



UNIWERSYTET
EKONOMICZNY
we Wrocławiu

TECHNOLOGII .NET– WYKŁAD 14



PARADYGMATY I TYPOLOGIA JĘZYKÓW. PODSUMOWANIE

DR RADOSŁAW WÓJTOWICZ

PARADYGMAT PROGRAMOWANIA

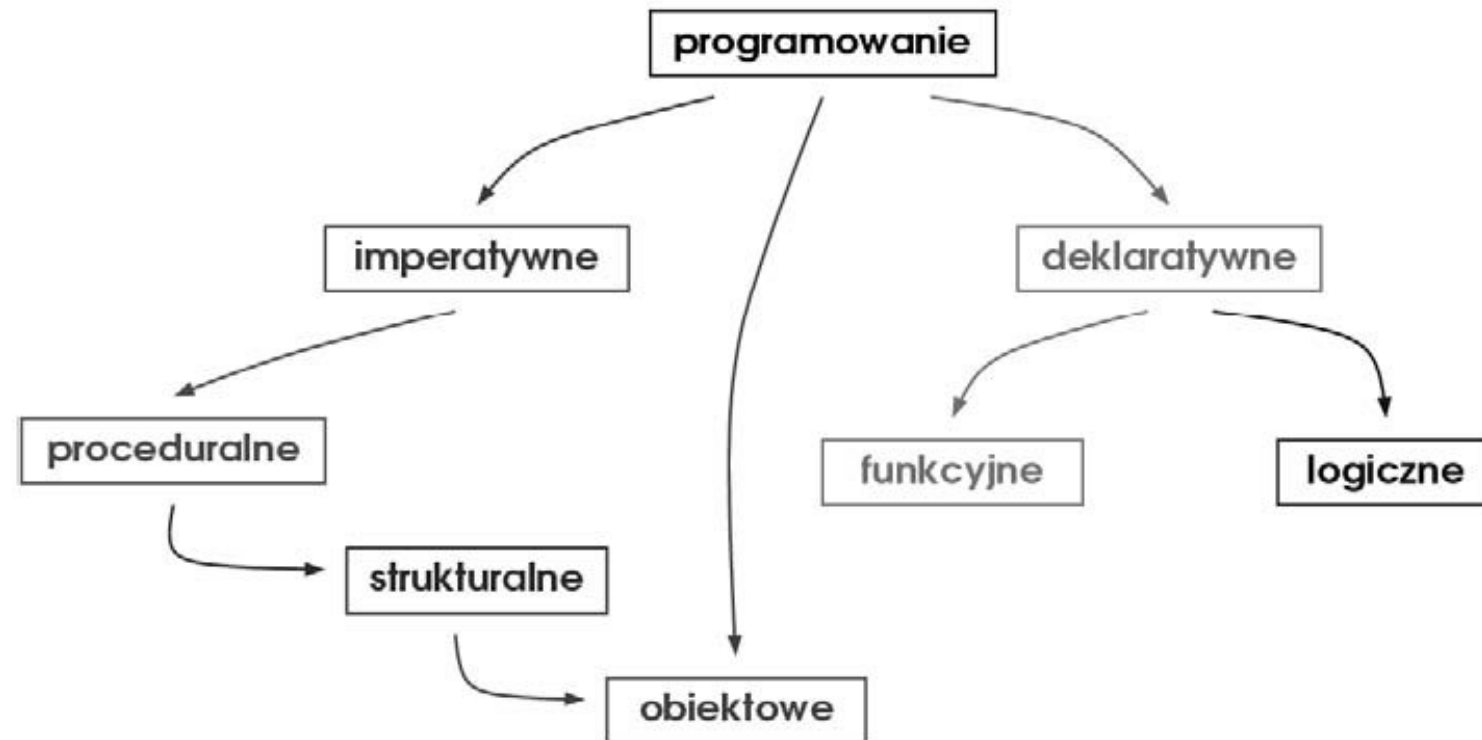


Definiuje sposób patrzenia programisty na przepływ sterowania i wykonywanie programu komputerowego.



To ogół oczekiwań programisty wobec języka programowania i komputera, na którym będzie działał program.

Paradygmaty programowania



PARADYGMATY PROGRAMOWANIA

■ Programowanie imperatywne:

- **opisywane są kolejne czynności**, które komputer (program) ma wykonać, by osiągnąć cel;
- program specyfikuje **w jaki sposób** maszyna ma działać;
- programista opisuje, **jak komputer ma działać**.

