

## Analiza obiektowa

### Metody wywoływane na rzecz obiektów

- są izolowane od rozważań o przyczynach i skutkach, tj.:
  - Metoda działa tak samo niezależnie od tego, jakie zdarzenie ją wywołało i jaki jest stan obiektów (wystarczy że spełniają ograniczenia)
  - Metoda w swym działaniu nie uwzględnia tego, jakie następne metody wywoła zdarzenie jej zakończenia
- zagnieżdżone struktury operacji. Można je specyfikować jako moduły zawierające różnorodne składniki: ciągi zdarzeń, wywołań i warunków sterujących;

48

## Analiza obiektowa

### Wyzwalacz

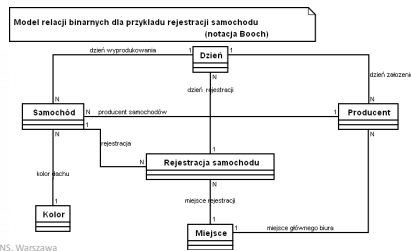
- połączenie zdarzenia i wywołanego procesu
  - Rodzaje wyzwalaczy == reguły wyzwalania
- Wyzwalacz może wyzwalać wiele procesów, które mogą być wykonywane sekwencyjnie lub równolegle

49

## Analiza obiektowa

### Reprezentowanie struktury obiektów

model relacji binarnych (diagram obiektowy) – wyraża związki między typami



50

## Analiza obiektowa

### Reprezentowanie zachowania obiektów

- Maszyny skończone: stany, przejścia między stanami (zależą od zdarzeń).
- Specyfikacje oparte na scenariuszach (diagramy sieci Petriego).
- Specyfikacje decyzyjne wyrażone za pomocą tabeli decyzyjnej i drzewa decyzyjnego.

51

## Analiza obiektowa

### Podejścia do analizy obiektowej - historia

- Martin/Odell
- Booch
- Coad/Yourdon
- Jacobson
- Shlaer/Mellor
- Rumbaugh i in.

Źródło: „Podstawy metod obiektowych”  
J. Martin, J.J. Odell, WNT, 1997

52

## Analiza obiektowa

	Martin/Odell	Booch	Coad/Yourdon	Jacobson	Shlaer/Mellor	Rumbaugh i in.
Typ obiektowy		Klasa	Klasa i obiekt	Typ obiektowy	Obiekt	Klasa
Obiekt		Obiekt	Obiekt	Obiekt, Egzemplarz	Egzemplarz	Obiekt
Typ związków		Używanie relacji	Związek obiektów		Związek	Powiązanie
Odwzorowanie		Rola		Związek znajomości	Związek, Odwzorowanie	Rola
Uogólnianie, Specjalizowanie		Dziedziczenie	Gen-Spec	Dziedziczy	Podtyp-Nadtyp	Uogólnianie
Składanie		Zawieranie	Część-Całość			Agregacja

53

## Analiza obiektowa

Odwzorowania: Czytając od lewej do prawej	A jest zawsze związane z jednym B	A jest zawsze związane z jednym lub wieloma B	A jest zawsze związane z żadnym lub jednym B	A jest zawsze zwią- zane z żadnym, jednym lub wieloma B
Martin/Odell				
Booch (drugie wydanie)				
Coad/Yourdon				
Jacobson (jednokierunkowe)				
Shlaer/Mellor				
Rumbaugh i in. <small>© UKSW, WMP, SNS, Warszawa</small>				

54

## Analiza obiektowa

Analizę obiektową wg metody Coad/Yourdon:

5 głównych czynności:

1. znajdowanie klas i obiektów
2. identyfikacja struktur
3. identyfikacja tematów
4. definiowania atrybutów
5. definiowania usług

model analizy obiektowej zawiera 5 warstw:

1. warstwa tematów
2. warstwa klas i obiektów
3. warstwa struktury
4. warstwa atrybutów
5. warstwa usług

© UKSW, WMP, SNS, Warszawa

55

## Analiza obiektowa

Analizę obiektową wg metody OMT (Rumbaugh):  
3 części składowe modelu, pokazujące różne jego aspekty:

**Model Obiektów (OMT Object Model):** statyczny obraz struktury modelu

- klasy
- atrybuty
- operacje
- relacje między klasami i instancjami

**Model Dynamiczny (OMT Dynamic Model):** współdziałanie obiektów (powiązania wyznaczone przez komunikaty).

Tu mieszczą się różne diagramy pokazujące przepływ sterowania, także ograniczenia i warunki na wartości atrybutów.

