06. Konstruktory

1 Konstruktor domniemany

Konstruktor domniemany to konstruktor, który można uruchomić bez argumentów. Jest on generowany automatycznie, jeżeli klasa nie posiada zdefiniowanego żadnego konstruktora.

Zadanie 01 Przeanalizuj kod zawarty w pliku źródłowym *pob06-figures.cpp*.

Zakomentuj na chwilę konstruktor bezargumentowy klasy `Figure` (z logicznego punktu widzenia nie jest on potrzebny, gdyż nie będziemy chcieli tworzyć „pustych" figur). Czy możesz teraz utworzyć obiekt klasy `Square`? Dlaczego tak się dzieje?

2 Listy inicjalizacyjne

Lista inicjalizacyjna zawiera listę *inicjalizatorów*, które są używane do nadawania początkowych wartości wybranym składowym podczas tworzenia obiektu przy pomocy konkretnego konstruktora. Inicjalizatory umożliwiają:

- inicjalizację zwykłych atrybutów z wykorzystaniem ich konstruktora,
- inicjalizację atrybutów stałych (`const`),
- inicjalizację atrybutów będących referencjami do innych obiektów,
- wywołanie konstruktora nadklasy.

```
1 class Circle : public Figure {
2   public:
3     Circle(float x, float y, float r)
4     : radius(r),      // nadanie wartosci atrybutowi
5       Figure(x, y)    // wywolanie konstruktora nadklasy
6     { }
7     // etc.
8 }
```

Zadanie 02a Sensowne wydaje się, aby raz zainicjowany środek figury nie mógł być zmieniany w klasie pochodnej bezpośrednio (może to prowadzić do problemów, jeżeli programista zapomni wraz ze zmianą środka zmienić współrzędne wierzchołków).

Czy możesz zmienić atrybut `position` w klasie `Figure` tak, aby był atrybutem prywatnym? Dlaczego tak się dzieje?

Zadanie 02b Przenieś inicjalizację atrybutów w konstruktorach klas `Figure` oraz `Square` do listy inicjalizacyjnej, zostawiając tam gdzie to możliwe puste ciała konstruktorów.

Zadanie 03 Zmodyfikuj konstruktor klasy `Square` tak, aby wykorzystywał trójargumentowy konstruktor klasy bazowej do inicjalizacji dziedziczonych atrybutów.

Czy teraz możesz uczynić atrybut `position` w klasie `Figure` atrybutem prywatnym?

Zadanie 04* Zauważ, że konstruktory w klasie `Figure` powielają inicjalizacje pewnych atrybutów. Wydaje się sensowne, aby konstruktor trójargumentowy wywoływał ten bezargumentowy.

Czy jest to możliwe? Przy jakich warunkach? Jak można zmniejszyć powielanie kodu między konstruktorami?

3 Konstruktor kopiujący

Konstruktor kopiujący to specjalny konstruktor, tworzący obiekt na podstawie podanego mu wzorca (w postaci referencji obiektu tego samego typu).

```
1 Circle c1(1.0, 2.0, 3.0);
2 Circle c2 = Circle(c2);   // jawne wywolanie konstrutora kopiujacego
```

Konstruktor kopiujący jest generowany automatycznie, jeżeli nie jest zdefiniowany. Przykładowa definicja:

```
1 class Circle {
2   public:
3     Circle(Circle& circle) { // lub Circle(const Circle& circle)
4       // ...
5     }
6   // etc.
```

Zadanie 05 Dodaj na chwilę „krzyczący” konstruktor kopiujący do klasy `Figure` w postaci `Figure(Figure &f)` oraz analogiczny konstruktor kopiujący do klasy `Square` w postaci `Square(Square &s)` (konstruktor „krzyczący” to taki, który wypisze pewien tekst, pozwalający na zidentyfikowanie jego wywołania).

Utwórz trzy obiekty klasy `Square`: (1) pierwszy poprzez wywołanie konstruktora trójargumentowego, (2) drugi poprzez przypisanie utworzonego obiektu do zmiennej typu `Square`, (3) trzeci poprzez jawne wywołanie konstruktora kopiującego.

Ile razy wywołano powyższe konstruktory podczas tworzenia obiektów?

Zadanie 06 Napisz funkcję `void pretty_print(Square square)`, która wypisze informacje o figurze korzystając z jej metody `print()`. Wywołaj ją z wcześniej utworzonym obiektem typu `Square`.

Czy zmieniła się liczba wywołań konstruktora kopiującego? Co należałoby zrobić, aby temu zapobiec? W jakich sytuacjach konstruktor kopiujący jest wywoływany niejawnie?

