# C++
# Odlomci iz knjige u pripremi
# -
# Enciklopedija

**Saša Malkov**

# 8 Enciklopedija

## 8.1 Zadatak

Napisati program koji korisniku prikazuje podatke pohranjene u bazi podataka enciklopedije. Podržani tipovi podataka su *tekst*, *slika*, *zvuk* i *film*. Korisnik pokreće program *Enciklopedija* navodeći redni broj podatka, a program:

- prikazuje na standardnom izlazu podatke sa datim rednim brojem;
- prikazuje na standardnom izlazu redne brojeve, naslove i tipove svih povezanih podataka i
- u slučaju da podatak nije tekst, binarni sadržaj podatka zapisuje u datoteci sa datim imenom.

Argumenti programa se navode u obliku:

```
Enciklopedija <id> [<bindat>]
```

Na primer:

```
Enciklopedija 42 bin.dat
Enciklopedija 38
```

Ukoliko se navede naziv datoteke, a podatak sa navedenim rednim brojem je tekst, program obaveštava korisnika da datoteka nije upotrebljavana. Ukoliko se ne navede naziv datoteke, a radi se o podatku koji ima binarni sadržaj (slika, zvuk ili film), potrebno je ispisati raspoložive informacije uz napomenu da je potrebno navesti naziv datoteke.

Na raspolaganju je biblioteka za upotrebu baze podataka enciklopedije (zaglavlje `BazaPodataka.h`) u kojoj su definisani tip `Podatak` i klasa `BazaPodataka`:

```
typedef map<string,string> Podatak;

class BazaPodataka {
public:
    static bool PročitajPodatak(
            int id, Podatak& x, string& tip
            );
```

```
        static void PročitajPovezane(
                int id, vector<int>& povezani
                );
};
```

Jedan `Podatak` se sastoji od kolekcije parova oblika (naziv atributa, vrednost atributa). Sve vrednosti atributa su zapisane tekstualno pa se po potrebi moraju kovertovati u cele ili realne brojeve. Ukoliko je u pitanju tekstualni podatak, opisan je atributima sa nazivima `id`, `naslov`, `tekst`, `autor`. Slike su opisane atributima sa nazivima `id`, `naslov`, `slika`, `napomena`, `širina`, `visina`, `autor`. Zvuk je opisan atributima sa nazivima `id`, `naslov`, `zvuk`, `napomena`, `trajanje`, `autor`. Filmovi su opisani atributima sa nazivima `id`, `naslov`, `film`, `napomena`, `trajanje`, `širina`, `visina`, `autor`.

Bazi podataka se pristupa posredstvom metoda klase `BazaPodataka`. Statički metod `PročitajPodatak` čita podatak sa rednim brojem `id` i upisuje ga u objekat `x`. Pri tome se niska sa opisom tipa podatka (koja može biti „`tekst`“, „`slika`“, „`zvuk`“ ili „`film`“) upisuje u parametar `tip`. Metod `PročitajPovezane` čita redne brojeve podataka koji imaju neke veze sa podatkom sa datim rednim brojem `id` i upisuje ih u niz `povezani`.

### *Cilj zadatka*

Rešavanjem ovog zadatka pokazaćemo:

- kako se formiraju hijerarhije klasa;
- kako se ponašanje spušta niz ili podiže uz hijerarhiju;
- kako se dinamički prave objekti hijerarhije klasa;
- neke elemente UML-a primenjene na analizu i projektovanje.

## 8.2 Rešavanje zadatka

Pri rešavanju zadatka aktivnosti ćemo jasno podeliti na tri faze:

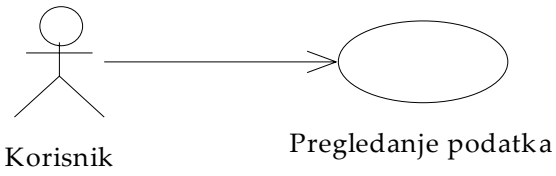
- analizu;
- projektovanje i
- implementaciju.

Najpre ćemo analizirati problem i do detalja precizno definišemo šta želimo da program radi. Zatim ćemo definisati klase i njihovo ponašanje, uzdržavajući se od bavljenja internom strukturom klasa i implementacijama metoda. Za opisivanje poslova i klasa upotrebljavaćemo dijagrame slučajeva i dijagrame klase u skladu sa UML-om.

Rešavanje će biti izloženo u nekoliko koraka:

### *Korak 1 - Analiza slučajeva upotrebe*

Počinjemo od slučajeva upotrebe. Program se upotrebjava na sasvim jednostavan način i ovaj deo analize je praktično trivijalan. Funkcionalnost programa se može predstaviti sasvim jednostavnim dijagramom:




Naš jedini slučaj upotrebe programa možemo detaljnije opisati:

**Naziv**:
Pregledanje podatka.

**Opis**:
Korisnik pregleda podatak sa datim rednim brojem.

**Preduslov**:
Podatak postoji u bazi podataka.

**Pauslov**:
Sadržaj baze podataka ostaje neizmenjen.

**Tok akcija**:

1. Korisnik pokreće program iz komandne linije navodeći redni broj podatka i naziv datoteke u koju se zapisuje binarni sadržaj.
2. Proverava se ispravnost podataka koje je naveo korisnik.
3. Iz baze podataka se čita traženi podatak.
4. Ako čitanje nije uspelo prekida se rad programa.
5. Korisniku se prikazuju podaci.
6. Čitaju se informacije o podacima koji imaju veze sa traženim podatkom.
7. Prikazuju se informacije o povezanim podacima, uređene po abecednom poretku po naslovu.
8. Ako postoji, binarni sadržaj se upisuje u datoteku. Ukoliko nije navedena datoteka, korisniku se prikazuje odgovarajuće obaveštenje.

### *Prototip izveštaja*

Da bi ponašanje program bilo tačno opisano potrebno je da odredimo u kom obliku se tražene informacije prikazuju korisniku. To možemo učiniti pisanjem prototipova izveštaja.

Prototip izveštaja za tekstualne podatke:

```
> Enciklopedija 42

Lav (tekst 42)
--------------
Lav (Leo ili Panthera leo) je velika snažna mačka iz porodice
Felidae, druga po veličini od velikih mačaka (posle tigra).
...

Pera Perić

Povezani podaci:
 - Lav (slika 47)
 - Lav u lovu (film 48)
 - Tigar (tekst 73)
 - Urlik lava (zvuk 46)
```

Prototip izveštaja za slike (prikazan tekst predstavlja sadržaj napomene):

```
> Enciklopedija 47 slika.dat

Lav (slika 47)
--------------
Odrasli primerak mužjaka u prirodnom okruženju.

Dimenzije: 1024 x 768

Fotko Fotkić

Povezani podaci:
 - Lav (tekst 42)
 - Lav u lovu (film 48)
 - Urlik lava (zvuk 46)
```

Prototip izveštaja za zvučne zapise (prikazan tekst predstavlja sadržaj napomene):

```
> Enciklopedija 46 zvuk.dat

Urlik lava (zvuk 46)
--------------------
Upozoravajući urlik lava spremnog da brani svoju teritoriju.

Trajanje: 21s

Sima Snimić

Povezani podaci:
 - Lav (slika 47)
 - Lav (tekst 42)
 - Lav u lovu (film 48)
```

Prototip izveštaja za slučaj filmova (prikazan tekst predstavlja sadržaj napomene):

```
> Enciklopedija 48

Lav u lovu (film 48)
--------------------
Lavica lovi antilopu. Obratite pažnju na pomoć koju joj pružaju
ostali lavovi onemogućavajući antilopi bekstvo.

Dimenzije: 1024 x 768
Trajanje: 1m 12s

Žika Žikić

Povezani podaci:
 - Lav (slika 47)
 - Lav (tekst 42)
 - Urlik lava (zvuk 46)
```

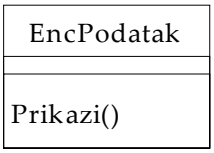
## *Korak 2 - Analiza i projektovanje prikazivanja podataka*

### *Osnovna ideja – klasa EncPodatak*

Osnovna složenost predstavljenog problema odnosi se na različitu prirodu podataka koji se mogu čitati iz enciklopedije. Sam program ima prilično jednostavnu strukturu, koju bismo mogli, u grubim crtama i bez ikakvih provera grešaka, opisati ovako:

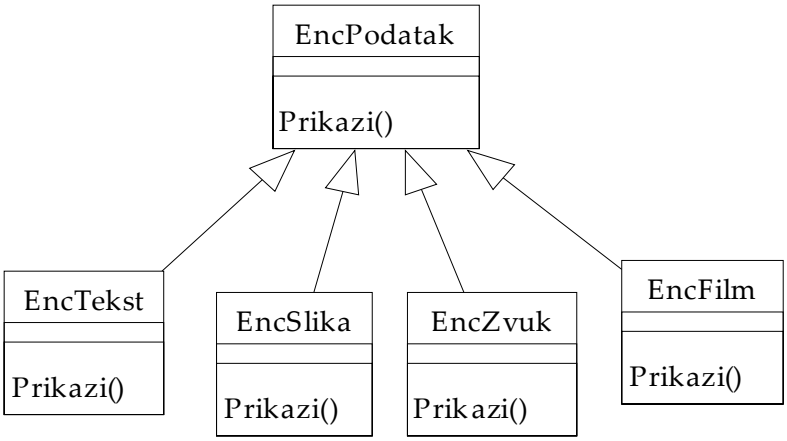
```
main( int argc, char** argv )
{
    int id = atoi( argv[1] );
    string bindat;
    if( argc > 2 )
        bindat = argv[2];
    EncPodatak* p = ProcitajIzBazePodataka( id );
    p->Prikazi( cout, bindat );
    delete p;
}
```

Nakon što smo preciznije definisali kakvo ponašanje želimo od našeg programa, potrebno je da prepoznamo koje su nam klase potrebne i kakvo ponašanje su one dužne da obezbede. Poći ćemo od jednostavne klase `EncPodatak` koja mora biti u stanju da formira izveštaj u traženom formatu:




### *Spuštanje ponašanja niz hijerarhiju*

Kako se prikazivanje podatka razlikuje za različite tipove podataka, možemo birati da li ćemo pri ispisivanju proveravati tip podatka i prilagođavati mu način ispisivanja, ili ćemo čitav postupak ispisivanja implementirati različito za različite tipove podataka. Jedna od osnovnih ideja vodilja pri objektno orijentisanom razvoju je da bi trebalo izbegavati eksplicitne analize tipova i izbore ponašanja – bolje je praviti hijerarhije klasa i upotrebljavati ih tako da se provere tipova i izbori ponašanja odvijaju implicitno, primenom dinamičkog vezivanja metoda. U skladu sa time formiraćemo više klasa podataka:




### *Podizanje ponašanja uz hijerarhiju*

Nije dobro sve elemente prikazivanja podatka implementirati iznova za svaku klasu. Jedna mana takvog pristupa je što ima više pisanja nego što je potrebno. Daleko značajniji problem, predstavlja činjenica da bi u tako pisanom kodu svaka izmena u ponovljenim delovima koda morala da bude implementirana u svim klasama. Da bismo obezbedili da se prikazivanje podataka razlikuje za različite tipove podataka, formirali smo nove klase i odlučili da na svakoj od njih implementiramo prikazivanje na odgovarajući način. To je bilo *spuštanje ponašanja niz hijerarhiju*. Sada nam je potreban upravo obrnut postupak, jer želimo da zajedničje delove ponašanja vratimo u baznu klasu. To postižemo deljenjem složenih postupaka na manje celine, zadržavajući zajedničke delove u baznoj klasi, a ostavljajući delove koji se razlikuju u izvedenim klasama. Da bismo mogli prepoznati zajedničke delove postupka prikazivanja podataka razmotrimo malo detaljnije primere opisane pri definisanju načina upotrebe programa. Potrebno je prikazati, redom:

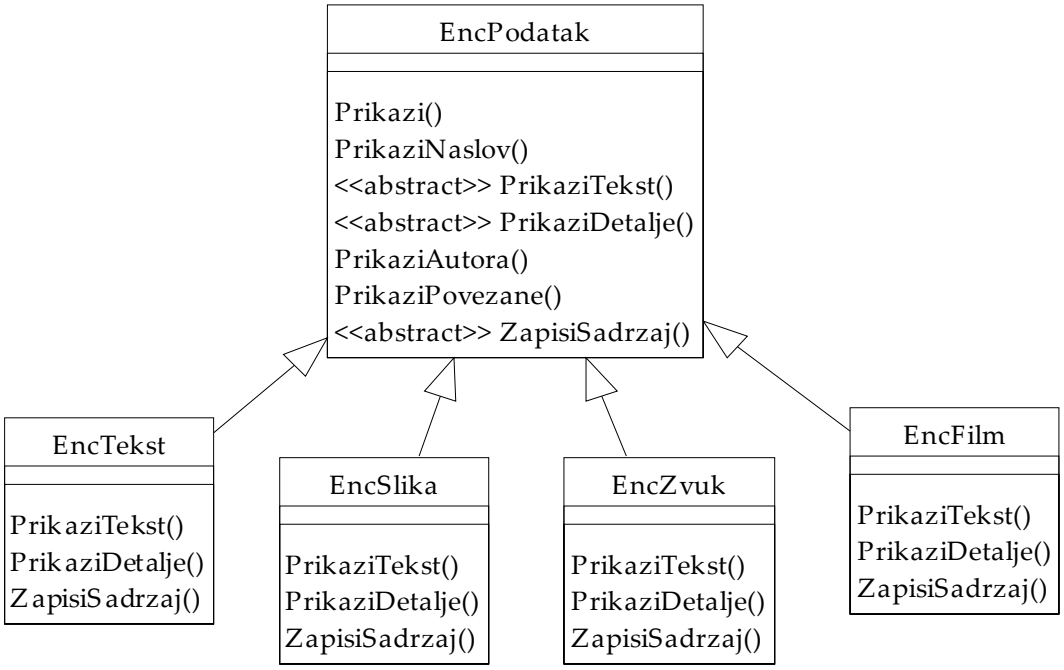
- naslov podatka i redni broj – ispisivanje se odvija na isti način za sve tipove podataka;
- tekst ili napomenu – u slučaju teksta ispisuje se tekst podatka, a u slučaju ostalih tipova podataka ispisuje se napomena;

- specifične podatke poput dimenzija i trajanja – ispisivanje podataka se razlikuje za sve tipove podataka;
- ime i prezime autora – ispisivanje se odvija na isti način za sve tipove podataka;
- naslove i redne brojeve povezanih podataka – isto za sve tipove podataka;
- binarni sadržaj je potrebno zapisati u datoteci – ovo je isto za sve tipove podataka osim za tekst.

Sada vidimo da bi implementacija metoda `EncPodatak::Prikazi` mogla da bude:

```
void EncPodatak::Prikazi( ostream& ostr, string bindat ) const
{
    PrikaziNaslov( ostr );
    PrikaziTekst( ostr );
    PrikaziDetalje( ostr );
    PrikaziAutora( ostr );
    PrikaziPovezane( ostr );
    ZapisiSadrzaj( bindat );
}
```

Možemo zaključiti da se metodi `Prikazi`, `PrikaziNaslov`, `PrikaziAutora` i `PrikaziPovezane` mogu implementirati u klasi `EncPodatak`, dok se metodi `PrikaziTekst`, `PrikaziDetalje` i `ZapisiSadrzaj` implementiraju različito za različite klase. Kada dijagram klasa prilagodimo novim zaključcima dobijamo:

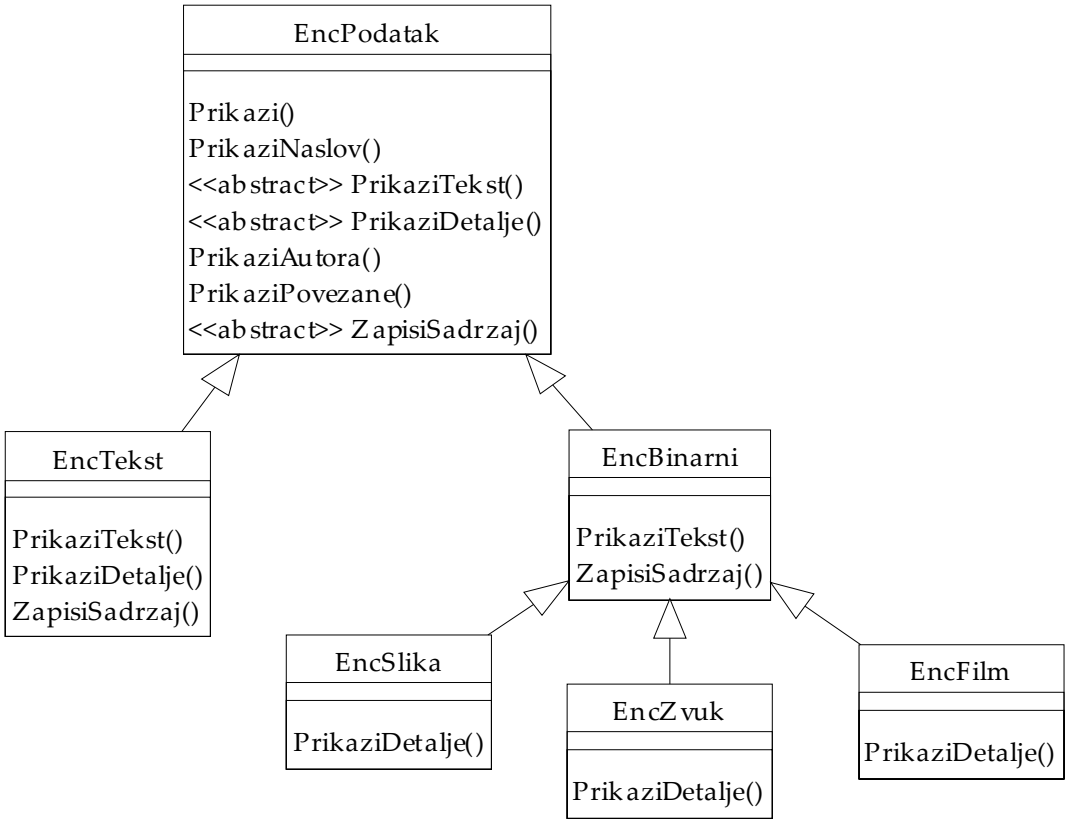



Obavljeni posao se naziva *podizanje zajedničkog ponašanja uz hijerarhiju*.

### *Formiranje klasa usred hijerarhije*

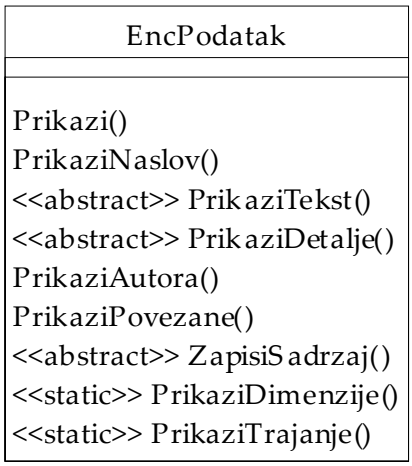
Ako posmatramo implementirane metode možemo lako uočiti da se prikazivanje teksta za sve tipove podataka, osim za tekst, obavlja na isti način. Takođe, i zapisivanje sadržaja u datoteku se

obavlja na sličan način – jedina razlika je u nazivu binarnog sadržaja u katalogu `Podatak`. Čini se da bi i za ove metode bilo dobro izvesti podizanje zajedničkog ponašanja uz hijerarhiju, ali ne smemo ih podići u klasu `EncPodatak` jer ona ima klasu naslednicu `EncTekst` koja se ponaša drugačije. U ovakvim slučajevima obično je korisno da uvedemo novu međuklasu koja istovremeno specijalizuje baznu klasu i apstrahuje jedan broj izvedenih klasa. Naša nova klasa bi trebalo da obuhvati najveći mogući presek ponašanja klasa `EncSlika`, `EncZvuk` i `EncFilm`. Njeno ponašanje se razlikuje od ponašanja klase `EncTekst`. Novu klasu ćemo nazvati `EncBinarni`:




### *Pomoćni metodi*

Da bi prikazivanje detaljnih informacija o podacima bilo ujednačeno, potrebno je sve one vrste informacija koje se mogu pojaviti za više vrsta objekata implementirati u klasi `EncPodatak`. Kako se nazivi podataka mogu razlikovati, moguće rešenje je definisanje statičkih metoda koji u željenom formatu ispisuju potrebne informacije:




***Razvajanje ponašanja***

Primetimo da se ime jednog od metoda razlikuje od ostalih metoda potrebnih za prikazivanje podataka: `ZapisiSadrzaj`. Da li je to slučajno? Ne. Dok se svi ostali metodi bave prikazivanjem podataka na standardnom izlazu, ovaj metod zapisuje binarni sadržaj podatka u datoteku. Zapravo, deluje neprirodno da se ta aktivnost odvija u okviru metoda `Prikazi`. Bilo bi bolje da se prikazivanje podatka na standardnom izlazu odvoji od zapisivanja njegovog binarnog sadržaja u datoteku. Štaviše, za korisnika klase može biti prilično opterećujuće, kako pri upoznavanju tako i pri upotrebi, ako metodi ne rede tačno ono što njihov naziv sugeriše. Nije dobro da metodi rade ni manje ni više od onoga što bi se na osnovu imena i arumenata moglo očekivati. Kako zapisivanje binarnog sadržaja u datoteci nema po svojoj prirodi, a ni po nazivu, ništa zajedničko sa prikazivanjem podataka na standardnom izlazu, dobro je ova dva postupka potpuno razdvojiti.

