Programowanie obiektowe C++

Definicja

Programowanie obiektowe (zorientowane obiektowo) to paradygmat programowania definiujący programy za pomocą struktur łączących stan i zachowanie pewnych bytów.

Stan – reprezentowany przez zmienne.

Zachowanie – reprezentowane przez metody.

Struktura łącząca stan z zachowaniem to klasa.

Klasa to wzorzec, na podstawie którego tworzone są unikalne byty (obiekty).

Podstawowe cechy języków obiektowych

- Abstrakcja
- Hermetyzacja (enkapsulacja)
- Dziedziczenie
- Polimorfizm

Podstawowe pojęcia – klasa i obiekt

- Klasa jest typem danych definiowanym przez użytkownika
- Reprezentuje rzeczywiste przedmioty, pojęcia lub zjawiska
- Klasa stanowi połączenie danych i metod służących do ich przetwarzania
- Klasa jest wzorcem, na podstawie którego powoływane są do życia konkretne byty nazywane obiektami

```
class NazwaKlasy
{
  //ciało klasy
};
```

Podstawowe pojęcia – klasa i obiekt

- Obiekt to konkretna instancja (reprezentant) klasy
- Obiekt reprezentuje pojedynczy, konkretny przedmiot, pojęcie lub zjawisko
- Wartości atrybutów obiektu to jego stan
- Obiekt zachowuje się zgodnie z zestawem zachowań zdefiniowanych w klasie

Definicja klasy

Definicja klasy zawiera:

- deklaracje pól
- deklaracje metod
- definicje metod mogą znajdować się w klasie lub poza nią

```
class NazwaKlasy
{
  //pola klasy
  //metody klasy
};
```

Definicja metod składowych

- Ciała metod składowych można umieszczać w klasie lub poza klasą
- Metody zdefiniowane w klasie mają domyślnie atrybut *inline*, czyli przy kompilacji ich wywołanie jest zastępowane ciałem metody (pod warunkiem, że ciało metody spełnia pewne warunki np. nie zawiera pętli)
- Metodom składowym definiowanym poza klasą też można nadać atrybut *inline*

Definicja klasy

```
class Punkt
{
   public:
   int x;
   int y;
   void przesunWPrawo()
   {
      x+=1;
      y+=1;
   }
   void przesunWLewo()
   {
      x-=1;
      y-=1;
   }

};
```

```
class Punkt
{
    public:
    int x;
    int y;
    void przesunWPrawo();
    void przesunWLewo();

};

void Punkt::przesunWPrawo()
{
    x+=1;
    y+=1;
}
void Punkt::przesunWLewo()
{
    x-=1;
    y-=1;
}
```

Składowe statyczne

- Składowe statyczne (*static*) są wspólne dla wszystkich obiektów danej klasy i istnieją nawet wówczas gdy nie istnieje żaden obiekt klasy.
- Statyczne metody składowe mogą odwoływać się tylko do składowych statycznych

```
class Stat
{
   static int licznik;
   public:
   static int podajIle()
   {
      return licznik;
   }
};
```

```
int Stat::licznik=0;

int main()
{
   cout<<Stat::podajIle();
   return 0;
}
```

Tworzenie obiektów

```
int main()
{
   Punkt P;                  //deklaracja obiektu
   P.x=10;                   //odwołanie do pola
   P.y=20;
   P.przesunWPrawo();        //wywołanie metody składowej

   Punkt *wsk = &P;          //deklaracja wskaźnika do obiektu
   wsk->x=10;                //odwołanie do pola za pomocą
   wsk->y=20;                //wskaźnika
   wsk->przesunWPrawo();

   return 0;
}
```

Dostęp do składników klasy

Modyfikatory dostępu:

- **private:** sekcja prywatna (domyślna dla klas) – składowe klasy są widoczne tylko w obrębie danej klasy oraz w klasach zaprzyjaźnionych
- **protected:** sekcja chroniona – składowe są widoczne w obrębie klasy oraz klas dziedziczących
- **public:** sekcja publiczna (domyślna dla struktur) – składowe są widoczne w całym programie

[specyfikator dostępu :] lista składowych klasy

Dostęp do składników klasy

```
class Punkt
{
  private:
  int x;
  int y;
  public:
  void przesunWPrawo();
  void przesunWLewo();

};
```

```
int main()
{
   Punkt P;
   P.x=10;      //błąd
   P.y=20;      //błąd
   P.przesunWPrawo(); //ok

   return 0;
}
```

