

Dr inż. Andrzej Czerepicki

Języki i paradygmaty programowania

Studia podyplomowe

Wykład 4.

Paradygmaty programowania

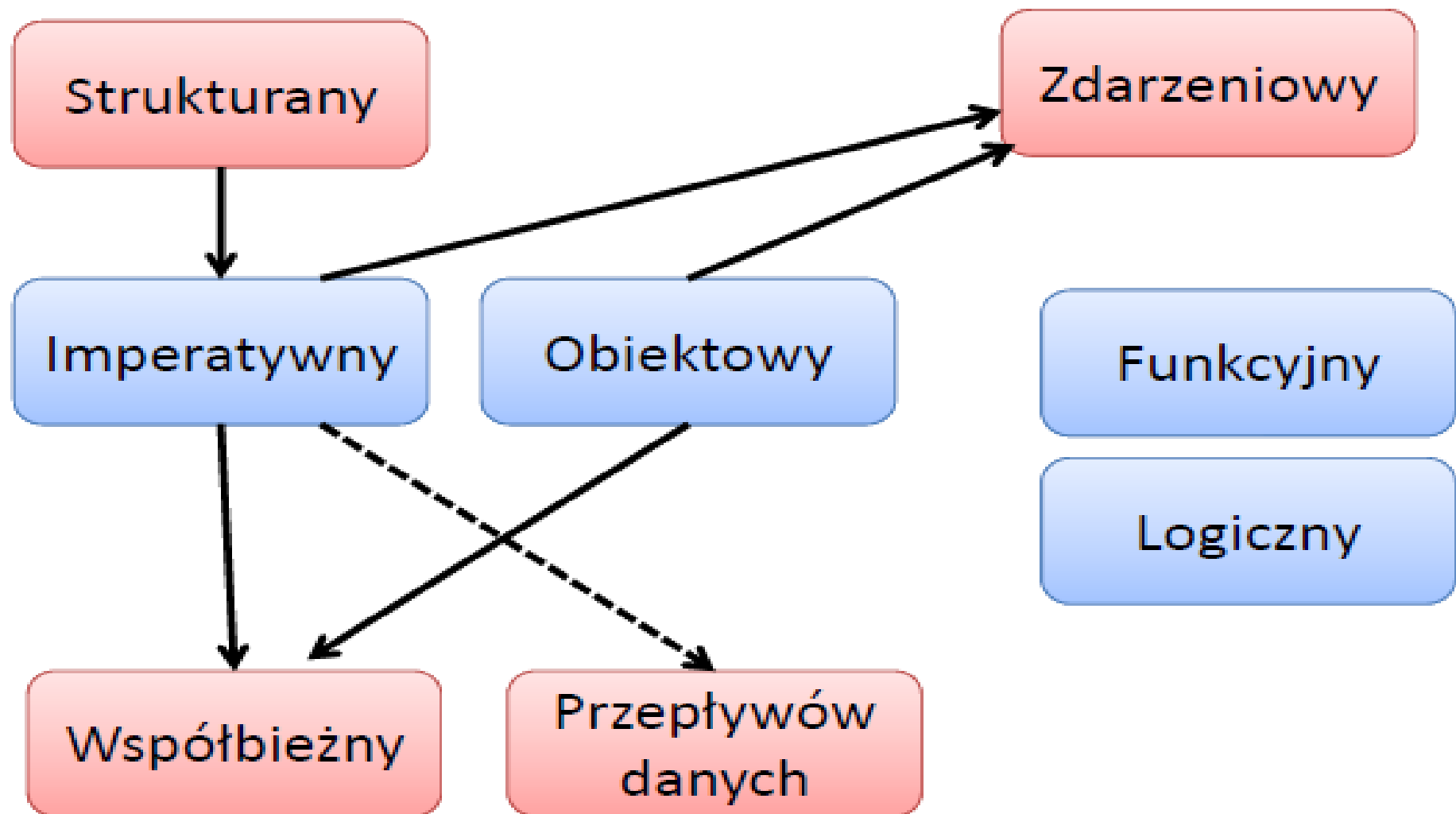
1. Definicja paradygmatu programowania

- Paradygmat programowania to:
 - Sposób podejścia do zaprogramowania algorytmu
 - Zbiór wzorców, którymi należy się posługiwać przy napisaniu programu
 - Model, określający sposób przełożenia rzeczywistych obiektów oraz zachodzących pomiędzy nimi relacji na ich odpowiedniki w programie
- Język programowania może reprezentować jeden bądź kilka paradygmatów

2. Historia rozwoju paradygmatów programowania

- Historia paradygmatów programowania jest ściśle związana z rozwojem języków programowania
 - Pierwszy paradygmat pojawił się wraz z pierwszym językiem
 - Nowe tendencje w programowaniu oraz coraz bardziej złożone języki przyczyniały się do powstania nowych paradygmatów
 - Nie każdy paradygmat wytrzymał weryfikację czasem, część z nich nie znalazła szerokiego praktycznego zastosowania

3. Klasyfikacja paradygmatów programowania



4. Paradygmat imperatywnego programowania

4. Paradygmat imperatywnego programowania

- Program komputerowy jest traktowany jako ciąg instrukcji wykonywanych w określonej kolejności
- Jest to historycznie pierwsze podejście do programowania
- Imperatywne programowanie jest naturalnym sposobem realizacji algorytmów dla komputerów zbudowanych w architekturze von Neumanna oraz Harvardzkiej

4. Paradygmat imperatywnego programowania (c.d.)

- Programowanie imperatywne opiera się na dwóch abstrakcjach:
 - **Obiekty** świata rzeczywistego w programie imperatywnym są reprezentowane przez **zmienne**
 - **Procesy** zachodzące pomiędzy obiektami są realizowane poprzez **podprogramy** w imperatywnym języku programowania

4.1. Zmienne

- Zmienna – obszar pamięci operacyjnej komputera, przechowujący dane opisujące pewien obiekt
- Zmienna:
 - Posiada własną unikalną nazwę
 - Należy do określonego typu danych
 - Przechowuje wartość charakteryzującą stan reprezentowanego obiektu

4.1. Zmienne (c.d.)

- Przykład: zmienna będąca abstrakcją piętra budynku:
 - nazywa się „piętro”
 - Nazwa oczywiście może być dowolna
 - ma typ „integer”
 - numer piętra jest liczbą całkowitą
 - przechowuje wartość 5
 - Co może oznaczać np. że dźwig jest obecnie na 5 pięttrze
 - Wartość zmiennej, jak sugeruje sama nazwa, oczywiście może być zmieniona