Dovoljno je da metod `Prikazi` obavi sve ostale predviđene aktivnosti:

```
void EncPodatak::Prikazi( ostream& ostr, string bindat ) const
{
    PrikaziNaslov( ostr );
    PrikaziTekst( ostr );
    PrikaziDetalje( ostr );
    PrikaziAutora( ostr );
    PrikaziPovezane( ostr );
}
```

Metodu `ZapisiSadrzaj` ćemo proširiti ranije pretpostavljeno ponašanje tako da vrati logičku vrednost `true` ako je sve proteklo u redu, a `false` u slučaju problema. Problemima ćemo smatrati sve obilike grešaka pri zapisivanju binarnog sadržaja u datoteku, kao i slučaj kada nije navedeno ime datoteke za zapisivanje binarnog sadržaja a on postoji.

Veoma je važno voditi računa o preciznom imenovanju metoda. Ime metoda mora ukazivati na operaciju koju metod izvodi. Ako metod radi bilo manje bilo više posla nego što njegovo ime ukazuje, možemo biti potpuno sigurni da će dolaziti do grešaka pri upotrebi metoda. Mnogi autori do te mere insistiraju na preciznom imenovanju metoda da smatraju da pisanje komentara u metodima uopšte nije potrebno:

- ukoliko na osnovu imena nije očigledno šta metod radi, to je zbog toga što metod obavlja više logički zasebnih celina – tada je potrebno metod podeliti na više metoda koji obavljaju po jednu celinu i sa kojima nema takvih problema;
- ukoliko je implementacija metoda složena do te mere da je neophodno komentarisanje delova koda, to je zato što se suviše složeno ponašanje pokušava implementirati jednim metodom – tada je potrebno svaku celinu iz implementacije metoda, za koju je potreban komentar, izdvojiti u poseban metod čije bi ime dovoljno dobro ilustrovalo obuhvaćeni deo algoritma, pa u polaznom metodu umesto više složenih segmenata koda staviti pozive novih metoda.

Primetimo da ekstremno pridržavanje ovakvog pristupa ima za posledicu veoma veliki broj jednostavnih metoda i klasa, što opet predstavlja problem (mada potpuno drugačiji) pri upoznavanju i upotrebi klase. Pri oblikovanju hijerarhije klasa i svake konkretne klase potrebno je pronaći dobru meru, kako bi skup klasa i metoda bio i pregledan i jasan.

### *Vidljivost metoda*

Oblikovanje naših klasa iz aspekta prikazivanja podataka je privedeno kraju. Primetimo da od svih navedenih metoda, samo metodi `Prikazi` i `ZapisiSadrzaj` moraju biti javni, dok svi ostali metodi mogu biti zaštićeni.

## *Korak 3 - Analiza i projektovanje konstruktora*

Da bismo mogli prikazivati enciklopedijske podatke, neophodno je da budemo u stanju da na osnovu podataka pročitanih iz baze podataka najpre napravimo odgovarajuće objekte. Kako podatke dobijamo iz baze podataka u formi kataloga `Podatak`, logično je da obezbedimo takve konstruktore koji kao argument imaju upravo `Podatak`:

```
EncPodatak::EncPodatak( const Podatak& p )
EncTekst::EncTekst( const Podatak& p )
EncBinarni::EncBinarni( const Podatak& p )
EncSlika::EncSlika( const Podatak& p )
EncZvuk::EncZvuk( const Podatak& p )
EncFilm::EncFilm( const Podatak& p )
```

Pri tome će se u svakoj od klasa najpre upotrebiti odgovarajući konstruktor bazne klase, a zatim iz podatka izdvojiti preostale informacije.

Neke informacije su tekstualne (`naslov`, `tekst`, `autor`, `napomene`), neke su celobrojne (`id`, `širina`, `visina`), neke realne (`trajanje`) a neke binarne (`slika`, `zvuk`, `film`). Za tekstualne i binarne podatke se mogu upotrebljavati upravo podaci tipa `string` koji su sadržani u objektu klase `Podatak`. U slučaju celobrojnih i realnih podataka potrebno je budemo u stanju da pročitamo brojeve iz njihove tekstualne reprezentacije. Zbog toga uvodimo nove statičke metode u klasu `EncPodatak`:

```
static int ProcitajCeoBroj( string s );
static double ProcitajRealanBroj( string s );
```

### *Dinamički konstruktor*

Osnovni problem pri pravljenju objekat predstavlja činjenica da pročitani `Podatak` može biti tekst, slika, zvuk ili film, pri čemu mi to ne znamo unapred. Kako onda napraviti objekat odgovarajuče klase? Prirodno rešenje bi bilo da pročitamo podatke iz baze podataka pa zatim analiziramo tip dobijenih podataka i pravimo odgovarajuće objekte:

```
...
Podatak podatak;
string tip;
EncPodatak* obj = 0;
if( ProcitajPodatak( id, podatak, tip )){
    if( tip == "tekst" )
        obj = new EncTekst( podatak );
    else if( tip == "slika" )
        obj = new EncSlika( podatak );
    else if( tip == "zvuk" )
        obj = new EncZvuk( podatak );
    else if( tip == "film" )
        obj = new EncFilm( podatak );
    }
...
```

Međutim, pre nekoliko trenutaka smo ustanovili da je u slučaju različitog ponašanja za različite tipove podataka potrebno izbegavati eksplicitnu analizu i izbore, već je bolje da se prave i upotrebljavaju hijerarhije klasa tako da se provere tipova i izbori ponašanja odvijaju implicitno, primenom dinamičkog vezivanja metoda. Da li možemo na ovom mestu upotrebiti dinamičko vezivanje metoda?

Da bismo mogli primeniti dinamičko vezivanje metoda neophodno je da imamo na raspolaganju objekat koji pripada nekoj od klasa iz hijerarhije. Ali mi ovde tek pravimo takav objekat i još uvek smo daleko od njegove upotrebe. Na žalost programera, nešto poput „dinamičkog konstruktora“ jednostavno ne postoji. Zapravo, uz makar malo poverenja u autore programskog jezika C++, možemo pomisliti da bi tako nešto sigurno postojalo ako bi uopšte moglo da postoji. I bili bismo u pravu. Problem je u tome što ma kako dobro da mi u konkretnoj situaciji znamo na osnovu kog parametra i na koji način želimo da napravimo i inicijalizujemo objekat tačno odgovarajuće klase, ne postoji neformalan način da se to saopšti prevodiocu već to moramo učiniti sasvim formalno – upravo pisanjem dela programa koji pravi novi objekat.

Deo programa koji smo napisali je sasvim u redu i u konkretnoj situaciji ga ne možemo značajno popraviti. Ipak, ima prostora za manje izmene. Ako posmatramo hijerarhiju enciklopedijskih podataka kao jednu celinu, a glavnu funkciju programa kao drugu, nije teško uočiti da autor funkcije `main` mora znati koji sve tipovi podataka postoje i kako se označavaju. To nije dobro jer predstavlja narušavanje enkapsulacije čitave hijerarhije. Daleko je bolje da se od korisnika hijerarhije sakrije informacija o tome koliko ima klasa i kako se prepoznaju. Sredstvo da to uradimo je prebacivanje dela programa koji se koristi za pravljenje novih objekata u samu hijerarhiju. To se može učiniti pisanjem odgovarajuće funkcije ili pisanjem statičkog metoda klase `EncPodatak`. Sasvim je jasno da se ta funkcija (ili metod) mora pisati, ili bar održavati, nakon svakog proširivanja hijerarhije. U ovom slučaju odlučujemo se za pisanje statičkog metoda:

```
EncPodatak* NapraviPodatak( string tip, const Podatak& p )
{
    EncPodatak* obj = 0;
```

```
        if( ProcitajPodatak( id, podatak, tip )){
            if( tip == "tekst" )
                obj = new EncTekst( podatak );
            else if( tip == "slika" )
                obj = new EncSlika( podatak );
            else if( tip == "zvuk" )
                obj = new EncZvuk( podatak );
            else if( tip == "film" )
                obj = new EncFilm( podatak );
            }
        return obj;
    }
```

Metod `NapraviPodatak` vraća prazan pokazivač ako se radi o podatku čiji tip nije podržan.

