

Mechanizm dziedziczenia

Dziedziczenie (ang. *inheritance*) to jedna z najważniejszych cech paradygmatu obiektowego. Polega na tym, że dana klasa może zostać zdefiniowana na podstawie innej, już istniejącej klasy i w ten sposób „odziedziczyć” po niej określone elementy członkowskie (ang. *members*).

Dziedziczone mogą być zarówno zmienne członkowskie (ang. *member variables*), jak i funkcje członkowskie (ang. *member functions*), inaczej metody (ang. *methods*). Przy tym, ażeby wspomniane zmienne i funkcje członkowskie mogły być dziedziczone do innych klas, powinny mieć status albo *public*, albo *protected*.

Specyfikator dostępu (ang. *access specifier*) *public* wyszczególniony w definicji klasy przed deklaracjami jej elementów członkowskich, tj.:

public:

deklaracje_elementów_członkowskich;

oznacza, że elementy członkowskie klasy wymienione w deklaracjach_elementów_członkowskich są dostępne w sposób bezpośredni zarówno wewnątrz tej klasy, jak i na zewnątrz, tj. w jej otoczeniu. Była o tym mowa w podrozdziale 11.2.

Użycie w odniesieniu do elementów członkowskich klasy specyfikatora dostępu *protected*, tj.:

protected:

deklaracje_elementów_członkowskich;

oznacza zaś, że składniki klasy wymienione w deklaracjach_elementów_członkowskich nie są dostępne w sposób bezpośredni w otoczeniu — na zewnątrz tej klasy (czyli tak samo jak prywatne elementy członkowskie), ale zarazem mogą być dziedziczone do innych klas.

Patrząc z drugiej strony, dziedziczeniu nie podlegają następujące składniki klasy:

- konstruktory i destruktor,
- prywatne zmienne członkowskie, czyli takie, które zostały określone za pomocą specyfikatora dostępu `private`.

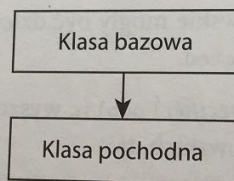
To samo dotyczy funkcji zaprzyjaźnionych z daną klasą (ang. *friend functions*).

UWAGA

Tematyka przyjaciół klas (ang. *class friends*) została omówiona w dalszej części podręcznika — w rozdziale 17. Obejmuje ona m.in. funkcje zaprzyjaźnione.

14.1. Definicja relacji dziedziczenia

Klasa istniejąca, na podstawie której tworzona jest nowa klasa, jest nazywana **klasą bazową** (ang. *base class*). Inne spotykane nazwy klasy bazowej to np. **klasa nadrzędna** (ang. *parent class*) i **superklasa** (ang. *superclass*). Nowo tworzona klasa, która dziedziczy określone pola i metody po klasie bazowej, jest zaś nazywana **klasą pochodną** (ang. *derived class*), a także **klasą dzieckiem** (ang. *child class*) lub **podklasą** (ang. *subclass*). Zilustrowano to na rysunku 14.1.



Rysunek 14.1. Interpretacja klasy podstawowej i klasy pochodnej

Relacja dziedziczenia jest deklarowana na poziomie definicji klasy pochodnej. Ogólna postać definicji relacji dziedziczenia jest następująca:

```
class nazwa_klasy_pochodnej : specyfikator_dostępu nazwa_klasy_bazowej {
    deklaracje_i_definicje_elementów_członkowskich;
};
```

gdzie:

- `nazwa_klasy_pochodnej` oznacza identyfikator klasy pochodnej,
- `specyfikator_dostępu` to specyfikator dostępu (ang. *access specifier*): `public`, `protected` lub `private`, określający tryb (rodzaj) dziedziczenia,
- `nazwa_klasy_bazowej` jest identyfikatorem klasy bazowej,
- `deklaracje_i_definicje_elementów_członkowskich` to deklaracje i definicje zmiennych członkowskich, metod i innych elementów składowych klasy pochodnej.

Jak wspomniano powyżej, **typ dziedziczenia** (ang. *type of inheritance*) jest określony za pomocą specyfikatora dostępu: `public`, `protected` lub `private`, wyszczególnionego w nagłówku definicji klasy pochodnej pomiędzy dwukropkiem : a identyfikatorem klasy bazowej `nazwa_klasy_bazowej`. Specyfikator ten definiuje **tryb dostępu** (ang. *access mode*) do elementów członkowskich klasy pochodnej, które zostały odziedziczone (lub nie) po klasie bazowej.

Typ (rodzaj) dziedziczenia wynikający z trybu dostępu do elementów członkowskich klasy pochodnej odziedziczonych po klasie bazowej jest nazywany **trybem dziedziczenia** (ang. *inheritance mode*).