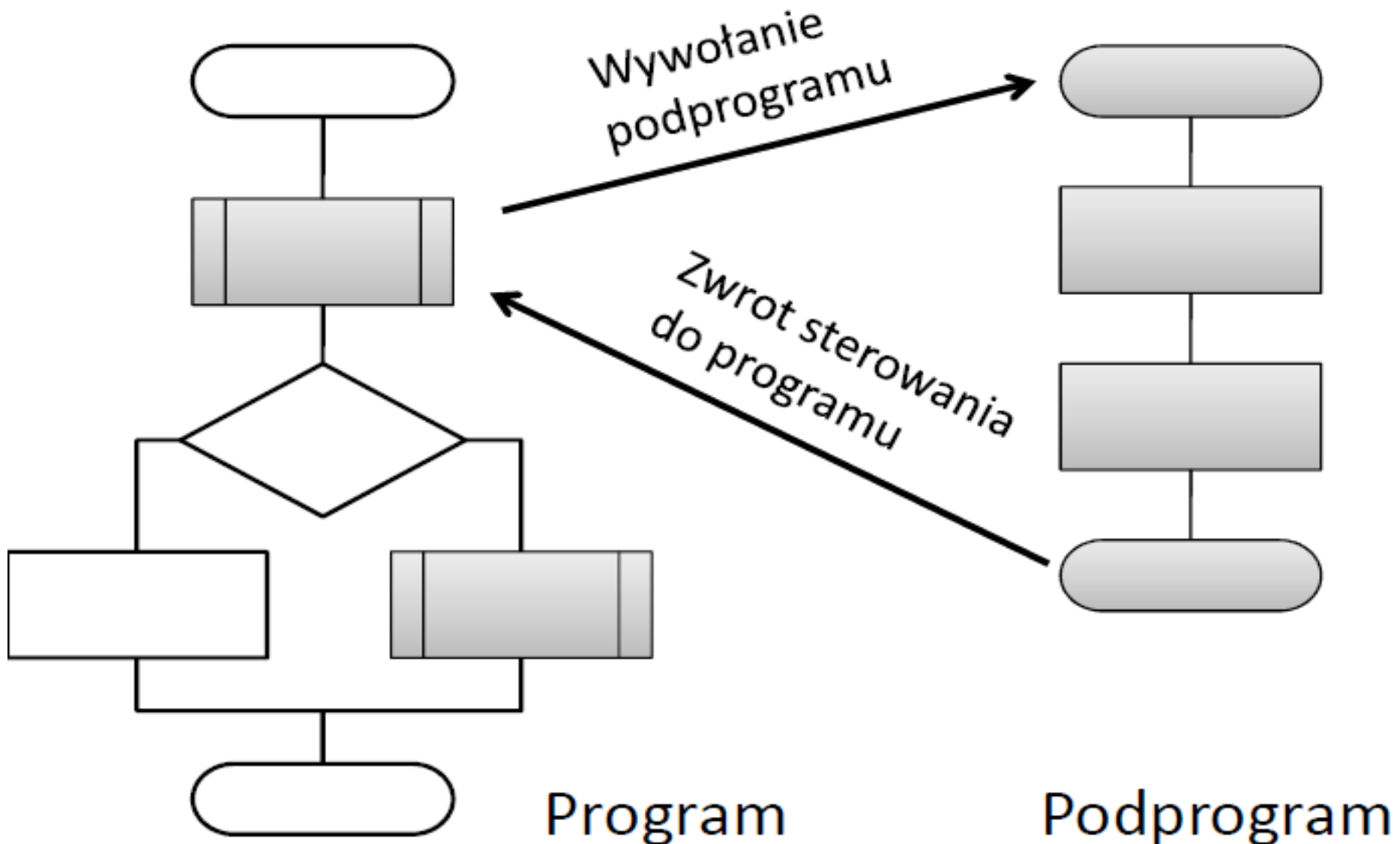
4.2. Podprogramy

- Podprogram – wyodrębniony fragment programu komputerowego, posiadający wszystkie cechy programu oraz realizujący określony algorytm
- Podprogramy są **abstrakcja procesów** zachodzących pomiędzy obiektami (danymi)

4.2. Podprogramy (c.d.)

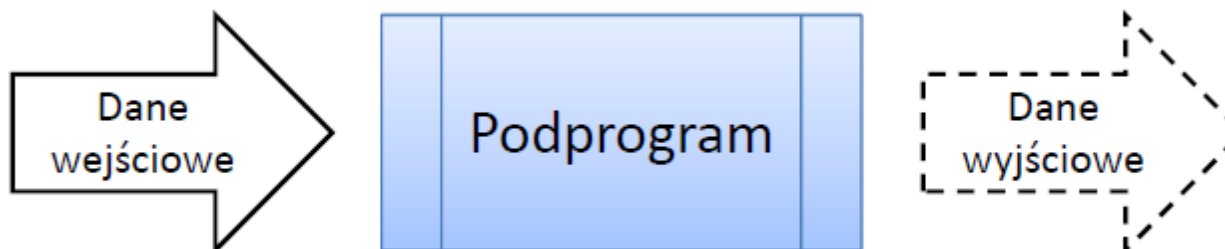
- Głównym celem podprogramów była eliminacja powtarzających się fragmentów kodu źródłowego programu
 - Kod jest opracowywany jeden raz i zamykany w ramach podprogramu, który następnie może być wielokrotnie wołany z różnych miejsc w programie
- Stosowanie podprogramów zwiększa czytelność kodu źródłowego programu, porządkuje kod oraz przyczynia się do zmniejszenia ilości błędów

4.2. Podprogramy (c.d.)



4.2. Podprogramy (c.d.)

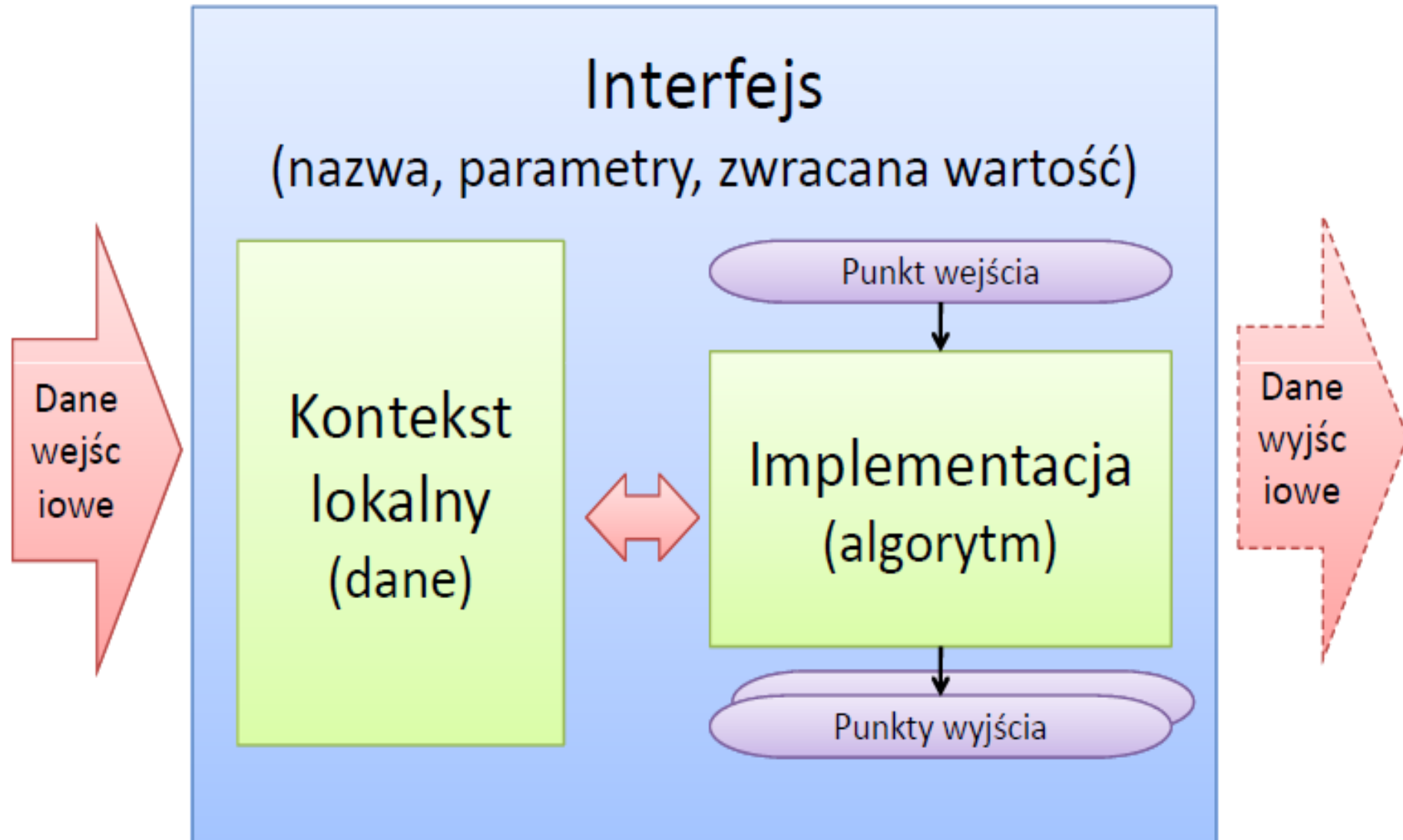
- Podprogramy mogą posiadać **parametry wejściowe**, przez które są przekazywane dane wewnątrz podprogramu
- Podprogramy mogą zwracać wartość będąca **wynikiem działania** podprogramu
 - W wielu językach podprogram zwracający wartość nazywa się **funkcją**, nie zwracający - **procedurą**



4.2. Podprogramy (c.d.)

- Charakterystyki podprogramów:
 - Posiadają własną nazwę, poprzez którą odbywa się wywołanie podprogramu
 - Posiada implementację („ciało” podprogramu, ciąg instrukcji wykonujących określony algorytm)
 - Posiadają jeden punkt wejścia (początek podprogramu) oraz jeden przynajmniej jeden punkt wyjścia (zakończenie podprogramu)

4.2. Podprogramy (c.d.)



4.2. Podprogramy (c.d.)

- Przykład: podprogram modelujący przemieszczanie się dźwigu o jedno piętro w górę lub w dół
 - Podprogram „Krok”
 - Parametry wejściowe:
 - *PietroBieżące* (liczba całkowita)
 - *Kierunek* (liczba całkowita), +1 lub -1
 - Zwracana wartość:
 - Piętro na którym dźwig się zatrzyma (liczba całk.)
 - Algorytm:
 - Dodaj do wartości zmiennej *PietroBieżące* wartość zmiennej *Kierunek* i zwróć obliczony wynik

4.3. Języki reprezentujące imperatywny paradygmat

- Każdy język 1 i 2 generacji
- Zdecydowana większość języków 3 generacji (Algol, Fortran, Pascal, C, Basic, itp.)
 - Za wyjątkiem „czysto obiektowych” języków, które nie odniosły sukcesów
- Większość języków 4 generacji również implementuje imperatywny paradygmat programowania (jako jeden z podstawowych)

