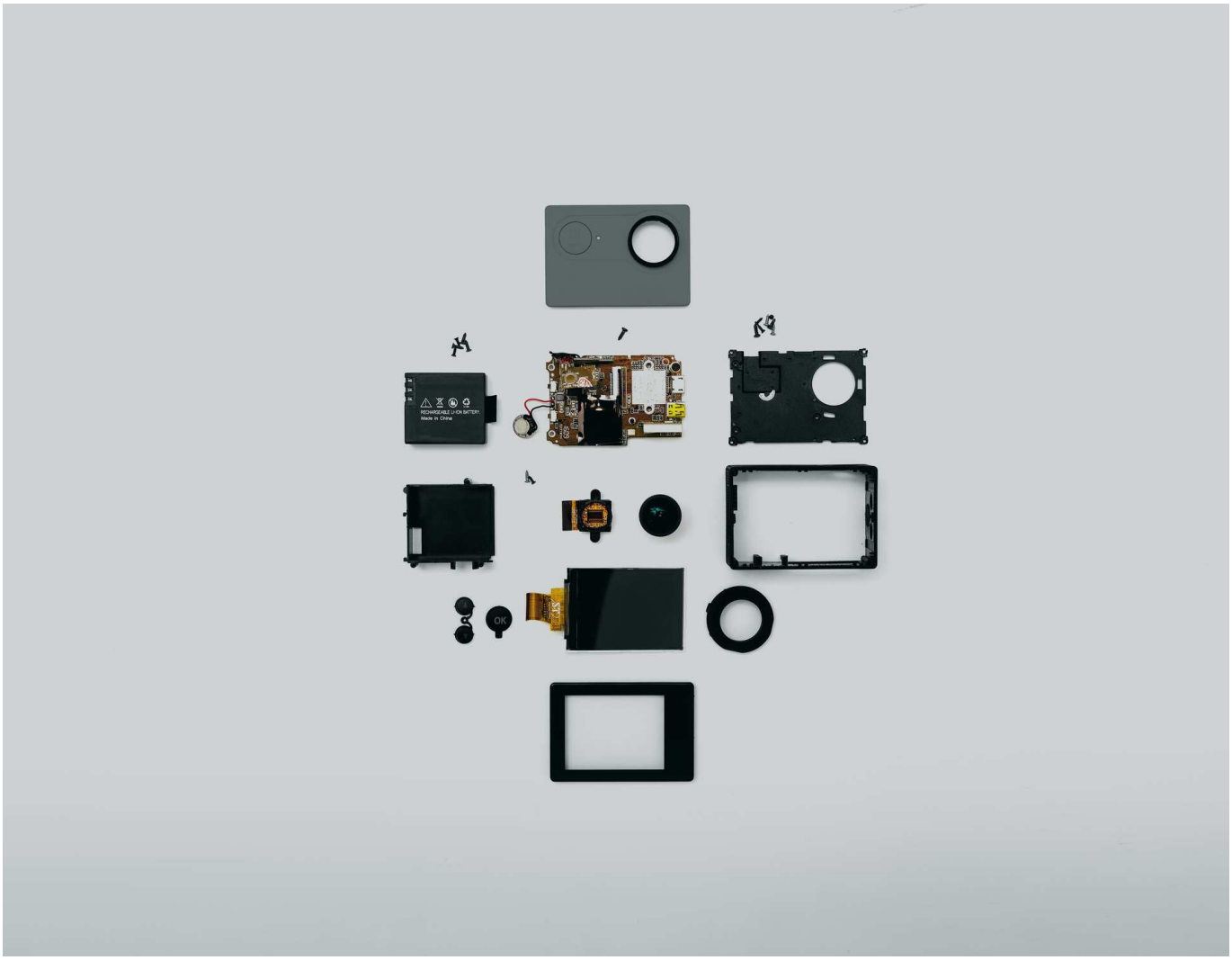

Wstęp do programowania obiektowego w języku C++

Źródło: Alexander Andrews, domena publiczna.

Programowanie obiektowe różni się od programowania strukturalnego tym, że podstawową strukturą danych, z którą pracuje programista, jest obiekt. W omawianym podejściu obiekty posiadają własne metody oraz atrybuty, z których mogą korzystać. Programowanie obiektowe ma za zadanie ułatwić pisanie kodu, sprawić, by był czytelniejszy oraz usprawnić jego edycję.

Podstawowe informacje na ten temat znajdziesz w e-materiale Wstęp do programowania obiektowego.

W tym e-materiale poznamy specyfikę programowania zorientowanego obiektowo w języku C++.

Jeśli chcesz dowiedzieć się, jak to zagadnienie wygląda w przypadku innych języków programowania, sięgnij do e-materiałów:

- Wstęp do programowania obiektowego w języku Java,
- Wstęp do programowania obiektowego w języku Python.

> **Twoje cele**
>
> - Zaimplementujesz klasy i obiekty w języku C++.
> - Zastosujesz dziedziczenie w programie napisanym w języku C++.

