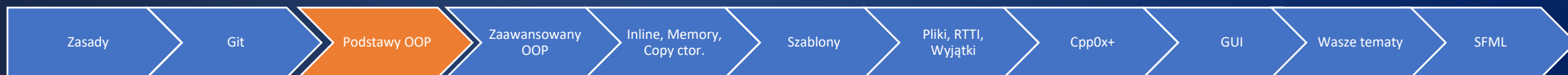


Języki i Paradygmaty Programowania II

Laboratorium 2: Podstawy OOP

W dzisiejszym odcinku

- Klasa a obiekt
- Podstawowe założenia OOP
- Klasy i obiekty w C++
 - Deklaracja i definicja
 - Dane i metody
 - Konstruktor i destructor
- Przestrzeń nazw
- Szybkie wprowadzenie do iostream



Paradygmat Programowania Obiektowego

- (Prawie) wszystko jest obiektem
- Interakcje
- Silne związanie danych z procedurami
- Mapowanie świata rzeczywistego na świat wirtualny

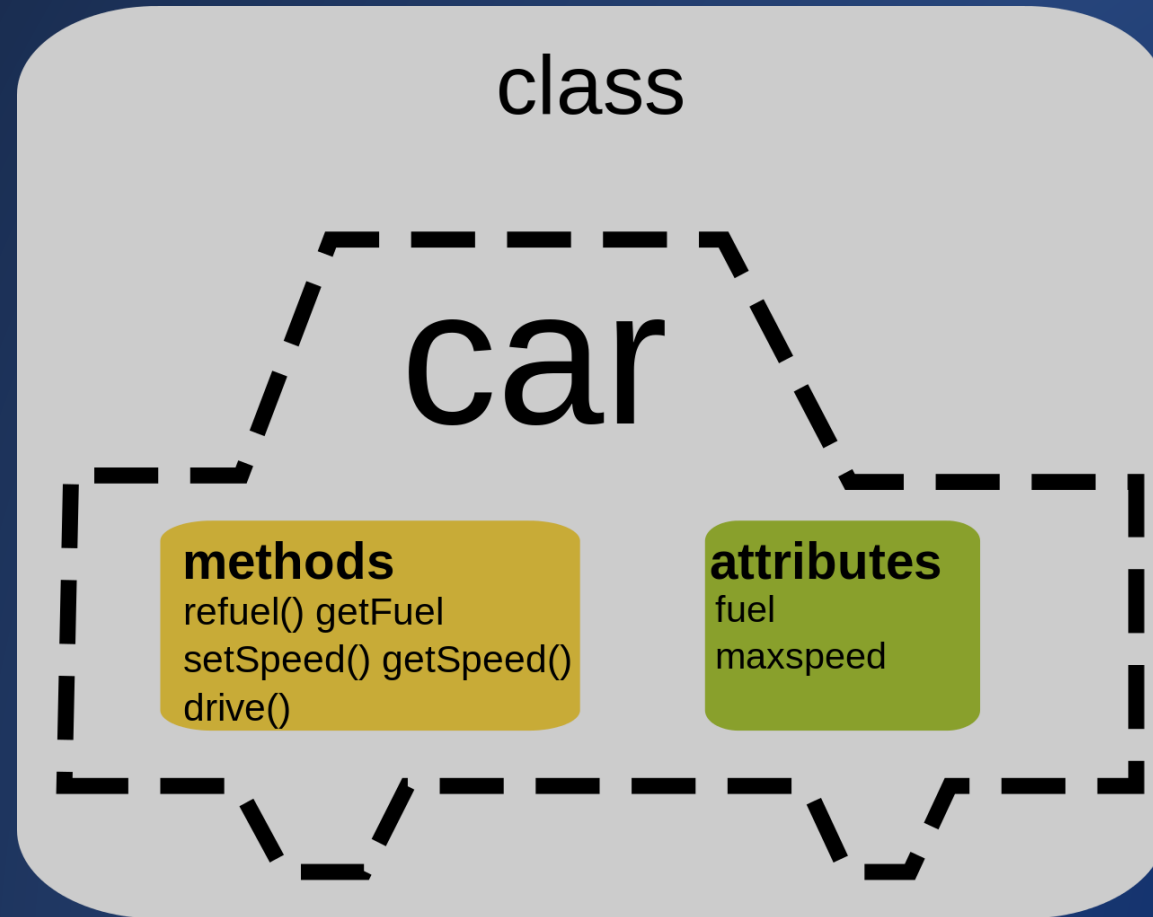


Czym jest klasa?