Biorąc pod uwagę powyższe, w języku C++ można wyróżnić następujące tryby dziedziczenia:

- publiczne — jeśli w nagłówku definicji klasy pochodnej wyszczególniono specyfikator dostępu określony za pomocą słowa kluczowego `public`,
- chronione — jeśli w nagłówku definicji klasy pochodnej podano specyfikator `protected`,
- prywatne — jeśli w nagłówku definicji klasy pochodnej podano specyfikator `private`.

Domyślnym typem dziedziczenia w języku C++ jest dziedziczenie prywatne.

UWAGA

W przypadku struktur (ang. *structs*) domyślnym rodzajem dziedziczenia w C++ jest dziedziczenie publiczne.

Dziedziczenie publiczne (ang. *public inheritance*) oznacza, że:

- publiczne elementy członkowskie klasy bazowej, czyli te, które są określone za pomocą specyfikatora dostępu `public`, są dostępne również jako publiczne w klasie pochodnej i stanowią publiczne elementy składowe obiektów klasy pochodnej,
- chronione elementy składowe klasy bazowej, określone za pomocą specyfikatora dostępu `protected`, są dostępne również jako chronione w klasie pochodnej,
- prywatne elementy klasy bazowej — zdefiniowane tam jako `private` — nie są dostępne bezpośrednio w klasie pochodnej, przy czym można uzyskać do nich dostęp w sposób pośredni, przez wykorzystanie publicznych i chronionych metod należących do klasy bazowej.

Dziedziczenie chronione (ang. *protected inheritance*) ma następujące cechy:

- publiczne elementy składowe klasy bazowej są dostępne jako chronione w klasie pochodnej,
- chronione elementy składowe klasy bazowej są dostępne jako chronione w klasie pochodnej,
- prywatne elementy klasy bazowej nie są dostępne bezpośrednio w klasie pochodnej.

Dziedziczenie prywatne (ang. *private inheritance*) zaś wiąże się z tym, że:

- publiczne elementy składowe klasy bazowej są dostępne jako prywatne w klasie pochodnej,
- chronione elementy składowe klasy bazowej są dostępne jako prywatne w klasie pochodnej,
- prywatne elementy klasy bazowej nie są dostępne bezpośrednio w klasie pochodnej.

Mechanizm dziedziczenia zilustrowano na podstawie dziedziczenia publicznego za pomocą dwóch przykładów. W przykładzie 14.1 zmienne członkowskie hermetyzowane w klasach bazowej i pochodnej charakteryzują się statusem `public`, a w przykładzie 14.2 przedstawiono zastosowanie zarówno mechanizmu dziedziczenia, jak i ukrywania danych.

UWAGA

Podział dziedziczenia uwzględniający strukturę klas, pomiędzy którymi zachodzi relacja dziedziczenia, został omówiony w następnym podrozdziale.

Przykład 14.1

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

// Definicja klasy bazowej Pracownik:
class Pracownik {
public: // wszystkie elementy członkowskie klasy są publiczne
    // Definicje zmiennych członkowskich:
    string imie {" "};
    string nazwisko {" "};
    // Definicja funkcji członkowskiej:
    string zwrocDane() {
        return imie + " " + nazwisko;
    }
};

// Definicja klasy pochodnej Nauczyciel:
class Nauczyciel: public Pracownik { // dziedziczenie publiczne
public:
    // Definicja zmiennej członkowskiej:
    string przedmiot {" "};
    // Definicja funkcji członkowskiej:
    string zwrocDane() {
        return imie + " " + nazwisko + ", przedmiot: " + przedmiot;
    }
};
```



```

int main() {
    // Utworzenie obiektu nauczyciel jako instancji klasy pochodnej Nauczyciel:
    Nauczyciel nauczyciel;
    // Nadanie wartości zmiennym członkowskim obiektu nauczyciel:
    nauczyciel.imie = "Jan"; // zmienna odziedziczona po klasie bazowej
    nauczyciel.nazwisko = "Kowalski"; // zmienna odziedziczona
    nauczyciel.przedmiot = "elektronika"; // zmienna natywna — zdefiniowana w klasie pochodnej
    /* UWAGA
    * Ze względu na to, że dziedziczenie jest publiczne, w klasie pochodnej dostępne są wszystkie publiczne zmienne
    * członkowskie odziedziczone po klasie bazowej. W klasie pochodnej zmienne te (imie i nazwisko)
    * również są publiczne.
    */
    // Wywołanie metody zwrocDane() zdefiniowanej w klasie pochodnej:
    cout << "Dane nauczyciela: " << nauczyciel.zwrocDane() << endl;

