

Objektno orijentisano programiranje u C++-u

Projektni uzorci

1

PROJEKTNI UZORCI PONAŠANJA OBJEKATA

**POSETILAC
PODSETENIK**

❑ Ime i klasifikacija

- ❑ Posetilac (engl. Visitor)
- ❑ Projektni uzorak ponašanja objekata

❑ Namena

- ❑ Dodavanje „virtuelne funkcije“ u postojeću hijerarhiju klase
- ❑ Omogućava dodavanje nove operacije bez izmene hijerarhije klase elemenata nad kojima se ta operacija izvršava
- ❑ Posetilac omogucava razdvajanje algoritama od strukture objekta (za razliku od dekoratera ne menja strukturu objekta). Praktični rezultat ovog razdvajanja je mogućnost dodavanja novih operacija na postojeću strukturu objekta bez potrebe da se struktura objekta modifikuje.

❑ Primenljivost

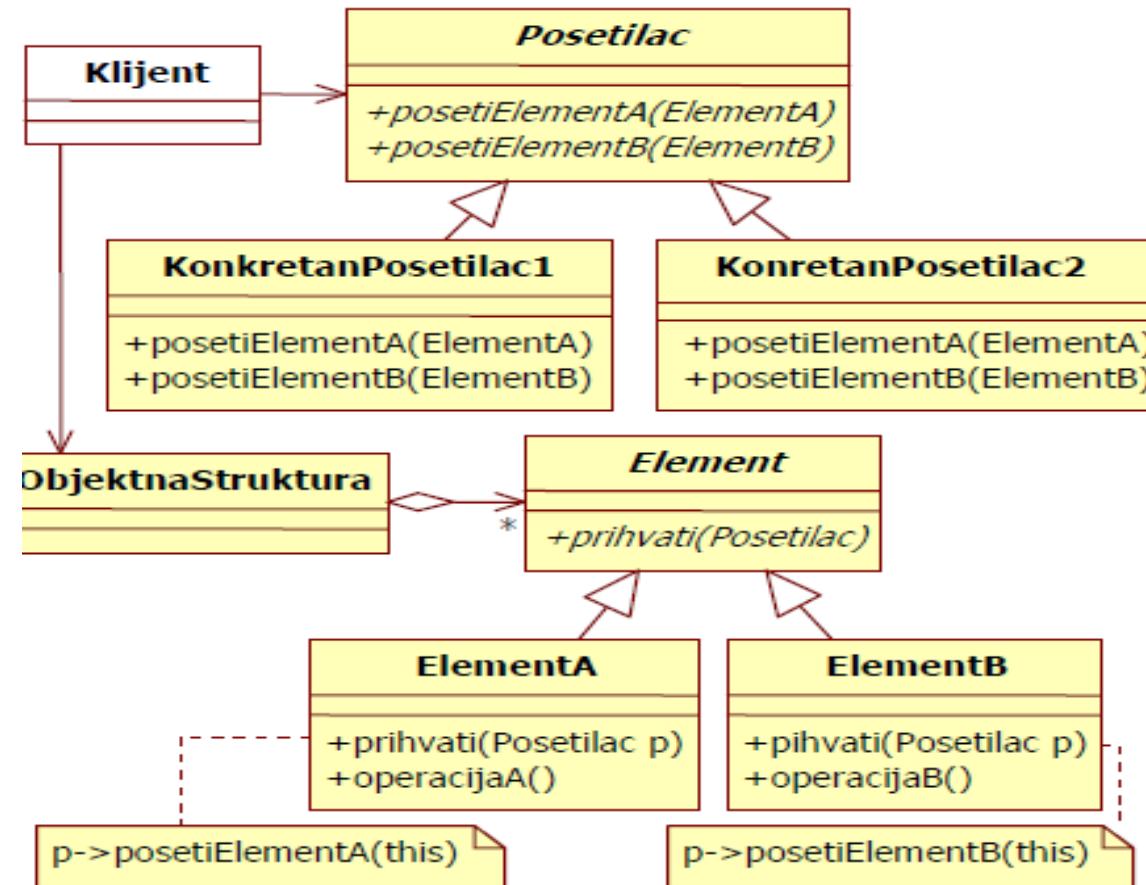
- ❑ Nove operacije se dodaju bez promene strukture objekta
- ❑ Dodavanje funkcija u biblioteku klase za koju ili ne postoji izvorni kod ili izvorni kod ne može da se menja.
- ❑ Više, međusobno nepovezanih operacija treba izvršavati nad objektima hijerarhije klase
- ❑ Posetilac omogućava povezivanje operacija jednog tipa nad različitim klasama unutar klase posetioca
- ❑ Hijerarhija klasa objekata nad kojima se operacije vrše se retko menjaju ali se često dodaju nove operacije

POSETILAC(engl. VISITOR)

1. Uvod

4

□ Struktura



❑ Učesnici

❑ **Posetilac**

- ❑ Po jedna operacija poseti() za svaku potklasu klase Element
- ❑ Deklaracija opracije poseti() identificuje odgovarajuću potklasu

❑ **Konkretni posetilac**

- ❑ Implementira sve operacije koje deklariše posetilac
- ❑ Svaka operacija je implementirana za odgovarajuću potklasu u hijrarchiji klasa elemenata
- ❑ Dodavanje novih operacija znači dodavanje novih klasa Konkretnih posetilaca

❑ Učesnici

❑ Element

- ❑ Deklariše operaciju prihvati(Posetilac&) (ili docekaj(Posetilac&))

❑ ElementA, Element B

- ❑ Implementira operaciju prihvati(Posetilac&) (ili docekaj(Posetilac&)) tako što poziva posetilac.poseti(*this). Zahvaljujući preklapanju funkcija, na osnovu tipa reference *this se poziva odgovarajuća funkcija posetioca.

❑ Objektna struktura

- ❑ Definiše interfejs koji omogucava klijentu da obiđe elemente i salje im posetioca
- ❑ Može da bude stablo, kolekcija itd.

