

13

Hermetyzacja i ukrywanie danych

13.1. Hermetyzacja danych

Hermetyzacja danych (ang. *data encapsulation*) to jedno z najważniejszych założeń paradygmatu obiektowego.

UWAGA

Mechanizm hermetyzacji danych jest również nazywany **enkapsulacją** lub **kapsułkowaniem**.

W ogólności enkapsulacja polega na powiązaniu ze sobą danych z operacjami wykonywanymi na tych danych. Wspomniane powiązanie jest realizowane w ramach określonego kontenera (kapsuły).

Język C++ pozwala na obsługę mechanizmu enkapsulacji dzięki możliwości samodzielnego definiowania klas, które odgrywają rolę „inteligentnego” kontenera (kapsuły). W ramach klasy programista określa (definiuje) zmienne członkowskie, reprezentujące dane, oraz funkcje członkowskie (metody), które stanowią implementację operacji wykonywanych na danych.

Niezwykle istotne jest również to, że programista w ramach realizowanej koncepcji klasy ma możliwość określenia „wiązań” pomiędzy danymi a funkcjami oraz może ustalić indywidualnie poziom dostępu do każdego z elementów członkowskich klasy za pomocą specyfikatorów dostępu, np. `public`, `private`.

Przykład 13.1

```

#include <iostream>
using namespace std;
// Definicja klasy Prostokat:
class Prostokat {
public:
    // Definicje zmiennych członkowskich:
    double bok1 {0};
    double bok2 {0};
    // Prototypy konstruktorów:
    Prostokat();
    Prostokat(double, double);
    // Prototyp metody dostępowej:
    void wyswietlBoki();
};

// Definicje konstruktorów:
Prostokat::Prostokat() {}
Prostokat::Prostokat(double pBok1, double pBok2) {
    bok1 = pBok1;
    bok2 = pBok2;
}

// Definicja metody wyswietlBoki():
void Prostokat::wyswietlBoki() {
    cout << "Komunikat kontrolny: wywołanie metody wyswietlBoki()" << endl;
    cout << "Pierwszy bok = " << bok1 << endl;
    cout << "Drugi bok = " << bok2 << endl;
}

/* UWAGA
 * Powiązanie zmiennych członkowskich (danych) bok1 i bok2 z funkcją członkowską wyswietlBoki() następuje już
 * na poziomie definicji tej metody.
 */

// Definicja funkcji zewnętrznej:
void wyswietlBoki(double b1, double b2) {
    cout << "Komunikat kontrolny: wywołanie funkcji zewnętrznej wyswietlBoki()"
        << endl;
    cout << "Pierwszy bok = " << b1 << endl;
    cout << "Drugi bok = " << b2 << endl;
}

/* UWAGA
 * Funkcja wyswietlBoki() została zdefiniowana w przestrzeni globalnej — na zewnątrz klasy Prostokat.
 * Jest ona zatem w pełni niezależna od klasy Prostokat.
 */

```



```
int main() {
    // Deklaracja i inicjalizacja zmiennych reprezentujących boki prostokąta:
    double dlugosc = 1; // Pierwszy bok
    double szerokosc = 2; // Drugi bok
    // Wywołanie funkcji zewnętrznej wyswietlBoki():
    wyswietlBoki(dlugosc, szerokosc);
    cout << endl;

    // Utworzenie i inicjalizacja obiektu prostokat:
    prostokat prostokat(1, 2);
    // Wywołanie funkcji członkowskiej wyswietlBoki():
    prostokat.wyswietlBoki();

    return 0;
}
```

W programie zilustrowano mechanizm hermetyzacji danych. W tym celu zdefiniowano klasę `Prostokat`, zawierającą m.in. zmienne członkowskie `bok1` i `bok2` oraz funkcję członkowską `wyswietlBoki()`.

Zmienne `bok1` i `bok2` reprezentują dane — długości boków prostokąta. Funkcja członkowska `wyswietlBoki()` zaś odpowiada za wykonanie operacji na tych danych — wyświetlenie ich wartości na ekranie monitora.

W ciele metody `wyswietlBoki()` powiązano dane (`bok1` i `bok2`) ze wspomnianymi operacjami na tych danych. Należy podkreślić, że „wiązanie” to zostało zrealizowane już na poziomie definicji klasy `Prostokat`. Mamy więc tutaj do czynienia z mechanizmem hermetyzacji danych `bok1` i `bok2` powiązanych z metodą `wyswietlBoki()` w ramach klasy `Prostokat`.

