Sa proceduralnog na OO programiranje: klase i objekti

- Na ovaj način postiže se potpuna efikasnost — kod generisan za funkcije članice i njihov poziv, kao i pristup članovima objekata podjednako je efikasan kao i ekvivalentan C kod, jer nema nikakve dodatne režije (*overhead*)
- Sa druge strane, postiže se višestruka dobit u pisanju i strukturiranju programa, kao i u podršci prevodioca koji vrši odgovarajuće automatske provere (samo u vreme prevođenja), umesto da sve to zavisi samo od discipline i pedantnosti programera:
  - Koncept klase kao apstrakcije se jasno ističe i podstiče programera da je na ispravan način definiše, odnosno grupiše strukturu i ponašanje (operacije nad tom strukturom)
  - Struktura i operacije nad njom su "upakovani" u jedinstvenu, jasno istaknutu logičku celinu, i pripadaju *oblasti važenja klase* (*class scope*)
  - Podržana je enkapsulacija, jer prevodilac kontroliše i ne dozvoljava neovlašćen pristup do implementacije klase (privatnih članova)
  - Postoji koncept konstruktora (i destruktora), pa je inicijalizacija objekata lakša, očiglednija i manje podložna greškama

Sa proceduralnog na OO programiranje: klase i objekti

- Pokazivač *this* se može koristiti kao i svaki drugi pokazivač unutar funkcije članice, jer on prosto ukazuje na objekat (čija je funkcija pozvana), na primer, ako je drugom objektu potreba veza ka tom objektu
- Na primer, hoćemo da objekti klase *Clock* imaju vezu ka objektu klase *Lobby* koji ih sadrži:

```
class Clock {
public:
  Clock (Lobby* owner, int hh, int mm, int ss);
  …
private:
  Lobby* myOwner; // The Lobby that contains this Clock
  …
};

Clock::Clock (Lobby* owner, int hh, int mm, int ss) {
  this->myOwner = owner;
}

Lobby::Lobby (unsigned n, string ct[], int lg[]) {
  …
  for (int i=0; i<num; i++) {
    …
    this->clocks[i] = new Clock(this,h,0,0);
  }
}
```