### *Korak 4 - Ostali metodi*

Prethodno opisanim statičkim metodom obezbedili smo pravljenje novog objekta enciklopedijskog podatka na osnovu već pročitanog podatka. Radi jednostavnije upotrebe možemo dodati još jedan statički metod za pravljenje novih objekata koji za argument ima samo redni broj podatka koji će sam pročitati iz baze podataka:

```
EncPodatak* ProcitajPodatakIzBazePodataka( int id )
```

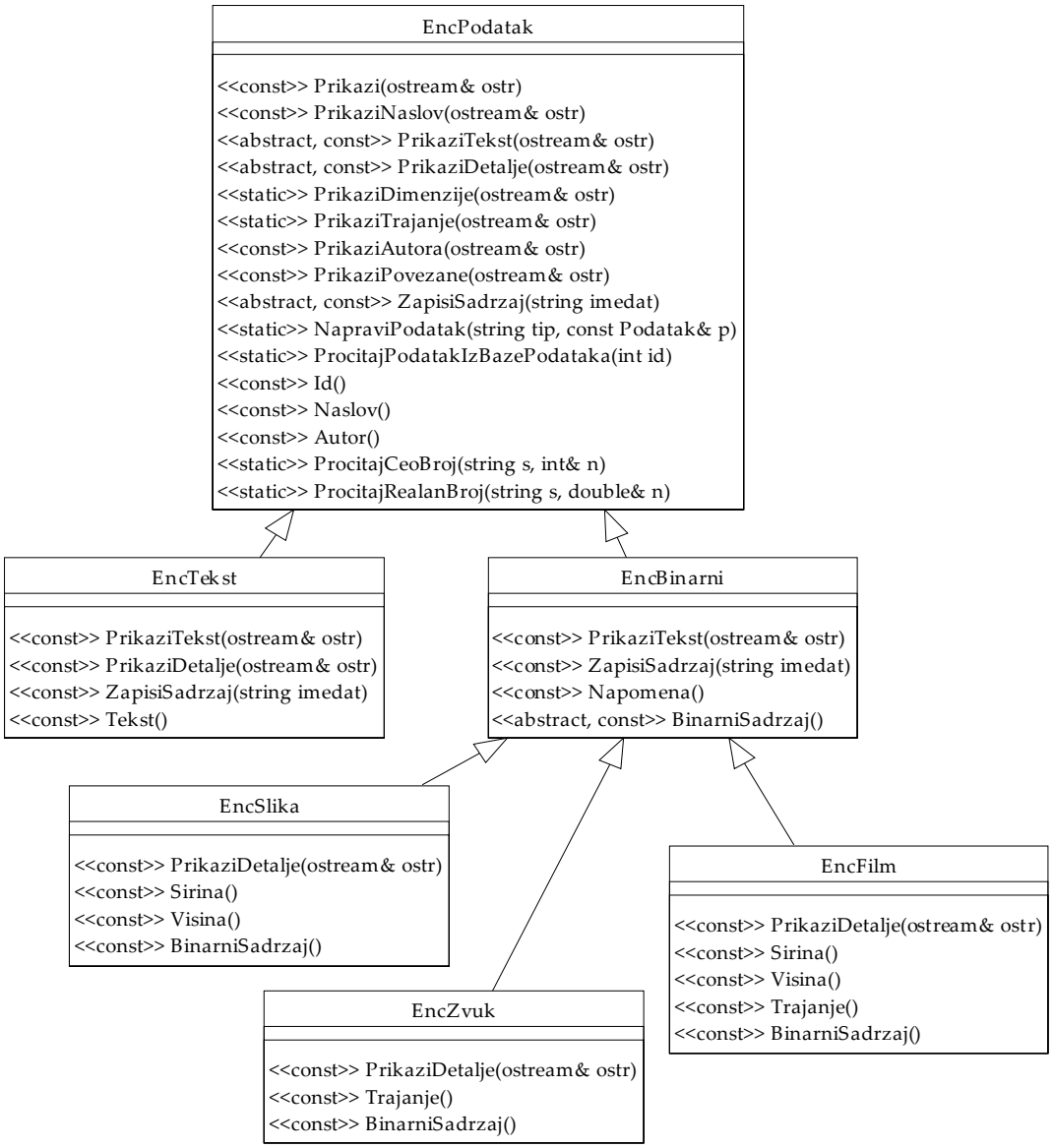
Da bi se jedan enciklopedijski podatak mogao predstaviti korisniku na zatevani način, potrebno je da budu na raspolaganju odgovarajuće informacije. Zbog toga u hijerarhiju klasa enciklopedijskih podataka dodajemo odgovarajuće pristupne metode, i to samo za čitanje:

- u klasu `EncPodatak` dodajemo metode za čitanje rednog broja, naslova i imena autora;
- u klasu `EncTekst` dodajemo metod za čitanje teksta;
- u klase `EncSlika` i `EndFilm` dodajemo metode za čitanje širine i visine;
- u klase `EncZvuk` i `EndFilm` dodajemo metod za čitanje trajanja;
- u klasu `EncBinarni` dodajemo metod za čitanje napomene.

Pored toga, u klasu `EncBinarni` dodajemo i apstraktni metod `BinarniSadrzaj` koji u konkretnik klasama izračunava sadržaj slike, zvuka ili filma.

Svi ovi podaci mogu da budu zaštićeni, jer u ovom trenutku ne postoji potreba da budu javni.

Sada se naša hijerarhija klasa može predstaviti sledećim dijagramom, koji obuhvata sve potrebne metode:




### *Korak 5 - Implementacija pomoćne „baze podataka“*

Da bismo mogli isprobavati kod koji pišemo nije dovoljno da imamo deklaraciju biblioteke za upotrebu baze podataka, onako kako je navedena u zaglavlju `BazaPodataka.h`, već je neophodno i da imamo primer implementacije baze podataka. Zato ćemo napisati pomoćnu biblioteku, koja bi trebalo da nam bude dovoljna tokom rešavanja zadatka. Neophodno je da obezbedimo implementaciju deklarisanih metoda, kao i da simuliramo čitanje primera podataka iz baze podataka.

Implementiranje klase `BazaPodataka` nije deo rešenja zadatka, već je pomoćno sredstvo koje nam omogućava da naš program isprobamo. Zbog toga ćemo implementaciju izvesti uz što manje napora, kako bismo pažnju što pre mogli posvetiti pisanju programa *Enciklopedija*.

Pored zahtevanih javnih metoda obezbedićemo i privatni metod

```
static bool ProcitajSve(
    int id, Podatak& x, string& tip, vector<int>& pov
    )
```

koji čita podatak, tip i niz rednih brojeva povezanih podataka. Javne metode ćemo implementirati pomoću metoda `ProcitajSve`. Metod `ProcitajSve` ćemo implementirati tako da pri prvoj upotrebi inicijalizuje statičke podatke koji će simulirati sadržaj baze podataka.

```
#include <map>
#include <vector>
using namespace std;

typedef map<string,string> Podatak;

class BazaPodataka
{
public:
    // Čitanje jednog podatka iz baze podataka
    static bool ProcitajPodatak( int id, Podatak& x, string& tip )
        {
        vector<int> povezani;
        return ProcitajSve( id, x, tip, povezani );
        }

    // Čitanje niza rednih brojeva povezanih podataka
    static void ProcitajPovezane( int id, vector<int>& povezani )
        {
        Podatak x;
        string tip;
        ProcitajSve( id, x, tip, povezani );
        }

private:
    // Čitanje svih informacijao podatku iz baze podataka
    static bool ProcitajSve(
            int id, Podatak& x, string& tip, vector<int>& pov
            )
        {
        // Statički podaci kojima simuliramo sadržaj baze podataka
        static map<int,Podatak> podaci;
        static map<int,vector<int> > povezani;
        static map<int,string> tipovi;
        // Podatke inicijalizujemo samo prvi put
        if( podaci.empty() ){
          { Podatak p;
            p["id"] = "42";
            p["naslov"] = "Lav";
            p["tekst"] =
                "Lav (Leo ili Panthera leo) je velika snažna "
                "mačka iz porodice Felidae, druga po veličini od "
                "velikih mačaka (posle tigra). ...";
            p["autor"] = "Pera Perić";
            podaci[42] = p;
            povezani[42].push_back( 46 );
            povezani[42].push_back( 47 );
            povezani[42].push_back( 48 );
            povezani[42].push_back( 73 );
            tipovi[42] = "tekst";
          }{
            Podatak p;
            p["id"] = "47";
            p["naslov"] = "Lav";
            p["slika"] = "...binarni zapis slike...";
```

```
            p["sirina"] = "1024";
            p["visina"] = "768";
            p["napomena"] =
                "Odrasli primerak mužjaka u prirodnom okruženju.";
            p["autor"] = "Fotko Fotkić";
            podaci[47] = p;
            povezani[47].push_back( 42 );
            povezani[47].push_back( 46 );
            povezani[47].push_back( 48 );
            tipovi[47] = "slika";
          }{
            Podatak p;
            p["id"] = "46";
            p["naslov"] = "Urlik lava";
            p["zvuk"] = "...binarni zapis zvuka...";
            p["trajanje"] = "3764,45";
            p["napomena"] =
                "Upozoravajući urlik lava spremnog da brani "
                "svoju teritoriju.";
            p["autor"] = "Sima Snimić";
            podaci[46] = p;
            povezani[46].push_back( 42 );
            povezani[46].push_back( 47 );
            povezani[46].push_back( 48 );
            tipovi[46] = "zvuk";
          }{
            Podatak p;
            p["id"] = "48";
            p["naslov"] = "Lav u lovu";
            p["film"] = "...binarni zapis filma...";
            p["sirina"] = "640";
            p["visina"] = "400";
            p["trajanje"] = "124,75";
            p["napomena"] =
                "Lavica lovi antilopu. Obratite pažnju na pomoć "
                "koju joj pružaju ostali lavovi onemogućavajući "
                "antilopi da pobegne.";
            p["autor"] = "Žika Žikić";
            podaci[48] = p;
            povezani[48].push_back( 42 );
            povezani[48].push_back( 46 );
            povezani[48].push_back( 47 );
            tipovi[48] = "film";
          }{
            Podatak p;
            p["id"] = "73";
            p["naslov"] = "Tigar";
            p["tekst"] = "...neki tekst o tigru...";
            p["autor"] = "Žika Žikić";
            podaci[73] = p;
            povezani[73].push_back( 42 );
            tipovi[73] = "tekst";
          }
        }
        // Izračunavamo tražene podatke
        x = podaci[id];
        tip = tipovi[id];
        pov = povezani[id];
        return !tip.empty();
        }
};
```

### *Korak 6 - Struktura klasa*

Kao rezultat analize problema i projektovanja dobili smo dijagram klasa i skicu implementacije nekih važnih metoda. Većina poslova koje nam je preostalo da uradimo tokom implementacije su prilično jednostavni. Jedina važnija odluka koju još nismo doneli odnosi se na internu strukturu klasa naše hijerarhije. I na ovom primeru vidimo da se ponašanje čitave hijerarhije klasa može u potpunosti opisati bez ikakvog zalaženja u određivanje strukture. Sada, kada znamo šta će i kako naše klase raditi, ipak moramo da vidimo i kako će izgledati.

