Objektno orijentisano programiranje

Nasleđivanje

Nasleđivanje

- Nasleđivanje je veza između klasa koja podrazumeva preuzimanje sadržaja nekih klasa, odnosno **klasa-predaka** i na taj način, uz mogućnost modifikacije preuzetog sadržaja i dodavanje novog dobija se **klasa-potomak**.
- Nasleđivanje je jedna od najznačajnijih karakteristika objektno orijentisanog programiranja.
- Nasleđivanje je veza između klasa koja opisuje odnos “izvodi se od”.
- Klasa (kojih može biti i više) od koje se preuzima sadržaj, tj. koja se nasleđuje naziva se klasa-predak, osnovna (bazna) klasa, klasa davalac ili natklasa.
- Klasa koja prima sadržaj, tj. klasa koja nasleđuje naziva se klasa-potomak, izvedena klasa, klasa-primalac ili potklasa.

Nasleđivanje

- Objekti izvedene klase poseduju sve sadržaje, sa ili bez modifikacija, koje poseduju i objekti njihove bazne klase, a pored toga poseduju i sadržaj koji je karakterističan samo za njih.
- U svakom objektu izvedene klase može se razlikovati roditeljski deo i deo koji je specifičan za samog potomka.
- Primer:
  - Klasa osoba ima polja ime i prezime, klasa student ima polja ime, prezime i broj indeksa, klasa lekar ima polja ime, prezime i broj licence. Za sve tri klase zajednička su polja ime i prezime, a klase student i lekar imaju dodatna polja koja ih opisuju. Dakle, klasa osoba je bazna klasa, a klase student i lekar koje je nasleđuju su izvedene klase.

Sintaksa

- Klasa Naslednik je izvedena klasa i ona nasleđuje klasu Roditelj koja je bazna klasa.

```
class Roditelj{

    ...

};

class Naslednik: način nasleđivanja Roditelj{

    ...

};
```

Način nasleđivanja

- Način nasleđivanja može biti:
  - **private**
  - **public**
  - **protected**
- U svim zadacima na vežbama, radimo **public** način nasleđivanja.
- Na predavanjima ćete čuti više o različitim načinima nasleđivanja.
- U zbirci koja prati predmet možete pročitati više o drugim načinima nasleđivanja i tome kako oni utiču na ponašanje izvedene klase.

Prava pristupa delovima klase

- Prilikom public nasleđivanja:
  - Članu klase koji je **private** može se direktno pristupati samo iz metoda te klase i njenih prijateljskih funkcija.
  - Članu klase koji je **protected** može se direktno pristupati iz metoda te klase, njenih prijateljskih funkcija i metoda njenih izvedenih klasa.
  - Potrebno je da izvedena klasa ima pristup poljima (i metodama) koje je nasledila. Isto tako, potrebno je da pristup poljima i dalje bude zabranjen ostalim funkcijama van klase i izvedenih klasa. Zbog toga najčešće koristimo **protected** pravo pristupa za sva polja u klasi koju želimo da nasledimo (baznoj klasi).

Šta se nasleđuje, a šta ne?

- Nasleđuju se:
  - polja
  - metode
- Ne nasleđuju se:
  - konstruktori
  - destruktori
  - prijateljske funkcije

Primer 1 - Klasa Trougao

1. Napisati klasu **Trougao**. Iz klase Trougao izvesti klasu **JKTrougao** (jednakokraki trougao). Iz klase JKTrougao izvesti klasu **JSTrougao** (jednakostranični trougao).

Rešenje: Trougao

```cpp
#ifndef TROUGAO_HPP_INCLUDED
#define TROUGAO_HPP_INCLUDED

#include <cmath>

class Trougao {
    protected:
        double a, b, c;
    public:
        Trougao() { a = 3; b = 4; c = 5; }
        Trougao(double aa, double bb, double cc) { a = aa; b = bb; c = cc; }
        Trougao(const Trougao &t) { a = t.a; b = t.b; c = t.c; }

        double getA() const { return a; }
        double getB() const { return b; }
        double getC() const { return c; }

        void setA(double aa) { a = aa; }
        void setB(double bb) { b = bb; }
        void setC(double cc) { c = cc; }

        double getO() const { return a + b + c; }

        double getP() const {
            double s = (a + b + c) / 2;
            return sqrt(s * (s - a) * (s - b) * (s - c));
        }
};

#endif // TROUGAO_HPP_INCLUDED
```

pošto će klasa biti nasleđena umesto private pišemo protected za polja

Rešenje: JKTrougao

- **JKTrougao** nasleđuje klasu **Trougao**,
  - na taj način preuzima polja a, b i c pošto i jednakokraki trougao ima tri stranice.
  - na taj način preuzima sve metode (osim konstruktora) iz klase **Trougao**, pošto i za jednakokraki trougao važe iste metode.
  - konstruktore ne nasleđuje: u konstruktoru treba “podesiti” stranice a, b i c, tako da dve stranice budu jednake.

