

Modelowanie i Programowanie Obiektowe



POLITECHNIKA
POZNAŃSKA

Wykład I: Wstęp
20 października 2012

Programowanie obiektowe

- Metodyka wytwarzania oprogramowania

Metodyka

Metodyka – ustandaryzowane dla wybranego obszaru podejście do rozwiązywania problemów. Metodyka abstrahuje od merytorycznego kontekstu danego obszaru, a skupia się na metodach realizacji zadań.

W odróżnieniu od metodologii, która się skupia na odpowiedzi na pytanie:

Co należy robić?

metodyka koncentruje się na poszukiwaniu odpowiedzi na pytanie:

Jak to należy robić?

- Program to zbiór elementów zwanych obiektami
- Obiekty łączą stan (dane) i zachowanie (procedury, metody)
- Obiekty komunikują się ze sobą w celu wykonania określonych przez programistę zadań

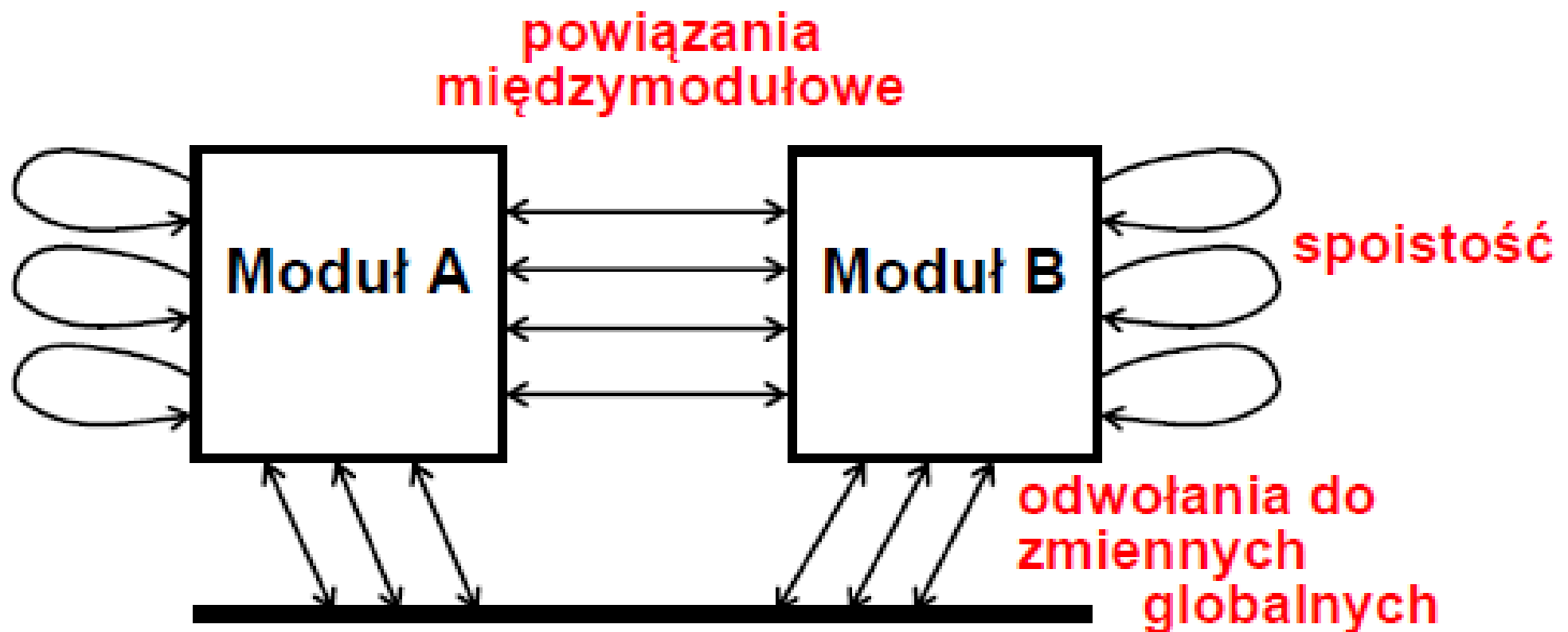
Programowanie obiektowe, a rzeczywistość

- Arystoteles analizując otaczającą go rzeczywistość wprowadził pojęcie **formy** i **materii**
 - Odrzucenie idei dualizmu platońskiego (**materia** i **idea**)
 - Forma nadaje kształt **materii**, a nie jest bytem niezależnym
- Forma = Klasa
- Materia = Instancja = Obiekt
- „Jest to najbardziej naturalny sposób rozumienia rzeczywistości - podstawową cechą mózgu ludzkiego jest klasyfikacja -

Architektura programów komputerowych

Dobra, a zła konstrukcja programów komputerowych

- liczne i niejasne zależności z innymi modułami
- odporność architektury na modyfikacje



Motywacje paradygmatu obiektowego

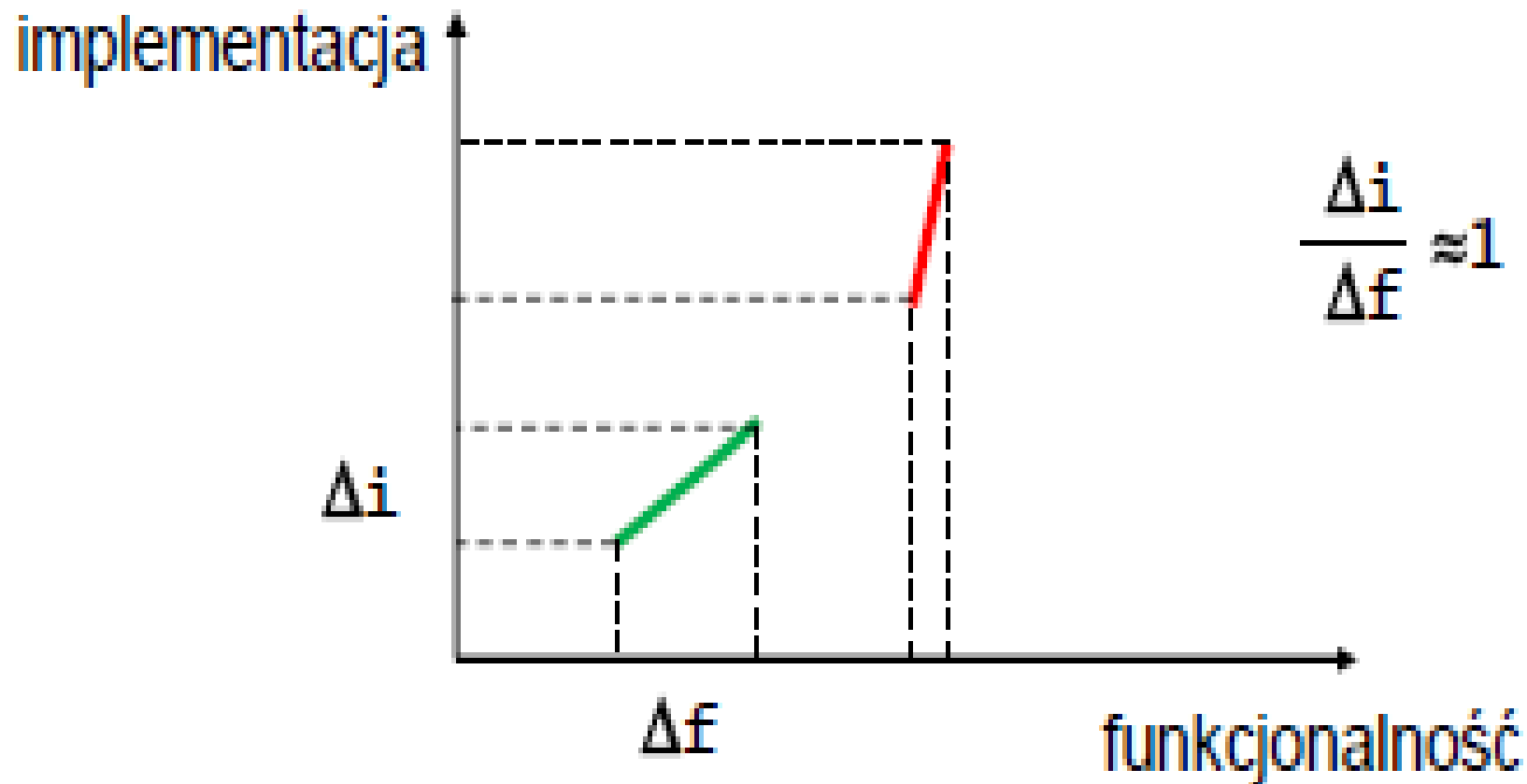
- Bardzo duża trudność implementacji złożonych systemów informatycznych
 - Niska jakość kodu (niewydajny, niebezpieczny, niestabilny)
 - Niezrozumiały kod (*ang. spaghetti code*)
 - Ogromne koszty utrzymania systemów informatycznych
 - Wielokrotna implementacja tej samej funkcjonalności
 - Krótki czas życia
- Słabość języków programowania
 - Mały zbiór prostych predefiniowanych typów
 - Rozwój aplikacji poprzez budowę i wykorzystanie typów

