Programowanie Obiektowe

Marcin Kamil Bączyk

Wykład 1

11 października 2018

Informacje o przedmiocie

- Prowadzący wykład : mgr inż. Marcin Bączyk M.K.Baczyk@elka.pw.edu.pl, pokój 449, konsultacje w poniedziałki w godzinach 11-12.
- istnieje możliwość umawiania się na inny termin konsultacji
- strona przedmiotu: http://staff.elka.pw.edu.pl/ mbaczyk1/PROE/index.html
- strona będzie systematycznie aktualizowana, aż do końca semestru.

Zaliczenie

- PROE jest przedmiotem zaliczeniowym - ostatni możliwy termin oddawania projektów, poprawiania sprawdzianów itp. przypada na ostatni dzień semestru - 25 stycznia 2019 roku
- Przedmiot podzielony jest na dwie części : wykład (14) oraz laboratoria (14)
- W trakcie wykładu odbędą się dwa sprawdziany wykładowe po 25 punktów każdy
- W trakcie zajęć laboratoryjnych odbędzie się pięć zajęć ocenianych po 5 punków każde
- W trakcie zajęć laboratoryjnych będą realizowane dwa małe projekty po odpowiednio 10 oraz 15 punktów
- Prowizoryczny harmonogram laboratorium dostępny na stronie przedmiotu
- Terminy sprawdzianów oraz zajęć laboratoryjnych ustalone zostaną w przyszłym tygodniu.

Zaliczenie

- Aby zaliczyć przedmiot należy spełnić następujące warunki:
  - uzyskać łącznie minimum 25 punktów ze sprawdzianów wykładowych
  - uzyskać łącznie minimum 25 punktów z laboratorium
  - uzyskać 51 punktów z całego przedmiotu
- Progi ocen
  - $\geq$ 91 - 5
  - $\geq$ 81 - 4.5
  - $\geq$ 71 - 4
  - $\geq$ 61 - 3.5
  - $\geq$ 51 - 3
  - $<$ 51 - 2

Laboratorium

- Terminy
  - wtorek 10:15 - 12:00 - 2 grupy
  - wtorek 16:15 - 18:00 - 1 grupa
  - piątek 12:15 - 14:00 - 2 grupy
- limit miejsc w grupie : 13 osób

Treść wykładu i efekty kształcenia

Treść wykładu

- Zapoznanie się z elementami projektowania obiektowego
- Zapoznanie się z możliwościami języka obiektowego C++
- Zapoznanie z możliwościami podstawowych bibliotek C++
- Zapoznanie się z elementami standardu C++11, C++14 oraz C++17

Efekty kształcenia

- Zapoznanie z paradygmatem programowania obiektowego.
- Zdobycie umiejętności rozumienia kodu napisanego w C++.
- Zdobycie umiejętności pisania własnych aplikacji C++.
- Zdobycie umiejętności samodzielnego poszukiwania niezbędnych informacji.

