

Objektno orijentisano programiranje u C++-u

Projektni uzorci

1

STRUKTURNI PROJEKTNI UZORCI

ADAPTER
MOST
KOMPOZITNI
DEKORATER

Ime i klasifikacija

- Adapter (engl. Adapter)
- Strukturni projektni uzorak

Drugo ime

- Omotač (engl. Wrapper)

Namena

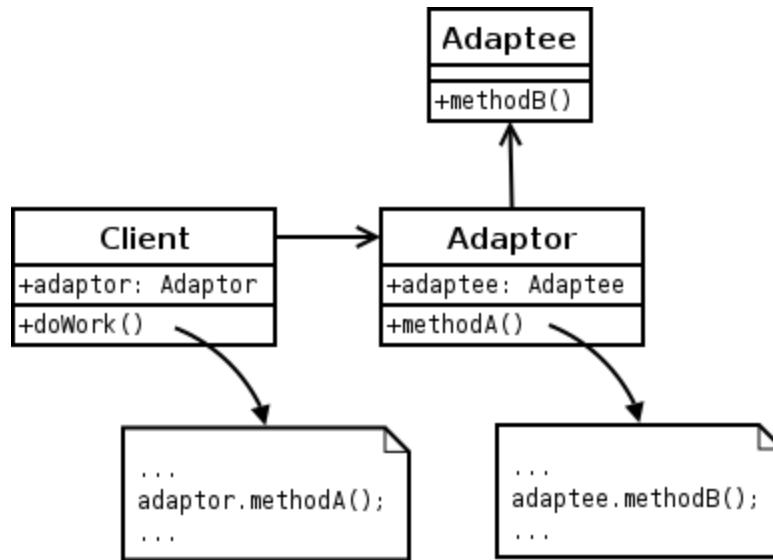
- Koristi se kada raspoloživa klasa nema interfejs kakav nam odgovara
- Koristi se nekoliko postojećih klasa, kada nije praktično da se njihovi interfejsi modifikuju višestrukim izvodenjem iz svake od tih klasa.

ADAPTER (engl. ADAPTER)

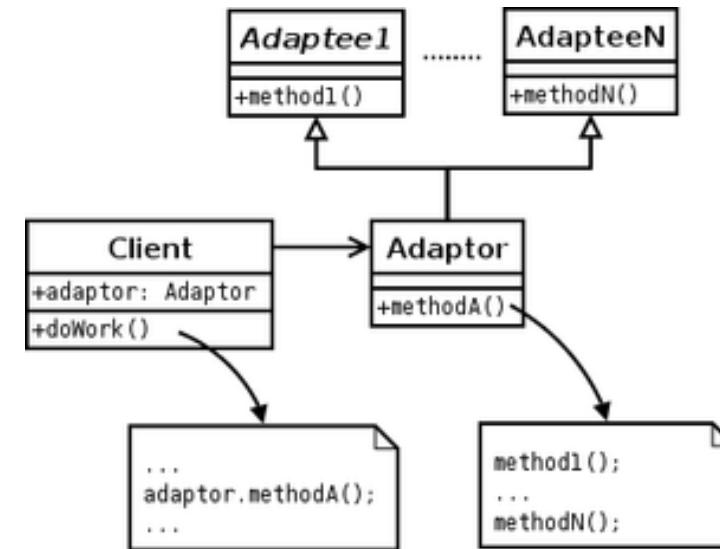
1. Uvod

3

□ Struktura



Adapter objekta



Adapter klase

❑ Učesnici

❑ Cilj

- ❑ Definiše interfejs koji koristi klijent i koji je specifičan za dati domen

❑ Klijent

- ❑ Saradjuje sa objektima koji poštuju interfejs **Cilj**

❑ Adaptirani

- ❑ Definiše postojeći interfejs koji zahteva adaptiranje. klasa koja se adaptira, pretpostavka je da se ova klasa ne može menjati jer nemamo pristup i prava da menjamo izvorni kod ili bi direktnе izmene nad izvornim kodom bile previše kompleksne.

❑ Adapter

- ❑ Adaptira interfejs Adaptirani na interfejs **Cilj**. Nadležnost ove klase su konverzije tipova, parametara i sl. Najčešće kao mehanizam prilagođavanja koristi delegiranje poziva metoda, ređe se koristi nasleđivanje.

❑ Saradnja

- ❑ **Klijenti pozivaju operacija objekata adaptera, a adapter poziva operacije adaptiranog pod-objekta (nasledjivanjem ili agregacijom) da izvrše zahtev**
- ❑ **Klasni Adapter koristi višestruko nasleđivanje za međusobno prilagođavanje interfejsa.**
- ❑ **Objektni Adapter koristi agregaciju objekata adaptirane klase.**

❑ Posledice

- ❑ **Prilagođavamo komponentu čija nam je funkcionalnost potrebna ali njen interfejs ne odgovara našim zahtevima.**

ADAPTER (engl. ADAPTER)

Primer 1: Adapter objekata – virtuelizacija nevirtuelnih funkcija

6

```
/* CILJNI Iterfejs koji klijent prepoznaje */
class IzvrsniInterfejs {
public:
    virtual ~IzvrsniInterfejs() {}
    virtual void izvrsi() = 0;
};

// ADAPTER
template <class TYPE>
class IzvrsniAdapter : public IzvrsniInterfejs {
public:
    IzvrsniAdapter( TYPE* pi, void (TYPE::*pf) () ) : ptrToInst(pi),
ptrToFun(pf) {}
    virtual ~IzvrsniAdapter(){ delete ptrToInst; }
    virtual void izvrsi(){ (ptrToInst->*ptrToFun)(); }
private:
    TYPE* ptrToInst; /* pokazivac na instancu */
    void (TYPE::*ptrToFun)(); /* pokazivac na funkciju clanicu objekta
tipe TYPE. Sve funkcije koje se "adaptiraju" moraju da imaju iste
parametre i isti tip rezultata */
};
```

ADAPTER (engl. ADAPTER)

Primer 1: Adapter objekata – virtuelizacija nevirtuelnih funkcija

7

```
// Nekompatibilne klase: one koje se adaptiraju
```

```
class A {  
public:  
    ~A(){ cout << "A::destruktor" << endl; }  
    void akcija() { cout<<"A::akcija()"<<endl; } // NIJE VIRTUELNA  
};
```



```
class B {  
public:  
    ~B(){ cout << "B::destruktor" << endl; }  
    void promena() { cout<<"B::promena()"<<endl; } // NIJE VIRTUELNA  
};
```



```
class C {  
public:  
    ~C(){ cout << "C::destruktor" << endl; }  
    void start() { cout<<"C::start()"<<endl; } // NIJE VIRTUELNA  
};
```

ADAPTER (engl. ADAPTER)

Primer 1: Adapter objekata – virtuelizacija nevirtuelnih funkcija

8

```
int main() {
    Array<IzvrsniInterfejs*> arrPtrToAdapt(3);
    niz[0] = new IzvrsniAdapter<A>(    new A(),      &A::akcija      );
    niz[1] = new IzvrsniAdapter<B>(    new B(),      &B::promena      );
    niz[2] = new IzvrsniAdapter<C>( new C(),     &C::start ) ;

    for (int Idx=0; Idx < 3; ++Idx) arrPtrToAdapt[Idx]->izvrsi();
    for (int Idx=0; Idx < 3; ++Idx) delete arrPtrToAdapt[Idx];
    return 0;
}
```

ADAPTER (engl. ADAPTER)

Primer 2: Adapter objekata – funkcije interfejsa sa različitim parametrima

9

```
#include <iostream>
using namespace std;
/* Interfejs koji klijent prepoznae */
struct IPrintable {
    virtual void print(const char* = 0) const = 0;
    virtual ~IPrintable() {}
};

/* Nekompatibilne klase: one koje se adaptiraju */
struct Printer {
    void print() const { cout << "Print internal PrintJob" << endl; }
};

struct Fax {
    void fax(const char* ptrToPrintable) const {
        cout << "Fax & Print " << ptrToPrintable << endl;
    }
}
```

ADAPTER (engl. ADAPTER)

Primer 2: Adapter objekata – funkcije interfejsa sa različitim parametrima

10

```
/* Primera resavanja problema razlicitih argumenata */
/* Primenom globalnog sablona funkcije, koji mora da ima "uniju"
   parametara (sa podrazumevanim vrednostim) svih funkcija za koje
   treba da se instancira. */
```

```
template <class T>
void adapt(T *t, const char* ptrToPrintable) {
    t->print(ptrToPrintable);
}

/* Specijalizacije sablona za razlicite tipove stampaca */
template<>
void adapt<Printer>(Printer *t, const char* ptrToPrintable) {
    t->print();
}

template<>
void adapt<Fax>(Fax *t, const char* ptrToPrintable) {
    t->fax(ptrToPrintable);
}
```