- Przeanalizujesz, jak kopiowane są zmienne w języku C++, w tym obiekty, jako argumenty funkcji i metod.

Audiobook

Polecenie 1

Przypomnij sobie najważniejsze informacje dotyczące paradygmatu programowania obiektowego, zapoznając się z audiobookiem przedstawiającym jego charakterystykę.

Audiobook można wysłuchać pod adresem: https://zpe.gov.pl/b/PfoUC4DDt

---

Paradygmat programowania obiektowego powstał z myślą, żeby jak najlepiej odwzorowywać rzeczywistość. Mówi się, że pierwszy język powstał w latach 60 XX. wieku jako wynik prac nad symulacją statków. Jego twórcy mieli bowiem problem z odwzorowaniem wszystkich zależności występujących pomiędzy poszczególnymi statkami w istniejących ówcześnie językach programowania.

Język C++ nie jest jednak językiem w pełni obiektowym, ponieważ pozwala pisać zgodnie z paradygmatem imperatywnym i strukturalnym, przez co jest również przez niektórych krytykowany.

Jednym z najpopularniejszych przykładów odwzorowania rzeczywistości w tym rodzaju programowania jest opis psa. Zastanówmy się więc, co taki przeciętny pies potrafi robić.

Z pewnością potrafi spać, pić, jeść, biegać, bawić się itp. W rozumieniu programowania obiektowego taki pies miałby metody, które odpowiadają powyższym czynnościom.

Ale zastanówmy się teraz, czy kot potrafi spać, pić, jeść, biegać oraz bawić się? Owszem - czy jest zatem jakakolwiek cecha, która psa od kota odróżnia?

Stereotypowy pies i stereotypowy kot różnią się lojalnością oraz zwinnością, jednak skupmy się na samych czynnościach, które obydwa zwierzęta wykonują.

Moglibyśmy teoretycznie stworzyć klasę pies, która byłaby identyczna jak klasa kot, tworząc nadmiarowy kod, ponieważ jedyną różnicą między tymi klasami byłyby poszczególne wartości atrybutów takich jak lojalność czy ulubiony typ pożywienia.

Możemy więc stworzyć klasę o nazwie „zwierzę domowe", która to łączyłaby takie metody jak picie, jedzenie czy spanie oraz atrybut typu lojalność.

Wtedy klasa kot mogłaby dziedziczyć po klasie zwierzę domowe, podobnie jak klasa pies. Dzięki temu obydwa zwierzaki miałyby swoje własne klasy, które mogłyby wypełnić unikalnymi własnościami, bez potrzeby dodawania dwa razy tych samych metod, czyli czynności, które potrafią robić.

Tutaj pojawia się jednak problem – czy klasa zwierzęcia domowego pasuje do złotej rybki? Czyli może rzeczywiście to nie tak, że każde zwierzę domowe potrafi się bawić. Jest to prawda, ponieważ co do zasady umiejętność zabawy nie jest zarezerwowana jedynie dla udomowionych zwierząt. Nawet małe lwy uczą się polowania poprzez zabawę.

Czy metoda zabawa nie powinna znaleźć się w klasie zwierzęcia domowego, ponieważ jest to rzecz bardziej ogólna? Załóżmy przez chwilę, że jedynie ssaki potrafią się bawić.

Teoretycznie rozwiązuje to problem: ssak potrafi się bawić, jeść, pić, spać i robić wszystko, co potrafi pies, bez uwzględnienia jego szczególnego węchu czy kwestii lojalności.

Pojawia się w tym dziedziczeniu kolejny problem, ponieważ delfin już różni się w sposobie poruszania istotnie od ssaka, a umiejętność pożywiania się i spania jest zdolnością każdego żywego zwierzęcia.

Dlatego jeśli tworzylibyśmy program opisujący królestwo zwierząt, to byłby on zbliżony mniej więcej do tego, co sugeruje nam klasyfikacja biologiczna. Przykładowo, każdy ssak dostałby w naszym programie zabawę oraz karmienie mlekiem – jednak już sama metoda karmienia mlekiem byłaby odpowiednio modyfikowana w zależności od dalszego podziału.

Dlatego też bardzo istotne podczas projektowania programów z zastosowaniem technik obiektowych jest szukanie wzorców i podobieństw pomiędzy poszczególnymi obiektami, ponieważ istnieje spora szansa, że dwie klasy mogą zostać uproszczone poprzez dodanie dziedziczenia (podobnie jak w przypadku kota i psa – zwierzęta te dziedziczyły z klasy zwierzę domowe). Warto jednak nadmienić, że o ile nie implementujemy całego królestwa zwierząt, a jedynie jeden szczególny przypadek psa, to nie ma sensu stosować wielopoziomowego dziedziczenia tylko po to, żeby opisać psa, którego wszystkie cechy szczególne i „funkcje” można zawrzeć w ramach jednej klasy i poszczególnych atrybutów.