Treść wykładu

Projektowanie obiektowe

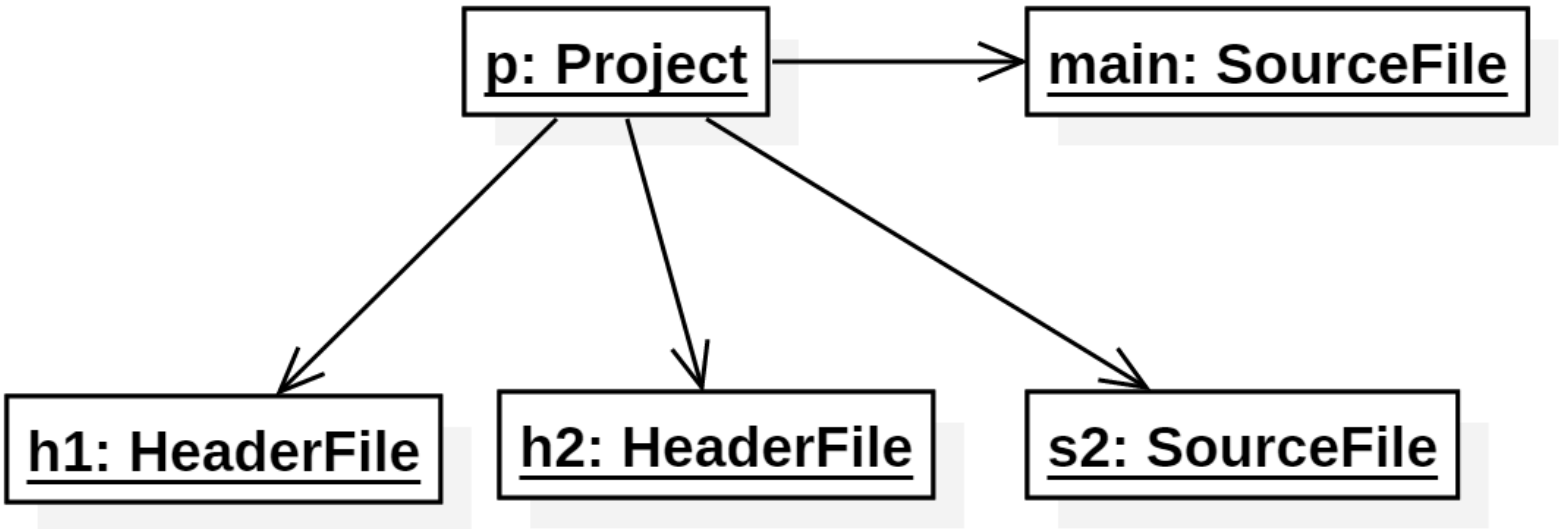
- Abstrakcja
- Hermetyzacja
- Polimorfizm
- Dziedziczenie