- Definicja dla obiektów
- Opisuje dopuszczalny stan i zachowanie obiektu
- Interfejs vs struktura – dane i metody / funkcje składowe
- Metody statyczne



Klasa

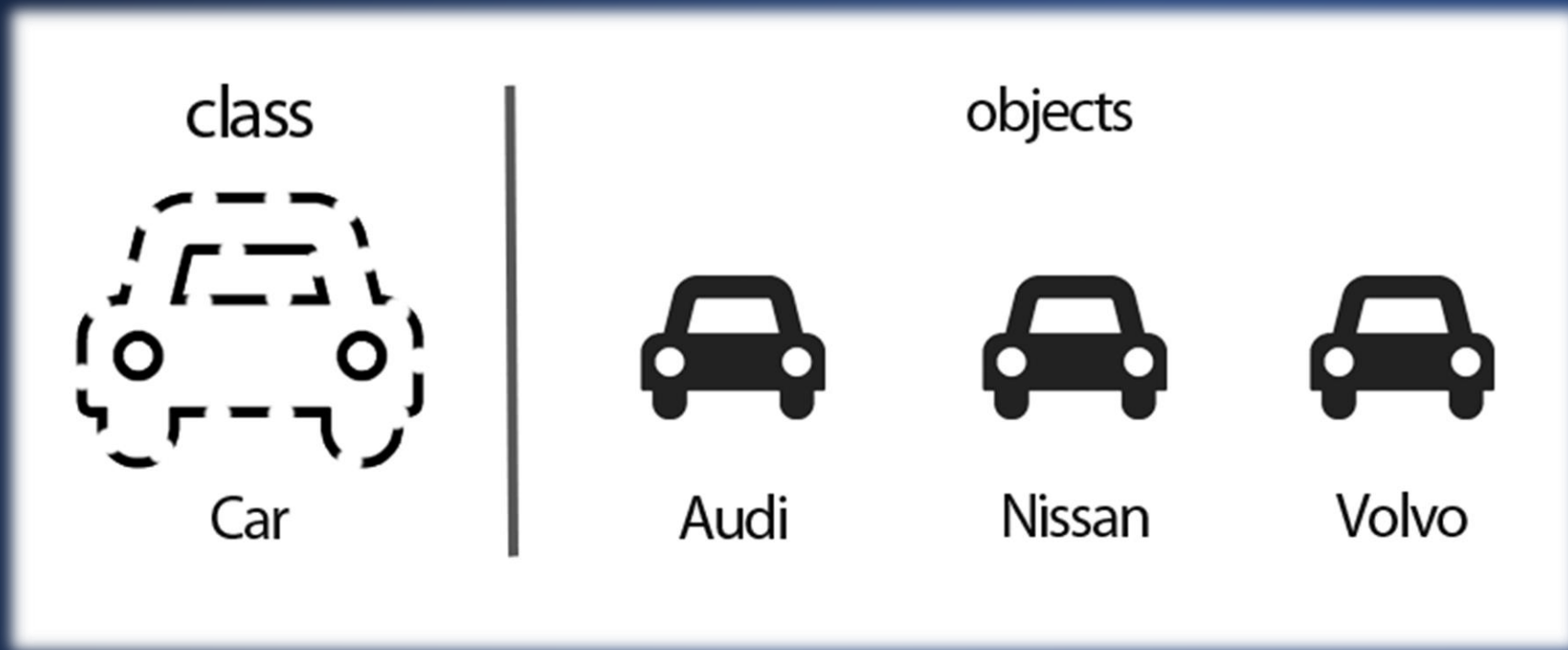


A czym zatem jest obiekt

- Instancją klasy
- Posiada swoje dane – „stan” bezpośrednio związany z tym jednym konkretnym obiektem
- Można na nim wywoływać metody niestatyczne