    return 0;
}

```

W programie zdefiniowano dwie klasy: `Pracownik` i `Nauczyciel`, przy czym klasa `Nauczyciel` została zdefiniowana na podstawie klasy `Pracownik` z wykorzystaniem mechanizmu dziedziczenia. Klasa `Pracownik` jest klasą bazową, a klasa `Nauczyciel` — pochodną.

Dziedziczenie określono jako publiczne: `class Nauczyciel: public Pracownik`. Tym samym publiczne elementy członkowskie klasy bazowej są dostępne również jako publiczne w klasie pochodnej.

Funkcja członkowska `zwrocDane()` należąca do klasy bazowej `Pracownik` została przesłonięta (ang. *overriding*) przez funkcję o tej samej nazwie zdefiniowaną w klasie pochodnej `Nauczyciel`.

Ćwiczenie 14.1

Zmodyfikuj program z przykładu 14.1 — zdefiniuj klasę `Wychowawca`, która będzie klasą pochodną klasy `Nauczyciel`. W klasie `Wychowawca` zdefiniuj natywną zmienną członkowską oraz metodę `zwrocDane()` o funkcjonalności analogicznej do funkcjonalności metod klasy oraz metodę `zwrocDane()` o funkcjonalności analogicznej do funkcjonalności metod o tej samej nazwie zdefiniowanych w klasach `Pracownik` i `Nauczyciel`. Zastosuj dziedziczenie publiczne. Utwórz obiekt `wychowawca` jako instancję klasy `Wychowawca`. Przypisz zadane wartości jego zmiennym członkowskim, a następnie wyświetl je kontrolnie na ekranie monitora za pomocą funkcji `zwrocDane()`.

Przykład 14.2

```

#include <iostream>
using namespace std;

// Definicja klasy bazowej Pracownik:

```

```

class Pracownik {
private:
    // Definicje prywatnych zmiennych członkowskich:
    string imie {" "};
    string nazwisko {" "};
public:
    // Definicje publicznych setterów:
    void setImie(string pImie) {
        imie = pImie;
    }
    void setNazwisko(string pNazwisko) {
        nazwisko = pNazwisko;
    }
    // Definicje publicznych getterów:
    string getImie() {
        return imie;
    }
    string getNazwisko() {
        return nazwisko;
    }
    // Definicja publicznej funkcji członkowskiej:
    string zwrocDane() {
        return imie + " " + nazwisko;
    }
};

// Definicja klasy pochodnej Nauczyciel:
class Nauczyciel: public Pracownik { // dziedziczenie publiczne
private:
    // Definicja zmiennej członkowskiej:
    string przedmiot {" "};
public:
    // Definicja settera:
    void setPrzedmiot(string pPrzedmiot) {
        przedmiot = pPrzedmiot;
    }
    // Definicja gettera:
    string getPrzedmiot() {
        return przedmiot;
    }
    // Definicja funkcji członkowskiej:
    string zwrocDane() {
        return getImie() + " " + getNazwisko() + ", przedmiot: " + przedmiot;
    }
}

```



```

/* UWAGA
 * Dostęp do prywatnych zmiennych członkowskich imie i nazwisko należących do klasy bazowej
 * uzyskano za pomocą publicznych metod dostępowych — setterów i getterów,
 * które również zostały zdefiniowane w tej klasie.
 */

};

int main() {
    // Utworzenie obiektu nauczyciel jako instancji klasy pochodnej Nauczyciel:
    Nauczyciel nauczyciel;
    // Nadanie wartości zmiennym członkowskim obiektu nauczyciel:
    nauczyciel.setImie("Jan");
    nauczyciel.setNazwisko("Kowalski");
    nauczyciel.setPrzedmiot("elektronika");
    /* UWAGA
     * Prywatne zmienne członkowskie imie i nazwisko należące do klasy bazowej Pracownik nie są dziedziczone
     * do klasy pochodnej Nauczyciel. Dostęp do tych zmiennych członkowskich z poziomu klasy Nauczyciel
     * uzyskano przez wykorzystanie publicznych metod dostępowych (setterów) zdefiniowanych w klasie Pracownik,
     * które podlegają dziedziczeniu. Dostęp do prywatnej zmiennej członkowskiej przedmiot zdefiniowanej
     * w klasie Nauczyciel uzyskano w analogiczny sposób.
     */
    // Wywołanie metody zwrocDane() zdefiniowanej w klasie pochodnej:
    cout << "Dane nauczyciela: " << nauczyciel.zwrocDane() << endl;