ADAPTER (engl. ADAPTER)

Primer 2: Adapter objekata – funkcije interfejsa sa različitim parametrima

11

```
/* Adapter */
template <class T>
class PrintableAdapter : public IPrintable {
public:
    PrintableAdapter(T* ptr) : ptrGenPrinter(ptr) {}
    virtual void print(const char* ptrToPrintable = 0) const {
        adapt(ptrGenPrinter, ptrToPrintable);
    }

private:
    /* Pokazivac na objekat koji je sposoban da
     stampa, prikaze neki sadrzaj */
    T* ptrGenPrinter;
    virtual ~PrintableAdapter() { delete ptrGenPrinter; } /* Obavezno
     virtuelan: objektu tipa PrintableAdapter<T> pristupa se preko
     pokazivaca tipa IPrintable */
};
```

ADAPTER (engl. ADAPTER)

Primer 2: Adapter objekata – funkcije interfejsa sa različitim parametrima

12

```
/* Klijent */
int main() {
    IPrintable* p = new PrintableAdapter<Fax>(new Fax); /* Fax sme da
    bude samo na heapu. PrintableAdapter<Fax> preuzima obavezu da ga
    obriše */
    p->print("Dokument");
    delete p;

    Printer printer;
    p = new PrintableAdapter<Printer>(new Printer); /* Printer sme da
    bude samo na heapu. PrintableAdapter<Printer> preuzima obavezu da
    ga obriše */
    p->print();
    delete p;

    return 0;
}
```

ADAPTER (engl. ADAPTER)

Primer 3: Adapter klase

13

```
class ITurbina{ /* Novi interfejs koji klijent očekuje */  
public:  
    virtual void PokreniTurbinu() = 0;  
};  
  
class Kompresor{ /* Klasa koja vec postoji i deo je starog koda */  
    string m_tip;  
public:  
    Kompresor(string tip) : m_tip(tip){}  
    /* Funkcija koja predstavlja deo postojeceg interfejsa */  
    void PokreniKompresor(int br_obrt){  
        cout << "\nKompresor tipa " << m_tip.c_str() << " se vrti na " <<  
        br_obrt << " obrtaja\n";  
    }  
};
```

ADAPTER (engl. ADAPTER)

Primer 3: Adapter klase

14

```
class Adapter : public ITurbina, private Kompresor{  
    int m_br_obrt;  
public:  
    Adapter(string tip, int br_obrt):  
        Kompresor(tip),m_br_obrt(br_obrt) {}  
    /* Implementacija novog ocekivanog interfejsa */  
    void PokreniTurbinu(){  
        PokreniKompresor(m_br_obrt);  
        /* Dostupna zahvaljujuci privatnom nasledjivanju */  
    }  
};
```

ADAPTER (engl. ADAPTER)

Primer 3: Adapter klase

15

```
/* Kod "klijenta" */

int main(){
    ITurbina* ptrTurbina = new Adapter("Bosh", 12500); /* Moguce
zahvaljujuci javnom nasledjivanju */

    /* Poziv funkcije koji adapter preusmerava ka starom kodu */
    ptrTurbina->PokreniTurbinu();
    delete ptrTurbina;
    return 0;
}
```

Ime i klasifikacija

- Most (engl. Bridge)
- Strukturni projektni uzorak

Namena

- Razdvaja apstrakciju od implementacije kako bi mogle nezavisno da se menjaju nasljeđivanjem

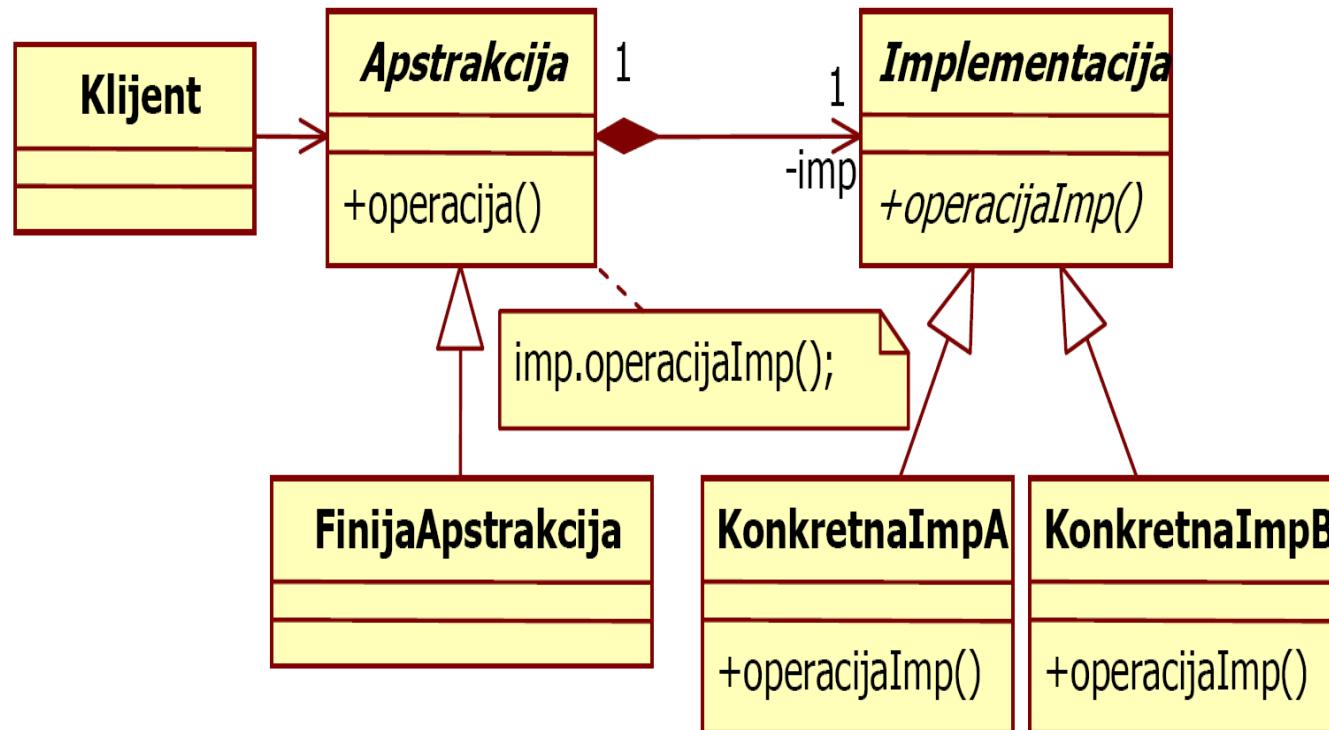
Drugo ime

- Ručica-Telo (engl. Handle – Body)

❑ Primenljivost

- ❑ **Most se koristi kada treba da se izbegne trajno vezivanje apstrakcije i njene implementacije**
 - ❑ Potrebno je, na primer, menjati implementaciju u toku izvršavanja
- ❑ **Apstrakciji i implementaciji je potrebno proširivanje kroz potklase**
 - ❑ Saradjuje sa objektima koji poštuju interfejs **Cilj**
- ❑ **Promena u implementaciji apstrakcije ne sme da utiče na klijente**
 - ❑ Definiše postojeći interferjs koji zahteva adaptiranje
- ❑ **U C++-u potpuno skrivanje implementacije klase od klijenta**
 - ❑ Definicije klase su u h fajlovima
- ❑ **Istu implementaciju treba da deli više objekata, a da to bude sakriveno od klijenta(uz eventualno brojanje referenci)**

□ Struktura



❑ Učesnici

❑ Apstrakcija (Apstraktna klasa)

- ❑ Definiše interfejs apstrakcije
- ❑ Sadrži referencu na objekat implementacije

❑ Naslednik Apstrakcije (konkretna klasa)

- ❑ Proširuje interfejs apstrakcije

❑ Implementacija

- ❑ Deklariše interfejs klase implementacije
- ❑ Interfejs ne mora da liči na interfejs apstrakcije

❑ Konkretna implementacija

- ❑ Implementira interfejs deklarisan u klasi Implementacija i definiše konkretnu implementaciju