Rešenje: JKTrougao

jktrougao.hpp

```
1   #ifndef JKTROUGAO_HPP_INCLUDED
2   #define JKTROUGAO_HPP_INCLUDED
3
4   #include "Trougao.hpp"
5
6   class JKTrougao : public Trougao {
7       public:
8           JKTrougao() : Trougao(1, 2, 2) {}
9           JKTrougao(double aa, double bb) : Trougao(aa, bb, bb) {}
10          JKTrougao(const JKTrougao &jkt) : Trougao(jkt.a, jkt.b, jkt.c) {}
11  };
12
13  #endif // JKTROUGAO_HPP_INCLUDED
```

Rešenje: JKTrougao

- Obratiti pažnju da u klasi **JKtrougao** nemamo polja, jer su nasleđena iz trougla.
- U klasi **JKTrougao** pomoću konstruktora inicijalizatora kreiramo roditeljski deo klase **JKTrougao** - klasu **Trougao**.
- Trouglu treba proslediti vrednosti stranica tako da dve budu međusobno jednake:

```
JKTrougao() : Trougao(1, 2, 2) {}

JKTrougao(double aa, double bb) : Trougao(aa, bb, bb) {}

JKTrougao(const JKTrougao &jkt) : Trougao(jkt.a, jkt.b, jkt.c) {}
```

Konstruktor inicijalizator

Kod kompozicije

```
class A{
private:
    int a;
public:
    A(){a=1;}
};

class Kompozicija{
private:
    A nazivObjekta;
public:
    Kompozicija():nazivObjekta(){}
};
```

Kod nasleđivanja

```
class A{
private:
    int a;
public:
    A(){a=1;}
};

class Naslednik:public A{
public:
    Naslednik():A(){}
};
```

Rešenje: JSTrougao

- **JSTrougao** nasleđuje klasu **JKTrougao**,
  - na taj način preuzima polja a, b i c među kojima su sve tri jednake pošto jednakostranični trougao ima takve stranice.
  - na taj način preuzima sve metode (osim konstruktora) iz klase **JKTrougao**, pošto i za jednakostranični trougao važe iste metode.
  - konstruktore ne nasleđuje: u konstruktoru treba “podesiti” stranice a, b i c, tako da sve tri stranice budu jednake.

Rešenje: JSTrougao

jstrougao.hpp

```
1  #ifndef JSTROUGAO_HPP_INCLUDED
2  #define JSTROUGAO_HPP_INCLUDED
3
4  #include "JKTrougao.hpp"
5
6  class JSTrougao : public JKTrougao {
7      public:
8          JSTrougao() : JKTrougao(1, 1) {}
9          JSTrougao(double aa) : JKTrougao(aa, aa) {}
10         JSTrougao(const JSTrougao &jst) : JKTrougao(jst.a, jst.b) {}
11 };
12
13 #endif // JSTROUGAO_HPP_INCLUDED
```

Rešenje: JSTrougao

- U klasi **JSTrougao** pomoću konstruktora inicijalizatora kreiramo roditeljski deo klase **JSTrougao** - klasu **JKTrougao**.
- Trouglu treba proslediti vrednosti stranica tako da sve tri budu međusobno jednake:

```
JSTrougao() : JKTrougao(1, 1) {}

JSTrougao(double aa) : JKTrougao(aa, aa) {}

JSTrougao(const JSTrougao &jst) : JKTrougao(jst.a, jst.b) {}
```

Primer 2 - Klase Osoba, Student i PhDStudent

- Napisati klasu **Osoba** koja sadrži polja: ime (**DinString**) i prezime (**DinString**). U klasi implementirati:

  - Osoba()

  - Osoba(const char*, const char*)

  - Osoba(const DinString&, const DinString&)

  - Osoba(const Osoba&)

  - metod predstaviSe() - ispisuje (na konzolu) ime i prezime osobe.