    return 0;
}

```

Funkcjonalność programu zawartego w przykładzie 14.2 jest analogiczna do funkcjonalności programu z przykładu 14.1, jednakże klasy Pracownik i Nauczyciel zaimplementowano w odmienny sposób. Tutaj zmienne członkowskie imie i nazwisko należące do klasy Pracownik oraz przedmiot z klasy Nauczyciel są prywatne. Dostęp do tych zmiennych jest realizowany za pomocą publicznych metod dostępowych — setterów i getterów, zdefiniowanych w obu wymienionych klasach. Mamy więc do czynienia z hermetyzacją — ukrywaniem danych.

Pomimo że zmienne członkowskie imie i nazwisko zostały zdefiniowane w klasie bazowej Pracownik jako prywatne — i tym samym nie są dziedziczone do klasy pochodnej Nauczyciel w sposób bezpośredni — można do nich uzyskać dostęp w klasie pochodnej Nauczyciel za pośrednictwem publicznych setterów i getterów zdefiniowanych w klasie Pracownik, które podlegają dziedziczeniu.

Ćwiczenie 14.2

Zmodyfikuj program z przykładu 14.2 — zdefiniuj klasę Wychowawca, która będzie klasą pochodną klasy Nauczyciel. W klasie Wychowawca zdefiniuj prywatną zmienną członkowską klasa oraz publiczne metody dostępowe do tej zmiennej, setter i getter, a także metodę

zwrocDane() o funkcjonalności analogicznej do funkcjonalności metody o tej samej nazwie w klasach Pracownik i Nauczyciel. Wykorzystaj dziedziczenie publiczne. Utwórz obiekt wychowawca jako instancję klasy Wychowawca, a następnie przypisz zadane wartości jego zmiennym członkowskim. Na końcu wyświetl kontrolnie wartości wszystkich zmiennych członkowskich obiektu wychowawca na ekranie monitora za pomocą funkcji zwrocDane() zdefiniowanej w klasie Wychowawca.

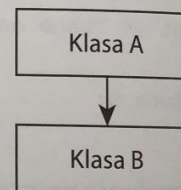
14.2. Rodzaje dziedziczenia

W poprzednim podrozdziale zostały omówione typy (rodzaje) dziedziczenia wynikające bezpośrednio z postaci definicji klasy pochodnej. Uwzględniają one tryb dostępu do elementów członkowskich klasy pochodnej wynikający z wykorzystanego specyfikatora dostępu (public, private i protected) w jej definicji.

Dziedziczenie można podzielić również w zależności od budowy (struktury wewnętrznej) tzw. **łańcucha dziedziczenia** (ang. *inheritance chain*), który jest zestawem klas powiązanych ze sobą relacją dziedziczenia. Uwzględniając to kryterium, można wyróżnić:

- dziedziczenie pojedyncze,
- dziedziczenie wielokrotne,
- dziedziczenie hierarchiczne,
- dziedziczenie wielopoziomowe,
- dziedziczenie mieszane.

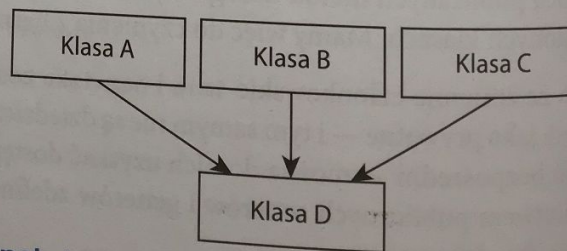
Dziedziczenie pojedyncze (ang. *single inheritance*) zachodzi wtedy, gdy klasa pochodna dziedziczy wyłącznie po jednej klasie bazowej (rysunek 14.2):



Rysunek 14.2. Interpretacja dziedziczenia pojedynczego

Łańcuch dziedziczenia na rysunku 14.2 składa się z dwóch klas. Klasa pochodna B dziedziczy po jednej klasie bazowej — klasie A. Zachodzi zatem dziedziczenie pojedyncze.

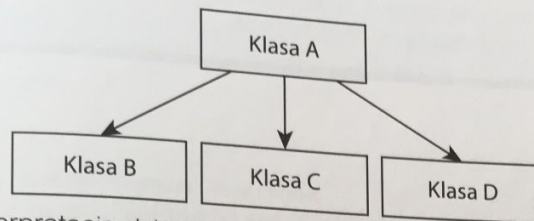
Dziedziczenie wielokrotne (ang. *multiple inheritance*) zachodzi wtedy, gdy dana klasa pochodna dziedziczy po większej liczbie klas bazowych (rysunek 14.3).



Rysunek 14.3. Interpretacja dziedziczenia wielokrotnego

Relacja dziedziczenia na rysunku 14.3 została określona pomiędzy czterema klasami. Łańcuch dziedziczenia jest zatem złożony z czterech klas. Klasa pochodna D dziedziczy po trzech klasach bazowych — A, B i C. Zachodzi dziedziczenie wielokrotne.

Dziedziczenie hierarchiczne (ang. *hierarchical inheritance*) to takie, w którym dana klasa bazowa ma wiele klas pochodnych (rysunek 14.4).

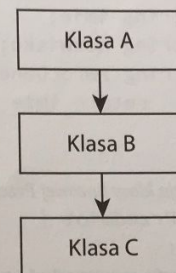


Rysunek 14.4. Interpretacja dziedziczenia hierarchicznego

Klasa bazowa A na rysunku 14.4 ma trzy klasy pochodne — B, C i D. Zachodzi dziedziczenie hierarchiczne.

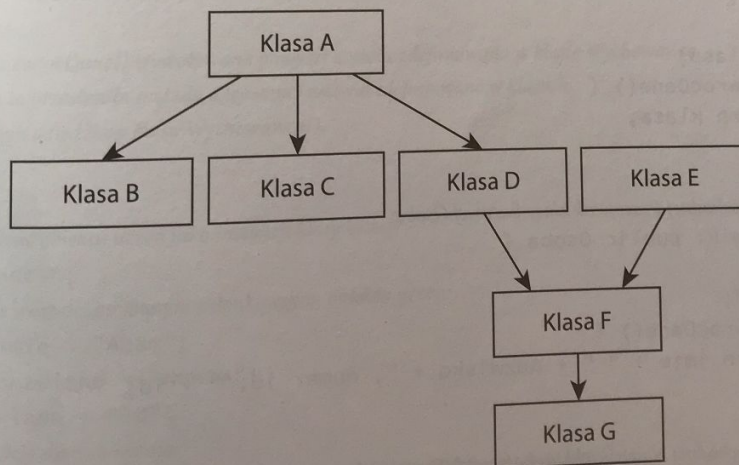
Dziedziczenie wielopoziomowe (ang. *multilevel inheritance*) zachodzi wówczas, gdy dana klasa pochodna dziedziczy po klasie bazowej, która sama jest już klasą pochodną innej klasy (rysunek 14.5).

Klasa pochodna C na rysunku 14.5 dziedziczy po klasie bazowej B, która sama jest klasą pochodną względem klasy bazowej A — zachodzi dziedziczenie wielopoziomowe.



Rysunek 14.5. Interpretacja dziedziczenia wielopoziomowego

Dziedziczenie mieszane (hybrydowe) (ang. *hybrid inheritance*) to dowolna kombinacja powyższych rodzajów dziedziczenia (rysunek 14.6).



Rysunek 14.6. Interpretacja dziedziczenia mieszanego

Relacje dziedziczenia zachodzące pomiędzy klasami, które należą do łańcucha dziedziczenia na rysunku 14.6, określają różne rodzaje dziedziczenia, np. dziedziczenie wielokrotne, hierarchiczne i wielopoziomowe. Mamy więc tutaj do czynienia z dziedziczeniem hybrydowym.

UWAGA

W praktyce struktury klas połączone ze sobą relacją dziedziczenia zazwyczaj są organizowane z uwzględnieniem różnych rodzajów dziedziczenia.

Przykład 14.3