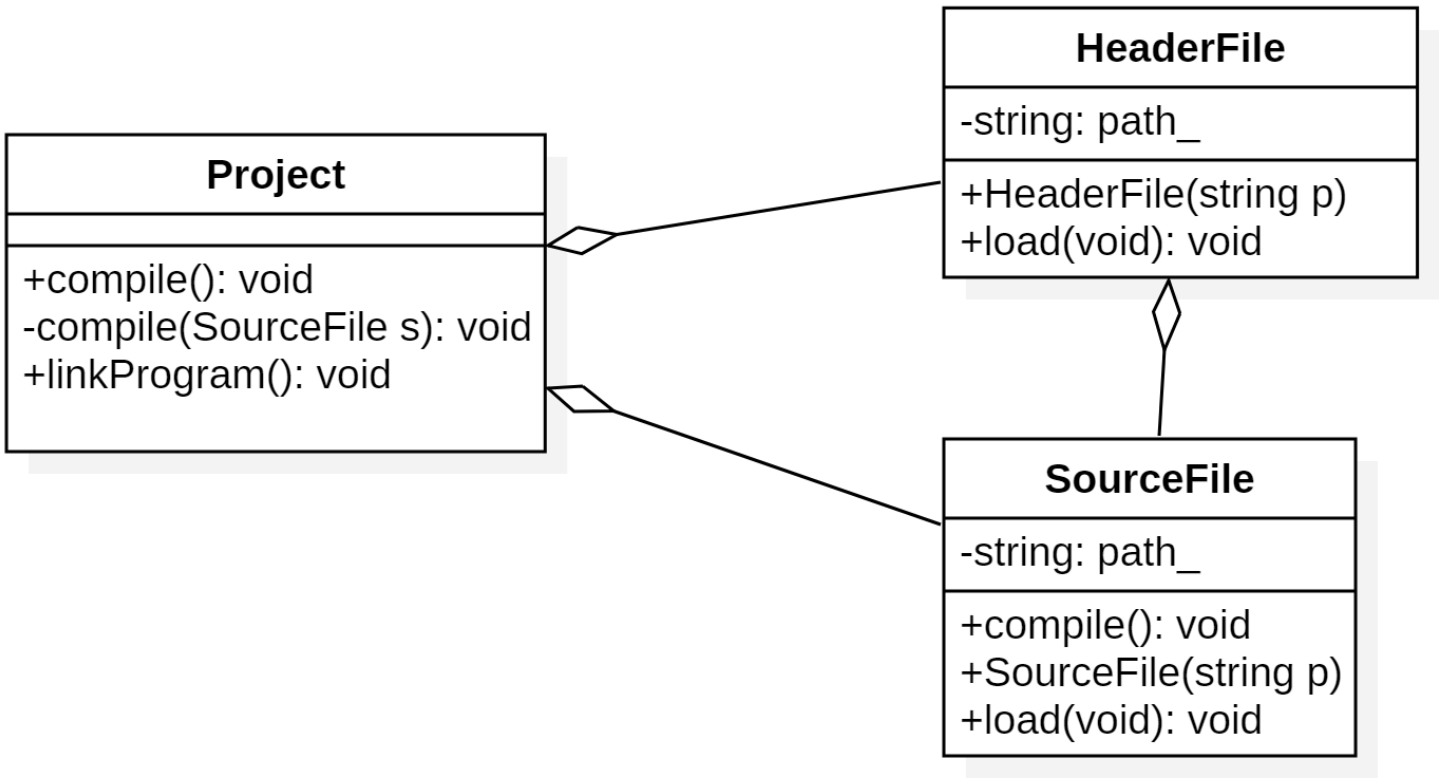
Składnia języka C++

- Klasy
- Konstruowanie i niszczenie obiektów
- Szablony
- Obsługa wyjątków
- Zarządzanie pamięcią
- ...

Nauka programowania

- W pierwszej kolejności uczestnictwo w zajęciach laboratoryjnych
- Samodzielna realizacja projektów
- Polecana literatura:
  - "Język C++. Kompendium wiedzy" - Stroustrup Bjarne
  - "Podstawy języka C++" - Stanley Lippman
  - "Poznaj C++ w 10 minut" - Jesse Liberty
  - https://en.cppreference.com/w/
  - https://stackoverflow.com/
  - "Język C++. Standardy kodowania. 101 zasad, wytycznych i zalecanych praktyk" - Herb Sutter, Andrei Alexandrescu
  - "Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku" - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides


p: Project
main: SourceFile
h1: HeaderFile
h2: HeaderFile
s2: SourceFile


HeaderFile
-string: path_
+HeaderFile(string p)
+load(void): void
Project
+compile(): void
-compile(SourceFile s): void
+linkProgram(): void
SourceFile
-string: path_
+compile(): void
+SourceFile(string p)
+load(void): void

Elementy C++ umożliwiające programowanie obiektowe

- Klasy oraz obiekty będące ich instancjami
- Dziedziczenie, dziedziczenie wielobazowe
- Konstruktory oraz destruktory
- Metody wirtualne klas
- Zarządzanie pamięcią
- Dynamiczna kontrola typów

C++ - klasa a obiekt

Klasa:

Jest to definicja wzorca (opis wszystkich danych i dostępnych metod) według którego będą tworzone obiekty danego typu. Innymi słowy klasa to nowy typ zmiennych definiowany przez użytkownika.

Obiekt:

Obiekty są to instancje konkretnych klas (typów własnych).

Pierwsza klasa

Podobnie jak język C pozwala stworzyć pustą strukturę, język C++ pozwala stworzyć pustą klasę.

```
class First;
class First {};

int main(void)
{
    First f1;
    return 0;
}
```

- słowo kluczowe
- deklaracja klasy
- definicja klasy
- obiekt

Czy istnieje jakieś wytłumaczenie dla istnienia pustej klasy? Jaki jest jej rozmiar?

