

(Średnio)zaawansowane programowanie w C++

Wykład 1 - wstęp

Robert Nowak

25L

Programowanie

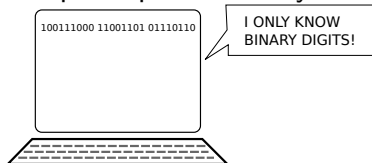
Programowanie

- ▶ umiejętność zapisu algorytmów w danym języku programowania
- ▶ umiejętność rozwiązywania problemów

umiejętność rozwiązywania problemów przy pomocy komputera

Czy problem może być rozwiązany przez komputer?

- ▶ komputer przetwarza tylko informację dostępną w formie cyfrowej



- ▶ komputer potrzebuje algorytmu (programu)
- ▶ tylko człowiek może stworzyć algorytm **silna Sztuczna Inteligencja (strong AI) nie istnieje!**

Cel i zakres przedmiotu

- ▶ zapoznanie słuchaczy z zagadnieniami tworzenia oprogramowania
- ▶ efektywne wykorzystanie języka C++
- ▶ współcześnie stosowane techniki i biblioteki
- ▶ zarządzanie zasobami,
- ▶ wykorzystywanie różnych języków programowania w tej samej aplikacji, na przykładzie C++, Pythona, Rust

Zakładana znajomość:

- ▶ podstawowych algorytmów i struktur danych
- ▶ programowania strukturalnego i obiektowego
- ▶ języka C++
- ▶ obiektowych wzorców projektowych
- ▶ czytanie diagramu klas UML

Podstawowe dane o przedmiocie

Wykład: wtorki 18¹⁵ – 20⁰⁰

Strona przedmiotu:

<https://staff.elka.pw.edu.pl/~rnowak2/dyd/zpr>

Prowadzący:

prof. dr hab. inż. Robert Nowak,

e-mail: robert.nowak@pw.edu.pl

dr inż. Łukasz Neumann

mgr. inż. Witold Wysota

Konsultacje: stacjonarnie lub online, patrz <http://repo.pw.edu.pl>

Tematyka wykładów

- ▶ Programowanie obiektowe w C++, polimorfizm, wzorce projektowe (krótki przegląd),
- ▶ zarządzanie zasobami, sprytne wskaźniki, mechanizm wyjątków,
- ▶ szablony, programowanie generyczne,
- ▶ stałość, `std::optional`, `std::variant`,
- ▶ biblioteka standardowa, rozszerzenia biblioteki standardowej - **boost**,
- ▶ Aplikacje współbieżne (wielowątkowe), synchronizacja
- ▶ Rust, łączenie C++ i Pythona

Literatura

- ▶ Gamma et al., *Wzorce projektowe*, WNT, 2005.
- ▶ Alexandrescu, *Nowoczesne projektowanie w C++*, 2005.
- ▶ Alexandrescu, Sutter, *Język C++. Standardy kodowania. 101 zasad, wytycznych i zalecanych praktyk*, Helion, 2005.
- ▶ Meyers, *50 efektywnych sposobów na udoskonalenie Twoich programów*, Helion, 2003.
- ▶ Stroustrup, *Programowanie. Teoria i praktyka z wykorzystaniem C++*, Helion 2010.
- ▶ Nowak, Pająk, *Język C++: mechanizmy, wzorce, biblioteki*, BTC, 2010.
- ▶ Hunt, Thomas, *Pragmatyczny programista*, Helion, 2011
- ▶ Software Engineering Body of Knowledge (SWEBOK) v3, IEEE Computer Society, 2014

Zaliczenie przedmiotu

kolokwium 1	0 – 10pkt
kolokwium 2	0 – 10pkt
laboratorium	0 – 20pkt
suma	40 pkt

37 – 40 pkt.	ocena 5
33 – 36	4.5
29 – 32	4
25 – 28	3.5
21 – 24	3

Projekt:

propozycje tematów	1, 5 marca
ustalenie składu zespołów, przydział zadań	11, 14 marca
dokumentacja wstępna, max. 3 strony A4	do 25 marca
oddanie szkieletu aplikacji	do 25 kwietnia
dostarczenie implementacji i dokumentacji	do 3 czerwca
projekty nie są przyjmowane po	10 czerwca

Zadanie dodatkowe, dla chętnych:

rezerwacja: 15 kwietnia, wykonanie: 1 czerwca.