❑ Saradnja

- ❑ Apstrakcija prosleđuje klijentske zahteve objektu Implementacija

❑ Posledice

- ❑ Razdvajanje interfejsa od implementacije

- ❑ Implementacija nije trajno vezana za apstrakciju i može da se konfiguriše dinamički

- ❑ Eliminisane su zavisnosti implementacije i apstrakcije u vreme prevodjenja

- ❑ Izmena klase apstrakcije ne zahteva prevodjenje klase implementacije
 - ❑ Izmena konkretnih implementacija ne izaziva prevodjenje klase apstrakcije

❑ Posledice (nastavak)

❑ **Bolje mogućnosti proširivanja**

- ❑ Hijerarhije apstrakcija i implementacija se mogu nezavisno proširivati

❑ **Skrivanje detalja implementacije od klijenata**

- ❑ Klijent ne vidi ništa od implementacije, vidi samo apstrakciju
- ❑ Bilo kakva izmena na Implementaciji ne utiče na klijenta, izmena Apstrakcije utiče

❑ Povezani uzorci

❑ **Apstraktna fabrika može da kreira i konfiguriše Most**

❑ **I Adapter i Most prilagođavaju klijentu interesne neke implementacije**

- ❑ Adapter se obično projektuje retroaktivno
- ❑ Most se obično projektuje unapred

MOST (engl. BRIDGE)

Primer 1. Implementator crtanja i apstrakcija slike

22

```
/* Implementacija */
class ImplementatorCrtanja {
public:
    virtual void crtajPortret(double, double) = 0;
    virtual ~ImplementatorCrtanja() {}
};

/* Konkretni Implementator crtanja LEONARDO */
class DaVinci: public ImplementatorCrtanja {
public:
    DaVinci() {}
    virtual ~DaVinci() {}

    /* Crtanje perom */
    void crtajPortret(double visina, double sirina) {
        cout << "\nLeonardov crtez velicine" << visina << " x " << sirina <<
    endl;
    }
};


```

MOST (engl. BRIDGE)

Primer 1. Implementator crtanja i apstrakcija slike

23

```
/* Konkretni Implementator crtanja PICASSO */
class Picasso: public ImplementatorCrtanja {
public:
    Picasso(){}
    virtual ~Picasso() {}
    /* Crtanje olovkom */
    void crtajPortret(double visina, double sirina) {
        cout << "\nPikasov crtez velicine" << visina << " x " << sirina <<
    endl;
    }
};

/* Apstrakcija */
class Slika {
public:
    virtual void crtaj() = 0;
    virtual void promeniVel(double) = 0;
    virtual void promeniUmetnika(RCPointer<ImplementatorCrtanja>) = 0;
    virtual ~Slika(){}
};
```

```
/* Doradjena Apstrakcija */
class Portret : public Slika {
public:
    Portret(double vis, double sir, RCPointer<ImplementatorCrtanja> impC)
    :
        visina(vis), sirina(sir), pToUmetnik(impC) {}
    virtual ~Portret() {}

    /* Zavisi od Implementacije */
    void crtaj() { pToUmetnik->crtajPortret(visina, sirina); }

    /* Zavisi od Apstrakcije */
    void promeniVel(double pct) { visina *= pct; sirina *= pct; }

    /* Zavisi od Apstrakcije I Implementacije */
    void promeniUmetnika(RCPointer<ImplementatorCrtanja> impc) {
        pToUmetnik = impc;
    }

private:
    double visina, sirina;
    RCPointer<ImplementatorCrtanja> pToUmetnik;
};
```

MOST (engl. BRIDGE)

Primer 1. Implementator crtanja i apstrakcija slike

25

```
int _tmain(int argc, _TCHAR* argv[]){  
  
    RCPPointer<DaVinci> pLeonardo = new DaVinci();  
    RCPPointer<Picasso> pPablo = new Picasso();  
  
    Portret sqA(160, 80, pLeonardo);  
    Portret sqB(40, 40, pPablo);  
  
    Slika* ptrToOblik[2];  
    ptrToOblik[0] = &sqA;  
    ptrToOblik[1] = &sqB;  
  
    ptrToOblik[0]->promeniVel(10); ptrToOblik[0]->crtaj();  
    ptrToOblik[1]->promeniVel(10); ptrToOblik[1]->crtaj();  
    ptrToOblik[1]->promeniUmetnika(pLeonardo); ptrToOblik[1]->crtaj();  
    return 0;  
}
```

MOST (engl. BRIDGE)

Primer 2.

26

```
/* Apstrakti implementator*/
class ProtokolImpl{
public:
    virtual void send( const std::string& paketPodataka ) const = 0;
};

/* UDP (engl. User Datagram Protocol) se koristi za razmenu paketa poruka („datagrama“) između računara. Za razliku od protokola TCP, ovaj protokol ne podrazumeva stalnu vezu nego se paketi „bacaju“ odredišnom računaru, bez održavanja veze i provere grešaka. Na taj način, ovaj protokol ne garantuje isporuku paketa niti isti redosled isporuke paketa kao pri slanju. Zbog ovih osobina UDP protokol je brz i koristi se za aplikacije kojima je važna brzina a prispeće paketa i održavanje redosleda nije od velike važnosti, koristi ga veliki broj aplikacija, naročito multimedijalne aplikacije poput internet telefonije i video konferencije.*/
/* Konkretni implementator A*/
class UDPProtokolImpl: public ProtokolImpl{
public:
    virtual void send( const std::string& paketPodataka ) const {
        std::cout << "slanje: " << paketPodataka <<
                    " preko UDP protokola" << std::endl;
    }
};
```

MOST (engl. BRIDGE)

Primer 2.

27

```
/* Transmisioni kontrolni protokol (TCP, engl. Transmission control protocol) je protokol koji pripada sloju 4 OSI referentnog modela, ima za ulogu da obezbedi pouzdan transfer podataka u IP okruženju. Između ostalih servisa koje nudi, neki su: pouzdanost, efikasna kontrola toka podataka, operisanje u ful-dupleksu (istovremeno slanje i primanje podataka) i multipleksiranje koje omogućava istovremen rad niza procesa sa viših slojeva putem jedne konekcije. TCP vrši transfer podataka kao nestrukturisan niz bajtova koji se identificuju sekvencom. Ovaj protokol grupiše bajtove u segmente dodeli im broj sekvence, aplikacijama dodeli broj porta i prosledi ih IP protokolu. */
```

```
/* Konkretni implementator B*/
class TCPProtokolImpl: public ProtokolImpl{
public:
    virtual void send( const std::string& paketPodataka ) const {
        std::cout << "slanje: " << paketPodataka <<
                    " preko TCP protokola" << std::endl;
    }
};
```

MOST (engl. BRIDGE)

Primer 2.

28

```
/*Apstrakcija */
class IPaketPodataka {
public:
    IPaketPodataka(ProtokolImpl *ptrImp) : m_ptrImp(ptrImp) { }
    /* deo koji zavisi od implementacije*/
    virtual void send() { m_ptrImp->send(m_paketPodataka); }
protected:
    ProtokolImpl *m_ptrImp;
    std::string m_paketPodataka;
};

/* Konkretna apstrakcija A*/
class UDPPaket: public IPaketPodataka {
public:
    UDPPaket(const std::string& paketPodataka) :
        IPaketPodataka(new UDPProtokolImpl()) {
        m_paketPodataka = paketPodataka;
    }
};
```

/*Konkretna apstrakcija B*/

```
class TCPPacket: public IPaketPodataka {  
public:  
    TCPPacket(const std::string& paketPodataka) :  
        IPaketPodataka(new TCPProtokolImpl()) {  
            m_paketPodataka = paketPodataka;  
        }  
};
```

/* Umesto definisanja velikog broja klasa, implementiracemo sablon cijim se instanciranjem stvaraju konkretne apstrakcije */

/*Sablon Apstrakcije */

```
template<typename ProtokolImplType>  
class PaketPodataka:public IPaketPodataka{  
public:  
    PaketPodataka(const std::string& paketPodataka) :  
        IPaketPodataka(new ProtokolImplType()) {  
            m_paketPodataka = paketPodataka;  
        }  
};
```

MOST (engl. BRIDGE)

Primer 2.

30

```
/* Klijent */
int main(int argc, char *argv[])
{
/* Klijentski kod kada se koriste definicije konkretnih klasa */

IPaketPodataka *paketPtrA[]={  
    new UDPPaket("Paket podatka #1"),  
    new UDPPaket("Paket podatka #2"),  
    new TCPPacket("Paket podatka #3")  
};  
for (int i = 0; i < 3; i++)  
    paketPtrA[i]->send();  
}
```

MOST (engl. BRIDGE)

Primer 2.

