

Algoritamska dekompozicija

```
enum OpCode { add, sub, ... };

class Command {
public:
    Command (int out, string cmd);

    OpCode getOpCode ();
    int     getOut ();
    ...

private:
    string in;
    OpCode code;
    int out;
};

class Controller {
public:
    void main ();
    ...

private:
    Reader* myReader;
    Translator* myReader;
    StackMachine* mySMs[...];
};

void Controller::main () {
    Command* cmd = myReader->read();
    while (cmd!=nullptr) {
        myTranslator->translate(cmd);
        int out = cmd->getOut();
        mySMs[out]->perform(cmd);
    }
}
```

```
class Reader {
public:
    Command* read ();
    ...
};

class Translator {
public:
    void translate (Command*);
    ...
};

class StackMachine {
public:
    void perform (Command*);
    ...

protected:
    void push (int);
    int pop ();
    ...

private:
    int stack[MaxStackSize];
    unsigned sp;
    ...
};

void StackMachine::perform (Command* cmd) {
    switch (cmd->getOpCode()) {

        case add: {
            int op1 = this->pop();
            int op2 = this->pop();
            this->push(op1+op2);
            break;
        }

        case sub: ...
        ...
    }
}
```

Algoritamska dekompozicija

- ❖ Niži nivo granularnosti primene ovakve dekompozicije jeste u implementaciji pojedinačnih operacija neke klase. Kada se takva operacija dekomponuje na korake (potprograme), ti koraci mogu biti delegirani drugim učesnicima (klasama) u scenariju, ali mogu biti i dalje u opsegu odgovornosti iste klase
- ❖ Ove poslednje pomenute operacije nazivaju se *pomoćne (helper)*, jer se koriste u implementaciji ostalih operacija iste klase
- ❖ Neki saveti za to kada dekomponovati metodu (implementaciju operacije):
 - Kada implementacija ima previše koda, postaje teška za praćenje i razumevanje; po pravilu, metoda sa preko 15-20 linija koda predstavlja signal da treba razmisliti o dekomponovanju na potprograme - krupnije potkorake
 - Kada postoje dublje ugneždene uslovne strukture (*if-then-else*) ili petlje: po pravilu, nije dobro imati više od dva nivoa ugneždivanja, jer to otežava čitanje i razumevanje; tada treba razmisliti o izdvajanju posla koji rade ugneždene strukture u potprograme; to onda olakšava i samo programiranje
 - Kada se delovi implementacije jedne operacije koriste i u drugim - veoma jak razlog za izdvajanje u potprogram, radi izbegavanja dupliranja koda; *copy-paste* je veoma korisna, ali može biti i veoma loša praksa u programiranju; svako kopiranje koda treba da bude signal za razmišljanje da li je propuštena neka apstrakcija (uključujući i algoritamsku dekompoziciju - izdvajanje u potprogram)
- ❖ Ovakve pomoćne operacije najčešće nisu deo interfejsa klase, već deo njene implementacije, pa *ne treba* da budu javne (*public*), već privatne (*private*) ili češće *zaštićene (protected)*, kako bi bile dostupne izvedenim klasama; ovo stoga što ako su već korišćene za implementaciju drugih operacija u osnovnoj klasi, velika je šansa da budu korisne i potrebne u implementaciji (npr. redefinisanih) operacija u izvedenim klasama
- ❖ Često su ovakvi koraci - potprogrami u implementaciji operacija osnovne klase *polimorfni*, tj. imaju svoje podrazumevano ponašanje, ali se to ponašanje može redefinisati u izvedenim klasama; ovakve operacije nazivaju se *kukice (hook methods)*, jer se na njih može “okačiti” drugačija implementacija i time promeniti ponašanje operacije osnovne klase u nekom delu