**Model Funkcyjny (OMT Functional Model)**

specyfikacja operacji jako funkcji przekształcających wejście na wyjście, warunki poprawności (asercje).

© UKSW, WMP, SNS, Warszawa

56

56

## PROJEKTOWANIE OBIEKTOWE

© UKSW, WMP, SNS, Warszawa

57

57

## Projektowanie obiektowe

- Za budowę modelu odpowiedzialni są analitycy, którzy mają kontakt z klientem
- Analitycy uzgadniają z inżynierami oprogramowania poprawność budowanego modelu, dostosowując go do możliwości narzędzia programistycznego, które zostanie użyte podczas fazy implementacji
- Rola inżyniera oprogramowania:
  - jednocześnie znać kilka metodyk projektowania lub przynajmniej zdawać sobie sprawę z różnic, jakie między nimi występują
  - potrafić przekładać niejasne żądania klientów na precyzyjne specyfikacje
  - umieć rozmawiać z zamawiającym oprogramowanie posługując się terminami z dziedziny aplikacji, a nie terminami informatycznymi

© UKSW, WMP, SNS, Warszawa

58

58

## Projektowanie obiektowe

Obszary wiedzy inżyniera oprogramowania

- Inżynieria oprogramowania, informatyka
- Znajomość środowiska programistycznego (język programowania, platforma systemowa, środowisko projektowe)
- Znajomość organizacji (przedsiębiorstw, administracji publicznej)
- Znajomość dziedziny aplikacji (np. awioniki, operacji bankowych, logistyki, chemii, itp.)
- Psychologia i socjologia (etyka, komunikatywność, gospodarowanie czasem, itp.)

© UKSW, WMP, SNS, Warszawa

59

59

## Projektowanie obiektowe

- Podczas analizy wzajemne oddziaływania między obiektami są reprezentowane jako zdarzenia.
- Projektant ma za zadanie wybrać rodzaj przepływu sterowania, jaki będzie implementowany w projekcie.
- Zewnętrzne sterowanie:
  - Sekwencyjne proceduralne
  - Sekwencyjne zdarzeniowe
  - Współbieżne
- Wewnętrzne sterowanie

© UKSW, WMP, SNS, Warszawa

60

60

## Projektowanie obiektowe

### Systemy sterowane procedurami

- Sterowanie proceduralne umiejscowione jest w kodzie programu:
  - zaczyna się na wierzchołku hierarchii podprogramów i przez wywołania podprogramów przechodzi do niższych poziomów, albo
  - jeden z komponentów jest menedżerem systemu: steruje rozpoczynaniem, zatrzymywaniem i koordynacją innych

© UKSW, WMP, SNS, Warszawa

61

61

## Projektowanie obiektowe

### Systemy sterowane procedurami

- Procedury generują żądania wprowadzenia danych i czekają na ich pojawienie się. Po wprowadzeniu danej sterowanie przekazywane jest zwrótnie do procedury. Wartość licznika rozkazów, zmiennych lokalnych i stosu, na którym odkładane są wywołania procedur, definiują stan systemu.
- Można stosować tylko, gdy schemat zmiany stanów wykazuje regularne następstwo zdarzeń wejściowych i wyjściowych.
- Trudno jest stworzyć elastyczne interfejsy użytkownika.

© UKSW, WMP, SNS, Warszawa

62

62

## Projektowanie obiektowe

### Systemy sterowane zdarzeniami

- Sterowanie zdarzeniowe rezyduje wewnątrz programu koordynującego lub monitorującego.
- Procedury aplikacji są przywiązane do zdarzeń i są wywoływane, gdy pojawiają się odpowiadające im zdarzenia, tj. podsystemy same decydują, które zdarzenia są dla nich interesujące.
- Odwołania do programu koordynującego pozwalają na wprowadzanie danych wejściowych oraz wysyłanie danych wyjściowych (procedury nie zachowują sterowania np. czekając na dane).