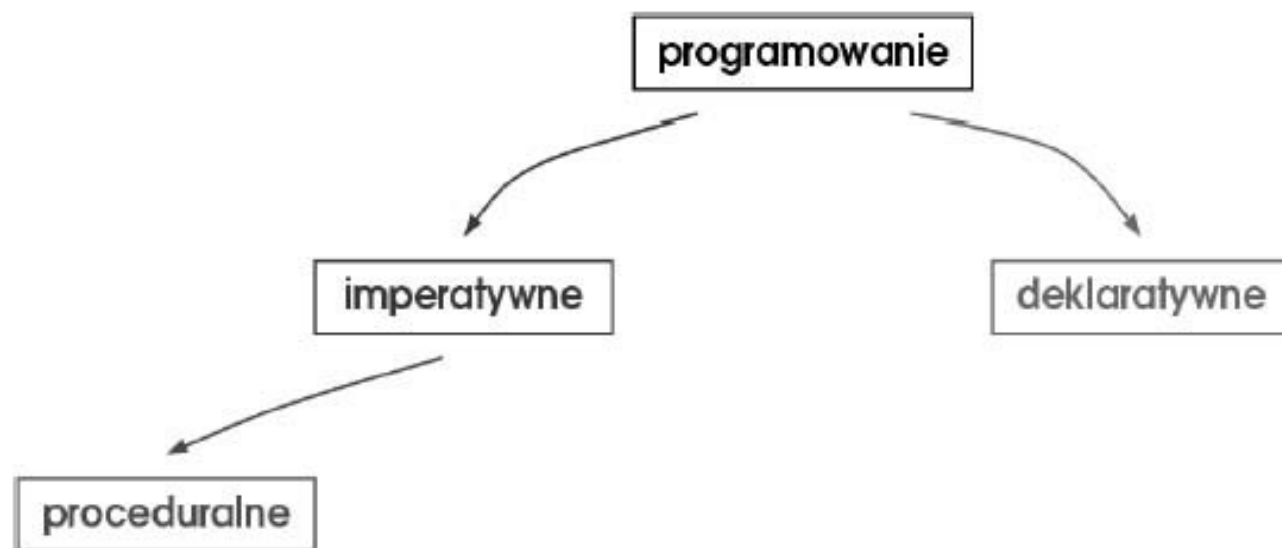
■ Programowanie deklaratywne

- **opisuje się cel**, który komputer (program) ma osiągnąć;
- program specyfikuje **wynik**, jaki maszyna ma uzyskać, ale nie określa sposobu uzyskania wyniku;
- programista opisuje, **co komputer ma osiągnąć**.

PROGRAMOWANIE IMPERATYWNE

- Programy imperatywne (najbardziej pierwotny sposób programowania) – gdzie program postrzegany jest jako ciąg poleceń dla komputera
- **Instrukcje podstawienia (przypisania)** działają na danych pobranych z pamięci i umieszczają wynik w tejże pamięci, zaś abstrakcją komórek pamięci są zmienne.
- **Instrukcje pętli** umożliwiają wielokrotne wykonanie tego samego kodu - w zależności od potrzeb, "wielokrotność" może oznaczać pewną określoną z góry ilość powtórzeń lub wykonywanie do czasu spełnienia pewnych warunków.
- **Instrukcje warunkowe** wykonują pewien blok kodu tylko wtedy, kiedy spełniony jest określony warunek. W przeciwnym razie blok ten jest pomijany podczas wykonywania.
- **Przekazanie sterowania** do zupełnie innej części programu. Realizowane jest to poprzez bezwarunkowy skok zwany „goto” oraz wywołanie podprogramu (procedury bądź funkcji).
- Programowanie imperatywne uważane jest często za synonim programowania proceduralnego.

Paradygmaty programowania



PROGRAMOWANI E PROCEDURALNE

- Podział skomplikowanych programów na odrębne moduły – podprogramy (procedury, funkcje) lub makroinstrukcje (makrodefinicje)
- Struktura:
 - grupa podprogramów,
 - program główny.
- Program = algorytmy + struktury danych
- Elementy:
 - stała, zmienna, typ stałej lub zmiennej, wyrażenie, argument, operator, relacja,
 - instrukcje (prosta, złożona, przypisania, iteracyjna, warunkowa, wyboru, we/wy),
 - procedury, funkcje, program, makroinstrukcje.

OGÓLNA POSTAĆ DEFINICJI PROCEDURY – C++

```
void nazwaProcedury ( typ_1 nazArg_1, ..., typ_n nazArg_N)  
{  
    blok instrukcji;  
}
```

Wywołanie procedury:

```
nazwaProcedury (arg_1, ..., arg_N);
```


OGÓLNA POSTAĆ DEFINICJI FUNKCJI - C++

```
typZwracanejWartości nazwaFunkcji ( typ_1 nazArg_1, ..., typ_n nazArg_N)  
  
{  
  
    blok instrukcji;  
  
}
```

Wywołanie funkcji:

```
nazwaZmiennej = nazwaFunkcji (arg_1, ..., arg_N);
```

```
typZwracanejWartości = typFunkcji
```

PROCEDURY I FUNKCJE – ZASTOSOWANIE

Ukrycie nieistotne szczegółów wykonania pewnych operacji

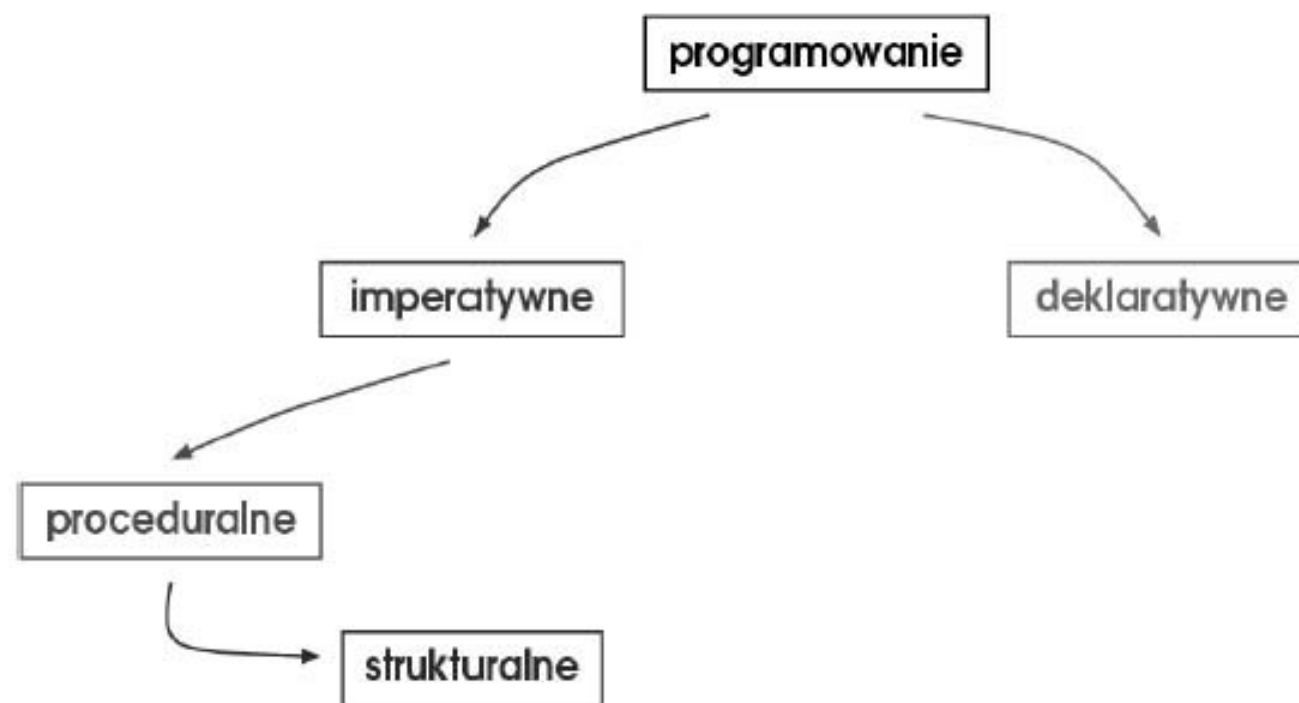
Poprawienie czytelności programu przez przeniesienie fragmentów stanowiących wyodrębnioną całość