Obično se faze razvoja koje prethode određivanju interne strukture klasa nazivaju *projektovanjem*, a preostale faze, počev od određivanja interne strukture, se nazivaju *implementacijom*.

#### *Članovi podaci*

Imamo na raspolaganju dva osnovna pristupa definisanju strukture klasa hijerarhije enciklopedijskih podataka. Jedan je da se za svaku konkretnu informaciju o enciklopedijskom podatku definiše po član podatak u klasi u kojoj postoji odgovarajući pristupni metod. Tako bi klasa `EncPodatak` imala naslov i ime autora, klasa `EncTekst` bi imala tekst, klasa `EncBinarni` bi imala napomenu, a ostale klase bi imale njima specifične informacije. U tom slučaju konstruktori bi bili odgovorni da inicijalizuju te članove podatke, a pristupni metodi bi jednostavno čitali vrednosti članova podataka.

Drugi pristup problemu interne strukture je da se u okviru klase `EncPodatak` definiše član podatak klase `Podatak`. U tom slučaju klase naslednice ne bi imale nikakve nove članove podatke. Konstruktori izvedenih klasa bi samo prenosili odgovornost na konstrukor bazne klase, a on bi iskopirao čitav katalog tipa `Podatak`. Pristupni metodi bi na osnovu sadržaja kataloga klase `Podatak` izračunavali odgovarajuće vrednosti.

Nijedan od dva pristupa nema neku značajnu prednost u odnosu na drugi. Na primer, u prvom slučaju se svi podaci tačno jedanput čitaju iz kataloga i po potrebi konvertuju u brojeve, dok se u drugom slučaju čitaju samo oni podaci koji se upotrebljavaju u programu, ali se to može ponoviti više puta. Prvi pristup obezbeđuje preglednu i jasnu strukturu, ali se ona mora dopunjavati sa svakom novom klasom, dok drugi pristup prilično sakriva strukturu (što je upravo nešto dublja enkapsulacija) i oslobađa nas obaveze da dodajemo članove u nove klase. U ovom trenutku izbor pada na prvi pristup formiranju strukture klasa.

#### *Destruktori*

Kada koristimo hijerarhije klasa kao u primeru funkcije `main` postavlja se pitanje kako se ponaša operator `delete`? Dovoljno je da posmatramo primer poput:

```
delete EncPodatak::ProcitajPodatakIzBazePodataka( id );
```

Metod `EncPodatak::ProcitajPodatakIzBazePodataka` izračunava pokazivač na neki enciklopedijski podatak. Rezultat je tipa `EncPodatak*`. Operator `delete` se može bezbedno primeniti i ako je rezulttat 0. Kao što već znamo, operator `delete` najpre poziva destruktor, pa zatim oslobađa memoriju koju je objekat zauzimao. To će se desiti i u navedenom primeru. Prevodilac će prepoznati da operator `delete` ima operand tipa `EncPodatak*`. Na osnovu toga će doći do pozivanja destruktora klase `EncPodatak` i do oslobađanja memorije koju je objekat zauzimao. To bi trebalo da je sasvim ispravno?

Ne! Doći do veoma ozbiljnih problema, jer pozivanjem destruktora bazne klase najčešće ne može da se ispravno deinicijalizuje objekat izvedene klase.

Neka je, na primer, pročitan podatak klase `EncSlika`, koji ima član podatak `_Slika` tipa `string` koji zadrži binarni zapis slike. Kako klasa `string` ima ispravno definisan destruktor, podrazumevani destruktor klase `EncSlika` će sasvim ispravno deinicijalizovati podatak `_Slika` i osloboditi odgovarajuću memoriju. Na osnovu toga možemo zaključiti da nema potrebe eksplicitno pisati destruktor klase `EncSlika`. Takav zaključak je potpuno ispravan. Do istog zaključka možemo doći u odnosu na bilo koju klasu hijerarhije enciklopedijskih podataka. Ni u jednoj od klasa nije potrebno pisati destruktor jer podrazumevani destruktori sasvim ispravno rade svoj posao.

Jedini, ali nimalo bezazlen, problem je u činjenici da je taj ispravan podrazumevani destruktor klase `EncSlika` potrebno i pozvati. A u prethodnom primeru se to neće desiti jer se odluka o pozivanju destruktora donosi u fazi prevođenja programa. Umesto destruktora klase `EncSlika` upotrebiće se destruktor klase `EncPodatak` i neće se deinicijalizovati neki podaci.

Jedino rešenje ovog problema koje je u skladu sa principima programiranja na programskom jeziku C++ jeste da se destruktor vezuje dinamički a ne statički. To se postiže stavljanjem ključne reči `virtual` ispred deklaracije destruktora bazne klase hijerarhije:

```
virtual ~EncPodatak();
```

Ostaje još da vidimo kako ćemo implementirati destruktor. Kao što smo već videli, u slučaju klase `EncPodatak` potrebno je da eksplicitno definišemo destruktor isključivo zbog toga da bismo označili da je neophodno da se njegovo vezivanje odvija dinamički. U skladu sa time, implementacija je trivijalna:

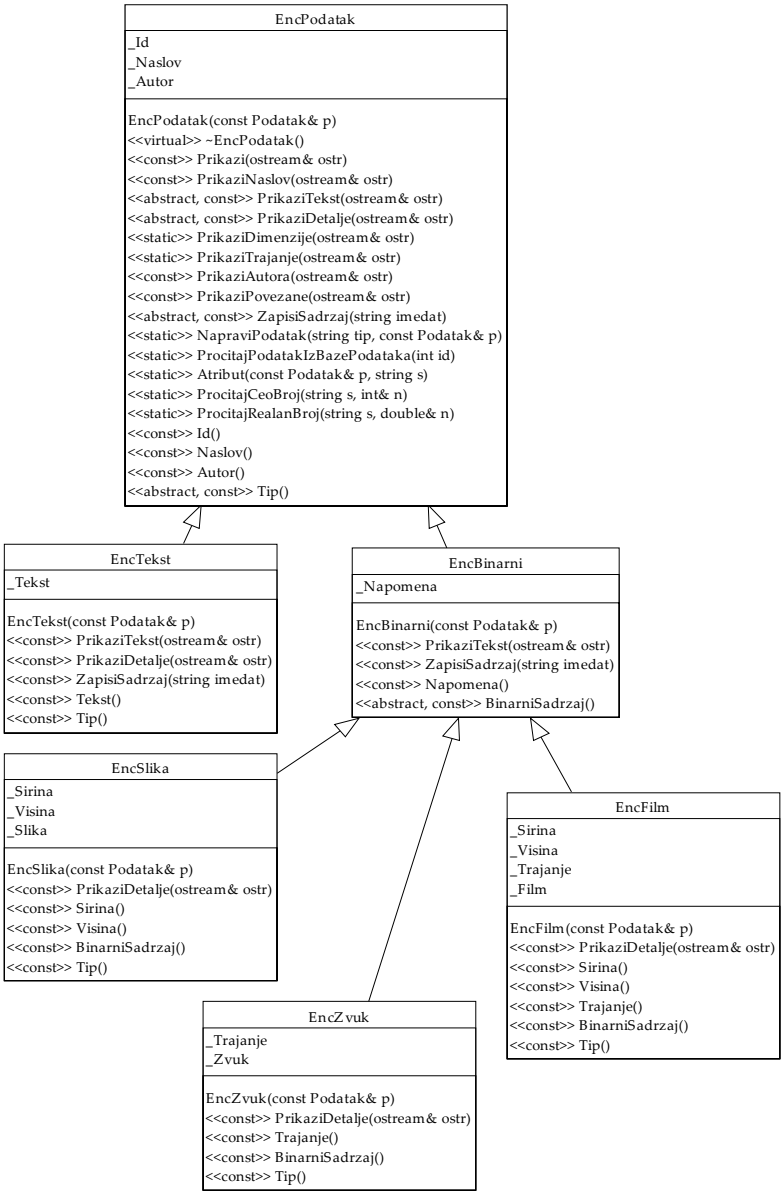
```
virtual ~EncPodatak()
    {}
```

### *Pomoćni metodi*

Pri ispisivanju enciklopedijskih podataka potrebno je ispisivati i tip podataka. Zbog toga u baznu klasu uvodimo apstraktan metod `Tip` koji izračunava nisku sa nazivom tipa i koji implementiramo u svim konkretnim klasama.

Da bismo mogli jednostavno da čitamo atribute iz objekta tipa `Podatak`, definisaćemo i pomoćni statički metod `Atribut` u klasi `EncPodatak`.

Kada prethodnom dijagramu klasa dodamo članove podatke, konstruktore, destruktor bazne klase i upravo predstavljene metode, dobijamo potpuniju sliku hijerarhije:




## *Korak 7 - Implementacija hijerarhije klasa*

Poći ćemo od bazne klase `EncPodatak`. Zbog toga što ona ima veći broj metoda, najpre ćemo deklarisati klasu a zatim implementirati metode. Za razliku od nje, ostale klase hijerarhije su sasvim jednostavne i sve metode ćemo implementirati već pri definisanju klasa.

### *Klasa EncPodatak*

Klasa `EncPodatak` predstavlja osnovu hijerarhije. Zbog toga u njoj imamo nekoliko apstraktnih metoda. Pored toga, imamo i nekoliko statičkih metoda koji omogućavaju funkcionisanje čitave hijerarhije. Sada ćemo napisati deklaraciju klase `EncPodatak` i sve metode koji nisu apstraktni, osim metoda `NapraviPodatak`. Da bismo mogli napisati metod `NapraviPodatak` neophodno je da imamo na raspolaganju bar deklaracije svih klasa podataka čije objekte ovaj metod mora da pravi. Zato implementaciju tog metoda ostavljamo za kraj.

