Fundusze Europejskie
Wiedza Edukacja Rozwój

Unia Europejska
Europejski Fundusz Społeczny

Programowanie zespołowe 2.0

Programowanie obiektowe – wprowadzenie

Scenariusz zajęć

Czas realizacji: 180 minut
Cele ogólne: Tworzenie nowych typów oraz ich użycie w praktyce.
Cel szczegółowy: Przygotowanie narzędzi do zaprojektowania typów obiektów wykorzystywanych aplikacji powstającej w ramach projektu.
Konieczne wiadomości wstępne: Podstawy programowania (strukturalnego) w języku C++
Metoda prowadzenia zajęć: Dyskusja z elementami wykładu.

Podstawowe pojęcia

1. obiekt – abstrakcja danych lub model istniejącego przedmiotu lub zjawiska;
2. klasa – typ obiektu; grupuje cechy oraz funkcje obiektu;
3. pole – pojedyncza cecha obiektu; może również być obiektem;
4. funkcja składowa/metoda – działanie wykonywane przez obiekt;
5. hermetyzacja – ograniczenie dostępu do wrażliwych cech lub funkcji obiektu;
6. konstruktor – funkcja składowa odpowiadająca za prawidłową inicjalizację obiektu;

Obiekty

Istotą programowania obiektowego jest powiązanie danych oraz funkcji, które można na nich wykonać. Polega to na użyciu obiektów będących abstrakcją danych oraz modelujących rzeczywiste przedmioty i zjawiska. Każdy obiekt posiada zestaw określających go cech oraz zestaw związanych z nim działań.

Przykład 1. Przykładem obiektu może być sześcienna kostka do gry. Posiada ona m.in. następujące cechy: *kolor*, *rozmiar* czy *wskazywana wartość*, natomiast dostępnymi działaniami mogą być: *pobranie wskazywanej wartości*, *obrót w dowolną stronę*, *losowanie wartości*, itp.

Przykład 2. Innym przykładem może być obiekt reprezentujący abstrakcyjne pojęcie czasu. W tym przypadku cechami będą np.: *godzina*, *minuta*, *sekunda* oraz znacznik *zimowy* dla czasu zimowego. Dostępnymi działaniami mogą być: *ustawienie wartości czasu*, *zmiana czasu na zimowy/letni*, *pobranie wartości godziny/minuty/sekundy*, *przesunięcie czasu o ustaloną liczbę godzin/minut/sekund*, *zmiana strefy czasowej*, itp.

Ćwiczenie 1. Wypisz cechy oraz możliwe działania dla obiektu reprezentującego datę w kalendarzu.

Klasy – typy tworzone przez programistę

Klasa jest typem definiowanym przez programistę. Reprezentuje ona cechy obiektu oraz zawiera opis operacji możliwych do wykonania na nim. Każdy obiekt używany w programie jest instancją pewnej klasy. W deklaracji klasy wyróżniamy pola opisujące cechy składowe obiektu oraz funkcje składowe (zwane również metodami) określające możliwe zachowania obiektu.

Przykład 3. Przykład deklaracji klasy będącej typem dla obiektu z Przykładu 2.

```
class Czas {
  public:

    /* Pola określające cechy obiektu */

    int godzina;
    int minuta;
    int sekunda;
    bool zimowy;

    /* Funkcje składowe określające zachowanie obiektu */

    void ustawCzas(int g=0, int m=0, int s=0);
    void przesunCzas(int g=0, int m=0, int s=0);
    int getGodzina();
    int getMinuta();
    int getSekunda();
    bool czyZimowy();
    void aktywujCzasZimowy();
    void aktywujCzasLetni();
};
```

Uwaga. Pamięć dla pól rezerwowana jest oddzielnie dla każdego obiektu. Funkcje składowe trzymane są w pamięci tylko raz.

Uwaga. Reguły przeciążania funkcji oraz możliwość określenia domyślnych wartości argumentów obowiązują również w przypadku funkcji składowych klas.

Ćwiczenie 2. Zaproponuj szkielet klasy dla obiektu zaprojektowanego w ćwiczeniu 1.

Implementacja funkcji składowych klasy

Zanim będzie możliwe użycie obiektów zadeklarowanego przez nas typu musimy dostarczyć implementację wszystkich używanych funkcji składowych. Definicje krótkich funkcji składowych mogą być zawarte wewnątrz deklaracji klasy. Dla dłuższych funkcji składowych wygodniej jest umieścić implementację na zewnątrz klasy. W takim przypadku wymagane jest użycie specyfikatora zasięgu (w naszym przypadku `Czas::`) określającego, że definiowana funkcja jest składnikiem zadeklarowanej wcześniej klasy.