❑ Saradnja

❑ Klijent

- ❑ Stvara objekat konkretnog posetioца
- ❑ Obilazi strukturu objekata posećujući svaki njen element
- ❑ Svakom elementu prosledi posetioца

❑ Element

- ❑ Poziva operaciju posetioца kojom mu traži da ga poseti

❑ Posledice

- ❑ **Olakšava dodavanje novih operacija kada dodavanje novih klasa elemenata nije često niti lako**
- ❑ **Grupiše srodne operacije (i razdvaja različite)**
- ❑ **Poseduje mogućnost akumuliranja stanja prilikom posete elementima**

❑ Povezani uzorci

- ❑ **Kompozicija – posetilac može da se koristi da bi se neka operacija primenila na strukturu objekata definisanu kao kompozicija**
- ❑ **Iterator se često koristi zajedno sa Posetiocem – služi da omogući obilazak strukture čijim se elementima šalju posetioci**
- ❑ **Interpreter – posetilac se može iskoristiti da obavi interpretaciju**

POSETILAC(engl. VISITOR)

Primer 1. Klasično rešenje problema dodavanja novih operacija

9

```
/* HIJERARHIJA KLASA KOJE IMPLEMENTIRAJU POJEDINE DELOVE NEKOG DOKUMENTA */
/* Jedna od implementiranih opracija je ToHTML koja ima zadatak da odgovarajuci deo dokumenta konvertuje u HTML tagovima */

class IDeoDokumenta {
protected:
    string m_str;
public:
    IDeoDokumenta(string s):m_str(s){}
    virtual ~IDeoDokumenta(){}
    void SetTekst(string s) { m_str = s; }
    string GetTekst() const { return m_str; }
    virtual string ToHTML() const = 0;
};
```

POSETILAC(engl. VISITOR)

Primer 1. Klasično rešenje problema dodavanja novih operacija

10

```
class Tekst : public IDeoDokumenta {
public:
    Tekst(string s) : IDeoDokumenta(s) {}
    virtual string ToHTML() const { return GetTekst(); }
};

class BoldTekst : public IDeoDokumenta {
public:
    BoldTekst(string s) : IDeoDokumenta(s) {}
    virtual string ToHTML() const { return "<b>" + GetTekst() +
"</b>"; }
};

class Hyperlink : public IDeoDokumenta {
    string m_url;
public:
    Hyperlink(string s, string url) : IDeoDokumenta(s), m_url(url) {}
    virtual string ToHTML() const { return "<a href=\"" + m_url + "\">" +
GetTekst() + "</a>"; }
};
```

POSETILAC(engl. VISITOR)

Primer 1. Klasično rešenje problema dodavanja novih operacija

11

/ U klasi dokument se u odredjenoj strukturi, u ovom slučaju to je lista, cuvaju delovi dokumenta. Konverzija celog dokumenta se odvija tako što se struktura obidje i svaki element se konvertuje */*

```
class Dokument: public list<IDeoDokumenta*> {
public:
    typedef list<IDeoDokumenta*> base_t;
    string ToHTML() const {
        string output = "";
        for (base_t::const_iterator it = begin(); it != end(); ++it) {
            output += (*it)->ToHTML();
        }
        return output;
    }
};
```

POSETILAC(engl. VISITOR)

Primer 1. Klasično rešenje problema dodavanja novih operacija

12

```
/* Posle izvesnog vremena zelimo da dodamo operaciju konverzije  
delova dokumenta u u LaTeX format */  
/* Na zaost dodavanje ove operacije podrazumeva da se kroz citavu  
hijerarhiju delova dokumenta doda jos jedna virtuelna funkcija I da  
se cela hijerarhija ponovo prevede */  
  
class IDeoDokumenta {  
protected:  
    string m_str;  
public:  
    IDeoDokumenta(string s):m_str(s){}  
    virtual ~IDeoDokumenta(){}  
    void SetTekst(string s) { m_str = s; }  
    string GetTekst() const { return m_str; }  
    virtual string ToHTML() const = 0;  
    virtual string ToLaTeX() const = 0;  
};
```

POSETILAC(engl. VISITOR)

Primer 1. Klasično rešenje problema dodavanja novih operacija

13

```
class Tekst : public IDeoDokumenta {
public:
    Tekst(string s) : IDeoDokumenta(s) {}
    virtual string ToHTML() const { return GetTekst(); }
    virtual string ToLaTeX() const { return GetTekst(); }
};

class BoldTekst : public IDeoDokumenta {
public:
    BoldTekst(string s) : IDeoDokumenta(s) {}
    virtual string ToHTML() const { return "<b>" + GetTekst() +
"</b>"; }
    virtual string ToLaTeX() const { return "\textbf{" + GetTekst() +
"}"; }
};
```

POSETILAC(engl. VISITOR)

Primer 1. Klasično rešenje problema dodavanja novih operacija

14

```
class Hyperlink : public IDeoDokumenta {
    string m_url;
public:
    Hyperlink(string s, string url): IDeoDokumenta(s), m_url(url){}
    virtual string ToHTML() const { return "<a href=\"" + m_url + "\">"
+ GetTekst() + "</a>"; }
    virtual string ToLaTeX() const { return "\\href{" + m_url + "}{"
GetTekst() + "}"; }
};
```

POSETILAC(engl. VISITOR)

Primer 1. Klasično rešenje problema dodavanja novih operacija

15

```
class Dokument: public list<IDeoDokumenta*> {
public:
    typedef list<IDeoDokumenta*> base_t;
    string ToHTML() const {
        string output = "";
        for (base_t::const_iterator it = begin(); it != end(); ++it) {
            output += (*it)->ToHTML();
        }
        return output;
    }
    string ToLaTeX() const {
        string output = "";
        for (base_t::const_iterator it = begin(); it != end(); ++it) {
            output += (*it)->ToLaTeX();
        }
        return output;
    }
};
```

POSETILAC(engl. VISITOR)

Primer 1. Klasično rešenje problema dodavanja novih operacija

16

```
Int main()
{
    Dokument d;
    /* Kreiranje dokumenta */
    d.push_back(new BoldTekst("NASLOV"));
    d.push_back(new Hyperlink("", "www.tmp.rs"));
    d.push_back(new Tekst("Tekst"));

    /* Konvertovanje dokumenta u HTML format */
    string s_HTML = d.ToHTML();
    cout<< s_HTML << endl;

    /* Konvertovanje dokumenta u LaTeX format */
    string s_LaTeX = d.ToLaTeX();
    cout<< s_LaTeX << endl; */

    return 0;
}
```

POSETILAC(engl. VISITOR)