Ćwiczenie 13.1

Szczegółowo przeanalizuj kod programu zawartego w przykładzie 13.1. Na podstawie przeprowadzonej analizy odpowiedz na pytania:

- Czy funkcja zewnętrzna `wyswietlBoki()` zdefiniowana w przestrzeni globalnej jest powiązana funkcjonalnie na poziomie jej definicji z danymi reprezentowanymi w programie przez zmienne `dlugosc` i `szerokosc`?
- Czy wywołanie funkcji zewnętrznej `wyswietlBoki()` z argumentami — danymi `dlugosc` i `szerokosc` — wiąże te dane ze wspomnianą funkcją?

13.2. Ukrywanie danych

Mechanizm hermetyzacji danych może prowadzić do **ukrywania danych** (ang. *data hiding*) przed „światem zewnętrznym” — rozumianym jako otoczenie klasy, w której te dane zostały określone (zdefiniowane). W tym celu wykorzystuje się specyfikator dostępu (ang. *access specifier*) `private` w odniesieniu do zmiennych członkowskich klasy reprezentujących wspomniane dane.

UWAGA

Dane można ukrywać również z zastosowaniem specyfikatora dostępu o nazwie `protected`. Specyfikator `protected` jest wykorzystywany w implementacji mechanizmu dziedziczenia, omówionego w rozdziale 14. podręcznika.

Użycie specyfikatora dostępu `private` względem zmiennych członkowskich klasy zapewnia, że dane, które są przechowywane w tych zmiennych, są dostępne wyłącznie w obrębie definicji tej klasy. Patrząc na to z drugiej strony — dane pamiętane w prywatnych zmiennych członkowskich klasy są ukryte przed światem zewnętrznym.

W praktyce mechanizm enkapsulacji pozwala na **ukrywanie danych wrażliwych** (ang. *sensitive data hiding*) przed nieautoryzowanym lub nieprawidłowym dostępem, czyli zapewnia zwiększenie bezpieczeństwa tych danych.

Dostęp do prywatnych zmiennych członkowskich na poziomie otoczenia (na zewnątrz) klasy, w której zostały zdefiniowane, można uzyskać za pomocą metod publicznych nazywanych **akcesorami** (ang. *accessors*) lub **metodami dostępowymi** (ang. *access methods*). Akcesory, jako publiczne metody dostępowe do prywatnych zmiennych członkowskich klasy, można podzielić na dwie zasadnicze grupy: **setter** i **getter**.

Setter (ang. *setter*) to publiczna metoda dostępową, która umożliwia nadanie oraz zmianę wartości prywatnej zmiennej członkowskiej klasy.

Getter (ang. *getter*) to metoda publiczna, która umożliwia odczyt — pobranie wartości zmiennej członkowskiej klasy.

Status publiczny (`public`) setterów i getterów oznacza, że można z nich korzystać w dowolnym otoczeniu klasy, w której zostały zdefiniowane — oczywiście z zachowaniem ograniczeń związanych z zakresem widoczności klasy.

Podsumowując, mechanizm ukrywania danych polega na definiowaniu klas, które zawierają prywatne zmienne członkowskie reprezentujące dane. „Prywatność” tych zmiennych — danych — oznacza, że nie są one dostępne na zewnątrz klasy w sposób bezpośredni. Dostęp do prywatnych zmiennych członkowskich w otoczeniu klasy, w której zostały zdefiniowane, zapewniają publiczne metody dostępowe tej klasy — **setter** i **getter**.