31

```
/* Klijent */
int main(int argc, char *argv[])
{
/* Klijentski kod kada se koristi instanciranje sablona, da bi se doobile
konkretne klase */

IPaketPodataka *paketPtrB[] = {
    new PaketPodataka<UDPProtokolImpl>("Paket podatka #1"),
    new PaketPodataka<UDPProtokolImpl>("Paket podatka #2"),
    new PaketPodataka<TCPProtokolImpl>("Paket podatka #3")
};

for (int i = 0; i < 3; i++)
    paketPtrB[i]->send();
return 0;
}
```

MOST (engl. BRIDGE)

Primer 3.

32

```
template<typename T>
class IMemorija{
public:
    virtual void Dodaj(T)=0;
    virtual T Uzmi() = 0;
};

template< typename T>
struct Cvor {
    T info;
    Cvor<T> *sled;
    Cvor(T n_info, Cvor<T> *n_sled = 0):info(n_info), sled(n_sled){}
};

template<typename T>
class LancanaLista:public Imemorija<T> {
    Cvor<T> *glava;
public:
    bool JePrazna() { return (glava == 0); }
    LancanaLista():glava(0){};
    void Dodaj(T info) { glava = new Cvor<T>(info, glava); }
    ...
}
```

MOST (engl. BRIDGE)

Primer 3.

33

```
...
T Uzmi () {
    T tmp = glava->info;
    Cvor<T> *p = glava;
    glava = glava->sled;
    delete p;
    return tmp;
}
~LancanaLista () {
    while (glava) {
        Cvor<T> *p = glava;
        glava = glava->sled;
        delete p;
    }
}
};
```

MOST (engl. BRIDGE)

Primer 3.

34

```
template<typename T>
class Niz:public Imemorija<T> {
    T *ptr;
    int topIdx;
    int trenVel;
    static const int DFLT_SIZE = 2;
    void Realociraj(){
        T *p = new T[trenVel << 1];
        for(int i = 0; i < trenVel; i++) p[i]= ptr[i];
        trenVel <= 1;
        delete [] ptr;
        ptr = p;
    }
public:
    . . .
};
```

MOST (engl. BRIDGE)

Primer 3.

35

public:

```
Niz():ptr(new T[DFLT_SIZE]),topIdx(0), trenVel(DFLT_SIZE){}
void Dodaj(T info) {
    if(topIdx == trenVel) Reallociraj();
    ptr[topIdx++] = info;
}

T Uzmi() { return ptr[--topIdx]; }

~Niz() { delete [] ptr; }
};
```

MOST (engl. BRIDGE)

Primer 3.

36

// AGREGACIJA

```
template<typename T,>
    template<typename> class Memorija >
class StekAgr {
    Memorija<T> mem;
public:
    StekAgr(in cap): mem(cap) {}
    void Push(T info){ mem.Dodaj(info); }
    T Pop() { return mem.Uzmi(); }
};

// Instanciranje sablonu
StekAgr<int,Niz> s;
s1.Push(3);
//-----
StekAgr<double, LancanaLista> s2;
s2.Push(4.2);
```

MOST (engl. BRIDGE)

Primer 3.

37

// NASLEDJIVANJE SABLONA

```
template< typename T, template<typename> class Memorija>
class StekNas: private Memorija<T>{
public:
    StekNas(int cap) : Memorija<T>(cap) {}
    void Push(T info) { Memorija<T>::Dodaj(info); }
    T Pop() { return Memorija<T>::Uzmi(); }
};
```

// Instanciranje sablona

```
StekNas<int,Niz> s;
s1.Push(3);
//-----
StekNas<double, LancanaLista> s2;
s2.Push(4.2);
```

Ime i klasifikacija

- Kompozitni (engl. Composite)**
- Strukturni projektni uzorak**

Drugo ime

- Sklop, Sastav**

Namena

- Komponuje objekte u strukturu stabla (hijerarhija celina – deo)**
- Omogućava klijentima da uniformno tretiraju**
 - Individualne objekte i**
 - Njihove kompozicije**

Kompozitni (engl. COMPOSITE)

1. Uvod

39

□ **Primenljivost**

- **Kompoziciju treba koristiti kada postoje hijerarhije celina-deo takve da su celina i deo iste vrste**
- **Klijent mogu da ignorišu razlike između kompozicije i pojedinih objekata**
- **Klijent zahteva da sa složenom celinom i delom komunicira na isti način**

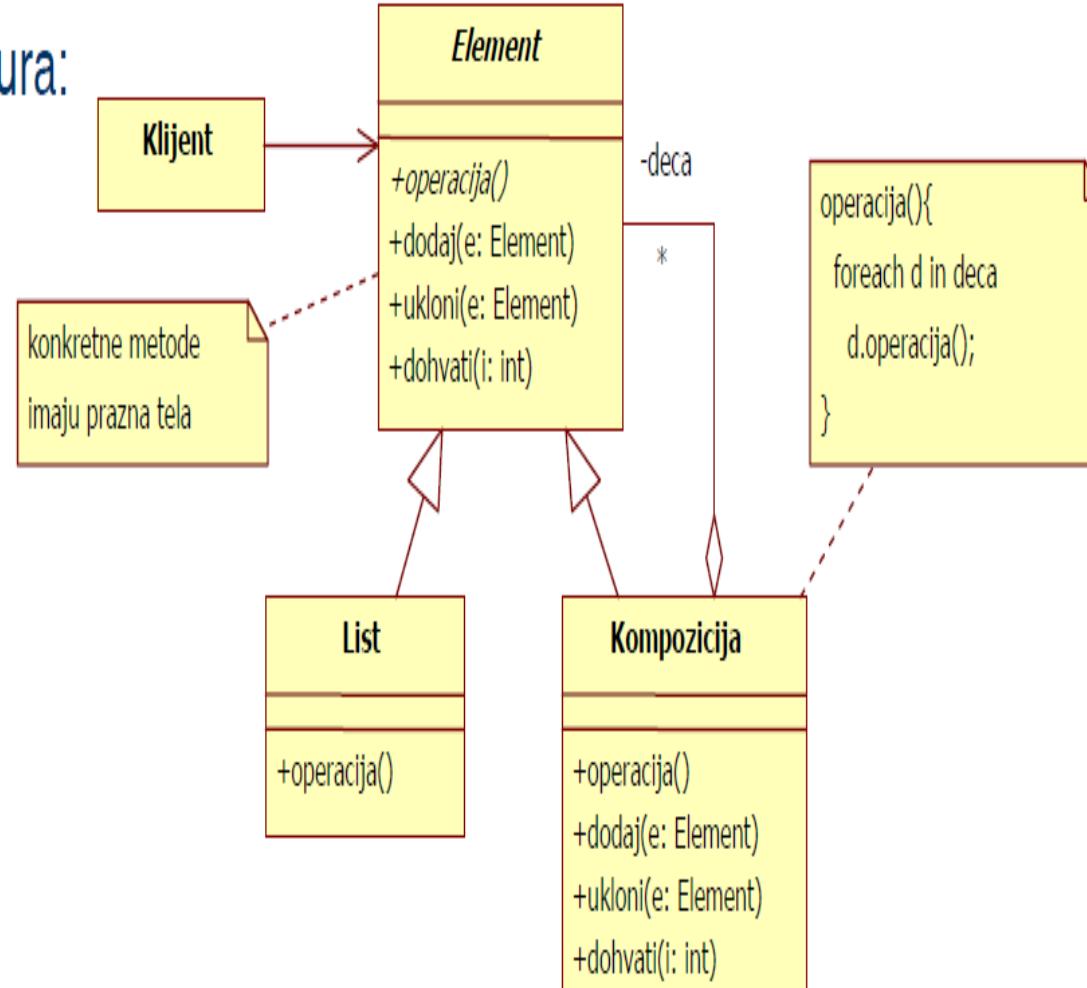
Kompozitni (engl. COMPOSITE)

