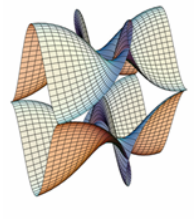

Prirodoslovno-matematički fakultet
Matematički odsjek
Sveučilište u Zagrebu

RAČUNARSKI PRAKTIKUM I

Vježbe 08 - Nasljeđivanje

v2018/2019.

Sastavio: Zvonimir Bujanović

Osnovni principi objektno-orijentiranog programiranja

1. Enkapsulacija - korisnik strukture vidi i može koristiti samo one podatke koji mu zaista i trebaju za rad sa strukturom.

2. Nasljeđivanje - specijalizirane strukture mogu dijeliti zajedničku funkcionalnost s općenitijim strukturama.

3. Polimorfizam - srodni objekti sa zajedničkim sučeljem mogu implementirati to sučelje na različite načine.

Enkapsulacija

Treba li korisnik stoga pristup do varijabli `element` i `size`?

```
struct stack
{
    int element[100], size;

    int top() { ... }
    void push( int x ) { ... }
    void pop() { ... }
    stack() { ... }
};
```

Ne!

- Korisniku stoga dovoljno je samo pozivati funkcije. On ne treba znati detalje implementacije.
- Izmjenom tih varijabli korisnik može “pokvariti” stog tako da on više ne bude funkcionalan.

Enkapsulacija

Rješenje: `element` i `size` su privatni, ostali elementi klase su javni.

```
class stack
{
private:
    int element[100], size;

public:
    int top() { ... }
    void push( int x ) { ... }
    void pop() { ... }
    stack() { ... }
};
```

Enkapsulacija

Sada privatne članove **`element`** i **`size`** smiju koristiti samo funkcije članice! Javne članove smije koristiti bilo tko.

```
int stack::top()
{
    return element[size-1]; // OK
}

int main( void )
{
    stack S; int a;
    ...
    a = S.top(); // OK
    a = S.element[S.size-1] // krivo! compile error
    ...
}
```

Enkapsulacija

`struct` ili `class`?

- Unutar `struct` članove također možemo označiti s `private` i `public`.
- `struct` - po defaultu su svi članovi `public`.
- `class` - po defaultu su svi članovi `private`.
- Ovo je jedina razlika između `struct` i `class`.
- Obično se u objektno-orijentiranom programiranju koristi `class`.
- `struct` tipično koristimo za male strukture koje nemaju funkcije kao članove, nego samo javno dostupne podatke.

I podaci i funkcije mogu biti označeni s `private` ili `public`.

Nasljeđivanje

Promotrimo ovakvu klasu (zasad su svi članovi javni):

```
class Zivotinja
{
public:
    int starost;

    Zivotinja() { starost = 0; }
    Zivotinja( int s ) { starost = s; }

    void glasaj_se() { cout << "zivotinja se glasa"; }
};
```

Nasljeđivanje

Želimo uvesti nove tipove podataka: `Pas` i `Kokos`.
Oba ova tipa također imaju starost i mogu se glasati.

- `Pas` još ima ime.
- `Kokos` još ima broj jaja koje je snijela.

Vrijedi:

- `Pas` je "podvrsta" `Zivotinje`.
- `Kokos` je "podvrsta" `Zivotinje`.

Nasljeđivanje

Jedna moguća implementacija klase `Pas`:

```
class Pas
{
public:
    int starost; string ime;

    Pas( int s, string i ) { starost = s; ime = i; }

    void glasaj_se() { cout << "zivotinja se glasa"; }
};
```

Ovo je loše rješenje:

- Klase `Pas` i `Zivotinja` nisu povezane ni na koji način.
- `Pas` nije podvrsta `Zivotinje`.
- Postoji veliko preklapanje (označeno) s implementacijom klase `Zivotinja`.

Nasljeđivanje

Rješenje: klasa `Pas` je nasljeđena iz klase `Zivotinja`.

```
class Pas : public Zivotinja
{
public:
    string ime;

    Pas( int s, string i ) { starost = s; ime = i; }
};
```

`Pas` je ovako automatski dobio sve članove - i podatke i funkcije - koje ima `Zivotinja`.

