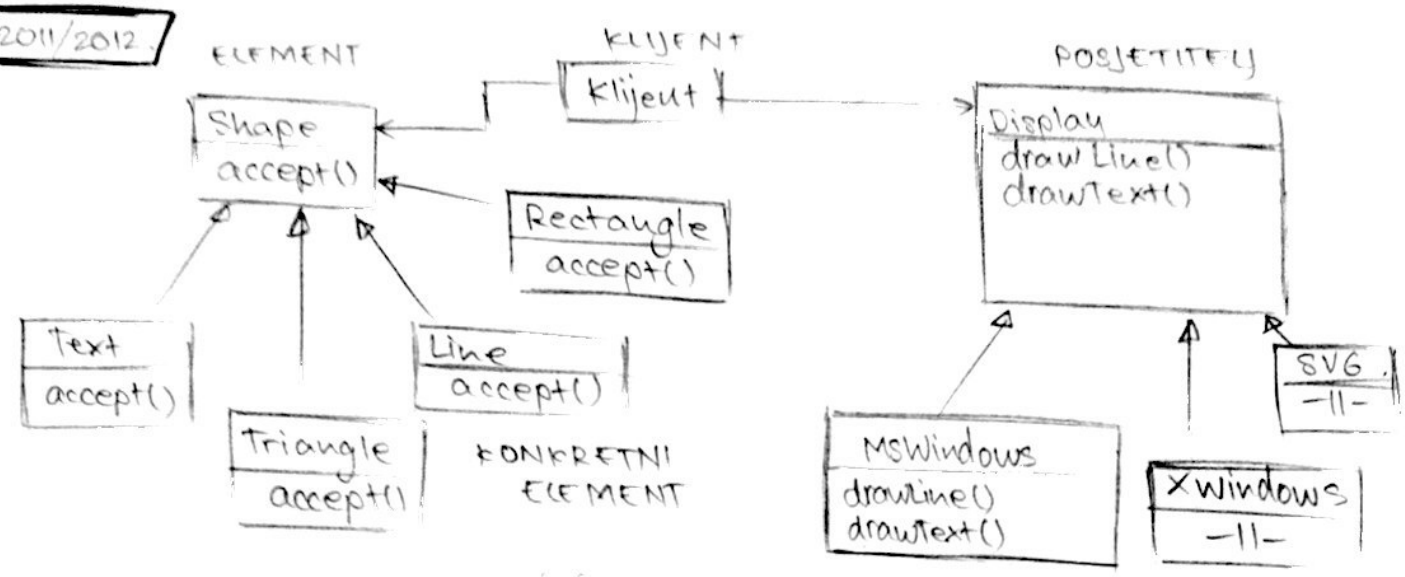
NBP - Načelo Nadogradnje Bez Promjene
LNS - Liskovino Načelo Supstitucije
NIO - Načelo Inverzije Ovisnosti
NJO - Načelo Jedinstvene Odgovornosti
NIS - Načelo Izdvajanja Sučelja



MODEL-VIEW-CONTROLLER (MVC)



21. 2011/2012.



POSJETITELJ

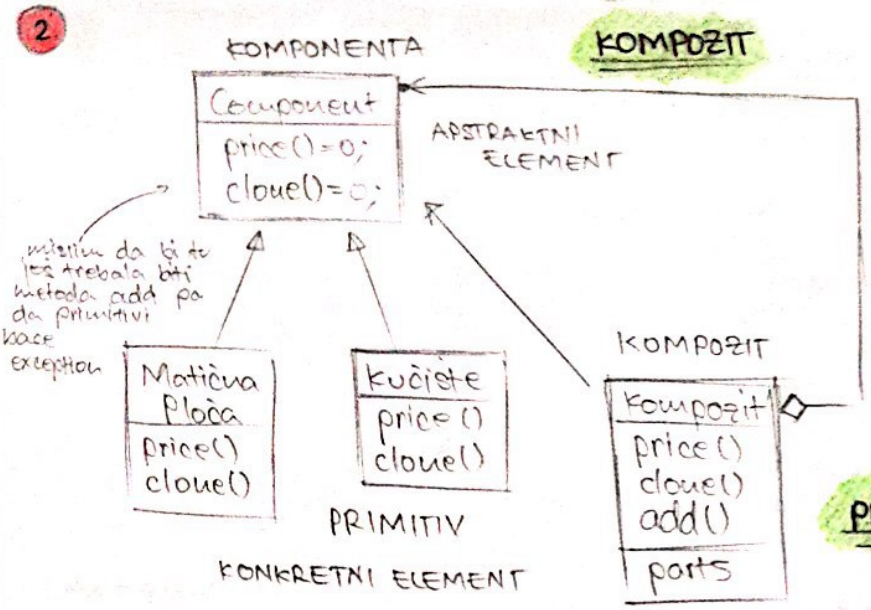
```
public interface Shape {
        public void accept (Display d);
        }
public class Text implements Shape {
        public void accept (Display d) {
                d.drawText(text)
                }
        }
public class Triangle implements Shape {
        Point p1;
        Point p2;
        Point p3;
        public Triangle ( Point p1, Point p2, Point p3) {
                this.p1 = p1;
                this.p2 = p2;
                this.p3 = p3;
                }
        public void accept ( Display d) {
                d.drawLine ( p1, p2);
                d.drawLine ( p2, p3);
                d.drawLine ( p1, p3);
                }
        }
public interface Display {
        public void drawLine ( Point p1, Point p2);
        public void drawText ( String text);
        }
public class MSWindows implements Display {
        public void drawLine ( Point p1, Point p2) {
                // crta liniju pod MS Windowsima
                }
        public void drawText (String text) {
                // ispisuje tekst pod MS Windowsima
                }
        }
```

```
String text;
public Text ( String text) {
        this.text = text;
        }
```

```
public class Demo {
        public static void
        main(String[] args) {
                Shape[] shapes =
                { new Text (), new
                  Triangle () };
                for (Shape s : shapes)
                        s.accept (d)
        }
```

```
Display d = new MSWindows();
```

2



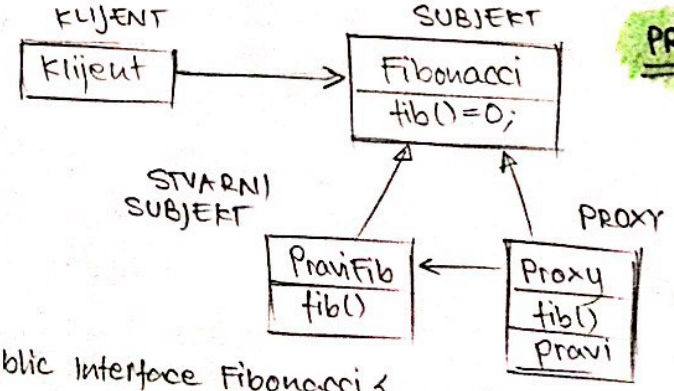
PROTOTIP

```
public interface Component{
    public int price();
    public Component cloue();
}
public class MatičnaPloča implements
    Component{
      int priceTag;
      public MatičnaPloča
       (int price){
          priceTag=price;
       }
      public int price(){
          return priceTag;
       }
      public Component cloue(){
          return new Matična
              Ploča();
       }
}
```

```
public class Kompozit implements Component{
    Component[] parts;
    public Kompozit ( Component[] parts){
        this.parts.addAll (parts);
    }
    public void add (Component comp){
        parts.add(comp);
    public int price(){          int price=0;
        for( Component c : parts){
            price += c.price()
        }
        return price;
    public void done (){
        Kompozit k=new Kompozit();
        for ( Component c : parts){
            k.add (c.cloue());
        }
        return k;
    }
}
```

```
public class Demo{
    public static void main
    (String[] args){
      Component k1 = new
      MatičnaPloča (1000);
      Component k2 = new
       Kućište (500)
      Component[2] parts = {
          k1, k2 };
      Component računalo =
      new Kompozit(parts);
       Component kopija =
        računalo.cloue();
       kopija.price();
    }
}
```

3



PROXY

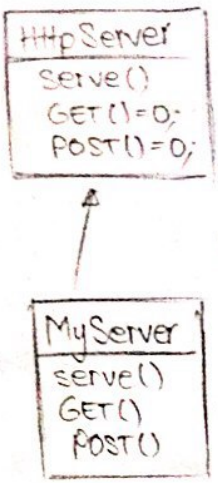
```
tib(0)  1
tib(1)  1
tib(2)  1+1+1=3
tib(3)  1+3+1=5
tib(4)  1+5+3=9
```

```
public interface Fibonacci{
    public int tib(int n);
}
public class Proxy implements Fibonacci{
    PraviFib izracun;      HashMap<Integer, Integer> cache;
    public Proxy(){
        izracun = new PraviFib();
        cache = new HashMap<Integer, Integer>();
}
```

```
public int tib(int n){
    if( cache.get(n)!=null){
        return cache.get(n);
    }
    else{
        int pom=izracun.tib(n)
        cache.put(n, pom);
        return pom;
    }
}
public class PraviFib
implements Fibonacci{
  public int fib(int n){
     if(n<2) return n;
     return fib(n-1)+
          fib(n-2);
  }
}
```

2011/2012.

