

Temat: Klasy w języku C++

Klasy:

- ▶ **Klasa** - definiuje dane (zmiennie) oraz zachowanie bytów (funkcje) operujących na tych danych.
- ▶ Zmienne w klasie nazywamy polami, a funkcje na nich działające nazywamy metodami.
- ▶ Klasy stanowi wzór, na podstawie którego powołane są do życia konkretne jej egzemplarze zwane obiektami.
- ▶ Różnica pomiędzy strukturą a klasą polega na tym, że w strukturach wszystkie pola / metody są domyślnie publiczne, a w klasach prywatne.

Jak używać klas:

- ▶ Konstrukcja klas jest analogiczna do konstrukcji struktur
- ▶ `class nazwa {};`

```
1  #include <iostream>
2  using namespace std;
3
4  class Klasa {
5      int pole;
6  public:
7      Klasa(int wartosc) {
8          pole=wartosc;
9      }
10     void metoda() {
11         cout<<"pole="<<+pole<<endl;
12     }
13 };
14
15 int main()
16 {
17     Klasa k(10);
18     k.metoda();
19     k.metoda();
20     k.metoda();
21     return 0;
22 }
```

```
pole=11
pole=12
pole=13
```

Hermetyzacja:

- ▶ **Hermetyzacja** - pozwala sterować dostępem do pól i metod, które nie powinny być dostępne "na zewnątrz" w sposób bezpośredni.

```

1  #include <iostream>
2  using namespace std;
3
4  class Herm {
5      int pole_prywatne1;
6      void metoda_prywatna1 () {cout<<"Pryw1"<<endl;}
7  private:
8      int pole_prywatne2;
9      void metoda_prywatna2 () {cout<<"Pryw2"<<endl;}
10 public:
11     int pole_publiczne;
12     void metoda_publiczna () {cout<<"Publ"<<endl;}
13 protected:
14     int pole_chronione;
15     void metoda_chroniona () {cout<<"Chron"<<endl;}
16 };
17
18 int main()
19 {
20     Herm p;
21     p.pole_publiczne=12;
22     p.metoda_publiczna ();
23     return 0;
24 }
    
```

Publ

- ▶ **public** – elementy zadeklarowane w tej sekcji we wszystkich miejscach programu,
- ▶ **private** – elementy zadeklarowane w tej sekcji mogą być wykorzystywane wyłącznie przez metody składowe danej klasy.
- ▶ **protected** – elementy zadeklarowane w tej sekcji mogą być wykorzystywane przez metody składowe danej klasy lub klasy po niej dziedziczące (więcej pojawi się na kolejnych zajęciach)

Wskaźnik do obiektu this:

```

1  #include <iostream>
2  using namespace std;
3
4  class TH {
5      int pole;
6  public:
7      TH(int pole) {
8          this->pole=pole;
9      }
10     void metoda() {
11         ++this->pole;
12         cout<<"pole="<<this->metoda2 ()<<endl;
13     }
14     int metoda2() {
15         return this->pole;
16     }
17 };
18
19 int main()
20 {
21     TH k(10);
22     k.metoda ();
23     cout<<"odczytano:"<<k.metoda2 ()<<endl;
24     return 0;
25 }
    
```

```

class TH2 {
    int pole;
public:
    TH2(int pole) {
        this->pole=pole;
    }
    void metoda () {
        ++pole;
        cout<<"pole="<<metoda2 ()<<endl;
    }
    int metoda2 () {
        return pole;
    }
};
    
```

this->nazwa_pola daje dostęp do nazwa_pola,
this-> można opuścić, z wyjątkiem:
this->pole (ponieważ nazwa argumentu metody jest taka sama jak nazwa pola)
this->pole ⇔ (*this).pole

Przeciążanie metod:

```

1  #include <iostream>
2  using namespace std;
3
4  class PM {
5      double bilans=0;
6  public:
7      void dodaj(int a) {bilans+=a;}
8      void dodaj(double a) {bilans+=a;}
9      void dodaj(int a, int b) {bilans+=a+b;}
10     void dodaj(double a, double b) {bilans+=a+b;}
11     void dodaj(int a, double b) {bilans+=a+b;}
12     void dodaj(double a, int b) {bilans+=a+b;}
13     void wypisz() {cout<<"bilans="<<bilans<<endl;}
14 };
15 int main()
16 {
17     PM p;
18     p.dodaj(1,1.2);
19     p.dodaj(1);
20     p.dodaj(2.2,3.3);
21     p.wypisz();
22     return 0;
23 }

```

Lista inicjalizacyjna konstruktora:

```

1  #include <iostream>
2  using namespace std;
3
4  class Prost {
5      double a, b, P, Ob;
6  public:
7      Prost(double x, double y): a(x), b(y) {
8          P=a*b;
9          Ob=2*a+2*b;
10     }
11     void wypisz() {
12         cout<<"a="<<a<<" b="<<b<<endl;
13         cout<<"P="<<P<<endl;
14         cout<<"Ob="<<Ob<<endl<<endl;
15     }
16 };
17
18 int main()
19 {
20     Prost p1(1,2), p2(3.3,4.4);
21     p1.wypisz();
22     p2.wypisz();
23     return 0;
24 }

```

```

a=1, b=2
P=2
Ob=6

a=3.3, b=4.4
P=14.52
Ob=15.4

```

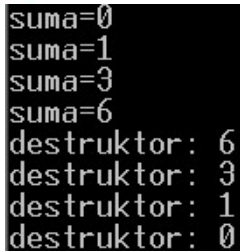
this->nazwa_pola daje dostęp do nazwa_pola,
this-> można opuścić, z wyjątkiem: **this->pole** (ponieważ nazwa argumentu metody jest taka sama jak nazwa pola)
this->pole ⇔ (*this).pole

- ▶ Lista inicjalizacyjna w czytelny sposób oddziela część konstruktora, odpowiedzialną za nadanie wartości pól, od części konstruktora, realizującej bardziej złożone operacje na polach w celu przygotowania obiektu do pracy.