1. Uvod

40

□ Struktura

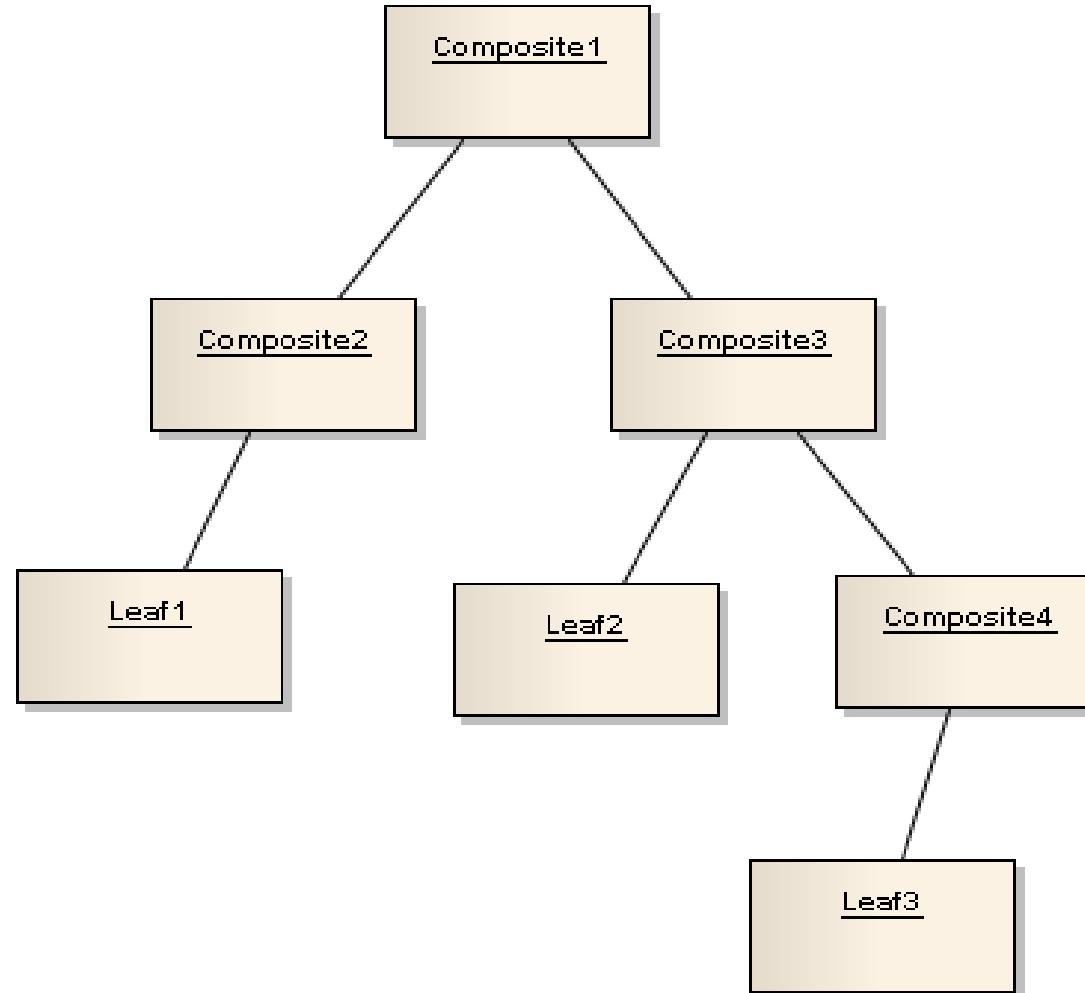
Struktura:



Kompozitni (engl. COMPOSITE)

1. Uvod

41



Kompozitni (engl. COMPOSITE)

1. Uvod

❑ Učesnici

42

❑ Element

- ❑ Deklariše zajednički interfejs za sve objekte u kompoziciji
- ❑ Implementira podrazumevano ponašanje zajedničko za sve klase
- ❑ Deklariše interfejs za pristupanje i upravljanje decom
- ❑ Implementira prazne metode za pristup i upravljanje decom (zbog listova)
- ❑ Opciono deklariše i implementira interfejs za pristup roditelju

❑ List

- ❑ Predstavlja individualne objekte – listove u stablu
- ❑ Definiše ponašanje primitivnih objekata u kompoziciji

❑ Kompozicija

- ❑ Definiše ponašanje objekata koji imaju „decu“
- ❑ Sadrži komponentu koja čuva „decu“
- ❑ Implementira operacije za pristup i upravljanje decom

❑ Klijent

Projektni uzorci Manipuliše objektima u kompoziciji kroz interfejs klase Element

27.11.2018.

❑ Saradnja

- ❑ **Klijent koristi interfejs apstraktne klase Element za komunikaciju sa objektima složene strukture**
 - ❑ Ako je primalac poruke (zahteva) list zahtev se izvršava neposredno
 - ❑ Ako je primalac poruke (zahteva) kompozicija, zahtev se izvršava i prosledjuje deci na dalje izvršavanja

❑ Posledice

- ❑ **Klijent na jedinstven načine tretira i listove (nezavisne objekte) i kompozicije**
- ❑ **Dodavanje novih elemenata je jednostavno**
- ❑ **Nije jednostavno ograničiti vrste elemenata koje kompozicije mogu da sadrže**

Kompozitni (engl. COMPOSITE)

1. Uvod

44

```
/* Komponenta */
class Komponenta{
public:
    Komponenta(std::string ime, double plata)
        : m_Ime(ime), m_plata (plata) {}
    virtual void getPlata(int nivo) const = 0;
protected:
    string m_Ime;
    double m_plata;
};

/* LIST */
class Radnik : public Komponenta{
public:
    Radnik(std::string ime , double plata): Komponenta(ime,plata){}
    void getPlata(int nivo) const{
        for(int j=0; j < nivo; ++j) cout << "\t";
        cout <<"Radnik : "<<m_Ime.c_str()<<,plata: "<<m_plata<<"$\n";
    }
};
```

Kompozitni (engl. COMPOSITE)

1. Uvod

45

```
/* KOMPOZITNI */
class Menadzer: public Komponenta{
public:
    Menadzer(string ime , double plata) : Komponenta(ime,plata) {}
    void add(Komponenta *cmp){ m_tim.push_back(cmp);}

    void getPlata(int nivo) const {
        for(int j=0; j < nivo; ++j) cout << "\t";
        cout <<"Menadzer : "<<m_Ime.c_str()<<,plata: "<<m_plata<<"$\n";
        if(!m_tim.empty()){
            for(int i=0; i < nivo; ++i) cout << "\t";
            cout << "Tim saradnika "<<m_Ime.c_str()<<":\n";
            ++nivo;
            for(list<Komponenta*>::const_iterator it = m_tim.begin();
                                            it != m_tim.end(); ++it)
                (*it)->getPlata(nivo);
        }
    }
private:
    list<Komponenta*> m_tim;
};
```

Kompozitni (engl. COMPOSITE)

1. Uvod

46

```
int main() {
    //Direktor
    Menadzer direktor ("Joksim Mitrovic", 120000.0);

    //Sefovi departmana
    Menadzer sefProizvodnje ("Petar Mitic", 90000.0);
    Menadzer glavniInzenjer ("Miroslava Stevanovic", 90000.0);
    Menadzer sefKontroleKvaliteta ("Danica Pekic", 80000.0);
    Menadzer sefProdaje ("Maksimilijan Topalovic", 75000.0);

    //Vodje timova odeljenja inzenjera
    Menadzer vodjaTimARD ("Jovana Stankovic", 70000.0);
    Menadzer vodjaTimaQA ("Aleksandar Veljovic", 70000.0);

    //Inzenjeri u odeljenju razvoja
    Radnik programer1 ("Andrija Djordjevic", 200000.0);
    Radnik programer2 ("Malica Tasic", 240000.0);
    Radnik tester ("Tomislav Stanimirovic", 130000.0);
    . . .
```

Kompozitni (engl. COMPOSITE)

1. Uvod

47

```
int main() {
    // Dodajemo neposredne saradnike direktoru
    direktor.add(&sefProizvodnje);
    direktor.add(&glavniInzenjer);
    direktor.add(&sefKontroleKvaliteta);
    direktor.add(&sefProdaje);

    // Dodajemo neposredne saradnike glavnog inzenjeru
    glavniInzenjer.add(&vodjaTimuRD);
    glavniInzenjer.add(&vodjaTimuQA);

    // Dodajemo neposredne saradnike vodji tima za istrazivanje i razvoj
    vodjaTimuRD.add(&programer1);
    vodjaTimuRD.add(&programer2);

    // Dodajemo neposredne saradnike vodji tima za osiguravanje kvaliteta
    vodjaTimuQA.add(&tester);
    cout << "Hijerarhija kompanije\n direktor i njegovi saradnici :\n\n";
    direktor.getPlata(0);
    cout << '\n';
}
```

Kompozitni (engl. COMPOSITE)

1. Uvod

48

```
C:\Windows\system32\cmd.exe
Hijerarhija kompanije
direktor i njegovi saradnici :

Menadzer : Joksim Mitrovic,plata: 120000$
Tim saradnika Joksim Mitrovic:
    Menadzer : Petar Mitic,plata: 90000$
    Menadzer : Miroslava Stevanovic,plata: 90000$
    Tim saradnika Miroslava Stevanovic:
        Menadzer : Jovana Stankovic,plata: 70000$
        Tim saradnika Jovana Stankovic:
            Radnik : Andrija Djordjevic,plata: 200000$
            Radnik : Malica Tosic,plata: 240000$
        Menadzer : Aleksandar Veljovic,plata: 70000$
        Tim saradnika Aleksandar Veljovic:
            Radnik : Tomislav Stanimirovic,plata: 130000$
    Menadzer : Danica Pekic,plata: 80000$
    Menadzer : Maksimilijan Topalovic,plata: 75000$


Press any key to continue . . .
```

Kompozitni (engl. COMPOSITE) Primer 2.

