Programowanie Obiektowe

Marcin Kamil Bączyk

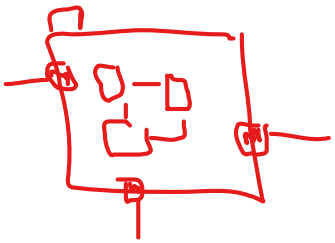
Wykład 10

10 grudnia 2020

Treść dzisiejszego wykładu

- model - widok - kontroler
- programowanie zdarzeniowe
- wzorzec projektowy «obserwator»
- wprowadzenie do biblioteki graficznej Qt
- przykłady

model - widok - kontroler



definiuje między innymi:
- logikę biznesową aplikacji oraz
- strukturę danych

Model

aktualizacja wyświetlanych treści

przekazanie rozkazów do wykonania

Widok

komunikacja na potrzeby realizacji interfejsu użytkownika

Kontroler

definiuje interfejs użytkownika

tworzy rozkazy i komunikuje je modelowi

programowanie zdarzeniowe

Metodyka oparta o założenie, że w danym systemie istnieje ograniczona ilość możliwych zdarzeń, które mogą wystąpić:

- wciśnięcie klawisza,
- kliknięcie lewym/prawym przyciskiem myszy,
- ...

Programista decyduje, które zdarzenia obsługuje i jaki jest efekt obsłużenia danego zdarzenia. Zazwyczaj programista nie decyduje o tym jaki mechanizm jest wykorzystany do powiązania danego zdarzenia z procedurą jego obsługi – jest narzucony przez środowisko.

Metodyka szeroko wykorzystywana do tworzenia aplikacji graficznych z interfejsem użytkownika. W systemie implementowana jest pętla komunikatów, do której trafiają konkretne zdarzanie i która informuje inne obiekty o zaistniałym zdarzeniu. Mechanizm ten jest zazwyczaj niedostępny dla programisty (prawie nigdy).

Często obsługa zdarzeń w systemie impementowana według wzorca projektowego «obserwator».

Ciekawostka

Podobny mechanizm implementowany jest w układach mikrokontrolerów (procesorów). Zdarzeniu odpowiada przerwanie zgłaszane od na przykład. kontrolera pamięci lub innego urządzenia zewnętrznego / wewnętrznego. Mechanizm obsługi wektora przerwań polega na podpięciu adresów funkcji wywoływanych w odpowiedzi na dane przerwanie. Na przykład przerwanie od portu szeregowego wyzwala procedurę odbiorku danych przez ten port.