Nie jest wymagana obecność na wykładzie.

Wymaganie oprogramowanie

- ▶ kompilatory:
 - ▶ the GNU Compiler Collection 12 lub nowszy
 - ▶ Microsoft Visual Studio 2019 lub nowszy
- ▶ biblioteki:
 - ▶ stl
 - ▶ boost - <http://www.boost.org>
- ▶ inne:
 - ▶ repozytorium (np. git)
 - ▶ edytor tekstu (np. emacs)
 - ▶ debugger (np. gdb)
 - ▶ narzędzie do automatycznej kompilacji (np. SCons)
 - ▶ narzędzie do generowania dokumentacji (np. doxygen)
 - ▶ optymalizacja kodu, profiler (np. gprof)
 - ▶ narzędzie do wirtualizacji (np. kvm)

Pewne fakty związane z tworzeniem oprogramowania

Większość projektów informatycznych kończy się niepowodzeniem.

*R. Glass, Frequently Forgotten Facts about Software Engineering, 2001

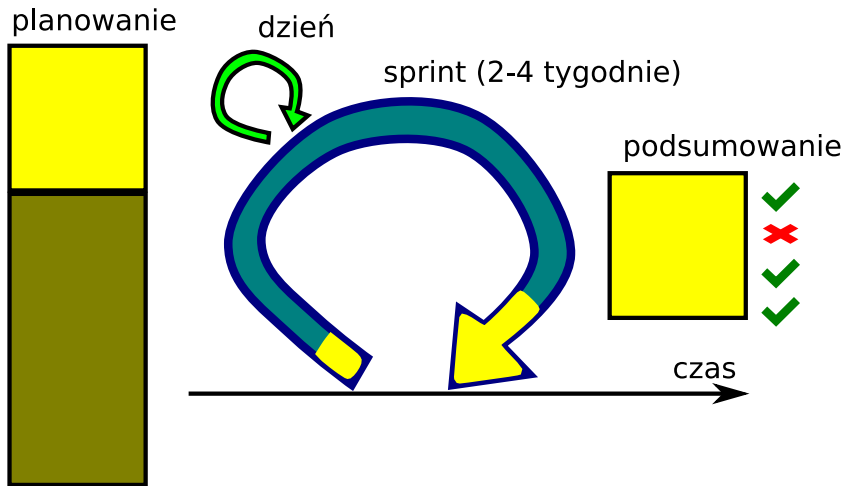
Przyczyny:

- ▶ niestabilne wymagania
- ▶ optymistyczna estymacja kosztów, głównie czasu
- ▶ niska jakość oprogramowania

Sterowanie projektem:

- ▶ koszt
- ▶ czas
- ▶ jakość
- ▶ zakres

Cykl tworzenia oprogramowania - SCRUM



Metodologie lekkie (extreme programming, SCRUM)

Właściwości:

- ▶ krótki czas sprzężenia zwrotnego
- ▶ mniejsze ryzyko niepowodzenia projektu

Możliwe gdy koszt zmian w projekcie (względem czasu) nie rośnie wykładniczo

- ▶ kod źródłowy dobrej jakości, stosowanie narzędzi, unikanie redundancji
- ▶ działający prototyp
- ▶ odpowiednie zarządzanie, w tym komunikacja z odbiorcą

Zmniejszanie kosztów zmian - kod źródłowy

Kody źródłowe dobrej jakości przez cały czas trwania projektu.

- ▶ czytelny kod, KISS (Keep It Simple Software), standardy kodowania
- ▶ zmiana struktury programu bez zmiany funkcjonalności (refactoring)
- ▶ wzorce projektowe

Komentarze - poprawiają czytelność kodu.

Komentarz mówi DLACZEGO, kod mówi JAK.

- ▶ należy opisywać odpowiedzialność modułów (plików), klas, metod
- ▶ każdy byt powinien mieć pojedynczą odpowiedzialność
- ▶ należy opisywać niebanalne algorytmy
- ▶ dokumentacja projektowa powinna być generowana z kodu

Zmniejszenie kosztów zmian - działający prototyp

Utrzymywanie działającej wersji przez cały czas trwania projektu.