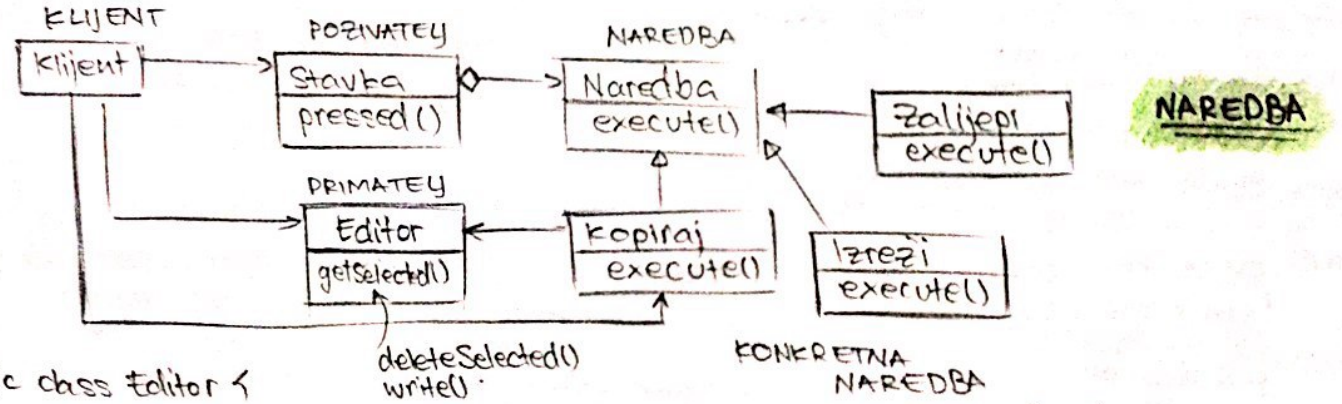
4




```
public class Demo{
 public static void main
  (String[] args){
   HttpServer server =
    new MyServer();
     server.serve();
  }
}
```

```
public abstract class HttpServer{
    public final void serve(){
        GET()
        POST()
    }
    public abstract void GET();
    public abstract void POST();
}
public class MyServer extends HttpServer{
    public void GET(){
        // obrađuje GET zahtjeve
    }
    public void POST(){
        // obrađuje POST zahtjeve
    }
```

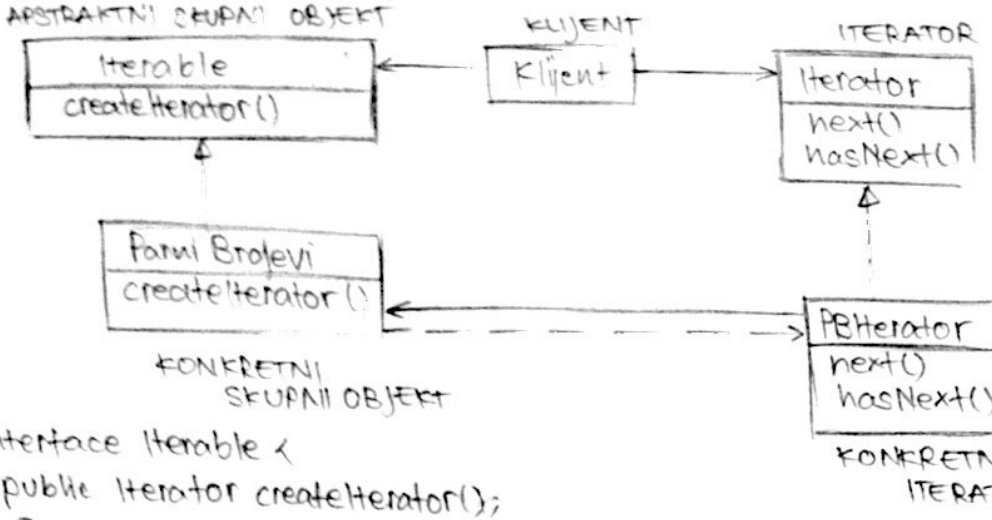
5.




```
public class Editor {
    String tekst;
    public String getSelected(){
        return tekst;
    }
    // ... + metode za selektiranje
}
```

```
publica class Stavka {
    Naredba cmd;
    public Stavka(Naredba cmd) {
        this.cmd = cmd;
    }
    public void pressed(){
        cmd.execute();
    }
}
```

```
public interface Naredba { String tekst;
    public void execute();
}
public class Kopiraj implements Naredba{
    Editor editor;
    public Kopiraj(Editor editor){
        this.editor = editor;
    }
    public void execute(){
        tekst = editor.getSelected();
    }
    // ... + get metoda za tekst
}
public class Klijent {
    public static void main(String[] args){
        Editor editor = new Editor();
        Naredba kopiraj = new Kopiraj(editor);
        Stavka stavka = new Stavka
            (kopiraj);
        stavka.pressed()
```



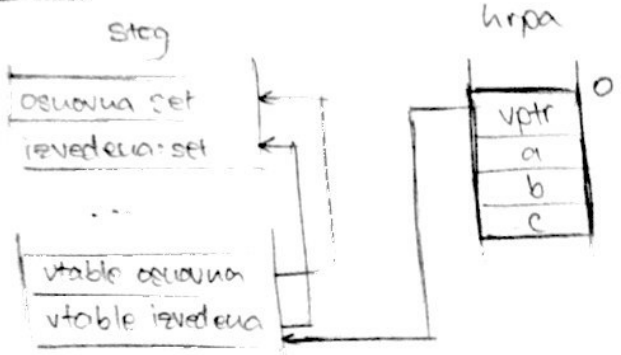
ITERATOR

```
public interface Iterable {
    public Iterator createIterator();
}

public class ParniBrojevi implements Iterable {
    public Iterator createIterator() {
        return new PBIterator(this);
    }
}

public interface Iterator {
    public int next();
    public bool hasNext();
}

public class PBIterator implements Iterator {
    ParniBrojevi kolekcija;
    int position;
    public PBIterator(ParniBrojevi kolekcija) {
        this.kolekcija = kolekcija;
    }
    public bool hasNext() {
        if (position < kolekcija.n - 1)
            return true;
        return false;
    }
    public int next() {
        if (hasNext()) {
            int result = kolekcija.first + 2 * position;
            position++;
            return result;
        }
    }
}

public class Klijent {
    public static void main(String[] args) {
        Iterable kolekcija = new ParniBrojevi(2, 11);
        Iterator it = kolekcija.createIterator();
        while (it.hasNext()) {
            System.writeLine(it.next());
        }
    }
}
```

```
int n;
int first;
public ParniBrojevi(first, n) {
    this.first = first;
    this.n = n;
}
```

1 MI 2011/2012

a)



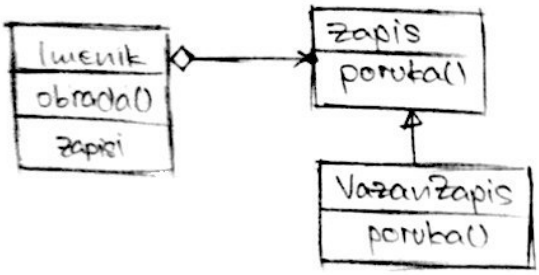
b) osnovna - 12 (a, b, vptr)
izvedena - 16 (a, b, c, vptr)

c) 0 -> vptr[0] (0, 5, 12)
this

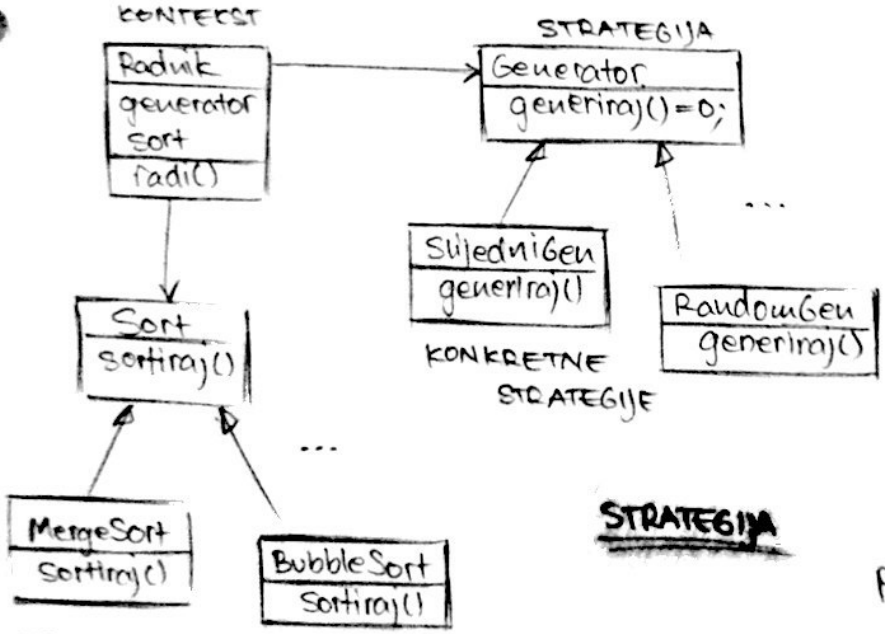
d) p->vptr = &vtable0osnovna
(postavljanje tablice virtualnih funkcija)

