Relacije i zavisnosti između klasa

- Sa druge strane, veze mogu da nadžive izvršavanje operacije, pa se moraju skladištiti unutar objekta:

```
class Controller {
  …
private:
  Reader* myReader;
  Translator* myReader;
  StackMachine* mySMs[…];
};
```

- Ukoliko je gornja granica multiplikativnosti odgovarajućeg kraja asocijacije veća od 1, za implementaciju se mora koristiti neka kolekcija:

```
class Teacher {
public:
  …
  void addCourse(Course*);
  void removeCourse(Course*);
private:
  list<Course*> myCourses;
  …
};
```

Operacije koje uspostavljaju i raskidaju veze između objekata klasa *Teacher* i *Course*

Relacije i zavisnosti između klasa

- Umesto da objekat-klijent sam kreira potrebne objekte-servere sa kojima sarađuje, ili sam traži pristup do njih i identifikuje ih (npr. pozivom nekih usluga drugih klasa), praksa pokazuje da je bolji pristup da neko drugi, i to onaj ko koristi posmatranu klasu, spolja, *injektira* te veze, odnosno definiše zavisnosti od servera
- Ovo se može uraditi na sledeće načine:
  - kroz konstruktor posmatrane klase, ali samo ako su te veze obavezne (minimalna multiplikativnost je veća od 0, odnosno objekat *mora* biti vezan):

```
class Controller {
public:

  Controller (Reader* reader, Translator* translator);
  …

};

Controller::Controller (Reader* reader, Translator* translator) {
  this->myReader = reader; this->myTranslator = translator;
}
```

  - kroz posebnu operaciju klase kojom se ova veza uspostavlja, ako se takva veza može menjati tokom života objekta:

```
class Controller {
public:

  void setReader (Reader* reader);
  void setTranslator (Translator* translator);
  …

};

void Controller::setReader (Reader* reader) {
  this->myReader = reader;
}
```

  - ili oba, ako važi i jedno i drugo:

```
Controller::Controller (Reader* reader, Translator* translator) {
  this->setReader(reader); this->setTranslator(translator);
}
```