Deklaraciju klase pišemo u datoteci `EncPodatak.h`:

```
class EncPodatak
{
public:
    //  Konstruktor
    EncPodatak( const Podatak& p );

    //  Destruktor
    virtual ~EncPodatak()
        {}

    //  Prikazivanje i pisanje
    void Prikazi( ostream& ostr ) const;
    virtual bool ZapisiSadrzaj( string imedat ) const = 0;

    //  "Konstruktori"
    static EncPodatak* NapraviPodatak(
                           string tip, const Podatak &p
                           );
    static EncPodatak* ProcitajPodatakIzBazePodataka( int id );

protected:
    //  Pristupni metodi
    int Id() const
        { return _Id; }
    string Naslov() const
        { return _Naslov; }
    string Autor() const
        { return _Autor; }
    virtual string Tip() const = 0;

    //  Prikazivanje dela podatka
    void PrikaziNaslov( ostream& ostr ) const;
    void PrikaziAutora( ostream& ostr ) const;
    void PrikaziPovezane( ostream& ostr ) const;
    virtual void PrikaziTekst( ostream& ostr ) const = 0;
    virtual void PrikaziDetalje( ostream& ostr ) const = 0;

    //  Pomoćni metodi za prikazivanje
    static void PrikaziDimenzije( ostream& ostr, int s, int v );
    static void PrikaziTrajanje( ostream& ostr, double sec );

    //  Pomoćni metodi za čitanje podataka iz strukture Podatak
    static string Atribut( const Podatak& p, string s );
    static int ProcitajCeoBroj( string s )
        { return atoi( s.c_str() ); }
    static double ProcitajRealanBroj( string s )
        { return atof( s.c_str() ); }

private:
    //  Članovi podaci
    int     _Id;
    string  _Naslov;
    string  _Autor;
};
```

Implementacija destruktora i pristupnih metoda je trivijalna. Implementaciji ostalih metoda ćemo posvetiti malo više pažnje.

Pri implementaciji konstruktora upotrebljavamo pomoćni metod `Atribut` za pristupanje atributima pročitanog podatka i metod `ProcitajCeoBroj` za konverziju niske u ceo broj. Sve članove podatke inicijalizujemo navodeći ih u listi inicijalizacija:

```
EncPodatak::EncPodatak( const Podatak& p )
    : _Id( ProcitajCeoBroj(Atribut(p,"id")) ),
      _Naslov( Atribut(p,"naslov") ),
      _Autor( Atribut(p,"autor") )
{}
```

Prikazivanje definišemo tako da bude u skladu sa navedenim prototipovima izveštaja:

```
void EncPodatak::Prikazi( ostream& ostr ) const
{
    ostr << endl;
    PrikaziNaslov( ostr );
    PrikaziTekst( ostr );
    PrikaziDetalje( ostr );
    ostr << endl;
    PrikaziAutora( ostr );
    ostr << endl;
    PrikaziPovezane( ostr );
    ostr << endl;
}

void EncPodatak::PrikaziNaslov( ostream& ostr ) const
{
    ostr << Naslov()
         << " (" << Tip() << " " << Id() << ")" << endl;
    ostr << "----------------------------------------" << endl;
}

void EncPodatak::PrikaziAutora( ostream& ostr ) const
{
    ostr << Autor() << endl;
}

void EncPodatak::PrikaziDimenzije( ostream& ostr, int s, int v )
{
    ostr << "Dimenzije: " << s << " x " << v << endl;
}

void EncPodatak::PrikaziTrajanje( ostream& ostr, double sec )
{
    ostr << "Trajanje: ";
    if( sec >= 60 ){
        int m = sec / 60;
        ostr << m << "m ";
        sec -= m * 60;
        }
    ostr << sec << "s" << endl;
}
```

Prikazivanje povezanih podataka je nešto složenije. Najpre čitamo redne brojeve povezanih podataka. Zatim redom čitamo sve povezane podatke i formiramo njihove opise od naslova i naziva tipa podataka. Tako formirane opise i redne brojeve pamtimo u katalogu. Na taj način implicitno uređujemo podatke u abecednom poretku:

```
void EncPodatak::PrikaziPovezane( ostream& ostr ) const
{
    vector<int> povezani;
    BazaPodataka::ProcitajPovezane( Id(), povezani );
    if( povezani.empty() )
        ostr << "Nema povezanih podataka." << endl;
    else{
        ostr << "Povezani podaci:" << endl;
        map<string,int> naslovi;
        for( unsigned i=0; i<povezani.size(); i++ ){
            EncPodatak* p
                = ProcitajPodatakIzBazePodataka( povezani[i] );
            if( p ){
                naslovi[ p->Naslov() + " (" + p->Tip() ]
                    = p->Id();
                delete p;
                }
            }
        map<string,int>::iterator
            i = naslovi.begin(),
            e = naslovi.end();
        for( ; i!=e; i++ )
            ostr << " - " << i->first << " "
                 << i->second << ")" << endl;
        }
}
```

Pravljenje novih objekata i njihovo čitanje iz baze podataka rešavamo na ranije opisan način:

```
EncPodatak* EncPodatak::NapraviPodatak(
                            string tip, const Podatak& p
                            )
{
    EncPodatak* obj = 0;
    if( tip == "tekst" )
        obj = new EncTekst( p );
    else if( tip == "slika" )
        obj = new EncSlika( p );
    else if( tip == "zvuk" )
        obj = new EncZvuk( p );
    else if( tip == "film" )
        obj = new EncFilm( p );
    return obj;
}

EncPodatak* EncPodatak::ProcitajPodatakIzBazePodataka( int id )
{
    Podatak p;
    string tip;
    return BazaPodataka::ProcitajPodatak( id, p, tip )
                ? NapraviPodatak( tip, p )
                : 0;
}
```

Pomoćni metod `Atribut` upotrebljavamo za izdvajanje atributa podataka:

```
string EncPodatak::Atribut( const Podatak& p, string s )
{
```

```
    string v;
    Podatak::const_iterator i = p.find(s);
    if( i != p.end() )
        v = i->second;
    return v;
}
```

### *Klasa EncTekst*

Klasa `EncTekst` predstavlja konkretan tekstualni enciklopedijski podatak. Potrebno je definisati konstruktor i nove pristupne metode i implementirati nasleđene apstraktne metode:

```
class EncTekst : public EncPodatak
{
public:
    //  Konstruktor
    EncTekst( const Podatak& p )
        : EncPodatak( p ),
          _Tekst( Atribut(p,"tekst") )
        {}

    //  Prikazivanje i pisanje
    bool ZapisiSadrzaj( string ) const
        { return true; }

protected:
    //  Pristupni metodi
    string Tekst() const
        { return _Tekst; }
    string Tip() const
        { return "tekst"; }

    //  Prikazivanje dela podatka
    void PrikaziTekst( ostream& ostr ) const
        { ostr << Tekst() << endl; }
    void PrikaziDetalje( ostream& ) const
        {}

private:
    //  Članovi podaci
    string  _Tekst;
};
```

### *Klasa EncBinarni*

Klasa `EncBinarni` apstrahuje sve tipove ekciklopedijskih podataka koji imaju neki binarni sadržaj. Kao u u slučaju klase `EncTekst`, potrebno je definisati konstruktor i nove pristupne metode. Implementiraju se metodi za prikazivanje teksta i zapisivanje binarnog sadržaja, ali i deklariše novi apstraktan metod za izdvajanje binarnog sadržaja:

```
class EncBinarni : public EncPodatak
{
public:
    //  Konstruktor
    EncBinarni( const Podatak& p )
        : EncPodatak( p ),
          _Napomena( Atribut(p,"napomena") )
        {}
```

```
    //  Prikazivanje i pisanje
    bool ZapisiSadrzaj( string imedat ) const
        {
        if( imedat.empty() )
            return false;

        ofstream f( imedat.c_str(), ios::binary );
        if( !f )
            return false;
        const string& s = BinarniSadrzaj();
        f.write( s.c_str(), s.length() );
        if( !f )
            return false;

        return true;
        }

protected:
    //  Pristupni metodi
    string Napomena() const
        { return _Napomena; }
    virtual const string& BinarniSadrzaj() const = 0;

    //  Prikazivanje dela podatka
    void PrikaziTekst( ostream& ostr ) const
        { ostr << Napomena() << endl; }

private:
    //  Članovi podaci
    string  _Napomena;
};
```

### *Ostale klase*

Klase `EncSlika`, `EncZvuk` i `EncFilm` su međusobno slične, pa ćemo ovde predstaviti samo definiciju klase `EncSlika`. Definišemo konstruktor i nove pristupne metode i implementiramo sve preostale apstraktne metode:

```
class EncSlika : public EncBinarni
{
public:
    //  Konstruktor
    EncSlika( const Podatak& p )
        : EncBinarni( p ),
          _Sirina( ProcitajCeoBroj(Atribut(p,"sirina")) ),
          _Visina( ProcitajCeoBroj(Atribut(p,"visina")) ),
          _Slika( Atribut(p,"slika") )
        {}

protected:
    //  Pristupni metodi
    int Sirina() const
        { return _Sirina; }
    int Visina() const
        { return _Visina; }
    const string& BinarniSadrzaj() const
        { return _Slika; }
    string Tip() const
```

```
        { return "slika"; }

    //  Prikazivanje dela podatka
    void PrikaziDetalje( ostream& ostr ) const
        {
        ostr << endl;
        PrikaziDimenzije( ostr, Sirina(), Visina() );
        }

private:
    //  Članovi podaci
    int     _Sirina;
    int     _Visina;
    string  _Slika;
};
```

### *Korak 8 - Implementacija glavne funkcije programa*

U glavnoj funkciji programa posebnu pažnju posvećujemo mogućim greškama pri pokretanju ili izvršavanju programa.

```
int main(int argc, char* argv[])
{
    // Ako nema dovoljno argumenata, ispišemo uputstvo
    if( argc < 2 ){
        cerr << "Upotreba: " << endl
             << "    " << argv[0] << " <id> [<bindat>]" << endl;
        return 2;
        }

    // Pročitamo redni broj i odgovarajući podatak
    int id = atoi( argv[1] );
    EncPodatak* p
        = EncPodatak::ProcitajPodatakIzBazePodataka( id );

    // Ako nema traženog podatka obavestimo korisnika i završimo
    if(!p){
        cerr << "Ne postoji podatak sa rednim brojem "
             << id << "!" << endl;
        return 1;
        }

    // Prikažemo podatak,
    p->Prikazi( cout );

    // Ako je na raspolaganju datoteka,
    // zapišemo binarni sadržaj podatka u datoteku
    if( argc > 2 ){
        if( !p->ZapisiSadrzaj( argv[2] ))
            cerr << "Nije uspelo pisanje u datoteku \""
                 << argv[2] << "\"!" << endl;
        }
    // inace proverimo da li je bila potrebna
    else if( !p->ZapisiSadrzaj( "" ))
        cout << "Binarni sadrzaj nije zapisan "
             << "jer nije naveden naziv datoteke!" << endl;

    // Uklonimo viškove
    delete p;
```

```
    // Sve je u redu
    return 0;
}
```

### *Korak 9 - Organizacija teksta programa*

Do sada se nismo mnogo bavili organizovanjem teksta programa po datotekama. U nekim jednostavnijim situacijama to pitanje nije posebno važno. Međutim, kada program počne da raste i broj klasa postane veći, možemo se susresti sa nekim novim problemima koji se jednostavno rešavaju dobrim organizovanjem programskog teksta.