Primer 2. Primena POSETIOCA u dodavnju novih operacija

17

```
class IPosetilac;
/* Hijerarhija klase kojoj se NIKAD (ili vrlo retko) dodaju nove
klase, ali se zato cesto dodaju nove virtuelne funkcije */
class IDeoDokumenta {
protected:
    string m_str;
public:
    IDeoDokumenta(string s):m_str(s){}
    virtual ~IDeoDokumenta(){}
    void SetTekst(string s) { m_str = s; }
    string GetTekst() const { return m_str; }
    virtual void Docekaj(IPosetilac&) const = 0;
};
```

POSETILAC(engl. VISITOR)

Primer 2. Primena POSETIOCA u dodavnju novih operacija

18

```
/* Svaki element teksta mora da bude sposoban da doceka posetioce  
razlicitih tipova */

class Tekst : public IDeoDokumenta {
public:
    Tekst(string s) : IDeoDokumenta(s) {}
    void Docekaj(IPosetilac &p) const;
};

class BoldTekst : public IDeoDokumenta {
public:
    BoldTekst(string s) : IDeoDokumenta(s) {}
    void Docekaj(IPosetilac &p) const;
};

class Hyperlink : public IDeoDokumenta {
    string m_url;
public:
    Hyperlink(string s, string url) : IDeoDokumenta(s), m_url(url) {}
    string GetUrl() const { return m_url; }
    void Docekaj(IPosetilac &p) const;
}, Projektni uzorci
```

POSETILAC(engl. VISITOR)

Primer 2. Primena POSETIOCA u dodavnju novih operacija

19

```
/* Struktura koja cuva pojedine elemente mora da bude sposobna da  
dozvoli da svaki element doceka posetioce razlicitih tipova */  
  
class Dokument: public list<IDeoDokumenta*> {  
public:  
    typedef list<IDeoDokumenta*> base_t;  
  
    void Docekaj (IPosetilac &p) const {  
        for (base_t::const_iterator it = begin(); it != end(); ++it) {  
            (*it)->Docekaj (p);  
        }  
    }  
};
```

POSETILAC(engl. VISITOR)

Primer 2. Primena POSETIOCA u dodavnju novih operacija

20

/ Hjerarhija klasa posetilaca. Dodavanje nove klase posetioca ekvivalentno je dodavanju nove virtuelne funkcije u prethodnu hjerarhiju klasa. */*

```
class IPosetilac{
protected:
    string m_output;
public:
    IPosetilac(): m_output("") {}
    virtual ~IPosetilac() {}
    /* Po jedna funkcija za svaku klasu iz hjerarhije elemenata strukture koja treba da se posecuje. Dodavanje nove klase u hjerarhiji elemenata podrezumeva da cela hjerarhija posetioca mora da se promeni dodavanjem nove virtuelne funkcije i, naravno, da se posle dodavanja ponovo prevede. */
    virtual void Poseti(const Tekst&) = 0;
    virtual void Poseti(const BoldTekst&) = 0;
    virtual void Poseti(const Hyperlink&) = 0;
    string Get() const { return m_output; }
};
```

POSETILAC(engl. VISITOR)

Primer 2. Primena POSETIOCA u dodavnju novih operacija

21

```
class HTMLPosetilac: public IPosetilac{
public:
    HTMLPosetilac():IPosetilac(){}
    virtual ~HTMLPosetilac(){}
    virtual void Poseti(const Tekst &deo) {
        m_output += deo.GetTekst();
    }
    virtual void Poseti(const BoldTekst &deo) {
        m_output += "<b>" + deo.GetTekst() + "</b>";
    }
    virtual void Poseti(const Hyperlink &deo) {
        m_output += "<a href=\"" + deo.GetUrl() + "\">" +
                    deo.GetTekst() + "</a>";
    }
};
```

POSETILAC(engl. VISITOR)

Primer 2. Primena POSETIOCA u dodavnju novih operacija

22

/*Neophodno je implementiranje funkcije docekaj u svakoj od klase elemenata dokumenta. Zasto?*/

```
void Tekst::Docekaj(IPosetilac &p) const {
    return p.Poseti(*this);
}
void BoldTekst::Docekaj(IPosetilac &p) const {
    return p.Poseti(*this);
}
void Hyperlink::Docekaj(IPosetilac &p) const {
    return p.Poseti(*this);
}
```

POSETILAC(engl. VISITOR)

Primer 2. Primena POSETIOCA u dodavnju novih operacija

23

```
/*Neophodno je implementiranje funkcije docekaj u svakoj od klasa
elemenata dokumenta. Zasto?*/
/* Zato sto je *this razlicitog tipa u svakoj od navedenih funkcija:
➤ u Tekst::Docekaj *this je tipa Tekst&
➤ u BoldTekst::Docekaj *this je tipa BoldTekst&
➤ u Hyperlink::Docekaj *this je tipa Hyperlink&

void Tekst::Docekaj(IPosetilac &p) const {
    return p.Poseti(*this);
}
void BoldTekst::Docekaj(IPosetilac &p) const {
    return p.Poseti(*this);
}
void Hyperlink::Docekaj(IPosetilac &p) const {
    return p.Poseti(*this);
}
```

POSETILAC(engl. VISITOR)

Primer 2. Primena POSETIOCA u dodavnju novih operacija

24

```
/* Dodavanje nove operacije nad svim elementima sada se svodi na jednostavno dodavanje nove klase u hijerarhiji posetilaca. */

/* "OPERACIJA" konvertovanja dokumenta u LaTeX format */
/* Obratite paznju na to da nikakva druga izmena nije potrebna sem dodavanje sledece klase. */
class LaTeXPosetilac: public IPosetilac{
public:
    LaTeXPosetilac(): IPosetilac(){}
    virtual ~LaTeXPosetilac(){}
    virtual void Poseti(const Tekst &deo) {
        m_output += deo.GetTekst();
    }
    virtual void Poseti(const BoldTekst &deo) {
        m_output += "\\textbf{" + deo.GetTekst() + "}";
    }
    virtual void Poseti(const Hyperlink &deo) {
        m_output += "\\href{" + deo.GetUrl() + "}{" +
                    deo.GetTekst() + "}";
    }
};
```

POSETILAC(engl. VISITOR)

