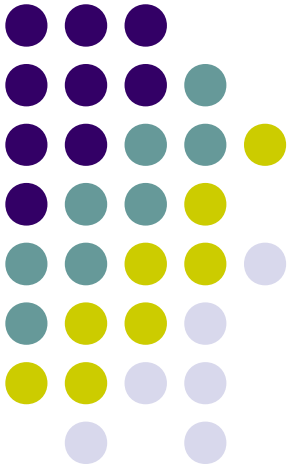
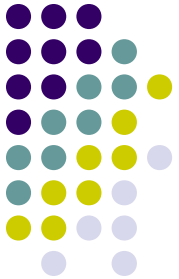
Średniozaawansowane programowanie w C++

Wykład #2

20 października 2016 r.



Plan



1. Kontenery std/boost:

a) std::bitset, boost::dynamic_bitset
b) boost::optional
c) std::pair, std::tuple

2. Wzorce projektowe

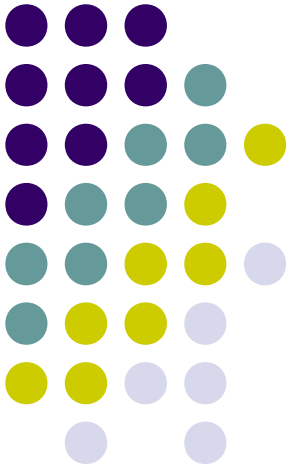
a) singleton
b) proxy
c) obserwator
d) fabryka

Kontenery std/boost

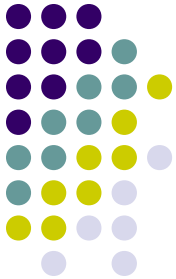
std::bitset,
boost::dynamic_bitset

boost::optional

std::pair, std::tuple



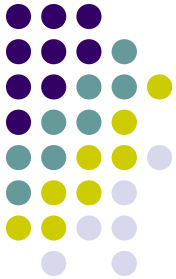
std::bitset



```
#include <bitset>

std::bitset<12> wyjscie, wejscie;       // rejest 12-bitowy

wyjscie [3] = 1;         // ustawiamy 3. bit na '1'
wyjscie.set (8) = 1;     // ustawiamy 8. bit na '1'

wyslij_przez_spi (~wyjscie);    // funkcja użytkownika (op. negacji bit.)

wejscie = czytaj_przez_spi ();  // funkcja użytkownika
wejscie.flip ();         // negacja bitowa

if (wejscie [11])        // MSB zapalony
{
        // (…) coś robimy
}
```

Zalety: dowolny rozmiar (niekoniecznie 8*n*), łatwy dostęp do poszczególnych bitów, metody wykonujące podstawowe operacje
Wady: stały rozmiar

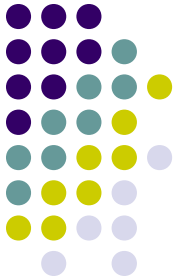
boost::dynamic_bitset



```
#include <boost/dynamic_bitset.hpp>

boost::dynamic_bitset<> bajt_h (4, 10ul);        // ----1010
boost::dynamic_bitset<> bajt_l (8, 255ul);       // 11111111
// albo: boost::dynamic_bitset<> bajt_l ("11111111");
// albo: boost::dynamic_bitset<> bajt_l = 0b11111111;   // C++14

boost::dynamic_bitset<> rej12 (bajt_h);          // 1010

rej12.resize (12);                               // 000000001010
rej12 <<= 8;                                     // 101000000000
rej12 |= bajt_l;                                 // 101011111111

std::cout << "W rejestrze zapalonych jest " << rej12.count() << " bajtów";
```

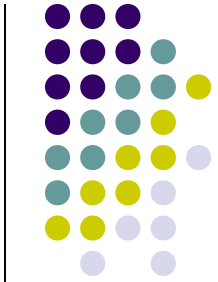
Zalety: jak std::bitset + możliwość zmiany rozmiaru, dobrze działające operatory przesunięcia bitowego
Wady: nieco rozwlekła składnia

