Statički životni vek

- Zbog svega ovoga, umesto statičkih objekata koji nisu lokalni (npr. globalni ili podaci članovi), bolje je koristiti lokalne statičke objekte, odnosno statičke objekte "umotati" u funkciju (po pravilu nečlanicu ili statičku članicu). Tako se garantuje propisna inicijalizacija, ali i bolja enkapsulacija. Na primer:

```
class Clock {
public:
  Clock (…) { getClockRegister()->add(this); }
  static const ClockRegister* getClocks () { return getClockRegister(); }
private:
  static ClockRegister* getClockRegister ();
};

ClockRegister* Clock::getClockRegister () {
  static ClockRegister clockRegister(…);
  return &clockRegister;
}
```

Lokalni statički objekat. Enkapsuliran je u ovu funkciju, pa mu se može pristupiti samo preko nje. Pri prvom pozivu te funkcije (a drugačije mu se i ne može pristupiti), on će biti inicijalizovan pozivom konstruktora, koji može da uradi bilo kakvu dinamičku inicijalizaciju.

- Svi statički objekti žive do kraja programa i uništavaju se nakon završetka funkcije *main*. Ako neki statički objekat (npr. lokalni) nije inicijalizovan, neće biti ni uništen (neće biti pozvan njegov destruktor)

Copyright 2018-2024 by Dragan Milićev

Dinamički životni vek

- Dinamički životni vek objekata neposredno se kontroliše logikom i dinamikom programa: dinamički objekti se prave i uništavaju eksplicitno:
  - svako izvršavanje izraza (operatora) *new* pravi nov dinamički objekat
  - tako napravljen dinamički objekat živi dok se ne uništi operatorom *delete*

```
Clock* pc = new Clock(9,0,0);
…
delete pc;
```

- Životni vek dinamičkih objekata nije implicitan i vezan za neki opseg, kao što je to slučaj sa automatskim i statičkim objektima; on se kontroliše eksplicitno, pa dinamički objekti mogu da nadžive izvršavanje funkcije u kojoj su napravljeni, što tipično i jeste slučaj: oni se prave u jednom scenariju, izvršavanjem jedne funkcije, a uništavaju možda u nekom poptuno drugom scenariju i u drugoj funkciji:

```
template <typename T>
List<T>& List<T>::addBack (T t) {
  ListElem<T>* e = new ListElem<T>(t, tail, nullptr);
  if (!head) head = e;
  tail = e;
  return *this;
}

template <typename T>
T List<T>::removeFront () {
  ListElem<T>* e = head;
  …
  delete e;
  …
}
```


Copyright 2018-2024 by Dragan Milićev