49

```
class DijagnostickiInterfejs {  
public: virtual void pokreniDijagnostiku(int nivo) = 0;  
};  
  
class KompozitniInterfejsDijagnostike : public DijagnostickiInterfejs{  
public:  
    void pokreniDijagnostiku(int nivo) {  
        if (!m_delovi.empty()) {  
            ++nivo;  
            for (int idx = 0; idx < nivo; ++idx) cout << "\t";  
            cout << "Dijagnostika komponenti\n";  
            for (list<DijagnostickiInterfejs*>::iterator  
                it = m_delovi.begin(); it != m_delovi.end(); ++it)  
                (*it)->pokreniDijagnostiku(nivo);  
        }  
    }  
    void dodajKomponentu(DijagnostickiInterfejs *ptrKomp) {  
        m_delovi.push_back(ptrKomp);  
    }  
protected:  
    list< DijagnostickiInterfejs *> m_delovi;  
};
```

Kompozitni (engl. COMPOSITE) Primer 2.

50

```
class DijagnostikaTelefona : public KompozitniInterfejsDijagnostike{
public:
    void pokreniDijanostiku() {
        cout << "Izvrsava se DijagnostikaTelefona..." << endl;
        KompozitniInterfejsDijagnostike::pokreniDijagnostiku();
    }
};

class DijagnostikaHardvera : public KompozitniInterfejsDijagnostike{
public:
    void pokreniDijagnostiku() {
        cout << "Izvrsava se DijagnostikaHardvera..." << endl;
        KompozitniInterfejsDijagnostike::pokreniDijagnostiku();
    }
};
```

Kompozitni (engl. COMPOSITE)

Primer 2.

51

```
class DijagnostikaMreze : public KompozitniInterfejsDijagnostike {
public:
    void pokreniDijagnostiku(int nivo) {
        for (int idx = 0; idx < nivo; ++idx) cout << "\t";
        cout << "Izvrsava se DijagnostikaMreze..." << endl;
        KompozitniInterfejsDijagnostike::pokreniDijagnostiku(nivo);
    }
};

class TouchScreenDijagnostika : public DijagnostickiInterfejs {
public:
    void pokreniDijagnostiku(int nivo) {
        for (int idx = 0; idx < nivo; ++idx) cout << "\t";
        cout << "Izvrsava se TouchScreenDijagnostika..." << endl;
    }
};
```

Kompozitni (engl. COMPOSITE)

Primer 2.

52

```
class OnOffKeyDijagnostika : public DijagnostickiInterfejs {  
public:  
    void pokreniDijagnostiku(int nivo) {  
        for (int idx = 0; idx < nivo; ++idx) cout << "\t";  
        cout << "Izvrsava se OnOffKeyDijagnostika..." << endl;  
    }  
};  
class WiFiDijagnostika : public DijagnostickiInterfejs {  
public:  
    void pokreniDijagnostiku(int nivo) {  
        for (int x = 0; x < nivo; ++x) cout << "\t";  
        cout << "Izvrsava se WiFiDijagnostika..." << endl;  
    }  
};  
class TriGDijagnostika : public DijagnostickiInterfejs {  
public:  
    void pokreniDijagnostiku(int nivo) {  
        for (int idx = 0; idx < nivo; ++idx) cout << "\t";  
        cout << "Izvrsava se TriGDijagnostika..." << endl;  
    }  
};
```

Kompozitni (engl. COMPOSITE) Primer 2.

53

```
int main() {
    DijagnostickiKompozitniInterfejs *dijagnostikaTelefona =
        new DijagnostikaTelefona();

    DijagnostickiKompozitniInterfejs *dijagnostikaHardvera =
        new DijagnostikaHardvera();

    DijagnostickiKompozitniInterfejs *dijagnostikaMreze =
        new DijagnostikaMreze();

    DijagnostickiInterfejs *touchScreenDijagnostika =
        new TouchScreenDijagnostika();

    DijagnostickiInterfejs *onOffKeyDijagnostika =
        new OnOffKeyDijagnostika();

    DijagnostickiInterfejs *wiFiDijagnostika = new WiFiDijagnostika();

    DijagnostickiInterfejs *triGDijagnostika = new TriGDijagnostika();
    . . .
    return 0;
}
```

Kompozitni (engl. COMPOSITE) Primer 2.

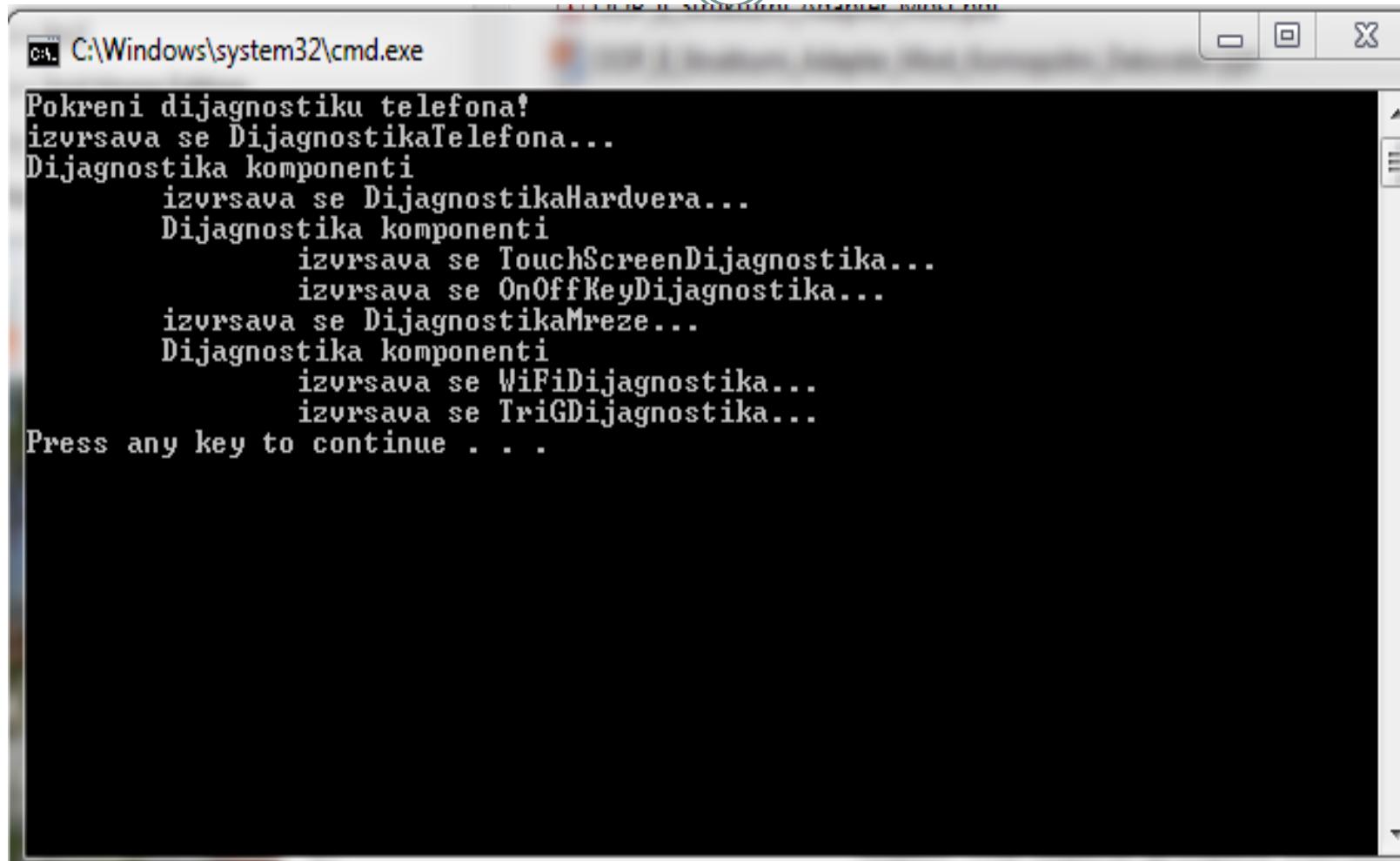
54

```
int main() {  
    . . . .  
  
    dijagnostikaHardvera->dodajKomponentu(touchScreenDijagnostika) ;  
    dijagnostikaHardvera->dodajKomponentu(onOffKeyDijagnostika) ;  
  
    dijagnostikaMreze->dodajKomponentu(wiFiDijagnostika) ;  
    dijagnostikaMreze->dodajKomponentu(triGDijagnostika) ;  
  
    dijagnostikaTelefona->dodajKomponentu(dijagnostikaHardvera) ;  
    dijagnostikaTelefona->dodajKomponentu(dijagnostikaMreze) ;  
  
    cout << "Pokreni dijagnostiku telefona!" << endl ;  
    dijagnostikaTelefona->pokreniDijagnostiku(0) ;  
    return 0;    return 0;  
}
```