```
#include <iostream>
using namespace std;
// Definicja klasy bazowej Osoba:
class Osoba {
public:
    string imie;
    string nazwisko;
    string zwrocDane() {
        return imie + " " + nazwisko;
    }
};
// Definicja klasy bazowej Przedmiot:
class Przedmiot {
public:
    string przedmiot;
    string zwrocDane() {
        return przedmiot;
    }
};
// Definicja klasy bazowej Klasa:
class Klasa {
public:
    string klasa;
    string zwrocDane() {
        return klasa;
    }
};
// Definicja klasy pochodnej Pracownik klasy bazowej Osoba:
class Pracownik: public Osoba {
public:
    string id;
    string zwrocDane() {
        return imie + " " + nazwisko + ", numer id: " + id;
    }
};
// Definicja klasy pochodnej Nauczyciel klas bazowych Pracownik, Przedmiot:
class Nauczyciel: public Pracownik, public Przedmiot {
public:
    string zwrocDane() {
        return imie + " " + nazwisko + ", przedmiot " + przedmiot;
    }
};
```



```

// Definicja klasy pochodnej Wychowawca klas bazowych Nauczyciel i Klasa:
class Wychowawca: public Nauczyciel, public Klasa {
public:
    string zwrocDane() {
        return imie + " " + nazwisko + ", przedmiot " + przedmiot + ", klasa "
            + klasa;
    }
};