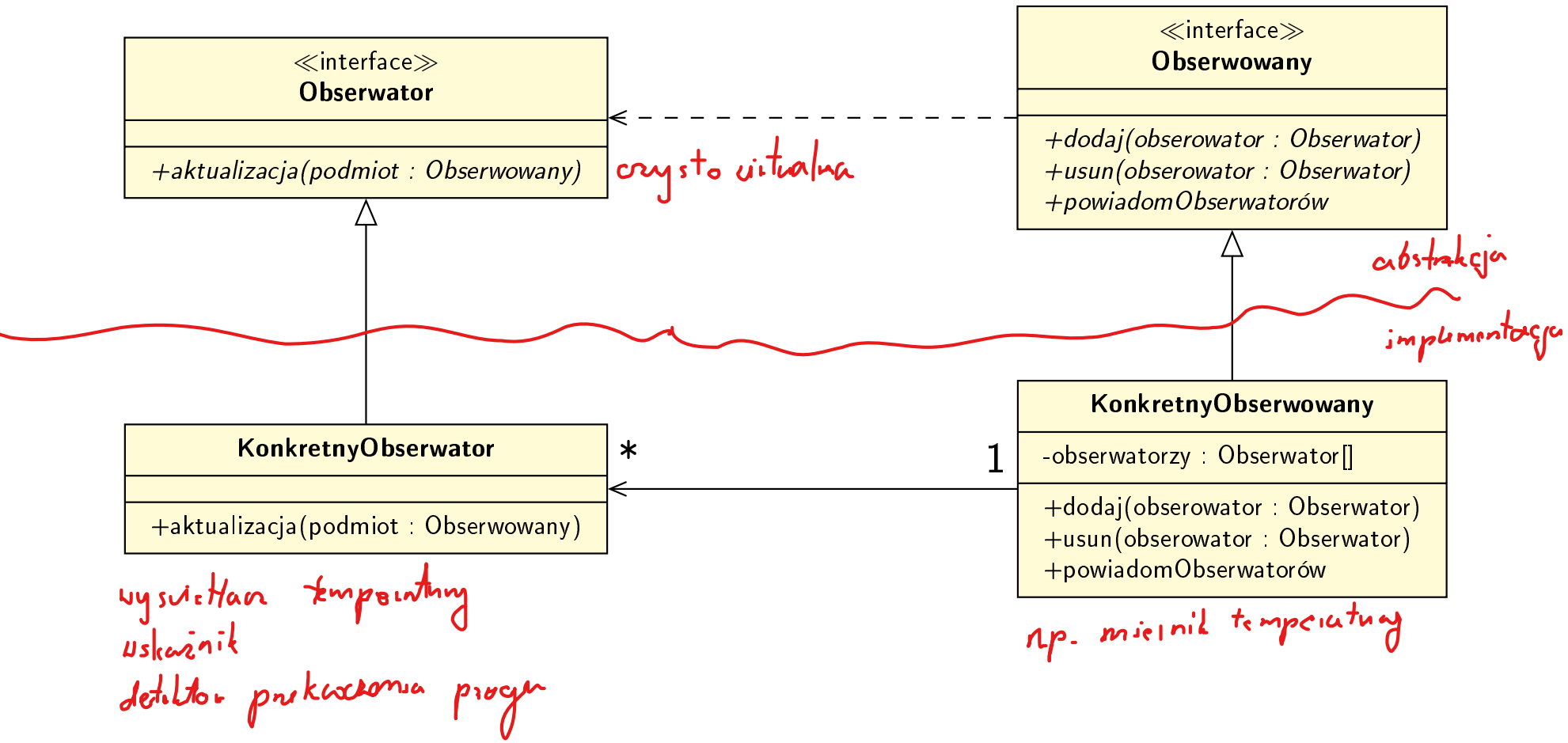
zalety

- ograniczenie kodu do opisu tylko obsługiwanych zdarzeń
- łatwość opisania interakcji z otoczeniem
- posługiwanie się inną metodologią podczas samego opisu funkcji obsługi zdarzeń

wady

- wykorzystanie ukrytej, lecz de facto istniejącej części programu, odpowiadającej za obsługę pętli komunikatów (czytaj: dodatkowe sprzężenia)
- konieczność znajomości możliwych zdarzeń i sposobu dostarczenia ich parametrów (czytaj: mozolne przeglądanie dokumentacji)
- skomplikowana budowa systemu (czytaj: programowania w danym środowisku trzeba się nauczyć jak programowania w danym języku)

wzorzec projektowy «obserwator»



wzorzec projektowy «obserwator»

Obserwowany

- Zna swoich obserwatorów i może go obserwować dowolna ich liczba.
- Zapewnia interfejs dodawania i usuwania obserwatorów.

Obesrwator

- Definiuje interfejs uaktualniania dla obiektów, które powinny być powiadomione o zmianach.

wzorzec projektowy «obserwator»

KonkretnyObserwowany

- Przechowuje stan.
- W razie zmiany stanu wysyła powiadomienia do swoich obserwatorów.

KonkretnyObserwator

- Przechowuje odwołanie do obserowanego obiektu.
- Przechowuje stan zgodny ze stanem obserowanego obiektu.
- Implementuje interfejs obserwatora w celu zachowania spójności swojego stanu ze stanem obserwatora.

KonkretnyObserwowany zawsze powiadamia swoich obserwatorów, gdy wystąpi zmiana jego stanu. Po otrzymaniu powiadomienia KonkretnyObserwator aktalizuje swój stan. W zależności od interfejsu powiadamiania, konieczne może być dodatkowa komunikacja w celu uzupełnienia informacji.

Qt to cały zestaw bibliotek i narzędzi programistycznych. Oprócz biblioteki do tworzenia graficznych interfejsów użytkownika – Qt GUI – dostępne są również narzędzia między innymi do komunikacji sieciowej lub między procesowej. W bibliotece Qt Core znajdują się narzędzia znane z biblioteki standardowej takie jak kontenery, operacje na ciągach znaków czy obsługa wejścia / wyjścia.
W grudniu 2020 roku wyszła najnowsza wersja biblioteki – 6. Od tej wersji biblioteka wspiera standard c++17. Wcześniejsze bazowały na wersji c++98.