Nasljeđivanje

Uz nasljeđivanje, svaki `Pas` je zaista ujedno i `Zivotinja`.

```
Zivotinja z( 5 );
Pas floki( 3, "Floki" );

// Pas je dobio sve funkcije iz klase Zivotinja:
floki.glasaj_se(); //  OK, ispise "zivotinja se glasa"

// floki je Pas, a svaki Pas je Zivotinja:
// floki.starost se prekopira u z.starost
// floki.ime se ignorira.
z = floki; //  OK

// Nije svaka Zivotinja ujedno i Pas:
// Sto bi se prekopiralo u floki.ime?
floki = z; //  Krivo! Compile error!
```

Nasljeđivanje

Terminologija i pravila nasljeđivanja:

- `Zivotinja` je tzv. bazna klasa, a `Pas` je izvedena klasa.

- Ako je član (podatak, funkcija) u `Zivotinja` bio `public`,
  - on ostaje `public` i u klasi `Pas`;
  - dakle, dostupan je i funkcijama članicama u klasi `Pas` i korisnicima objekta tipa `Pas`.

- Ako je član (podatak, funkcija) u `Zivotinja` bio `private`,
  - on nije dostupan članicama u klasi `Pas`, nego samo u `Zivotinja`.

- Ako je član (podatak, funkcija) u `Zivotinja` bio `protected`,
  - on je dostupan funkcijama članicama u `Zivotinja` i `Pas`;
  - on nije dostupan nikome izvan klasa `Zivotinja` i `Pas`;
  - on ostaje `protected` i u klasi `Pas`.

"Privatne" podatke u klasama najčešće označavamo s `protected`.

Nasljeđivanje

```
class Pas : public Zivotinja
{
public:
    string ime;

    Pas( int s, string i ) { starost = s; ime = i; }
};
```

Kod deklaracije

```
Pas floki( 3, "Floki" );
```

1. prvo se poziva (defaultni, zasad) konstruktor za baznu klasu, tj. za `Zivotinja`;
2. zatim se poziva konstruktor za `Pas` s odgovarajućim parametrima.

Nasljeđivanje

Konstruktor klase **Pas** zasad inicijalizira i dio koji se odnosi na **Zivotinju** (starost). To je uloga konstruktora bazne klase!

Konstruktor izvedene klase **Pas** može prvo pozvati konstruktor bazne klase **Zivotinja**:

```
class Pas : public Zivotinja
{
public:
    string ime;

    Pas( int s, string i ) : Zivotinja( s ) { ime = i; }
};
```

Uoči:

- do sada je konstruktor za **Pas** pristupao varijabli **starost**;
- zbog toga je **starost** morala biti ili **public** ili **protected**;
- sada možemo **starost** označiti i sa **private**.

Zadatak 1

- Napravite hijerarhiju klasa: `cetverokut`, `pravokutnik`, `kvadrat`.
- Članovi `cetverokuta` neka budu duljine svih stranica i funkcija za računanje opsega.
- Neka svaka klasa ima konstruktor pomoću kojeg se može zadati duljina stranica lika; neka još svaki konstruktor ispisuje poruku o kojem je liku riječ.
- Neka svaka klasa ima destruktor koji ispisuje o kojem je liku riječ.
- Neka `main` učitava duljinu stranice kvadrata i ispisuje njegov opseg.

Uočite redoslijed pozivanja konstruktora i destruktora!

Nasljeđivanje

`Pas` se zna ponašati kao `Zivotinja` u svakoj situaciji.

```
void nahrani( Zivotinja &z )
{
    z.glasaj_se();
}

Pas floki( 3, "floki" );

Zivotinja  z = floki;  // OK
Zivotinja &r = floki;  // OK
Zivotinja *p = &floki; // OK

nahrani( floki ); // OK
```

Polimorfizam

Međutim, `Pas` se glasa drugačije nego ostale životinje.
Zato u toj klasi ponovno implementiramo funkciju `glasaj_se`.

```
class Pas : public Zivotinja
{
    ...
    void glasaj_se() { cout << ime << ": vau-vau!"; }
};

Pas floki( 3, "floki" );

// ispis: "floki: vau-vau!"
floki.glasaj_se();
```

Polimorfizam

No specijalizirana funkcija se sada ne prepoznaje u svim situacijama!

```
Pas floki( 3, "floki" );
Zivotinja  z = floki;
Zivotinja &r = floki;
Zivotinja *p = &floki;

// ispis: "zivotinja se glasa"
z.glasaj_se();

// ispis: "zivotinja se glasa"
r.glasaj_se();

// ispis: "zivotinja se glasa"
p->glasaj_se();

// ispis: "zivotinja se glasa"
nahrani( floki );
```

