Język C i C++. Polimorfizm

Motto tego artykułu: *Jak to zrobić, żeby się nie narobić i zarobić?*

Programowanie proceduralne nadaje się do rozwiązywania problemów polegających na przetwarzaniu danych według określonego z góry scenariusza. Sprawdza się zwłaszcza wtedy, gdy realizujemy niezbyt złożone algorytmy.

Niestety, w chwili, gdy aplikacja zaczyna przybierać na wadze lub zaczynamy przenosić do proceduralnego języka programowania coraz bardziej złożone zagadnienia, coraz trudniejsze może być opisanie problemu przy pomocy z góry zdefiniowanych funkcji i struktur danych. Warto wtedy poszukać lepszych rozwiązań.

W takich sytuacjach wybawieniem może być programowanie zorientowane obiektowo (OOP, *Object Oriented Programming*). Nie oznacza to, że OOP jest rozwiązaniem wszystkich problemów i że zawsze jest lepszym podejściem niż programowanie strukturalne lub proceduralne. Problemem, z jakim należy się zmierzyć, jest opanowanie umiejętności projektowania obiektowego. Zanim w ogóle włączymy komputer, warto najpierw w pamięci, a później na kartce albo lepiej na ścieralnej tablicy zaplanować przedstawienie problemu w postaci diagramu klas. Od jakości i staranności wykonania tego kroku będzie zależeć w przyszłości możliwość swobodnej rozbudowy i konserwacji projektu programistycznego. Niewielkie nawet błędy na tym etapie mogą z kolei spowodować kosztowne konsekwencje w przyszłości.

Ten etap prac (projektowanie przyszłej aplikacji, jej model obiektowy) realizuje w dużych projektach nie programista, lecz analityk/architekt systemowy. Zadaniem zespołu programistów jest jedynie implementacja wizji architekta. Analogicznie jak w branży budowlanej, kto inny przygotowuje projekt, a kto innych kopie fundamenty i stawia ściany. Jeszcze inna, specjalizowana ekipa zajmuje się wykończeniówką.

Poniższe przykłady są pierwszym podejściem do rozwiązania zadania, jakim jest zaprojektowanie i wykonanie klas systemu odpowiedzialnego za obsługę dowolnej liczby czujników (na początek załóżmy, że będzie ich nie więcej niż 10). Każdy z czujników mierzy pewną wielkość fizyczną i przechowuje w postaci wartości typu double. Każdy typ czujnika w specyficzny sposób wyświetla w konsoli zmierzone wartości.

Nie ograniczamy z góry liczby typów czujników. Zakładamy, że będzie można w miarę potrzeb dodawać do aplikacji kolejne, nowe typy. Nie wiadomo, po ile czujników każdego rodzaju zostały podłączone do systemu. Każdy czujnik w dowolnej chwili może być podłączony, odłączony lub zastąpiony egzemplarzem innego typu.

Zagadnienia do opanowania

- Klasy abstrakcyjne
- Dziedziczenie i polimorfizm
- Przesłanianie metod
- Konstruktory i destruktory
- Przeciążanie konstruktorów i metod
- Wskaźnik **this**
- Operatory new/delete
- Sekcje public, protected, private

  Wszystkie te pojęcia są niezbędne do realizacji kolejnych zadań w tym ćwiczeniu.

Założenia i projekt klasy bazowej

Zmierzona wartość będzie wewnętrznie przechowywana w polu double wartość w taki sposób, aby nie było możliwe uzyskanie dostępu do niego z zewnętrz (tzn. spoza klasy). Nie ma takiej potrzeby – czujnik we własnym zakresie wykonuje metodę void pomiar() oraz void wyswietl(). W przyszłości będzie możliwe rozszerzenie możliwości czujnika o obsługę innych typów danych (np. akcelerometr 3-osiowy XYZ będzie przechowywał dane jako wektor 3-elementowy a nie skalar).

Chcemy obsługiwać początkowo 2 rodzaje czujników: ciśnienia, wyskalowany w hPa, oraz temperatury, wyskalowany w stopniach Celsjusza.

Klasa bazowa Czujnik może wyglądać następująco:

```
class Czujnik {
  public:
     void wyswietl();
     void pomiar();
     Czujnik(string nazwa);
  protected:
     string nazwa;
     double wartosc;
};

class CzujnikX : public Czujnik
{
  public:
     void wyswietl()
     {
          cout << "Zmierzona wartosc: " << wartosc << endl;
     }
};

void main()
{
```

```
    Czujnik *czujniki[10] = { NULL };
    czujniki[0] = new CzujnikTemperatury("silnik");
    czujniki[5] = new CzujnikTemperatury("tarcze");
    czujniki[7] = new CzujnikCisnienia("olej");

    for(int i=0; i<10; i++)
        if(czujniki[i] != NULL)
        {
            czujniki[i]->pomiar();
            czujniki[i]->wyswietl();
        }
    //…
    for(int i=0; i<10; i++)
        if(czujniki[i] != NULL)
            delete czujniki[i];
}
```

Zadania do realizacji

Pamiętaj o dodaniu odpowiednich plików nagłówkowych, aby możliwe było użycie cout, time(), srand(), rand(), string.

Przykładowy kod konstruktora:

```
Czujnik(string nazwa)
{
    this->nazwa = nazwa;
}
Początkowo możesz utworzyć obiekty wyłącznie klasy Czujnik():
czujniki[0] = new Czujnik ("silnik");
czujniki[5] = new Czujnik ("tarcze");
czujniki[7] = new Czujnik ("olej");
```

1. Uzupełnij kod klasy bazowej sposób, aby działały metody pomiar() i wyświetl(). Każdy czujnik powinien w osobnej linii wyświetlać swoją nazwę i zmierzoną wartość.
2. Dodaj klasy pochodne CzujnikTemperatury oraz CzujnikCisnienia w taki sposób, aby każdy z obiektów tych klas wyświetlał swoją nazwę, wartość, jednostkę (hPa, st.C).
3. Zmodyfikuj kod klasy bazowej w taki sposób, aby stała się klasą abstrakcyjną (tzn. nie będzie możliwe tworzenie obiektów tej klasy). Powinno być natomiast możliwe tworzenie obiektów klas pochodnych.
4. Niech wszystkie czujniki temperatury ‘mierzą’ wartości losując liczbę z zakresu -40..180. Podobnie, wszystkie czujniki ciśnienia niech losują wartość z przedziału 800..5000.
5. Czy program działa zgodnie z oczekiwaniami? Dodaj do systemu 3 czujniki ciśnienia i 5 czujników temperatury. Czy wylosowały się wartości z podanych zakresów? Wskazówka: polimorfizm

6. Do każdej z klasy dodaj destruktor, który wyświetli komunikat „Destruktor klasy XXX”. Niech dodatkowo każdy konstruktor wyświetla komunikat „Konstruktor klasy XXX”. Kiedy wykonywane są konstruktory, a kiedy destruktory? Jak zmieni się sposób działania aplikacji, gdy destruktor zdefiniujesz jako wirtualny?