Obiekt vs klasa



Czterech jeźdźców apokalipsy



Cztery założenia OOP

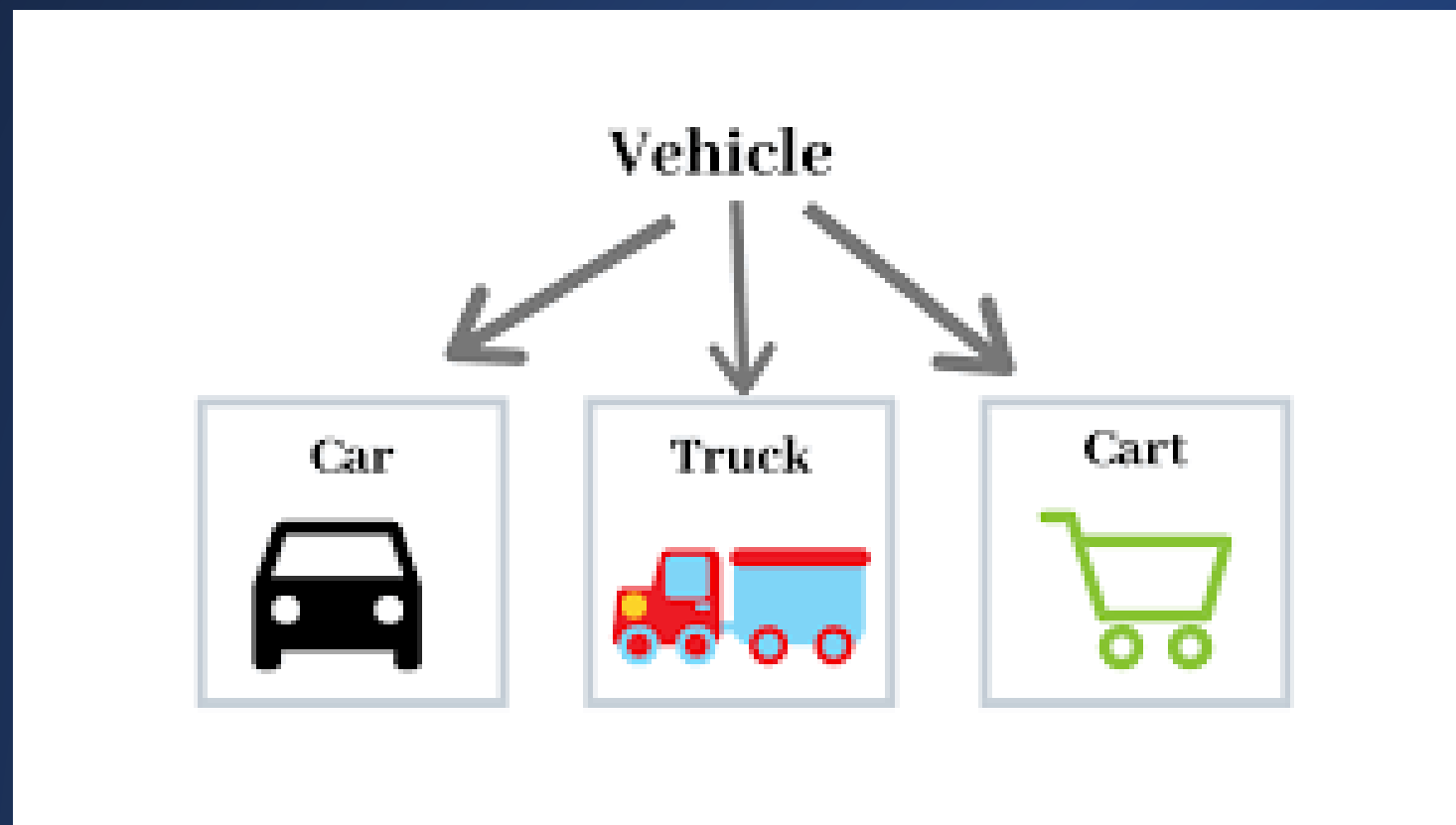
- Abstrakcja
- Enkapsulacja
- Dziedziczenie
- Polimorfizm