Zmienna `this`

- W każdym obiekcie dostępna jest zmienna *this* wskazująca na ten obiekt.
- Za jej pomocą można zwracać w funkcjach składowych bieżący obiekt (**this*).
- Wszystkie odwołania do składowych z wnętrza obiektu są domyślnie poprzedzane *this->.*

```
class Punkt
{
   private:
   int x;
   int y;
   public:
   void ustawX(int x) {this->x = x;};
   void ustawY(int y) {this->y = y;};
};
```

Konstruktor

- Konstruktor jest specjalną metodą, wywoływaną zawsze gdy tworzony jest obiekt klasy
- Konstruktor nie jest wywoływany jawnie
- Nazwa konstruktora musi być taka sama jak nazwa klasy
- Konstruktor nie może zwracać żadnej wartości
- Zwykle konstruktory są publiczne

Konstruktor

```
class Punkt
{
   private:
   int x;
   int y;
   public:

   //konstruktor klasy punkt
   Punkt(int x, int y)
   {
       this->x = x;
       this->y = y;
   }
};
```

Deklaracja obiektu:

```
Punkt P(10,20);
```

Przeciążenie konstruktora

```
class Punkt
{
   int x, y;
   public:
   Punkt()
   {
      this->x = 0;  this->y = 0;
   }
   Punkt(int x, int y)
   {
      this->x = x;  this->y = y;
   }
};
```

Deklaracja obiektu:

```
Punkt P1;  //wywołanie konstruktora bezparametrowego
Punkt P2(10,20); //wywołanie konstr. z parametrami
```

Lista inicjalizacyjna konstruktora

- Wygodny sposób nadawania początkowych wartości składowym w konstruktorach
- Jedyny sposób inicjalizacji składowych stałych lub będących obiektami
- Jedyny sposób wywołania konstruktora klasy bazowej w konstruktorze klasy pochodnej

```
class Punkt
{
   int x, y;

   public:
   Punkt(int a, int b): x(a),y(b) {};
};
```

lista inicjalizacyjna

Konstruktor obiektu zawierajacego inny obiekt

- Obiekty składowe tworzone są przez ich konstruktory zanim zacznie działać konstruktor obiektu otaczającego
- Muszą korzystać z listy inicjalizacyjnej, w przeciwnym wypadku dla obiektów składowych wywołane zostaną ich domyślne bezargumentowe konstruktory (jeśli istnieją)
- Destruktory wywoływane są w odwrotnej kolejności

Konstruktor obiektu zawierajacego inny obiekt

```
class Punkt
{
   int x, y;
   public:
   Punkt(int x, int y){ this->x = x;  this->y = y;}
};

class Okrag
{
   Punkt srodek;
   int promien;
   public:
   Okrag (int x, int y, int p): srodek(x,y) { promien = p;};
};
```

Konstruktor kopiujący

- Uruchamia się w momencie tworzenia nowego obiektu na podstawie już istniejącego
- Jest generowany automatycznie, o ile nie zdefiniujemy własnego
- Kiedy należy zdefiniować konstruktor kopiujący?

```
class Student
{
   char *nazwisko;
   public:
   Student (char *nz) {nazwisko = new char[strlen(nz)+1];
                       strcpy(nazwisko,nz);}
   Student (const Student &s) {nazwisko = new char[strlen(s.nazwisko)+1];
                               strcpy(nazwisko,s.nazwisko);}
   ……
};
```

```
Student s1(„Kowalski”);
Student s2(s1);       //konstruktor kopiujący
```

Destruktor

- Wywoływany niejawnie, gdy obiekt jest niszczony
- Domyślnie tworzony jest destruktor, który nic nie robi
- Należy definiować destruktor, gdy konieczne jest dokonanie porządków po obiekcie np. zwolnienie przydzielonej dynamicznie pamięci
- Nazwa destruktora to nazwa klasy poprzedzona tyldą
- Destruktor nie ma parametrów i nie zwraca żadnej wartości
- Zwykle jest publiczny

Destruktor

```
class Student
{
    char *nazwisko;
    public:
    Student (char *nz)
    {
       nazwisko = new char[strlen(nz)+1];
       strcpy(nazwisko,nz);
    }
    Student (const Student &s)
    {
       nazwisko = new char[strlen(s.nazwisko)+1];
       strcpy(nazwisko,s.nazwisko);
    }
    ~Student()
    {
       delete [] nazwisko;
    }
};
```