- ▶ testowanie automatyczne:
 - ▶ jednostkowe
 - ▶ podczas implementacji - zapewnienie pokrycie kodu testami
 - ▶ jeżeli znajdzie się błąd, to należy napisać test, który go wychwyci
 - ▶ funkcjonalne
- ▶ ciągła integracja

Zmniejszanie kosztów zmian - zarządzanie

- ▶ definicja tego co to znaczy, że zadanie zostało wykonane (projekt, implementacja, testy jednostkowe, testy funkcjonalne, wdrożenie)
- ▶ jawne sterowanie projektem
 - ▶ jawna lista zadań
 - ▶ kolektywne prawo do zmian kodu (narzędzia zarządzające wersjami)
 - ▶ kolektywne szacowanie złożoności zadań
- ▶ nastawienie na zmiany
 - ▶ planowanie czasu pracy
 - ▶ dyskusja z użytkownikiem lub właścicielem biznesowym
 - ▶ podsumowania, wyciąganie wniosków

Zmniejszanie kosztów zmian - wsparcie odbiorcy oprogramowania

Zaangażowanie odbiorcy oprogramowania:

- ▶ pozwala uniknąć tworzenia rozwiązań, które nie będą używane
- ▶ dostarcza dodatkową kontrolę poprawności

Aby zrozumieć najważniejsze wymagania, programiści powinni pracować w organizacji, która zleciła utworzenie oprogramowania, jako użytkownik aplikacji, przez 1-2 tygodnie. ^a

^aHunt, Thomas, The Pragmatic Programmer, Addison-Wesley, 2000

PRINCE2 - projekty w sterowalnym środowisku

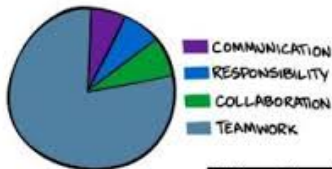
kierownictwo



- zarządzanie strategiczne: odpowiedzialne za sukces projektu, m.in. ciągła zasadność biznesowa
- zarządzanie operacyjne: delegowanie zadań, monitorowanie, alokacja zasobów

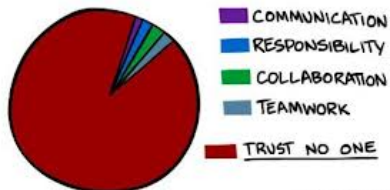
Po co prowadzi się projekty zespołowe na studiach

WHAT GROUP PROJECTS ARE
SUPPOSED TO TEACH YOU



Endless Origami

WHAT GROUP PROJECTS TAUGHT ME



endlessorigami.com

*<http://endlessorigami.com>

Popularność języków programowania

Wybór odpowiedniego języka programowania

Miara popularności:

- ▶ liczba linii kodu napisanych w danym języku (w oprogramowaniu które jest obecnie wykorzystywane)
- ▶ liczba odwołań do stron opisujących dany język / liczba sprzedanych książek opisujących dany język
- ▶ liczba ofert pracy dla programistów danego języka programowania

Wszelkie miary są przybliżone, ponieważ:

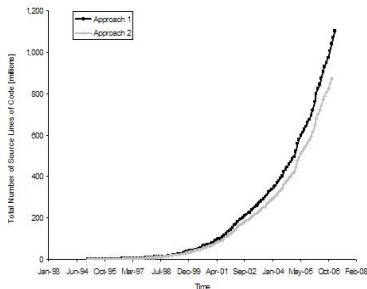
- ▶ nie uwzględnia komercyjnego (przemysłowego) kodu źródłowego
- ▶ liczba odwołań do opisu języka może świadczyć o jego złożoności (a nie tylko popularności)
- ▶ może oznaczać mobilność programistów

Liczba linii dla wybranych projektów

nazwa	LOC ¹	języki programowania
Linux Kernel	12M	C(11M) Assembler (250k)
MySql	12.5M	C++(7.1M) C(3.8M)
Firefox	9M	C++(3.7M) C(1.8M) JavaScript (1.4M)

Próba oszacowania wielkości oprogramowania

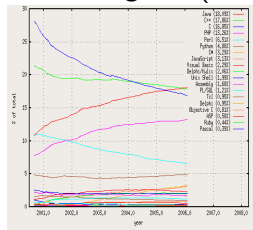
- ▶ biznesowe (Gartner Group, 1997): $310 * 10^9$ LOC (COBOL 60%)
- ▶ szacunki 2008 rok: $10 * 10^{12}$ LOC
- ▶ Open Source, 2008 rok: $1.2 * 10^9$ LOC