Primer 2. Primena POSETIOCA u dodavnju novih operacija

25

```
int main()
{
    Dokument d;
/* Kreiranje dokumenta */
    d.push_back(new BoldTekst("NASLOV"));
    d.push_back(new Hyperlink("", "www.tmp.rs"));
    d.push_back(new Tekst("Tekst"));

/* Konvertovanje dokumenta u HTML format */
    HTMLPosetilac *pHTML = new HTMLPosetilac();
    d.Docekaj(*pHTML);
    string s_HTML = pHTML->Get();
    cout<< s_HTML << endl;

/* Konvertovanje dokumenta u LaTeX format */
    LaTeXPosetilac *pLaTeX = new LaTeXPosetilac();
    d.Docekaj(*pLaTeX);
    string s_LaTeX = pLaTeX->Get();
    cout<< s_LaTeX << endl;

    return 0;
}
```

POSETILAC(engl. VISITOR)

Primer 3. CRTP i POSETILAC: Pojednostavljenje hijerarhije klasa
koje docekuju posetioce

26

```
class IPosetilac;  
/*Hijerarhija klasa kojoj se NIKAD (ili vrlo retko) dodaju nove  
klase, ali se zato cesto dodaju nove virtuelne funkcije */  
  
class IDeoDokumenta {  
protected:  
    string m_str;  
public:  
    IDeoDokumenta(string s):m_str(s){}  
    virtual ~IDeoDokumenta(){}  
    void SetTekst(string s) { m_str = s; }  
    string GetTekst() const { return m_str; }  
    virtual void Docekaj(IPosetilac&) const = 0;  
};
```

POSETILAC(engl. VISITOR)

Primer 3. CRTP i POSETILAC: Pojednostavljenje hijerarhije klasa
koje docekuju posetioce

27

/* CURIOUSLY RECURRING TEMPLATE PATTERN */

```
template<typename Izvedena>
class DeoDokZaPosetu : public IDeoDokumenta{
public:
    DeoDokZaPosetu(string s):IDeoDokumenta(s) {}
    virtual ~DeoDokZaPosetu() {}
    virtual void Docekaj(IPosetilac&) const;
};
```

POSETILAC(engl. VISITOR)

Primer 3. CRTP i POSETILAC: Pojednostavljenje hijerarhije klasa
koje docekuju posetioce

28

/* CURIOUSLY RECURRING TEMPLATE PATTERN */

```
class Tekst : public DeoDokZaPosetu<Tekst> {
public:
    Tekst(string s) : DeoDokZaPosetu<Tekst>(s) {}
};

class BoldTekst : public DeoDokZaPosetu<BoldTekst> {
public:
    BoldTekst(string s) : DeoDokZaPosetu<BoldTekst>(s) {}
};

class Hyperlink : public DeoDokZaPosetu<Hyperlink> {
    string m_url;
public:
    Hyperlink(string s, string url) : DeoDokZaPosetu<Hyperlink>(s),
    m_url(url){}
    string GetUrl() const { return m_url;}
};
```

POSETILAC(engl. VISITOR)

Primer 3. CRTP i POSETILAC: Pojednostavljenje hijerarhije klasa
koje docekuju posetioce

29

/* CURIOUSLY RECURRING TEMPLATE PATTERN */

/* Klasa koja cuva pojedine elemente je ostala nepromenjena */

```
class Dokument: public list<IDeoDokumenta*> {
public:
    typedef list<IDeoDokumenta*> base_t;
    void Docekaj (IPosetilac &p) const {
        for (base_t::const_iterator it = begin(); it != end(); ++it) {
            (*it)->Docekaj(p);
        }
    }
};
```

POSETILAC(engl. VISITOR)

Primer 3. CRTP i POSETILAC: Pojednostavljenje hijerarhije klasa
koje docekuju posetioce

30

```
/* HIJERARHIJA KLASA POSETILACA JE OSTALA NEPROMENJENA */
class IPosetilac{
protected:
    string m_output;
public:
    IPosetilac(): m_output("") {}
    virtual ~IPosetilac() {}

    virtual void Poseti(const Tekst&) = 0;
    virtual void Poseti(const BoldTekst&) = 0;
    virtual void Poseti(const Hyperlink&) = 0;
    string Get() const { return m_output; }
};
```

POSETILAC(engl. VISITOR)

Primer 3. CRTP i POSETILAC: Pojednostavljenje hijerarhije klasa
koje docekuju posetioce

31

/* HIJERARHIJA KLASA POSETILACA JE OSTALA NEPROMENJENA */

```
class HTMLPosetilac: public IPosetilac{
public:
    HTMLPosetilac():IPosetilac(){}
    virtual ~HTMLPosetilac(){}
    virtual void Poseti(const Tekst &deo) {
        m_output += deo.GetTekst();
    }
    virtual void Poseti(const BoldTekst &deo) {
        m_output += "<b>" + deo.GetTekst() + "</b>";
    }
    virtual void Poseti(const Hyperlink &deo) {
        m_output += "<a href=\"" + deo.GetUrl() + "\">" +
                    deo.GetTekst() + "</a>";
    }
};
```

POSETILAC(engl. VISITOR)

Primer 3. CRTP i POSETILAC: Pojednostavljenje hijerarhije klasa
koje docekuju posetioce

32

/* HIJERARHIJA KLASA POSETILACA JE OSTALA NEPROMENJENA */

```
class LaTeXPosetilac: public IPosetilac{
public:
    LaTeXPosetilac():IPosetilac(){}
    virtual ~LaTeXPosetilac(){}
    virtual void Poseti(const Tekst &deo) {
        m_output += deo.GetTekst();
    }
    virtual void Poseti(const BoldTekst &deo) {
        m_output += "\\textbf{" + deo.GetTekst() + "}";
    }
    virtual void Poseti(const Hyperlink &deo) {
        m_output += "\\href{" + deo.GetUrl() + "}{" +
                    deo.GetTekst() + "}";
    }
};
```

POSETILAC(engl. VISITOR)

Primer 3. CRTP i POSETILAC: Pojednostavljenje hijerarhije klasa
koje docekuju posetioce

