Napredne tehnike programiranja

Vježba 8

Teme

- Odnosi među klasama
  - Nasljeđivanje
  - Ugnježdene klase

Pojam nasljeđivanja

- Jedan on najvažnijih OOP koncepata
- Omogućava ponovno iskorištavanje koda tako što se koriste postojeće klase za kreiranje novih

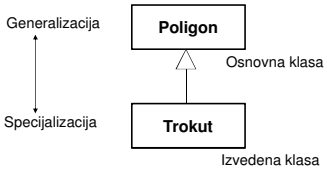
- Najprije se kreira općenita klasa A
- Zatim se kreira izvedena klasa B (koja nasljeđuje sve elemente klase A)
- Na kraju se u klasi B definiraju elementi koji su specifični za klasu B (nema ih u klasi A)

Nasljeđivanje

- Nasljeđivanje omogućava da jedna klasa preuzme svojstva druge klase (podatkovne članove i funkcije)
- Klasa koja nasljeđuje se zove izvedena klasa (dijete), a klasa od koje se nasljeđuje se zove bazna klasa (roditelj)
- Dijete može imati i dodatne podatkovne članove i funkcije koji nisu naslijeđeni

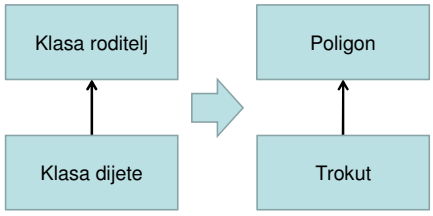
Nasljeđivanje

- Nasljeđivanje opisuje vezu tipa 'jest':
  - Trokut jest poligon



Nasljeđivanje

- Klasa roditelj – općenita klasa
- Klasa dijete – podvrsta roditelja



Nasljeđivanje

- Sintaksa deklaracije izvedene klase:

```
class izvedena_klasa : public osnovna_klasa
{
   // definiranje dodatnih  članova klase
}
```

Nasljeđivanje

- **Osoba.h** (zaglavlje bazne klase *Osoba*)

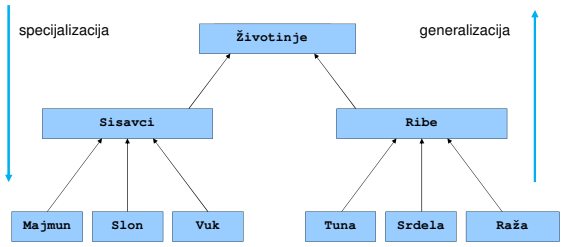
```
class Osoba{
  protected:
        string JMBG;
  public:
        string ime;
        string prezime;
        void Radi();
        void Odmori();
};
```

- **Student.h** (zaglavlje klase *Student*)

```
class Student : public Osoba{
 public:
        string JMBAG;
        string NazivStudija;
        int semestar;
        void Uci();
};
```

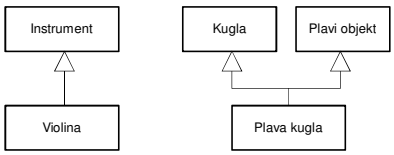
```
int main(){
        Student Ante;
        Ante.ime = "Ante";
        Ante.prezime = "Antic";
        Ante.JMBAG = "024600112233";
        Ante.NazivStudija = "TVZ";
        Ante.semestar = 3;
        Ante.Radi();
        Ante.Uci();
        Ante.Odmori();
```

Hijerarhija nasljeđivanja



Višestruko nasljeđivanje

- Izvedena klasa može naslijediti jednu ili više osnovnih klasa



Nasljeđivanje

- Jednostruko naslijeđivanje:
  `class Medvjed: public Zivotinja{ ... };`
- Višestruko naslijeđivanje:
  `class Cekic: public Alat,public Predmet { ... };`
- Za svaku navedenu baznu klasu potrebno je specificirati način pristupa (public, protected ili private)

Nasljeđivanje

- Primjer:

```
class Temelj {
 public;
     Temelj() { elem0=0; }
     int elem0;
};
class Izveden : public Temelj {
 public:
     Izvedena() {elem1 = 0}
     int elem1;
};
```

Nasljeđivanje

- Primjer

```
class Prva {
      public;
            int a;
};
class Druga: public Prva{
      public:
            int b;
};
```

Deklaracija objekta izvedene klase:

```
Druga objekt1;
objekt1.a=1;
objekt1.b=2;
```

Nasljeđivanje – prava pristupa

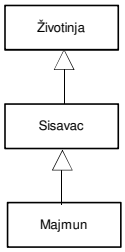
```
class Osnovna
{
      public:
            int x;
      protected:
            int y;
      private:
            int z;
};

class Izvedena: public Osnovna
{
      // x is public
      // y is protected
      // z nije dostupan
};
```

Privatni dio osnovne klase nije dostupan iz nasljeđene

Nasljeđivanje

- Klasa dijete može imati više predaka (postoji cijela hijerarhija nasljeđivanja

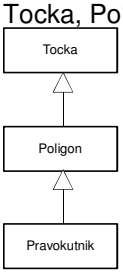


Zadatak

- Napravite hijerarhiju klasa: Tocka, Poligon, Trokut

Podatkovni članovi:

Tocka:
int x
int y
Poligon:
-
Pravokutnik:
int a
int b



Nasljeđivanje - prekrivanje

```
#include <iostream.h>
class Osnovna {
public:
      int i; //clan i u baznoj klasi
      void Var() { cout << "Osnovna::i " << i << endl; }
};
class Izvedena : public Osnovna {
public:
      int i; // prekriva Osnovna::i !!!
      void Int() { cout << "Izvedena::i " << i << endl; }
};
int main() {
      Izvedena izv;
      izv.i = 9; // pristupa se Izvedena::i
      izv.Osnovna::i = 20; // pristupa se Osnovna::i
      izv.Var(); // ispisuje 'Osnovna::i 20'
      izv.Int(); // ispisuje 'Izvedena::i 9'
      return 0;
}
```

Nasljeđivanje

```
class A {
public:
      void Opis() { cout << "Ovo je klasa A" << endl; } //!!!
};
class B {
public:
      void Opis() { cout << "Ovo je klasa B" << endl; } //!!!
};
class D : public A, public B { };

int main(){
      D obj;
      obj.Opis(); // pogreška: dvosmislenost
      obj.A::Opis(); // ispisuje 'Ovo je klasa A'
      obj.B::Opis(); // ispisuje 'Ovo je klasa B'
```

Ugniježđene klase

- Ugniježđena klasa je klasa definirana unutar neke druge
- klase ("klasa u klasi")
- može se koristiti samo od klase u koju je ugniježđena

Podstrukture (ugnježđene strukture)

```
struct B{
	int x;
	//B(){x=0;}	// nema konstruktor bez parametara
	B(int t){x=t;} //konstruktor sa parametrom
};

struct A{
	B b; //podstruktura
	double c; //standardni tip
	/*A():b(0){
	} */

};

int main(){
	A a; // In constructor 'A::A()': [Error] no matching
function for call to 'B::B()'- Greška kod prevodenja, automatski
se kreira defaultni konstrukor za A ali nema onog za B
}
```

Rješenje: možemo ručno kreirati konstruktor za B ili ga eksplicitno pozvati iz konstruktora za A

Podstrukture (ugnježđene strukture)

```
#include <iostream>
using namespace std;
struct B{
	int x;
	B(){x=0;}	//konstruktor bez parametara
	B(int t){x=t;} //konstruktor
};

struct A{
	B b; //podstruktura
	double c; //standardni tip
	/*A():b(0){
	} */

};
int main(){
	A a; // sad je OK program ne javlja grešku kod prevođenja
```

Rješenje 1 : podstruktura B ima default konstruktor koji se poziva kad kreiramo objekt A

Podstrukture (ugnježđene strukture)

```
#include <iostream>
using namespace std;
struct B{
	int x;
	//B(){x=0;}	// nema konstruktor bez parametara
	B(int t){x=t;} //konstruktor
};

struct A{
	B b; //podstruktura
	double c; //standardni tip
	A():b(0){} // A konstruktor poziva konstruktor za B
};
int main(){
	A a; // sad je OK program ne javlja grešku kod prevođenja
```

Rješenje 2 : konstruktor za A eksplicitno zove konstruktor za B

Podkonstruktori

- Ako u strukturi A imamo podstrukturu B koja ima samo konstruktor sa parametrima, onda konstruktor od A mora "pretkonstruirati" B

```
struct B { int x; B(int t){x=t;}};

struct A { B b; double c;
     A(int y, double z) : b(y) {
c=z;}
}; //podkonstruktor kreira b
```

Podkonstruktori

- Podkonstruktori rade i za standardne tipove podataka
- U priloženom primjeru konstruktor za A predkonstruira i B i c

```
struct B { int x; B(int t){x=t;}};

struct A { B b; double c;
     A(int y, double z) : b(y), c(z){ }
}; //podkonstruktor kreira b i c
```

Podstrukture

- Zaključak:
  - Ako eksplicitno kreiramo konstruktor bez parametara za sve strukture izbjegli smo probleme
  - Ako ne kreiramo esplicitno kostruktor bez parametara za podstrukturu onda moramo eksplicitno pozivati konstruktor

Podstrukture

- Ako se objekt B može pojaviti samo kao dio objekta A onda njegovu deklaraciju možemo staviti u deklaraciju od A

```
struct Automobil {
      struct Motor {
            int snaga;
            double obujam;
            Motor( int s, double o )
            { snaga = s; obujam = o; }
      };
      Motor m;
      int brojVrata;
      Automobil() : m( 90, 1.4 ) { ... }
      Automobil( int s, double o ) : m( s, o ) {
      ... }
};
```

Motor je podstruktura automobila i postoji samo kao dio njega

Podstrukture

- Možemo li tada podstrukturu kreirati kao samostalni objekt (motor koji ne pripada nekom automobilu)

Kod kreiranja samostalne podstrukture moramo navesti punu stazu do nje koristeći operator ::

```
Automobil yugo( 45, 1.2 );
Motor tdi( 120, 1.9 ); // krivo!
Automobil::Motor turboDiesel( 120, 1.9 ); // OK
yugo.m = turboDiesel;

cout << yugo.m.snaga;
cout << yugo.brojVrata;
```