Programabilni uređaji i objektno orjentisano programiranje

Nasljeđivanje

Nasljeđivanje

- Nasljeđivanje je čudan pojam za programiranje, ali predstavlja jedan od najvažnijih OO koncepata.
- Pored pojma nasljeđivanja **(uz razlike u semantici – značenju)** srijeću se i pojmovi generalizacije i izvođenja.
- Nasljeđivanje je fundamentalni koncept ljudske logike: na osnovu jednog koncepta zaključujemo o sličnim događajima.
- Da bi objasnili potrebu za nasljeđivanjem, posmatrajmo sljedeći misaoni eksperiment.
- U sistemu smo realizovali klasu Radnik i na nju potrošili dosta vremena i truda. U sistemu treba kreirati i klasu Šef.
- Klasa Šef je veoma slična klasi Radnik, ali ipak ima i neke izmijenjene funkcionalnosti (dodate ili, recimo, promijenjene članove).
- Kako ovo možemo izvesti?

Radnik - Šef

- Jedan od načina je da realizujemo klasu Šef od početka, realizujući sve metode za nju, pa i one koje su iste kao u klasi Radnik. Stoga se programiranje svodi na copy-paste operaciju za veliki dio koda.
- Ovo nije dobro, jer se može dogoditi da početni dizajn nekog metoda za klasu Radnik nije bio dobar, pa smo tu grešku prenijeli i u klasu Šef. Isti problem postoji i ako je i polazni metod dobar, a u međuvremenu se pojavila potreba za njegovom promjenom.
- Sada bi izmjena metoda značila da editujemo na više mjesta kod.
- Problem se multiplikuje ako u sistemu postoji više klasa sličnih Radniku: Honorarac, Magacioner, Spoljni, Inspektor, ...
- U tom slučaju, editovali bismo na mnoštvo mijesta, a to povlači mnoštvo šansi za pogrešku. Kod postaje nepregledan i nepraktičan, i naša priča o OO programiranju bi se vratila na početak, na nejasan i veliki kod koji je težak za održavanje.

Radnik - Šef

- Rješenje srećom postoji i ogleda se u konceptu nasljeđivanja: Neka klasa sadrži sve što i druga uz još neke eventualne funkcionalnosti ili možda izmjene.

- Na ovaj način se jedna stvar, odnosno funkcija, ne mora ponavljati u sličnim klasama.

Nasljeđivanje u kodu C++

- Nasljeđivanje je prirodan koncept, ali nije jednostavan za kodiranje.
- Programerska teorija poznaje više vrsta nasljeđivanja i, što je nekarakteristično za današnje programske jezike, C++ ih sve podržava.
- Nasljeđivanje se naglašava u zaglavlju izvedene klase, na sljedeći način:

```
class izvedena_klasa:tip_nasljeđivanja osnovna_klasa
{//razlike i dodaci u odnosu na osnovnu klasu
};
```

- Dakle, na ovaj način saopštavamo da imamo klasu koja se naziva izvedenom (engl. derived), koja ima sve što i osnovna klasa (engl. base), uz eventualne dodatke i izmjene koji su saopšteni u tijelu klase.

Primjer nasljeđivanja

- Prije nego damo jednostavan primjer nasljeđivanja, napomenimo da ćemo obično kao tip nasljeđivanja koristiti public (javno nasljeđivanje ili izvođenje), ali ćemo to nešto kasnije objasniti.

```
class Student{
      protected:
            int gs,gr;
      public:
            Student();~Student();
            int DajGs() const; int DajGr() const;
            void SetGs(int); void SetGr(int);
};
```

Ilustrativna osnovna klasa ima samo konstruktor, destruktor, osnovne inspektore i mutatore i ništa više.

- Osnovna uočena razlika je nova protected (zaštićena) sekcija!
- Elementi ove sekcije su vidljivi unutar posmatrane klase, kao i klasa izvedenih iz te klase, ali ne i "spolja".