// Definicja klasy pochodnej Ucen klas bazowych Osoba i Klasa:
class Ucen: public Osoba, public Klasa {
public:
    string zwrocDane() {
        return imie + " " + nazwisko + ", klasa " + klasa;
    }
};

int main() {
    // Utworzenie obiektu wychowawca klasy Wychowawca:
    Wychowawca wychowawca;
    // Nadanie wartości zmiennym członkowskim obiektu wychowawca:
    wychowawca.imie = "Jan"; // zmienna członkowska odziedziczona po klasie Osoba
    wychowawca.nazwisko = "Kowalski"; // zmienna odziedziczona po klasie Osoba
    wychowawca.przedmiot = "informatyka"; // zmienna odziedziczona po klasie Przedmiot
    wychowawca.klasa = "4A"; // zmienna odziedziczona po klasie Klasa
    // Wywołanie metody zwrocDane() zdefiniowanej w klasie Wychowawca:
    cout << "Dane nauczyciela wychowawcy: " << wychowawca.zwrocDane() << endl;
    /* UWAGA
    * Funkcja zwrocDane() wywoływana powyżej została zdefiniowana w klasie Wychowawca.
    * Metoda ta przesłoniła metody o tej samej nazwie zdefiniowane w klasach,
    * po których dziedziczy klasa Wychowawca().
    */

    // Utworzenie obiektu uczen jako instancji klasy Ucen:
    Ucen uczen;
    // Nadanie wartości zmiennym członkowskim obiektu uczen:
    uczen.imie = "Adam";
    uczen.nazwisko = "Nowak";
    uczen.klasa = "3B";
    // Prezentacja danych ucznia:
    cout << "Dane ucznia: " << uczen.zwrocDane() << endl;

    return 0;
}

```

W programie zademonstrowano różne typy dziedziczenia, które zachodzą pomiędzy klasami wchodzącymi w skład łańcucha dziedziczenia.

Na przykład klasa bazowa *osoba* ma dwie klasy pochodne: *Pracownik* i *Uczen*. Tym samym pomiędzy tymi klasami zachodzi relacja dziedziczenia hierarchicznego, które cechuje się tym, że dana klasa bazowa ma wiele klas pochodnych.

Klasy *Nauczyciel*, *Wychowawca* i *Uczen* dziedziczą po większej liczbie klas bazowych. Klasa *Nauczyciel* dziedziczy po klasach *Pracownik* i *Przedmiot*, klasa *Wychowawca* — po klasach *Nauczyciel* i *Klasa*, a klasa *Uczen* — po klasach *osoba* i *Klasa*. W tym przypadku mamy zatem do czynienia z dziedziczeniem wielokrotnym, którego cechą charakterystyczną jest to, że dana klasa pochodna może dziedziczyć po większej (niż jedna) liczbie klas bazowych.

Klasa *Wychowawca* jest „prawnukiem” klasy *osoba*, klasa *Nauczyciel* — „wnukiem”, a klasa *Pracownik* — „synem”. Wynika z tego, że relacje dziedziczenia, które łączą wymienione klasy, definiują łańcuch dziedziczenia określający dziedziczenie wielopoziomowe.

Uwzględniając to, że w łańcuchu dziedziczenia obejmującym wszystkie klasy zdefiniowane w programie występują opisane powyżej różne relacje dziedziczenia, możemy powiedzieć, że mamy tutaj do czynienia z dziedziczeniem mieszanym.

Ćwiczenie 14.3

Zmodyfikuj program z przykładu 14.3 — uwzględnij hermetyzację i ukrywanie danych we wszystkich klasach wchodzących w skład łańcucha dziedziczenia. Dostęp do ukrytych zmiennych członkowskich zrealizuj za pomocą zdefiniowanych samodzielnie metod dostępowych (setterów i getterów).

14.3. Dziedziczenie a konstruktory

Pomimo że klasa pochodna nie dziedziczy konstruktorów po klasie bazowej, przy tworzeniu obiektu klasy pochodnej — czyli przy wywołaniu konstruktora tej klasy — wywoływany jest automatycznie konstruktor jej klasy bazowej. Przy tym, jeśli w definicji konstruktora klasy pochodnej nie określono inaczej, zostanie wywołany konstruktor domyślny klasy bazowej. To samo dotyczy wywołania destruktora klasy pochodnej, które skutkuje wywołaniem konstruktora klasy bazowej.

Przykład 14.4

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

```
// Definicja klasy bazowej Pracownik;
class Pracownik {
public:
    string imie;
```