Zlikwidowanie wielokrotnego powtarzania tych samych fragmentów kodu opisujących często występujące czynności

ARGUMENTY PROCEDUR/FUNKCJI

- Argumenty pozwalają nadać uniwersalny charakter podprogramom
- Ten sam podprogram wykonuje te same czynności dla różnych wartości argumentów
- Argumenty mogą być przekazywane przez wartość lub zmienną

Paradygmaty programowania



```

program sortowanie;
const zakres = 99;
var
    Liczby:array [1..zakres] of integer;
    i,j,k,n:integer;

procedure czytaj;
begin
    writeln('Program sortuje dane metoda babelkowa');
    write('Podaj ilosc liczb: '); read(n);
    for i:=1 to n do begin
        write('Liczba ',i,' = '); read (Liczby[i]);
    end;
end;

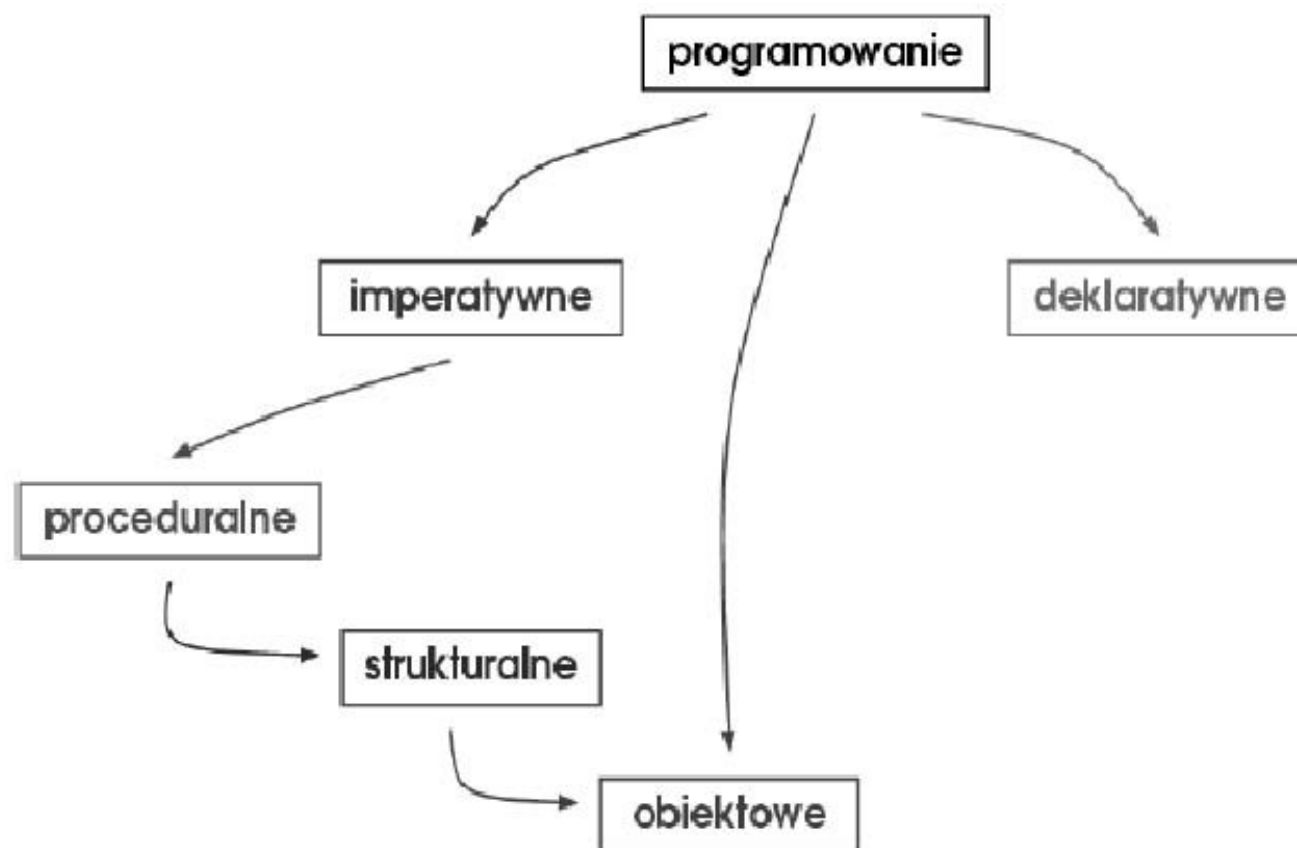
procedure pisz;
begin
    writeln('Oto posegregowane elementy:');
    for i:=1 to n do write(liczby[i],' ');
    readln;
end;

begin
    czytaj;
    for i:=2 to n do begin
        for j:=n downto i do begin
            if Liczby[j-1] > Liczby[j] then begin
                k:=liczby[j-1];
                Liczby[j-1]:=Liczby[j];
                Liczby[j]:=k;
            end;
        end;
    end;
    pisz;
    readln;
end.