Primjer nasljeđivanja

- Izvedena klasa:

```
class Diplomac:public Student{
    protected:
          char Diplomski[20];
    public:
          Diplomac();
          ~Diplomac(); //itd.
};
```

- Sada se može deklarisati:

```
Diplomac Marko;
Marko.SetGs(4);
```

Ne nasljeđuju se:

konstruktori
destruktori
operator dodjele
prijatelji!

Sve ostalo se nasljeđuje (na ovaj ili onaj način)!

- `SetGs()` je naslijeđena iz osnovne klase `Student`.
- Memorija objekta izvedene klase (bez obzira na tip nasleđivanja) se sastoji od podobjekta osnovne klase i dodataka u izvedenoj klasi!

Tipovi nasljeđivanja

- private – privatno nasljeđivanje (ovo je podrazumijevano i biće tako ako se izostavi tip nasljeđivanja). Kod ovog tipa nasljeđivanja svi elementi interfejsa osnovne klase postaju privatni i metodi iz izvedene klase im ne mogu pristupiti. Ovakav tip nasljeđivanja naziva se ponekad sadržavanjem. Ovo nije rijedak, ali ni dominantan tip nasljeđivanja.
- public – javno nasljeđivanje. Kod ovog tipa nasljeđivanja svi elementi iz osnovne klase u izvedenoj klasi zadržavaju vidljivost (public – ostaje public, protected – protected i private – private). Metodi iz izvedene klase mogu pristupati public i protected elementima interfejsa osnovne klase. Ovo je dominantan tip nasljeđivanja – pravo nasljeđivanje.

Tipovi nasljeđivanja

- Moguć, ali nepraktičan i izuzetno rijetko se primjenjuje, je treći tip nasljeđivanja: protected – zaštićeno nasljeđivanje. Kod ovog nasljeđivanja javni djelovi interfejsa osnovne klase postaju u izvedenoj klasi zaštićeni, dok ostali elementi zadržavaju vidljivost iz prethodnog slučaja.
- Može se dogoditi i sljedeća situacija: Izvršili smo privatno nasljeđivanje, ali, recimo, želimo da jedan dio interfejsa osnovne klase ostane zaštićen ili javan.
- To se radi tako što se u interfejsu izvedene klase naglase elementi kojima se želi zadržati vidljivost kao što je bila u osnovnoj klasi.

Vidljivost kod izvedene klase

- Primjer zadržavanja vidljivosti kao u osnovnoj klasi:

```
class B{private: int a; protected: int b;
public: int c;};
class D:private B{protected: int B::b;};
```

- Na ovaj način su svi elementi iz osnovne klase postali privatni u izvedenoj klasi, osim elementa **b** koji je ostao zaštićen.
- Uočite upotrebu operatora dosega preko kojega smo naglasili da je u pitanju element iz osnovne klase. Da smo ga izostavili, deklarisali bismo ovako **b** kao podatak izvedene klase!!!
- Važno: Izvedena klasa može da zadrži vidljivost elemenata osnovne klase ili da vidljivost pogorša, ali ne može da popravlja vidljivost (recimo, sa privatne na javnu), jer bi na taj način ugrozili enkapsulaciju, odnosno potpunu kontrolu klase nad samom sobom (neko spolja bi mogao da učini djelove interfejsa vidljivim spolja).

Primjer prava pristupa

- Posmatrajmo sljedeći primjer:

```
class B {
    int i;
    public: void f();};
class D:public B{
    int j;
    public: void g();};
B b; D d;
b.f(); d.g(); d.f(); //dozvoljeni pozivi
b.g(); //nije dozvoljeno, osnovna klasa
// nema informacija o izvedenoj klasi!
```

- U slučaju da u klasi **B** imamo privatni metod, taj metod se ne bi mogao pozivati preko objekata klase **D**, niti iz funkcija članica klase **D**. Eventualno se može pozivati neka javna ili zaštićena funkcija klase **B** koja zatim poziva privatne funkcije članice.