Kompozitni (engl. COMPOSITE)

1. Uvod

55



A screenshot of a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The window contains the following text output:

```
Pokreni dijagnostiku telefona!
izvrsava se DijagnostikaTelefona...
Dijagnostika komponenti
    izvrsava se DijagnostikaHardvera...
    Dijagnostika komponenti
        izvrsava se TouchScreenDijagnostika...
        izvrsava se OnOffKeyDijagnostika...
    izvrsava se DijagnostikaMreze...
    Dijagnostika komponenti
        izvrsava se WiFiDijagnostika...
        izvrsava se TriGDijagnostika...
Press any key to continue . . .
```

❑ Ime i klasifikacija

- ❑ Dekorater (engl. Decorator)
- ❑ Strukturni projektni uzorak

❑ Drugo ime

- ❑ Dopuna, Omotač (eng. Wrapper)

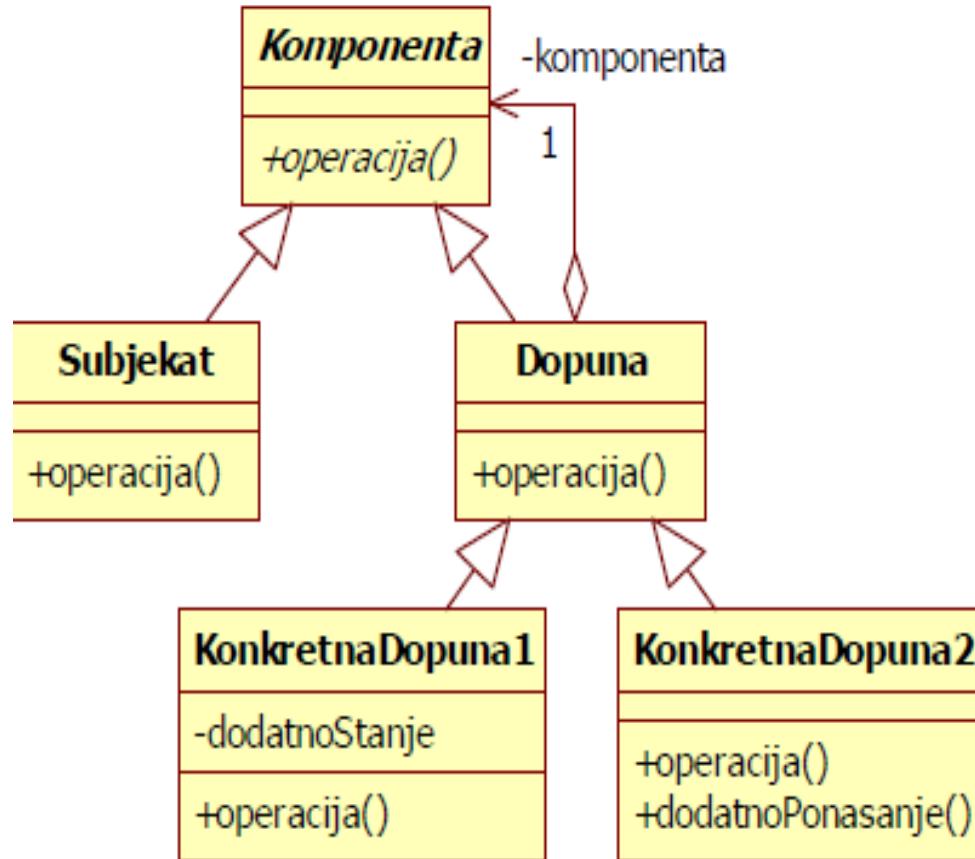
❑ Namena

- ❑ Dinamičko dodavanje odgovornosti (funkcionalnosti) nekom objektu
- ❑ Dinamička, fleksibilna alternativa izvodjenju
- ❑ Ne moraju svi objekti neke klase da budu „dekorisani“

❑ Primenljivost

- ❑ Potrebno je na transparentan način (vidljiv klijentu) dodavati funkcionalnosti (odgovornosti) objektima
- ❑ Proširivanje funkcionalnosti (odgovornosti) nasleđivanjem nije moguće, ili nije praktično zbog eksponencijalnog porasta broja klasa u hijerarhiji

□ Struktura



❑ Učesnici

❑ Komponenta

- ❑ Definiše interfejs za objekte kojim se dinamički dodaju odgovornosti, funkcionalnosti

❑ Subjekat

- ❑ Definiše klasu objekata kojima se dodaju odgovornosti

❑ Dekorater(Dopuna)

- ❑ Sadrži referencu na objekat klase Komponenta i nasleđuje interfejs klase Komponenta

❑ Konkretni Dekorater

- ❑ Implementira nove odgovornosti (funkcionalnosti) i dodaje ih objektu tipa Komponenta

❑ Saradnja

❑ Konkretni Dekorater

- ❑ Prosledjuje zahteve obuhvaćenom objektu tipa Komponenta
- ❑ Izvršava dodatne operacije pre ili posle prosledjivanja zahteva

❑ Posledice

❑ Prednosti

- ❑ Veća fleksibilnost od statičkog nasleđivanja – dinamičko vezivanje funkcionalnosti
- ❑ Izbegavanje eksplozije potklasa koje kombinuju dodatke

❑ Nedostaci

- ❑ Identitet dekoratera i dekorisanog objekta su različiti
- ❑ Nasledjeni atributi komponente ako postoje. Treba da se izbegavaju atributi u apstraktnoj komponenti.

Dekorater (engl. DECORATOR)

Primer 1. Dinamicko dodavanje osobina objektima

61

```
class BiblElem{ // Bibiopecka jedinica
public:
    BiblElem(int b) :m_brKopija(m_brKopija) {}

    void SetBrKopija(unsigned value) { m_brKopija = value; }

    int GetBrKopija() const{return m_brKopija; }

    virtual void BiblElemInfo(void)=0;

private:
    unsigned m_brKopija;
};
```

Dekorater (engl. DECORATOR)

Primer 1. Dinamicko dodavanje osobina objektima

62

```
Knjiga : public BiblElem{ // Konkretna bibliotecka jedinica - klasa #1
public:
    Knjiga(string autor, string naslov, int brKopija) :
BiblElem(brKopija), m_Autor(autor), m_Naslov(naslov) {}

    void BiblElemInfo() {
        cout<<"\nKnjiga ----- "<<endl;
        cout<<" Autor : "<<m_Autor<<endl;
        cout<<" Naslov : "<<m_Naslov<<endl;
        cout<<" Broj kopija : "<<GetBrKopija()<<endl;
    }

private:
    string m_Autor;
    string m_Naslov;
};
```

Dekorater (engl. DECORATOR)