```

PROGRAM SORTUJĄCY TABLICĘ (PASCAL)

Paradygmaty programowania



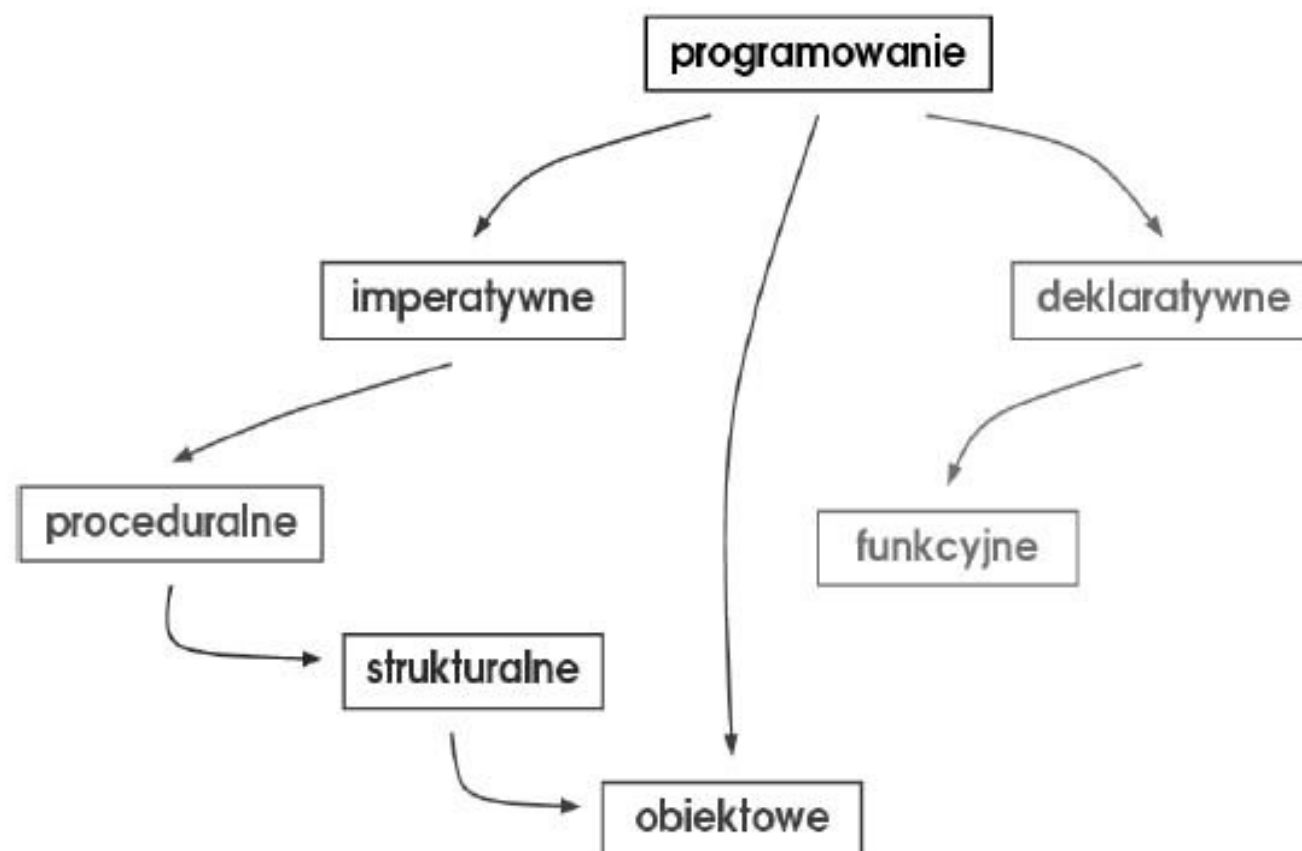
PROGRAMOWANIE OBJEKTOWE

- Programowanie obiektowe (ang. *object-oriented programming*) — metodyka tworzenia programów komputerowych, która definiuje programy za pomocą obiektów — elementów łączących *stan* (czyli dane) i *zachowanie* (metody).
- Program jest zbiorem wielu niezależnych komponentów (obektów), które komunikują się ze sobą za pomocą przesyłanych komunikatów
- Program = obiekty + obiekty
 - Obiekt = dane + metody
- Cechy obiektów:
 - dziedziczenie
 - polimorfizm
 - hermetyczność
 - enkapsulacja

JĘZYK PROGRAMOWANIA: PROCEDURALNEGO A OBIEKTOWEGO

- **Język programowania proceduralnego**
(inaczej język algorytmiczny):
 - to typ języka programowania, w którym programista, tworząc kod programu, nie operuje na obiektach, lecz tworzy kod programu linijka po linijce, pisząc procedury i funkcje.
- **Język programowania obiektowego:**
 - to typ języka programowania, w którym programista tworzy moduły programowe (obiekty zawierające dane i procedury manipulowania tymi danymi), a kod programu tworzy przez operowanie obiektami i tworzenie powiązań między nimi.