Polimorfizam

Rješenje:

- Ako želimo dozvoliti da izvedena klasa napravi svoju implementaciju neke funkcije članice, onda ćemo tu funkciju u baznoj klasi označiti kao virtualnu.

```
class Zivotinja
{
protected:
    int starost;

public:
    Zivotinja( int s ) { starost = s; }

    virtual void glasaj_se()
    {
        cout << "zivotinja se glasa";
    }
}
```

Polimorfizam

Uz virtualnu funkciju u baznoj klasi, reference i pokazivači na objekt izvedene klase sada pozivaju funkciju iz izvedene klase!

```
Pas floki( 3, "floki" );
Zivotinja  z = floki;
Zivotinja &r = floki;
Zivotinja *p = &floki;

// ispis: "zivotinja se glasa"
// logicno, varijabla z vise nema nikakvu vezu s flokijem.
z.glasaj_se();

// ispis: "floki: vau-vau"
r.glasaj_se();

// ispis: "floki: vau-vau"
p->glasaj_se();

// ispis: "floki: vau-vau"
nahrani( floki );
```

Zadatak 2

1. Nadopunite hijerarhiju iz Zadatka 1 klasama `paralelogram` (koja ima dodatni član `kut`) i `romb`; nemojte stavljati da je `pravokutnik` vrsta `paralelograma`.

2. Nadopunite sve klase funkcijom `povrsina` (za četverokut neka vraća `-1`).

3. Napravite polje koje se sastoji od 10 raznovrsnih četverokuta i ispisuje površine svakog od njih.

Uoči:

- polje se mora sastojati od pokazivača na dinamički alocirane objekte, sa `new`!
- destruktor u baznoj klasi mora biti označen kao virtualan!

Što ako elementi polja nisu pokazivači nego "obični" četverokuti? Što ako destruktor u baznoj klasi nije virtualan? Probajte i objasnite.

Apstraktne klase

- Uočite da zapravo nikad nećemo deklarirati objekt tipa `Zivotinja`. Možemo imati objekte tipa `Pas`, `Macka`, `Kokos`, ali `Zivotinja` je apstraktna, preopćenita.

- Kod takvih baznih klasa redovito postoje funkcije za koje znamo da ih svaka izvedena klasa mora imati, ali su preopćenite i ne znamo ih implementirati kod bazne klase.

- Na primjer, svaka `Zivotinja` se može glasati, ali ne znamo točno kako. S druge strane, `Pas` laje, `Macka` mijauče, `Kokos` kokodače, itd.

Apstraktne klase

Takve “preopćenite” funkcije u baznoj klasi zovemo čiste (pure) virtualne funkcije. Bazne klase koje imaju bar jednu čistu virtualnu funkciju zovemo apstraktnim.

```
class Zivotinja {
    ...
    virtual glasaj_se() = 0;
};
```

Čiste virtualne funkcije nikad nemaju implementaciju.
Zabranjeno je napraviti objekt apstraktne klase.

```
// compile error! Zivotinja je apstraktna klasa.
Zivotinja z( 5 );

Pas floki( 3, "floki" ); // OK, Pas je implementirao glasaj_se().

// OK, r je referenca na Psa, a ne na Zivotinju!
Zivotinja &r = floki; r.glasaj_se();
```

Apstraktne klase

Ako izvedena klasa ne napravi implementaciju čiste virtualne funkcije, onda je i ta klasa apstraktna!

- Na primjer, `KucniLjubimac` bi isto mogla biti apstraktna klasa ako u njoj ne implementiramo funkciju `glasaj_se()`.
- U hijerarhiji `Zivotinja` $\rightarrow$ `KucniLjubimac` $\rightarrow$ `Pas` bi tada prve dvije klase bile apstraktne.

Zadatak 3

1. U hijerarhiju iz Zadatka 2 dodajte klase `Trokut` i `Krug`.

2. Dodajte i polaznu baznu klasu `Lik` iz koje su izvedene sve ostale.

3. Izmijenite glavni program iz Zadatka 2 tako da ima vektor raznih likova kojima računa površinu; omogućite korisniku da odabere broj i vrstu likova, te njihove parametre.