Polecenie 2

Zaprojektuj schemat prezentujący dziedziczenie. Swoją strukturę spróbuj oprzeć o obserwacje otaczającego świata np. klasa jodła dziedzicząca po klasie iglaste, dziedzicząca po klasie drzewo. Zaproponuj pola i metody dla każdej klasy.

Przeczytaj

Język C++

Zacznijmy od przytoczenia kilku istotnych informacji dotyczących języka C++.

Operator ++ znajdujący się w nazwie języka C++ oznacza inkrementację, czyli kolejne ulepszenia – iteracje, które zostały wprowadzone w języku C. Pamiętajmy, że C++ jest wstecznie zgodny z językiem C.

Język C++ pozwala korzystać z mechanizmów programowania obiektowego i m.in. tym różni się od C, który jest językiem imperatywnym oraz strukturalnym. Jednak dowolny kod napisany w języku C powinien się skompilować za pomocą kompilatora języka w C++.

Podczas nauki języka C++ dowiadujemy się m.in., czym są funkcje, wskaźniki, operatory i struktury.

> **Już wiesz**
>
> Wskaźnik (ang. *pointer*) to zmienna służąca do przechowywania adresu.
> Operator `*` w językach C i C++ jest wykorzystywany w celu odwoływania się do komórki pamięci, znajdującej się pod adresem przechowywanym we wskaźniku.
> Operator `&` w językach C i C++ służy do pobrania adresu zmiennej.

Klasy i obiekty

Zanim przejdziemy do wyjaśnienia, czym jest programowanie obiektowe, omówmy pojęcia: klasa i obiekt.

> **Definicja: Klasa**
>
> Klasa jest opisem (definicją) tworzonych na jej podstawie obiektów. Składa się z pól i metod. Pola (atrybuty) przechowują wartości, które są cechami obiektu, z kolei metody pozwalają na manipulacje tymi wartościami. W języku C++ klasy definiują nowe typy danych. Klasa o nazwie `Pies` definiuje typ `Pies`, który z kolei może składać się z atrybutów: `imię`, `rasa` czy `wiek`.
> **Definicja: Obiekt**
>
> Obiekt jest instancją klasy, powstaje w wyniku utworzenia zmiennej typu obiektowego (danej klasy). Obiekt jest zbudowany z pól zadeklarowanych w klasie. Do zmian wartości pól możemy wykorzystać zaimplementowane w klasie metody. Obiekty możemy jednoznacznie identyfikować (takie rozróżnienie umożliwia wskaźnik `this`).

Z przedstawionych uproszczonych definicji klasy i obiektu wynika, że jedno nie może istnieć bez drugiego (nie można stworzyć obiektu, nie implementując wcześniej klasy). Przeanalizujmy tę relację na przykładzie datownika.

> **Przykład 1**
>
> Datownik jest specjalnym rodzajem pieczątki, umożliwiającym ustawienie dnia, miesiąca i roku, a następnie odciśnięcie ich na kartce papieru. Możemy go porównać do klasy. Datownik składa się z trzech danych (pól): dzień, miesiąc i rok; natomiast system pokręteł pozwala na modyfikacje wartości. Przybicie datownika powoduje odciśnięcie ustawionych na pokrętłach wartości. W ten sposób powstaje instancja klasy – obiekt. Zauważmy, że pieczątkę możemy przybić wielokrotnie. Każde odciśnięcie może przechowywać inne dane, które jednak zawsze będą opierać się na tym samym wzorcu, czyli klasie.

Klasy i obiekty w języku C++

W języku C++ klasy definiujemy w sposób podobny do struktur.

```
1 class NazwaKlasy {
2 };
```

Po słowie kluczowym `class` podajemy nazwę klasy. Dobrym zwyczajem jest rozpoczynanie nazwy klasy wielką literą. Poza tym nazewnictwo klas podlega takim samym zasadom, jakie stosowane są podczas nadawania nazw funkcjom i zmiennym.

W definicji klasy możemy umieścić następujące elementy:

- **atrybuty** (pola) – elementy określające jakąś cechę lub daną przechowywaną wewnątrz obiektu; mogą nimi być dowolne zmienne, w tym obiekty klas albo tablice obiektów klas;
- **metody** – określają zachowanie obiektów; są to funkcje składowe (funkcje związane z klasą) mające dostęp do atrybutów klasy;
- informacje, z jakich klas dziedziczy definiowana klasa;
- **konstruktory** oraz **destruktory** – pierwsze służą do tworzenia instancji klasy, drugie uruchamiane są przed usunięciem obiektu; konstruktor musi nazywać się tak samo jak klasa; nazwa destruktora składa się z nazwy konstruktora poprzedzonej znakiem tyldy (~);
- **specyfikatory dostępu** – określające, kto ma dostęp do elementu klasy.