Typy w C++

typ prosty

Typ nazwaZmiennej;

np. float, double, int, ... oraz

np. klasy, struktury, ...

typ wskaźnikowy

Typ * nazwaZmiennejWskaznikowej = &nazwaZmiennej

Typ wskaźnikowy oznacza adres pod jakim zapisana jest dana zmienna

typ referencyjny

Typ & nazwaZmiennejReferencyjnej = nazwaZmiennej

Typ referencyjny należy rozumieć jako odnośnik do zmiennej

Typ referencyjny

L-referencja

Typ & nazwaZmiennejReferencyjnej = nazwaZmiennej

- definicja zmiennej referencyjnej zawsze wymaga podania obiektu (inicjacji)
- związku pomiędzy zmienną referencyjną a obiektem nie można zerwać
- nie można zdefiniować referencji do referencji
- nie można zdefiniować wskazania na referencję (ale referencję do wskazania już tak) - w związku z powyższym nie można używać tablic referencji

R-referencja

Typ && nazwaZmiennejReferencyjnej = zmiennaNieposiadajaNazwy

Typy w C++ - przykłady

```
int i = 1;          // i = 1
int *pi;
pi = &i;            // i = 1
*pi = 2;            // i = 2
// int & ri - Uwaga Blad
int & ri = i;       // i = 2
ri++;               // i = 3

First f;
First *pf = &f;
// First &rf;       //'rf': references must be initialized
First &rf = f;
```

Referencja wskazuje na dany obiekt, również wtedy gdy obiekt przestanie istnieć (np. pamięć zostanie zwolniona). Język C++ pozwala zwrócić referencję do obiektu tymczasowego - to nigdy nie kończy się dobrze.

Klasa nieco bardziej złożona

W jaki sposób możemy przechowywać informację od czasie?

```
class Time
{
public:
    short h_;
    short m_;
    double s_;

    double timeDifference(const Time t);
};

double Time::timeDifference(const Time t)
{
    return ((h_ - t.h_) * 60 + m_ - t.m_) * 60 + s_ - t.s_;
}
```

Czy klasa potrzebuje dodatkowych metod? Czy są potrzebne inne pola klasy?

Klasa nieco bardziej złożona

Jak wykorzystać przygotowaną klasę?

```
#include <iostream>
#include "Time.hpp"

int main(void){
Time t1 = { 14, 15, 0 };
Time t2 = { 16,  0, 0 };

std::cout << "Wyklad␣trwa␣";
std::cout << t2.timeDifference(t1);
std::cout << "␣sekund." << std::endl;
}
```

std::cout jest obiektem globalnym dołączanym wraz z biblioteką iostream. std::cout obsługuje standardowe wyjście. « oznacza specjalną metodę klasy std::ostream, której instancją jest obiekt std::cout. Metoda ta wypisuje na ekran informacje o obiektach podstawowych.

Przykład bardziej użyteczny - cd

W jaki sposób możemy przechowywać informację od czasie?

```
class Time
{
public:
    Time(short h, short m, double s);
    Time(double s);
    Time(const Time& t);

    double timeDifference(const Time& t);
private:
    double s_;
};
```

W jaki sposób w C++ realizowane są abstrakcja, hermetyzacja oraz polimorfizm (statyczny)?

Zakres dostępu do elementów i metod klasy

publiczny

- słowo kluczowe: public
- możliwe jest wywoływanie metod oraz modyfikowanie pól klasy poza klasą
- zbyt szeroka zakres publiczny przeczy idei hermetyzacji
- publiczne powinny być jedynie te pola i metody które są niezbędne to realizacji odpowiedzialności obiektu

prywatny

- słowo kluczowe: private
- zabronione jest wywoływanie metod oraz modyfikowanie pól poza klasą
- jedynie inne metody tej klasy mogą wywoływać metody prywatne i modyfikować prywatne pola klasy

Pola klasy

```
private:
    double s_ = 0;
```

- reprezentują stan wewnętrzny obiektu
- w miarę możliwości powinny pozostawać prywatne
- mogą być to typy podstawowe lub inne typy złożone
- pola mogą być inicjalizowane wartością lub innym obiektem
- klasa może mieć dowolną ilość pól, jednakże ich zbyt duża ilość sugeruje, że klasa ma zbyt wiele odpowiedzialności!