LNS - Liskovino načelo supstitucije

→ da bi ispravno radio NoviModul mora prihvatiti x>=6.0, što je strože preduvjet od preduvjeta osnovnog razreda Modul iz kojeg je izveden (x>=0.0)



- dinamički
  zapisi[i]->**poruka()**;
- statički
  zapisi[i]->poruka();

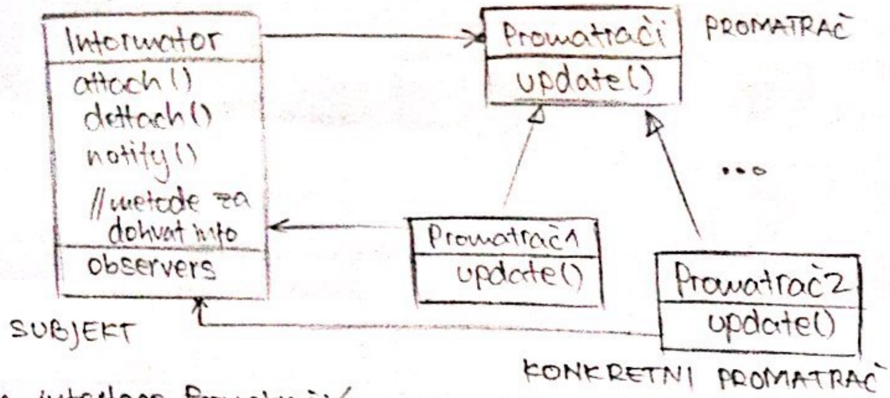



```
public class Radnik{
   Generator gen;
   Sort sort;  List<int> niz;
   public Radnik (Generator gen,
         Sort sort){
      this.gen = gen;
      this.sort = sort;
   }
   public void radi(){
      niz = gen.generiraj();
      sort.sortiraj(niz);
   }
}
public interface Generator{
   public List<int> generiraj();
}
public interface Sort{
   public void sortiraj(List<int> lista);
}
```

```
public class SljedniGen implements Generator{
   public List<int> generiraj(){
      List<int> lista = new ArrayList<int>();
      for(i=0; i<100; i++){
         lista.add(i);
      }
      return lista;
   }
}
```

```
public class BubbleSort implements Sort{
   public void sortiraj(List<int> lista){
      // implementacija bubble sorta
   }
}
```

5.

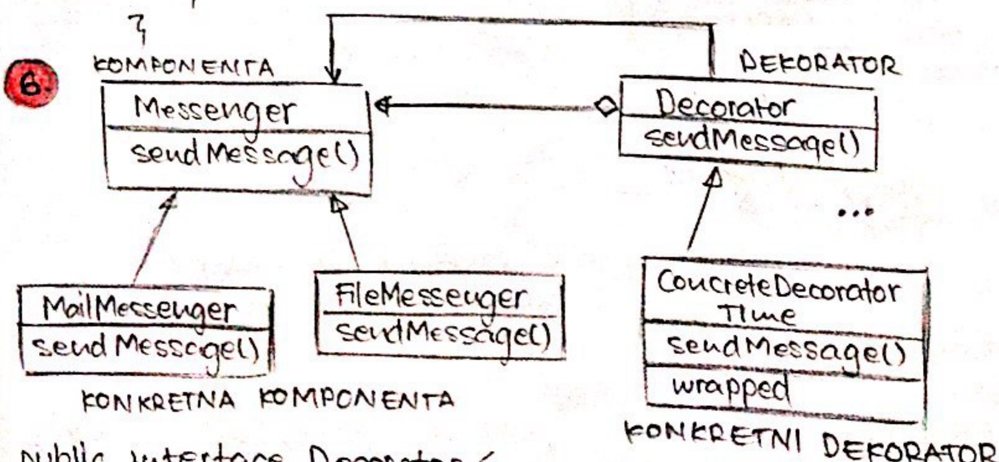


PROMATRAČ

```
public interface Promatrači{
    public void update();
}

public class Promatrač1 implements Promatrači{
    Informator info;
    public Promatrač1(Informator info){
        this.info = info;
        this.info.attach(this);
    }
    public void update(int[] info)
        // osvježavaju se informacije
    }
}

public class Demo{
    public static void main(String[] args){
        Informator info = new Informator();
        Promatrači p1 = new Promatrač1(this);
        Promatrači p2 = new Promatrač2(this);
        info.getInfo();
        info.notify();
    }
}
```

```
public class Informator{
    ArrayList<Promatrači> observers;
    --> ne može biti inicijalizacija
        observera u konstruktoru jer
        imamo kružnu vezu onda ???
        (može ako maknemo attach iz konstrukt.
        Promatrača i napravimo set metodu)
    public void attach(Promatrač p){
        observers.add(p);
    }
    public void dettach(Promatrač p){
        observers.remove(p);
    }
    public void notify(){
        for(Promatrač p : observers){
            p.update(int[] info)
        }
    }
    // metode za dohvat informacija
}
```

6.



DEKORATOR

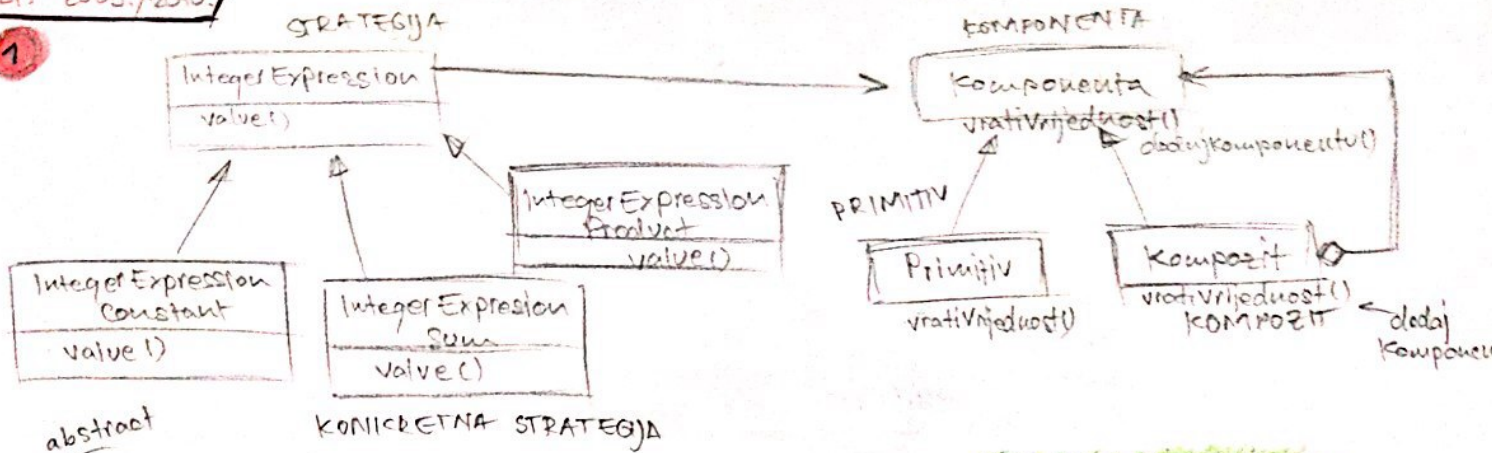
```
public interface Messenger{
    public void sendMessage();
}
public class MailMessenger
implements Messenger{
    public void sendMessage(){
        //implementacija slanja
          poruka na mail
    }
}
```

string msg

```
public interface Decorator{
    public void sendMessag(string msg);
}
public ConcreteDecoratorTime implements Decorator{
    Messenger wrapped;
    public ConcreteDecoratorTime(Messenger msg){
        wrapped = msg;
    }
    public void sendMessage(string msg){
        String buffer = sprintTime();
        msg = msg.concat(buffer);
        wrapped.sendMessage(msg);
    } }
```

ZI. 2009./2010.