Pravo pristupa

- Dozvoljeno je da u izvedenoj klasi imamo preklopljenu funkciju iz osnovne klase.
- Ta funkcija se mora razlikovati po argumentima, odnosno ne smije se razlikovati samo po tipu rezultata koji vraća.
- U slučaju da se pozove data funkcija za objekat izvedene klase, bira se ona iz izvedene ili osnovne klase koja po argumentima najbolje odgovara pozvanoj funkciji bez "gledanja " u tip vidljivosti.
- "Najbolja" funkcija se zatim poziva, ali ako nemamo pravo pristupa njoj onda kompajler prijavljuje grešku.
- Izbor najbolje funkcije se obavlja nezavisno od vidljivosti!
- Pored preklapanja metoda, postoji i zasjenjivanje metoda, ali ćemo to upoznati nešto kasnije.

Konstrukcija/destrukcija objekata izvedene klase

- Podsjetimo se da izvedena klasa ne nasljeđuje konstruktor i destruktor osnovne klase, jer se objekat izvedene klase sastoji od podobjekta osnovne klase i dodatka iz izvedene klase, tako da nije dovoljno preuzeti ove funkcije iz osnovne klase. Odnosno, ne možemo konstruktorom iz iosnovne klase inicijalizovati (čak ni zauzeti memoriju) sve podatke članove izvedene klase

Primjer konstruktora

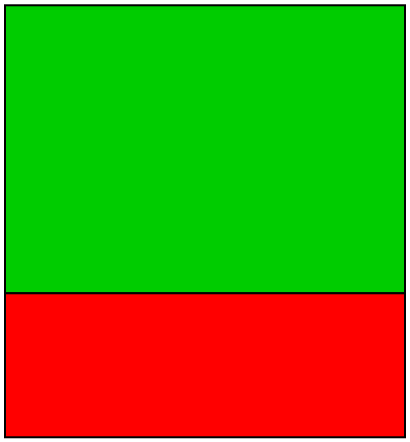
```
class B{
      private:
            int b;
      public:
            B(){}
            B(int i):b(i){/*koješta*/}
};
class D:public B{
      private:
            int d;
      public:
            D(){}
            D(int); /*koješta*/};
            D::D(int a):B(a),d(a+1){};
```

Ovo je interesantan dio, jer je ovo dio sekcije za incijalizaciju u kojoj se podobjekat osnovne klase inicijalizuje na vrijednost a pozivanjem konstruktora osnovne klase!

Redosljed operacija kod konstrukcije i destrukcije

- Prilikom kreiranja objekta izvedene klase prvo se poziva konstruktor osnovne klase, zatim se vrši inicijalizacija objekta osnovne klase (ako sekcija za inicijalizaciju postoji), tek na kraju se izvršava konstruktor izvedene klase.
- Destruktor se izvršava suprotnim redosljedom: destruktor izvedene klase, sa ukidanjem članova koje je izvedena klasa dodala, i na kraju se poziva destruktor osnovne klase.
- Pri svim ovim operacijama nije neophodno da se eksplicitno iz izvedene klase pozivaju konstruktor i destruktor osnovne klase, već će se to dogoditi implicitno (prilikom implicitnog poziva se poziva default konstruktor – ukoliko želimo poziv nekog drugog konstruktora, to moramo eksplicitno učiniti).

Memorija kod izvedene klase



Imajte na umu da memorija koja se zauzima za objekte izvedene klase izgleda na način kao što je dato na slici. Pored podobjekta osnovne klase postoji i dodatak koji "kalemi" izvedena klasa.

- Dozvoljeno je u izvedenoj klasi imati promljenljivu koja se zove isto kao promjenljiva u osnovnoj klasi. To su, zapravo, dvije promjenljive - jedna pripada "zelenom" dijelu sa dijagrama gore, dok druga pripada "crvenom" dijelu.
- Da bi se pristupilo iz metoda izvedene klase promjenljivoj iz osnovne klase može se, a u slučaju promjenljivih sa istim imenom `i` mora, koristiti operator dosega `B::i`.