Metody klasy

```
public:
    Time(short h, short m, double s);
    Time(double s);
    Time(const Time& t);

    double timeDifference(Time t);
    void addSeconds(double s);
    void addMinutes(int m);
    void addHours(int h);
```

- reprezentują czynności jakie mogą być wykonane na obiektach
- mogą być prywatne i publiczne
- jedna klasa może mieć kilka metod o tej samej nazwie, jednak muszą się one różnić typem lub ilością przyjmowanych argumentów - polimorfizm statyczny
- klasa może mieć dowolną ilość metod

Metody klasy

```
public:
    double timeDifference(Time t);
```

- mają dostęp do wszystkich pól prywatnych i publicznych danego obiektu.
- mogą wywoływać wszystkie metody prywatne i publiczne niebędące konstruktorami
- deklaracja znajduje się w pliku nagłówkowym (pliku definicji)
- implementacja może (powinna) znajdować w pliku implementacji

Metody specjalne - konstruktory

```
public:
    Time(short h, short m, double s);
    Time(double s);
    Time(const Time& t);
```

- tak samo jak inne metody mogą być przeciążane,
- wywoływane są w momencie tworzenia obiektu / służy do inicjalizowania obiektu
- brak zwracanego typu
- nazwa identyczna z nazwą klasy
- mogą być metodami publicznymi i prywatnymi - do czego może służyć prywatny konstruktor?
- można wyodrębnić kilka rodzajów konstruktorów: domyślny (bezparametrowy), zwykły, kopiujący oraz przenoszący

Metody specjalne - konstruktor zwykły

Który sposób inicjowania zmiennych jest lepszy?

```
public:
    Time(short h, short m, double s)
    {
        s_ = (h*60 + m)*60+s;
    }
```

```
public:
    Time(short h, short m, double s)
        :  s_((h*60 + m)*60+s)
    {
    }
```

Metody specjalne - konstruktor domyślny

```
public:
    Time(void) : s_(0) {}
```

C++

```
class{
    double s_ = 0;
public:
    Time(void) {}
};
```

- brak jakiegokolwiek konstruktora spowoduje, że kompilator wygeneruje konstruktor domyślny postaci:

  ```
  Time(void) {}
  ```

- w innym przypadku konstruktor taki musi napisać programista:

  ```
  Time(void) = default;
  ```

Metody specjalne - konstruktor kopiujący

```
public:
    Time(const Time&t) : s_(t.s_) {}
```

- służy do poprawnego kopiowania obiektów
  - inicjalizacja jednego obiektu innym

    ```
    Time t1;
    Time t2(t1);
    Time t3 = t1;
    ```

  - przekazywanie obiektów do funkcji/metody

    ```
    void setTime(Time t) { /*...*/}
    ```

  - zwracanie obiektów z funkcji/metody

    ```
    Time getCurrentTime(void) { Time t; /*...*/ return t; }
    ```

Metody specjalne - konstruktor przenoszący

C++11

```
public:
    Time( Time&&t) : s_(std::move(t.s_)) {}
```

- służy do przeniesienia danych z innego obiektu, zostawiając go w stanie nieprawidłowym
  - inicjalizacja jednego obiektu innym

    ```
    Time t1;
    Time t2 = std::move(t1);
    ```

  - przekazywanie obiektów do funkcji/metody

    ```
    setTime(std::move(t2));
    ```

  - zwracanie obiektów z funkcji/metody

    ```
    return t;
    ```

Metody specjalne - destruktor

```
public:
    ~Time();
```

- każda klasa może mieć tylko jeden destruktor
- destruktor wywoływany jest automatycznie gdy niszczony jest obiekt
- służy do wykonania niezbędnych czynności przed likwidacją obiektu
- w przypadku tej metody nie podaje się typu zwracanego wyniku
- umieszcza się ją w części publicznej definicji klasy
- destruktor można wywołać w dowolnym momencie