1

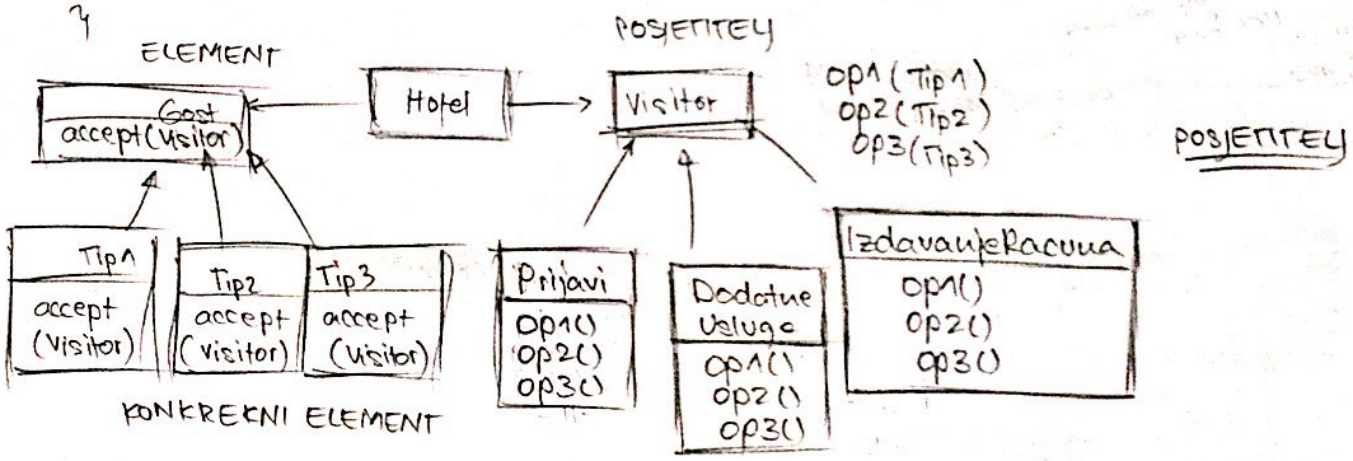


STRATEGIJA, KOMPOZIT

```
public abstract class IntegerExpression {
  abstract int value();
}

public class IntegerExpressionSum implements IntegerExpression {
        public IntegerExpressionSum ( Komponenta k1, Komponenta k2){
                this.k1 = k1;
                this.k2 = k2;
        }
      public int value (){
          return k1.vratiVrijednost() + k2.vratiVrijednost()
        }
}
```

private Komponenta k1;
private Komponenta k2;

2

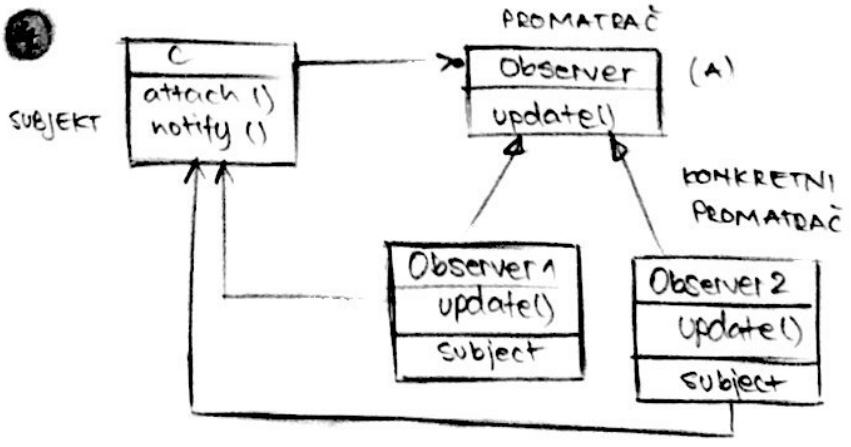



```
public class Tip1 implements Gost{
        public void accept (Visitor s){
                s.op1(this);
        }
}

public class Prijavi implements Visitor{
        public void op1 (Tip1 tip1) {
                // prijavljuje gosta tipa 1
        }
        public void op2 (Tip2 tip2){
                // prijavljuje gosta tipa 2
        }
        public void op3 (Tip3 tip3){
                // prijavljuje gosta tipa 3
        } }
```

PROXY


```
public class Proxy implements Slika{
    Slika slika;
    public int width(){
        if(slika==null)
            slika=new RGBSlika();
        return slika->width();
    }
    public int height(){
        if(slika==null)
            slika=new RGBSlika();
        return slika->height();
    }
    public const *getRGBPixels(){
        if(slika==null)
            slika=new RGBSlika();
        return slika->getRGBPixels();
    }
}
```

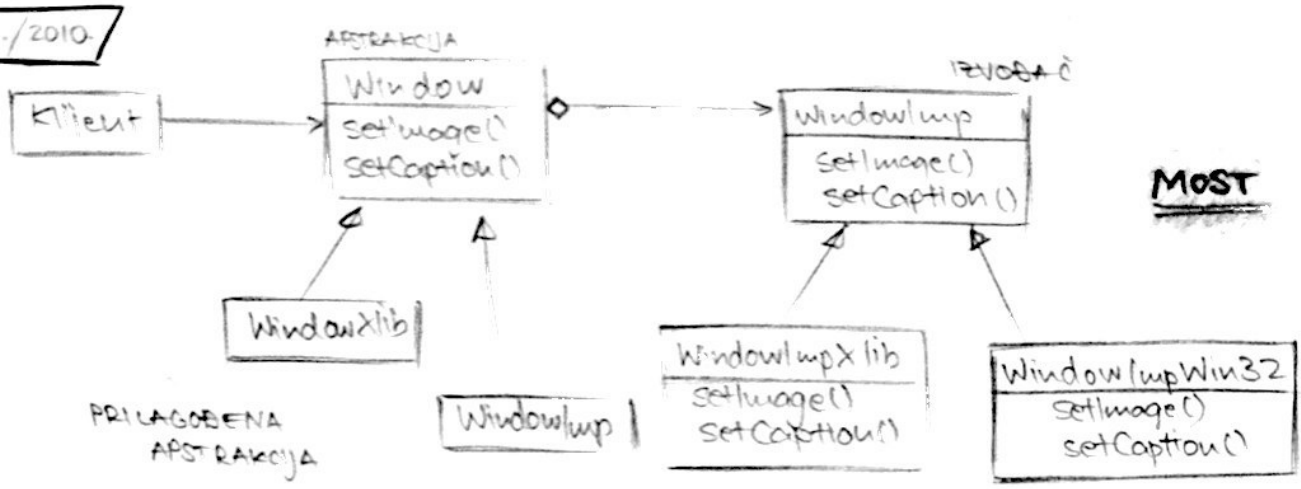
PROMATRAČ




```
public class Observer1 implements
    Observer{
    C subject;
    public Observer1(C subject){
        this.subject = subject;
        this.subject.attach(this);
    }
    public void update(){
        //nesto radi
    }
}
```

zi. 2008./2010.

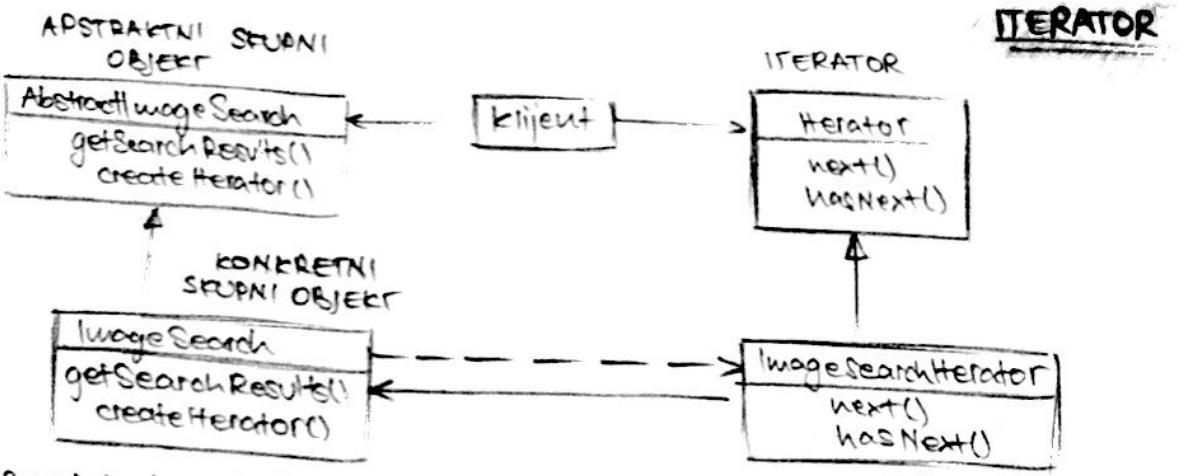
5.




```
public class Window {
    WindowImp implementation;
    public Window (WindowImp imp){
        implementation = imp
    }
    public void setImage (Image img) {
        return implementation.setImage(img);
    }
    public void setCaption ( string cap) {
        return implementation.setCaption (cap);
    }
}
public class WindowXlib implements Window {
    public WindowXlib() {        ??
        implementation = WindowImpXlib.createWindowImp();
    }
}
```

6.

