Obiekty i klasy

ZPK 2016

8 marca 2017

oraz paradygmat programowania obiektowego.

Zanim zaczniemy

Kilka uwag o stylu kodowania.

`http://courses.cms.caltech.edu/cs11/material/cpp/donnie/cppstyle.html`

- Prawidłowo rozmieszczaj spacje
- Konsekwentnie rozmieszczaj nawiasy klamrowe
- Używaj nawiasów klamrowych jeżeli instrukcja zajmuje więcej niż jedną linię.
- Używaj wcięć
- Zwróć uwagę na szerokość tekstu
- Używaj nawiasów do oznaczenia kolejności działań
- Wykorzystuj puste linie do oddzielenia bloków kodu
- Pisz komentarze

Klasy i obiekty

Obiekt łączy w sobie:

- *stan* - konkretny zestaw danych
- *zachowanie* - kod działający na tych danych
- obiekt = dane + kod

Klasa to sztanca, za pomocą której formowane są obiekty.

Każda klasa definiuje nowy typ w programie.

Minimalny przykład


```
class Point
{
};

int main()
{
    Point p;
}
```

Oczywiście obiekty tak zadeklarowanej klasy Point są puste i nie można nic sensownego z nimi zrobić. Ale powyższy kod się skompiluje.

- Klasa składa się ze składowych (*members*).
- Składowymi są albo dane (zmienne) albo metody (funkcje).
- Dane przechowują stan klasy; metody pozwalają ten stan modyfikować (określają zachowanie klasy)
- Można tworzyć wiele obiektów z danej klasy. Każdy ma swój zestaw zmiennych (danych - stan).
- Wywoływanie metod obiektu zmienia stan wyłącznie tego obiektu, nie zmieniając stanów innych obiektów z danej klasy (są wyjątki - składowe statyczne).
- Pojedyncze obiekty nazywamy instancjami danej klasy.
- Klasa nie jest obiektem.

Dane i ich inicjalizacja


```
class Point
{
    double x,y;

public:
    Point(double _x, double _y) {
        x = _x;
        y = _y;
    }
};
```

- Sekcje publiczne i prywatne
- Zazwyczaj chcemy ukryć zmienne klasy

Konstruktory i destruktory

- Konstruktor tworzy nową instancję klasy.
- Konstruktor może mieć argumenty, ale nie musi. Konstruktor bezargumentowy to konstruktor domyślny.
- Konstruktor nie zwraca żadnej wartości.
- Może być wiele konstruktorów, zawsze jest przynajmniej jeden.

- Destruktor "sprząta" po obiekcie.
- Nie ma parametrów i nie zwraca wartości.
- Jest tylko jeden destruktor.

Destruktory zaczną nam być potrzebne dopiero, gdy zaczniemy umieszczać obiekty na stercie.

Jak modyfikować wartości zmiennych?

Zmienne są w sekcji prywatnej - jak je modyfikować?

```
class Point
{
        double x,y;

public:
        double getX() { return x; }
        double getY() { return y; }

        void setX(double _x) { x = _x; }
        void setY(double _y) { y = _y; }
};
```

Metody

- Metody dostępowe

  Umożliwiają odczytanie stanu obiektu.
- Metody ustawiające

  Umożliwiają ustawienie stanu obiektu.
- Metody modyfikujące

  Metody umożliwiające działanie na obiekcie.
- Modyfikatory private:, protected: i public:

Projektowanie klas

Myślenie obiektowe

- Uchwycenie istotnych cech obiektu (abstraction)
- Oddzielenie funkcjonalności od implementacji (encapsulation)

Proste na płaszczyźnie

Chcemy utworzyć klasę, której obiekty reprezentują proste na płaszczyźnie.

Chcemy móc sprawdzać równoległość i wyznaczać punkt przecięcia dwóch prostych.

```
class Line
{
public:
        bool parallelTo(Line);
        Point intersectionWith(Line);
};
```

(Powyższa klasa jest niepełna - brak implementacji metod i zmiennych klasy. Uwaga na różnicę pomiędzy deklaracją i definicją funkcji/metody.)

Jak implementować klasę Line?

Musimy zdecydować, jak reprezentować nasze proste.

1. Pamiętając współczynniki A, B, C równania $Ax + By + C = 0$?
2. Pamiętając parę punktów P, Q, przez które przechodzi prosta?
3. Pamiętając punkt zaczepienia P i wektor kierunkowy v prostej?

Jak implementować klasę Line?

Zwróć uwagę, że już samo pytanie o *równość* dwóch prostych nie jest łatwe w żadnej z tych reprezentacji.

Szczegóły implementacji metod parallelTo i intersectionWith są zupełnie inne w każdym przypadku.

Nie jest to jednak istotne dla użytkownika klasy (jak długo umie on tworzyć i modyfikować obiekty tej klasy).

Do klasy Line wrócimy na następnych zajęciach

Implementacja klas

Implementacja klasy Point


```
class Point
{
    double x, y;

public:
    Point();
    Point(double, double);

    ~Point();

    void setX(double);
    void setY(double);

    double getX();
    double getY();

    friend istream& operator>>(istream&, Point&);
};

ostream& operator<<(ostream &, Point);
istream& operator>>(istream &, Point&);
```

Wskaźniki kontra referencje

Metody obiektu zawsze przekazują zmienne przez wartość.

By oszczędzić na kopiowaniu używamy referencji.

Operatory « i »


```
ostream& operator<<(ostream &o, Point p)
{
    o << "(" << p.getX() << "," << p.getY() << ")";

    return o;
}

istream& operator>>(istream &i, Point &p)
{
    i >> p.x;
    i >> p.y;

    return i;
}
```

Zadanie


```
class Plane {
public:
        Plane(Point A, Point B, Point C);
        moveTo(Point P);
        distanceFrom(Point P);
        parallelTo(Plane &P);
};
```