Przykład 13.2

```
#include <iostream>
using namespace std;
```

```
// Definicja klasy Prostokat:
```

```
class Prostokat {
private:
```

```
// Definicje prywatnych zmiennych członkowskich:
```



```

double bok1 {0};
double bok2 {0};

public:
    // Prototypy konstruktorów:
    Prostokat();
    Prostokat(double, double);

    // Prototypy publicznych metod dostępowych — akcesorów:
    void setBok1(double); // setter zmiennej członkowskiej bok1
    double getBok1(); // getter zmiennej członkowskiej bok1
    void setBok2(double); // setter zmiennej członkowskiej bok2
    double getBok2(); // getter zmiennej członkowskiej bok2

};

// Definicje konstruktorów:
Prostokat::Prostokat() {}
Prostokat::Prostokat(double pBok1, double pBok2) {
    bok1 = pBok1;
    bok2 = pBok2;
}

// Definicje akcesorów — setterów i getterów:
void Prostokat::setBok1(double pBok1) {
    bok1 = pBok1;
}

double Prostokat::getBok1() {
    return bok1;
}

void Prostokat::setBok2(double pBok2) {
    bok2 = pBok2;
}

double Prostokat::getBok2() {
    return bok2;
}

int main() {
    // Deklaracje zmiennych reprezentujących boki prostokąta:
    double dlugosc = 1; // pierwszy bok
    double szerokosc = 2; // drugi bok

    // Utworzenie obiektu prostokat:
    Prostokat prostokat;

    /* UWAGA
    * Wywołanie zdefiniowanego konstruktora domyślnego nie skutkuje inicjalizacją zmiennych
    * członkowskich bok1 i bok2.
    */
}

```



```
// Nadanie wartości początkowych zmiennym członkowskim bok1 i bok2:
prostokat.setBok1(dlugosc); // wywołanie settera setBok1() z argumentem dlugosc
prostokat.setBok2(szerokosc); // wywołanie settera setBok2() z argumentem szerokosc
// Prezentacja danych prostokąta na ekranie monitora:
cout << "Długości boków prostokąta: " << endl;
cout << "Pierwszy bok = " << prostokat.getBok1() << endl; // wywołanie gettera
// getBok1()
cout << "Drugi bok = " << prostokat.getBok2() << endl; // wywołanie gettera getBok2()

return 0;
}
```

W programie zdefiniowano klasę `Prostokat` zawierającą definicje prywatnych zmiennych członkowskich `bok1` i `bok2`. Oprócz tego w klasie `Prostokat` zdefiniowano publiczne metody dostępne — akcesory do wymienionych zmiennych.

Akcesory `setBok1()` i `setBok2()` pełnią funkcję setterów dla zmiennych członkowskich `bok1` i `bok2`. Akcesory `getBok1()` i `getBok2()` to gettery.

Zmienne członkowskie `bok1` i `bok2`, reprezentujące dane — długości boków prostokąta — są ukryte przed „światem zewnętrznym” w ramach definicji klasy `Prostokat`. Dostęp do wspomnianych danych z poziomu otoczenia klasy `Prostokat` można uzyskać wyłącznie za pomocą wymienionych wcześniej publicznych metod dostępowych — setterów i getterów, które zostały zdefiniowane dla każdej zmiennej członkowskiej w tej klasie.

Konstruktor parametryczny pełni funkcję settera specjalnego, ale w odniesieniu do obu zmiennych członkowskich jednocześnie.

Ćwiczenie 13.2

Na podstawie przykładu 13.2 napisz program o takiej samej funkcjonalności, ale bez użycia żadnych obiektów. W tym celu zapisz dane prostokąta w strukturze traktowanej jako C-struktura. Zdefiniuj niezależne od siebie funkcje pozwalające na dostęp do danych prostokąta w trybie „zapisz/modyfikuj” i w trybie „odczytaj”.

13.3. Pytania i zadania kontrolne

13.3.1. Pytania

1. Na czym polega mechanizm hermetyzacji danych, a na czym mechanizm ukrywania danych?
2. Co to jest akcesor? Jakie są rodzaje akcesorów?

13.3.2. Zadania

1. Napisz program pozwalający obliczyć objętość, pole powierzchni całkowitej oraz długość wszystkich krawędzi prostopadłościanu. Dane wejściowe — wartości parametrów prostopadłościanu — mają być wprowadzane z klawiatury, a wyniki wyświetlane na ekranie monitora. Wykorzystaj zdefiniowaną samodzielnie klasę *Prostopadloscian*, która będzie zawierać zmienne członkowskie odpowiadające parametrom prostopadłościanu. Zastosuj do tych danych mechanizm hermetyzacji — ukrywania ich przed „światem zewnętrznym”.
2. Napisz program pozwalający przechować w zmiennych członkowskich klasy *Uczen* następujące dane ucznia: imię, nazwisko, datę urodzenia, klasę, grupę. Do wymienionych danych zastosuj mechanizm hermetyzacji — ukrywania. Dane ucznia powinny zostać wprowadzone z klawiatury, a następnie wyświetlone kontrolnie na ekranie monitora.