4.4. Podsumowanie

- Imperatywny paradygmat programowania jest historycznie pierwszym paradygmatem
- Programowanie imperatywne opiera się na zmiennych jako abstrakcji danych, oraz procedurach jako abstrakcji procesów
 - Synonim: programowanie proceduralne
- Sterowanie programem odbywa się za pomocą instrukcji jednoznacznie wskazujących jak komputer ma realizować algorytm
- Zdecydowana większość języków realizuje imperatywny paradygmat programowania

5. Paradygmat programowania obiektowego

5.1. Przesłanki powstania programowania obiektowego

- Języki imperatywne opierają się o abstrakcje sztuczne, wymuszone przez architekturę komputera i słabo odzwierciedlające faktyczne obiekty i zjawiska nas otaczające
- Procesy naturalne dla świata rzeczywistego często są trudne do zaimplementowania w postaci sztywnych imperatywnych procedur
 - Najtrudniejsze jest zapanowanie nad systemem, który może znajdować się w wielu stanach, kiedy przejścia pomiędzy stanami mogą być uzależnione od zewnętrznych zdarzeń oraz stanu bieżącego systemu

5.1. Przesłanki powstania programowania obiektowego (c.d.)

- Wraz z wzrostem złożoności systemu programy imperatywne stają się coraz trudniejsze w utrzymaniu
 - Zmiana w jednej procedurze może negatywnie odbić się na całym systemie, gdyż procedura może być wywołana z każdego miejsca
 - Złożona hierarchia wywołań podprogramów komplikuje zrozumienie działania całego systemu
- Wszystko to doprowadziło do poszukiwania koncepcji programowania, która pozwoliła by zachowując zalety programowania imperatywnego wyeliminować ww. wady

5.2. Koncepcja programowania obiektowego

- Otaczający nas świat składa się z ogromnej ilości obiektów
 - Budynek, piętro, dźwig, ulica, samochód itp.
- Każdy obiekt należy do pewnej grupy (klasy) obiektów, posiadających identyczny zbiór cech
 - Klasa „samochód” ma cechy „model”, „kolor”, „rok produkcji”, „ilość miejsc” itp.
- Obiekty współpracują pomiędzy sobą
 - *Dźwig* przemieszcza się na kolejne *piętro*
 - *Kierowca* uruchomił *silnik* samochodu

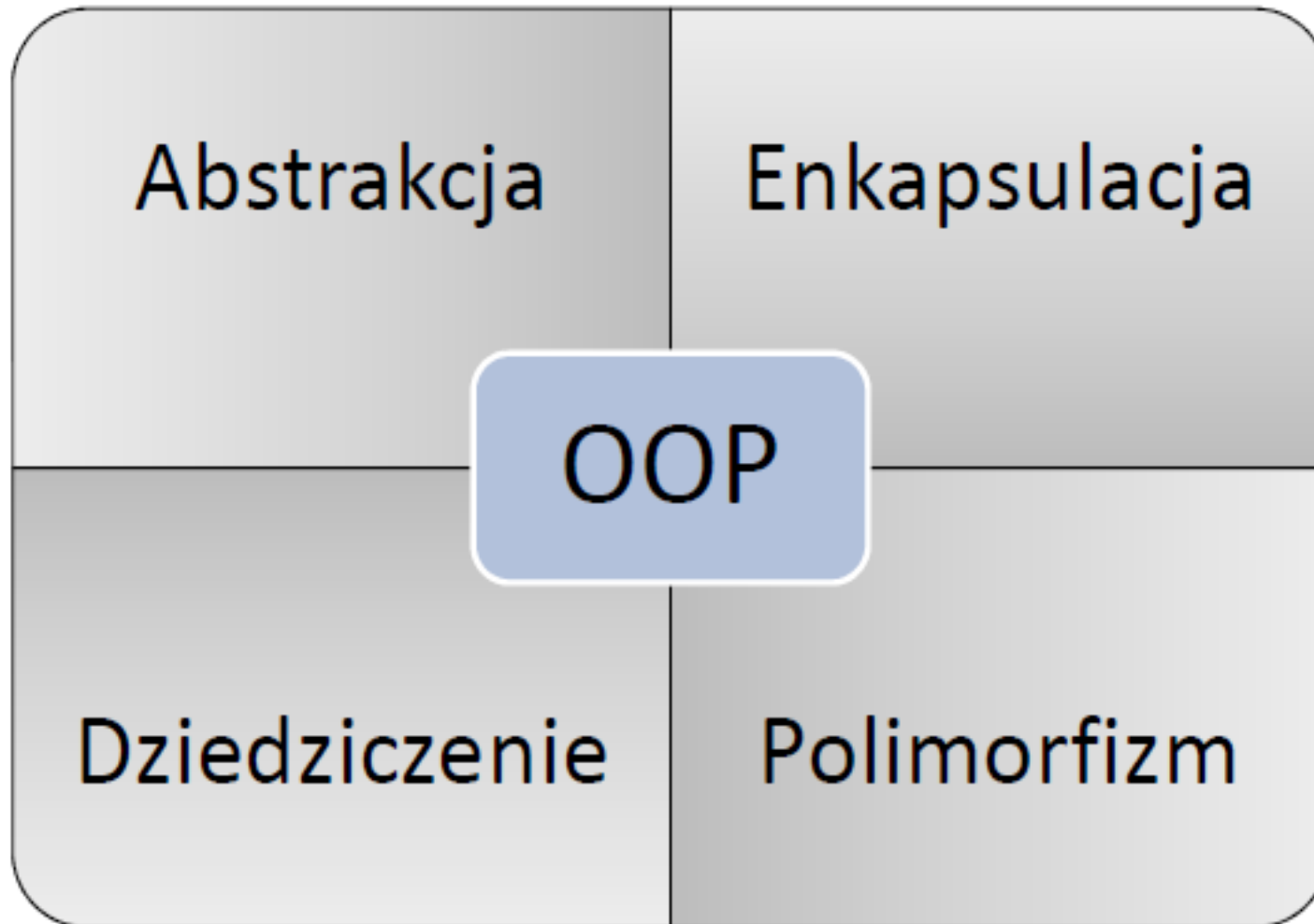
5.2. Koncepcja programowania obiektowego (c.d.)

- Pojedynczy obiekt posiada indywidualny zbiór cech jednoznacznie go charakteryzujący
 - Obiekt klasy „Dźwig”:
 - Model = „.....”
 - Maksymalne obciążenie = 500 kg
 - Drzwi = otwarte
 - Piętro = 6
- Każdy **obiekt** zachowuje się zgodnie z zasadami określonymi przez jego **klasę**

5.2. Koncepcja programowania obiektowego (c.d.)

- **Suma cech obiektu** reprezentuje jego **stan**
- **Operacje** które mogą być wykonane na obiekcie (bądź z jego udziałem) określają **zachowanie obiektu**
 - Obiekty klasy „Dźwig” posiadają funkcjonalność:
 - DoGóry
 - WDół
 - OtwórzDrzwi
 - ZamknijDrzwi
 - ...