Abstrakcja

- „Uproszczenie pewnych rzeczy poprzez usunięcie” informacji
- Od ogółu do szczegółu
- Zmniejszenie stopnia skomplikowania programu
- Ukrycie nieistotnych szczegółów





Enkapsulacja / Hermetyzacja

- Ukrycie rzeczy i zabezpieczenie przed niepowołanym dostępem
- Ukrycie danych – „private” i dostęp przez metody



Przykład pralki

- Dostęp publiczny: włącz, wybierz program, temp., wirowanie, opcje
- Dostęp prywatny: Włącz pompkę, obróć silnik w prawo, zablokuj zamek etc.

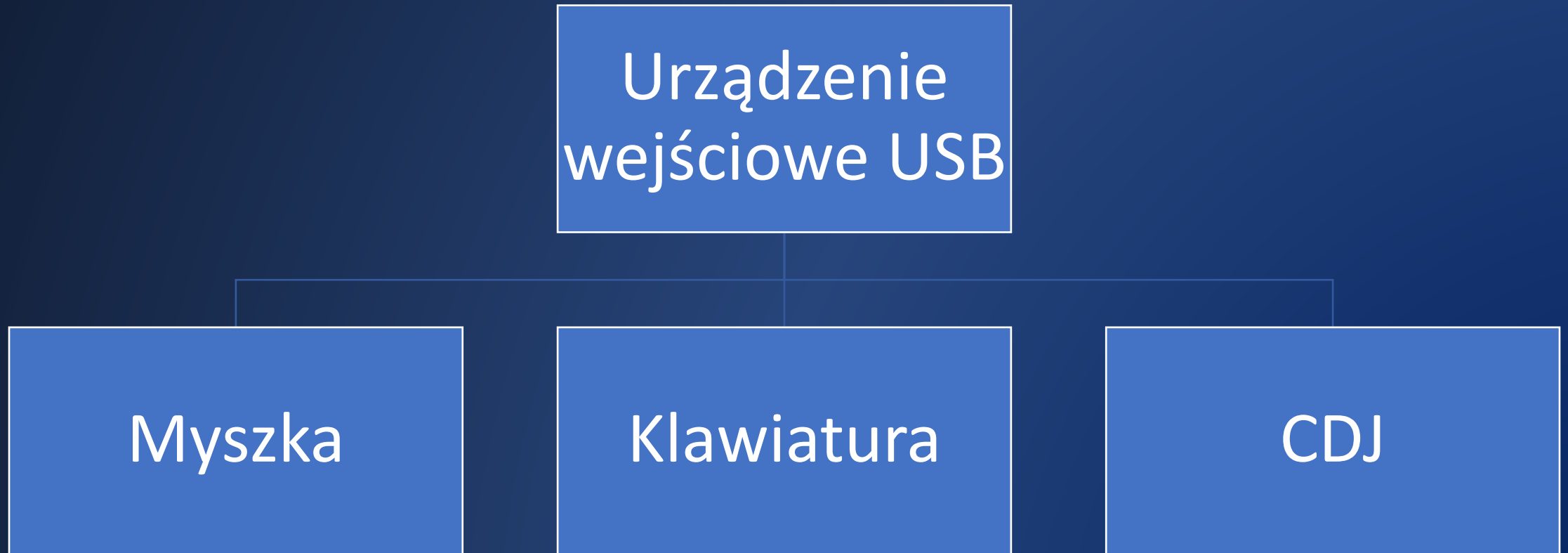


Dziedziczenie

- Podobieństwo
- Klasy bazowe i pochodne
- Cechy wspólne



Urządzenie USB



Polimorfizm

- „Wielopostaciowość”
- Metody wirtualne
- Wskaźniki



Przykład - Figura

- Figura – klasa bazowa
- Kwadrat, Koło, Trójkąt
- Metoda: pole



Jak to się robi w CPP



Zasady

- Jedna klasa – jedna para plików
- Plik nagłówkowy – Definicja klasy
- Plik Cpp – Deklaracja metod