```

string nazwisko;
// Definicje konstruktorów:
Pracownik() {}; // konstruktor domyślny
Pracownik(string pImie, string pNazwisko) { // konstruktor z parametrami
    imie = pImie;
    nazwisko = pNazwisko;
}

string zwrocDane() {
    return imie + " " + nazwisko;
}
};
// Definicja klasy pochodnej Nauczyciel:
class Nauczyciel: public Pracownik {
public:
    string przedmiot;
    // DEFINICJE KONSTRUKTORÓW:
    Nauczyciel() {}; // konstruktor domyślny
    /* UWAGA
    * Przy wywołaniu konstruktora domyślnego zdefiniowanego powyżej zostanie wywołany konstruktor domyślny
    * klasy bazowej, ponieważ nie zdecydowano inaczej.
    */
    Nauczyciel(string pImie, string pNazwisko, string pPrzedmiot) { // konstruktor
                                                                    // z parametrami

        imie = pImie;
        nazwisko = pNazwisko;
        przedmiot = pPrzedmiot;
    }
    /* UWAGA
    * Przy wywołaniu konstruktora parametrycznego zdefiniowanego powyżej zostanie wywołany konstruktor
    * domyślny klasy bazowej, ponieważ nie zdecydowano inaczej.
    */
    // Definicja konstruktora parametrycznego z określeniem konstruktora klasy bazowej:
    Nauczyciel(string pImie, string pNazwisko) : Pracownik(pImie, pNazwisko) {}
                                                                    // konstruktor z parametrami

    /* UWAGA
    * Przy wywołaniu konstruktora parametrycznego zdefiniowanego powyżej zostanie wywołany określony —
    * zdefiniowany konstruktor klasy bazowej.
    */
    string zwrocDane() {
        return imie + " " + nazwisko + ", przedmiot: " + przedmiot;
    }
};

```

```

int main() {
    // Utworzenie obiektu nauczyciel jako instancji klasy Nauczyciel:
    Nauczyciel nauczyciel1;
    /* UWAGA
    * Przy tworzeniu obiektu nauczyciel1 z wykorzystaniem konstruktora domyślnego zostaje wywołany
    * automatycznie konstruktor domyślny (bezparametrowy) klasy bazowej Pracownik.
    */
    // Nadanie wartości polom obiektu nauczyciel1:
    nauczyciel1.imie = "Jan";
    nauczyciel1.nazwisko = "Kowalski";
    nauczyciel1.przedmiot = "język angielski";
    // Wywołanie metody zwrocDane() zdefiniowanej w klasie pochodnej Nauczyciel:
    cout << "Dane nauczyciela: " << nauczyciel1.zwrocDane() << endl;

    // Utworzenie obiektu nauczyciel2 klasy Nauczyciel przez użycie konstruktora z parametrami:
    Nauczyciel nauczyciel2("Adam", "Nowak", "język niemiecki");
    /* UWAGA
    * Przy tworzeniu obiektu nauczyciel2 z użyciem konstruktora z parametrami zostanie wywołany automatycznie
    * konstruktor domyślny (bezparametrowy) klasy bazowej Pracownik.
    */
    cout << "Dane nauczyciela: " << nauczyciel2.zwrocDane() << endl;

    // Utworzenie obiektu nauczyciel3 klasy Nauczyciel:
    Nauczyciel nauczyciel3 = Nauczyciel("Jan", "Nowak");
    /* UWAGA
    * Przy tworzeniu obiektu nauczyciel3 z użyciem konstruktora z parametrami zostanie wywołany zdefiniowany
    * konstruktor parametryczny klasy bazowej Pracownik.
    */
    nauczyciel3.przedmiot = "język francuski";
    cout << "Dane nauczyciela: " << nauczyciel3.zwrocDane() << endl;