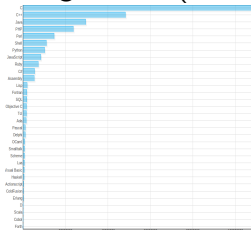
¹Lines of Code, <http://www.ohloh.net>

Oprogramowanie Open Source przy podziale na języki

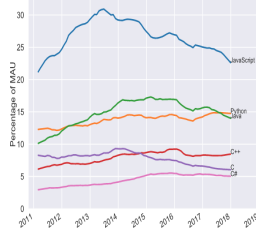
sourceforge (2007)



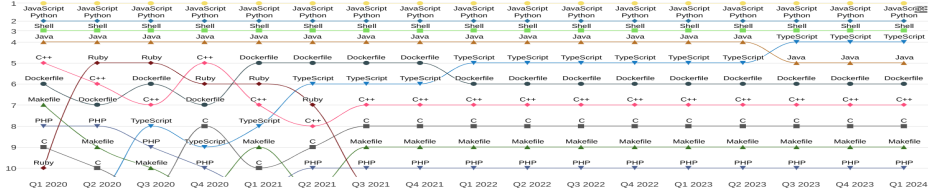
Google code (2012)



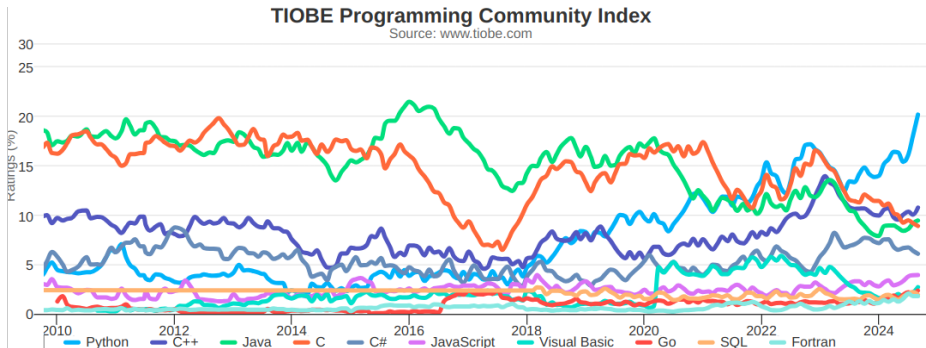
GitHub 2019



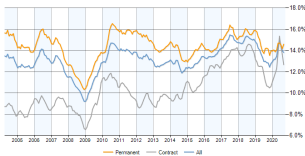
GitHub 2024



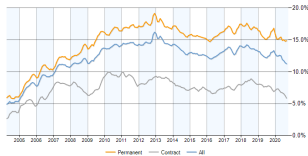
Zliczanie trafień dla stron internetowych



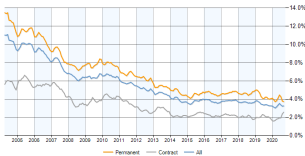
Oferty pracy <http://www.itjobswatch.co.uk>