Često se u deklaracijama argumenata ili rezultata metoda pojavljuje ime neke druge klase. Ako se argument ili rezultat prenosi po vrednosti, neophodno je da takvoj deklaraciji prethodi definicija odgovarajuće klase. Ako se prenosi po imenu (tj. primenom pokazivača ili reference) tada je dovoljno da toj deklaraciji prethodi deklaracija imena klase. Isto važi i za članove podatke – ako član podatak predstavlja objekat druge klase, neophodno je da deklaraciji prethodi definicija te druge klase, a ako predstavlja referencu ili pokazivač na objekat druge klase, dovoljno je da prethodi deklaracija imena druge klase. Ukoliko implementacija metoda koristi više informacija o argumentu ili rezultatu koji je pokazivač ili referenca na neku drugu klasu, tj. članove podatke ili metode te klase, tada implementaciji metoda mora da prethodi definicija te druge klase.

Deklaracija imena klase ima oblik:

```
class ImeKlase;
```

U našem slučaju, pri implementaciji hijerarhije klasa enciklopedijskih podataka dolazimo u situaciju da implementaciji metoda `NapraviPodatak` klase `EncPodatak` moraju prethoditi definicije svih ostalih klasa hijerarhije. To je sasvim jednostavan problem, ali se njegovo rešavanje ne razlikuje značajno od rešavanja daleko složenijih problema tog tipa.

Osnovna ideja je u razdvajanju definicija klasa i implementacija metoda u zasebne datoteke. Definicije klase se obično zapisuju u *zaglavljima*. Zaglavlja su datoteke koje obično imaju nastavak imena `.h` ili `.hpp`. Implementaciju metoda zapisujemo u programskim datotekama, koje obično imaju nastavak imena `.cpp` ili `.C`.

Definiciju klase `EncPodatak` ćemo zapisati u datoteci `EncPodatak.h`:

```
#ifndef EncPodatakH
#define EncPodatakH

#include <iostream>
using std::ostream;

#include "BazaPodataka.h"

//-------------------------------------------------------------
// Klasa EncPodatak
//-------------------------------------------------------------
// Predstavlja osnovnu klasu hijerarhije enciklopedijskih
// podataka. Obuhvata i pomoćne metode za čitanje podataka
// iz baze podataka i njihovo prikazivanje.
//-------------------------------------------------------------
class EncPodatak
{
```

```
...
};

//-------------------------------------------------------------
#endif  //  #ifndef EncPodatakH
```

Za deklarisanje metoda `NapraviPodatak` nije potrebno da se zna koje su to ostale klase hijerarhije. Jedino zaglavlje koje moramo uključiti jeste `iostream`. Umesto uobičajene deklaracije upotrebe prostora imena:

```
using namespace std;
```

deklarisali smo samo konkretna imena iz prostora imena `std` koja koristimo. Ovde je to učinjeno radi ilustracije, ali u nekim situacijama takvo pojedinačno deklarisanje omogućava fleksibilniju upotrebu različitih biblioteka.

Implementaciju metoda klase `EncPodatak` zapisaćemo u datoteci `EncPodatak.cpp`. Da bi implementaciji metoda `NapraviPodatak` prethodile definicije svih ostalih klasa hijerarhije, uključićemo zaglavlje `EncKlase.h` u kome će one biti definisane.

```
#include "EncPodatak.h"
#include "EncKlase.h"

//-------------------------------------------------------------
// Klasa EncPodatak - Implementacija metoda
//-------------------------------------------------------------
// Predstavlja osnovnu klasu hijerarhije enciklopedijskih
// podataka. Obuhvata i pomoćne metode za čitanje podataka
// iz baze podataka i njihovo prikazivanje.
//-------------------------------------------------------------
EncPodatak::EncPodatak( const Podatak& p )
...
```

Ostale klase su sasvim jednostavne. Zbog relativno malog broja klasa i njihove jednostavnosti možemo se odlučiti da sve klase zapišemo u jednoj datoteci, kao i da implementacije metoda izvedemo u okviru definicija klasa. Kao što smo već nagovestili, to je datoteka `EncKlase.h`:

```
#ifndef EncKlaseH
#define EncKlaseH

#include "EncPodatak.h"

//-------------------------------------------------------------
// Klasa EncTekst
//-------------------------------------------------------------
// Tekstualni enciklopedijski podatak.
//-------------------------------------------------------------
class EncTekst : public EncPodatak
{
...
};

//-------------------------------------------------------------

...
```

```
//-------------------------------------------------------------
#endif  //  #ifndef EncKlaseH
```

Glavni program zapisujemo u posebnoj programskoj datoteci `Enciklopedija.cpp`.

```
#include <iostream>
using std::cout;
using std::cerr;
using std::endl;

#include "EncPodatak.h"

//-------------------------------------------------------------
// Program Enciklopedija.
//-------------------------------------------------------------
// Omogućava čitanje i prikazivanje podataka iz enciklopedije
// i zapisivanje binarnih sadržaja u datoteke.
//-------------------------------------------------------------
int main(int argc, char* argv[])
{
...
}
```

# 8.3 Rešenje

### *Datoteka EncPodatak.h*

```
#ifndef EncPodatakH
#define EncPodatakH

#include <iostream>
using std::ostream;

#include "BazaPodataka.h"

//-------------------------------------------------------------
// Klasa EncPodatak
//-------------------------------------------------------------
// Predstavlja osnovnu klasu hijerarhije enciklopedijskih
// podtaka. Obuhvata i pomoćne metode za čitanje podataka
// iz baze podataka i njihovo prikazivanje.
//-------------------------------------------------------------
class EncPodatak
{
public:
    //  Konstruktor
    EncPodatak( const Podatak& p );

    //  Destruktor
    virtual ~EncPodatak()
        {}

    //  Prikazivanje i pisanje
    void Prikazi( ostream& ostr ) const;
    virtual bool ZapisiSadrzaj( string imedat ) const = 0;

    //  "Konstruktori"
```

```
    static EncPodatak* NapraviPodatak(
                            string tip, const Podatak &p
                            );
    static EncPodatak* ProcitajPodatakIzBazePodataka( int id );

protected:
    //  Pristupni metodi
    int Id() const
        { return _Id; }
    string Naslov() const
        { return _Naslov; }
    string Autor() const
        { return _Autor; }
    virtual string Tip() const = 0;

    //  Prikazivanje dela podatka
    void PrikaziNaslov( ostream& ostr ) const;
    void PrikaziAutora( ostream& ostr ) const;
    void PrikaziPovezane( ostream& ostr ) const;
    virtual void PrikaziTekst( ostream& ostr ) const = 0;
    virtual void PrikaziDetalje( ostream& ostr ) const = 0;

    //  Pomoćni metodi za prikazivanje
    static void PrikaziDimenzije( ostream& ostr, int s, int v );
    static void PrikaziTrajanje( ostream& ostr, double sec );

    //  Pomoćni metodi za čitanje podataka iz strukture Podatak
    static string Atribut( const Podatak& p, string s );
    static int ProcitajCeoBroj( string s )
        { return atoi( s.c_str() ); }
    static double ProcitajRealanBroj( string s )
        { return atof( s.c_str() ); }

private:
    //  Članovi podaci
    int     _Id;
    string  _Naslov;
    string  _Autor;
};

//---------------------------------------------------------------
#endif  //  #ifndef EncPodatakH
```

