Objektno orijentisano programiranje

Priprema za prvi kolokvijum

Sta sve ide na prvi kolokvijum?

- Gradivo sa prvih 3 vežbi: Klase, kompozicija
- Zadatak obično ima i dijagram stanja
- Znanje iz C-a se podrazumeva

Zadatak za pripremu

- Napisati klase: **Screen** i **Monitor**

(Tekst zadatka se nalazi na repozitorijumu)

Rešenje: Klasa Screen

- Da bismo implementirali klasu **Screen**, prvo moramo implementirati nabrojivi tip ScreenStates, koji ima dozvoljene vrednosti ssON i ssOFF
- Definiciju nabrojivog tipa pisemo u istom fajlu kao i samu klasu screen tj. u *screen.hpp*
- Kao što je rečeno, klasa **Screen** ima polja *state* (nabrojivog tipa) koje reprezentuje stanje ekrana i polje *brightness* (int) koje reprezentuje osvetljenost ekrana
- Takođe u istom fajlu su navedene i konstante za maksimalni i minimalnu vrednost osvetljenja (*brightness*), kao i korak za koji se osvetljenje menja

```
#define BRIGHTNESS_STEP 2
#define BRIGHTNESS_MAX 20
#define BRIGHTNESS_MIN 0

enum ScreenStates { ssON, ssOFF};

class Screen{
private:
    ScreenStates state;
    int brightness;
```

Rešenje: Klasa Screen - konsturktor i geteri

- Dalje je potrebno napisati konstruktore kao i getere za sva polja
- Konstruktor bez parametara postavlja vrednosti za polja na vrednosti iz teksta zadatka
- **Podsećanje**: geteri imaju povratnu vrednost istu kao polje čiju vrednost dobavljaju
- Pored getera imamo i seter za polje state
- **Podsećanje**: seteri najčešće imaju povratnu vrednost void, dok kao parametar primaju vrednost na koju žele da postave polje
- Tip parametra je uvek isti kao tip polja koje seter postavlja

```
public:
    Screen(){
        state = ssOFF;
        brightness = 0;
    }

    void setState(ScreenStates ss){
        state = ss;
    }

    ScreenStates getState() const {
        return state;
    }

    int getBrightness() const {
        return brightness;
    }
```

Rešenje: Klasa Screen - metode

- Metode incB() i decB() imaju bool kao povratni tip kako bi vratile informaciju da li je osvetljenost ekrana uspešno povećana/smanjena ili ne
- Osvetljenost (*brightness*) je moguće menjati samo u slučaju da je stanje ekrana u ssON, što se proverava u if naredbi
- Osvetljenost takodje nije moguće povećati preko maksimalne ili smanjiti ispod minimalne vrednosti, što se takođe proverava u if naredbi

```
bool incB(){
    if (state == ssON && brightness + BRIGHTNESS_STEP <= BRIGHTNESS_MAX){

        brightness += BRIGHTNESS_STEP;
        return true;
    }
    return false;
}

bool decB(){
    if (state == ssON && brightness - BRIGHTNESS_STEP >= BRIGHTNESS_MIN){
        brightness -= BRIGHTNESS_STEP;
        return true;
    }
    return false;
}
```

Rešenje: Klasa Monitor

- Da bismo implementirali klasu Monitor , prvo moramo implementirati nabrojivi tip MonitorStates, koji ima dozvoljene vrednosti sON, sOFF, sOUT, sTEST i sSTANDBY
- Slično kao i kod Screen klase, nabrojivi tip (enum) pišemo u istom fajlu kao klasu Monitor, tj. u *monitor.hpp*
- Kao što piše i u tekstu zadatka, klasa **Monitor** ima polja *state* (nabrojivog tipa) koje reprezentuje trenutno stanje monitora i polje *screen* (objekat tipa klase **Screen**) koje reprezentuje komponentu ekran koji se nalazi unutar monitora
- Kako unutar klase **Monitor** imamo komponentu tipa **Screen** potrebno je uključiti odgovarajući fajl u kom je napisana klasa **Screen** u okviru *monitor.hpp* fajla - ovo se postiže *include* direktivom

```
#include "screen.hpp"

enum MonitorStates{ sON, sOFF, sOUT, sTEST, sSTANDBY};

class Monitor{
private:
    MonitorStates state;
    Screen screen;
```

Rešenje: Klasa Monitor - konstruktor

- Sledeći korak je implementacija konstruktora
- Polje *state* se postavlja na podrazumevanu vrednost iz teksta zadatka - sOFF
- Polje *screen* se inicijalizuje u delu konstruktora izmedju : i { - takozvani konstruktor inicijalizator, gde se u ovom slučaju poziva konstruktor bez parametara za klasu Screen, koji inicijalizuje polje *screen*
- Treba primetiti da se konstruktor za klasu **Screen** u ovom slučaju ne poziva na isti način kao sto bi se pozivao u main metodi, već se koristi naziv polja koje treba inicijalizovati (u ovom slučaju pišemo screen() umesto Screen s;)

```
public:
    Monitor() : screen() {
        state = sOFF;
    }

    MonitorStates getMonitorState() const {
        return state;
    }

    ScreenStates getScreenStates() const {
        return screen.getState();
    }

    int getBrightness() const {
        return screen.getBrightness();
    }
```