W uproszczeniu możemy porównać klasę do przepisu w książce kucharskiej, a obiekt do ugotowanej na jego podstawie potrawy.

Przeanalizujmy kod:

```
1  #include <iostream>
2  #include <cstdlib>
3  #include <string>
4
5  using namespace std;
6
7  class NazwaKlasy {
8      private:
9          int atrybut;
10     public:
11         void metoda() {
12             cout << "Cialo metody" << endl;
13         }
14         NazwaKlasy(int a) {
15             atrybut = a;
16             cout << "To jest konstruktor klasy" << endl;
17         }
18         ~NazwaKlasy() {
19             cout << "To jest destruktor klasy" << endl;
20         }
21 };
22
23 int main(int argc, char** argv) {
24     NazwaKlasy dowolnaNazwaObiektu(5);
25     return 0;
26 }
```

Zdefiniowaliśmy klasę `NazwaKlasy`, a następnie utworzyliśmy obiekt o nazwie `dowolnaNazwaObiektu`, korzystając z konstruktora opisanego w linii 14.

Po uruchomieniu programu na ekranie pojawią się dwie linie:

```
To jest konstruktor klasy
To jest destruktor klasy
```

Wynika to z faktu, że przed zakończeniem programu (instrukcja `return 0` w linii 25.) wszystkie obiekty są usuwane.

> **Przykład 2**

Napiszmy program symulujący działanie przywoływanego wcześniej datownika. Data jest opisywana trzema wartościami: dzień, miesiąc i rok – są to pola, które wypełniamy danymi. Datownik umożliwia manipulowanie pokrętłami i ustawienie interesującej nas daty. W programie za tę operację odpowiada metoda `zmienDate`. Odciśnięcie daty na kartce papieru symuluje metoda `odcisnijDate`. Jej zadaniem jest wydrukowanie na standardowym wyjściu wartości ustawionych w polach.

```
#include <iostream>
#include <cstdlib>
#include <string>

using namespace std;

class Datownik {
    protected:
        int dzien;
        int miesiac;
        int rok;
    public:
        void zmienDate(int dzien, int miesiac, int rok)
        {
            this->dzien = dzien;
            this->miesiac = miesiac;
            this->rok = rok;
        }
        Datownik(int dzien, int miesiac, int rok)
        {
            this->dzien = dzien;
            this->miesiac = miesiac;
            this->rok = rok;
        }
        void odcisnijDate()
        {
            cout<<"Data:"<<dzien<<"."<<miesiac<<"."<<rok <<"\n"
        }
};



int main(int argc, char** argv) {
    Datownik datownik1(10,10,2010);
```

```
    datownik1.odcisnijDate();
    return 0;
}
```

Ważne!

W języku C++ istnieją trzy specyfikatory dostępu:

- `public` – informujący, że z każdego miejsca programu mamy dostęp do takiego elementu obiektu;
- `private` – oznaczający, że tylko dany obiekt ma dostęp do danego elementu;
- `protected` – specyfikator podobny do `private`.

Jeśli zastosowany jest konstruktor prywatny, to nie można zarówno utworzyć obiektu danej klasy, jak i dziedziczyć po niej.

Warto podkreślić, że możliwość tworzenia prywatnego konstruktora nie jest błędem twórców języka, a świadomą decyzją. Konstruktor prywatny przydaje się m.in. wtedy, gdy zależy nam istnieniu tylko jednej instancji danej klasy w całym programie (tzw. singleton).

Dziedziczenie

Dziedziczenie jest procesem, podczas którego kopiuje się funkcjonalności oraz zastosowania klasy bazowej do klasy pochodnej (dziedziczącej).

Ciekawostka

Jednym z pięciu podstawowych założeń programowania obiektowego (określanych akronimem SOLID) jest zasada sformułowana przez Barbarę Liskov. Mówi ona, że klasę dziedziczącą powinno dać się użyć w miejscu klasy bazowej. Innymi słowy, dobrą praktyką jest korzystanie z mechanizmu dziedziczenia w taki sposób, aby klasa pochodna nie zmieniała metod klasy bazowej.

Przykład 3

```
#include <iostream>
#include <cstdlib>
#include <string>

using namespace std;

class Pojazd {
    protected:
        int aktualnaPozycjaX;
        int aktualnaPozycjaY;
```

```
11      public:
12          void przemieszczenie(int oIleWx, int oIleWy) {
13              aktualnaPozycjaX += oIleWx;
14              aktualnaPozycjaY += oIleWy;
15          }
16          Pojazd(int aktualnaPozycjaX, int aktualnaPozycjaY) {
17              this->aktualnaPozycjaX = aktualnaPozycjaX;
18              this->aktualnaPozycjaY = aktualnaPozycjaY;
19          }
20  };
21
22  class Motocykl : public  Pojazd {
23      public:
24          Motocykl(int aktualnaPozycjaX, int aktualnaPozycjaY):Po
25              cout << "Jestem motocyklem" << endl;
26          };
27          int pobierzAktualnaPozycjeX() {
28              return aktualnaPozycjaX;
29          }
30          int pobierzAktualnaPozycjeY() {
31              return aktualnaPozycjaY;
32          }
33  };
34
35
36  int main(int argc, char** argv) {
37      Motocykl honda(0, 0);
38      honda.przemieszczenie(5, 10);
39      cout << "Aktualna pozycja x: "<< honda.pobierzAktualnaPozyc
40      cout << "Aktualna pozycja y: " << honda.pobierzAktualnaPozy
41      return 0;
42  }
```