Motywacje paradygmatu obiektowego

- Programowanie obiektowe zbliża programy do „naszego” sposobu postrzegania rzeczywistości
- Zmniejsza *lukę reprezentacji*
 - łatwiej „zapanować” nad kodem
 - łatwiej zrozumieć co ktoś inny miał na myśli...
 - łatwiejsza współpraca z innymi
 - łatwiejsze wyobrażenie problemu i jego reprezentacji

Motywacje paradygmatu obiektowego

Zależność między modyfikacjami funkcjonalnymi i implementacyjnymi



Paradygmat proceduralny vs. obiektowy

Proceduralny

- Świat modelowany jako algorytm
- Problemy = proces dekompozycji funkcjonalnej
- Dane mają znaczenie drugorzędne
- Program = zbiór procedur implementujących algorytm
- Modułami architektury programów są procedury
- Hierarchia funkcji
- Model przepływu danych

Obiektowy

- Świat modelowany jako zbiór obiektów
- Problem = hierarchia specyfikacji klas obiektów (abstrakcyjnych typów danych)
- Znaczenie algorytmów jest drugorzędne
- Program = zbiór klas implementujących operacje abstrakcyjnych typów danych
- Modułami architektury są obiekty
- Zbiór powiązanych, kooperujących obiektów
- Model przepływu danych

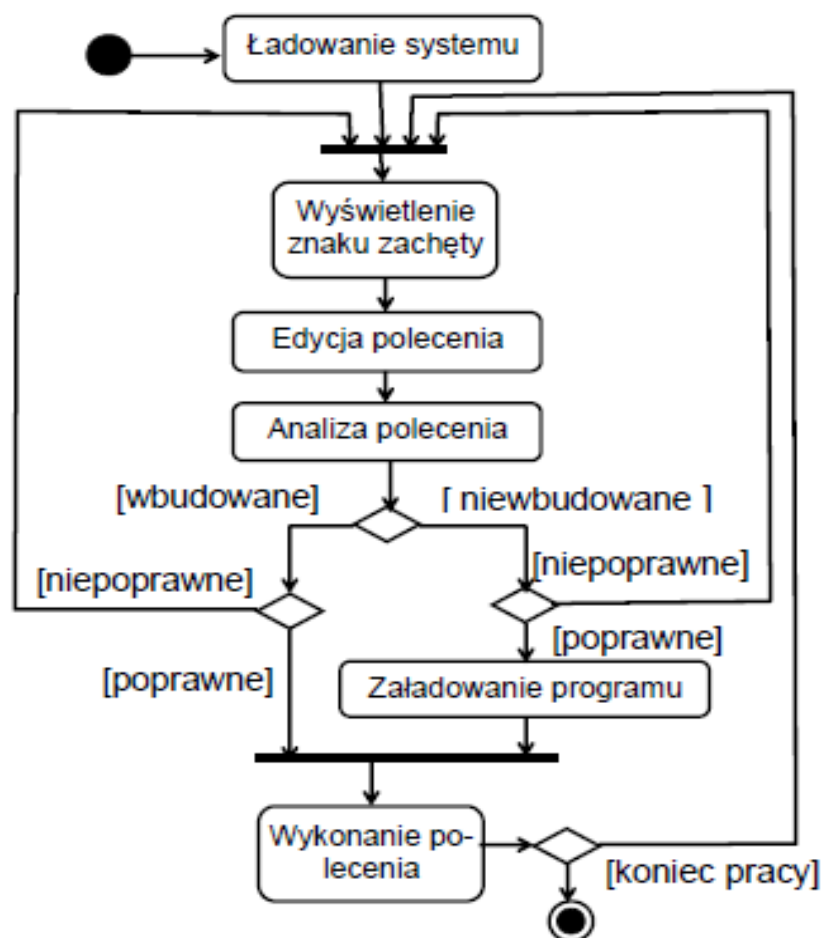
Co daje paradygmat obiektowy?

- W konstrukcji złożonych systemów pozwala tworzyć aplikacje o wyższej jakości:
 - Niski koszt utrzymania
 - Łatwa modyfikacja systemu
 - Re-używalność
 - Czytelniejszy kod