© UKSW, WMP, SNS, Warszawa

63

63

## Projektowanie obiektowe

### Sterowanie wewnętrzne

#### Proceduralne

•przekazywanie sterowania (wywołania procedur, wywołania między zadaniowe) jest kontrolowane przez program i jest poza kontrolą użytkownika

#### Zdarzeniowe

•wzajemne oddziaływania między obiektami są takie same jak zewnętrzne, za wyjątkiem braku oczekiwań na zdarzenia: obiekty mogą wymuszać odpowiedzi innych obiektów

### Sterowanie zewnętrzne

#### Proceduralne

•Jest kontrolowane przez użytkownika (trudne w realizacji)

#### Zdarzeniowe

•Oczekiwanie na zdarzenia, które są pod kontrolą użytkownika (dość intuicyjne)

© UKSW, WMP, SNS, Warszawa

64

64

## Projektowanie obiektowe

- W procesie rozwoju oprogramowania:
- Faza analizy – opisuje wymagania
- Faza projektu – definiuje strategię rozwiązania, tj. projekt klas obiektów przez:
  - Zdefiniowanie interfejsów
  - Zdefiniowanie algorytmów użytych do zaimplementowania operacji
  - Optymalizowanie struktur danych i algorytmów

W fazie projektu wprowadza się również dodatkowe „wewnętrzne” klasy obiektów niezbędne dla implementacji

© UKSW, WMP, SNS, Warszawa

65

65

## Projektowanie obiektowe

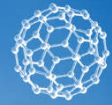
### Kolejność tworzenia projektu obiektów

1. Wpisać do klas obiektów wszystkie operacje z modeli analitycznych.
2. Zaprojektować algorytmy implementujące operacje, biorąc pod uwagę złożoność obliczeniową, łatwość implementacji, zrozumiałość i elastyczność.
3. Zoptymalizować ścieżki dostępu do danych.
4. Zaimplementować sterowanie, które odwzoruje model dynamiczny (system sterowany procedurami lub zdarzeniami).
5. Zmienić strukturę klas aby rozszerzyć związki dziedziczenia.
6. Zaprojektować implementacje związków klas (nadmiarowe związki gmatwiają analizę, ale na etapie implementacji mogą poprawić wydajność i uprościć realizację niektórych procesów).
7. Określić reprezentację obiektu.
8. Umieścić klasy i związki w modułach.

© UKSW, WMP, SNS, Warszawa

66

66



## PROGRAMOWANIE OBIEKTOWE

© UKSW, WMP, SNS, Warszawa

67

67

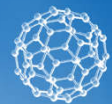
## Programowanie obiektowe

- Język programowania C++ pozwala na obiektową realizację projektu systemu
- Język C++ jest nadzbiorem języka C  
(*prawie..*)
- Występują różnice między C i C++ w obszarze właściwości programowania strukturalnego, jednak program napisany wg reguł C w większości przypadków zostanie zaakceptowany przez kompilator C++.

© UKSW, WMP, SNS, Warszawa

68

68



## C++ - KLASY

© UKSW, WMP, SNS, Warszawa

69

69

## C++ - klasy

- C++ jest językiem programowania obiektowego – pozwala na budowaniu programów działających na obiektach.
- Obiekty, to egzemplarze/instancje utworzone wg opisu zawartego w definicji typu.
- Typy obiektowe w C++ są rozszerzeniem typów używanych do tworzenia zmiennych w C
- Rozszerzenie polega na:
  - dodaniu reguł dostępu do składników/atributów/pól obiektu
  - dodaniu właściwości usługowych – metod, które mogą wykonywać działania wykorzystując składniki obiektu
  - dodaniu usługi inicjalizacji przy tworzeniu i usługi kończenia przy usuwaniu obiektów

© UKSW, WMP, SNS, Warszawa

70

70

## C++ - klasy

Deklarowanie klasy przypomina deklarowanie typu strukturalnego:

```
struct osoba1                class osoba2
{
    char imie[20];
    char nazwisko[30];
    int wiek;
};

{
    public :
    char imie[20];
    char nazwisko[30];
    int wiek;
};
```