biblioteka Qt

- - specyficzna, nieintuicyjna i nieco przestarzała – pierwsze wersje sięgają początku lat 90-tych XX wieku
- - niektóre założenia leżące u podstaw biblioteki odzwierciedlają (niestesty) ówczesny sposób myślenia o obiektowości
- - wprowadza rozszeżenie do języka C++ w postaci dodatkowych etapów kompilacji
- - trzeba się przestawić na „właściwe” myślenie
- + wciąż intensywnie rozwijana
- + duża baza użytkowników i dostępność projektów / dokumentacji
- + wsparcie nie tylko dla GUI
- + relatywnie duża ilość ofert pracy z wymaganą znajomością Qt

Qt : pierwszy program

```
#include <QApplication>
#include <QPushButton>

int main(int argc, char** argv)
{
    QApplication application(argc, argv);

    QPushButton button("Hello Qt");
    button.resize(200, 300);
    button.show();

    return app.exec();
}
```

Qt : pierwszy program

Utworzenie obiektu application jest niezbędne do uruchomienia środowiska graficznego. Bez utworzenia tego obiektu, żadej inny nie zostanie utworzony, pomimo braku widocznej zależności pomiędzy poszczególnymi obiektami.
To co znajduje się w warstwie graficznej programu definiowane jest między utworzeniem obiektu klasy QApplication a wywołaniem metody exec na tym obiekcie.
W przykładzie tworzony jest jeden przycisk. Z wciśnieciem przycisku nie jest związana żadna inna akcja. Program prócz wyświetlenia przycisku z napisem Hello Qt nie robi nic.

```
#include <QApplication>
#include <QMainWindow>
#include <QPushButton>
#include <QLabel>

int main(int argc, char** argv) {
    QApplication application(argc, argv);

    QMainWindow main_window;
    main_window.setWindowTitle("main window");
    main_window.setGeometry(100, 100, 300, 300);

    QLabel label("Just a label", &main_window);
    label.setGeometry(10, 10, 100, 50);

    QPushButton button("Hello Qt", &main_window);
    button.setGeometry(10, 60, 100, 50);

    main_window.show();

    return application.exec();
}
```

przekazanie przez wskaźnik (adres)

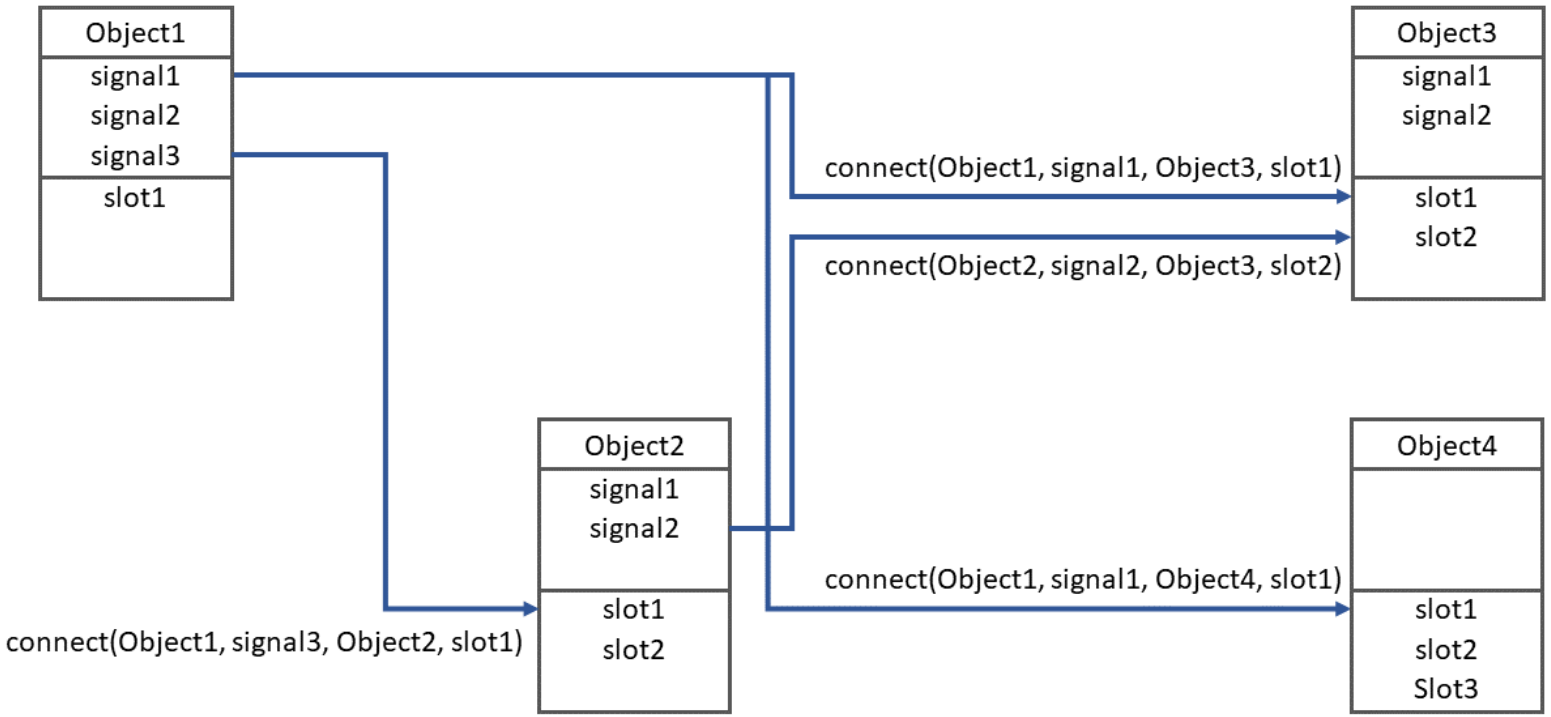
Utworzenie okna głównego pozwala na dodawanie kolejnych elementów do tego okna i sterowanie ich pozycjami.
Dodawane elementy do okna muszą mieć przekazany adress swojego „rodzica”.
Klasy reprezentujące elementy graficzne w bibliotece Qt dziedziczą po klasie QWidget, również QMainWindow. Dlatego w pierwszym programie możliwe było wyświetlenie pojedynczego przycisku jako całego okna. Jedna tylko niektóre elementy graficzne pozwalają dodawać kolejne elementy i wyświetlać je w spósób prawidłowy.

```
QPushButton button("Hello Qt", &main_window);
button.setGeometry(0, 0, 100, 50);

auto fun = [&]() {button.move(button.pos() + QPoint(5, 5));};

QObject::connect(&button, &QPushButton::clicked, &button, fun);
```