5.3. Fundamenty programowania obiektowego

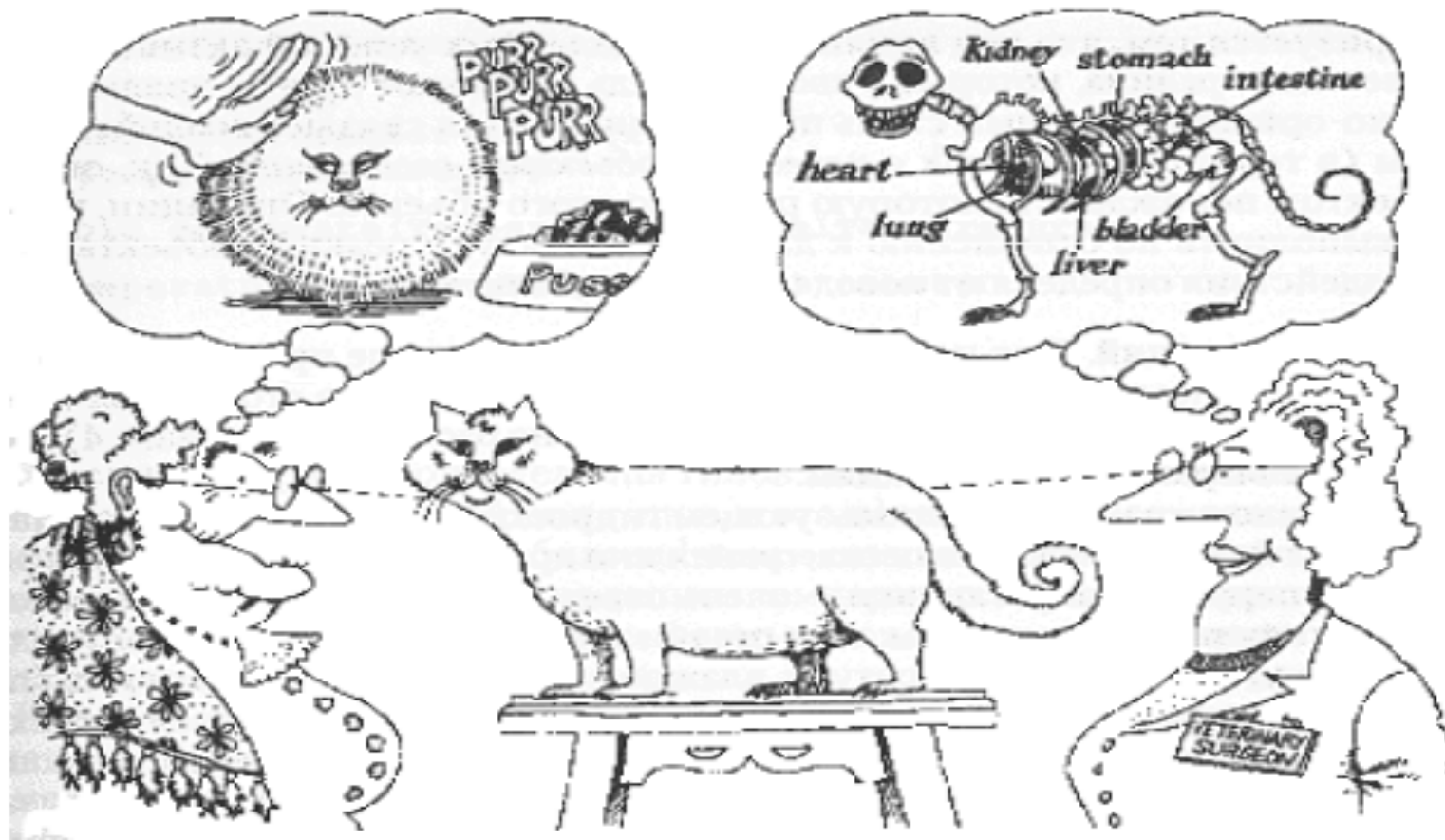


5.3. Fundamenty programowania obiektowego (c.d.)

- **Abstrakcja** – sposób przedstawienia obiektu rzeczywistego w programie
 - Obiekt w rzeczywistości jest niezmiernie złożony
 - W aspekcie programu nas interesują tylko wybrane jego cechy
 - Obiekt w programie jest abstrakcją obiektu rzeczywistego zawierającą wyłącznie niezbędne cechy
 - W zależności od programu abstrakcje tego samego obiektu rzeczywistego mogą być zupełnie odmienne

5.3. Fundamenty programowania obiektowego (c.d.)

- Abstrakcja



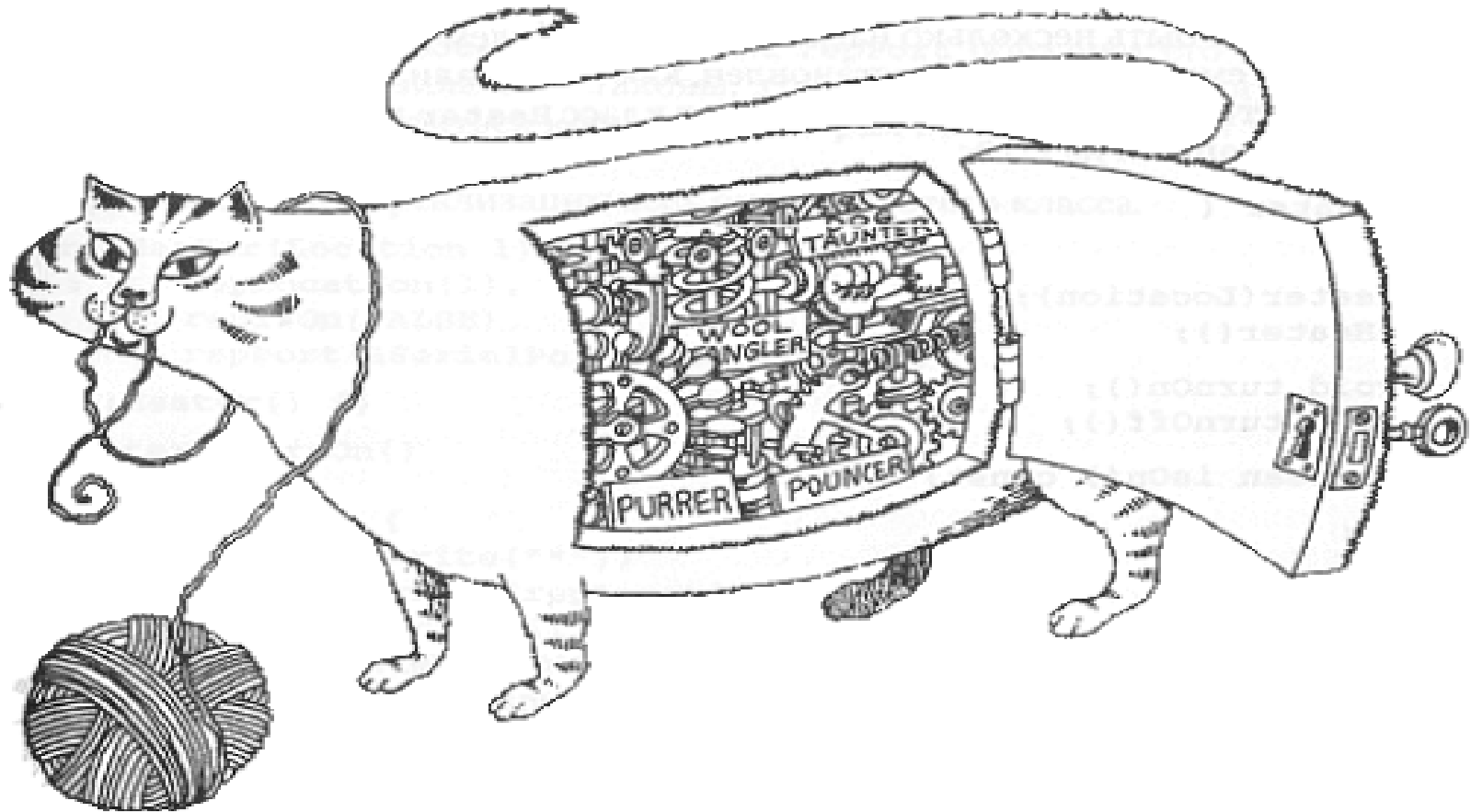
G. Booch. Object-Oriented Analysis and Design with Applications

5.3. Fundamenty programowania obiektowego (c.d.)

- **Enkapsulacja** (hermetyzacja) – ukrywanie wewnętrznej realizacji obiektu
 - **Obiekt** jest traktowany jako „czarna skrzynka”
 - Dostęp do danych obiektu odbywa się za pomocą **właściwości** (specjalnych podprogramów, przekazujących wartość do zmiennej wewnątrz obiektu lub zwracających jej wartość)
 - Stan obiektu jest zmieniany za pośrednictwem wywołania jego **metod** (podprogramów zmieniających wewnętrzny stan obiektu)