© UKSW, WMP, SNS, Warszawa

71

71

## C++ - klasy

Deklaracja instancji:

```
osoba1 PanX;  
osoba2 PanY;
```

Odwołanie:

```
strcpy(PanX.imie, "Luke");  
strcpy(PanX.nazwisko, "Hobbs");  
PanX.wiek = 50;  
strcpy(PanY.imie, "Deckard");  
strcpy(PanY.nazwisko, "Shaw");  
PanY.wiek = 54;
```

*Czym różnią się te dwa obiekty?*

© UKSW, WMP, SNS, Warszawa

72

72

## C++ - klasy

Projektując klasę należy odpowiedzieć sobie na kilka pytań:

1. Co charakteryzuje stan wewnętrzny obiektów i jak go reprezentuje?
2. Czy można definiować jakieś niezmienniki stanu wewnętrznego obiektów?
3. Które z cech obiektu można udostępniać publicznie, a które powinny być kontrolowane?
4. Jak będą tworzone i inicjalizowane obiekty; czy dopuszczamy istnienie obiektów z nieokreślonym stanem wewnętrznym?
5. Czy likwidacja obiektu wymaga czynności porządkowych?
6. Jakie operacje będą wykonywane na obiektach?
7. Jak program ma korzystać z definicji klasy?

© UKSW, WMP, SNS, Warszawa

73

73

## C++ - klasy

Odpowiedzi dla klasy reprezentującej *butelkę mleka*:

1. *Co charakteryzuje stan wewnętrzny obiektów i jak go reprezentuje?*  
- Maksymalna pojemność butelki i jej zawartość.
2. *Czy można definiować jakieś niezmienniki stanu wewnętrznego obiektów?*  
- Zawartość niemniejsza od zera i nie większa od pojemności.
3. *Które z cech obiektu można udostępniać publicznie, a które powinny być kontrolowane?*  
- Pojemność musi pozostać niezmienna, zawartość może być modyfikowana tylko tak, aby zachować niezmienniki.
4. *Jak będą tworzone i inicjalizowane obiekty; czy dopuszczamy istnienie obiektów z nieokreślonym stanem wewnętrznym?*  
- Tworzony obiekt – butelka mleka musi być na początku pusta (zawartość równa zero). Nie ma butelek z nieokreślonym stanem wewnętrznym.

© UKSW, WMP, SNS, Warszawa

74

74

## C++ - klasy

Odpowiedzi dla klasy reprezentującej *butelkę mleka*:

5. *Czy likwidacja obiektu wymaga czynności porządkowych?*  
- Jeżeli w butelce jest mleko, to należy najpierw je wylać.
6. *Jakie operacje będą wykonywane na obiektach?*  
- Wlać mleko i wylać mleko. Można jeszcze umyć, wstawić kwiatki ..
7. *Jak program ma korzystać z definicji klasy?*  
Może tworzyć obiekty, a może to być tylko klasa bazowa (o tym później).

© UKSW, WMP, SNS, Warszawa

75

75

## C++ - klasy

Prawa dostępu:

- **public** - składniki są dostępne wszędzie gdzie jest użyty obiekt danej klasy
- **private** - składniki są dostępne tylko wewnątrz danej klasy. W praktyce oznacza to, że tylko metody danej klasy mogą czytać i zapisywać tego typu składniki
- **protected** – w porównaniu z *private*, zakres widoczności jest poszerzony o klasy wywodzące się z aktualnej klasy, tzw. klasy dziedziczące (to zostanie wyjaśnione przy okazji omawiania dziedziczenia)

© UKSW, WMP, SNS, Warszawa

*Po co komu prawa dostępu?*

76

76

## C++ - klasy

kapsułkowanie/hermetyzacja

ukrywaniu pewnych danych składowych lub metod obiektów danej klasy tak, aby były one dostępne tylko metodom wewnętrznym danej klasy lub funkcjom z nią zaprzyjaźnionym (*o funkcjach zaprzyjaźnionych będzie jeszcze mowa*).