Prosto izvođenje (nasljeđivanje)

- Prostim izvođenjem (nasljeđivanjem) se naziva situacija kada jedna klasa nasljeđuje osobine jedne osnovne klase.
- Prosto nasljeđivanje uključuje i situaciju kada je jedna klasa izvedena na osnovu neke druge klase, a ona je ujedno i osnovna za neku treću klasu:

```
class A{};
class B: public A{};
class C: public B{};
class D: private A{};
```

Ovo su sve primjeri prostog nasljeđivanja jer jedna klasa zna da nasljeđuje samo jednu osnovnu klasu. Sada se objekat klase C sastoji od podobjekta klase B (ovaj je jednak podobjektu klase A, plus dodatak koji je dodala klasa B), plus dodatak koji dodaje klasa C.

Primjer pozivanja funkcija

```
class A {
  public:
      void f();};
class B: public A{};
class C: public B
  {void f();};
void C::f(){
f(); //rekurzivni poziv f-je iz klase C
A::f(); //iz klase A
B::f(); //kako nema f-je u klasi B
//poziva se ona iz osnovne klase, odnosno A::f();
}
```

Konverzija pokazivača na izvedenu klasu

- Moguće je izvršiti konverziju pokazivača na izvedenu klasu u pokazivač na osnovnu klasu pod uslovom da je osnovna klasa dostupna.
- Kažemo da je osnovna klasa dostupna onda kada su joj, na mjestu na kojem se vrši konverzija, vidljivi javni članovi, što se svodi na to da je izvođenje javno.
- Za sada ova operacija djeluje veoma nebitno i može predstavljati samo jednu od "atrakcija" koje nam C++ nudi, ali će se pokazati nešto kasnije da omogućava pisanje izuzetno elegantnih programskih rješenja u nekim situacijama.

Konverzija pokazivača - primjer

```
class B{/**/};
class Der1:private B{/**/};
class Der2:public B{/**/};
main(){
Der1 d1; //objekat jedne od izvedenih klasa
Der2 d2;
B *pb1=&d1; //nije dozvoljeno da pb1 pokazuje na d1
            //jer u Der1 osnovna klasa nije dostupna
B *pb2=&d2; //dozvoljeno
}
```

Isto pravilo važi i za referencu!

Referenca na objekat javno izvedene klase može se konvertovati u referencu na objekat osnovne klase.

Još oko konverzija

- Konverzija objekta izvedene klase u objekat osnovne klase nije predviđena jezikom (za razliku od konverzije kod pokazivača i referenci), ali se podrazumijeva da je uvijek moguće inicijalizovati objekat osnovne klase objektom izvedene klase, jer izvedena klasa sadrži podobjekat osnovne klase, pa nije problem da objekat osnovne klase preuzme taj dio za sebe.
- U slučaju potrebe za drugim oblicima konverzija: iz izvedene klase u osnovnu i obrnuto možete se uvijek poslužiti konstruktorom sa odgovarajućim argumentom!!!

Višestruko nasljeđivanje

- Pored prostog nasljeđivanja, C++ posjeduje mogućnost višestrukog nasljeđivanja, odnosno situacije kada jedna klasa nasljeđuje direktno osobine više osnovnih klasa.
- Većina današnjih OO jezika ne podržava ovu opciju (recimo Java, C#).
- Pored toga, kompletan VCL (Visual Component Library) u Borland Builderu je realizovan kroz složenu hijerarhiju prostog nasljeđivanja.
- Dakle, gotovo sve što se može uraditi može se uraditi preko prostog nasljeđivanja, a ujedno je prosto nasljeđivanje u modi i neki jezici zabranjuju višestruko nasljeđivanje.