5.3. Fundamenty programowania obiektowego (c.d.)

- Enkapsulacja



5.3. Fundamenty programowania obiektowego (c.d.)

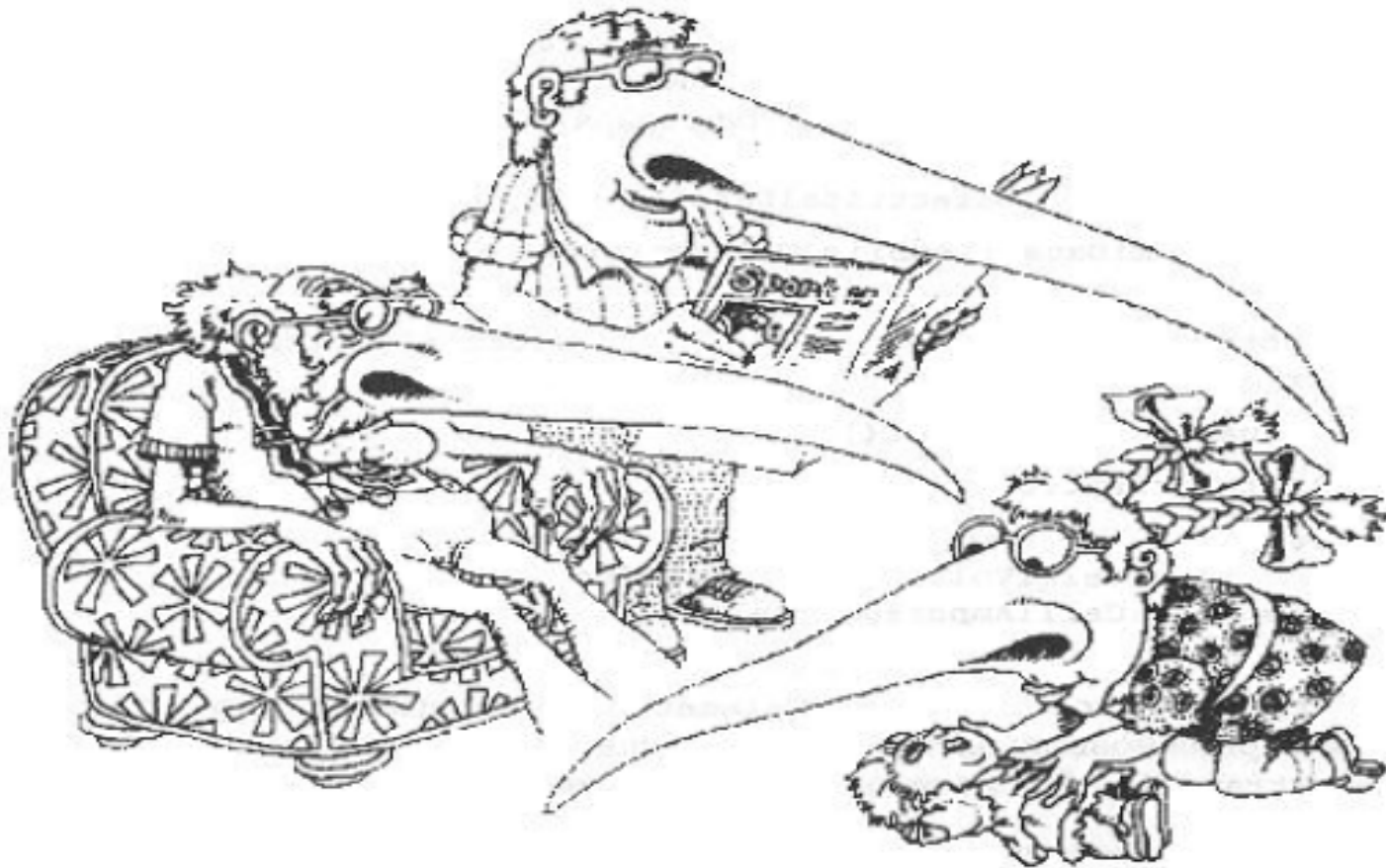
- Enkapsulacja (c.d.)
 - Przykład:
 - Właściwość *Piętro* obiektu klasy *Dźwig* zwraca bieżącą pozycję (piętro)
 - Wywołanie metody *DoGóry* obiektu klasy *Dźwig* zmienia jego stan (*piętro*)
 - Wywołanie metody *OtworzDrzwi* obiektu klasy *Dźwig* również zmienia jego stan

5.3. Fundamenty programowania obiektowego (c.d.)

- **Dziedziczenie** – mechanizm pozwalający budowę nowych klas obiektów na bazie istniejących z wykorzystaniem już zaimplementowanej funkcjonalności
 - Opiera się na relacji „przodek - potomek”
 - Pozwala na budowę hierarchii klas mających wspólne cechy
 - Eliminuje konieczność implementacji powtarzających się funkcjonalności

5.3. Fundamenty programowania obiektowego (c.d.)

- Dziedziczenie



5.3. Fundamenty programowania obiektowego (c.d.)

- Dziedziczenie (c.d.)
 - Przykład:
 - Klasa *Dźwig* realizuje wspólną funkcjonalność dla wszystkich rodzajów dźwigów
 - DoGory / wDol
 - OtworzDrzwi / ZamknijDrzwi
 - Klasy *DźwigOsobowy* oraz *DźwigTowarowy* dziedziczą z klasy *Dźwig* jej funkcjonalność oraz mogą dodawać własne unikatowe atrybuty oraz metody

5.3. Fundamenty programowania obiektowego (c.d.)

- **Polimorfizm** – mechanizm realizujący odmienne (indywidualne) zachowanie obiektów powiązanych relacją dziedziczenia
- Metody klasy realizujące odmienne zachowanie obiektów nazywają się **metodami wirtualnymi**
- Przykład:
 - Atrybut *MaksymalneObciążenie* dla klas *DźwigOsobowy* oraz *DźwigTowarowy* będzie zwracał różne wartości w zależności od indywidualnych cech każdej klasy

5.4. Języki obiektowe

- Simula 67
 - Pierwsza zrealizowana w praktyce koncepcja języka obiektowego (na bazie imperatywnego)
 - Od początku zorientowany na modelowanie zachowania złożonych obiektów (statki, samochody, samoloty itp.)
 - Wyprzedził swój czas i nie został doceniony przez ówczesnych programistów
 - m. innymi przez niezbyt udaną implementację
 - Był aktywnie wykorzystywany w procesie nauczania

5.4. Języki obiektowe (c.d.)

- Smalltalk (lata 70)
 - Pierwszy „czysto obiektowy” język programowania
 - Realizuje koncepcję obiektów współdziałających za pośrednictwem komunikatów
 - Model który stał się standardem we współczesnym programowaniu obiektowym
 - Wiele rewolucyjnych pomysłów w programowaniu w latach 80-90 wywodzi się ze Smalltalk
 - Mimo stosunkowo niewielkiej popularności, ma stałe grono zwolenników i jest wciąż rozwijany

5.4. Języki obiektowe (c.d.)

- C++ (lata 80.)
 - Opracowany przez B.Stroustrup'a, szybko stał się najbardziej popularnym językiem programowania na świecie:
 - Niewątpliwie przyczyniła się do tego składnia oparta na składni dobrze znanego języka C, który wówczas miał ogromną armię zwolenników
 - Realizuje wszystkie postulaty programowania obiektowego
 - Jest językiem bardzo elastycznym i uniwersalnym jednocześnie, pozwala na programowanie w bardzo szerokim zakresie zastosowań

5.4. Języki obiektowe (c.d.)

- Lata 90. Rozwój języków obiektowych
 - Większość z języków programowania wprowadza cechy obiektowe
 - Są opracowywane rozszerzenia dla języków PASCAL, LISP, Basic, itp.
 - Rozbudowa już istniejących języków sprawia coraz większe kłopoty z kompatybilnością oraz stabilnością aplikacji
 - Nawet C++ coraz bardziej odczuwa wady języka C
 - Tym nie mniej paradygmat obiektowy staje się najbardziej popularnym zwłaszcza w programowaniu aplikacji użytkowych

5.4. Języki obiektowe (c.d.)

- Koniec lat 90. Kryzys paradygmatu obiektowego
 - Coraz głośniej się mówi o wadach programowania obiektowego
 - Programy są zbyt wolne, kod skompilowany jest nadmiarowy, nie nadają się do zastosowania w systemach czasu rzeczywistego
 - Czysto obiektowym językom brakuje funkcjonalności imperatywnych, rozbudowa imperatywnych języków nie przyniosła oczekiwanych efektów
 - Język JAVA staje się pierwszą udaną realizacją języka uniwersalnego (obektowego oraz imperatywnego) zbudowanego od podstaw

5.4. Języki obiektowe (c.d.)

- 2000 – 2010
 - Sukces języka Java przyczynia się do powstania platformy .NET firmy Microsoft, również łączącej w sobie cechy imperatywnych oraz obiektowych języków programowania
 - Programowanie obiektowe zdobywa rynek aplikacji internetowych
 - Język PHP nabywa cech obiektowych
 - Niespodziewany sukces odnosi czysto obiektowy język Ruby (nowość spośród języków progr.)
 - Książka z programowania w języku Ruby jest bestsellerem roku 2008

5.4. Języki obiektowe (c.d.)

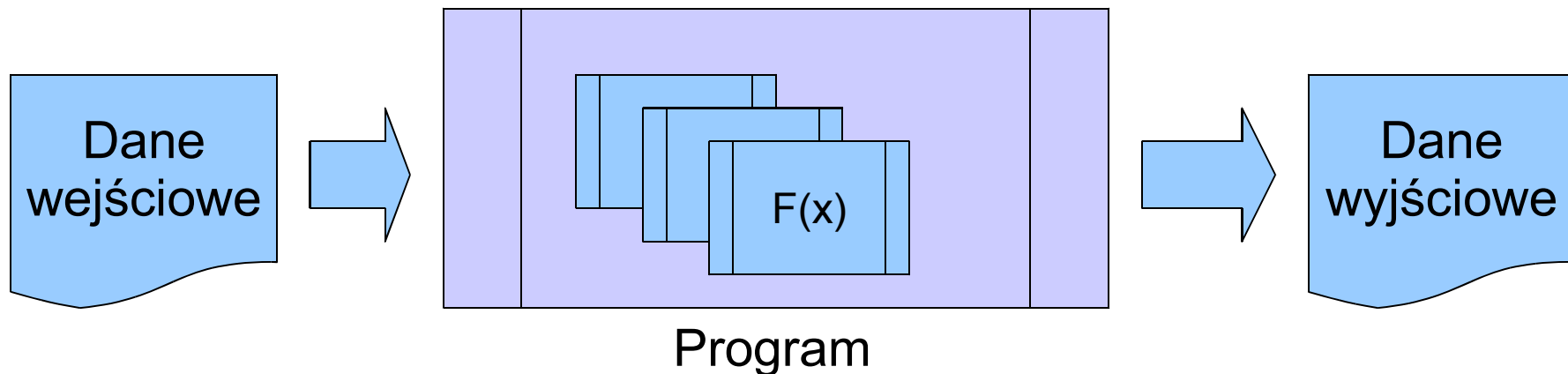
- Podsumowanie
 - Programowanie obiektowe znalazło swoją niszę na rynku i sukcesywnie jest stosowane do rozwiązywania określonego rodzaju zadań, przede wszystkim:
 - Programowania graficznych interfejsów użytkownika
 - Realizacji złożonej logiki biznesowej programów
 - Modelowania funkcjonowania złożonych systemów
 - Jednocześnie wady charakterystyczne dla języków obiektowych nie pozwalają na ich stosowanie w dziedzinach wymagających zapewnienia wysokiej wydajności oraz stabilności systemu