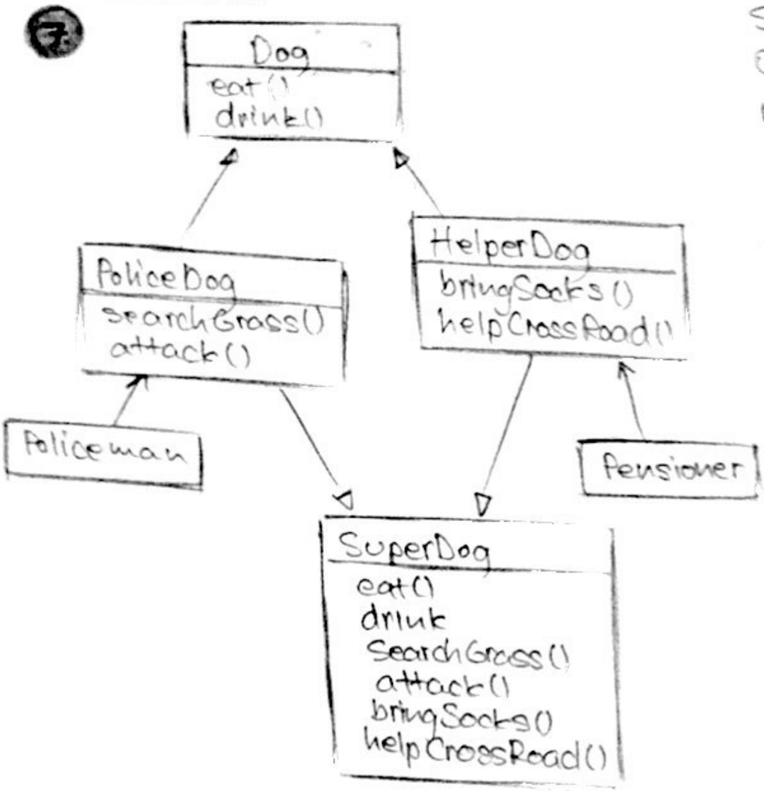
ITERATOR




```
public class ImageSearch implements AbstractImageSearch {
ArrayList<Image> getSearchResults( string search, int pageNumber){
public      // me no worry, me no care
    }
    public Iterator createIterator (){
        return new ImageSearchIterator (this);
    }
}
public class ImageSearchIterator implements Iterator {
    ArrayList<Image> list;        int position = 0;
    public ImageSearchIterator ( AbstractImageSearch is, string search)
        list = is.getSearchResult ( this.search, 1);
        this.search = search;
        this.is = is;
string search;
AbstractImage
Search is;
int page = 1;
else {
page++;
list = this.is.get
    SearchResult
    (search, page)
}
public bool hasNext () {
    if ( position < list.size() - 1)
        return false;
    return true;
}
public Image next () {
    if ( hasNext) {
    Image img = list.get(
        position);
    position++;
    return img;
    }
    else throw new
NoSuchElementException
```

1 MI 2011/2012

7.



Single Responsibility – nije zadovoljeno

Open-Closed – moguće dodavanje novih vrsta pasa

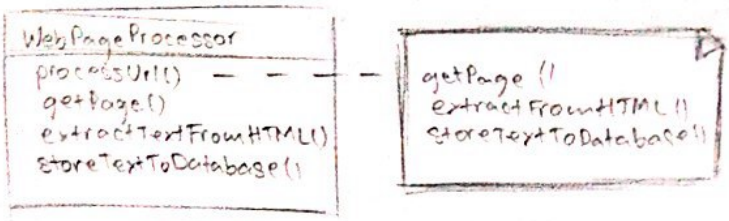
Liskov Supstitution – kao Dog možemo koristiti bilo PoliceDog bilo HelperDog

Interface Segregation – klijent vidi samo one metode koje koristi

Dependency Inversion – Policeman i Pensioner ovise o sučeljima

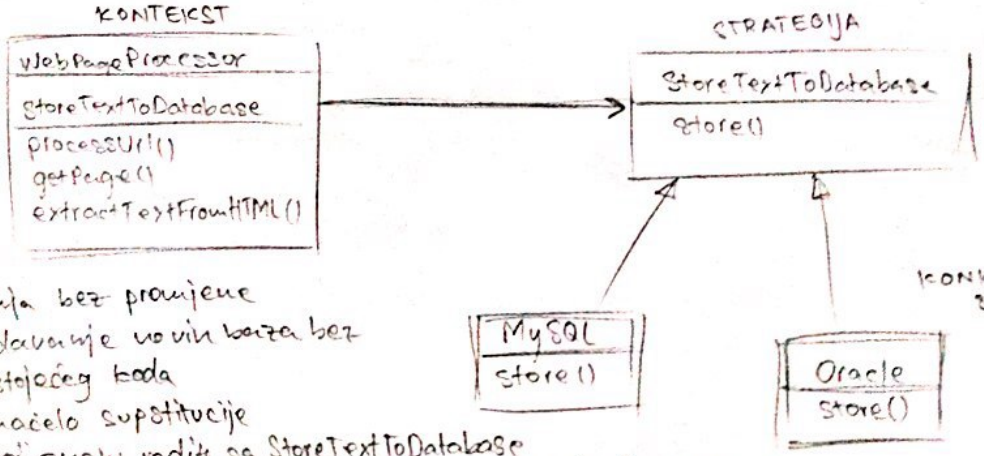
1. MI. 2009/2010

1.



NIO – Načelo inverzije ovisnosti
→ WebPageProcessor ne zna ništa o bazi (ovisi o sučelju)

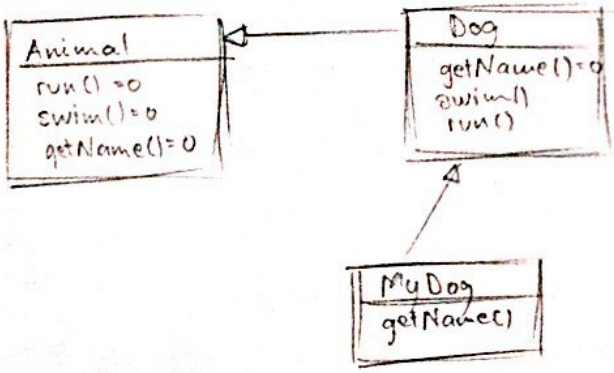
STRATEGIJA



NBP – nadograđuje bez promjene
→ omogućuje dodavanje novih baza bez promjene postojećeg koda
LNS – Liskovino načelo supstitucije
→ svi razredi koji znaju raditi sa StoreTextToDatabase znat će raditi i s MySQL i Oracle i drugim izvedenim klasama
NJO – načelo jedinstvene odgovornosti
→ StoreTextToDatabase zadužen je samo za spremanje podataka i izdvajamo ga iz WebPageProcessor

2.

a)

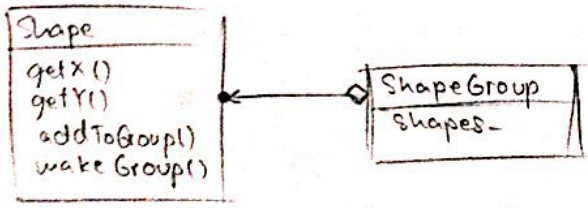


b) vt → MyDog: destruktor, run, swim, getName; Dog: destruktor, run, swim; Animal: destruktor

c) Stog → Heap
MyDog getName()

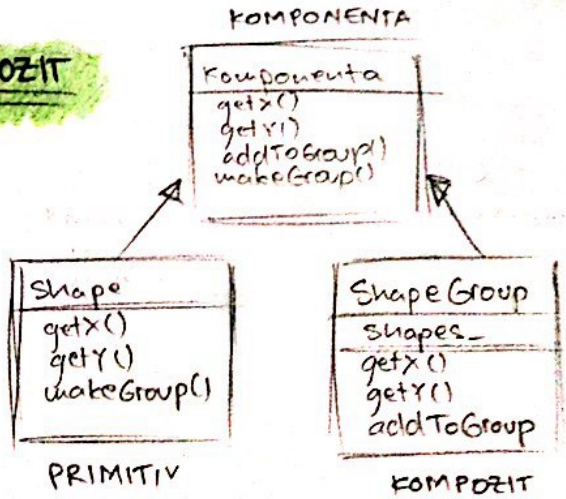
d) dinamički

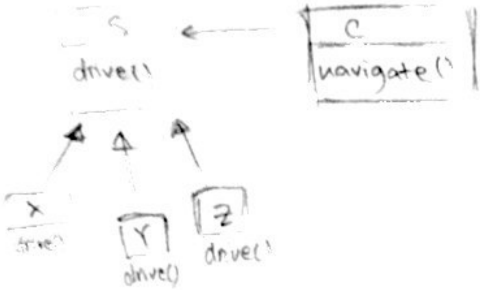
3.



→ Grupa ne može sadržavati druge grupe.