Zasady dziedziczenia

Jeśli klasa, po której dziedziczymy, nie ma konstruktora bezparametrowego, należy jawnie wywołać konstruktor klasy bazowej (tak jak jest to widoczne w linii 24. programu z przykładu 3). Podczas tworzenia obiektu klasy pochodnej zawsze uruchamiany jest konstruktor klasy bazowej, nawet jeśli dzieje się to niejawnie.

Dostępność elementów klasy bazowej w klasie pochodnej zależy od użytych specyfikatorów:

- jeśli danym lub metodom towarzyszy specyfikator `private`, to elementy te nie są dostępne w klasie pochodnej;
- jeśli danym lub metodom towarzyszy specyfikator `public` lub `protected`, są one dostępne w klasie pochodnej.

W linii 23. programu po dwukropku użyliśmy specyfikatora `public`. Warto pamiętać, że jeśli nie zastosujemy żadnego operatora, domyślnie zostanie użyty modyfikator `private`. Specyfikator w tym miejscu służy do ograniczania widoczności w klasie. Jeśli zastosujemy w danej klasie specyfikator `public`, wszystkie jej elementy również przyjmują typ `public` (mają najwyższą możliwą widoczność, czyli w praktyce nie zachodzi żadna zmiana w dostępności).

Modyfikator `protected` umożliwia dostęp do pola bądź metody w obrębie klasy bazowej oraz klas pochodnych (dziedziczących). Próba odwołania się do elementu, który został oznaczony `protected`, spoza wspomnianych wcześniej klas zakończy się błędem kompilacji.

Najbardziej restrykcyjnym modyfikatorem dostępu jest `private`. Uniemożliwia odwołanie się do elementu poprzez klasę dziedziczącą. Sprawa to, że tylko klasa, w której zostało utworzone pole bądź metoda, może korzystać z tak oznaczonego elementu.

> Więcej na temat modyfikatorów dostępu dowiesz się z e-materiału poświęconego paradygmatom programowania obiektowego w języku C++.
> **Ważne!**
>
> W języku C++ istnieje wielodziedziczenie. Oznacza to, że klasa może dziedziczyć po wielu klasach jednocześnie.
>
> Może to powodować problem w sytuacji, gdy dziedziczenie odbywa się po dwóch klasach, w których istnieje metoda o takiej samej nazwie. Przykładowo:

```
1  #include <iostream>
2  using namespace std;
3  class Klasa1
4  {
5      public:
6      void test()
7      {
8          cout<<"test1";
9      }
10 };
11 class Klasa2
12 {
13     public:
```

```
    void test()
    {
        cout<<"test2";
    }
};
class Test : public Klasa1, public Klasa2
{
    public:
    Test(){
        test();
    }
};

int main()
{
    Test test;
    return 0;
}
```

Problem w tym przypadku polega na tym, że kompilator nie jest w stanie stwierdzić, do której metody `test()` powinien się odwołać. Pojawia się błąd wskazujący na niejednoznaczność wywołanej metody:

```
main.cpp:34:9: error: reference to ‘test’ is ambiguous
   34 |         test();
      |         ^~~~
main.cpp:24:10: note: candidates are: ‘void Klasa2::test()’
   24 |     void test()
      |          ^~~~
main.cpp:15:10: note:                 ‘void Klasa1::test()’
   15 |     void test()
      |          ^~~~
```

Wskaźnik `this`

Wskaźnik `this` sygnalizuje początek obszaru pamięci zajmowanego przez obiekt (adres jego pierwszej komórki) i może służyć do jego jednoznacznej identyfikacji. Dwie różne instancje tej samej klasy będą miały dwa różne wskaźniki `this`, ponieważ są od siebie niezależne.

Widoczne jest tu podobieństwo do dwóch różnych zmiennych typu `int`. Mimo że obie zmienne mogą mieć taką samą wartość, to modyfikacja jednej z nich nie wpływa na wartość drugiej, ponieważ obie są zapisane pod innymi adresami w pamięci.

```
1  class Pojazd() {
2      protected:
3          int aktualnaPozycjaX;
4          int aktualnaPozycjaY;
5      public:
6          Pojazd(int aktualnaPozycjaX, int aktualnaPozycjaY) {
7              this->aktualnaPozycjaX = aktualnaPozycjaX;
8              this->aktualnaPozycjaY = aktualnaPozycjaY;
9          }
10 }
```

Użycie wskaźnika `this` (jawne lub niejawne) jest konieczne, jeśli chcemy zapisać dane w polach. Za każdym razem, gdy modyfikujemy atrybut w danym obiekcie, robimy to za pośrednictwem wskaźnika `this`. W poprzednim fragmencie kodu w linii 6. zdefiniowaliśmy argumenty nazwane tak samo jak atrybuty klasy, jednak dzięki użyciu wskaźnika `this` w 7. i 8. linii kompilator rozpoznał, do której komórki pamięci należy przypisać wartości.

