Klasa kao realizacija softverske mašine

```
class DocumentSaver {
public:
  …
  DocumentSaver (Document*);
 ~DocumentSaver ();
  …
private:

  static void run (DocumentSaver*);

  void save (); // Performs document saving
  Document* myDocument;
  thread* myThread;
  …
};

void DocumentSaver::run (DocumentSaver* ds) {
  if (ds) ds->save();
  delete ds;
}

DocumentSaver::DocumentSaver (Document* d) {
  this->myDocument = d;
  if (d) myThread = new thread(run, this);
}

DocumentSaver::~DocumentSaver () {
  delete myThread;
}
```

Destruktor (destructor)

- Kada treba pokrenuti akciju snimanja dokumenta, samo treba kreirati jedan objekat klase *Document Saver*:

```
new DocumentSaver(theDocument);
```

- Sada su objekti klase *DocumentSaver aktivni*: svaki objekat ove klase ima jednu nit u kojoj se izvršava operacije *save*
- Kada se ova operacija završi, objekat ove klase se uništava
- Ova nit, pa time i operacija *save*, izvršava se uporedo sa svim drugim nitima u programu
- Ova nit, odnosno operacija *save* koju ona izvršava, može da pristupa *deljenim podacima*: pristupa strukturi podataka koja predstavlja dokument (*Document*), kako bi ga snimila u fajl
- Uporedo sa tim, ostale niti rade svoj posao: na primer, "glavna" nit može da obrađuje akcije korisnika, pa korisnik ne mora više da čeka na završetak operacije *save*
- Operacija *save* ne mora da se deli na korake, već se programira kao jedinstvena, sekvencijalna operacija, što olakšava programiranje

Bibliotečna klasa *thread*. Objekat ove klase predstavlja jednu nit koja funkciju zadatu kao argument konstruktora izvršava uporedo sa drugim nitima u programu. Toj funkciji se kao argumenti dostavljaju ostali argumenti konstruktora.

Glava 5: Objektna dekompoziija

- Karakteristike lošeg i dobrog softvera
- Raspodela odgovornosti
- Algoritamska dekompozicija
- Relacije i zavisnosti između klasa
- Enkapsulacija
- Hijerarhijska dekompozicija