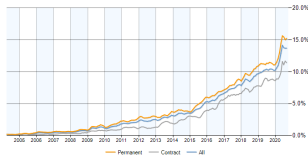
Java



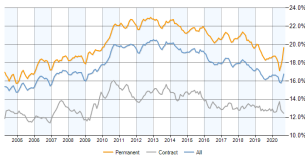
C#



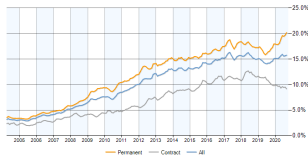
C++



Python



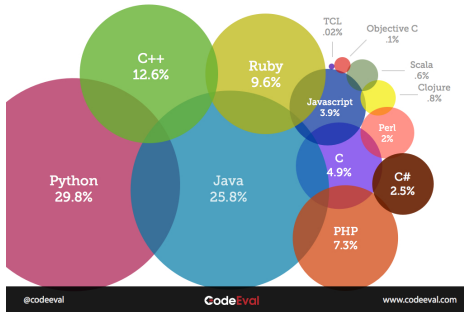
SQL



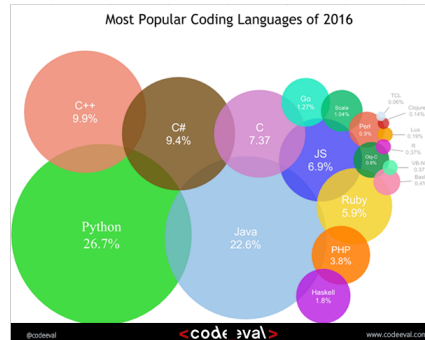
JavaScript

Ankiety wśród pracodawców <http://blog.codeeval.com>

Most Popular Coding Languages of 2013

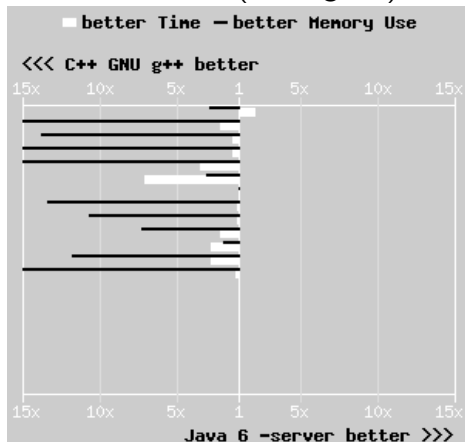


Most Popular Coding Languages of 2016



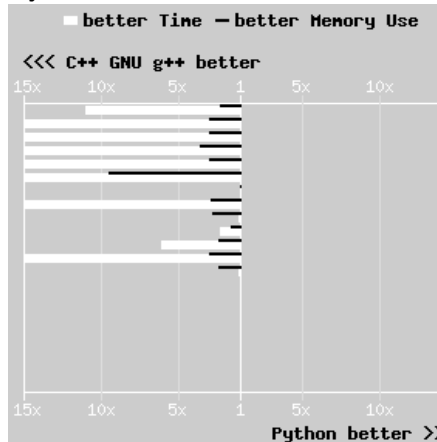
Wydajność (The computer language benchmarks game)

Java kontra C++ (GNU g++)



x64 Ubuntu, Intel Q6600 (quad-core)

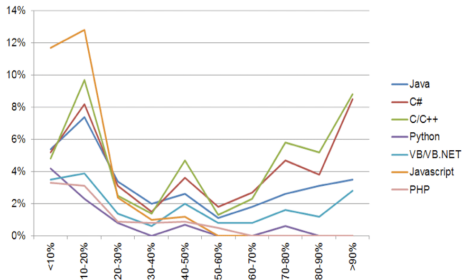
Python kontra C++



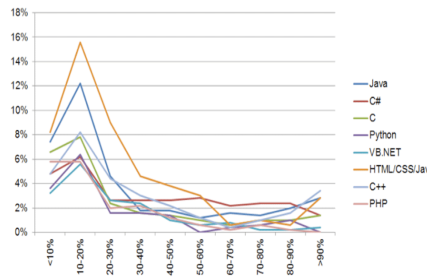
Zmiana akcentów w tworzeniu programowania

Programiści w ciągu ostatnich 2 lat zaczęli masowo używać więcej niż jednego języka programowania jednocześnie (The Quiet Revolution in Programming, By Andrew Binstock, April 03, 2013, Dr.Dobb's Journal)

2010



2012



Zalety i wady języka C++

- + standard języka jest tworzony przez organizację standaryzacyjną ANSI/ISO (nie przez firmę)
 - formalny (długi) proces standaryzacji (C++11, C++14, C++17, C++20, C++23)
- + język kompilowany do kodu maszynowego, możliwość tworzenia bardzo wydajnych aplikacji
- + wspiera wiele paradygmatów: programowanie strukturalne, obiektowe, generyczne
 - złożony, wiele poprawnych konstrukcji do rozwiązywania tego samego problemu
- + duża popularność, dostępność wielu narzędzi dla wielu platform (w tym wbudowanych)
- + duża stabilność
 - uboga biblioteka standardowa

Języki programowania

Kompromis:

- ▶ jednoznaczny (komputer)
- ▶ pojęcia bliskie programiście (człowiek)

Translacja: tłumaczenie z postaci źródłowej (zrozumiała dla człowieka) do wynikowej (zrozumiała dla komputera)

- ▶ kompilacja
- ▶ interpretacja

Teza

- ▶ nie istnieje jeden, najlepszy język programowania
- ▶ stosowanie zawsze jednego języka programowania - niezbyt wyrafinowane rozwiązania

Potrzeba użycia różnych języków w aplikacjach

System komputerowy zawsze:

- ▶ ma ograniczenia czasowe, więc tworzenie całości powinno być możliwie szybkie
- ▶ posiada pewne elementy, które są „wąskim gardłem” - powinny być zaimplementowane wydajnie (20% kodu)