33

```
/* UMETNO implementacija funkcije Docekaj za svaki pojedinacni  
element . . .*/
```

```
void Tekst::Docekaj(IPosetilac &p) const {  
    return p.Poseti(*this);  
}
```

```
void BoldTekst::Docekaj(IPosetilac &p) const {  
    return p.Poseti(*this);  
}
```

```
void Hyperlink::Docekaj(IPosetilac &p) const {  
    return p.Poseti(*this);  
}
```

```
/* . . . SADA POSTOJI JEDINSTVENA IMPLEMENTACIJE FUNKCIJE Docekaj ZA  
CELU HIJERARHIJU ELEMENATA DOKUMENTA */
```

```
template<typename Izvedena>  
void DeoDokZaPosetu<Izvedena>::Docekaj(IPosetilac &p) const {  
    return p.Poseti(static_cast<const Izvedena&>(*this));  
}
```

POSETILAC(engl. VISITOR)

Primer 3. CRTP i POSETILAC: Pojednostavljenje hijerarhije klasa
koje docekuju posetioce

34

```
/* UPOTREBA POSETILACA JE ISTA KAO U PRETHODNOM PRIMERU */
int main()
{
    Dokument d;
    /* Kreiranje dokumenta */
    d.push_back(new BoldTekst("NASLOV"));
    d.push_back(new Hyperlink("", "www.tmp.rs"));
    d.push_back(new Tekst("Tekst"));

    /* Konvertovanje dokumenta u HTML format */
    HTMLPosetilac *pHTML = new HTMLPosetilac();
    d.Docekaj(*pHTML);
    string s_HTML = pHTML->Get();
    cout << s_HTML << endl;

    /* Konvertovanje dokumenta u LaTeX format */
    LaTeXPosetilac *pLaTeX = new LaTeXPosetilac();
    d.Docekaj(*pLaTeX);
    string s_LaTeX = pLaTeX->Get();
    cout << s_LaTeX << endl;
}
```

static_cast

- Konverzija koja koristi “*sati type information*”, statičke informacije o tipu, poznate u toku kompajliranja
- Obuhvata:
 - Konverzije numeričkih tipova bez poruke o gubitku informacija;
 - Konverzije u hijerarhiji klasa (bezbednost konverzije garantuje programer)
 - Konverzije pomoću unitarnog konstruktora
 - Konverzije pomoću operatora konverzije
 - Konverzije void* u odgovarajući tip i obrnuto.
- Static_cast konverzija nije bezbedna ko dynamic_cast konverzija, zato što ne vrši provere u toku izvršavanja (run time type check)
- Dynamic_cast dvosmislenog pointera vraća 0 kao rezultat, dok static_cast vraća pokazivač koji ukazuje na „narušenu“ memorijsku strukturu i potencijalno kasnije izazove run time error.
- Dynamic_cast je bezbedan ali funkcioniše samo nad pokazivačima i referencama a neophodan je run time type check.

STATIC_CAST operator

36

```
/*PRIMER 1*/  
class Osnovna {};  
class Izvedena : public Osnovna {};  
  
void f(Osnvna* pOsn) {  
/* BEZBEDNO SAMO AKO pIzv zaista ukazuje na objekat tipa Izvedena */  
    Izvedena* pIzv = static_cast<Izvedena*>(pOsn);  
}
```

STATIC_CAST operator

37

```
/*PRIMER 2*/
class Osnovna {
    int m_i;
public:
    Osnovna(int i) :m_i(i){}
    virtual void Test() const{ std::cout << m_i << "\n"; }
};

class Izvedena : public Osnovna {
    int m_j;
public:
    Izvedena(int i, int j) :Osnovna(i), m_j(j){}
    void Test() const{
        Osnovna::Test();
        std::cout << m_j << "\n";
    }
};
```

STATIC_CAST operator

38

```
/*PRIMER 2 nastavak*/  
void f(Osnovna* pOsn) {  
    /* Ako pOsn zaista ukazuje na objekat tipa Izvedena obe konverzije  
    daju isti rezultat. Isti rezultat imaju i kada je pOsn = 0 */  
    Izvedena* pIzv1 = dynamic_cast<Izvedena*>(pOsn);  
    Izvedena* pIzv2 = static_cast<Izvedena*>(pOsn);  
    /* Ako pOsn ukazuje na objekat tipa Osnovna dynamic_cast vraca 0, dok  
    static_cast vraca pokazivac tipa Izvedena na nekonstruisani objekat  
    tipa Izvedena */  
    pIzv1->Test();  
    pIzv2->Test();  
}  
  
int main(){  
    Osnovna a(3);  
    Izvedena b(3, 4);  
    f(&b); /* Funkcija Test pozvana preko oba pokazivaca daje  
istи rezultat */  
    f(&a); /* RUN TIME ERROR */
```

STATIC_CAST operator

39

```
int main() {  
  
    /* Konverzije numerickih tipova */  
  
    int n = static_cast<int>(3.14); /* NEMA UPOZORENJA O GUBITKU  
INFORMACIJE */  
  
    n = 5.12; /* POSTOJI UPOZORENJE O GUBITKU INFORMACIJE */  
    std::cout << "n = " << n << '\n';  
  
}
```

STATIC_CAST operator

40

```
template<typename T>
struct A {
    T m_i, m_j;
    explicit A(T i, T j = 10) : m_i(i), m_j(j) {}

};

int main() {

    /* Konverzije prilikom inicijalizacije */
    A<int> a1 = 10; /* NIJE DOZVOLJENO ZBOG EKSPLICITNO KONSTRUKTORA */
    /* PORUKA KOMPAJLERA: ne postoji odgovarajuci konstruktor*/

    A<int> a1 = static_cast<A<int>>(10); /* EKSPLICITNA KONVERZIJA int
u A<int> objekat : Operator konverzije koristi konstruktor */

}
```

STATIC_CAST operator

41

```
template<typename T>
struct A {
    T m_i, m_j;
    explicit A(T i, T j = 10) : m_i(i), m_j(j) {}
    operator int() const { return m_i; }

};

int main() {
    /* NAREDNE LINIJE KODA DOZVOLJENE OD STRANE KOMPJALERA SAMO AKO
    POSTOJI OPERATOR KONVERZIJE*/
    int ia1 = a1;
    int ia2 = static_cast<int>(a1); /*static_cast koristi korisnicki
    definisani operator konverzije*/
}
```