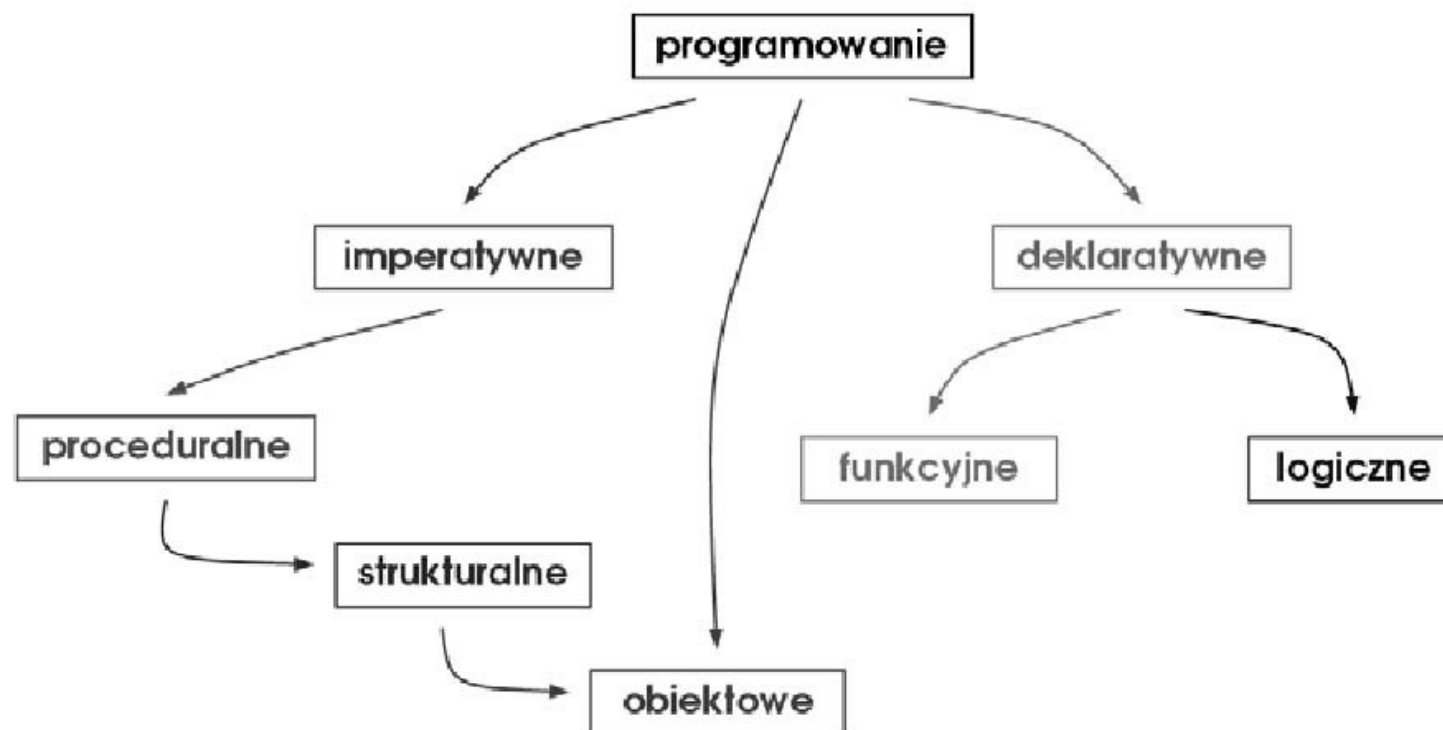
Paradygmaty programowania



PROGRAMOWANI E FUNKCYJNE (FUNKCJONALNE)

- Program to po prostu złożona funkcja (w sensie matematycznym), która otrzymawszy dane wejściowe wylicza pewien wynik.
- Nie ma zmiennych.
- Nie ma tradycyjnie rozumianych pętli (imperatywnych z natury) .
- Konstruowanie programów to składanie funkcji.
- Zazwyczaj z istotnym wykorzystaniem rekurencji (rekursji) (tam, gdzie w programowaniu imperatywnym wykorzystujemy pętle) .
- Podział języków funkcyjnych:
 - Języki czysto funkcyjne – np. Haskell
 - Języki mieszane - przykładowe języki:
 - Lisp (*język programowania stosowany do symulowania ludzkiej inteligencji*);
 - Nemerle- hybrydowy język programowania; zawiera elementy programowania funkcyjnego i obiektowego.

Paradygmaty programowania



PROGRAMOWANIE LOGICZNE

- Program jest zbiorem reguł definiujących wyrażenia logiczne.
- Wykonanie programu to zadanie wyszukania pewnych kombinacji reguł, dla których opisujące je wyrażenia logiczne są prawdziwe.
- Na program składa się zbiór zależności (przesłanek) oraz pewne stwierdzenie/pytanie (cel).
- Wykonanie programu to próba udowodnienia celu bazując na podanych przesłankach.
- Przykładowy język: Prolog. Zapoczątkował programowanie oparte na logicznych regułach.

JĘZYK PROGRAMOWANI A

- Pozwala na precyzyjny zapis algorytmów oraz innych zadań, które ma wykonać komputer.
- Służy do dialogu pomiędzy programistą a komputerem.
- Stanowi zbiór zasad.
- Składa się ze zbiorów reguł syntaktycznych oraz semantycznych, które opisują, jak należy budować poprawne wyrażenia oraz jak komputer ma je rozumieć.
- Pierwszym językiem był **kod binarny**, składający się z ciągu rozkazów w postaci zer i jedynek.