    return 0;
}

```

Klasa Nauczyciel została zdefiniowana na podstawie klasy Pracownik z zastosowaniem relacji dziedziczenia. Klasa Pracownik jest klasą bazową, a klasa Nauczyciel — pochodną. W klasie bazowej Pracownik zdefiniowano dwa konstruktory: domyślny i parametryczny. Klasa pochodna Nauczyciel z kolei zawiera definicje trzech konstruktorów: domyślnego i dwóch parametrycznych.

Utworzenie obiektu nauczyciel1 jest związane z niejawnym wywołaniem konstruktora domyślnego klasy pochodnej, co skutkuje automatycznym wywołaniem konstruktora domyślnego klasy bazowej. Wynika to z definicji konstruktora domyślnego klasy pochodnej, w której nie określono żadnego konstruktora klasy bazowej.

Obiekt `nauczyciel2` zaś został utworzony przez niejawne wywołanie konstruktora parametrycznego klasy pochodnej, co również powoduje automatyczne wywołanie konstruktora domyślnego klasy bazowej.

W przypadku obiektu `nauczyciel3`, utworzonego przez wywołanie konstruktora parametrycznego klasy pochodnej w sposób jawny, wywołany zostanie zdefiniowany konstruktor parametryczny klasy bazowej. Wynika to z definicji tego konstruktora w klasie pochodnej, w której ustalono, jaki konstruktor klasy bazowej ma być wywołany podczas tworzenia obiektu będącego instancją klasy pochodnej.

Ćwiczenie 14.4

Zmodyfikuj program zawarty w przykładzie 14.4 — uzupełnij ciała konstruktorów domyślnych klas `Pracownik` i `Nauczyciel` o instrukcje pozwalające na inicjalizację ich zmiennych członkowskich. Zaimplementuj hermetyzację — ukrycie danych przechowywanych w zmiennych członkowskich obu klas.

14.4. Pytania i zadania kontrolne

14.4.1. Pytania

1. Na czym polega mechanizm dziedziczenia? Jakie są korzyści wynikające z zastosowania dziedziczenia?
2. Wyjaśnij znaczenie pojęć: klasa bazowa, klasa pochodna.
3. Co to jest łańcuch dziedziczenia?
4. Wymień i scharakteryzuj tryby (rodzaje) dziedziczenia uwzględniające tryb dostępu do elementów członkowskich klasy pochodnej.
5. Na czym polega dziedziczenie hierarchiczne, a na czym dziedziczenie wielokrotne?
6. Czy klasa pochodna dziedziczy konstruktory po klasie bazowej? Czy wywołanie konstruktora klasy pochodnej może powodować wywołanie konstruktora klasy bazowej?

14.4.2. Zadania

1. Napisz program pozwalający przetwarzać dane komputerów. Uwzględnij komputery stacjonarne PC oraz laptopy. Przyjmij następujące założenia:
 - cechy wspólne każdego komputera to marka i model,
 - cecha charakterystyczna komputera PC to rodzaj obudowy, np. slim,
 - cechy charakterystyczne laptopa to długość przekątnej ekranu i kolor obudowy.

Wykonaj program zgodnie z zasadami programowania obiektowego — wykorzystaj zdefiniowane samodzielnie klasy powiązane ze sobą relacjami dziedziczenia. Zastosuj dziedziczenie hierarchiczne w trybie publicznym. Niech w celach testowych będzie

możliwe wprowadzenie danych określonego laptopa z klawiatury, zapamiętanie ich w zmiennych członkowskich obiektu, a następnie wyświetlenie kontrolnie na ekranie monitora.

2. Zrób tak jak w zadaniu 1., z tym że dodatkowo uwzględnij hermetyzację — ukrywanie danych komputerów. Dostęp do ukrytych danych zrealizuj za pomocą publicznych metod dostępowych (setterów i getterów).
3. Napisz program pozwalający przetwarzać dane pracowników w szkole. Uwzględnij następujące stanowiska: dyrektor, nauczyciel, wychowawca, sekretarka. Wykonaj program zgodnie z zasadami programowania obiektowego — wykorzystaj zdefiniowane samodzielnie klasy powiązane ze sobą relacjami dziedziczenia. Niech w celach testowych będzie możliwe wprowadzenie danych określonego wychowawcy z klawiatury, zapamiętanie ich w zmiennych członkowskich obiektu, a następnie wyświetlenie kontrolnie na ekranie monitora.
4. Zrób tak jak w zadaniu 3., z tym że dodatkowo uwzględnij hermetyzację — ukrywanie danych pracowników. Dostęp do ukrytych danych zrealizuj za pomocą publicznych setterów i getterów.
5. Napisz program pozwalający przetwarzać dane samochodów w przedsiębiorstwie transportowym. Załóż, że firma ma na stanie samochody ciężarowe i autobusy. Każdy pojazd można opisać za pomocą cech wspólnych: marki, modelu, roku produkcji. Oprócz wymienionych danych samochody ciężarowe charakteryzują się unikatowym przeznaczeniem — wywrotka, chłodnia i cysterna, a autobusy — liczbą miejsc stojących i siedzących. Wykorzystaj mechanizm dziedziczenia. W każdej klasie wchodzącej w skład łańcucha dziedziczenia zdefiniuj konstruktor domyślny i parametryczny. Niech w celach testowych będzie możliwe wprowadzenie danych określonego autobusu z klawiatury, zapamiętanie ich w zmiennych członkowskich obiektu, a następnie wyświetlenie ich kontrolnie na ekranie monitora. Obiekt odpowiadający przetwarzanemu autobusowi utwórz przez wywołanie konstruktora parametrycznego zdefiniowanego w klasie pochodnej, który będzie wywoływał konstruktor parametryczny zdefiniowany w klasie bazowej.
6. Zrób tak jak w zadaniu 5., z tym że dodatkowo uwzględnij hermetyzację — ukrywanie danych samochodów. Dostęp do ukrytych danych zrealizuj za pomocą publicznych setterów i getterów.