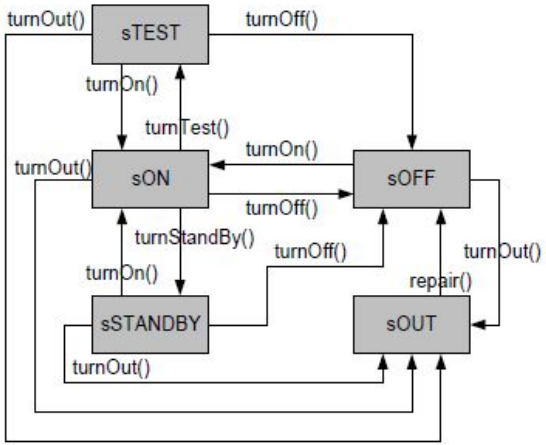
Rešenje: Klasa Monitor - geteri

- Pored getera za polje state iz klase Monitor implementirane su i metode za dobavljanje vrednosti polja *brightness* i *state* iz klase **Screen**
- Kako ove vrednosti nisu u klasi **Monitor** direktno, već se nalaze u okviru komponente *screen*, njima je potrebno pristupiti preko odgovarajućih getera

```
public:
    Monitor() : screen() {
        state = sOFF;
    }

    MonitorStates getMonitorState() const {
        return state;
    }

    ScreenStates getScreenStates() const {
        return screen.getState();
    }

    int getBrightness() const {
        return screen.getBrightness();
    }
```

Rešenje: Klasa Monitor - metode

- Za svaku metodu gledamo na dijagramu za koja stanja je operacija dozvoljena da se izvrši
- U slučaju da je Monitor u stanju koje dozvoljava pomenutu operaciju, metode vraćaju *true* i izvršavaju odgovarajuću operaciju, inače vraćaju *false*
- Stanje Monitora proveravamo if naredbom



Rešenje: Klasa Monitor - metode

- Metoda turnOn() izvršava uspešno operacuju uključivanja monitora u stanjima: sSTANDBY, sTEST i sOFF
- Prilikom prebacivanja monitora u uključeno stanje (sON), potrebno je i ekran prebaciti u uključeno stanje (ssON), što se postiže pomoću set metode - setState(ssON)
- Slična logika je i za metodu turnOff(), s tim da su stanja u kojima se operacija izvršava drugačija - sTEST, sON i sSTANDBY
- Obe metode vraćaju false u slučaju da se pozovu u stanju pri kom se operacija ne može izvršiti

```
bool turnOn(){
    if (state == sSTANDBY || state == sTEST || state == sOFF){
        state = sON;
        screen.setState(ssON);
        return true;
    }
    return false;
}

bool turnOff(){
    if (state == sTEST || state == sON || state == sSTANDBY){
        state = sOFF;
        screen.setState(ssOFF);
        return true;
    }
    return false;
}
```

Rešenje: Klasa Monitor - metode

- Metoda turnOut() uspešno izvršava operaciju samo ukoliko je monitor u stanju sOUT, dok metoda turnTest() to čini u stanju sON
- Obe metode postavljaju i polje *screen* (tipa klase **Screen**) u odgovarajuće stanje - ssOFF ukoliko je monitor u stanju sOUT, a ssON ukoliko je monitor u stanju sTEST

```
bool turnOut(){
    if (state != sOUT){
        state = sOUT;
        screen.setState(ssOFF);
        return true;
    }
    return false;
}

bool turnTest(){
    if (state == sON){
        state = sTEST;
        screen.setState(ssON);
        //moze i bez ovoga, jer je u sON stanje ekrana vec ssON
        return true;
    }
    return false;
}
```

Rešenje: Klasa Monitor - metode

- Metode turnStandBy() i repair() su implementirane na analogan način prethodnim metodama
- Treba voditi računa u kojim stanjima se operacija uspešno izvršava kao i stanja u kojima i monitor i ekran treba da budu na kraju izvrsavanja operacije

```
bool turnStandBy(){
    if (state == sON){
        state = sSTANDBY;
        screen.setState(ssOFF);
        return true;
    }
    return false;
}

bool repair(){
    if (state == sOUT){
        state = sOFF;
        screen.setState(ssOFF);
        // moze i bez ovoga, jer je u sOUT stanje ekrana vec ssOFF
        return true;
    }
    return false;
}
```