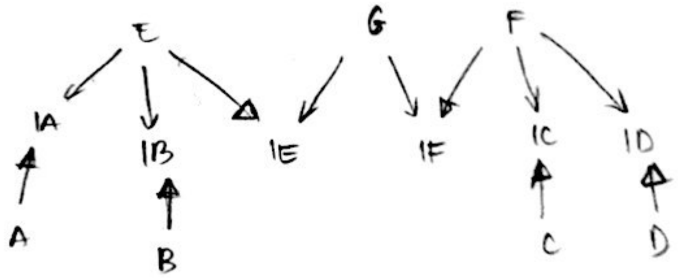
KOMPOZIT

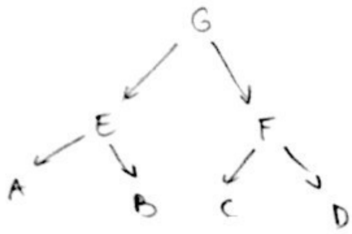


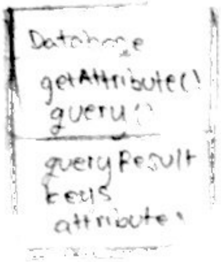



```
class S {
  public:
    virtual void drive()=0;
}
class X : S {
  public:
    virtual void drive();
}
void X::drive(){
    //nešto radi
}
//isto i Y i Z
```

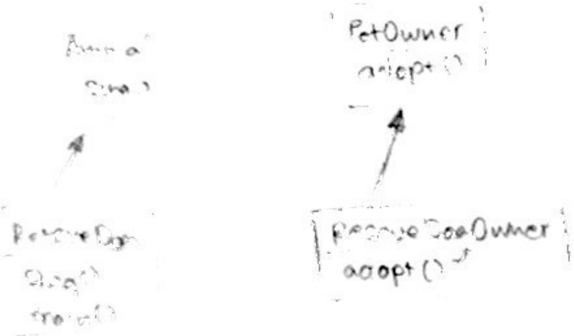
```
class C {
  public:
    void navigate();
    C(S& s);
  private:
    S s;
}
C::C(S &s){
  this.s = s;
}
C::navigate(){
   s.drive()
   //još nešto radi
}
```







No?



LNS - izvedeni razred ima strože preduvjete od razreda iz kojeg je izveden

provjeriti tip i baciti iznimku ako nije RescueDog

umjesto adopt Animal staviti adopt RescueDog

Scanned by CamScanner

MI 2012/2013

```
#include <stdio.h>
#include <stdlib.h>

struct vtable;
typedef struct {
    struct vtable *vtbl;
} Robot;
struct vtable {
    void (*kreni)(Robot *this);
    void (*stani)(Robot *this);
};
typedef struct {
    struct vtable *vtbl;
    int brzina;
} R2D2;

void testRobot(Robot *p){
    p->vtbl->kreni(p);
    sleep(2);
    p->vtbl->stani(p);
}
```

```
void R2D2_kreni(Robot *this){
    // potrebe R2D2
}
void R2D2_stani(Robot *this){
    // zaustavlja R2D2
}
struct vtablica R2D2_vtable = {
    R2D2_kreni, R2D2_stani};
R2D2 R2D2_stvori(int speed){
    R2D2* this = malloc(sizeof(R2D2));
    if(this == null) return null;
    this->vtbl = &R2D2_vtable;
    this->brzina = speed;
    return this;
}
int main(){
    testRobot((Robot*)R2D2_stvori(1));
}
```

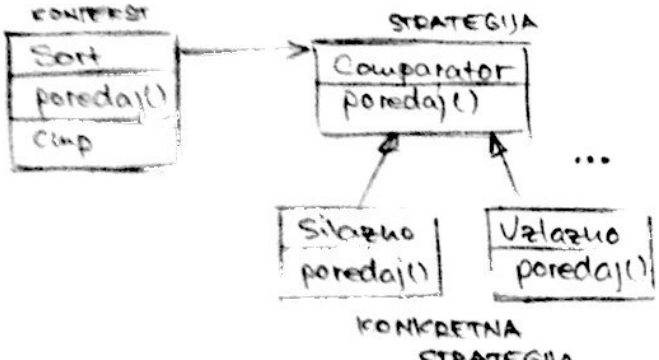
a)
```
class Comparator {
    virtual bool Compare(int, int) = 0;
};
void poredaj(std::vector<int>& L, Comparator& cmp){
    ...
    if(cmp.compare(L[j], L[min_j])){
    ...
    }
}
```
← statičko tipiziranje (java, c++)? dinamički polimorfizam

PRETEND LIKE IT DID NOT HAPPEN.

objekt koji uspoređuje

b)
```
def poredaj(L, cmp):
    ...
    if(cmp(L[j], L[min_j])):
    ...
```
← implicitno tipiziranje (python) dinamički polimorfizam

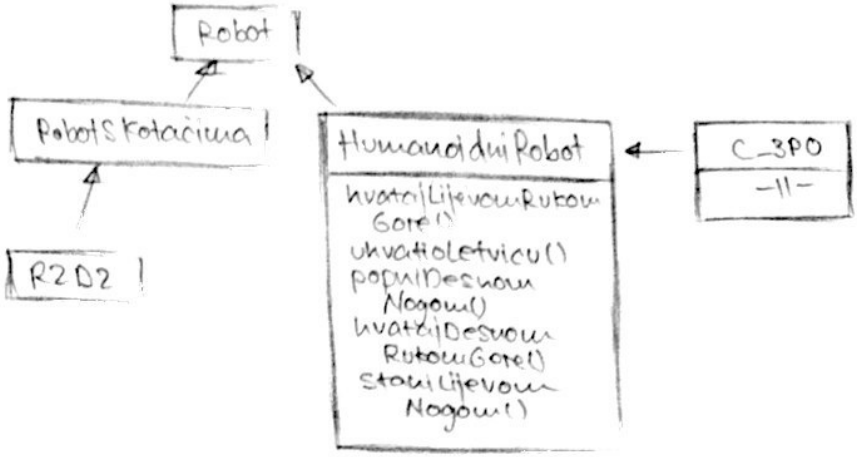
c)
```
template <typename T, typename Cmp>
void poredaj(std::vector<T>& L, Cmp& cmp){
    ...
    if(cmp.compare(L[j], L[min_j])){
    ...
    }
}
```
← statički polimorfizam (c++)

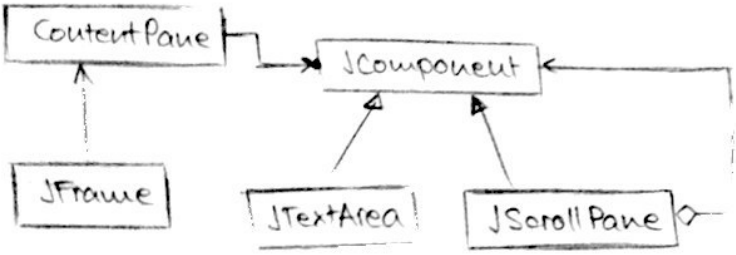


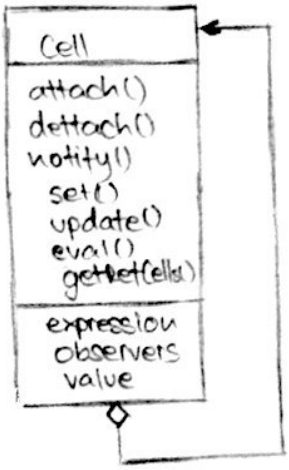
STRATEGIJA

LNS - Listovino Načelo Supstitucije

→ roboti s kotačima se ne mogu penjati, a nasljeđuju robota





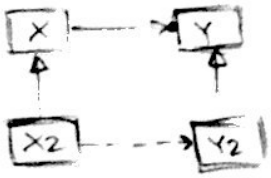


SUBJEKT
PROMATRAČ

```
public class Cell {
    String expression;
List<Cell> observers;
    int value;
    public void set (String exp) {
        for ( Cell c : getRefCells(expression))
            c.dettach(this);
        for ( Cell c : getRefCells(exp))
            c.attach(this);
        expression = exp;
        notify();
    }
public void attach(Cell c) {
    observers.add(c);
    }
public void dettach(Cell c) {
    observers.remove(c);
    }
public void update() {
    value = eval(expression);
    notify();
```

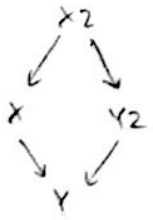
PROMATRAČ

```
public void notify() {
    for ( Cell c : observers) {
        c.update();
    }
}
```