std::pair



```
#include <utility>

typedef std::pair<int, int> Pixel;

Pixel znajdz_dziure_w_calym (BITMAP *bmp)
{
        int x, y;
        // (…) szuka dziury w całym
        return std::make_pair (x, y);
}

int main ()
{
        Pixel polozenie_dziury = znajdz_dziure_w_calym (screen);
        int ded_x, ded_y;
        std::cin >> ded_x >> ded_y;
        if (polozenie_dziury == std::tie (ded_x, ded_y))
                std::cout << "Zgadles polozenie dziury!" << std::endl;
        else
                std::cout << "Nie zgadles!\nPolozenie dziury to:" <<
                polozenie_dziury.first << ',' << polozenie_dziury.second;
}
```

boost::tuple / std::tuple



```
#include <boost/tuple/tuple.hpp>
#include <boost/tuple/tuple_io.hpp>
#include <tuple>

boost::tuple <float, float, float, float> punkt_pomiarowy;
std::cin >> punkt_pomiarowy;    // wczytuje 4 liczby zmiennoprzec.

float temperatura = boost::get<1> (punkt_pomiarowy);

float t, hf;
boost::tie(t, boost::tuples::ignore, hf, boost::tuples::ignore) =
        punkt_pomiarowy;        // zapisuje 1. i 3. składową w t i hf

boost::tuple <std::string, float> param (std::string ("Rozmiar"), 3.1415);
```

Więcej fantastycznych sposobów na zabawę z tuplami:
http://www.boost.org/doc/libs/1_46_0/libs/tuple/doc/tuple_users_guide.html

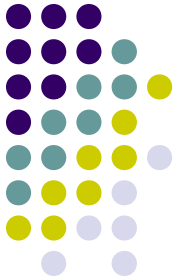
C++11
std::tuple

boost::optional

```
#include <boost/optional/optional.hpp>

class Pracownik
{
        private:
                std::string imie_;
                std::string nazwisko_;
                int wiek_;
                boost::optional<std::string> wyznanie_;
        public:
                Pracownik (/* (…) */ std::string wyzn = std::string ()) {
                        // (…) inicjalizacja innych składowych
                        if (!wyzn.empty()) wyznanie_ = wyzn;
                }
                boost::optional<std::string> wyznanie () const {
                        return wyznanie_;
                }
};

Pracownik jasio (/* inicjalizacja */);
if (jasio.wyznanie ()) std::cout << *jasio.wyznanie ();
        else std::cout << "Jasio nie podał swojego wyznania";
```

C++17
std::optional

Inne ciekawe kontenery



variant – nowoczesna unia

a) potrafi przechowywać obiekty typów inne, niż wbudowane (tj. klasy użytkownika, kontenery STL itd.)
b) kontrola typów podczas odczytu

any – przechowuje cokolwiek

a) przechowuje obiekty dowolnego typu
b) kontrola typów podczas odczytu
c) doskonały do przechwytywania wartości o nieznanym/zmiennym typie (np. wyniki zapytać SQL)

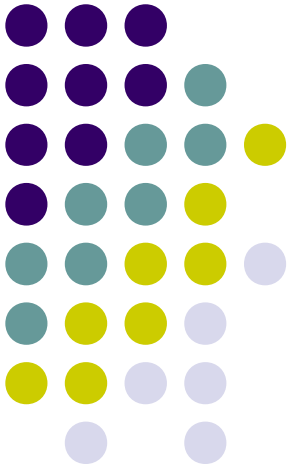
Wzorce projektowe

singleton

proxy

obserwator

fabryka



Singleton

```
class Singleton
{
        private:
                static Singleton *instancja_;
                Singleton ();   // prywatny konstruktor!
                // (…) inne składowe prywatne
        public:
                static Singleton &instancja () {
                        if (!instancja_) // tworzy obiekt przy 1. użyciu
                                instancja_ = new Singleton;
                        return *instancja_;
                }
                double rob_cos (int k);
};

// dynamiczne stworzenie obiektu przy pierwszym użyciu
std::cout << Singleton::instancja().rob_cos (17) << std::endl;
// używanie tego samego obiektu
std::cout << Singleton::instancja().rob_cos (45) << std::endl;
```

Wzorzec zapewnia istnienie tylko jednego obiektu klasy Singleton w programie!