JĘZYKI PROGRAMOWANIA – PODZIAŁ

Wewnętrzne (maszynowe) – języki zrozumiałe dla komputera program binarny (wynikowy) składający się z listy rozkazów komputera;

Symboliczne (zestawiające; assembly) – powstają przez parametryzację niektórych części instrukcji w języku wewnętrznym (najczęściej adresów) języki adresów symbolicznych

Zewnętrzne (wyższego rzędu) – języki najbardziej zbliżone do naturalnego dla człowieka zapisu algorytmu program źródłowy (niezależny od realizacji maszynowej)

TYPOLOGIA JĘZYKÓW PROGRAMOWANIA

Generacja języka

Paradygmat programowania

Sposób wykonywania (kompilacja,
interpretacja)

Sposób kontroli typów

Przeznaczenie

TYPOLOGIA JĘZYKÓW PROGRAMOWANIA

Ze względu na generację wyróżniamy języki należące do generacji:

- ▮ Pierwszej
- ▮ Drugiej
- ▮ Trzeciej
- ▮ Czwartej
- ▮ Piątej

GENERACJA PIERWSZA

- Języki maszynowe - inaczej: języki procesorów; języki wewnętrzne; języki binarne
- Program (kod maszynowy) stanowi binarny zapis (ciąg zer i jedynek) funkcji mikroprocesora wraz z ich parametrami.
- Każdy procesor ma swój język wewnętrzny.
- Wady:
 - trudny w użyciu,
 - łatwo o błędy,
 - trudna modyfikacja.

```
1000010110101010100010101011101010011010
1010101010101010010101010010101010010101
0101011111111111010101101111111100000010
1010101010101010101010101010010101001010
```

GENERACJA DRUGA

- Języki niskiego poziomu (zwane assemblerami lub symbolicznymi)
- Ze względu na składnię są identyczne z językami maszynowymi, z tą różnicą, że zamiast liczb używa się mnemoników.
- Składnia programu - krótkie i proste polecenia, będące dokładnymi odpowiednikami procedur, które może wykonać procesor.
- Ułatwienia:
 - mnemoniczne nazwy operacji,
 - operowanie nazwami a nie adresami,
 - możliwość swobodnego (prostego) modyfikowania programu.
- Każdy assembler jest związany z danym typem procesora.

```
mov bx, 12
mov ax, 10
add ax, bx
mov [3987], 20
int 32
```

GENERACJA TRZECIA

- Języki wysokiego poziomu – kod zbliżony do języka naturalnego (angielskiego) i matematycznego.
- Języki jedynie są zrozumiałe dla człowieka - trzeba korzystać ze specjalnych programów tłumaczących zwanych translatorami (są to interpretery, kompilatory i konsolidatory).
- Języki te są już w mniejszym stopniu zależne od konkretnego typu komputera (procesora). Są one ukierunkowane na problem, a nie na procesor.
- Najliczniejsza grupa języków. Pierwszym językiem tego typu był Algol. Inne przykładowe to: Fortan, Cobol, Pascal, Visual Basic, C, C++, C# Java.

GENERACJA TRZECIA – JĘZYKI WYSOKIEGO POZIOMU

- język imperatywny — C

```
int dodaj()  
{ int a = 2;  
  int b = 2;  
  return(a + b);}   
  
int l = dodaj();
```

- język aplikatywny — Lisp

```
(defun dodaj()           ;aplikatywnie  
  ((lambda (x y) (+ x y)) 2 2))  
(defun dodaj()           ;imperatywnie  
  (let (x y)(setq x 2)(setq y 2)(+ x y)))  
  
(dodaj)
```

- język deklaratywny — Prolog

```
dodaj(X) :- X is 2 + 2.  
  
dodaj(X).
```

- język symboliczny — Mathematica

```
dodaj := Module[{a = 2, b = 2}, a + b];  
l = dodaj;
```

- język obiektowy — C++

```
class CDodaj  
{ public:  
  int a;  
  int b;  
  CDodaj()  
  { a = 2;  
    b = 2;};  
  int Dodaj()  
  { return(a + b);}  
};  
  
/*statycznie*/  
Cdodaj d;  
int l = d.Dodaj();  
  
/*dynamicznie*/  
Cdodaj *d = new CDodaj;  
int l = d->Dodaj();
```