Primer 1. Dinamicko dodavanje osobina objektima

63

```
class Video : public BiblElem{ // Konkretna bibliotecka jedinica - klasa  
#2  
public:  
    Video(string reziser, string naslov, int trajanje, int brKopija) :  
BiblElem(brKopija), m_Reziser(reziser), m_Naslov(naslov),  
m_Trajanje(trajanje){}  
  
    void BiblElemInfo(){  
        cout<<"\nVideo ----- "<<endl;  
        cout<<" Reziser : "<<m_Reziser<<endl;  
        cout<<" Naslov : "<<m_Naslov<<endl;  
        cout<<" Trajanje : "<<m_Trajanje<<" minuta"<<endl;  
        cout<<" Broj kopija : "<<GetBrKopija()<<endl;  
    }  
  
private:  
    string m_Reziser;  
    string m_Naslov;  
    unsigned m_Trajanje;  
};
```

Dekorater (engl. DECORATOR)

Primer 1. Dinamicko dodavanje osobina objektima

64

```
class Dekorater{ /* Osnovna klasa Dekorater */
public:
    Dekorater(BiblElem* ptr) : m_ptrBiblElem(ptr) {}
/* VAZNO: Interfejs dekoratera implementira osnovni interfejs
dekorisanih objekata */
    virtual void BiblElemInfo() {
        /* Poziv cisto virtuelne funkcije */
        m_ptrBiblElem->BiblElemInfo();
    }
    virtual int GetBrKopija() const {
        return m_ptrBiblElem->GetBrKopija();
    }
    virtual void SetBrKopija(unsigned brKopija) const {
        m_ptrBiblElem->SetBrKopija(brKopija);
    }
protected:
/* Pokazivac na element koji se dekorise, tj. kome se dinamicki dodaju
nove osobine */
    BiblElem* m_ptrBiblElem;
};
```

Dekorater (engl. DECORATOR)

Primer 1. Dinamicko dodavanje osobina objektima

65

```
class ZaIzdavanje : public Dekorater{
/* Izvedena klasa Dekorater omogucava da bibliotecka jedinica moze da
se izdaje korisnicima. Dodatna infomracija u odnosu na "obicnu"
bibliotecku jedinicu je lista korisnika koji su vec iznajmili neke od
primeraka bibliotecke jedinice. */
public:
    ZaIzdavanje(BiblElem* ptrBiblElem) : Dekorater(ptrBiblElem) {}

    void Pozajmi(string name){
        unsigned brKopija = m_ptrBiblElem->GetBrKopija();
        if(brKopija > 0){
            m_pozajmljivaci.push_back(name);
            m_ptrBiblElem->SetBrKopija(--brKopija);
        }
    }
    . . .
protected:
    /* Informacija koja omogucava "dekorisanje" - lista korisnika
    bibliotecke jedinice */
    list<string> m_pozajmljivaci;
};
```

Dekorater (engl. DECORATOR)

Primer 1. Dinamicko dodavanje osobina objektima

66

```
class ZaIzdavanje : public Dekorater{
/* Izvedena klasa Dekorater omogucava da bibliotecka jedinica moze da
se izdaje korisnicima. Dodatna infomracija u odnosu na "obicnu"
bibliotecku jedinicu je lista korisnika koji su vec iznajmili neke od
primeraka bibliotecke jedinice. */
public:
. . .
void Vrati(string name){
    list<string>::iterator it = m_pozajmljivaci.begin();
    while(it != m_pozajmljivaci.end()){
        if(*it == name) { m_pozajmljivaci.erase(it); break; }
        ++it;
    }
    m_ptrBiblElem->SetBrKopija(m_ptrBiblElem->GetBrKopija() + 1);
}
. . .
protected:
    list<string> m_pozajmljivaci;
};
```

Dekorater (engl. DECORATOR)

Primer 1. Dinamicko dodavanje osobina objektima

67

```
class ZaIzdavanje : public Dekorater{
/*. . . */
public:
    . . .
void BiblElemInfo()  {
    Dekorater::BiblElemInfo();
    cout<<" Broj pozajmljenih kopija : "
        << m_pozajmljivaci.size()<<endl;
    list<string>::iterator it = m_pozajmljivaci.begin();
    while(it != m_pozajmljivaci.end())
    {
        cout<<" Korisnik: "<<*it<<endl;
        ++it;
    }
}

protected:
    list<string> m_pozajmljivaci;
};
```

Dekorater (engl. DECORATOR)

Primer 1. Dinamicko dodavanje osobina objektima

68

```
int main(){ //Klijent
    Knjiga knjiga("Radoje Domanovic","Vodja",237);
    knjiga.BiblElemInfo();

/*Transforacija knjige u bibliotecki element koji moze da se izdaje */
    ZaIzdavanje knjigaZaIzd(&video);
    videoZaIzd.Pozajmi("Svetozar Stancic");
    videoZaIzd.BiblElemInfo();

/* Kreiranje biblioteckog elementa koji predstavlja video sadrzaj*/
    Video video("Slobodan Sijan", "Ko to tamo peva", 86, 24);
    video.BiblElemInfo();

/* Transformacija videa u bibliotecki element koji moze da se izdaje */
    ZaIzdavanje videoZaIzd(&video);
    videoZaIzd.Pozajmi("Petar Petrovic");
    videoZaIzd.Pozajmi("Nikola Nikolic");
    videoZaIzd.BiblElemInfo();

    return 0;
}
```

Dekorater (engl. DECORATOR)

Primer 1. Dinamicko dodavanje osobina objektima

69

A screenshot of a Windows Command Prompt window titled "cmd" with the path "C:\Windows\system32\cmd.exe". The window displays the output of a program. The output is organized into sections labeled "Knjiga ----", "Video ----", and "Video ----". Each section contains properties of an object, such as "Autor : Radoje Domanovic", "Naslov : Uvodja", etc. The "Video ----" sections also include "Broj kopija" and "Broj pozajmljenih kopija". At the end of the output, there is a prompt "Press any key to continue . . .".

```
Knjiga ----  
Autor : Radoje Domanovic  
Naslov : Uvodja  
Broj kopija : 237  
  
Video ----  
Reziser : Slobodan Sijan  
Naslov : Ko to tamo peva  
Trajanje : 86 minuta  
Broj kopija : 24  
  
Video ----  
Reziser : Slobodan Sijan  
Naslov : Ko to tamo peva  
Trajanje : 86 minuta  
Broj kopija : 22  
Broj pozajmljenih kopija : 2  
Korisnik: Petar Petrovic  
Korisnik: Nikola Nikolic  
Press any key to continue . . .
```

Dekorater (engl. DECORATOR)

Primer 2. “Dekorisanje objekata” bez brisanja njihovih osobina

70

```
class AbsOsnovna{
public:
    virtual void Fun() const = 0;
    virtual string Ime() const { return "AbsOsnovna"; }
    virtual ~AbsOsnovna() {}
};

class Izvedena_A : public AbsOsnovna{
public:
    virtual void Fun() const {
        cout<<"Izvedena_A: implementacija Fun"<<endl;
    }
    virtual string Ime() const { return "Izvedena_A"; }
    void Fun_A() const {
        cout<<"Izvedena_A: implementacija Fun_A"<<endl;
    }
    virtual ~Izvedena_A() {}
};
```

Dekorater (engl. DECORATOR)

Primer 2. “Dekorisanje objekata” bez brisanja njihovih osobina

71

```
class Izvedena_B : public AbsOsnovna{
public:
    virtual void Fun() const {
        cout<<"Izvedena_B: implementacija Fun"<<endl;
    }
    virtual string Ime() const { return "Izvedena_B"; }
    void Fun_B() const {
        cout<<"Izvedena_B: implementacija Fun_B"<<endl;
    }
    virtual ~Izvedena_B(){}
};
```

Dekorater (engl. DECORATOR)

Primer 2. “Dekorisanje objekata” bez brisanja njihovih osobina

72

```
class Dekorater : public AbsOsnovna{
    AbsOsnovna *m_ptrDekorisani;
public:
    Dekorater(AbsOsnovna *ptr) :m_ptrDekorisani(ptr) {}

    virtual void Fun() const {
        std::cout << "Dekorater dodaje funkcionalnost objektu klase "
        <<m_ptrDekorisani->Ime().c_str()<<std::endl;

    // Postojeća funkcionalnost objekta
    m_ptrDekorisani->Fun();
}
};
```

Dekorater (engl. DECORATOR)

Primer 2. “Dekorisanje objekata” bez brisanja njihovih osobina

73

```
int _tmain(int argc, _TCHAR* argv[]) {  
  
    Izvedena_A a;  
    Izvedena_B b;  
  
    AbsOsnovna *p1 = new Dekorater(&a);  
    p1->Fun();  
    delete p1;  
  
    AbsOsnovna *p2 = new Dekorater(&b);  
    p2->Fun();  
    delete p2;  
  
}
```