Operowanie na obiektach

Obiekty klas – podobnie jak zmienne każdego typu podstawowego – możemy umieścić w tablicy oraz podawać jako argumenty funkcji.

Funkcje w języku C++ domyślnie operują na kopiach zmiennych, które są przekazywane jako parametry. Tak samo jest w przypadku obiektów.

Powinniśmy jednak pamiętać, że obiekty, które są przekazywane jako argumenty do metod i funkcji, są kopiowane jedynie płytko.

```
1  #include <iostream>
2  #include <cstdlib>
3  #include <string>
4
5  using namespace std;
6
7  class NazwaKlasy {
8      public:
```

```
        int a;
        int* tab;
        NazwaKlasy(int a, int * tab) {
            this->a = a;
            this->tab = tab;
        }
};

class DrugaKlasa {
    public:
        static void przykladowaMetoda(NazwaKlasy p) {
            p.tab[0] = 5;
        }
};

int main(int argc, char** argv) {
    int tab[5] = { 0, 1, 2, 3, 4 };
    NazwaKlasy przykladowaKlasa(5, tab);
    DrugaKlasa::przykladowaMetoda(przykladowaKlasa);
    cout << tab[0];
    return 0;
}
```

Wynikiem działania przedstawionego programu będzie wypisanie na ekrannie liczby „5”, ponieważ kopiowanie było płytkie i metoda statyczna `przykladowaMetoda()` operowała na oryginalnej tablicy.

Przy okazji zastosowaliśmy również metodę statyczną, może być ona użyta bez powoływania obiektów. Metody statyczne wywołuje się zgodnie z następującą formułą:

```
NazwaKlasy::nazwaMetody()
```

Słownik

> **funkcja**
>
> inaczej procedura, bądź podprogram, to wydzielona część programu wykonująca jakąś operacje; funkcja może być wykorzystywana wielokrotnie, upraszczając kod źródłowy programu
> **język imperatywny**
>
> paradygmat programowania polegający na podziale programu na szereg instrukcji, które zmieniają stan programu (np. zmiennych); prawie każdy sprzęt komputerowy pracuje

w sposób imperatywny

język strukturalny

paradygmat programowania polegający na podziale kodu źródłowego programu na mniejsze części (procedury i bloki) z wykorzystaniem instrukcji kontrolnych (warunkowych) oraz pętli

kopiowanie płytkie

(ang. *shallow*) polega na utworzeniu kopii obiektu ze skopiowanymi wartościami wszystkich atrybutów; kopiowane są również adresy wskaźników, jeśli któryś z atrybutów tego obiektu był tablicą lub wskaźnikiem

operator

konstrukcja językowa jedno-, bądź wieloargumentowa zwracająca wartość

struktura danych

sposób przechowywania danych w pamięci komputera; na strukturach danych operują algorytmy

SOLID

mnemonik zaproponowany przez Roberta C. Martina, opisujący pięć podstawowych założeń programowania obiektowego: zasady jednej odpowiedzialności (ang. *single responsibility*), zasady otwarte-zamknięte (ang. *open-close*), zasady podstawienia Liskov (ang. *Liskov substitution principle*), zasady segregacji interfejsów (ang. *interface segregation principle*) oraz zasady odwrócenia zależności (ang. *dependency inversion principle*)

wskaźnik

logiczny adres wskazujący na miejsce w pamięci, w którym została umieszczona wartość zmiennej (zarówno statycznej, jak i dynamicznej)

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Napisz program, w którym utworzysz klasę o nazwie `Publikacja`, zawierającą następujące elementy: `tytul`, `autor` oraz `rokWydania` (ich wartości są ustawiane w konstruktorze).

Następnie zaimplementuj w programie klasę `Ksiazka`, która będzie dziedziczyła po klasie `Publikacja`. Nadaj jej również właściwości: `liczbaStron` oraz `numerIsbn`. Klasa `Ksiazka` powinna wykorzystywać konstruktor z odziedziczonej klasy we własnym konstruktorze oraz posiadać metodę `wypiszDane()`, wypisującą wartości wszystkich pól dostępnych w tej klasie. Dodatkowa metoda `ileLatOdWydania()` niech zwraca liczbę całkowitą określającą, ile lat minęło od wydania książki.