Primer 2 - Klasa DinString

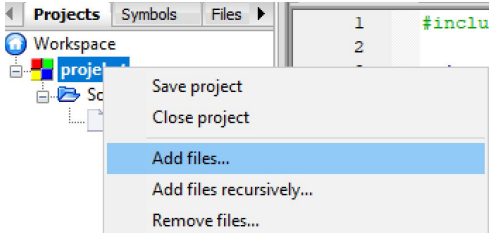
- Klasa **DinString** je klasa koja modeluje dinamički alociran niz karaktera.
- Na vežbama i kolokvijumima je dobijate gotovu i potrebno je naučiti samo kako je koristiti.
- Polja u klasi **DinString** su:

```
private:
    int duzina;
    char *text;
```

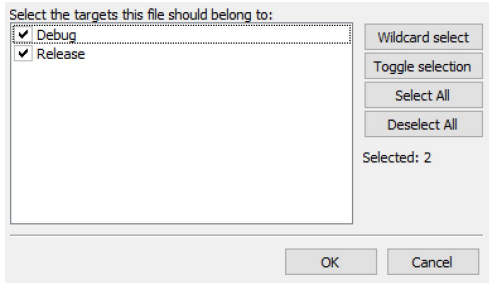
- String je vezan za pokazivač text, a na kraju stringa se nalazi karakter `NULL`, tj. `'\0'`. Podatak-član duzina sadrži broj karaktera u stringu.

Primer 2 - Kako dodati klasu DinString u projekat

- Fajlove dinstring.hpp i dinstring.cpp koje dobijete od nas potrebno je dodati u projekat. To se radi na sledeći način:
  - u fajl sistemu kopirati fajlove u direktorijum projekta u koji se dodaju fajlovi
  - u CodeBlocks-u desnim klikom odabrati projekat u koji se dodaju fajlovi
  - odabrati opciju “Add files”



  - odabrati dinstring.hpp i dinstring.cpp
  - kliknuti “OK” na sledeći dijalog



Primer 2 - Klasa DinString

- Metode i prijateljske funkcije u klasi **DinString** su:

```
public:
    DinString();
    DinString(const char[]);
    DinString(const DinString&);
    ~DinString();

    int length() const;

    char& operator[] (int);
    char operator[] (int) const;

    DinString& operator=(const DinString&);
    DinString& operator+=(const DinString&);

    friend bool operator==(const DinString&, const DinString&);
    friend bool operator!=(const DinString&, const DinString&);

    friend DinString operator+(const DinString&, const DinString&);

    friend ostream& operator<<(ostream&, const DinString&);
```

Primer 2 - Klase Osoba, Student i PhDStudent

- Klasa **Osoba** sadrži ime i prezime koji su objekti klase **DinString**
- Kako ona sadrži objekte drugih klasa kao polja, to znači da je klasa **Osoba** kompozicija.

```
class Osoba
{
    private:
        DinString ime, prezime;

};
```

Primer 2 - Klasa Osoba

- Konstruktor bez parametara i konstruktor sa parametrima su spojeni u konstruktor sa parametrima sa podrazumevanim vrednostima. On pomoću konstruktora inicijalizatora inicijalizuje objekte članove klase **DinString** pozivajući konstruktor sa parametrima iz klase **DinString**

```
Osoba(const char* s1 = "", const char* s2 = "") : ime(s1), prezime(s2) { }

Osoba(const DinString& ds1, const DinString& ds2) : ime(ds1), prezime(ds2) { }
```

- Konstruktor kopije kreira objekat na osnovu reference na objekat klase **Osoba** i pomoću konstruktora inicijalizatora inicijalizuje objekte članove klase **DinString**, pozivajući konstruktor kopije iz **DinString**-a.

```
Osoba(const Osoba& osoba) : ime(osoba.ime), prezime(osoba.prezime) { }
```

Primer 2 - Klasa Osoba

- Metoda predstaviSe() u klasi **Osoba** ispisuje ime i prezime za **Osobu**.

```
void predstaviSe() const
{
    cout << "Zovem se " << ime << " " << prezime << "." << endl;
}
```

- Ne zaboraviti uključiti potrebne fajlove i biblioteke:

```
#include "dinstring.hpp"
#include <iostream>

using namespace std;
```

Primer 2 - Klasa Osoba

- Metoda predstaviSe() u klasi **Osoba** ispisuje ime i prezime za **Osobu**.

```
void predstaviSe() const
{
    cout << "Zovem se " << ime << " " << prezime << "." << endl;
}
```

- Primetiti da u metodi predstaviSe() šaljemo objekte klase **DinString** na ispis (ime i prezime). Ovo je moguće uraditi zato što u klasi **DinString** imamo preklopljen operator ispisa.

Primer 2 - Klasa Osoba

- Preklapanje operatora je mehanizam pomoću koga “učimo” operator kako da se ponaša ukoliko mu operandi budu objekti nekih klasa.
- Primer za ovo je preklapanje operatora za ispis << koje je urađeno u klasi **DinString**. Ono nam omogućava da ispisujemo objekat klase **DinString**. Zbog toga smo mogli da napišemo:
  - `cout<<ime;`
  - `cout<<prezime;`
- Više o preklapanju operatora ispisa biće urađeno u sledećem terminu vežbi, a više o preklapanju ostalih operatora biće urađeno na predavanjima.