Definiujemy pierwszą klasę

```
1  class Car
2  {
3      public:
4          Car();
5          ~Car();
6
7          void accelerate();
8          void brake();
9
10         void setRegistration(std::string number);
11         float getSpeed();
12
13     private:
14         float speed;
15         std::string registerPlate;
16         Engine engine;
17         Transmission transmission;
18
19
20 }
```

Deklarujemy metody

```
1  Car::Car()
2  {
3      speed = 0.f;
4  }
5
6  Car::~~Car()
7  {
8      //Nothing
9  }
10
11 void Car::accelerate()
12 {
13     speed += 1.f;
14 }
15
16 void Car::brake()
17 {
18     speed -= 1.f;
19 }
20
21 void Car::setRegistration(std::string number)
22 {
23     registerPlate = number;
24 }
25
26 float getSpeed()
27 {
28     return speed;
29 }
30
```

Zakresy dostępu do danych

- Public – dostęp z wewnątrz i z zewnątrz
- Private – Dostęp tylko z metod klasy
- Protected - Dostęp z metod klasy własnej i klas pochodnych
- Deklaracje przyjaźni
- Defaultowo jest private



Dane

- Mogą być dowolnego znanego typu prostego lub złożonego
- Pamiętamy o enkapsulacji - zwykle dane są prywatne
- Dostęp poprzez nazwę lub wskaźnik this:

`this->speed = 0.f;`



Metody niestatyczne

- Operują na danych klasy
- Jak klasyczne funkcje w C



Zmienne i metody statyczne

- Wywoływane są w kontekście bezobiekowym – powiązane z klasą a nie obiektem
- Dostęp odbywa się poprzez operator ::

```
Klasa::PublicznaStatyczna = 0;  
Klasa::PublicznaStatycznaMetoda();
```



Deklaracja i definicja składowych stałych

- Słowo kluczowe static w definicji
- W deklaracji już nie



```
/// <param name="lParam">The lParam.</param>
```

```
/// <returns>LRESULT (For OS)</returns>
```

```
static LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam);
```

```
/// <summary>
```

```
/// The window handle
```

```
/// </summary>
```



```

/*****
LRESULT CALLBACK Window::WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
    if (msg == WM_CREATE)
    {
        Window* pOpenGLWindow = (Window *)((LPCREATESTRUCT)lParam)->lpCreateParams;

        SetWindowLongPtr(hwnd, GWLP_USERDATA, (LONG_PTR)pOpenGLWindow);
    }

    PrE::Framework::Renderer::Window* win = hwnd ? reinterpret_cast<::PrE::Framework::Renderer::Window*>(GetWindowLongPtr(hwnd, GWLP_USERDATA)) : nullptr;

    if (win != nullptr)
    {
        win->decodeEvent(msg, wParam, lParam);

        if (win->m_callback)
            return CallWindowProcW(reinterpret_cast<WNDPROC>(win->m_callback), hwnd, msg, wParam, lParam);
    }

    if (msg == WM_CLOSE)
    {
        return 0;
    }
}

```

Konstruktor

- Funkcja wywoływana w momencie tworzenia obiektu
- Musi nazywać się tak jak klasa
- Umożliwia inicjalizację wszystkich danych
- Brak zwracanego typu danych
- Możliwość określenia wielu konstruktorów
- Konstruktor domyślny



namespace System

```
/// <summary>
/// Utility class for Text Files Input/Output
/// </summary>

class PREENGINE_FRAMEWORK FileIO
{
public:

    /// <summary>
    /// Initializes a new instance of the <see cref="FileIO"/> class.
    /// </summary>

    FileIO();
```

namespace System

```
{

    /*******
    FileIO::FileIO()
    {
    }
```

Lista inicjalizacyjna

- Skrócony sposób inicjalizacji klasy
- Konstrukcja : a(b), c(d) ...
- Zwiększenie czytelności kodu



```
namespace Renderer
```

```
{
```

```
    /***/
```

```
    StaticMesh::StaticMesh() : m_color(255, 255, 255, 255), m_texture(NULL), m_vertices(PrimitivesTypes::TrianglesStrip)
```

```
    {
```

```
    }
```