dijagram razreda

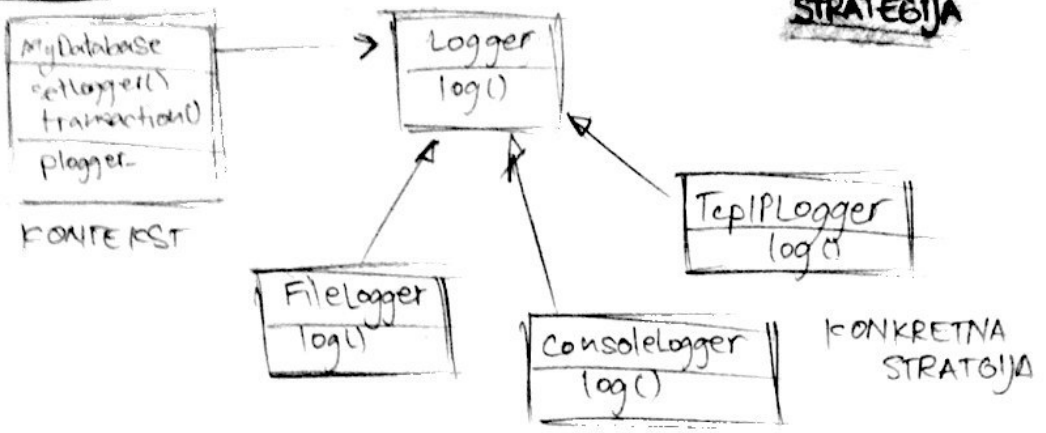
dijagram ovisnosti



METODA TVORNICA

MI 2013/2014

STRATEGIJA



STRATEGIJA

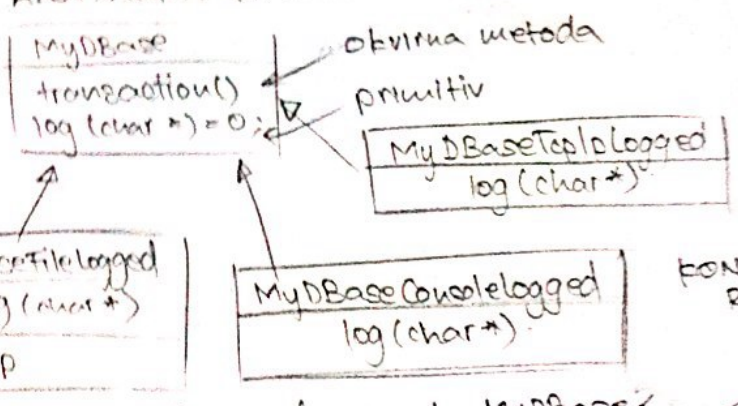
samo logiranje → NIO — Single Responsibility Principle
moguće dodavanje novih loggera → NBP — Open-Closed Principle
izvedeni razredi se mogu koristiti umjesto osnovnog → LNS — Liskov Substitution Principle
kada bi se izdvojile funkcionalnosti ne bi više bila očuvala jedinstvenost operacije → NIS — Interface Segregation Principle
NIO — Dependency Inversion Principle
↑
baza ovisi o sučelju, a ne implementaciji

```
#include <stdio.h>
#include <stdlib.h>
struct vtablica;
typedef struct {
    struct vtablica * vtbl;
} Logger;
struct vtablica {
    void (*log)(Logger *this, char *report);
};
typedef struct {
    Logger * plogger;
} MyDBase;
void setLogger(MyDBase *this, Logger * p1) {
    this->plogger = p1;
}
void transaction(MyDBase *this)
{
    char * report;
    // ..
    this->plogger->vtbl->log(this->plogger, report);
}
MyDBase * myDBase_stvori() {
    return malloc(sizeof(MyDBase));
}
```

```
typedef struct {
    struct vtablica * vtbl;
    FILE *t;
} FileLogger;
void filelogger_log(Logger *t, char
    *report) {
    FileLogger *this = (fileLogger*) t;
    fprintf(this->t, "%s\n", report);
    fflush(this->t);
}
struct vtablica filelogger_tablica =
    { filelogger_log };
FileLogger *filelogger_stvori(char filename) {
    FileLogger * this = malloc(sizeof(
        FileLogger));
    if(this == null) return null;
    this->vtbl = &filelogger_tablica;
    this->t = fopen(filename, "wt+");
    return this;
}
int main() {
    Logger * log = (Logger*)
        filelogger_stvori("D://log.txt");
    MyDBase *db = myDBase_stvori();
    setLogger(db, log);
    transaction(db);
}
```

3

APSTRAKTNI RAZRED

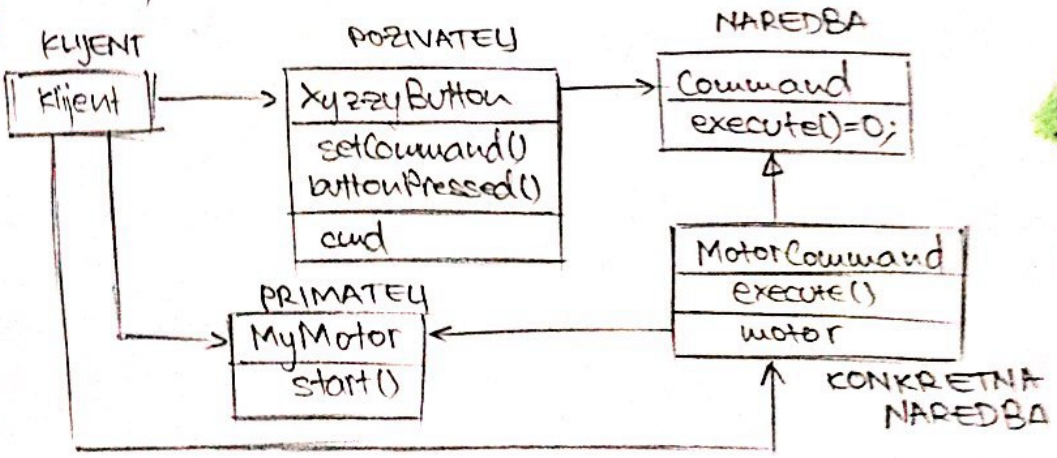


OKVIRNA METODA

```
public class MyDBaseFileLogged extends MyDBase {      // FILE fp;
    public MyDBaseFileLogged (string filename) {
        fp = open (filename, "w+");
    }
    public void log (char* report) {
        print (fp, "%s\n", report);
        flush (fp);
    }
}

public abstract class MyDBase {
    public final void transaction () {
        string report;
        // ...
        log(report);
    }
    public abstract log(char * report);
}
```

```
int main() {
    MyDBase db = new
    MyDBaseFileLogged
    ("D:\log.txt");
    db.transaction();
}
```

4



NAREDBA

```
MotorCommand cmd =
    new MotorCommand
    (&myMotor);
theButton.setCommand
    (cmd);
theButton.buttonPressed();
```

```
public class XyzzyButton {
    Command cmd;
    public void setCommand (Command cmd) {
        this.cmd = cmd;
    }
    public void buttonPressed () {
        cmd.execute();
    }
}
```

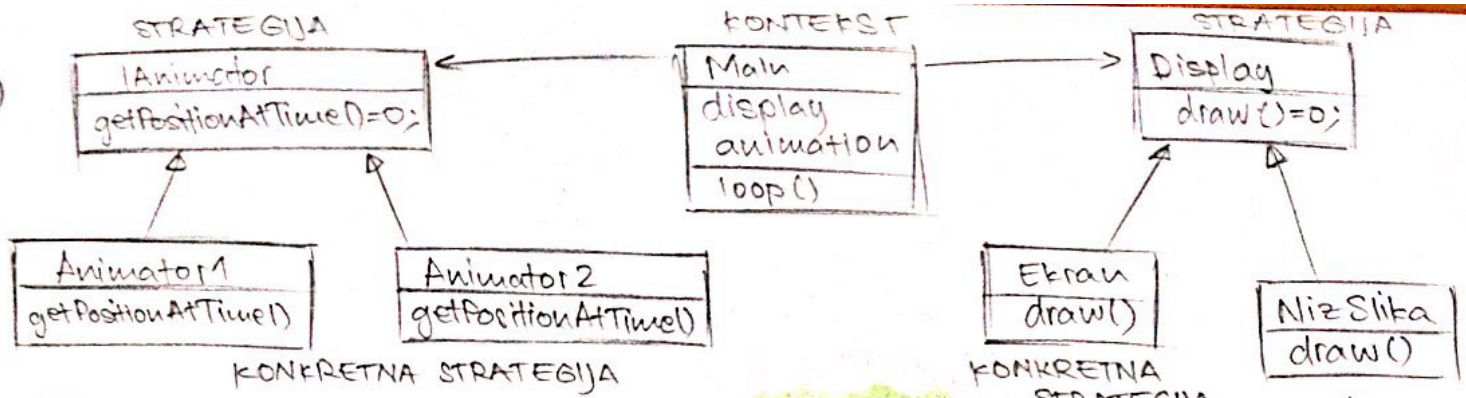
```
public class MotorCommand extends Command {
    MyMotor motor;
    public MotorCommand (MyMotor motor) {
        this.motor = motor;
    }
    public void execute() {
        motor.start();
    }
}
```

```
public class MyMotor {
    public void start() {
        // pokreće motor
    }
}
```

```
int main() {
    MyMotor theMotor = new
    XyzzyButton theButton = new
```

5

a)



STRATEGIJA

```
void loop(){
    double time=0;
    While (true){
        Point p = new Point();
        double w=animator.getPositionAtTime(time, p);
        display.draw(p);
        sleep(w);
        time+=w;
        }
}
```