5.4. Języki obiektowe (c.d.)

- Podsumowanie (c.d.)
 - Programowanie obiektowe przyczyniło się do powstania nowych technologii programowania (np. wzorce projektowe) oraz uniwersalnego języka projektowania UML
 - Języki obiektowe oraz imperatywne wzajemnie się uzupełniają i razem stanowią największą grupę spośród języków programowania
 - Czysto obiektowe języki nie zdobyły szerokiej popularności ze względu na wąski obszar zastosowań i są rozwijane głównie przez środowiska akademickie

6. Paradygmat programowania funkcyjnego

- Koncepcja paradygmatu programowania funkcyjnego polega na traktowaniu programu jako łańcucha wywołań funkcji
 - Każda funkcja (podprogram) przyjmuje dane wejściowe oraz zwraca wynik swego działania
 - Dane wyjściowe funkcji mogą być danymi wejściowymi dla następnej funkcji



6. Paradygmat programowania funkcyjnego (c.d.)

- Proces programowania w języku funkcyjnym polega na konstruowaniu programu z funkcji
 - $F(x,y,z) = F4(F3(F1(x),F2(y)),z)$
- W programie funkcyjnym nie ma zmiennych (!)
- Obliczenia powtarzające się (konstrukcje pętli w imperatywnych językach) są realizowane poprzez rekurencję

6. Paradygmat programowania funkcyjnego (c.d.)

- Rekurencja – wywołanie podprogramu przez samego siebie
 - Przykład: obliczenie wartości silni ($n!$)
 - a) tradycyjnie: liczymy iloraz $1 * 2 * 3 .. * n$ wykorzystując instrukcję pętli (FOR, WHILE itp.)
 - b) poprzez rekurencję: ciało funkcji $F(n)$ zawiera instrukcje:
 - Jeśli $n = 0$, zwracamy jako wynik obliczeń 1
 - Jeśli $n > 0$, zwracamy jako wynik $n * F(n-1)$
 - Rekurencja jest powszechnie stosowana w imperatywnych językach programowania

6. Paradygmat programowania funkcyjnego (c.d.)

- Jedną z najbardziej istotnych cech funkcyjnych języków programowania jest możliwość **matematycznego udowodnienia poprawności algorytmu**
 - Co niewątpliwie przyczyniło się do wzrostu popularności języka w środowisku naukowym
- Pozwala to m. innymi na budowanie programów niezawodnych, działanie którym może być zweryfikowane za pomocą automatycznych testów
 - cecha niedostępna np. dla obiektowych języków

6. Paradygmat programowania funkcyjnego (c.d.)

- Język LISP (1959)
 - Drugi (po Fortranie) język programowania wysokiego poziomu
 - Powstał jako język przetwarzania danych tekstowych (*ang.* LISt Processing)
 - Istotną cechą języka jest zdolność programu do modyfikacji własnego kodu
 - Obecna wersja języka nosi nazwę *Common Lisp* i posiada również możliwości imperatywnego oraz obiektowego programowania

6. Paradygmat programowania funkcyjnego (c.d.)

- Język LISP (c.d.)
 - Obecnie jest popularnym językiem skryptowym, wykorzystywanym do automatyzacji wykonania poleceń m. innymi:
 - w systemie projektowania AutoCAD
 - w edytorze tekstowym Emacs
 - w edytorze graficznym GIMP
 - Osobno należy wyróżnić stosowanie klasycznej wersji LISP w systemach telekomunikacji przy obróbce sygnałów ze względu na szerokie możliwości równoległego programowania

6. Paradygmat programowania funkcyjnego (c.d.)

- Podsumowanie
 - Programowanie funkcyjne polega na przedstawieniu programu jako sekwencji wywołania funkcji (podprogramów), każdy z których może wywoływać kolejne funkcje
 - Zalety paradygmatu:
 - Stabilność i odporność na błędy
 - Szybkość wykonania obliczeń
 - Możliwość automatycznego testowania programu
 - Wady:
 - Ograniczony zakres zastosowania
 - Złożoność programowania

7. Paradygmat programowania logicznego

- Programowanie logiczne (*syn.* Programowanie w logice) – programowanie mające na celu udowodnienie pewnego postulatu w oparciu o fakty wejściowe
 - Przykład:
 - Dane wejściowe
 - A jest większe od B
 - B jest większe od C
 - Dane wyjściowe:
 - Czy A jest większe od C ? (tak / nie)

7. Paradygmat programowania logicznego (c.d.)

- Proces programowania w języku logicznym polega na sformułowaniu **przesłanek**, na podstawie których komputer powinien wydać odpowiedź na zadane pytanie
- Sposobem rozwiązywania problemu jest zbudowanie odpowiedniej struktury (drzewo, graf, tabela itp.) uwzględniającej wszystkie możliwe rozwiązania oraz zawierającej odpowiedź na pytanie
 - Problem równie dobrze może nie znaleźć rozwiązania

7. Paradygmat programowania logicznego (c.d.)

- Język PROLOG (koniec lat 70.)
 - Jest jedyną praktyczną realizacją paradygmatu logicznego programowania
 - Należy do kategorii języków deklaratywnych
 - Programista definiuje co należy zrobić, komputer decyduje o tym jak to należy zrobić
 - W latach 80. uważany za język programowania, który wskaże kierunek rozwoju języków 5 generacji (sztuczna inteligencja)
 - Bardzo popularny w ZSRR oraz Japonii

7. Paradygmat programowania logicznego (c.d.)

- Język PROLOG (c.d.)
 - Ciekawą cechą języka jest zdolność do zbierania oraz przechowania informacji, tworząc tym samym bazę wiedzy
 - Pozwala to na tworzenie systemów rozwijających się
 - Do wad języka należy odnieść
 - trudność programowania
 - Problemy z wydajnością oraz optymalizacją
 - Perspektywy rozwoju:
 - Standard Web 3.0 wykorzystuje do obróbki rozproszonych danych niektóre z koncepcji Prolog

8. Pochodne paradygmaty programowania

8.1. Paradygmat programowania strukturalnego

- Jest podzbiorem **imperatywnego paradygmatu** programowania
- Powstał jako koncepcja programowania w klarownym stylu, produkującego czytelny, łatwy w utrzymaniu i rozwoju kod źródłowy
- Pomysł należy do holenderskiego matematyka E.Dijkstrę w latach 70.
- Wspierany przez N.Wirtha, twórcę języka programowania Pascal (co znalazło odzwierciedlenie w składni języka)

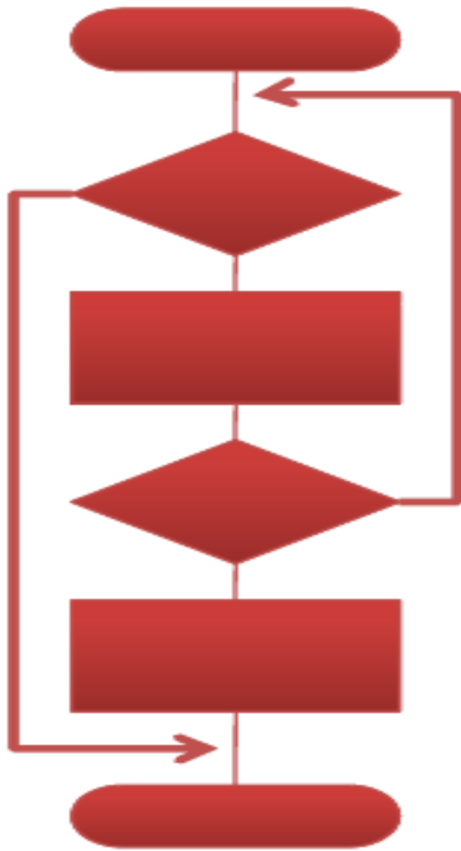
8.1. Paradygmat programowania strukturalnego (c.d.)