STATIC_CAST operator

42

```
struct Osnovna {};
        struct Izvedena : Osnovna {};
```

```
int main(){
    /* STATIC konverzija u pokazivac ili referencu izvedene klase
    (downcast) */
    Izvedena oI;
    Osnovna& rO = oI; /* Implicitna konverzija (upcast) */
    Izvedena& rI = static_cast<Izvedena&>(rO); /* Neophodna eksplicitna
    konverzija (downcast) */

    /* INVERTOVANJE IMPLICITNE KONVERZIJE */
    void* nv = &n;
    int* ni = static_cast<int*>(nv);
    std::cout << "*ni = " << *ni << '\n';

}
```

STATIC_CAST operator

43

```
template<typename T>
struct A {
    T m_i, m_j;
    explicit A(T i, T j = 10) : m_i(i), m_j(j) {}
    operator int() const { return m_i; }
    enum E{eA=1, eB};
};

int main(){

/* void* u bilo koji tip pokazivaca */
A<int>::E e = A<int>::E::eA;
void* voidp = &e;
A<int>* p = static_cast<A<int>*>(voidp);
std::cout << "A<int> *p->m_i = " << p->m_i << "\n";
std::cout << "A<int> *p->m_j = " << p->m_j << "\n";
}
```

❑ **dynamic_cast**

- ❑ Konverzija pokazivača ili reference polimorfнog tipa u pokazivač ili referencu drugog polimorfнog tipa.
- ❑ Provera korektnosti konverzije se vrši u toku izvršavanja
- ❑ Pokušaj konverzije u pokazivač tipa koji nije tip objekta na koji konvertovani pokazivač ukazuje daje NULL kao rezultat.
- ❑ Pokušaj konverzije u referencu tipa koji nije tip objekta na koji konvertovana referencia ukazuje prosleđuje std::bad_cast izuzetak

DYNAMIC_CAST operator

45

```
struct A {  
    virtual void f() { }  
};  
struct B : public A { };  
struct C { };  
  
void f () {  
    A a;  
    B b;  
  
    A* ap = &b;  
    B* b1 = dynamic_cast<B*> (&a); /* b1 = NULL, zato sto je a tipa A,  
a ne tipa B */  
    B* b2 = dynamic_cast<B*> (ap); /* b2 = &b */  
    C* c = dynamic_cast<C*> (ap); /* c = NULL, zato sto ap ukazuje na  
objekat tipa B, a ne tipa C */  
  
    A& ar = dynamic_cast<A&> (*ap); /* Korektna konverzija */  
    B& br = dynamic_cast<B&> (*ap); /* Korektna konverzija */  
    C& cr = dynamic_cast<C&> (*ap); /* Konverzija prosledjuje  
std::bad_cast */  
}
```

Ime i klasifikacija

- Podsetnik (engl. Memento)**
- Projektni uzorak ponašanja objekata**

Namena

- Snima i spolja čuva unutrašnje stanje nekog objekta**
- Omogućava da se objekti mogu vratiti u dato stanje**

Drugo ime

- Oznaka (engl. Token)**

❑ Primenljivost

- ❑ Kada treba snimiti stanje nekog objekta da bi se to stanje kasnije restauriralo
- ❑ Direktan interfejs za dobijanje stanja narušava inkapsulaciju objekta koji se čuva u podsetniku

PODSETNIK(engl. MEMENTO)

1. Uvod

48

□ Struktura

```
napraviPodsetnik(){
```

```
    p = new Podsetnik();
```

```
    p.postaviStanje(stanje);
```

```
    return p;
```

```
}
```

```
restaurirajStanje(p:Podsetnik){
```

```
    stanje=p.dohvatiStanje();
```

```
}
```

Cuvar

Subjekat

-stanje

+napraviPodsetnik()

+restaurirajStanje(p: Podsetnik)

-podsetnik

Podsetnik

-stanje

+dohvatiStanje()

+postaviStanje()

❑ Učesnici

❑ Podsetnik

- ❑ Čuva unutrašnje stanje objekta klase Subjekat
- ❑ Ne dozvoljava pristup stanju drugim objektima osim objektu klase Subjekat
- ❑ Ima dva interfejsa: Čuvar vidu uski interfejs, Subjekat vidi široki interfejs

❑ Subjekat

- ❑ Kreira objekat klase Podsetnik koji sadrži snimak njegovog tekućeg stanja
- ❑ Ima diskreciono pravod da odluči koji deo stanja se čuva
- ❑ Koristi objekat klase Podsetnik da restaurira stanje

❑ Čuvar

- ❑ Odgovoran za bezbedno čuvanje objekta Podsetnik
- ❑ Ne ispituje i ne koristi stanje objekta Podsetnik
- ❑ Prosledjuje objekat klase Podsetnik objektu klase Subjekat za restauraciju stanja

❑ Saradnja

- ❑ Objekat klase Subjekat kreira objekat klase Podsetnik i predaje ga Čuvaru na čuvanje

❑ Povezani uzorci

- ❑ U uzorku Iterator podsetnik može da se koristi za čuvanje podataka o tekućem elementu
- ❑ U uzorku Komanda Podsetnik može da se koristi za čuvanja stanja poništavanih operacija

PODSETNIK(engl. MEMENTO)

Primer 1. Jednostavni podsetnik

51

```
/* Podsetnik je odgovoran za cuvanje stanja Subjekta. Poseduje dva
interfejsa. Cuvar vidi uzi interfejs*/
/* Subjekat vidi siri interfejs, omogucava pristup svim podacima
neophodnim da rekonstruise sopstveno stanje */

class Podsetnik {
private:
/* Samo Subjekat, cije stanje treba da se cuva, moze da kreira
podsetnik */
friend class Subjekat;
Podsetnik(const string& stanje) :m_stanje(stanje) {}
~Podsetnik(){}
void SetStanje(const string& stanje){ m_stanje = stanje; }
string GetStanje() const { return m_stanje; }
string m_stanje;
};
```

PODSETNIK(engl. MEMENTO)

Primer 1. Jednostavni podsetnik

52

```
/* Subjekat odluce da kreira podsetnik za neko svoje interno stanje  
i odluce kada da se vrati na sacuvano stanje */

class Subjekat{
public:
    Subjekat(const string& stanje) :m_stanje(stanje) {}
    ~Subjekat(){}
    void VratiSeNa(const Podsetnik* ptrPodsetnik) {
        m_stanje = ptrPodsetnik->GetStanje();
    }
    /* Jednom sacuvano stanje ne sme da se menja*/
    const Podsetnik* KreirajPodsetnik() const {
        return new Podsetnik(m_stanje);
    }
    /* Interfejs koji omogucava "normalan" rad subjekta */
    void SetStanje(const string& stanje){ m_stanje = stanje; }
    string GetStanje() const { return m_stanje; }
    void Prikazi() const { cout << m_stanje << endl; }
private:
    string m_stanje;
};
```