Rešenje: Klasa Monitor - metode

- Metoda incB() pojačava osvetljenje monitora
- Osvetljenje monitora se čuva u okviru polja *screen*, tako da je neophodno promeniti osvetljenje pozivanjem metode incB() nad poljem *screen* (ova metoda se nalazi u okviru klase **Screen**)
- Osvetljenje može da se pojačava/smanjuje samo u stanjima sOFF, sOUT i sSTANDBY

```
bool incB(){
    //mozemo i bez ovih uslova ako pazimo da nam se stanja ekrana menjaju ispravno
    if (state != sOFF && state != sOUT && state != sSTANDBY){
        return screen.incB();
    }
    return false;
}

bool decB(){
    if (state != sOFF && state != sOUT && state != sSTANDBY){
        return screen.decB();
    }
    return false;
}
```

Rešenje: Funkcija za ispis polja ekrana

- Slobodna funkcija za ispis polja treba da ispiše vrednost svakog od polja objekta
- Ovu funkciju pišemo u *main.cpp* fajlu
- Geter za polje *state* klase **Screen** vraća nabrojiv tip podataka
- Prilikom ispisa nabrojivog tipa podataka C++ konvertuje vrednosti u celobrojne vrednosti (ssON u 0, ssOFF u 1)
- Kako bismo ispisali nabrojivi tip kao tekst, moramo da proverimo pomoću neke od naredbi grananja koja vrednost se nalazi u tom polju - u ovom slučaju se koristi ternarni operator
- Ako je stanje ssOFF ispisuje se na ekran **OFF**, inače se ispisuje **ON**

```
void printScreen(Screen s){
    cout << "-- Screen --" << endl;
    cout << "Brightness: " << s.getBrightness() << endl;
    cout << "Screen state: "<< (s.getState() == ssOFF ? "OFF" : "ON") << endl;
}
```

Rešenje: Funkcija za ispis polja ekrana

- Drugi način da se reši ispis nabrojivog tipa na ekran je pomoću if naredbe
- Princip je isti kao u prethodnom slučaju, samo umesto ternarnog operatora koristimo if
- Ovaj problem je takođe moguće rešiti pomoću switch-case konstrukcije

```
void printScreen(Screen s){
    cout << "-- Screen --" << endl;
    cout << "Brightness: " << s.getBrightness() << endl;
    cout << "Screen state: ";

    if(s.getState()==ssOFF){
        cout<<"OFF"<<endl;
    }
    else{
        cout<<"ON"<<endl;
    }
}
```

Rešenje: Funkcija za ispis polja monitora

- Slično kao i kod prethodne funkcije za ispis i ovde ispisujemo vrednosti svih polja
- U ovom primeru ispis nabrojivog tipa je implementiram pomoću switch-case konstrukcije, naravno ovo je moglo da se odradi kao i u prethodnim primerima
- Treba obratiti pažnju da je ovde potrebno ispisati vrednosti svih polja klase **Monitor**, međutim kako je polje screen samo po sebi poseban objekat, potrebno je ispisati i sva njegova polja (zato ispisujemo *brightness* i *state* polja iz klase **Screen** - ova polja dobijamo pomoću odgovarajućih get metoda iz klase **Monitor** slajd 9.)

```
void printMonitor(Monitor m){
    cout << "-- Monitor --" << endl;

    cout << "Brightness: " << m.getBrightness() << endl;
    cout << "Screen state: " << (m.getScreenStates() == ssOFF ? "OFF" : "ON") << endl;

    cout << "Monitor state: ";
    switch (m.getMonitorState()){
        case sOUT:
            cout << "OUT";
            break;
        case sTEST:
            cout << "TEST";
            break;
        case sSTANDBY:
            cout << "STANDBY";
            break;
        case sOFF:
            cout << "OFF";
            break;
        case sON: cout << "ON";
            break;
        default:
            cout << "Nepostojece stanje";
    }
    cout << endl;
}
```

Rešenje: Main funkcija

- U main funkciji je **obavezno** pozvati svaku metodu i konstruktor iz svake klase **bar jednom**
- U rešenju ovog zadatka to je postignuto implementacijom menija - međutim ovo nije obavezno, dovoljno je na bilo koji način pozvati sve metode/konstruktore
- U *main.cpp* fajlu neophodno je uključiti fajlove gde su implementirane klase **Screen** i **Monitor** tj. fajlove *monitor.hpp* i *screen.hpp*
- Ovo se radi kao i inače pomoću direktive *include*

```
int main()
{

    Screen s;
    Monitor m;

    int n;
    do {

        n = meni();

        switch (n){
            case 1: s.setState(ssON); break;
            case 2: s.setState(ssOFF); break;
            case 3: s.incB(); break;
            case 4: s.decB(); break;
            case 5: m.turnOn(); break;
            case 6: m.turnOff(); break;
            case 7: m.turnOut(); break;
            case 8: m.repair(); break;
            case 9: m.turnTest(); break;
            case 10: m.turnStandBy(); break;
            case 11: m.incB(); break;
            case 12: m.decB(); break;
            case 13: cout << "Kraj"; break;
            default: cout << "Nepostojeca operacija" << endl;
        }
        printScreen(s);
        printMonitor(m);

    } while (n != KRAJ);

    return 0;
}
```

Literatura

1. Kupusinac A. : Zbirka rešenih zadataka iz programskog jezika C++. Novi Sad: FTN, 2011.