Przykład 4. Definicja funkcji składowej wewnątrz deklaracji klasy.

```
class Czas {
  public:
    int godzina;
    int minuta;
    int sekunda;
    bool zimowy;

    bool czyZimowy() {
        return zimowy;
    }
};
```

Przykład 5. Implementacja funkcji składowej na zewnątrz klasy.

```
class Czas {
  public:
    int godzina;
    int minuta;
    int sekunda;
    bool zimowy;

    void ustawCzas(int g=0, int m=0, int s=0);
};

void Czas::ustawCzas(int g, int m, int s) {
    godzina = g % 24;
    minuta  = m % 60;
    sekunda = s % 60;
}
```

Uwaga. Wartości domyślne dla argumentów funkcji składowych podawane są wyłącznie wewnątrz deklaracji klasy.

Tworzenie oraz użycie obiektów

Obiekty tworzonych przez nas typów mogą być deklarowane podobnie jak zmienne typów wbudowanych.

```
Czas c1;              // Zmienna typu Czas
Czas *c2 = new Czas;  // Wskaźnik na obiekt typu Czas
Czas &c3 = c1;        // Referencja do obiektu typu Czas
```

Dostęp do pól oraz funkcji składowych obiektu uzyskujemy za pomocą operatora „kropka” `.`. W przypadku obiektów dostępnych przez wskaźnik w celu skrócenia zapisu możemy również użyć operatora „strzałka” `->`.

```
c1.ustawCzas(10, 5, 12);
cout << c1.godzina << ":" << c1.minuta << ":" << c1.sekunda << endl;

if(c3.czyZimowy())
    cout << "Aktualnie wskazywany jest czas zimowy" << endl;

if(c2->czyZimowy())
    c2->aktywujLetni();
```

Ćwiczenie 3. Przygotuj implementacje dla wszystkich funkcji składowych klasy zaprojektowanej w ćwiczeniu 2. Pamiętaj o poprawnej obsłudze lat przestępnych.

Kontrola dostępu do składowych klasy (hermetyzacja)

Dane reprezentowane przez obiekt zaprojektowanej przez nas klasy powinny być spójne. Przykładowo, dla pola `minuta` nie powinno być możliwe nadanie wartości 70. W tym celu możemy ograniczyć dostęp do niektórych składowych, dzięki czemu zmiana ich wartości będzie możliwa wyłącznie za pomocą funkcji składowych klasy, które będą zaimplementowane w sposób zapewniający wewnętrzną spójność reprezentowanych danych.

W języku C++ możliwe są trzy tryby dostępu do składowych klasy:

1. `public` – składniki dostępne bez ograniczeń,
2. `private` – składniki dostępne wyłącznie dla obiektów tej samej klasy,
3. `protected` – składniki dostępne wyłącznie dla obiektów tej samej klasy oraz klas z niej dziedziczących.

Przykład 6. Zmiana trybu dostępu do składowych dla klasy z przykładu 3.

```
class Czas {
  public:
    void ustawCzas(int g=0, int m=0, int s=0);
    void przesunCzas(int g=0, int m=0, int s=0);
    int getGodzina();
    int getMinuta();
    int getSekunda();
    bool czyZimowy();
    void aktywujCzasZimowy();
    void aktywujCzasLetni();
  private:
    int godzina;
    int minuta;
    int sekunda;
    bool zimowy;
};
```

Powyższy przykład obrazuje wydzielenie publicznego interfejsu umożliwiającego wykonanie dozwolonych działań na obiekcie oraz prywatnej części chronionej przed nieuprawnionym dostępem z zewnątrz.

Specyfikatory `public`, `private` oraz `protected` mogą występować w dowolnym porządku (również wielokrotnie). Każdy specyfikator określa tryb dostępu do następujących po nim składowych aż do momentu pojawienia się kolejnego specyfikatora. W przypadku braku specyfikatora domyślnym trybem dostępu jest `private`.

Ćwiczenie 4. Zmodyfikuj implementację klasy z ćwiczenia 3 ukrywając wrażliwe dane oraz udostępniając publiczny interfejs.

Konstruktor

Konstruktor jest specjalną funkcją składową, która jest wywoływana za każdym razem kiedy tworzony jest obiekt klasy. Jego zadaniem jest poprawna inicjalizacja składowych obiektu. Nazwa funkcji konstruktora jest zgodna z nazwą klasy, nie zwraca on żadnej wartości, może natomiast być przeciążany i otrzymywać domyślne wartości argumentów.