Dekompozycja

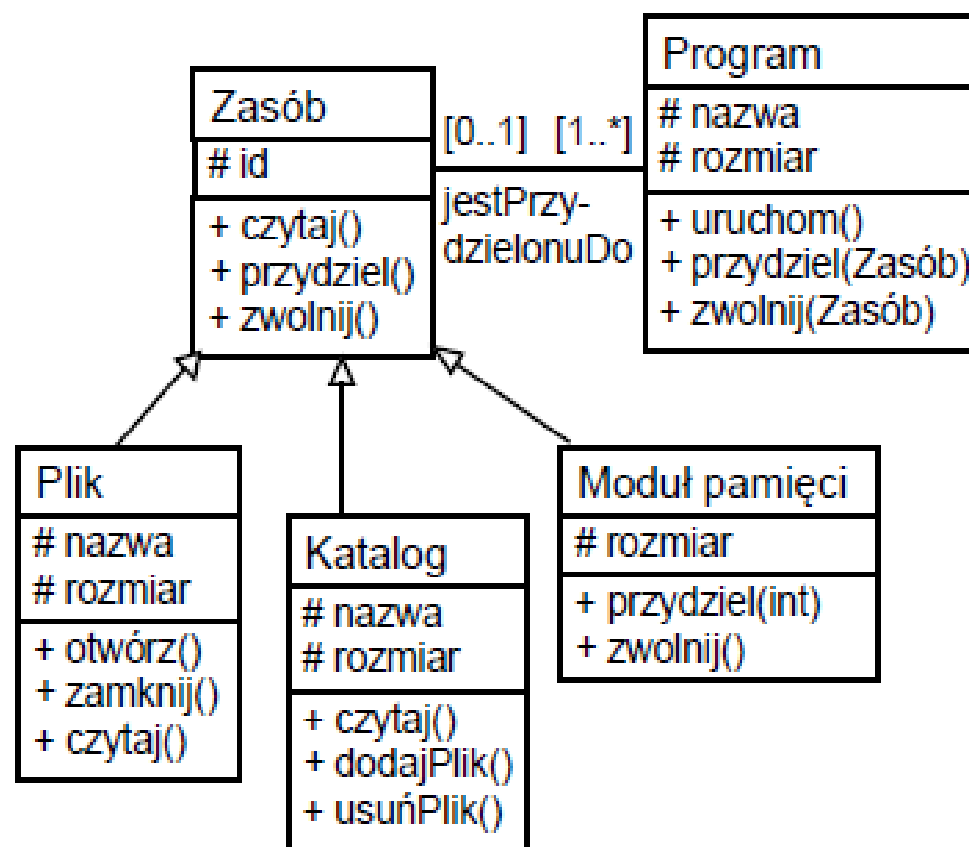
Architektura odzwierciedlająca dekompozycję funkcjonalną