Primer 2 - Klase Osoba, Student i PhDStudent

- Napisati klasu **Student** koja nasleđuje klasu **Osoba**. Klasa sadrži dodatno polje brojIndeksa (int). U klasi implementirati:

  - Student()

  - Student(const char*, const char*, int)

  - Student(const DinString&, const DinString&, int)

  - Student(const Osoba&, int)

  - Student(const Student&)

  - Redefinisan metod predstaviSe() - dopisuje i broj indeksa

Primer 2 - Klasa Student

- Klasa **Student** nasleđuje klasu **Osoba**. To znači da ona preuzima sva polja i sve metode klase osoba. Pored toga ona može i da izmeni (redefiniše) neku metodu, ali i da ima novu sopstvenu metodu. Dakle, izvedena klasa može imati sledeće vrste metoda:

    1. Nasleđene
    2. Redefinisane
    3. Nove

Primer 2 - Klasa Student

- Klasa **Student** je preuzela polja ime i prezime od klase **Osoba**, ali njoj treba dodati i novo polje: polje broj indeksa. Zato sada u klasi **Osoba** dodajemo to novo polje. Polja ime i prezime ne pišemo ponovo jer su nasleđena. Polje brojIndeksa pišemo u protected delu ukoliko rešimo da nasledimo klasu Student.

```
#include "osoba.hpp"

class Student : public Osoba
{
    protected:
        int brojIndeksa;
};
```

Primer 2 - Klasa Student

- Klasa **Student** nije nasledila konstruktore, potrebno ih je napisati.
- Prva tri konstruktora su tri različita konstruktora sa parametrima. Koristi se konstruktor inicijalizator kako bi se kreirao roditeljski deo (Osoba) i inicijalizovalo dodatno polje - polje broj indeksa.

```
Student(const char* s1 = "", const char* s2 = "", int i = 0) : Osoba(s1, s2), brojIndeksa(i) { }

Student(const DinString& ds1, const DinString& ds2, int i) : Osoba(ds1, ds2), brojIndeksa(i) { }

Student(const Osoba& os, int i) : Osoba(os), brojIndeksa(i) { }
```

Primer 2 - Klasa Student

- Konstruktor kopije u klasi **Student**:

```
Student(const Student& s) : Osoba((Osoba)s), brojIndeksa(s.brojIndeksa) { }
```

- Koristi se konstruktor inicijalizator kako bi inicijalizovali roditeljski deo i dodatno polje - polje broj indeksa. Roditeljski deo se inicijalizuje pomoću konstruktora kopije iz klase **Osoba** na sledeći način: Osobi se prosleđuje roditeljski deo objekta klase **Student**.

Primer 2 - Klasa Student

- Klasa **Student** je metode od klase **Osoba** nasledila, tj. preuzela je metodu predstaviSe(), ali ovu metodu je potrebno redefinisati kako bi prilikom predstavljanja pored imena i prezimena bio ispisan i broj indeksa.
- Redefinisanje metode vrši se tako što se u izvedenoj klasi piše isti prototip (deklaracija) metode koja se redefiniše kao u baznoj klasi.
- Da bi pozvali metodu nad roditeljskim delom klase osoba, pišemo naziv bazne klase::naziv metode, kao što se vidi u nastavku.

```
void predstaviSe() const
{
    Osoba::predstaviSe();
    cout << "Broj mog indeksa je " << brojIndeksa << "." << endl;
}
```

Primer 2 - Klasa Student

- Napomena: U klasi **Osoba** polja su napisana kao private, ali su mogla biti napisana i kao protected.
- Da su polja bila protected, mogli smo napisati i sledeći kod.

```
void predstaviSe() const
{
    cout << "Zovem se " << ime << " " << prezime << "." << endl;
    cout << "Broj mog indeksa je " << brojIndeksa << "." << endl;
}
```

Primer 2 - Samostalno uraditi

Napisati klasu **PhDStudent** koja nasleđuje klasu Student. Klasa sadrži dodatno polje prosecnaOcena (double). U klasi implementirati:

- PhDStudent()

- PhDStudent (const char*, const char*, int, double)

- PhDStudent (const DinString&, const DinString&, int, double)

- PhDStudent (const Osoba&, int, double)

- PhDStudent (const Student&, double)

- PhDStudent (const PhDStudent&)

- Redefinisan metod predstaviSe() - dopisuje i prosečnu ocenu

Literatura

1. Kupusinac A. : Zbirka rešenih zadataka iz programskog jezika C++. Novi Sad: FTN, 2011.