Specyfikacja:

Dane:

Publiczne pola klasy `Publikacja`:

- `tytul` – łańcuch znaków
- `autor` – łańcuch znaków
- `rokWydania` – liczba całkowita

Publiczne pola klasy `Ksiazka`:

- `liczbaStron` – liczba naturalna z przedziału [1, 5000]
- `numerIsbn` – łańcuch znaków

Wynik:

Na standardowym wyjściu program wypisuje wartości wprowadzone podczas tworzenia obiektu zgodnie ze schematem: `tytul autor rokWydania liczbaStron numerIsbn`.

Swoje rozwiązanie przetestuj dla obiektu typu `Ksiazka` z następującymi wartościami:
`Alicja w Krainie Czarów` (tytuł), `Lewis Carroll` (autor), `1865` (rok wydania), `241` (liczba stron), `9788377797662` (numer ISBN).

Twoje zadania

1. Program wypisuje oddzielone spacjami wszystkie dane obiektu `Ksiazka`, np.: `Alicja w Krainie Czarów Lewis Carroll 1865 241 9788377797662`.

Ćwiczenie 2

Napisz program, w którym utworzysz klasę o nazwie `Pojazd`, zawierającą następujące elementy: `marka`, `model` oraz `rokProdukcji` (ich wartości są ustawiane w konstruktorze). Dodatkowo klasa powinna posiadać metodę `wypiszDane()` wypisującą wartości wszystkich swoich pól.

Następnie stwórz w programie klasę `Samochod`, która będzie dziedziczyła publicznie z klasy `Pojazd`. Zaimplementuj jej również pola: `liczbaDrzwi` oraz `rodzajPaliwa`. Klasa `Samochod` powinna wykorzystywać konstruktor z odziedziczonej klasy we własnym konstruktorze oraz przesłaniać metodę `wypiszDane()` wypisującą wartości wszystkich pól (ma wykorzystywać implementację tej metody z klasy dziedziczonej). Stwórz metodę `kosztUtrzymania()`, której zadaniem jest zwrócenie łańcucha znaków mówiącego, jak drogi w utrzymaniu jest samochód. Zaproponuj własną implementację tej metody. Pod uwagę weź dwa parametry: `rodzajPaliwa` i `liczbaDrzwi`.

Specyfikacja:

Dane:

Publiczne pola klasy `Pojazd`:

- `marka` – łańcuch znaków
- `model` – łańcuch znaków
- `rokProdukcji` – liczba całkowita

Publiczne pola klasy `Samochod`:

- `liczbaDrzwi` – liczba naturalna z przedziału [2, 5]
- `rodzajPaliwa` – łańcuch znaków

Wynik:

Na standardowym wyjściu program wypisuje wartości wprowadzone podczas tworzenia obiektu zgodnie ze schematem: `marka model rokProdukcji liczbaDrzwi rodzajPaliwa`.

Swoje rozwiązanie przetestuj dla obiektu typu `Samochod` z następującymi wartościami: `Małe autko` (marka), `Compact+` (model), `2019` (rok produkcji), `4` (liczba drzwi), `Elektryk` (rodzaj paliwa).

Twoje zadania

1. Program wypisuje oddzielone spacjami wszystkie dane obiektu `Samochod`, np.: `Małe autko Compact+ 2019 4 Elektryk`.

Ćwiczenie 3

Napisz program, w którym utworzysz klasę o nazwie `Urzadzenie`, zawierającą następujące elementy: `marka`, `model` oraz `rokProdukcji` (ich wartości są ustawiane w konstruktorze). Dodatkowo klasa powinna posiadać metodę `wypiszDane()` wypisującą wartości wszystkich swoich pól.

Następnie stwórz w programie klasę `Telefon`, która będzie dziedziczyła publicznie z klasy `Urzadzenie`. Zaimplementuj jej również pola: `iloscWbudowanejPamieciDyskowej` (wyrażona w GB) oraz `ksiazkaTelefoniczna`. Klasa `Telefon` powinna wykorzystywać konstruktor z odziedziczonej klasy we własnym konstruktorze oraz przesłaniać metodę `wypiszDane()` wypisującą wartości wszystkich właściwości (ma wykorzystywać implementację tej metody z klasy dziedziczonej). Dodaj do klasy `Telefon` metodę `dodajKontakt(nazwaKontaktu)`. Metoda powinna dodawać do pola `ksiazkaTelefoniczna` nowy kontakt.

Na koniec stwórz w programie klasę `Smartfon`, która będzie dziedziczyła publicznie z klasy `Telefon`. Niech posiada ona również swoje właściwości: `systemOperacyjny` oraz `iloscPamieciRAM`. Klasa `Smartfon` powinna wykorzystywać konstruktor z odziedziczonej klasy we własnym konstruktorze oraz przesłaniać metodę `wypiszDane()` wypisującą wartości wszystkich pól (ma wykorzystywać implementację tej metody z klasy dziedziczonej).