Kapsułkowanie uodparnia tworzony model systemu na błędy nieprawidłowego zapisu lub niepowołanego odczytu składowych klasy (*kto te błędy może popełnić? – nieumiejętny programista*)

Kapsułkowanie nie ma sensu w przypadku struktur w C, bo struktury nie mają własnych metod, stąd kapsułkowanie zamknęłoby całkowicie dostęp do składowych struktury

© UKSW, WMP, SNS, Warszawa

77

77

## C++ - klasy

- **Abstrakcja proceduralna i hermetyzacja** to fundamentalne założenia modelu obiektowego: na danym poziomie szczegółowości użytkownikowi (programiście) udostępnia się tylko te właściwości obiektu, które są istotne z punktu widzenia jego potrzeb na tym poziomie szczegółowości.
- Pozostałe właściwości, choć istnieją i są konieczne dla poprawnego funkcjonowania obiektu, mogą pozostać dla użytkownika całkowicie nieznanymi.

*Przykład z życia: znajomość budowy silnika samochodowego i swobodny dostęp do jego elementów nie są konieczne do poprawnego korzystania z samochodu*

© UKSW, WMP, SNS, Warszawa

78

78

## C++ - klasy

### Różnice między `struct` i `class`

- Domyślnie wszystkie składowe obiektu typu `struct` są publicznie dostępne – można ich używać (odczyt/zapis, wywołanie) wszędzie tam, gdzie widoczna jest definicja struktury tego typu
- Domyślnie wszystkie składowe obiektu typu `class` są publicznie niedostępne. Aby stały się dostępne, musimy je zadeklarować jako *dostępne*. Kod deklaracji klasy jest podzielony na trzy rodzaje sekcji. Przynależność składowych do sekcji określa prawa dostępu.
- Prawa dostępu do składowych klasy dzielą się na:
  - typu **public**
  - typu **private**
  - typu **protected**

© UKSW, WMP, SNS, Warszawa

79

79

## C++ - klasy

```
class JakasKlasa {
public:
    int s1;
    int s2;
private:
    int s3;
    int s4;
    int s5;
protected:
    int s6;
};

class JakasInnaKlasa {
    int d1; // domyślnie: private
public:
    int d2;
protected:
    int d3;
private:
    int d4;
public:
    int d5;
};
```

(Tak też można, ale większy bałagan w kodzie..)

© UKSW, WMP, SNS, Warszawa

80

80

## C++ - klasy

### Metody

Skoro składowe klasy mogą być niedostępne z zewnątrz (**private**), to po co je w ogóle deklarować – i tak nikt z nich nie skorzysta (nie będzie do nich zapisywał, ani też je odczytywał)?

- Składowymi klasy mogą być nie tylko zmienne (pola), ale również funkcje. Takie funkcje nazywane są **metodami klasy**.
- Metody podlegają takim samym ograniczeniom praw dostępu co pola.
- Wszystkie metody będące składowymi danej klasy mają prawa dostępu do pól zadeklarowanych jako *private* i *protected* w tej klasie.

© UKSW, WMP, SNS, Warszawa

81

81

## C++ - klasy

Przykład:

```
class water_temp {
    double t; // składowa private
public:
    double get_temp() { // metoda #1 - składowa klasy
        return t; // zdefiniowana w klasie -> jest inline
    } // (traktowana tak jak funkcja inline)
    double set_temp(double nt); // metoda #2 - składowa klasy
};

double water_temp::set_temp(double nt) {
    if (nt < 100 && nt > 0) // metoda #2 zdefiniowana poza klasą
        return (t = nt); // nie jest traktowana przez
    else // kompilator jako inline
        return t;
}
```

© UKSW, WMP, SNS, Warszawa

82

82

## C++ - klasy

Przykład:

Mam pustą szklankę  
oraz gąbkę.



Nasączam gąbkę 100 ml wody, po czym trzy razy wyciskam wodę z gąbki do szklanki. Jedno wyciśnięcie powoduje, że z gąbki ubywa połowa zawartości wody.

*Ile wody jest teraz w szklance?*

Visual Studio  
Program #5

© UKSW, WMP, SNS, Warszawa

83

83