C++ - klasa a obiekt

Klasa:

Jest to definicja wzorca (opis wszystkich danych i dostępnych metod) według którego będą tworzone obiekty danego typu. Innymi słowy klasa to nowy typ zmiennych definiowany przez użytkownika.

Obiekt:

Obiekty są to instancje konkretnych klas (typów własnych).

Referencje

Język C++ wprowadza typ referencyjny. Definiowany jest on w następujący sposób:

```
Typ & nazwaZmiennejReferencyjnej = nazwaZmiennej;
```

lub lepiej:

```
auto & nazwaZmiennejReferencyjnej = nazwaZmiennej;
```

Słowo auto oznacza, że kompilator sam podstawi najbardziej odpowiedni typ zmiennej. Referencje należy traktować jako dodatkowe nazwy tego samego obiektu. Z tej przyczyny niemożliwe jest stworzenie pustej referencji:

```
auto & nazwaZmiennejReferencyjnej; // blad kompilatora
```

Referencje

Obiektu wskazywanego przez referencję nie można podmienić tak jak to się dzieje w przypadku wskaźników. Niemożliwa jest następująca operacja:

```
auto & nazwaZmiennejReferencyjnej = nazwaZmiennej;
nazwaZmiennejReferencyjnej = nazwaInnejZmiennej; //blad
//  kompilatora
```

Od momentu utworzenia referencji, można jej używać tak samo jak można używać obiektu. Należy pamiętać, że każda operacja wykonana na referencji oznacza, że operacja ta jest wykonywana na obiekcie do którego się ona odnosi.

```
int i = 1;          // i = 1
// int & ri - Uwaga Blad
int & ri = i;       // i = 1
ri++;               // i = 3
```

Referencje

Referencje są szeroko wykorzystywane do przekazywania obiektów do funkcji lub metod, zwłaszcza gdy przekazywany obiekt, jest duży i jego kopiowanie jest nieoptymalne.

```
void swap(double& a, double& b){
    double & tmp = a;
    a = b;
    b = tmp;
}
int main(void){
int x = 1, y = 2;
swap(x,y);
// x == 2, y==1
return 0;
}
```

Ponieważ do funkcji zamieniającej miejscami przekazywane są referencje, to efekt jest taki, że zmienne przekazane do funkcji zamienią się wartościami.

Definicja klasy

Poniższa klasa łączy wartość kąta ze sposobem w jaki jest wyrażany. Ponieważ wartość ta może być wyrażona w stopniach, radianach, ... istnieje mniejsza szansa na zrobienie pomyłki.

```
class DegreeAngle                                          //(1)
{
public:                                                    //(2)
    DegreeAngle(void);                                     //(3)
    DegreeAngle(double angle);                             //(4)
    DegreeAngle(const DegreeAngle& angle);                 //(5)

    double value(void) const { return value_; }            //(6)

private:                                                   //(7)
    double value_ = 0;                                     //(8)
};                                                         //(9)
```

Uwaga. Definicję klasy należy rozumieć jako wszystkie niezbędne informacje kompilatorowi do określenia jej interfejsu oraz rozmiaru jaki posiada obiekt tej klasy (rozmiary i ilość pól klasy)

Definicja klasy

- (1) - słowo kluczowe class informujące, że definiowany typ jest klasą oraz nazwa klasy. Nazwa klasy musi być unikalna w całym systemie (uwaga: przestrzenie nazwa pomagają niekiedy zapewnić unikalność nazw)
- (2) - instrukcje poniżej tego zapisu stanowią publiczny interfejs klasy
- (3) - deklaracja konstruktora domyślnego
- (4) - deklaracja konstruktora przyjmującego jeden parametr
- (5) - deklaracja konstruktora kopiującego
- (6) - funkcja zwracająca wartość zapisaną w polu klasy, krótkie funkcje mogą być implementowane w ciele klasy
- (7) - instrukcje poniżej tego zapisu stanowią szczegóły implementacyjne, niedostępne poza klasą
- (8) - pole, w którym zapisywana jest wartość, standard C++11 umożliwia inicjalizację niestatycznych pól klasy również w ten sposób
- (9) - koniec definicji klasy (uwaga: bardzo ważny średnik)

Implementacja metod klasy

```
DegreeAngle::DegreeAngle(void) {}                              //(1)

DegreeAngle::DegreeAngle(double angle)                         //(2)
    : value_(angle) {}                                         //(3)

DegreeAngle::DegreeAngle(const DegreeAngle& angle)             //(4)
    : DegreeAngle(angle.value_) {}                             //(5)
```

Szczegóły implementacyjne poszczególnych metod, w tym konstruktorów nie stanowią interfejsu klasy. Mogą być zapisane pod klasą lub innym pliku. Implementacja metody nie może pojawić się wcześniej niż jej deklaracja w ciele .