Przykład 7. Fragment kodu klasy z przykładu 3 z dodanym konstruktorem.

```
clas Czas {
  public:
    Czas(int g=0, int m=0, int s=0) {
        ustawCzas(g, m, s);
    }

      ⋮

  private:
    int godzina;
    int minuta;
    int sekunda;
    bool zimowy;
};
```

Konstruktor domyślny

Konstruktor, który może być wywołany bez podawania argumentów (nie posiada argumentów lub wszystkie argumenty mają wartości domyślne) nazywany jest konstruktorem domyślnym. Jeśli klasa nie posiada **żadnego** konstruktora kompilator generuje publiczny konstruktor domyślny inicjujący wszystkie pola klasy na ich wartości domyślne.

Konstruktor kopiujący

Konstruktor umożliwiający utworzenie obiektu na podstawie już istniejącego obiektu tej samej klasy nazywamy konstruktorem kopiującym. Może on być wywołany jawnie podczas tworzenia obiektu lub niejawnie podczas przekazywania obiektu jako argument do

funkcji (przez wartość) lub podczas zwracania obiektu jako wynik działania funkcji (przez wartość). Jeśli klasa nie posiada konstruktora kopiującego kompilator generuje publiczny konstruktor kopiujący wartości poszczególnych pól.

Przykład 8. Fragment kodu klasy z przykładu 3 z dodanym konstruktorem kopiującym.

```
clas Czas {
  public:
    Czas(const Czas &c) {
        ustawCzas(c.g, c.m, c.s);
    }

      ⋮

  private:
    int godzina;
    int minuta;
    int sekunda;
    bool zimowy;
};
```

Uwaga. Jeśli składnikami klasy są wskaźniki, strumienie, itp. w większości przypadków konieczna jest własna implementacja konstruktora kopiującego.

Przykład 9. Tworzenie obiektów z użyciem konstruktorów.

```
Czas c1(6, 25, 34);
Czas *c2 = new Czas(15, 45);
Czas c3;                        // Konstruktor domyślny
Czas c4(c1);                    // Konstruktor kopiujący
```

Destruktor

Destruktor jest specjalną funkcją składową wywoływaną automatycznie kiedy obiekt jest likwidowany. Nazwa funkcji destruktora jest zgodna z nazwą klasy poprzedzoną znakiem tyldy (w naszym przypadku `~Czas()`). Destruktor nie pobiera argumentów, nie zwraca wartości oraz nie może być przeciążany. Jeśli klasa nie posiada destruktora kompilator wygeneruje destruktor o domyślnym działaniu.

Uwaga. Jeśli składnikami klasy są wskaźniki, strumienie, itp. w większości przypadków konieczna jest własna implementacja destruktora.

Ćwiczenie 5. Do klasy przygotowanej w ćwiczeniu 4 dodaj konstruktor oraz konstruktor kopiujący.

Obiekty jako elementy składowe innych obiektów

Jeśli polami tworzonej przez nas klasy są obiekty innych klas, konieczne będzie wykonanie konstruktorów składników podczas inicjalizacji obiektu je zawierającego. Wywołania konstruktorów składników umieszczamy na tzw. liście inicjalizacyjnej konstruktora klasy je zawierającej. W przypadku gdy klasa składnika posiada konstruktor domyślny możliwe jest jego pominięcie na liście inicjalizacyjnej.

Przykład 10. Przykład klasy zawierającej konstruktory z listą inicjalizacyjną składników.

```
class Zegar {
  public:
    Zegar(Czas cz, Czas al) : czas(cz), alarm(al) {
        alarmWlaczony = true;
    }

    Zegar(Czas cz) : czas(cz) {
        alarmWlaczony = false;
    }
```

⋮

```
  private:
    Czas czas;
    Czas alarm;
    bool alarmWlaczony;
};
```

Ćwiczenie 6. Zaprojektuj klasę `Terminarz` umożliwiającą przechowywanie informacji o zaplanowanych zdarzeniach. Data każdego zdarzenia powinna być reprezentowana za pomocą obiektu klasy zaprojektowanej w ramach poprzednich ćwiczeń, opis zdarzenia może być obiektem typu `string`.

Odnośniki

[1] B. Stroustrup *Język C++*

[2] J. Grębosz *Symfonia C++ standard*

[3] J. Grębosz *Pasja C++*

[4] B. Eckel *Thinking in C++*