Usuń krzyczące konstruktory kopiujące.

Zadanie 07a Uzupełnij metodę `print()` w klasie `Square` o wypisywanie wierzchołków figury.

Zadanie 07b Do klasy `Square` dodaj metodę `scale(float scale_value)`, która przeskaluje figurę poprzez odpowiednią zmianę współrzędnych wierzchołków.

(Uwaga: aby nie musieć przesuwać każdego wierzchołka, łatwiej będzie wydzielić z konstruktora trójargumentowego kod odpowiedzialny za tworzenie wierzchołków obiektu do osobnej metody prywatnej, a następnie (1) zmienić długość boku `a *= scale_value` i (2) wywołać wydzieloną metodę.)

Zadanie 08 Odkomentuj kod skalujący obiekty klasy `Square`. Czy efekty są takie jak byśmy przewidywali? Dlaczego figury zostały źle przeskalowane?

Aby naprawić problem, zdefiniuj w klasie `Figure` odpowiedni konstruktor kopiujący `Figure(Figure& figure)`, który skopiuje wierzchołki, zamiast wskaźników.

4 Czas życia obiektów

Obiekty automatycznie mają zakres bloku, tzn. istnieją od momentu deklaracji do końca bloku, w którym zostały powołane do życia. Obiekty globalne mają zakres pliku, natomiast obiekty dynamiczne istnieją aż do wywołania operatora `delete`. Zatem każdy obiekt dynamiczny należy zniszczyć:

```
1 Circle* circle = new Circle();
2 //...
3 delete circle;
```

Zadanie 09a Dopisz „krzyczący" destruktor do klasy `Point`, a w destruktorze klasy `Figure` dodaj zwalnianie dynamicznie alokowanej pamięci poprzez operator `delete vertices`. Ile razy jest wywoływany destruktor klasy `Point` przy niszczeniu obiektu klasy `Square`? Napraw działanie destruktora korzystając z `delete[]`.

Zadanie 09b Jak długo żyją obiekty dla których dynamicznie zaalokowano pamięć, a jak długo obiekty zainicjalizowane standardowo? Odpowiedz na pytanie tworząc obiekty wewnątrz bloku (fragment kodu zawarty między nawiasami klamrowymi) i obserwując wywołania destruktora.

5 Słowo kluczowe `const`

Słowo kluczowe `const` informuje kompilator, że wartość zmiennej opatrzonej tym modyfikatorem, nie powinna być zmieniana w trakcie działania programu. Często stosuje się je podczas przekazywania referencji do obiektu jako argumentu funkcji/metody/konstruktora kopiującego, aby zabezpieczyć się przed jego nieporządaną zmianą.

Zadanie 10 Zmodyfikuj konstruktor kopiujący klasy `Figure`, tak aby jako argument przyjmował referencję `const`. Dlaczego jest to uzasadnione?

Zadanie 11 Zamień `int verticesAmount` na `const int verticesAmount`. Czy można teraz manipulować wartością `verticesAmount` np. w ciele konstruktora? Gdzie można zainicjować wartość atrybutu typu `const`?

Zadanie domowe 12 (1 pkt) Dodaj nową klasę `Rectangle` implementującą wymagane metody (`area`, `print`, `scale`) i posiadającą czteroargumentowy konstruktor `Rectange(float x, float y, float a, float b)`, który będzie wykorzystywał listy inicjalizacyjne.

Poprawnie zaimplementowana metoda `print()` we wszystkich klasach potomnych powiela ten sam kod. Zmień ją w taki sposób, aby nie było konieczności jej implementowania dla każdej nowej podklasy, a wciąż wypisywała informacje jak do tej pory:

- W klasie `Figure` dodaj definicję metody `print()` (podobnie jak wcześniej w klasach pochodnych).
- W klasie `Figure` brakować będzie informacji o typie figury (np. kwadrat, prostokąt), dlatego dodaj abstrakcyjną metodę `get_type()`.
- W klasach potomnych zaimplementuj tę metodę, tak aby w klasie `Square` zwracała napis *„Kwadrat”* itd.
- Zaktualizuj metodę `print()` w klasie `Figure`.

Do klasy `Point` dopisz metodę `to_string()`, która będzie działać podobnie do metody `print()`, z tą różnicą, że zamiast pisać na standardowe wyjście, zwróci łańcuch znaków. Wykorzystaj ją w metodzie `print()` klasy `Figure`.

Dodaj metodę `move(float x, float y)`, która przesunie środek figury oraz jej wierzchołki o wektor $[x, y]$. Zdecyduj gdzie najlepiej dodać tę metodę.