Višestruko nasljeđivanje - Ipak

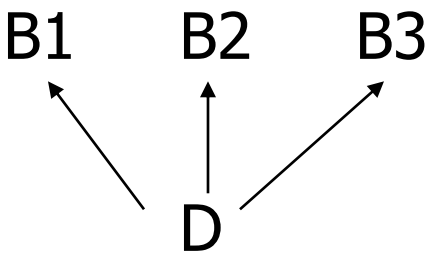
- Savremena programerska teorija (u ovoj oblasti teorija često prednjači u odnosu na praksu) ukazuje da višestruko nasljeđivanje može da riješi neke probleme na mnogo elegantniji način nego prosto nasljeđivanje, te da ima ogromne aplikacione potencijale.
- Dodatni problem je što neki programski jezici uvode nove programerske koncepte da bi odradili ono što se može uraditi sa C++-om.
- Dakle, da zaključimo: višestruko nasljeđivanje, premda stvar koja trenutno nije u modi, veoma je dobar programerski koncept i velika je "sreća" što ćemo ga izučiti u okviru C++-a kao programskog jezika koji ovaj koncept podržava.

Sintaksa višestrukog nasljeđivanja

- Sve klase koje jedna klasa nasljeđuje navode se u zaglavlju klase. Ukoliko se ne navede tip nasleđivanja podrazimijeva se private:

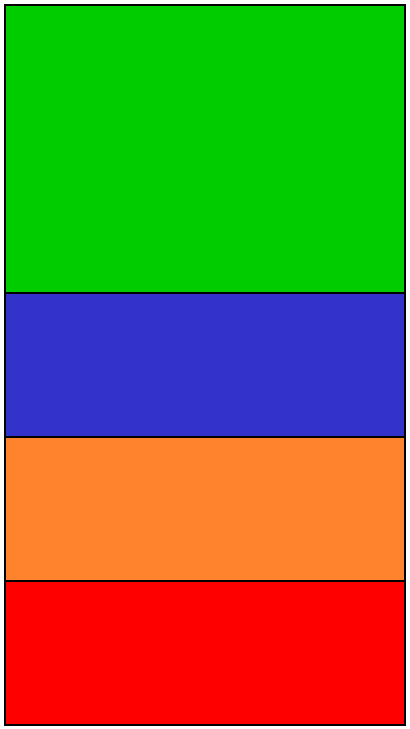
```
class D: public B1,B2, private B3 {/**/};
```

- Kod programskih sistema koji se dizajniraju višestrukim nasljeđivanjem programeri često koriste "trikove" za vizuelizaciju odnosa između pojedinih klasa.
- Jedan od najpoznatijih metoda je zasnovan na usmjerenom acikličnom grafu nasljeđivanja:



Klasa D nasljeđuje osobine tri osnovne klase.

Memorijska reprezentacija



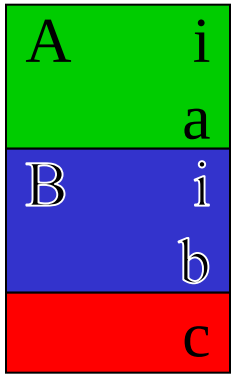
Moguće je da u hijerarhiji nasljeđivanja postoje dvije promjenljive u pojedinim osnovnim klasama koje se zovu isto. U tom slučaju se za razrješenje kojoj se promjenljivoj pristupa koristi operator dosega, recimo `B1::i` ili `B2::i`. Ako postoji dvosmislenost doći će do prekida izvršavanja programa.

Primjer višestrukog nasljeđivanja

- Ovo je samo generički primjer koji ilustruje prethodnu priču:

```
class X{public: int i;};
class A : public X{public: int a;};
class B : public X{public: int b;};
class C : public A, public B {
    public:
        int c;
        void f();};
```