PODSETNIK(engl. MEMENTO)

Primer 1. Jednostavni podsetnik

53

/* Cuvar je odgovoran samo za cuvanje podsetnika. Sve ostale operacije nad podsetnikom su zabranjene */

```
class Cuvar{
public:
    Cuvar() {}
    ~Cuvar() {}
    void SetPodsetnik(const Podsetnik *ptrPod) { m_pPod = ptrPod; }
    const Podsetnik* GetPodsetnik() const { return m_pPod; }
private:
    const Podsetnik *m_pPod;
};
```

PODSETNIK(engl. MEMENTO)

Primer 1. Jednostavni podsetnik

54

/* Upotreba podsetnika - klijent */

```
int _tmain(int argc, _TCHAR* argv[]){
```

```
    Subjekat* pSub = new Subjekat("Staro stanje");  
    pSub->Prikazi();
```

/*Kreiranje cuvara koji treba da sacuva Podsetnik */

```
Cuvar* pCuvar = new Cuvar();  
pCuvar->SetPodsetnik(pSub->KreirajPodsetnik());
```

/*Promena stanja*/

```
pSub->SetStanje("Novo stanje");  
pSub->Prikazi();
```

/* Restauracija stanja */

```
pSub->VratiSeNa(pCuvar->GetPodsetnik());  
pSub->Prikazi();
```

```
return 0;
```

```
}
```

PODSETNIK(engl. MEMENTO)

Primer 2. Podsetnik kompleksnijih struktura

55

```
class Podsetnik {  
    /* Klasa Stek, cije stanje ovaj podsetnik treba da cuva  
    ja jedina klasa koja sme da kreira podsetnik */  
  
    friend class Stek;  
    int *m_pNiz, m_brElem;  
    Podsetnik(int* pNiz, int n) {  
        m_pNiz = new int[m_brElem = n];  
        for (int i = 0; i < m_brElem; i++) m_pNiz[i] = pNiz[i];  
    }  
    public:  
  
    /* U ovoj implementaciji, klasa Stek, koja kreira objekat podsetnik,  
    ne moze da ga brise vec to radi Cuvar i zato destruktor mora da bude  
    javni */  
  
    ~Podsetnik() { delete m_pNiz; }  
};
```

PODSETNIK(engl. MEMENTO)

Primer 2. Podsetnik kompleksnijih struktura

56

```
class Stek {
    int m_pNiz[10], m_top;
public:
    /* Standardni interfejs steka */
    Stek(){ m_top = -1; }
    void push(int in) { m_pNiz[++m_top] = in; }
    int pop() { return m_pNiz[m_top--]; }
    bool isEmpty() { return m_top == -1; }

    Podsetnik* kreirajPodsetnik() {
        return new Podsetnik(m_pNiz, m_top + 1);
    }

    void VratiNa(Podsetnik* ptrPodsetnik) {
        m_top = ptrPodsetnik->m_brElem - 1;
        for (int i = 0; i < ptrPodsetnik->m_brElem; i++)
            m_pNiz[i] = ptrPodsetnik->m_pNiz[i];
    }

    void Prikazi() const { /* prikaz sadrzaja steka */ }
};
```

PODSETNIK(engl. MEMENTO)

Primer 2. Podsetnik kompleksnijih struktura

57

```
int _tmain(int argc, _TCHAR* argv[]) {  
  
    /* U ovom primeru Cuvar je f-ja main */  
    Stek m_s;  
    for (int i = 0; i < 5; i++) m_s.push(i);  
    m_s.Prikazi();  
    Podsetnik* ptrPod = m_s.kreirajPodsetnik();  
    for (int i = 5; i < 10; i++) m_s.push(i);  
    while (!m_s.isEmpty()) cout << m_s.pop() << ' '; cout << endl;  
  
    m_s.VratiNa(ptrPod);  
  
    /* Subjekat, Stek, kreira podsetnik ali ga brise Cuvar, koji je u  
    ovom slucaju, funkcija main. */  
    delete ptrPod_1;  
}
```

PODSETNIK(engl. MEMENTO)

Primer 2. Podsetnik sa pametnim pokazivačima

58

```
class TipSubjekta; /* Forward deklaracija tipa subjekta */

class Podsetnik{
private:
    /* Subjekat kreira podsetnik */
    friend Subjekat;
    Podsetnik(const TipSubjekta& stanje) : m_stanje(stanje) {}

    /* SmartPtr<Podsetnik> brie podsetnik */
    friend SmartPtr<Podsetnik>;
    virtual ~Podsetnik() {}

    TipSubjekta m_stanje;
};
```

PODSETNIK(engl. MEMENTO)

Primer 2. Podsetnik sa pametnim pokazivačima

59

```
/* Sablon klase cuvara podsetnika */

template< class TPodsetnik >
class CuvarPodsetnika{
private:
    typedef list<SmartPtr<TPodsetnik> >  ListaPodsetnika_t;
    ListaPodsetnika_t m_lista;
public:
    void push(SmartPtr<TPodsetnik> m) { m_lista.push_back(m); }
    SmartPtr<TPodsetnik> pop() {
        if (m_lista.size() < 1)
            throw runtime_error("Prazna lista podsetnika");
        SmartPtr<TPodsetnik> res = m_lista.front();
        m_lista.pop_front();
        return res;
    }
};
```

PODSETNIK(engl. MEMENTO)

Primer 2. Podsetnik sa pametnim pokazivačima

60

/* Klasa Subjekat koja pravi podsetnik */

```
class TipSubjekta{
private:
    string m_s;
public:
    TipSubjekta(string s) :m_s(s){}
    void Set(string s){m_s = s;}

    typedef Podsetnik<TipSubjekta> Podsetnik_t;
    typedef CuvarPodsetnika<Podsetnik_t> CuvarPodsetnika_t;

    void Prikazi() const { cout<< m_s<<endl; }

    SmartPtr<Podsetnik_t> KreirajPodsetnik(){
        return SmartPtr<Podsetnik_t>(new Podsetnik_t(*this));
    }

    void VratiNaPodsetnik(SmartPtr<Podsetnik_t> podsetnik){
        *this = podsetnik->m_stanje;
    }
}, Projektni uzorci
```

