Enkapsulacija

- U mnogim slučajevima postoji potreba za čuvanje informacija (podataka) koji nisu svojstva svakog pojedinačnog objekta, već cele klase, odnosno zajednički su za sve objekte te klase
- Na proceduralnom jeziku, poput jezika C, ovakve podatke moramo definisati kao globalno dostupne (po oblasti važenja), što narušava enkapsulaciju, jer su oni onda dostupni svim delovima programa
- U OOP i na jeziku C++, kao i na mnogim drugim jezicima, na raspolaganju su *statički podaci članovi* (*static data members*): postoji samo po jedna instanca za svaki definisan statički podatak član, on je jedna instanca, deljena između svih objekata te klase
- Na primer, želimo da brojimo koliko je objekata kalse *Clock* ukupno kreirano - to je informacija bitna za celu klasu:

```
class Clock {
public:
  Clock ();
  …

private:
  static int count;
  …
};

void Clock::Clock () {
  …
  count++;
}

int Clock::count = 0;
```

Statički podatak član

Pristup statičkom podatku članu ne zahteva objekat kome pripada

Statički podatak član mora se definisati i inicijalizovati

- Pristup statičkom podatku članu ne zahteva objekat (kao levi operand operatora .) ili pokazivač na objekat (kao levi operand operatora ->), jer on pripada klasi, a ne pojedinačnom objektu; taj objekat/pokazivač se ipak može i zadati

Enkapsulacija

- Ako je statički podatak član instanca neke klase, njegova inicijalizacija zahteva poziv odgovarajućeg konstruktora, a kod za taj poziv se izvršava u vreme izvršavanja programa i prevodilac treba negde da ga generiše; jedino što se od prevodioca zahteva i garantuje jeste to da se ta inicijalizacija sigurno vrši pre bilo kog pristupa tom objektu ili poziva funkcije članice te klase koji se nalazi u istom fajlu u kom je taj statički podatak član definisan; ovo ne mora biti *pre* početka izvršavanja funkcije *main*
- Zbog toga je korišćenje statičkih podataka članova kao instanci klasa nepouzdano, jer ne moraju obavezno biti propisno inicijalizovani pre svakog korišćenja; zato je umesto njih bolje koristiti lokalne statičke objekte (detalji kasnije)
- Statički podaci članovi klase imaju isti životni vek i skladište se na isti način u memoriji kao i globalni statički objekti, ali je korišćenje statičkih podataka članova bolje u mnogim slučajevima, jer je statički podatak član:
  - deo klase kao logičke celine, logički je "upakovan" u nju, pa je jasnija njegova upotreba i namena - program je čitljiviji i lakši za razumevanje
  - u oblasti važenja klase, a nije globalan, pa se ne sukobljava po imenu (*name clashing*) sa ostalim gobalnim imenima (može da se zove isto)
  - član klase, kao i svaki drugi, pa se može (i po pravilu treba) enkapsulirati: on može da bude zaštićen ili privatan, pa tako i nedostupan ostalim delovima programa (osim izvedenim klasama, ako je zaštićen)
- Zbog toga, neki noviji jezici (npr. Java) i ne omogućavaju globalne objekte (tačnije reference na njih), a umesto njih podržavaju statičke podatke članove: svaka referenca na objekat mora biti članica neke klase (statička ili nestatička)