### *Datoteka EncPodatak.cpp*

```
#include "EncPodatak.h"
#include "EncKlase.h"

//---------------------------------------------------------------
// Klasa EncPodatak - Implementacija metoda
//---------------------------------------------------------------
// Predstavlja osnovnu klasu hijerarhije enciklopedijskih
// podtaka. Obuhvata i pomoćne metode za čitanje podataka
// iz baze podataka i njihovo prikazivanje.
//---------------------------------------------------------------
EncPodatak::EncPodatak( const Podatak& p )
    : _Id( ProcitajCeoBroj(Atribut(p,"id")) ),
      _Naslov( Atribut(p,"naslov") ),
      _Autor( Atribut(p,"autor") )
{}
```

```
//  Prikazivanje i pisanje
void EncPodatak::Prikazi( ostream& ostr ) const
{
    ostr << endl;
    PrikaziNaslov( ostr );
    PrikaziTekst( ostr );
    PrikaziDetalje( ostr );
    ostr << endl;
    PrikaziAutora( ostr );
    ostr << endl;
    PrikaziPovezane( ostr );
    ostr << endl;
}

//  Prikazivanje dela podatka
void EncPodatak::PrikaziNaslov( ostream& ostr ) const
{
    ostr << Naslov()
         << " (" << Tip() << " " << Id() << ")" << endl;
    ostr << "----------------------------------------" << endl;
}

void EncPodatak::PrikaziAutora( ostream& ostr ) const
{
    ostr << Autor() << endl;
}

//  Pomocni metodi za prikazivanje
void EncPodatak::PrikaziDimenzije( ostream& ostr, int s, int v )
{
    ostr << "Dimenzije: " << s << " x " << v << endl;
}

void EncPodatak::PrikaziTrajanje( ostream& ostr, double sec )
{
    ostr << "Trajanje: ";
    if( sec >= 60 ){
        int m = sec / 60;
        ostr << m << "m ";
        sec -= m * 60;
        }
    ostr << sec << "s" << endl;
}

void EncPodatak::PrikaziPovezane( ostream& ostr ) const
{
    vector<int> povezani;
    BazaPodataka::ProcitajPovezane( Id(), povezani );
    if( povezani.empty() )
        ostr << "Nema povezanih podataka." << endl;
    else{
        ostr << "Povezani podaci:" << endl;
        map<string,int> naslovi;
        for( unsigned i=0; i<povezani.size(); i++ ){
            EncPodatak* p
                = ProcitajPodatakIzBazePodataka( povezani[i] );
            if( p ){
                naslovi[ p->Naslov() + " (" + p->Tip() ]
                    = p->Id();
                delete p;
```

```
                }
            }
        map<string,int>::iterator
            i = naslovi.begin(),
            e = naslovi.end();
        for( ; i!=e; i++ )
            ostr << " - " << i->first << " "
                 << i->second << ")" << endl;
        }

}

//  "Konstruktori"
EncPodatak* EncPodatak::NapraviPodatak(
                            string tip, const Podatak& p
                            )
{
    EncPodatak* obj = 0;
    if( tip == "tekst" )
        obj = new EncTekst( p );
    else if( tip == "slika" )
        obj = new EncSlika( p );
    else if( tip == "zvuk" )
        obj = new EncZvuk( p );
    else if( tip == "film" )
        obj = new EncFilm( p );
    return obj;
}

EncPodatak* EncPodatak::ProcitajPodatakIzBazePodataka( int id )
{
    Podatak p;
    string tip;
    return BazaPodataka::ProcitajPodatak( id, p, tip )
                ? NapraviPodatak( tip, p )
                : 0;
}

//  Pomocni metodi za citanje broja iz niske
string EncPodatak::Atribut( const Podatak& p, string s )
{
    string v;
    Podatak::const_iterator i = p.find(s);
    if( i != p.end() )
        v = i->second;
    return v;
}
```

### *Datoteka EncKlase.h*

```
#ifndef EncKlaseH
#define EncKlaseH

#include <fstream>
using std::endl;
using std::ofstream;
using std::ios;

#include "EncPodatak.h"
```

```
//-------------------------------------------------------------
// Klasa EncTekst
//-------------------------------------------------------------
// Tekstualni enciklopedijski podatak.
//-------------------------------------------------------------
class EncTekst : public EncPodatak
{
public:
    //  Konstruktor
    EncTekst( const Podatak& p )
        : EncPodatak( p ),
          _Tekst( Atribut(p,"tekst") )
        {}

    //  Prikazivanje i pisanje
    bool ZapisiSadrzaj( string ) const
        { return true; }

protected:
    //  Pristupni metodi
    string Tekst() const
        { return _Tekst; }
    string Tip() const
        { return "tekst"; }

    //  Prikazivanje dela podatka
    void PrikaziTekst( ostream& ostr ) const
        { ostr << Tekst() << endl; }
    void PrikaziDetalje( ostream& ) const
        {}

private:
    //  Članovi podaci
    string  _Tekst;
};

//-------------------------------------------------------------
// Klasa EncBinarni
//-------------------------------------------------------------
// Osnova za sve binarne (multimedijalne) enciklopedijske podatke.
//-------------------------------------------------------------
class EncBinarni : public EncPodatak
{
public:
    //  Konstruktor
    EncBinarni( const Podatak& p )
        : EncPodatak( p ),
          _Napomena( Atribut(p,"napomena") )
        {}

    //  Prikazivanje i pisanje
    bool ZapisiSadrzaj( string imedat ) const
        {
        if( imedat.empty() )
            return false;

        ofstream f( imedat.c_str(), ios::binary );
        if( !f )
            return false;
        const string& s = BinarniSadrzaj();
```

```
        f.write( s.c_str(), s.length() );
        if( !f )
            return false;

        return true;
        }

protected:
    //  Pristupni metodi
    string Napomena() const
        { return _Napomena; }
    virtual const string& BinarniSadrzaj() const = 0;

    //  Prikazivanje dela podatka
    void PrikaziTekst( ostream& ostr ) const
        { ostr << Napomena() << endl; }

private:
    //  Članovi podaci
    string  _Napomena;
};

//---------------------------------------------------------------
// Klasa EncSlika
//---------------------------------------------------------------
// Slika iz enciklopedije.
//---------------------------------------------------------------
class EncSlika : public EncBinarni
{
public:
    //  Konstruktor
    EncSlika( const Podatak& p )
        : EncBinarni( p ),
          _Sirina( ProcitajCeoBroj(Atribut(p,"sirina")) ),
          _Visina( ProcitajCeoBroj(Atribut(p,"visina")) ),
          _Slika( Atribut(p,"slika") )
        {}

protected:
    //  Pristupni metodi
    int Sirina() const
        { return _Sirina; }
    int Visina() const
        { return _Visina; }
    const string& BinarniSadrzaj() const
        { return _Slika; }
    string Tip() const
        { return "slika"; }

    //  Prikazivanje dela podatka
    void PrikaziDetalje( ostream& ostr ) const
        {
        ostr << endl;
        PrikaziDimenzije( ostr, Sirina(), Visina() );
        }

private:
    //  Članovi podaci
    int     _Sirina;
    int     _Visina;
    string  _Slika;
```

```
};

//-------------------------------------------------------------
// Klasa EncZvuk
//-------------------------------------------------------------
// Zvuk iz enciklopedije.
//-------------------------------------------------------------
class EncZvuk : public EncBinarni
{
public:
    //  Konstruktor
    EncZvuk( const Podatak& p )
        : EncBinarni( p ),
          _Trajanje( ProcitajRealanBroj(Atribut(p,"trajanje")) ),
          _Zvuk( Atribut(p,"zvuk") )
        {}

protected:
    //  Pristupni metodi
    double Trajanje() const
        { return _Trajanje; }
    const string& BinarniSadrzaj() const
        { return _Zvuk; }
    string Tip() const
        { return "zvuk"; }

    //  Prikazivanje dela podatka
    void PrikaziDetalje( ostream& ostr ) const
        {
        ostr << endl;
        PrikaziTrajanje( ostr, Trajanje() );
        }

private:
    //  Članovi podaci
    double  _Trajanje;
    string  _Zvuk;
};

//-------------------------------------------------------------
// Klasa EncFilm
//-------------------------------------------------------------
// Film iz enciklopedije.
//-------------------------------------------------------------
class EncFilm : public EncBinarni
{
public:
    //  Konstruktor
    EncFilm( const Podatak& p )
        : EncBinarni( p ),
          _Sirina( ProcitajCeoBroj(Atribut(p,"sirina")) ),
          _Visina( ProcitajCeoBroj(Atribut(p,"visina")) ),
          _Trajanje( ProcitajRealanBroj(Atribut(p,"trajanje")) ),
          _Film( Atribut(p,"film") )
        {}

protected:
    //  Pristupni metodi
    int Sirina() const
        { return _Sirina; }
    int Visina() const
```

```
        { return _Visina; }
    double Trajanje() const
        { return _Trajanje; }
    const string& BinarniSadrzaj() const
        { return _Film; }
    string Tip() const
        { return "film"; }

    //  Prikazivanje dela podatka
    void PrikaziDetalje( ostream& ostr ) const
        {
        ostr << endl;
        PrikaziDimenzije( ostr, Sirina(), Visina() );
        PrikaziTrajanje( ostr, Trajanje() );
        }

private:
    //  Članovi podaci
    int     _Sirina;
    int     _Visina;
    double  _Trajanje;
    string  _Film;
};

//---------------------------------------------------------------
#endif  //  #ifndef EncKlaseH
```

### *Datoteka Enciklopedija.cpp*

```
#include <iostream>
using std::cout;
using std::cerr;
using std::endl;

#include "EncPodatak.h"

//---------------------------------------------------------------
// Program Enciklopedija.
//---------------------------------------------------------------
// Omogućava čitanje i prikazivanje podataka iz enciklopedije
// i zapisivanje binarnih sadržaja u datoteke.
//---------------------------------------------------------------
int main(int argc, char* argv[])
{
    // Ako nema dovoljno argumenata, ispišemo uputstvo
    if( argc < 2 ){
        cerr << "Upotreba: " << endl
             << "    " << argv[0] << " <id> [<bindat>]" << endl;
        return 2;
        }

    // Pročitamo redni broj i odgovarajući podatak
    int id = atoi( argv[1] );
    EncPodatak* p
        = EncPodatak::ProcitajPodatakIzBazePodataka( id );

    // Ako nema traženog podatka obavestimo korisnika i završimo
    if(!p){
        cerr << "Ne postoji podatak sa rednim brojem "
             << id << "!" << endl;
```

```
            return 1;
            }

        // Prikažemo podatak,
        p->Prikazi( cout );

        // Ako je na raspolaganju datoteka,
        // zapišemo binarni sadržaj podatka u datoteku
        if( argc > 2 ){
            if( !p->ZapisiSadrzaj( argv[2] ))
                cerr << "Nije uspelo pisanje u datoteku \""
                     << argv[2] << "\"!" << endl;
            }
        // inace proverimo da li je bila potrebna
        else if( !p->ZapisiSadrzaj( "" ))
            cout << "Binarni sadrzaj nije zapisan "
                 << "jer nije naveden naziv datoteke!" << endl;

        // Uklonimo viškove
        delete p;

        // Sve je u redu
        return 0;
    }
```

# 8.4 Rezime

Predlažemo da se za vežbu ovaj zadatak radi primenom druge ponuđene varijante za definisanje interne strukture: u baznoj klasi hijerarhije obezbeđuje se član podatak tipa `Podatak`; u klasama naslednicama nema potrebe za novim članovima podacima; pristupni metodi neposredno izdvajaju odgovarajuće sadržaje iz tog podatka.

Posebnu vežbu, koja se više tiče upoznavanja i korišćenja radnog okruženja (operativni sistem, razvojni alati i sl.), može predstavljati pisanje programa koji bi umesto zapisivanja binarnog sadržaja u datoteku izvodio predstavljanje tog binarnog sadržaja korisniku primenom odgovarajućih programa za pregledanje slika, slušanje zvučnih zapisa ili gledanje video zapisa.