Proxy

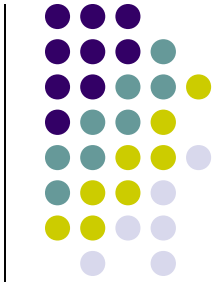
```
class Obraz
{
        public: virtual void rysuj (BITMAP *ekran) const = 0;
};

class ObrazProxy : public Obraz
{
        private:
                Obraz *obr_;
                std::string nazwa_pliku_;
        public:
                ObrazProxy (std::string nazwa) : nazwa_pliku_ (nazwa),
                        obr_ (NULL) {}
                ~ObrazProxy () { delete obr_; }
                virtual void rysuj (BITMAP *ekran) const {
                        if (!obr_) obr_ = new ObrazReal (nazwa_pliku_);
                        obr_->rysuj (ekran); // ObrazReal::rysuj (BITMAP*)
                }
};

class ObrazReal : public Obraz
{/*      przeciąża metodę wirtualną rysuj (faktycznie rysującą!)
        i dostarcza własnych, potrzebnych składowych */
};
```

Proxy (cd.)

```
std::string nazwa_pliku;
std::vector <Obraz*> obrazy;

while (std::cin >> nazwa_pliku)
        // tylko tworzy obiekty, nie wczytuje obrazów
        obrazy.push_back (new ObrazProxy (nazwa_pliku));

obrazy [4] -> rysuj (screen);   // wczytuje i rysuje!
obrazy [2] -> rysuj (screen);   // wczytuje i rysuje!

// (…)

obrazy [3] -> rysuj (screen);   // wczytuje i rysuje!
obrazy [2] -> rysuj (screen);   // już wczytany, tylko rysuje!
```

Wczytuje „prawdziwy” obiekt dopiero przy pierwszym użyciu.

Obserwator

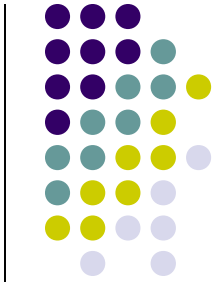


```
class Obserwowany
{
    public:
        void dodajObserwatora (Obserwator *o) {
                obs_.push_back (o);
        }
        void powiadomObserwatorow () {
                for (std::vector <Obserwator*>::iterator it = obs_.begin();
                        it != obs_.end(); ++it)
                        it -> odswiez ();
        }
        virtual ~Obserwowany ();
    private:
        std::vector <Obserwator*> obs_; // obserwatorzy
};

class Termometr : public Obserwowany
{
    public:
        double temperatura () const;    // zwraca mierzoną wartość
    private:
        // (…)
};
```

Obserwator (2)

```
class Obserwator
{
    public:
        virtual void odswiez () = 0;
        virtual ~Obserwator ();
};

class KontrolkaTemp : public Obserwator
{
    public:
        KontrolkaTemp (const Termometr *trmtr);
        virtual void odswiez ();        // odczytuje wartość z termometru
        virtual ~KontrolkaTemp ();
    private:
        // (…)
};

// Przykład użycia
Termometr okienny;
KontrolkaTemp lcd (okienny);
okienny.dodajObserwatora (&lcd);
```

Fabryka



```
class Jablko
{
        private:
                // (…)
        public:
                Jablko (unsigned ilosc_pestek);
                virtual ~Jablko ();
};

typedef (Jablko*) (*KreatorJablek) (unsigned ip); // wskaźnik na metodę

class Antonowka : public Jablko
{
        private:
                // (…)
        public:
                Antonowka (unsigned ip) : Jablko (ip) {/* (…) */}
                virtual ~Antonowka ();
                static Jablko *kreator (unsigned ip)
                {
                        return new Antonowka (ip);
                }
                static const Identyfikator ID;  // jakiś unikalny id.
};
```

Fabryka (2)

```
class FabrykaJablek
{
   private:
        static std::map <Identyfikator, KreatorJablek> znane_jablka;
   public:
        static void rejestruj (const Identyfikator &id, KreatorJablek kj)
        {
                znane_jablka [id] = kj;
        }
        Jablko *utworz (const Identyfikator &id, unsigned ip)
        {
                // woła odpowiedni kreator z parametrem konstruktora
                return znane_jablka [id] (ip);
        }
};

// rejestrujemy jabłuszka w fabryce
FabrykaJablek::rejestruj (Antonowka::ID, &Antonowka::kreator);
FabrykaJablek::rejestruj (Papierowka::ID, &Papierowka::kreator);

// dostajemy żądane jabłuszko
Jablko *ant_z_5_pestkami = FabrykaJablek::utworz (Antonowka::ID, 5);
```

Programowanie jest fantastyczne!!!