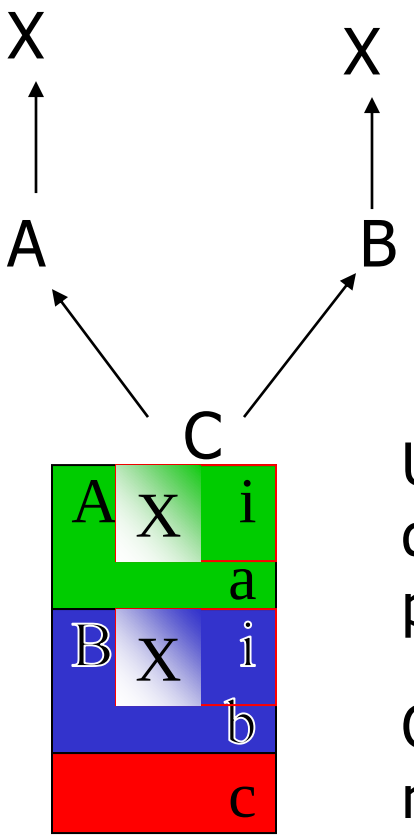
- Objekat klase C sada izgleda kao:




```
void C::f(){
      i=0; //ne može, ne zna se koje i
      A::i=0; //dozvoljeno
}
```

Graf nasljeđivanja - Ilustracija

- Graf nasljeđivanja iz prethodnog primjera je:



Nije dozvoljeno:

```
public C:public B, public B{};
```

jer se dvosmilenost u pristupu članovima klasa B nikako ne bi mogla izbjeći.

U primjeru kojeg smo analizirali klasa C je imala dva podobjekta koji potiču iz klase X: jedan je pokupljen preko klase A, a drugi preko klase B.

Ovo je ponekad ono što se traži, ali je često nepotrebno i nešto što se želi izbjeći.

Virtuelno nasljeđivanje

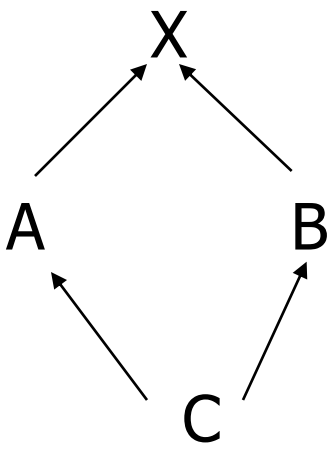
- Virtuelno nasljeđivanje je jednostavna tehnika kojom se može izbjeći ugradnja višestrukih podobjekata **(koji potiču od iste klase u hijerarhiji nasljeđivanja).**
- Recimo, svi studenti imaju ime i prezime, iz studenta su izvedene klase stud_akad_stud i diplomac, dok klasa stud_diplomac_na_akad nasljeđuje osobine i stud_adak_stud i diplomac, ali ne mora da naslijedi ime i prezime od osnovne klase dva puta.
- Nadam se da je sada jasno zbog čega je virtuelno nasljeđivanje koje sprečava nasljeđivanje višestrukih podobjekata neophodno.

Virtuelno nasljeđivanje - Primjer

- Koristimo isti malopređašnji primjer:

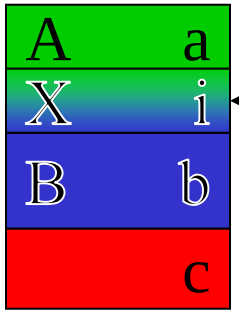
```
class X{public: int i;};
class A: virtual public X{public: int a;};
class B: virtual public X{public: int b;};
class C : public A, public B {
    public:
          int c;
          void f();};
```

Ključna riječ virtual znači da u daljem nasljeđivanju iz klase B, odnosno iz A, podobjekat klase X može biti zajednički, odnosno dijeljen sa drugim klasama koje imaju ovaj podobjekat.



Virtuelno nasljeđivanje - Memorija

- Memorijska reprezentacija u ovom slučaju **može** izgledati kao:



Podobjekat X kojega dijele A i B.

Rekli smo može, jer će najvjerovatnije (to je stvar diskrecije kompajlera) biti alocirana memorija za X na "vrhu" klase A, a zatim će nakon dodatka od strane klase A biti dodat podobjekat klase B koji nema "fizičku" vezu sa klasom X (podobjekat klase X je u ovom slučaju razdvojen od B) ali to i dalje ne smeta da je A::i u ovom slučaju jednako B::i.