obiekt funkcyjny
wywołanie (wołanie)
obserwowany
obserwator

Zdarzenia w Qt realizowane są porzez mechanizm gniazd (slotów) oraz sygnałów. Istnieje wiele różnych sygnałow, które mogą być wysłane. Każdy obiekt może wysyłać różne sygnały i wiele z nich odbierać.

Powiązanie sygnału z funkcją zwrotną realizującą ten sygnał odbywa się poprzez wyłowanie statycznej metody connect klasy QObject. Pierwszym argumentem metody connect jest obiekt wysyłający sygnał, drugim rodzaj sygnału, trzecim jest odbiorca sygnału. Ostatnim argumentem jest metoda, która ma być wywołana gdy odebrany zostanie sygnał danego typu. Z jednym sygnałem może być powiązanych więcej niż jedno gniazdo.


Object1
signal1
signal2
signal3
slot1
Object3
signal1
signal2
slot1
slot2
connect(Object1, signal1, Object3, slot1)
connect(Object2, signal2, Object3, slot2)
Object2
signal1
signal2
slot1
slot2
Object4
slot1
slot2
Slot3
connect(Object1, signal1, Object4, slot1)
connect(Object1, signal3, Object2, slot1)

my_main_window.hpp

```
#include <QMainWindow>
#include <QPushButton>

class MyMainWindow : public QMainWindow
{
public:
    MyMainWindow() : button("Hello Qt", this)
    {
        setWindowTitle("main window");
        setGeometry(100, 100, 300, 300);

        QObject::connect(&button, &QPushButton::clicked,
            this, &MyMainWindow::moveButton);
    }

private slots:
    void moveButton()
    {
        button.move(button.pos() + QPoint(5, 5));
    }

private:
    QPushButton button;
};
```

main.cpp

```
#include <QApplication>
#include "my_main_window.hpp"

int main(int argc, char** argv)
{
    QApplication application(argc, argv);

    MyMainWindow main_window;
    main_window.show();

    return application.exec();
}
```

W obiektowej strukturze oprogramowania łatwiej zapanować nad tym co się dzieje. Obiekty definiują swoje sygnały i gniazda. Przepływ sterowania jest zdecydowanie bardziej rozproszony. Podejście takie nazywane jest programowaniem zdarzeniowym.