PODSETNIK(engl. MEMENTO)

Primer 2. Podsetnik sa pametnim pokazivačima

61

```
int _tmain(int argc, _TCHAR* argv[])
{
    TipSubjekta s("string1");
    /* Kreiranje cuvara podsetnika*/
    TipSubjekta::CuvarPodsetnika_t cS;

    /* Kreiranje i prosledjivanje Podsetnika Cuvaru da ga cuva */
    cS.push(s.KreirajPodsetnik());

    s.Set("string2");

    /* Uzimanje podsetnika od cuvara */
    s.VratiNaPodsetnik(cS.pop());
    s.Prikazi();

    /*Svim podsetnicima se pristupa preko "pametnih" pokazivaca, koji
    su zaduzeni za njihovo automatsko brisanje */

    return 0;
}
```

PODSETNIK(engl. MEMENTO)

Primer 3. CRTP i Generički Podsetnik sa pametnim pokazivačima

62

```
template<typename>
class AbsSubjekat; /* Forward deklaracija sablona */

/* Sablon podsetnika */
template <class TipSubjekta>
class Podsetnik{
private:
    friend AbsSubjekat<TipSubjekta>;
    friend SmartPtr<Podsetnik>;

    TipSubjekta m_stanje;

    Podsetnik(const TipSubjekta& stanje) : m_stanje(stanje) {}
    virtual ~Podsetnik() {}
};


```

PODSETNIK(engl. MEMENTO)

Primer 3. CRTP i Generički Podsetnik sa pametnim pokazivačima

63

```
/* Sablon klase cuvara podsetnika */
template< class TPodsetnik >
class CuvarPodsetnika{
private:
    typedef list<SmartPtr<TPodsetnik> > ListaPodsetnika_t;
    ListaPodsetnika_t m_lista;
public:
    void push(SmartPtr<TPodsetnik> m) { m_lista.push_back(m); }
    SmartPtr<TPodsetnik> pop() {
        if (m_lista.size() < 1) throw runtime_error("Prazna lista
podsetnika");
        SmartPtr<TPodsetnik> res = m_lista.front();
        m_lista.pop_front();
        return res;
    }
};
```

PODSETNIK(engl. MEMENTO)

Primer 3. CRTP i Generički Podsetnik sa pametnim pokazivačima

64

/* CRTP apstrakne klase subjekta implementira KREIRANJE I VRACANJE NA PODSETNIK */

```
template<typename Izvedena>
class AbsSubjekat{
public:
    typedef typename Podsetnik<Izvedena> Podsetnik_t;
    typedef typename CuvarPodsetnika< Podsetnik_t> CuvarPodsetnika_t;

    /* Staticke konverzije u sledecim funkcijama su bezbedne
       zahvaljujuci CRTP-u */

    SmartPtr<Podsetnik_t> KreirajPodsetnik() {
        return SmartPtr<Podsetnik_t>(
            new Podsetnik_t(static_cast<const Izvedena&>(*this)));
    }

    void VratiNaPodsetnik(SmartPtr<Podsetnik_t> podsetnik) {
        static_cast<Izvedena&>(*this) = podsetnik->m_stanje;
    }
};
```

PODSETNIK(engl. MEMENTO)

Primer 3. CRTP i Generički Podsetnik sa pametnim pokazivačima

65

```
/* Dodavanje nove klase koja moze da kreira i da se vrati na  
podsetnik */  
  
class KlasaA:public AbsSubjekat<Subjekat>{  
private:  
    string m_s;  
public:  
    KlasaA(string s):m_s(s){}  
    void Set(string s){m_s = s;}  
    void Prikazi() const { cout<< m_s<<endl; }  
    /* Operacija AbsSubjekat::KreirajPodsetnik koristi konstruktor  
kopije koji kompjajler pravi */  
    /* Operacija AbsSubjekat::VratiNaPodsetnik koristi operator dodele  
vrednosti koji kompjajler pravi */  
};
```

PODSETNIK(engl. MEMENTO)

Primer 3. CRTP i Generički Podsetnik sa pametnim pokazivačima

66

```
/* Dodavanje novog sablona klase koja moze da kreira i da se vrati na  
podsetnik */  
  
template <class ValueType>  
class CT_SmplDynArray :  
    public AbsSubjekat< CT_SmplDynArray<ValueType> >{  
public:  
    /* Konstruktori */  
    CT_SmplDynArray(): m_Size(0), m_Capacity(0), m_pArray(0) {}  
    CT_SmplDynArray(size_t InitCapacity): m_Size(0),  
        m_Capacity(InitCapacity){ m_pArray = new Value_t[m_Capacity];}  
  
    /* Konstruktor kopije koristi operacija KreirajPodsetnik iz  
    AbsSubjekat */  
    CT_SmplDynArray(const CT_SmplDynArray &Other):  
        m_Size(Other.m_Size), m_Capacity(Other.m_Capacity) {  
        m_pArray = new Value_t[m_Capacity];  
        std::copy(Other.m_pArray, Other.m_pArray + m_Capacity, m_pArray);  
    }
```

PODSETNIK(engl. MEMENTO)

Primer 3. CRTP i Generički Podsetnik sa pametnim pokazivačima

67

```
/* Dodavanje novog sablona klase koja moze da kreira i da se vrati na
podsetnik */
template <class ValueType>
class CT_SmplDynArray :
    public AbsSubjekat< CT_SmplDynArray<ValueType> >{
public:
    /* Destruktor */
    virtual ~CT_SmplDynArray() { delete [] m_pArray; }
    /* Operator dodele vrednosti. Podsetimo se da ga operacija
VratiNaPodsetnik iz AbsSubjekat koristi */
    CT_SmplDynArray<Value_t>& operator=(const CT_SmplDynArray<Value_t>&
Other) {
        if(this != &Other) {
            delete [] m_pArray;
            m_Capacity = Other.m_Capacity;
            m_Size = Other.m_Size;
            m_pArray = new Value_t[m_Capacity];
            std::copy(Other.m_pArray, Other.m_pArray + m_Capacity, m_pArray);
        }
        return *this;
    }
```