Določanje dostopa do razrednih komponent

1

- za vsako razredno spremenljivko oz. metodo (z eno besedo člani) je določen način dostopa do le-te
- način dostopa določa, od kod so ti razredni člani dosegljivi
- poznamo naslednje načine dostopa
  - public člani, ki so javni, so dostopni od povsod
  - private člani, ki so zasebni, so dostopni zgolj znotraj samega razreda
  - protected člani, ki so zaščiteni, so dostopni znotraj samega razreda in v izpeljanih razredih
- privzeto določilo za razrede je **private**

Določanje dostopa do razrednih komponent

2

- običajno so razredne spremenljivke zasebne, metode pa javne
- mehanizem, ki omogoča spreminjanje dostopa do razrednih komponent, imenujemo **skrivanje podatkov**
- ZAKAJ TA MEHANIZEM POTREBUJEMO?
  - s skrivanjem podatkov zagotovimo integriteto razreda
    - uporabnik lahko dostopa do razrednih spremenljivk preko metod
    - v metodah imamo nadzor nad veljavnimi vrednostmi
    - v vsakem trenutku objekt predstavlja veljavni primerek razreda
  - uporabnik vidi le elemente, ki so zanj uporabni

Določanje dostopa do razrednih komponent

3

- dostop določimo na naslednjih način:

```
class Ime {
  public:
    // spremenljivke in metode
  private:
    // spremenljivke in metode
  protected:
    // spremenljivke in metode
};
```

- vrstni red in število določil nista pomembna
- določilo velja do naslednje spremembe ali konca razreda

Določanje dostopa do razrednih komponent primer CLok – kot smo ga napisali do sedaj

4

```
class CLok {
  public:
    string barva;
    int stPuscic;
    bool jeNapet;

    bool napni();
    int sprozi();
};
```

```
#include "Lok.h"
...
int main() {
  ...
  CLok mojLok;
  mojLok.barva="bela";
  mojLok.stPuscic = 5;
  ...
}
```

Določanje dostopa do razrednih komponent primer CLok – z zasebnimi spremenljivkami

5

```
class CLok {
  private:
    string barva;
    int stPuscic;
    bool jeNapet;
  public:
    bool napni();
    int sprozi();
};
```

```
#include "Lok.h"
...
int main() {
  ...
  CLok mojLok;
  mojLok.barva="bela";
  mojLok.stPuscic = 5;
  ...
}
```

NAPAKA! spremenljivki sta zasebni, torej kot uporabnik, nimamo direktnega dostopa

Dostop do zasebnih spremenljivk preko metod

6

- dostop do zasebnih spremenljivk lahko uporabniku razreda omogočimo preko javnih metod
- v metodi imamo nadzor nad morebitnimi spremembami
- to so običajno vrni/spremeni (angl. get/set) metode
- implementiramo tiste, ki jih želimo omogočiti
  - včasih uporabnik ne sme direktno spremeniti vrednosti, lahko pa pogleda, koliko tista vrednost je
  - v takem primeru implementiramo samo vrni() metodo

primer CLok – dostop do zasebnih spremenljivk

7

```
class CLok {
  private:
    string barva;
    int stPuscic;
    bool jeNapet;
  public:
    bool napni();
    int sprozi();
    int vrniStPuscic();
    void spremeniStPuscic(int n);
};
```

NALOGA: na podoben način implementirajte še vrni/spremeni funkciji za barvo loka in stanje jeNapet.

primer CLok – dostop do zasebnih spremenljivk

8

```
int CLok::vrniStPuscic() {
     return stPuscic;
}

void CLok::spremeniStPuscic(int n) {
     if (n>=0)
          stPuscic = n;
}
```

Funkcija spremeni št. puščic loka, spremembo dovolimo le, če je predlagano število puščic smiselno. Imamo nadzor!

primer CLok – dostop do zasebnih spremenljivk

9

```
#include "Lok.h"
...
int main() {
  ...
  CLok mojLok;

  // mojLok.stPuscic = 5;
  mojLok.spremeniStPuscic(5);
  ...
}
```

NAPAKA! spremenljivka je zasebni, torej kot uporabnik, nimamo direktnega dostopa!

Je pa na voljo metoda spremeniStPuscic(), ki je **javna** in jo uporabimo ravno v ta namen.

Kaj izpiše naslednji program?