Destruktor

- Uruchamiany automatycznie przy niszczeniu obiektu
- Nazwa: `~NazwaKlasy()`
- „Sprząta”
- Brak typu zwracanego i parametrów
- W klasie może istnieć tylko jeden destruktory



```
/**/
OpenGLRenderTarget::~OpenGLRenderTarget()
{
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
    glBindVertexArray(0);
}
```

Inicjalizacja elementów statycznych

```
/// <summary>  
/// The LInGArch instance  
/// </summary>  
  
static std::shared_ptr<LInGArch> m_lingarch;
```

```
{  
    /**/  
    std::shared_ptr<LInGArch> LInGArch::m_lingarch = nullptr;
```



Przestrzenie nazw

- Dodatkowy zakres
- Organizacja kodu – brak konfliktów nazw
- Operator przestrzeni nazwy ::
- Dyrektywa using





```
7
8 namespace PrE
9 {
10     namespace Framework
11     {
12         namespace Renderer
13         {
14             namespace DirectX11
15             {
16
```



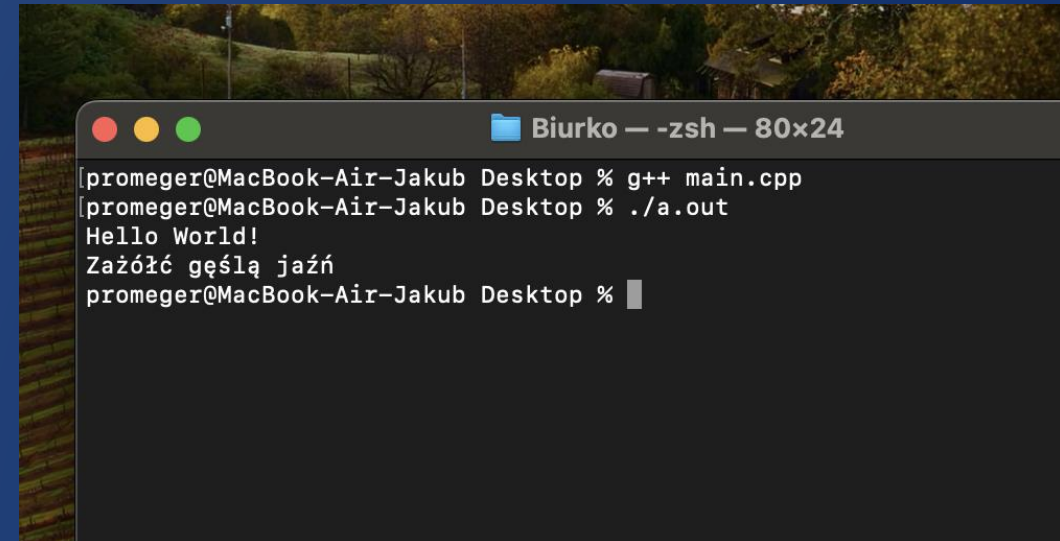
Biblioteka iostream

- Strumienie – koncept w C++
- cin cout cerr clog
- wcin wcout wcerr wclog
- Operatory << i >>
- Przestrzeń nazw std::
- Większe możliwości



Users > promeger > Desktop >  main.cpp >  main()

```
1  #include <iostream>
2  #include <locale>
3
4  int main()
5  {
6      std::wcout.sync_with_stdio(false);
7      std::wcout.imbue(std::locale("pl_PL"));
8
9
10     std::cout << "Hello World!" << std::endl;
11     std::wcout << L"Zażółć gęślą jaźń" << std::endl;
12     return 0;
13 }
```



```
Bioruko — -zsh — 80x24
[promeger@MacBook-Air-Jakub Desktop % g++ main.cpp
[promeger@MacBook-Air-Jakub Desktop % ./a.out
Hello World!
Zażółć gęślą jaźń
promeger@MacBook-Air-Jakub Desktop %
```

Co po tych zajęciach?

1. Zrobić klasę
2. Zadeklarować dane i metody w tym statyczne
3. Zadeklarować konstruktor (wraz z listą inicjalizacyjną) i destruktork
4. Znać przestrzenie nazw
5. Znać iostream



Zadanka



Dziękujemy za uwagę