Specyfikacja:

Dane:

Publiczne pola klasy `Urzadzenie`:

- `marka` – łańcuch znaków
- `model` – łańcuch znaków
- `rokProdukcji` – liczba całkowita

Publiczne pola klasy `Telefon`:

- `iloscWbudowanejPamieciDyskowej` – liczba naturalna z przedziału [2, 512]
- `ksiazkaTelefoniczna[]` – dziesięcioelementowa tablica łańcuchów znaków

Publiczne pola klasy `Smartfon`:

- `systemOperacyjny` – łańcuch znaków
- `iloscPamieciRAM` – liczba naturalna z przedziału [4, 32]

Wynik:

Na standardowym wyjściu program wypisuje wartości wprowadzone podczas tworzenia obiektu zgodnie ze schematem: `marka model rokProdukcji iloscWbudowanejPamieciDyskowej systemOperacyjny iloscPamieciRAM`.

Swoje rozwiązanie przetestuj dla obiektu typu `Smartfon` z następującymi wartościami: `PhonX` (marka), `PhonX Big` (model), `2021` (rok produkcji), `256` (wbudowana pamięć dyskowa), `SuperOS` (system operacyjny), `16` (pamięć RAM).

Twoje zadania

1. Program wypisuje oddzielone spacjami wszystkie dane obiektu `Smartfon`, np.: `PhonX PhonX Big 2021 256 SuperOS 16`.

Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Wstęp do programowania obiektowego w języku C++

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

I. Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

Treści nauczania – wymagania szczegółowe

I. Rozumienie, analizowanie i rozwiązywanie problemów.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

2) do realizacji rozwiązania problemu dobiera odpowiednią metodę lub technikę algorytmiczną i struktury danych;

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

1) projektuje i tworzy rozbudowane programy w procesie rozwiązywania problemów, wykorzystuje w programach dobrane do algorytmów struktury danych, w tym struktury dynamiczne i korzysta z dostępnych bibliotek dla tych struktur;

2) stosuje zasady programowania strukturalnego i obiektowego w rozwiązywaniu problemów;

3) sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Zaimplementujesz klasy i obiekty w języku C++.
- Zastosujesz dziedziczenie w programie napisanym w języku C++.
- Przeanalizujesz, jak kopiowane są zmienne w języku C++, w tym obiekty, jako argumenty funkcji i metod.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- burza mózgów;
- ćwiczenia praktyczne.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda;
- oprogramowanie dla języka C++, w tym kompilator GCC/G++ 4.5 (lub nowszej wersji) i Code::Blocks 16.01 (lub nowszej wersji), Orwell Dev-C++ 5.11 (lub nowszej wersji) lub Microsoft Visual Studio.

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Wstęp do programowania obiektowego w języku C++". Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Audiobook".

Faza wstępna:

1. Nauczyciel wyświetla temat i cele zajęć zawarte w sekcji „Wprowadzenie". Następnie wspólnie z uczniami ustala kryteria sukcesu.
2. Uczniowie metodą burzy mózgów przypominają sobie najważniejsze informacje związane z programowaniem obiektowym.

Faza realizacyjna:

1. **Praca z multimedium.** Nauczyciel ocenia, na podstawie informacji na platformie, stan przygotowania uczniów do zajęć. Uczniowie, którzy nie zapoznali się z Audiobookiem, robią to na lekcji. Pozostali wykonują Polecenie 2.
2. **Praca z tekstem.** Uczniowie w parach zapoznają się z treściami przedstawionymi w sekcji „Przeczytaj". Wypisują najważniejsze informacje. Nauczyciel w razie potrzeby odpowiada na pytania.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują indywidualnie ćwiczenia nr 1 i 2 z sekcji „Sprawdź się", a następnie porównują swoje odpowiedzi z kolegą lub koleżanką.
4. Chętne lub wybrane osoby prezentują swoje implementacje na forum klasy. Nauczyciel omawia je.

Faza podsumowująca:

1. Nauczyciel wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W kontekście ich realizacji podsumowuje przebieg zajęć, a także wskazuje mocne i słabe strony pracy uczniów.
2. Wybrany uczeń podsumowuje zajęcia, zwracając uwagę na nabyte umiejętności, omawia ewentualne problemy podczas rozwiązania ćwiczeń.

Praca domowa:

1. Uczniowie wykonują ćwiczenie 3 z sekcji „Sprawdź się".
2. Uczniowie, którzy nie zrobili tego na lekcji, wykonują Polecenie 2 z sekcji „Audiobook".

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka C++.
- Oficjalna dokumentacja techniczna dla kompilatora GCC/G++ 4.5 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania Code::Blocks 16.01 (lub nowszej wersji), Orwell Dev-C++ 5.11 (lub nowszej wersji) lub Microsoft Visual Studio.

Wskazówki metodyczne:

- Uczniowie mogą wykorzystać treści w sekcjach: „Audiobook”, „Przeczytaj”, „Sprawdź się” jako materiał do lekcji powtórkowej.