- Koncepcja polega na ograniczeniu stosowania niektórych konstrukcji języków imperatywnych, wnoszących chaos w kod źródłowy programu:
 - Nie można używać instrukcji GOTO przejścia bezpośredniego do dowolnego miejsca w programie
 - Każdy podprogram może mieć tylko jeden punkt wyjścia
 - Zabroniono przerwanie wykonania instrukcji powtarzania (pętli)

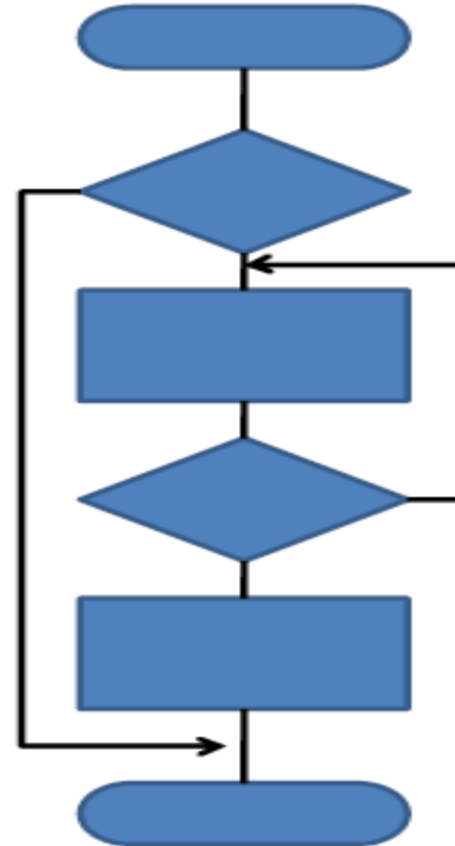
8.1. Paradygmat programowania strukturalnego (c.d.)

- Koncepcja (c.d.)
 - Cały program składa się z mniejszych bloków
 - Każdy blok powinien spełniać warunki strukturalności
 - Wówczas cały program również je spełnia
- Każdy program nie spełniający warunki strukturalności może zostać przekształcony do postaci strukturalnej

8.1. Paradygmat programowania strukturalnego (c.d.)



Program nie spełniający
warunki strukturalności



Program
strukturalny

8.1. Paradygmat programowania strukturalnego (c.d.)

- W latach 70-80 toczyła się ostra dyskusja pomiędzy zwolennikami a przeciwnikami podejścia strukturalnego
 - Typowym językiem reprezentującym podejście strukturalne był język Pascal
 - Po drugiej stronie popularność zdobywał język Basic, programowanie w którym bez instrukcji GOTO nie jest wyobrażalne
 - „Studenci, programujący w języku Basic, już nigdy się nie nauczą programować porządnie”
(E.Dijkstra)

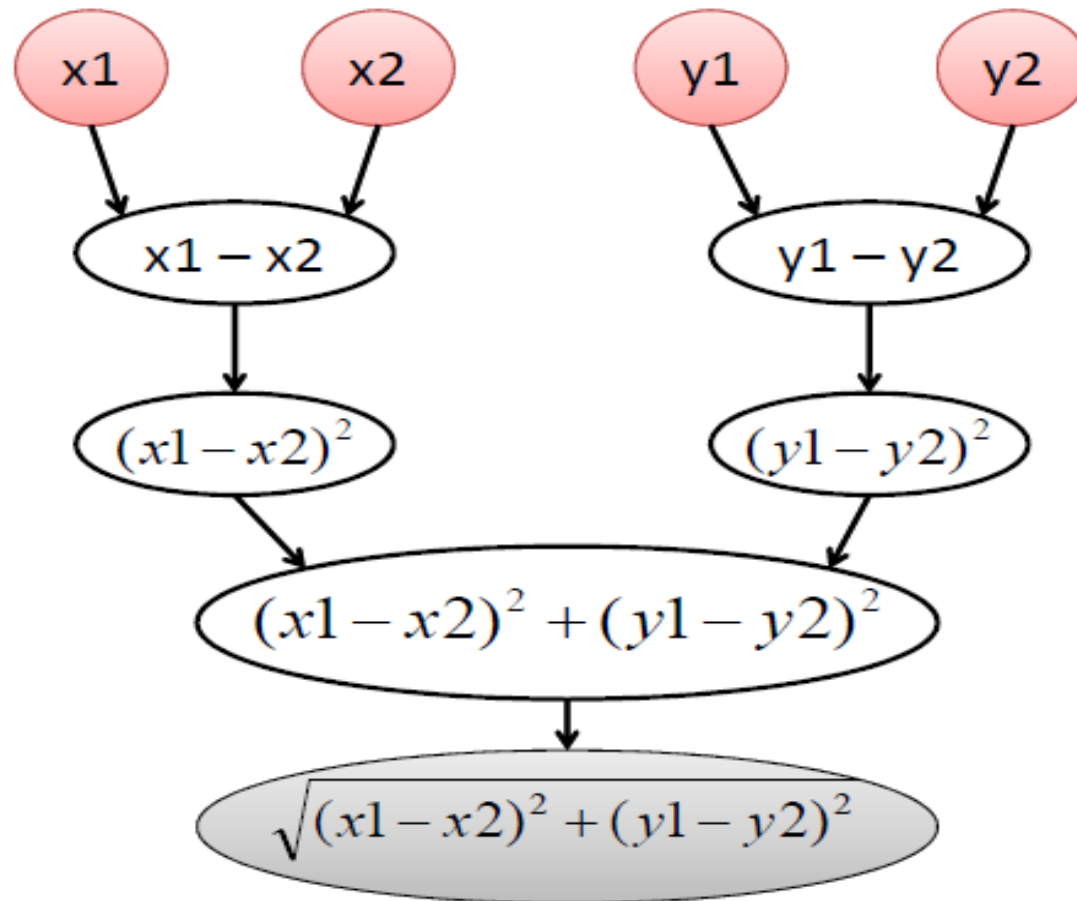
8.1. Paradygmat programowania strukturalnego (c.d.)

- Obecny stan rzeczy:
 - Nie odrzucając zalet programowania strukturalnego, jego zwolennicy musieli się pogodzić z tym, że w pewnych sytuacjach użycie instrukcji GOTO/BREAK/NEXT itp. jest uzasadnione i znacząco skraca kod programu
 - Zwłaszcza jeśli tego się nie nadużywa
 - Każdy współczesny imperatywny język programowania (w tym również Pascal) posiada podobne instrukcje, lecz o konieczności ich użycia decyduje programista

8.2. Paradygmat programowania sterowanego przepływem danych

- Tradycyjne komputery zbudowane w oparciu o architekturę von Neumanna zakładają pewien priorytet algorytmów nad danymi
 - Komputer jest sterowany przez instrukcję
 - Program jest aktywny, dane są pasywne
- Koncepcja **komputera sterowanego przez dane** pojawiła się już w latach 60, lecz praktycznej realizacji w czystej postaci nie doczekała się
- Za to koncepcja **programowania sterowanego przepływem danych** odniosła sukces

8.2. Paradygmat programowania sterowanego przepływem danych (c.d.)



Algorytm obliczenia odległości pomiędzy punktami X i Y
zapisany w postaci przepływów danych

8.2. Paradygmat programowania sterowanego przepływem danych (c.d.)

- Algorytmy oparte na przepływach danych idealnie nadają się do przetwarzania równoległego!
 - Algorytm obliczenia odległości pomiędzy dwoma punktami może być wykonany w 4 kroki, zaś jego klasyczna realizacja wymaga 6 kroków
- Język programowania SISAL wykorzystywany jest do programowania superkomputerów Cray
- Najbardziej popularnym zastosowaniem są jednak ... arkusze kalkulacyjne!