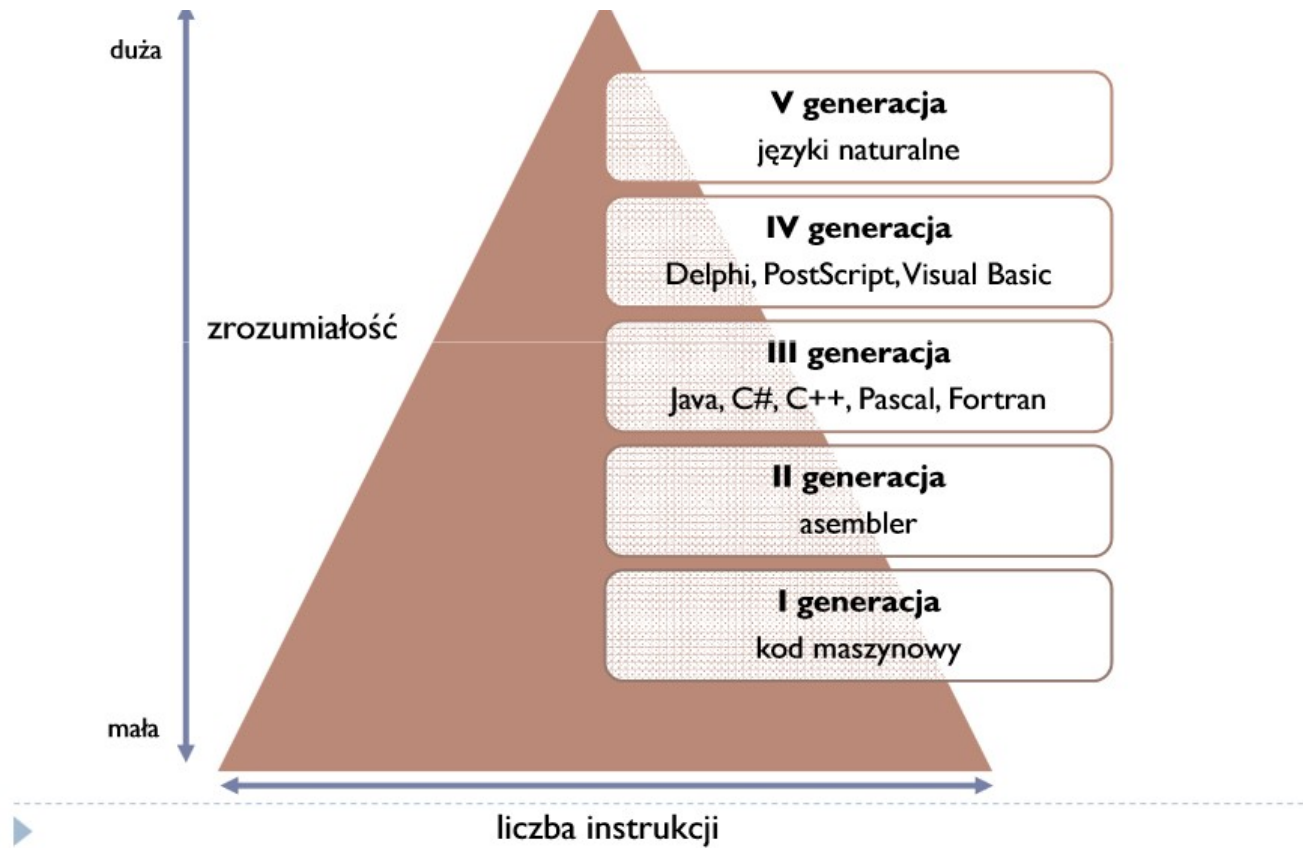
GENERACJA CZWARTA

To środowiska graficzne generowania gotowych aplikacji na podstawie predefiniowanych modułów.

Rozszerzają funkcjonalność istniejących języków

Czwartą generacją określa się również języki programowania obiektowego

GENERACJE JĘZYKÓW - PODSUMOWANI E





ASSEMBLER

- Pierwszego assemblera zaprojektował Konrad Zuse.
- Język, w którym jedno polecenie odpowiada jednemu rozkazowi procesora.
- Powstał na bazie języka maszynowego, w którym kody maszynowe zostały zastąpione ich mnemonicznymi odpowiednikami, co umożliwia pisanie programów w miarę zrozumiałych dla człowieka.
- Jest to język niskiego poziomu, obecnie służy głównie do pisania sterowników lub elementów programów wymagających wysokiej wydajności.

FORTRAN (FORMULA TRANSLATOR)

- Język programowania wysokiego poziomu stworzony w roku 1954.
- Pierwszy kompilator języka Fortran stworzył zespół Johna Backusa.
- Pierwsza standaryzacja języka odbyła się w roku 1960, kiedy to standard języka opisano jako Fortran VI. Kolejnym jego standardem był Fortran 66, a potem Fortran 77. Dwa następne standardy to Fortran 90 i Fortran 95.
- Język ten nadal jest stosowany m. in. przy obliczeniach aerodynamicznych, cieplnych oraz wytrzymałościowych.
- Jedną z przyczyn, dla których Fortran jest ciągle w użyciu, jest jego szybkość obliczeń oraz wysoka jakość kodu, co z kolei jest spowodowane jego długą obecnością na rynku programistycznym.
- Obecnie tworzone są kompilatory umożliwiające stosowanie grafiki.



PASCAL

- Nazwa języka pochodzi od nazwiska francuskiego fizyka Blaise'a Pascala.
- Język programowania wysokiego poziomu, stworzony przez Niklausa Wirtha w oparciu o język Algol w roku 1971. Został opublikowany pierwszy raport będący szczegółowym, formalnym opisem Pascala.
- Od tego czasu powstało wiele modyfikacji języka Pascal: powstanie wielu nowych standardowych funkcji i procedur (m.in. procedur programowania grafiki), powstanie i rozwój systemów pozwalających na redagowanie programów (tzw. edytory) oraz ich uruchamianie i testowanie; środowiska te stają się coraz bardziej przyjazne i coraz bogatsze (również o rozwiązania programowania obiektowego - Object Pascal).

VISUAL BASIC

- To język programowania wysokiego poziomu i narzędzie programowania firmy Microsoft.
- Zawiera kilkaset instrukcji, funkcji i słów kluczowych. Nie jest językiem w pełni obiektowym, gdyż nie udostępnia np. możliwości dziedziczenia, czy polimorfizmu.
- Visual Basic for Applications (VBA) – język programowania oparty na Visual Basicu (VB) zaimplementowany w aplikacjach pakietu Microsoft Office
- Wraz z pojawieniem się platformy .NET, ukazała się nowa wersja Visual Basica pod nazwą Visual Basic .NET (obiektywny język programowania).



C

- Język stworzony przez Dennisa Ritchie do programowania systemów operacyjnych oraz innych zadań niskiego poziomu.
- Po roku 1980 stał się dominującym językiem do programowania systemów operacyjnych oraz aplikacji.
- Na jego bazie powstał język C++.

Twórca systemu operacyjnego UNIX



C++

- Jest to obiektowy język programowania wysokiego poziomu, zaprojektowany przez Bjarne Stroustrupa w latach osiemdziesiątych na bazie języka C.
- Nazwa języka powstała w roku 1983, została zaproponowana przez Wcześniej używano nazwy „C z klasami”.
- W języku C++ pojawiły się m. in.:
klasy, szablony funkcji oraz klas, obsługa wyjątków, deklaracje zmiennych jako instrukcje.