Uniemożliwianie konstruowania obiektów w określony sposób

Dlaczego programista miałby uniemożliwić kopiowania obiektów?

```
private:
    Time(const Time& t) {};
```

C++11

```
public:
    Time(const Time& t) = delete;
```

- usunięte mogą być również inne konstruktory i metody
- programista przekazuje swoje intencje użytkownikowi
- czasami inne klasy mają dostęp do pól i metod prywatnych danej klasy

podstawowa obsługa wejścia wyjścia

```
#include <iostream>
```

Obiekty globalny obsługujące strumienie

- std::cout - standardowe wyjście

```
std::cout << "standardowe wyjscie" << std::endl;
```

- std::cerr - standardowe wyjście dla błędów

```
std::cerr << "standardowe wyjscie bledow" << std::endl;
```

- std::clog - standardowe wyjście dla logów

```
std::clog << "standardowe wyjscie log" << std::endl;
```

- std::cin - standardowe wejście

```
int A;
std::cin >> A;
```

obiekty typu string

Język C++ udostępnia wygodny dla użytkownika typ danych reprezentujący ciągi znaków - std::string.

```
#include <string>

int main()
{
std::string s1 = "John␣Smith";
std::string s2 = "Ann␣Johnson";

std::cout << s1 << std::endl;
std::cout << s1 << "␣and␣" << s2 << std::endl;

return 0;
}
```

Więcej na :
http://www.cplusplus.com/reference/string/string/

przestrzeń nazw

```
class First {};
namespace A {
    class First {};
    class Second {};
    void fun(void){}
}

int main(){
    First F1;
    A::First F2;
    A::Second S1;
    //Second S2; //niezdefiniony symbol
    A::fun();

    using namespace A;
    Second S3;
    //First F3; //niejednoznaczosc symboli
    fun();

    return 0;
}
```

przestrzeń nazw

- mechanizm wprowadzony w celu uniknięcia konfliktów nazw mogących wystąpić w przestrzeni globalnej
- mogą być zagnieżdżone
- mogą być definiowane rozłącznie, np. w wielu plikach
- mechanizm ułatwiający korzystanie z przestrzeni nazw:

```
using namespace nazwa_przestrzeni;
```

- biblioteka standardowa języka C++ posiada jedną przestrzeń nazw - **std**

Różnice między C a C++

C

```
unsigned int factorial(unsigned int i){
    if (0 == i)
        return 1;
    return i * factorial(i - 1);
}
```

C++

```
template <unsigned N>
struct Factorial{
    enum { value = N * Factorial<N - 1>::value };
};

template <>
struct Factorial<0>{
    enum { value = 1 };
};
```

Różnice między C a C++ - c.d.

C++

```
template <unsigned N>
struct Factorial{
    enum { value = N * Factorial<N - 1>::value };
};

template <>
struct Factorial<0>{
    enum { value = 1 };
};
```

nowszy C++

```
template<unsigned N>
constexpr auto Factorial()
{
    if constexpr(N == 0)    return 1;
    else return Factorial<N-1>()*N;
}
```

Różnice między C a C++ - c.d.

C++

```
int main(void)
{
std::cout << "C" << std::endl;
std::cout << "10!=" << factorial(10) << std::endl;

std::cout << "C++" << std::endl;
std::cout << "10!=" << Factorial<10>() << std::endl;
return 0;
}
```

Język programowania C++ **nie** jest językiem C z klasami!!!

Różnice między C a C++ - c.d.

C++

```
int main()
{
    constexpr auto k = Factorial<5>();
    return k;
}
```

assembler

```
main:
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-4], 120
        mov     eax, 120
        pop     rbp
        ret
```