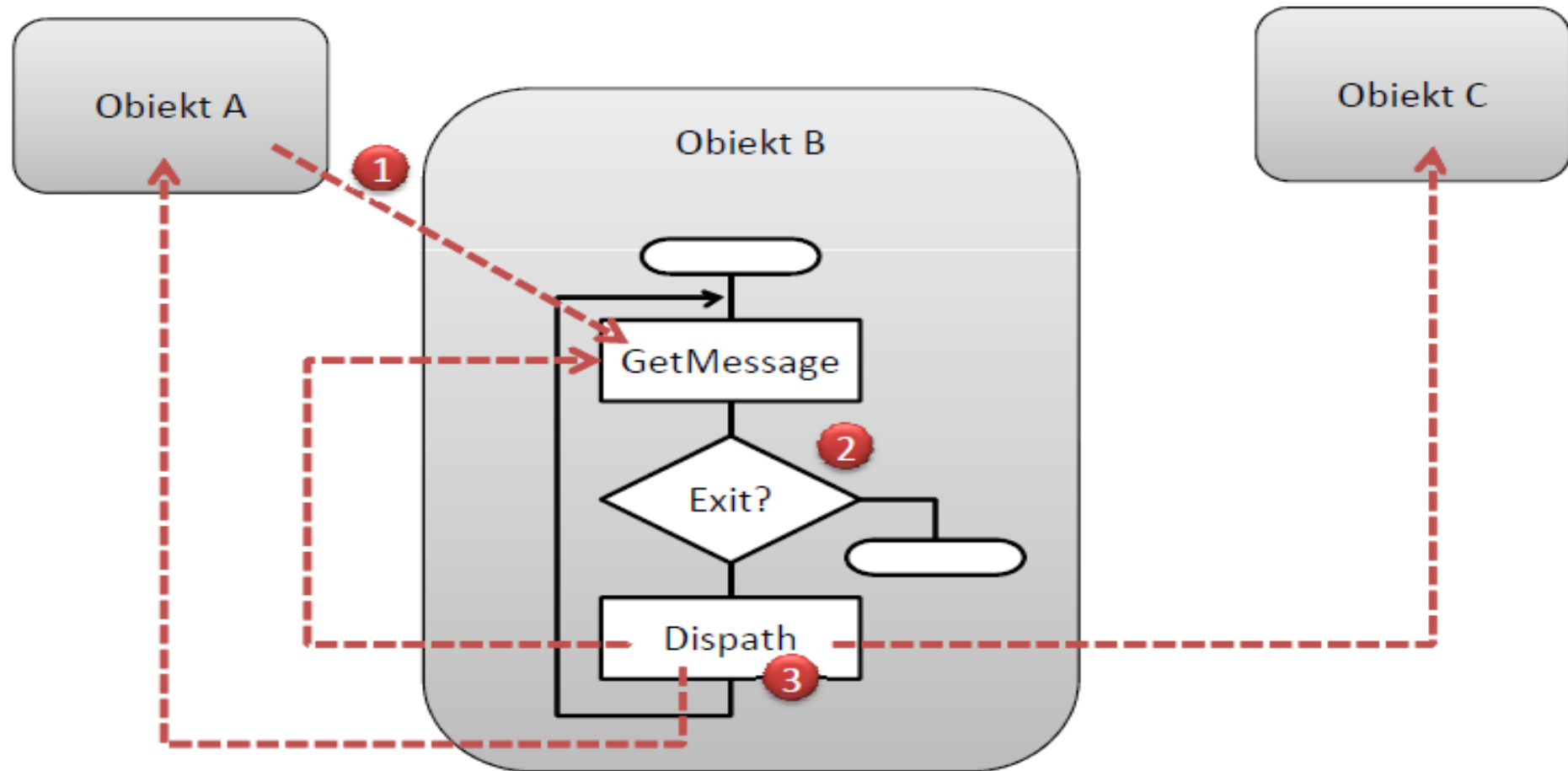
8.3. Paradygmat programowania zdarzeniowego

- Jest rozszerzeniem **paradygmatu obiektowego**
 - W programowaniu imperatywnym narzucanie sekwencji wykonania instrukcji przez algorytm było zjawiskiem naturalnym
 - Paradygmat programowania obiektowego pozwolił na budowę bardzo skomplikowanych systemów, złożonych z tysięcy obiektów, każdy z których ma własne zachowanie
 - Centralne sterowanie takim systemem powinno uwzględniać indywidualne cechy każdego obiektu, co przy rosnącej ilości klas jest bardzo skomplikowane

8.3. Paradygmat programowania zdarzeniowego (c.d.)

- Zrezygnowano z pomysłu centralnego sterowania programem na rzecz wielu mniejszych procedur **obsługi zdarzeń**
- Zdarzenia mogą być wywołane przez:
 - Interakcje programu z użytkownikiem
 - Ruch myszką
 - Wciśnięcie klawisza
 - Włożenie płyty CD itp.
 - Wydarzenie wewnątrz samego systemu
 - Zamknięcie okna, otwarcie pliku, błąd w programie
 - itp.

8.3. Paradygmat programowania zdarzeniowego (c.d.)



Uproszczony schemat obsługi zdarzeń w programie Windows

8.3. Paradygmat programowania zdarzeniowego (c.d.)

- Działanie programu opartego na zdarzeniach
 - Użytkownik (bądź system) dokonuje pewnej czynności (np. zamyka okno dialogowe programu klikając myszką w pole [x])
 - System operacyjny wysyła **komunikat** o określonym kodzie oraz parametrach
 - W tym przypadku MOUSE_CLICK ze wskazaniem współrzędnych pozycji kursora
 - Każda aplikacja (aktywna w danym momencie) posiada własną **procedurę obsługi komunikatów**, działającą w tle
 - Nie przeszkadza programowi wykonywać podstawową działalność

8.3. Paradygmat programowania zdarzeniowego (c.d.)

- Działanie programu opartego na zdarzeniach (c.d.)
 - Procedura aplikacji sprawdza typ komunikatu oraz określa czy jest on jej adresowany
 - W tym przykładzie komunikat będzie adresowany aplikacji, jeśli pozycja kursor myszy (parametr komunikatu) należy do jej okna
 - Program wykonuje określona czynność (zależną od rodzaju komunikatu oraz aplikacji)
 - W tym przypadku zamyka okno dialogowe oraz usuwa komunikat z kolejki (niszczenie)
 - Jeśli komunikat nie jest adresowany programowi, jest on ignorowany

8.3. Paradygmat programowania zdarzeniowego (c.d.)

- Zalety programowania zdarzeniowego:
 - Program można podzielić na niezależne fragmenty (klasy), połączone zdarzeniami
 - Procesy obsługi zdarzeń mogą być wykonywane jednocześnie dla wielu obiektów
 - np. dla wszystkich otwartych okien
 - Optymalizuje się wykorzystanie procesora
 - brak komunikatów = brak aktywności aplikacji

8.3. Paradygmat programowania zdarzeniowego (c.d.)

- Wady programowania zdarzeniowego
 - Śledzenie działania programu w czasie rzeczywistym jest bardziej skomplikowane, niż w przypadku klasycznego centralnego sterowania
 - Nie ma możliwości przewidzenia wszystkich ścieżek wykonania programu
 - Pewne kombinacje komunikatów mogą doprowadzić do błędów, np. zawieszenia się aplikacji
 - Nie nadaje się do programowania liniowych algorytmów

8.3. Paradygmat programowania zdarzeniowego (c.d.)

- Zakres zastosowania:
 - Programowanie aplikacji zorientowanych na interakcję z użytkownikiem
 - Aplikacje stosowane dla Windows, Linux
 - Programowanie systemów operacyjnych
 - Zarządzanie wielozadaniowością, procesami
 - Programowanie baz danych
 - Zmiana wartości w tabeli bazy danych generuje komunikat, który może być obsługiwany przez aplikację (np. w celu poinformowania użytkownika o zmianie salda na koncie, przekroczeniu pewnego limitu itp.)

8.3. Paradygmat programowania zdarzeniowego (c.d.)

- Do realizacji programu działającego w oparciu o zdarzenia nie jest wymagany specjalny język, wystarczy posłużyć się dowolnym językiem obiektowym
- Do większości popularnych obiektowych języków programowania są gotowe biblioteki wspomagające programowanie zdarzeniowe
 - Wbudowane wsparcie mają np. Java, .NET, C++ oraz inne popularne języki

8.4. Paradygmat programowania współbieżnego

- Współbieżność – wykonanie **wielu zadań** jednocześnie przez **wiele procesorów**
- Bazuje na:
 - **Przetwarzaniu równoległym** (wykonanie wielu zadań na jednym procesorze)
 - **Przetwarzaniu rozproszonym** (na wielu procesorach)
- Teoria opracowana w latach 60. nie została wówczas zrealizowana ze względu na możliwości techniczne

8.4. Paradygmat programowania współbieżnego (c.d.)

- Sposoby komunikowania się pomiędzy współbieżnymi zadaniami:
 - Poprzez wspólną pamięć (na jednym komputerze)
 - Historycznie pierwszy sposób komunikacji
 - Problem rozgraniczenia jednoczesnego dostępu do wspólnych danych
 - Poprzez komunikaty (sposób uniwersalny)
 - Łatwiejsze programowanie oraz bardziej przewidywalne zachowanie się programów
 - Wymaga wsparcia przez system operacyjny
 - Nie wszystkie algorytmy można rozłożyć na kilka procesów (zadań) wykonywanych równolegle

8.4. Paradygmat programowania współbieżnego (c.d.)

- Głównym problemem programowania współbieżnego jest słabe wsparcie przez języki programowania
 - np. w języku C++ wymaga to sporych umiejętności
- Nowe języki programowania wysokiego poziomu (4,5 generacja) wspierają programowanie współbieżne na poziomie konstrukcji wbudowanych w język
 - Programista nie musi się martwić o bezpieczeństwo i poprawność współbieżnego działania algorytmów
 - Spośród języków uniwersalnych należy wymienić przede wszystkim Java oraz .NET