C#











- Obiektowy język programowania zaprojektowany przez zespół pod kierunkiem Andersa Hejlsberga dla firmy Microsoft.
- Język C# ma wiele cech wspólnych z językami programowania Object Pascal, Delphi, C++ i Java.
- Rok powstania 2000r.



JAVA

- Obiektowy język programowania, stworzony przez grupę roboczą firmy Sun Microsystems pod nadzorem Jamesa Goslinga.
- Jest to język interpretowany, silnie typowany, podstawowe koncepcje zostały zaczerpnięte z języka Smalltalk (wirtualna maszyna) oraz C++ (składnia, słowa kluczowe).
- Java jest niezależna od systemu operacyjnego komputera oraz platformy (Windows, Linux) dzięki zastosowaniu wirtualnej maszyny, która tłumaczy uniwersalny, skompilowany kod pośredni na kod dostosowany do specyfiki konkretnego systemu operacyjnego.
- Rok powstania – 1995.

NAJPOPULARNIEJSZE JĘZYKI WG TIOBE INDEX

May 2024	May 2023	Change	Programming Language		Ratings	Change
1	1			Python	16.33%	+2.88%
2	2			C	9.98%	-3.37%
3	4	⬆		C++	9.53%	-2.43%
4	3	⬇		Java	8.69%	-3.53%
5	5			C#	6.49%	-0.94%
6	7	⬆		JavaScript	3.01%	+0.57%
7	6	⬇		Visual Basic	2.01%	-1.83%
8	12	⬆		Go	1.60%	+0.61%
9	9			SQL	1.44%	-0.03%
10	19	⬆		Fortran	1.24%	+0.46%

Źródło: www.tiobe.com

TRENDY ROZWOJOWE: NO-CODE I LOW-CODE, AI

Rosnąca potrzeba tworzenia aplikacji, w zestawieniu ze zbyt niską w stosunku do popytu dostępnością programistów, owocuje coraz większą ilością narzędzi, których zadaniem jest umożliwienie budowania rozwiązań przy mocnym ograniczeniu lub wręcz wyeliminowaniu konieczności programowania.

Platformy no-code adresowane są przede wszystkim do użytkowników biznesowych, natomiast low-code – do programistów, którzy chcą zaoszczędzić czas i obniżyć koszty realizacji danego projektu.

Platformy low-code wymagają od użytkownika podstawowej wiedzy na temat programowania, natomiast no-code to rozwiązania w 100 proc. oparte na interfejsach „przeciągnij i upuść”, zupełnie niewymagające wiedzy programistycznej.

PORÓWNANIE

No-code vs low-code – porównanie

	Zalety	Wady
Platformy no-code	Łatwa w użyciu • Umożliwia szybkie tworzenie aplikacji i przepływów pracy • Niemal wszystkie funkcje wprowadzamy metodą „przeciągnij i upuść” • Może być przydatna zarówno dla programistów, jak i zwykłych użytkowników • Nie wymaga znajomości j. programowania • Doskonała dla firm, które nie mają zespołu IT i/lub budżetu na programistów zewnętrznych	Nie można tworzyć bardziej złożonych, rozbudowanych aplikacji • Nie dostarcza pełnej elastyczności i możliwości, której mogą potrzebować programiści • Brak pełnej kontroli w zakresie bezpieczeństwa IT • Ograniczone możliwości w zakresie integracji z aplikacjami firm trzecich
Platformy low-code	• Znacznie przyspiesza realizację projektów dla programistów • Zapewnia elastyczność - można wprowadzać niestandardowe zmiany programistyczne • Duże możliwości w zakresie integracji z aplikacjami firm trzecich	• Trudniejsza w obsłudze niż no-code i wymaga znajomości j. programowania (podstawy to minimum) • Zaprojektowana głównie dla programistów (aby przyspieszyć ich pracę), a nie dla zwykłych użytkowników • Brak pełnej kontroli w zakresie bezpieczeństwa IT

PLATFORMA BUBBLE

<https://bubble.io/>

https://www.youtube.com/watch?v=07_9tqDhTks

JAK AI POMAGA PROGRAMIŚCIE W PRACY?

- **Zaawansowane autouzupełnianie**
- **Pisanie testów**
- **Tłumaczenie błędów i propozycje poprawek**
- **Transformacja danych na wygodny do wyświetlenia format**
- **Wyjaśnianie problemów z kodem (help)**
- **Przepisywanie kodu z jednego języka na inny**

PRZYKŁAD – GITHUB COPILOT W MS VISUAL STUDIO

- https://www.youtube.com/watch?v=kc_A12G4Elk

EGZAMIN – PRZEDTERMIN, TERMIN PIERWSZY I DRUGI

- **UWAGA:** Egzamin odbywa się wyłącznie stacjonarnie na terenie Uczelni na komputerach/terminalach dostępnych w salach laboratoryjnych. Do egzaminu w przedterminie mogą przystąpić wszyscy chętni. Do egzaminu w terminie I mogą przystąpić tylko studenci, którzy uzyskali zaliczenie laboratorium.
- **Przedtermin (termin zerowy):**
19.06.2024 (środa), godz. 13.00, lab. 104Z (gr. 02), godz. 15.15, lab. 104Z (gr. 01)
- **Termin 1:**
2.07.2024 (wtorek), godz. 10.00, lab. 003Z1 (gr. 01 i 02)
- **Termin II:**
10.07.2024 (środa), godz. 10.00, lab. 003Z1 (gr. 01 i 02)
- Test jednokrotnego wyboru na ePortalu, 30 pytań
- Czas na wykonanie: 30 min.



DZIĘKUJĘ
ZA UWAGĘ!

RADOSLAW.WOJTOWICZ@UE.WROC.PL