Algorytm działania systemu operacyjnego DOS



Architektura odzwierciedlająca dekompozycję obiektową

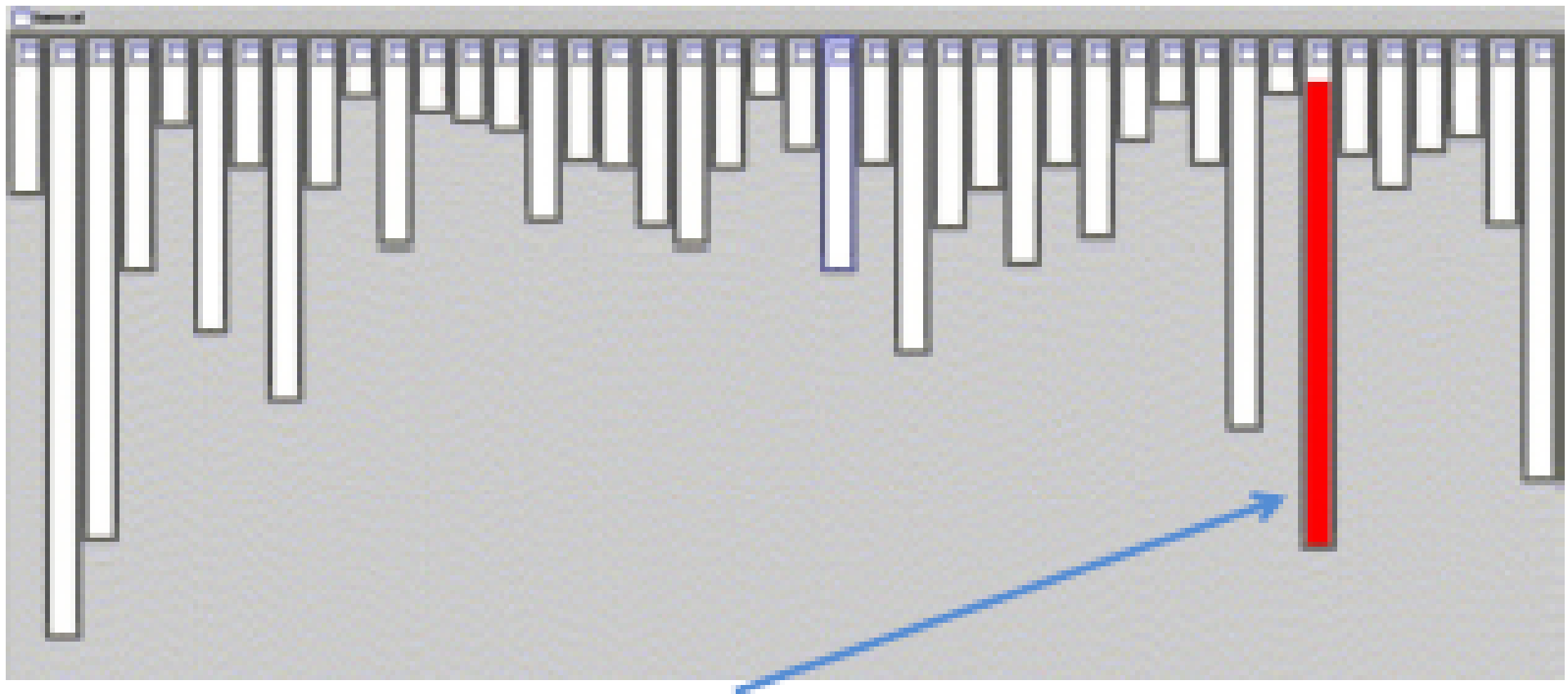
Diagram klas systemu operacyjnego DOS



Utrzymanie kodu

Dobra architektura

Odzwierciedla problemy i pojęcia

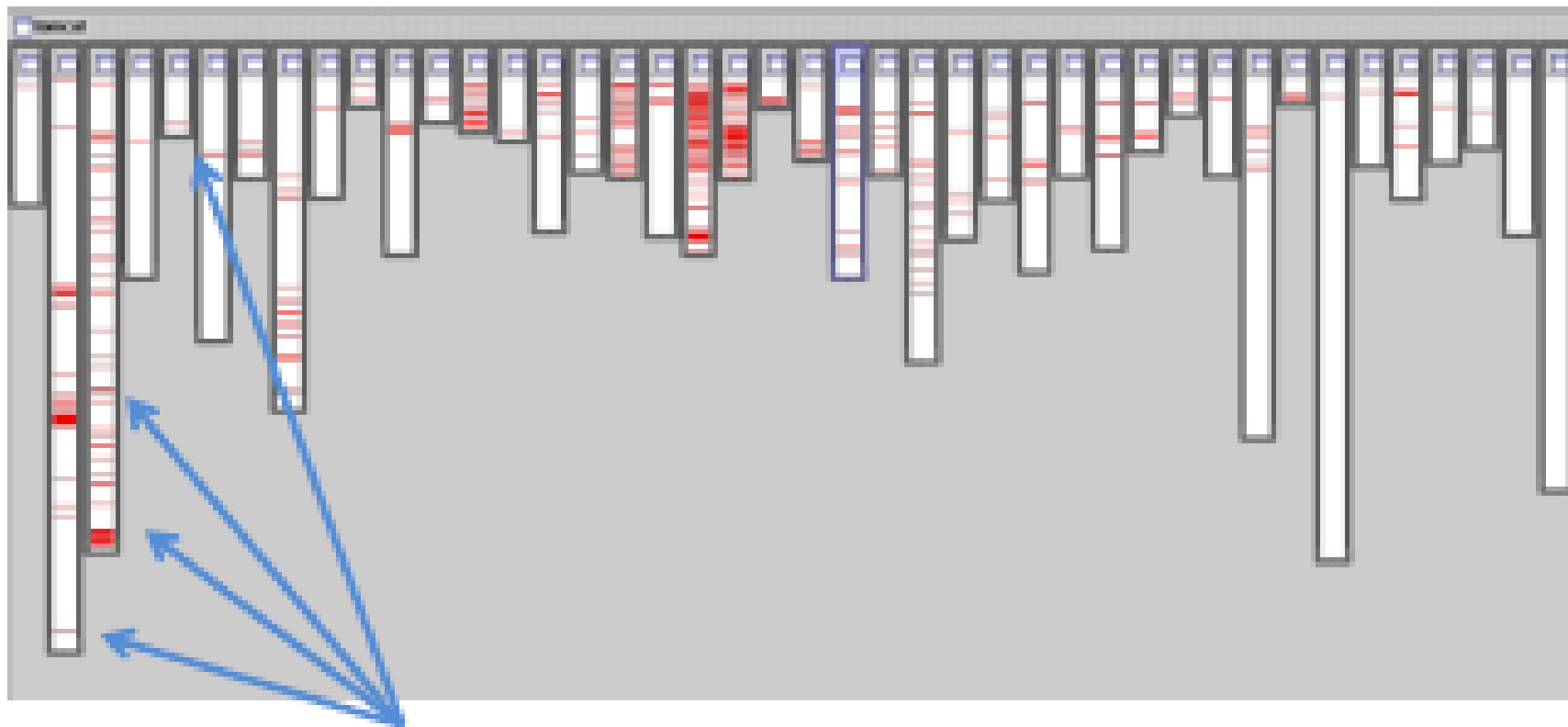


Analiza składniowa XML w Apache Tomcat.

Utrzymanie kodu

Zła architektura

Nie odzwierciedla problemów i pojęć



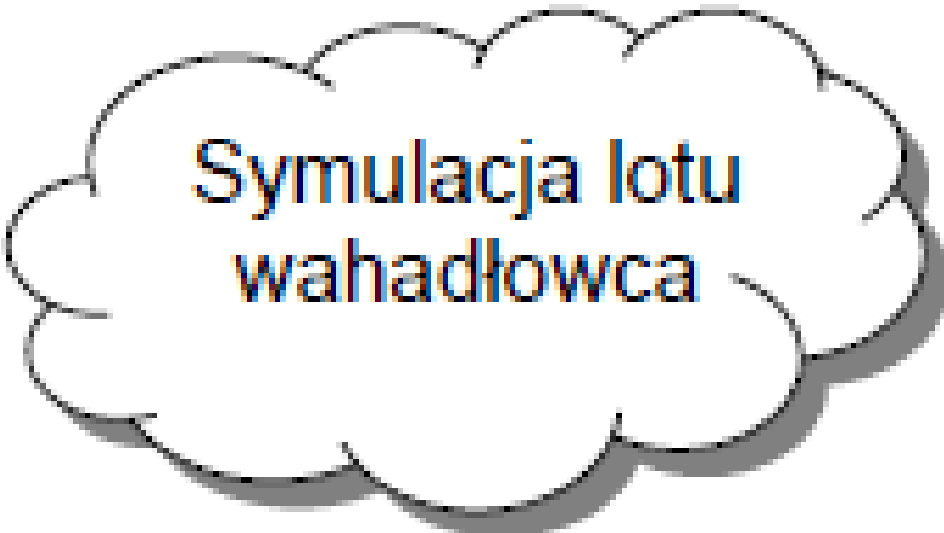
Składowanie informacji o zachowaniu aplikacji w logach - Apache Tomcat.

Klasyczne języki programowania

- **Klasyczne języki programowania**

Zamknięty zbiór predefiniowanych typów danych i operatorów

Reprezentowana
rzeczywistość



Symulacja lotu
wahadłowca

Siła języka
programowania

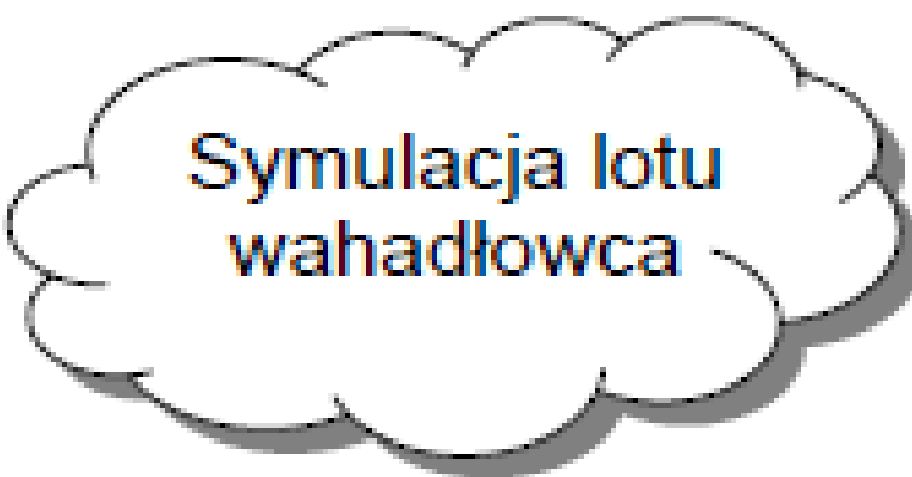
int, float, char, ...
+, -, *, /, ...
[], struct

Klasyczne języki programowania

- **Obiektowe języki programowania**

Możliwość nieograniczonego rozszerzania typów danych i operatorów

Reprezentowana
rzeczywistość



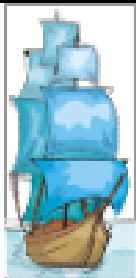
Symulacja lotu
wahadłowca

Siła języka
programowania

```
class Zespolone {...};  
class Wahadlowiec {...};  
...  
Zespolone z1(2.39, 1.92);  
z3 = z2 + z1;  
Wahadlowiec Columb(...);
```

Historia podejścia obiektowego

- Simula 67, (Ole-Johana Dahla i Kristena Nygaarda z Norsk Regnesentral w Oslo)
 - Symulacja statków – każdy statek inne atrybuty
 - Pogrupowanie statków na klasy obiektów definiujących dane i zachowanie
 - pierwotna koncepcja programowania obiektowego (pojęcie klasy i jej egzemplarza)
 - Wyspecjalizowany język do symulacji procesów

Żaglowiec	Parowiec	Kontenerowiec	Lotniskowiec
rodzajŻagli liczbaMasztów wysokośćNajwMasztu	liczbaPasażerów liczbaKotłów rodzajKotłów ciśnienieParyDolotowej	pojemnośćRejestrowa pojemnośćŁadunkowa	wyporność długośćPasaStartowego rodzajNapędu
			

