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

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.)

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

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

61

Projektowanie obiektowe

Systemy sterowane procedurami

- Procedury generują żądania wprowadzenia danych i czekają na ich pojawienie się. Po wprowadzeniu danej sterowanie przekazywane jest zwrotnie 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.

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).

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)

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

Projektowanie obiektowe

Kolejność tworzenia projektu obiektów

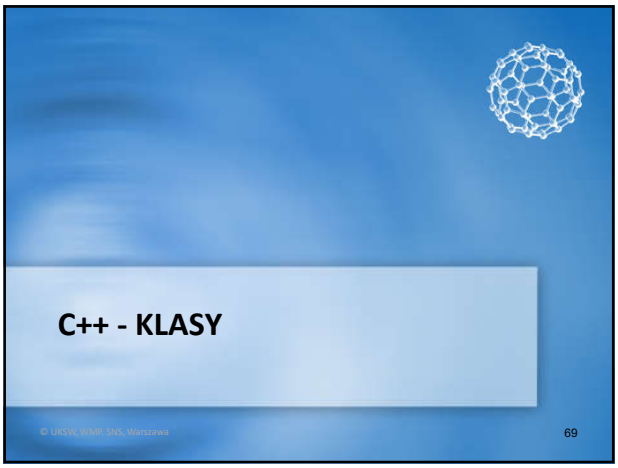
1. Wpisać do klas obiektów wszystkie operacje z modeli analitycznych.
2. Zaprojektować algorytmy implementujące operacje, biorąc pod uwagę złożoność obliczeniową, łatwość implementacji, zrozumiałość i elastyczność.
3. Zoptymalizować ścieżki dostępu do danych.
4. Zaimplementować sterowanie, które odwzoruje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

PROGRAMOWANIE OBIEKTOWE

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++.



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/atrybutów/pól obiektu
  - dodaniu właściwości usługowych – metod, które mogą wykonywać działania wykorzystując składniki obiektu
  - dodaniu usługi inicjalizacji przy tworzeniu i usługi kończenia przy usuwaniu obiektów

70

C++ - klasy

Deklarowanie klasy przypomina deklarowanie typu strukturalnego:

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

```
class osoba2
{
public :
  char imie[20];
  char nazwisko[30];
  int wiek;
};
```


71

C++ - klasy

Deklaracja instancji:

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

Odwołanie:

```
strcpy(PanX.imie,"Mel");
strcpy(PanX.nazwisko, "Gibson");
PanX.wiek = 58;
strcpy(PanY.imie, "Danny");
strcpy(PanY.nazwisko, "Glover");
PanY.wiek = 68;
```

Czym różnią się te dwa obiekty?

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 inicjowane 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?

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 inicjowane 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 niekreślonym stanem wewnętrznym.

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).

C++ - klasy

Prawa dostępu:

- **public** - składniki są widoczne wszędzie gdzie jest użyty obiekt danej klasy
- **private** - składniki są widoczne tylko wewnątrz danej klasy. W praktyce oznacza to, że tylko funkcje 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*)

Po co komu prawa dostępu?

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

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 nieznane.

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

C++ - klasy

Różnice miedzy `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**

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..*)

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 samych ograniczeniom praw dostępu co zmienne.
- Wszystkie metody będące składowymi danej klasy mają prawa dostępu do zmiennych zadeklarowanych jako *private* i *protected* w tej klasie.

C++ - klasy

Przykład:

```
class water_temp {
  double t;               // składowa private
public:
  double get_temp() {     // metoda – składowa klasy (inline)
    return t;
  }
  double set_temp(double nt);     // metoda – składowa klasy
};

double water_temp::set_temp(double nt) {
  if (nt<100 && nt>0)
    return (t = nt);
  else
    return t;
}
```


C++ - klasy

Przykład:

Mam pustą szklankę oraz gąbkę.

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

Ile wody jest teraz w szklance?

Visual Studio
Program #5

C++ - klasy

Związki z podręcznikami o modelowaniu obiektowym

- Struktury i klasy zawierające metody nazywane są *abstrakcyjnymi typami danych*
- zmienne utworzone za pomocą abstrakcyjnych typów danych – *obiektami*
- wywoływanie metod składowych obiektów określa się mianem *wysyłania do nich komunikatów*
- rezultatem wysyłania komunikatów jest uruchamianie procesów działających na atrybutach obiektów
- wysyłanie komunikatów do obiektów to podstawowe działanie związane z programowaniem obiektowym

C++ - klasy

Zmienne:

Typy zmiennych są określone w specyfikacji języka C i C++. Są to np.: `int`, `double`, `char`.

Zdefiniować można sobie typy reprezentujące struktury oraz stałe wyliczeniowe.

Zmienne to instancje.

W kodzie programu możemy do zmiennej zapisać wartość lub odczytać ze zmiennej wartość. Wobec zmiennej strukturalnej możemy to zrobić dla każdej z jej składowych.

Obiekty:

Typy obiektów są definiowane wyłącznie przez użytkownika. Każda klasa napisana przez programistę jest nowym typem obiektowym.

Obiekty to instancje.

W kodzie programu możemy do pól składowych obiektu zapisać wartość lub odczytać z nich wartość. Możemy też wywołać metodę należącą do obiektu.

C++ - klasy

- Definiując klasę (typ obiektowy) możemy w niej zadeklarować jedną lub więcej metod.
- Deklarując obiekt, możemy te metody wywołać na rzecz tego obiektu.
- Mając kilka obiektów tego samego typu, na rzecz każdego z nich możemy wywołać tę samą metodę. Ale.. jej działanie może być różne dla każdego z obiektów, ponieważ pola składowe tych obiektów nie muszą być zapisane tymi samymi wartościami.

C++ - klasy

Słownik programisty:

1. klasa
2. obiekt (instancja)
3. składowe obiektu: pola i metody
4. definicja vs. deklaracja
5. wywołanie metody na rzecz obiektu
6. kapsułkowanie (hermetyzacja)

C++ - klasy

Struktury jako typy abstrakcyjne

- W C++ struktury też mogą mieć składowe metody.
- Te metody są domyślnie deklarowane jako public.
  Przykład:

```
struct Stru {
        int a;
        void prin(int b) {
             printf("%i",c);
             };
};
```


C++ - klasy

Co jeszcze mogą mieć struktury w C++?

Mogą mieć definiowane prawa dostępu do swoich składowych, np.:

```
struct X {
  private:
    int x;
  public:
    void init();
    int fun(int y);
};
```

Struktury w C++ różnią się od klas tym, że ich domyślne składowe są publiczne, natomiast w klasach – prywatne.

Więcej różnic nie ma..

C++ - klasy

Zagnieżdżanie struktur

```
struct Bloczek {
        struct Scianka {
                int a;
        };
        int a;
        int b;
};
```

`Scianka` należy do przestrzeni nazw typu strukturalnego `Bloczek`. Obiekt typu `Bloczek` zawiera pola `a` i `b`. Dostęp do typu strukturalnego `Scianka` jest tylko za pośrednictwem `Bloczek`:

```
Bloczek::Scianka s;
```