Implementacja metod klasy

- (1) - implementacja konstruktora domyślnego. DegreeAngle::DegreeAngle oznacza, że chodzi o metodę DegreeAngle klasy DegreeAngle - czyli jej konstruktor
- (2) - implementacja konstruktora przyjmującego jeden argument.
- (3) - lista inicjalizacyjna konstruktora. wymieniana jest po dwukropku (:) przed ciałem metody między klamrami (). istnieje możliwość inicjowania w ten sposób wielu zmiennych wymienionych po przecinku
- (4) - implementacja konstruktora kopiującego.
- (5) - na liście inicjalizacyjnej można wywołać również inny konstruktor, w c++ kładzie się silny nacisk na re-używanie kodu

Zakres dostępu do elementów i metod klasy

publiczny

- słowo kluczowe: public
- możliwe jest wywoływanie metod oraz modyfikowanie pól klasy poza klasą
- zbyt szeroka zakres publiczny przeczy idei hermetyzacji
- publiczne powinny być jedynie te pola i metody które są niezbędne to realizacji odpowiedzialności obiektu

prywatny

- słowo kluczowe: private
- zabronione jest wywoływanie metod oraz modyfikowanie pól poza klasą
- jedynie inne metody tej klasy mogą wywoływać metody prywatne i modyfikować prywatne pola klasy

Pola klasy

```
private:
    double value_ = 0;
```

- reprezentują stan wewnętrzny obiektu
- w miarę możliwości powinny pozostawać prywatne
- mogą być to typy podstawowe lub inne typy złożone
- pola mogą być inicjalizowane wartością lub innym obiektem
- klasa może mieć dowolną ilość pól, jednak ich zbyt duża ilość sugeruje, że klasa ma zbyt wiele odpowiedzialności!

Metody klasy

```
public:
    DegreeAngle(void);
    DegreeAngle(double angle);
    DegreeAngle(const DegreeAngle& angle);

    double value(void) const;
```

- reprezentują czynności jakie mogą być wykonane na obiektach
- mogą być prywatne i publiczne
- jedna klasa może mieć kilka metod o tej samej nazwie, jednak muszą się one różnić typem lub ilością przyjmowanych argumentów - polimorfizm statyczny
- klasa może mieć dowolną ilość metod

Metody klasy

```
public:
    double value(void) const;
```

- mają dostęp do wszystkich pól prywatnych i publicznych danego obiektu.
- mogą wywoływać wszystkie metody prywatne i publiczne niebędące konstruktorami
- deklaracja znajduje się w pliku nagłówkowym (pliku definicji)
- implementacja może (powinna) znajdować w pliku implementacji
- metody z modyfikatorem const oznaczają, że dana metoda nie zmienia stanu wewnętrznego obiektu
- metody z modyfikatorem const mogą być wywoływane dla stałych obiektów oraz referencji

Metody specjalne - konstruktory

```
public:
    DegreeAngle(void);
    DegreeAngle(double angle);
    DegreeAngle(const DegreeAngle& angle);
```

- tak samo jak inne metody mogą być przeciążane,
- wywoływane są w momencie tworzenia obiektu / służy do inicjalizowania obiektu
- brak zwracanego typu
- nazwa identyczna z nazwą klasy
- mogą być metodami publicznymi i prywatnymi
- można wyodrębnić kilka rodzajów konstruktorów: domyślny (bezparametrowy), zwykły, kopiujący oraz przenoszący

Metody specjalne - konstruktor domyślny

```
public:
    DegreeAngle(void);
```

- brak jakiegokolwiek konstruktora spowoduje, że kompilator wygeneruje konstruktor domyślny postaci:

```
DegreeAngle(void) {}
```

- w innym przypadku konstruktor taki musi napisać programista
- istnieje możliwość wymuszenia wygenerowania konstruktora domyślnego przez kompilator:

```
DegreeAngle(void) = default;
```

Metody specjalne - konstruktor kopiujący

```
public:
    DegreeAngle(const DegreeAngle& angle);
```

- służy do poprawnego kopiowania obiektów
  - inicjalizacja jednego obiektu innym:

```
DegreeAngle a1(45);
DegreeAngle a2(a1);
DegreeAngle a3 = a2;
```

  - przekazywanie obiektów do funkcji/metody:

```
double sin(DegreeAngle angle) {/*...*/}
```

  - zwracanie obiektów z funkcji/metody:

```
DegreeAngle getCurrentAngle(void) {
    DegreeAngle a;
    /*...*/
    return a;
}
```

Metody specjalne - konstruktor przenoszący

```
public:
    DegreeAngle( DegreeAngle&&a)
        : value_(std::move(t.value_)) {}
```

- służy do przeniesienia danych z innego obiektu, zostawiając go w stanie nieprawidłowym
  - inicjalizacja jednego obiektu innym

    ```
    DegreeAngle a1;
    DegreeAngle a2 = std::move(a1);
    ```

  - przekazywanie obiektów do funkcji/metody

    ```
    sin(std::move(a2));
    ```

  - zwracanie obiektów z funkcji/metody

    ```
    DegreeAngle getCurrentAngle(void) {
        DegreeAngle a;
        /*...*/
        return a;
    }
    ```

Metody specjalne - destruktor

```
public:
    ~DegreeAngle();
```

- każda klasa może mieć tylko jeden destruktor
- destruktor wywoływany jest automatycznie gdy niszczony jest obiekt
- służy do wykonania niezbędnych czynności przed likwidacją obiektu
- w przypadku tej metody nie podaje się typu zwracanego wyniku
- umieszcza się ją w części publicznej definicji klasy
- destruktor można wywołać w dowolnym momencie