System komputerowy często:

- ▶ posiada pewne elementy, których autor nie chce udostępniać (kod ukryty przed użytkownikiem)
- ▶ posiada pewne fragmenty, które powinny być dostępne dla użytkownika (aby dostosować aplikację do indywidualnych potrzeb, np. konfiguracja)

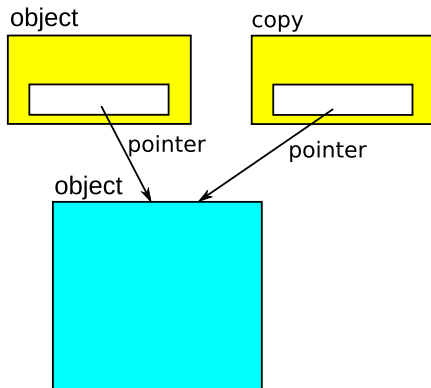
Wzorce projektowe

- ▶ standardowe rozwiązania często pojawiających się problemów projektowych
- ▶ sprawdzone w praktyce
- ▶ najczęściej dotyczą programowania obiektowego
- ▶ znajomość wzorców projektowych pozwala lepiej zrozumieć obiektowe podejście do programowania

Przykład - prototyp

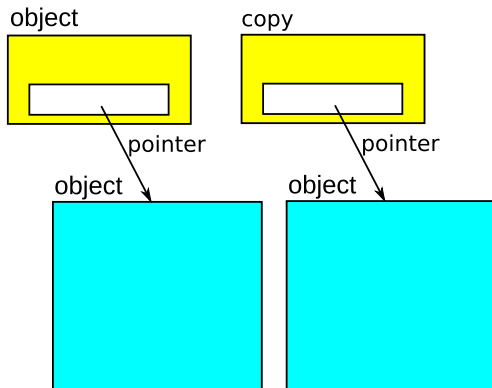
płytki kopia (shell copy)

```
Obj* obj = new Obj();  
Obj* copy = obj;
```



głęboka kopia (deep copy)

```
Obj* obj = new Obj();  
Obj* copy = new Obj(*obj);
```

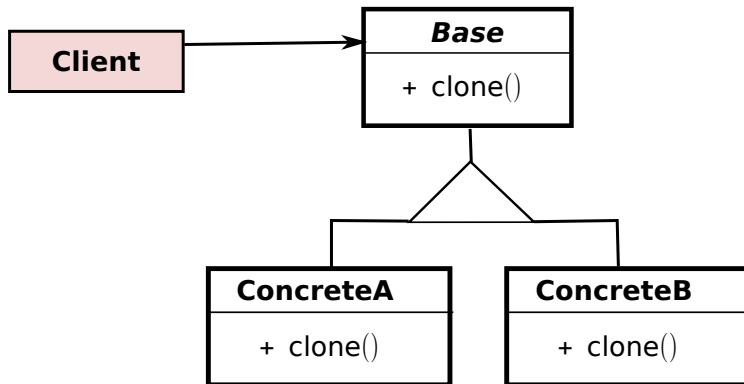


„głęboka” i „płytką” kopia

```
class B {};  
class D1 : public B {};  
class D2 : public B {};  
  
vector<B*> v; v.push_back(new D1); v.push_back(new D2);  
  
vector<B*> u = v; //Płytką kopia  
  
vector<B*> uu;  
for(const B* b : v)  
    uu.push_back(new B(*b)); //WYCINANIE!
```

Wzorzec prototypu

- ▶ odpowiedzialność przeniesiona na obiekty pochodne
- ▶ wykorzystanie mechanizmu funkcji wirtualnej



prototyp (clone, wirtualny konstruktor) - przykład

```
class B {
public:
    virtual B* clone() const = 0; //tworzy kopie danego obiektu
    B(const B&) = delete; //Zabroniony konstruktor kopiujący
};

class D1 : public B {
public:
    D1(const D1& r); //Konstruktor kopiujący
    B* clone() const override {
        return new D1(*this); //Już wie, jaki typ kopiować!
    }
};

vector<B*> v1, v2; //v2 będzie głęboką kopią v1
for(const B* b : v1) v2.push_back( b->clone() );
```

zalecenia końcowe

- ▶ Czytać kod innych.
- ▶ Programować.

Dziękuję

robert.nowak@pw.edu.pl