Wiele konstruktorów, jeden destruktork:

- ▶ Klasa może mieć wiele konstruktorów. Muszą one się różnić typami lub ilością parametrów. Domyślnie konstruktor jest bezargumentowy i nie trzeba go definiować.
- ▶ Klasa może posiadać tylko jeden destruktork. Destruktork nie posiada parametrów. Jeśli nie ma takiej potrzeby, to nie trzeba go definiować.

```
1  #include <iostream>
2  using namespace std;
3
4  class Suma {
5      int a,b,c;
6  public:
7      Suma() {a=0; b=0; c=0;}
8      Suma(int x) {a=x; b=0; c=0;}
9      Suma(int x, int y) {a=x; b=y; c=0;}
10     Suma(int x, int y, int z) {a=x; b=y; c=z;}
11     ~Suma() {cout<<"destruktor: "<<suma()<<endl;}
12     int suma() {return a+b+c;}
13     void wypisz() {cout<<"suma="<<suma()<<endl;}
14 };
15
16 int main()
17 {
18     Suma s1, s2(1), s3(1,2), s4(1,2,3);
19     s1.wypisz();
20     s2.wypisz();
21     s3.wypisz();
22     s4.wypisz();
23     return 0;
24 }
```



```
suma=0
suma=1
suma=3
suma=6
destruktor: 6
destruktor: 3
destruktor: 1
destruktor: 0
```


Pole i metody statyczne:

- ▶ Pole statyczne (zmienna klasowa) jest wspólna dla wszystkich obiektów danej klasy.

```
1  #include <iostream>
2  using namespace std;
3
4  class ST {
5      static int pole;
6  public:
7      void metoda() {
8          ++pole;
9          cout<<"pole="<<pole<<endl;
10     }
11 };
12 int ST::pole=0; //definicja pola statycznego
13
14 int main()
15 {
16     ST s1, s2;
17     s1.metoda();
18     s1.metoda();
19     s2.metoda();
20     return 0;
21 }
```

- ▶ Metody statyczne mogą być wywoływane, gdy nie ma obiektów. Trzeba użyć operatora zakresu.
- ▶ Metody statyczne mogą jedynie korzystać z pól statycznych i innych metod statycznych. Nie ma wskaźnika this.

```
1  #include <iostream>
2  using namespace std;
3
4  class MST {
5      static int pole;
6  public:
7      static void metoda() {
8          ++pole;
9          cout<<"pole="<<pole<<endl;
10     }
11 };
12
13 int MST::pole=0;
14
15 int main()
16 {
17     MST::metoda();
18     MST::metoda();
19     MST::metoda();
20     MST m;
21     m.metoda();
22     return 0;
23 }
```

```
pole=1
pole=2
pole=3
pole=4
```

Referencje:

- ▶ Wraz ze wskaźnikami poznaliśmy referencje, które w pewnym sensie nadają nową nazwę zmiennej:
 - ▶ `Typ & nazwa = zmienna;`
- ▶ Poprzez referencje przekazujemy **obiekty** jako **argumenty metod**.

```

1  #include <iostream>
2  using namespace std;
3
4  class Kopia {
5      double a=0, b=0;
6  public:
7      Kopia(double x, double y) {a=x; b=y;}
8      Kopia(const Kopia& k) {a=k.a; b=k.b;}
9      ~Kopia() {cout<<"destruktor: "<<suma()<<endl;}
10     int suma() {return a+b;}
11     void wypisz() {cout<<"Suma="<<suma()<<endl;}
12 };
13
14 int main()
15 {
16     Kopia k1(1,2), k2(3,4), k3(k1), k4(k2);
17     k1.wypisz();
18     k2.wypisz();
19     k3.wypisz();
20     k4.wypisz();
21     return 0;
22 }

```

Operator & oznacza referencję, umieszczony w deklaracji parametru oznacza przekazanie przez zmienną.

Zadania do samodzielnej realizacji:

- (2p) Zdefiniuj klasę o nazwie **Zliczanie** zawierającą:
 - jedno statyczne pole prywatne,
 - statyczną metodę **ile()** zwracającą liczbę aktualnie istniejących obiektów tej klasy.
 Przetestuj działanie metody **ile()**.
- (3p) Zdefiniuj klasę o nazwie **Prost** zawierającą:
 - dwa pola prywatne **a** i **b** typu *double* opisujące długości boków prostokąta,
 - konstruktor domyślny (bez parametrów) tworzący kwadrat o boku 1,
 - konstruktor jednoparametrowy tworzący kwadrat o podanym boku,
 - konstruktor dwuparametrowy tworzący prostokąt o podanych bokach,
 - metody **getA()** i **getB()** zwracające odpowiednie boki prostokąta,
 - metodę **dłPrzekatnej()** zwracającą długość przekątnej prostokąta,
 - metodę **pole()** zwracającą pole powierzchni prostokąta,
 - metodę **obwod()** zwracającą obwód prostokąta,
 - metodę **czyWiększyNiz(Prost&)** zwracającą *true* gdy ten prostokąt ma większe pole od tego przekazanego w parametrze, a *false* w przeciwnym przypadku,
 - metodę **info()** wypisującą informację o prostokącie, np. w formie "Prost[a, b]", gdzie *a* i *b* to wartości długości boków prostokąta.
 Przetestuj wszystkie zdefiniowane konstruktory i metody.