Historia podejścia obiektowego

- Simula 6

```
Class Rectangle (Width, Height); Real Width, Height;  
Begin  
    Real Area, Perimeter;  
  
    Procedure Update;  
    Begin  
        Area := Width * Height;  
        Perimeter := 2*(Width + Height)  
    End of Update;  
  
    Boolean Procedure IsSquare;  
    IsSquare := Width=Height;  
  
    Update; ! Aktywacja obiektu;  
    OutText("Rectangle created: "); OutFix(Width,2,6);  
    OutFix(Height,2,6); OutImage;  
End of Rectangle;  
  
Ref(Rectangle) R; (Class reference variable)  
  
R := New Rectangle(50, 40);
```

Historia podejścia obiektowego

- Smalltalk, Alan Kay (Xerox PARC)
 - Dopracowana koncepcja obiektowości
 - Wprowadzenie dziedziczenia
 - Stosowany w praktyce
 - opracowany w Xerox Palo Alto Research Center
- Eiffel:
 - Bazujący na języku Ada
 - Wprowadził koncepcję klas generycznych

Historia podejścia obiektowego

- Smalltalk,

```
| rectangles aPoint |
rectangles := OrderedCollection
    with: (Rectangle left: 0 right: 10 top: 100
        bottom: 200)
    with: (Rectangle left: 10 right: 10 top: 110
        bottom: 210).
aPoint := Point x: 20 y: 20.
collisions := rectangles select:
    [:aRect | aRect containsPoint: aPoint]
```

Historia podejścia obiektowego

- Eiffel,

```
class STACK [G]  
...  
remove is  
    require          -- warunki początkowe  
        not_empty: not empty  
    do  
        ...  
    ensure          -- warunki końcowe  
        not_full: not full  
        one_fewer_item: count = old count - 1  
    end  
end
```

Historia podejścia obiektowego

- C++:
 - Programowanie obiektowe zyskało status techniki dominującej w połowie lat 80., głównie ze względu na wpływ C++,
 - Obiektowe rozszerzenie ANSI C
 - cechy obiektowe dodano do wielu języków programowania, w tym Ady, BASIC-a, Lisp-a, Pascala i innych.
- Java i C#:
 - Składnia wzorowana na C++
 - Architektura SmallTalk-80
 - W C#: własności obiektów takie jak cechy i zdarzenia

Historia podejścia obiektowego

- C++:

```
class TablicaLiczby {  
    ...  
public:  
    TablicaLiczby (unsigned);  
    TablicaLiczby (const TablicaLiczby&);  
    ...  
};  
  
TablicaLiczby::TablicaLiczby(const TablicaLiczby& t) {  
    fp = new float[rozmiar=t.rozmiar];  
    for( int i=0; i<rozmiar; i++)  
        fp[i]=t.fp[i];  
}
```

- Java:

```
class MySecurityManager extends SecurityManager {  
    public void checkRead (String file) {  
        if (file.endsWith(".doc") )  
            throw new SecurityException ("Dostęp do  
                pliku " + file + " zabroniony");  
        }  
    }  
}
```

Historia podejścia obiektowego

- C#:

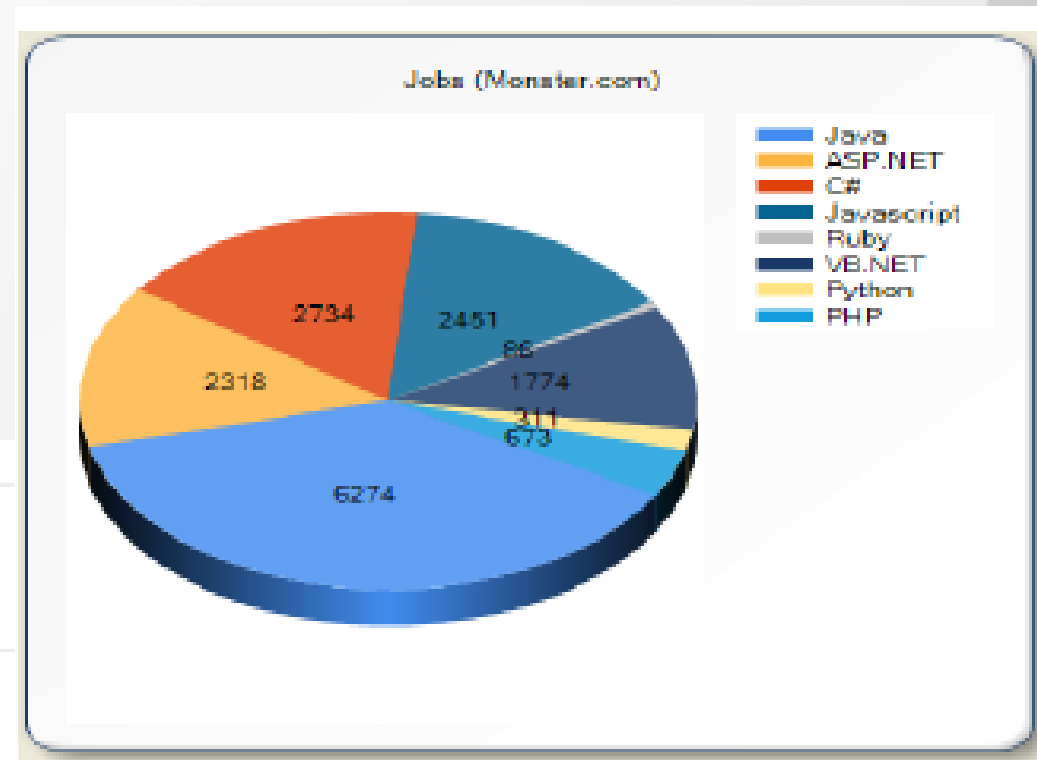
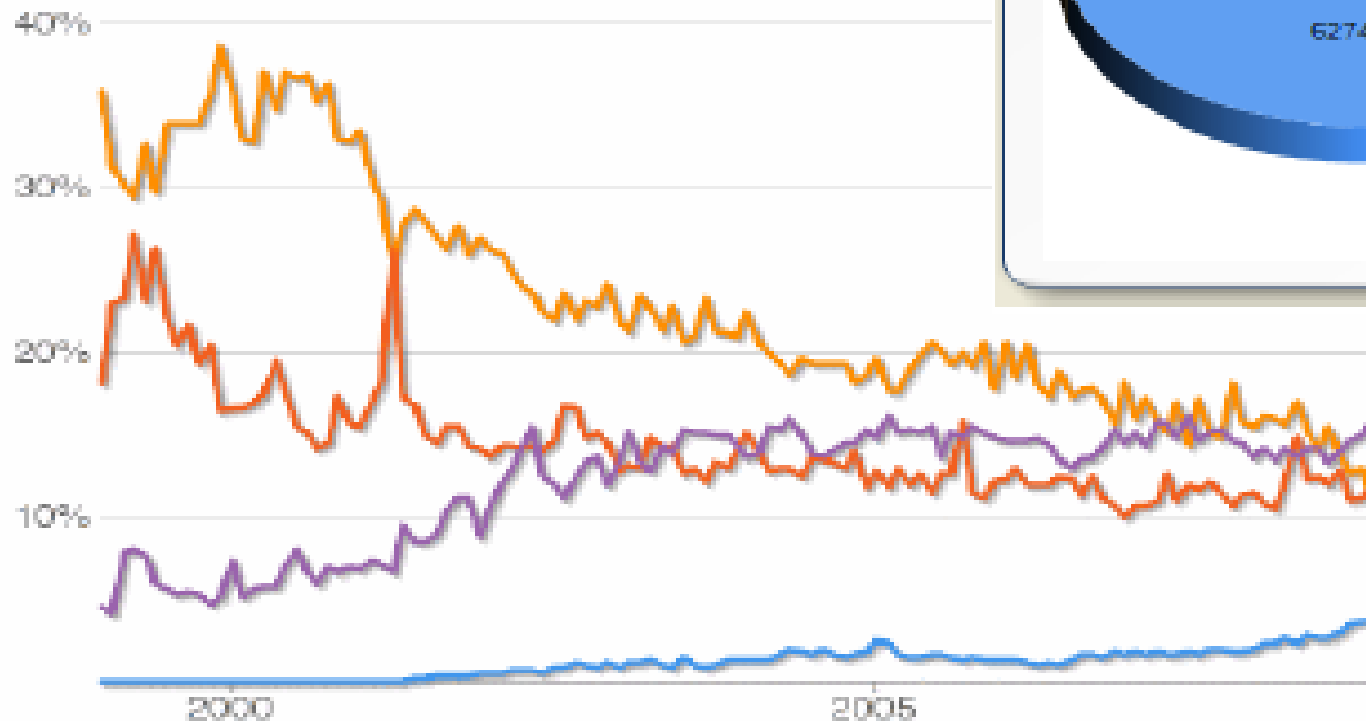
```
class TimePeriod
{
    private double seconds;
    public double Hours
    {
        get { return seconds / 3600; }
        set { seconds = value * 3600; }
    }
}

class Button
{
    // deklaracja zdarzenia
    public event ButtonEventHandler ButtonClick;

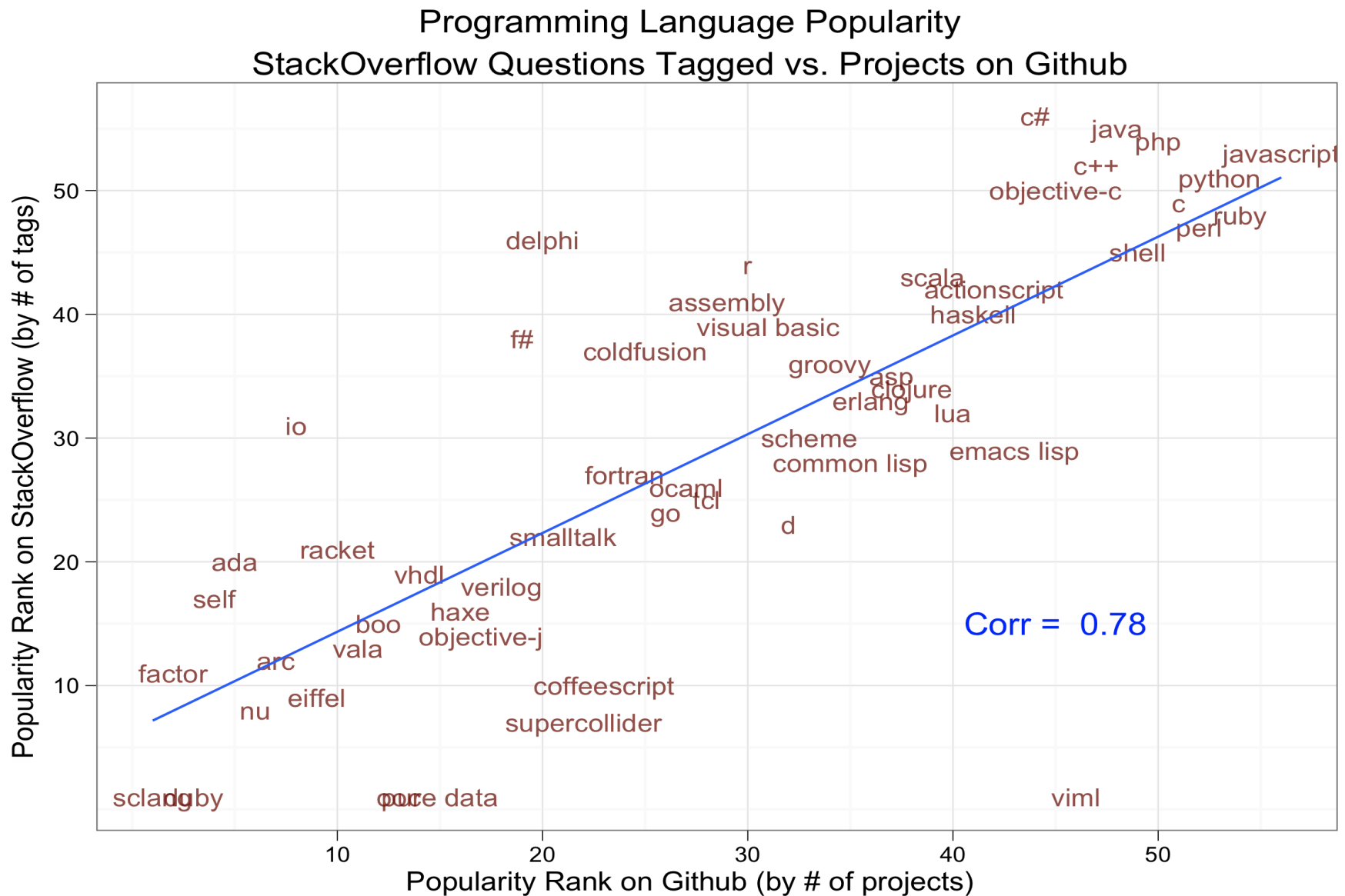
    // metoda wyzwalająca zdarzenie
    public void clicked(int count)
    {
        // wywołanie obsługi zdarzenia
        if (ButtonClick != null) ButtonClick (this, count);
    }
}
```

Języki obiektowe - praca

1. Język C – pomarańczowy
2. Język Java – fioletowy
3. Język C++ - czerwony
4. Język C# - niebieski



Języki obiektowe – community / projekty



Popularność [Tiobe 2012]

