Obiektowość

OOP, (*Object-oriented programming*), czyli programowanie obiektowe, to paradygmat programowania, w którym - w odróżnieniu od programowania proceduralnego, gdzie procedury nie są ze sobą powiązane i to od programisty zależy pilnowanie porządku – program składać się będzie w dużej mierze z *obiektów,* z których każdy ma jakiś zestaw cech, czyli pól przechowujących dane oraz zestaw zachowań, zwanych metodami.

W przełożeniu na przykłady mniej abstrakcyjne – Samochód ma zestaw cech, czyli markę, model, silnik, kolor, kierunkowskazy, bycie lepszym od tramwaju itd. oraz określony zestaw zachowań, czyli robienie „brum brum", włączanie kierunkowskazów itd. Taki samochód przeniesiony na rzeczywistość programistyczną byłby **obiektem.** Takich obiektów w ramach programu możemy utworzyć bardzo dużo, jednak niepraktycznym byłoby tworzenie nowej definicji samochodu za każdym razem, kiedy chcielibyśmy utworzyć taki obiekt.

Klasy

Klasę możemy rozumieć jako wzór, według którego będą tworzone nowe obiekty. Obiekty tej samej klasy będą mieć te same metody i pola, jednak ich wartości mogą już być zupełnie inne. Definicja wygląda bardzo podobnie do struktury.

```
class Car{
        public:
            int pole;
            void metoda();
    };
```

Słowo kluczowe 'class' w połączeniu z nazwą klasy i klamrą zawierającą definicje pół i metod utworzy nam nową klasę. W ten sposób utworzyliśmy jedynie definicję klasy, nie mamy jednak jeszcze żadnego obiektu tej klasy. Znów, podobnie jak w strukturach, możemy o klasie pomyśleć trochę jak o typie danych, a o obiekcie jak o nazwie nowotworzonej zmiennej. W poniższym przkładzie definiujemy sobie klasę Car, która zawiera pola 'make', 'model' i 'engine' oraz metodę broom() robiącą 'BRUM BRUM'. Poniżej tworzony jest obiekt hylaVehicle należący do klasy Car.

```
class Car{
        public:
            string make;
            string model;
            int silnik;
            void vroom(){
                cout<<"BRUM BRUM";
            };
    };

    car hylaVehicle;
```

Do pól i metod (tych publicznych, ale o tym następnym razem), dostęp uzyskujemy dokładnie tak samo jak w przypadku struktur, czyli operatorem wyłuskania (.)

```
    Car hylaVehicle;
    hylaVehicle.vroom();
    hylaVehicle.make="Citroen";
    cout<<hylaVehicle.make;
```

Zapewne pamiętacie, jak wcześniej kładłem nacisk na rozdzielenie deklaracji i definicji funkcji. W programowaniu obiektowym jest tak samo, tylko rozdzielać powinniśmy deklarację i definicję metody. W ciele klasy powinniśmy pozostawić jedynie deklarację, natomiast definicja powinna się znaleźć poza nim.

```
class Car{
    public:
        string make;
        string model;
        int engine;
        void vroom();
};
void Car::vroom(){
    cout<<"BRUM BRUM\n";
}
```

Wewnątrz metod możemy uzyskiwać również bezproblemowy dostęp do pól danej klasy. Jednak operator (.) działa tylko dla obiektów, a w tym wypadku tworzyć będziemy dopiero schemat działania dla wszystkich obiektów definiowanej klasy. W takim wypadku korzystać będziemy ze słówka kluczowego this.

W poniższym przykładzie klasa Car ma pole engine oraz metodę vroom().

```
class Car{
    public:
        string engine;
        void vroom();
};
void Car::vroom(){
    if(this->engine == "Benzyna")
        cout<<"BRUM BRUM\n";
    if(this->engine == "Diesel")
        cout<<"KLE KLE\n";
}

int main(){

    Car Citroen, Thalia;
    Citroen.engine="Diesel";
    Thalia.engine="Benzyna";
    Thalia.vroom();
    return 0;
}
```

Zwrócić należy uwagę, że zachowanie metody vroom() jest inne, zależnie od zdefiniowanego w obiekcie silnika. Mimo, że Citroen i Thalia są obiektami klasy Car i dzielą ze sobą pewne wspólne cechy (oba mają silnik i można je wkręcić na obroty) to ich zachowanie jest zupełnie inne, zależnie od rodzaju silnika.

Zadania

1. Napisz klasę samochód, która zawierać będzie pola: marka, model, pojemność silnika, rodzaj paliwa oraz metody:
    a. create() – która zapyta użytkownika o wszystkie dane pojazdu
    b. show() – która wyświetli wszystkie te informacje.

    W funkcji main powinniśmy jedynie tworzyć obiekt klasy samochód i wywoływać obie metody.
2. Napisz klasę rectangle, która będzie miała pola a i b oraz metody pozwalające na obliczanie pola i przekątnej tego prostokąta. W funkcji main stwórz nowy obiekt, poproś użytkownika o podanie a i b, a następnie wywołaj obie metody i wynik obliczeń wydrukuj na terminalu.