10

```
#include "Lok.h"

int main() {
  CLok mojLok;

  cout << mojLok.vrniStPuscic();

  return 0;
}
```

Ne moremo odgovoriti, saj nismo nikjer določili začetnih vrednosti razrednih spremenljivk!

Konstruktorji in destruktor

11

- vsak razred lahko ima še definirane posebne funkcije, ki jih imenujemo konstruktorji ali destruktor
- konstruktor se izvede samodejno, ko ustvarimo objekt danega tipa
- destruktor se izvede samodejno, ko se objekt briše iz pomnilnika
- s konstruktorji poskrbimo, da imajo razredne spremenljivke določene vrednost takoj, ko se ustvari objekt
- na ta način objekt že na začetku predstavlja veljaven primerek razreda

Konstruktorji

12

- konstruktor definiramo kot običajno funkcijo, z naslednjimi zahtevami:
  - je vedno **javna** funkcija
  - konstruktor **nima tipa**, niti void!
  - ime konstruktorja **mora** biti enako imenu razreda
  - parametri – ni omejitev, torej kot običajne funkcije
    - torej lahko imamo več konstruktorjev z različnimi parametri
    - to **prekrivanje funkcij** v splošnem velja za vse funkcije v C++
- pri ustvarjanju objekta s parametri povemo, kateri konstruktor želimo izvesti

Vrste konstruktorjev

13

- poznamo naslednje vrste konstruktorjev
  - **privzeti konstruktor**
    - brez parametrov
    - v njem običajno **vsem** razrednim spremenljivkam določimo začetne vrednosti
  - **kopirni konstruktor**
    - za parameter prejme referenco na objekt istega tipa
    - eksplicitno povemo, da se objekt v parametru ne spremeni
    - v tem konstruktorju vsem razrednim spremenljivkam določimo enako vsebino, kot jo imajo spremenljivke v objektu v parametru
  - **inicializacijski konstruktor**
    - za vsako razredno spremenljivko podamo ustrezno vrednost preko parametrov
  - **ostali konstruktorji**
    - imajo drugačne parametre in nimajo posebnih imen

primer CLok (lok.h)

14

```
class CLok {
  private:
    // spremenljivke kot prej
  public:
    CLok();
    CLok(int p, bool n, string b);
    CLok(const CLok &lok);
    // ostale funkcije kot prej
    // vključno s tistimi iz naloge
};
```

primer CLok (lok.cpp)

15

```
CLok::CLok() {
  stPuscic = 20;
  jeNapet = false;
  barva = "bela";
}

CLok::CLok(int p, bool n, string b){
  stPuscic = p;
  jeNapet = n;
  barva = b;
}
```

primer CLok - nadaljevanje (lok.cpp)

16

```
CLok::CLok(const CLok &aL){
  stPuscic = aL.vrniStPuscic();
  jeNapet = aL.vrniJeNapet();
  barva = aL.vrniBarvo();
}

// ostale funkcije kot prej
```

primer CLok (main.cpp)

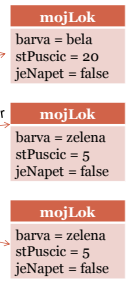
17

```
#include #lok.h"

int main() {

  CLok mojLok,
       tvojLok(5, false, "zelena"),
       njenLok(tvojLok);

  return 0;
}
```



Destruktor

18

- destruktor definiramo kot običajno funkcijo, z naslednjimi zahtevami:
  - je vedno **javna** funkcija
  - konstruktor **nima tipa**, niti void!
  - ime destruktorja **mora** biti enako imenu razreda, pred katerim pišemo ~ (tilda)
  - **nima** parametrov
    - torej lahko imamo **samo en** destruktor
- destruktor se izvede **samodejno**, ko se objekt briše iz pomnilnika
  - to je npr. ob koncu programa, ob koncu funkcije za lokalne objekte, ko eksplicitno brišemo z **delete**
  - v destruktorju poskrbimo za to, da sprostimo pomnilnik, ki smo ga sami rezervirali

primer CLok (lok.h)

19

```
class CLok {
  private:
    // spremenljvike kot prej
  public:
    ~CLok();
    // ostale funkcije kot prej
    // vključno s tistimi iz naloge
    // in konstruktorji
};
```

primer CLok (lok.cpp)

20

```
CLok::~CLok() {
  // tukaj sproščamo pomnilnik
  // v tem primeru ne storimo ničesar
}
```

Vsakemu razredu zapišemo konstruktor in destruktor, četudi sta prazna.

-- seveda za konstruktor ni smiselno, da je prazen --