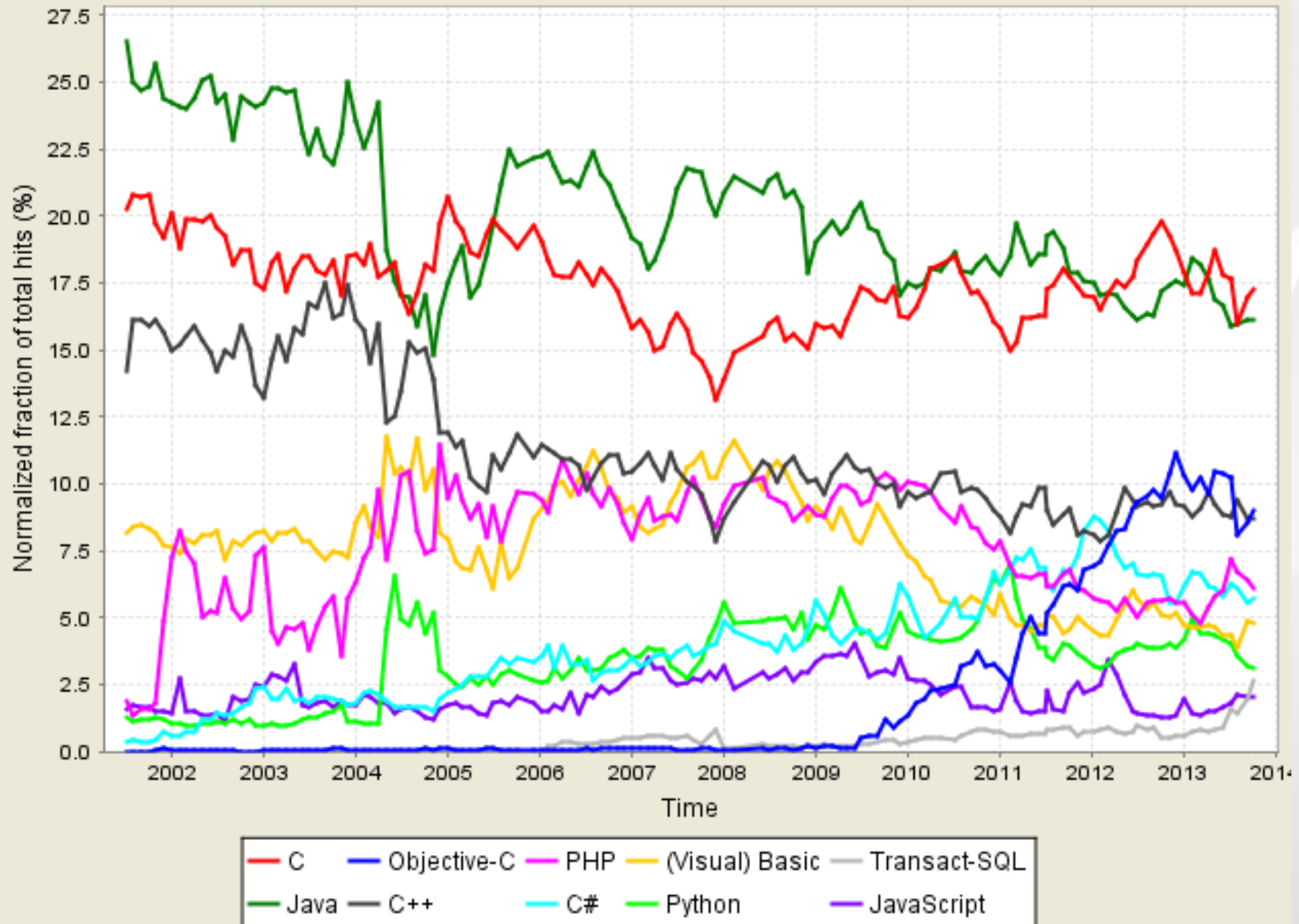
Position Nov 2013	Position Nov 2012	Delta in Position	Programming Language	Ratings Nov 2013	Delta Nov 2012	Status
1	1	=	C	18.155%	-1.07%	A
2	2	=	Java	16.521%	-0.93%	A
3	3	=	Objective-C	9.406%	-0.98%	A
4	4	=	C++	8.369%	-1.33%	A
5	6	↑	C#	6.024%	+0.43%	A
6	5	↓	PHP	5.379%	-0.35%	A
7	7	=	(Visual) Basic	4.396%	-0.64%	A
8	8	=	Python	3.110%	-0.95%	A
9	23	↑↑↑↑↑↑↑↑↑↑	Transact-SQL	2.521%	+2.05%	A
10	11	↑	JavaScript	2.050%	+0.77%	A

Popularność [Tiobe 2012]

11	15	↑↑↑↑↑	Visual Basic .NET	1.969%	+1.20%	A
12	9	↓↓↓	Perl	1.521%	-0.66%	A
13	10	↓↓↓	Ruby	1.303%	-0.44%	A
14	14	=	Pascal	0.715%	-0.17%	A
15	13	↓↓	Lisp	0.706%	-0.25%	A
16	19	↑↑↑	MATLAB	0.656%	+0.04%	B
17	12	↓↓↓↓↓	Delphi/Object Pascal	0.649%	-0.35%	A-
18	17	↓	PL/SQL	0.605%	-0.03%	A-
19	24	↑↑↑↑↑	COBOL	0.585%	+0.11%	B
20	20	=	Assembly	0.532%	-0.05%	B

Indeks popularności [Tiobe 2012]