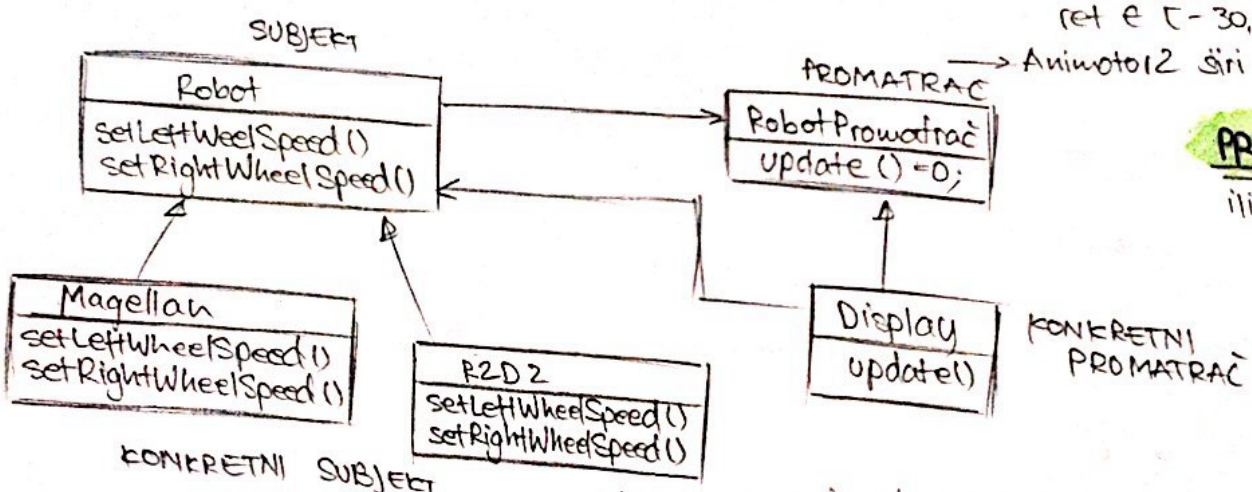
b) LNS - Listovino Načelo Supstitucije

Animator 1
$x \in [100, 299]$
$y \in \{75\}$
$ret \in [150, 199]$

Animator 2
$x \in [0, 700]$
$y \in \{75\}$
$ret \in [-30, 270]$

→ Animator2 širi od Animator1

6



PROMATRAČ ili DEKORATOR

da ima attach i notify onda bi ovo načelo bilo prekršeno

sadrži samo bitne operacije ← Single Responsibility Principle

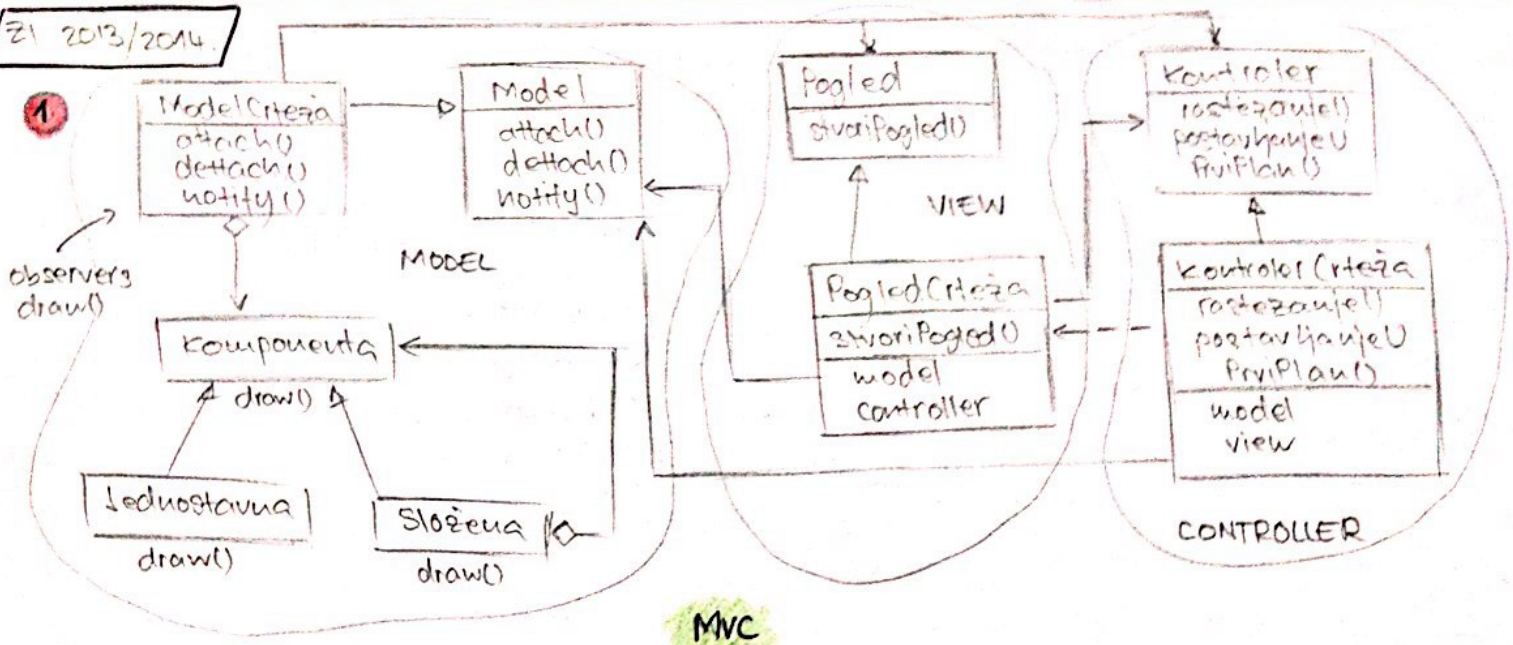
moguće dodavanje novih robota i promatrača ← Open-Closed Principle

može se koristiti i Magellan i R2D2 umjesto Robota ← Liskov Substitution Principle

ovise o sučeljima, ove o implementacijama (Robot, Display) ← Interface Segregation Principle / Dependency Inversion Principle

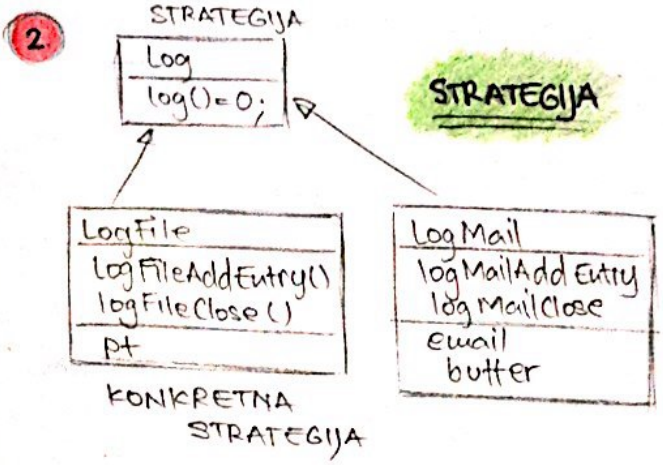
ZI 2013/2014.

1



MVC

2

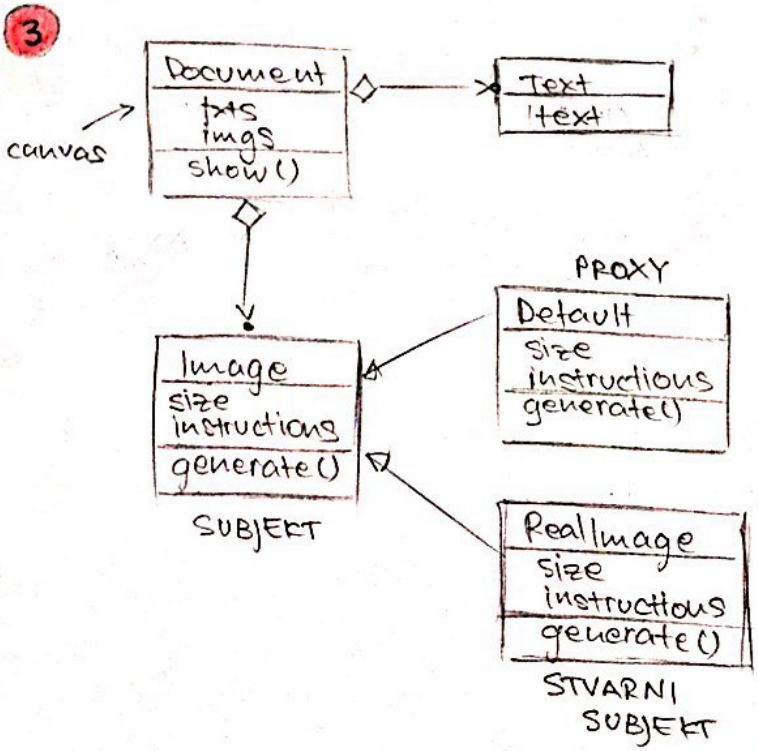


STRATEGIJA

```
LogFile* logFileCreate(char* filename){
    LogFile* this = malloc(sizeof(LogFile));
    if(this==null) return null;
    this->vtable = &logFile;
    this->pt = fopen(filename,"w+");
}
void logFileAddEntry(LogFile*this, char*text){
    (this->pt).twrite(text);
}
void logFileClose(LogFile*this){
    this->pt.fclose();
}
```

PROXY

3



PROTOTIP, KOMPOZIT