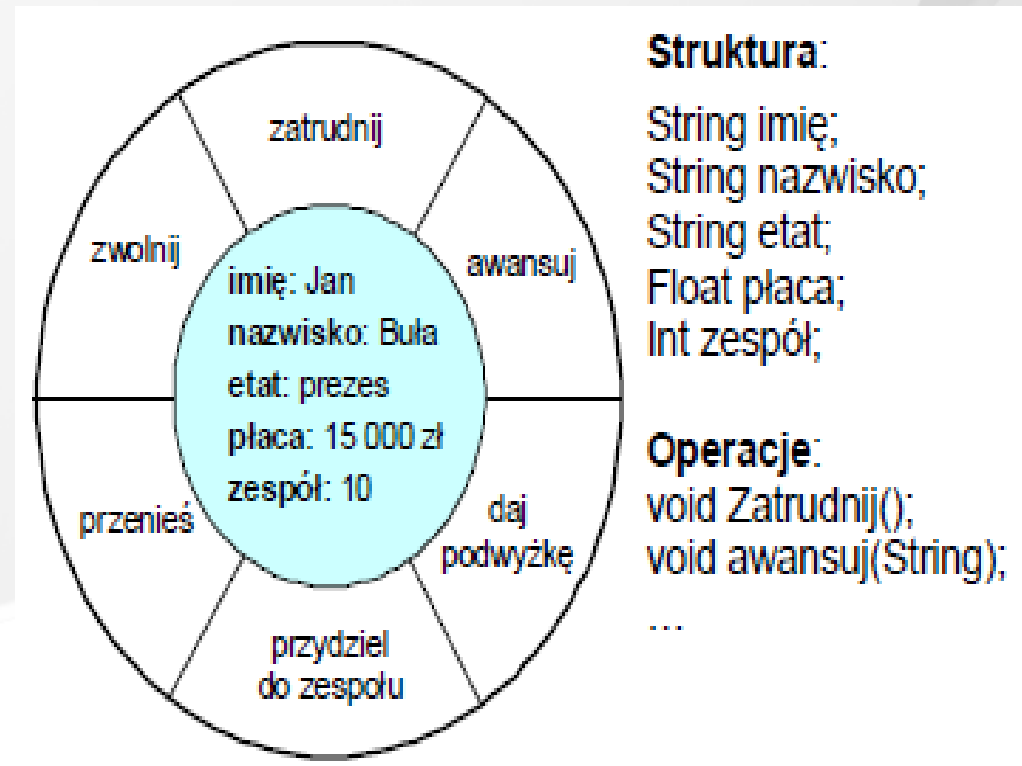
TIOBE Programming Community Index



Podstawowe pojęcia paradygmatu obiektowego

- **Obiekt:**

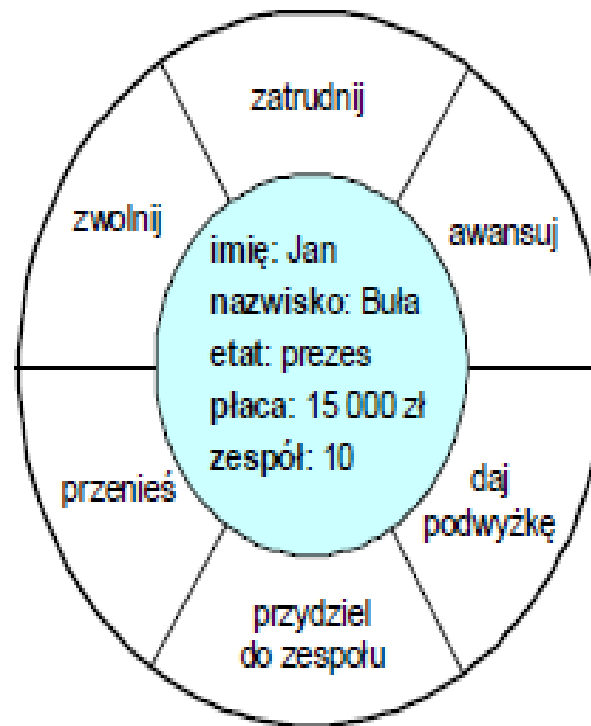
- Elementarna jednostka programowa
- Charakteryzuje się: strukturą, stanem oraz zachowaniem
- Obiekt w sposób logiczny wiąże dane (zmienne) oraz operacje na nich (metody)



Podstawowe pojęcia paradygmatu obiektowego

- **Komunikat:**

- Uaktywnia procedury związane z obiektem
- Zawiera on: nazwę procedury oraz opcjonalne parametry
- Zbiór wszystkich rozumianych przez obiekt komunikatów to interfejs obiektu



Struktura:

`String imię;`
`String nazwisko;`
`String etat;`
`Float płaca;`
`Int zespół;`

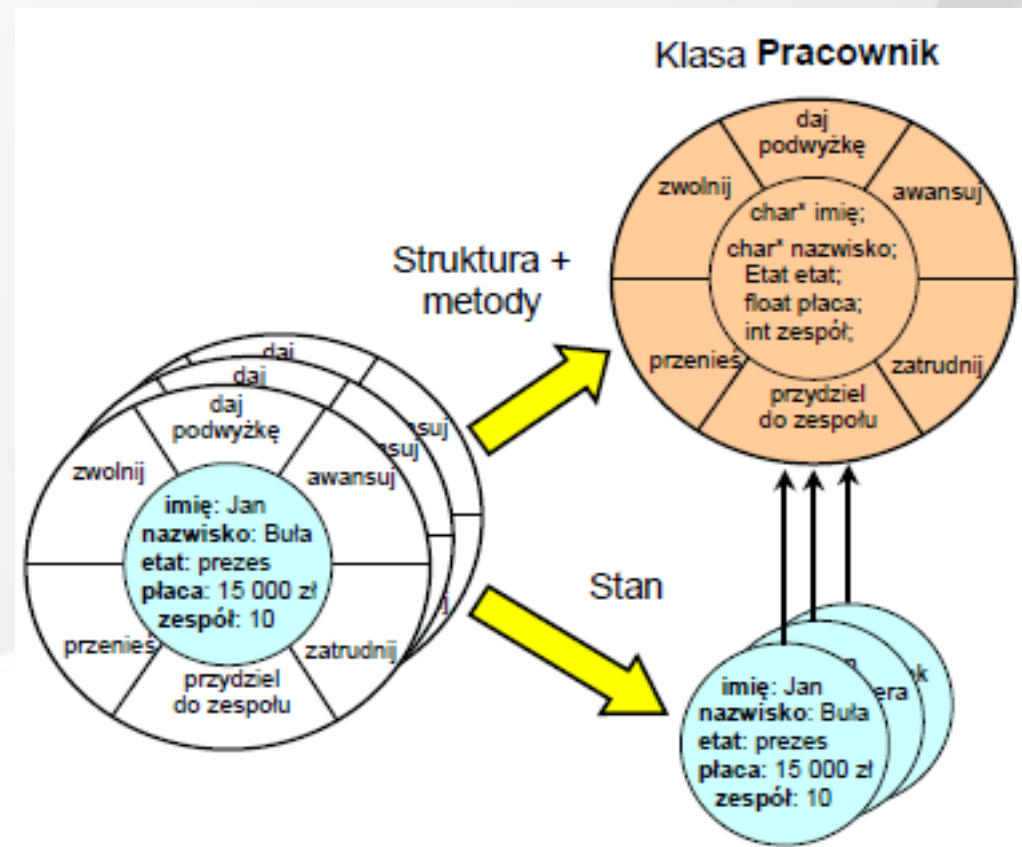
Operacje:

`void Zatrudnij();`
`void awansuj(String);`
...

Podstawowe pojęcia paradygmatu obiektowego

- **Klasa:**

- Specyfikacja i implementacja abstrakcyjnego typu danych
- Podstawowy moduł programu
- Obiekty to instancje klas
- Klasy definiują dane i metody obiektów



Założenia paradygmatu obiektowego

- **Abstrakcja**

- obiekt w systemie modeluje abstrakcyjnego „wykonawcę”
- Wykonuje on pracę, opisuje i zmienia swój stan oraz komunikuje się z innymi obiektami w systemie
- Ukrywa przy tym szczegóły implementacyjne określonych jego cech.
- Możliwość definiowania własnych typów danych

- **Dynamiczne wiązanie:**

- Wyszukiwanie metod w obiekcie dopiero podczas działania programu

Założenia paradygmatu obiektowego

- **Enkapsulacja (hermetyzacja)**

- ukrywanie implementacji
- Obiekt nie może zmieniać stanu innych obiektów w nieoczekiwany sposób.
- Zmiana stanu jest możliwa jedynie przez metody wewnętrzne obiektu.

- **Dziedziczenie**

- Tworzenie specjalizowanych obiektów na bazie bardziej ogólnych
- Klasy specjalizowane nie muszą definiować na nowo wspólnej z klasą generalizującą funkcjonalności
- Hierarchiczna struktura dziedziczenia

Założenia paradygmatu obiektowego

- **Polimorfizm (wielopostaciowość)**

- Referencje oraz kolekcje dotyczą różnych typów
- Wywołanie metody dla referencji powoduje zachowanie odpowiednie dla pełnego typu obiektu wywołanego
- Abstrahowanie w wyrażeniach od konkretnych typów
 - Ułatwia pisanie aplikacji
 - Większa czytelność kodu
- Wiązanie dynamiczne – w czasie wykonania
- Wiązanie statyczne – w czasie kompilacji