Konstruktory i destruktory

Konstruktor

Przypomnijmy, że tworzenie obiektów odbywa się przy pomocy specjalnych funkcji składowych zwanych **konstruktorami**, które nazywają się tak samo jak klasa.

- Konstruktor nie rezerwuje pamięci dla obiektu - to robi sam kompilator. Zadaniem konstruktora jest wypełnienie tej pamięci początkową treścią
- Konstruktor może być i często jest przeładowywany. Możemy więc mieć wiele różnych konstruktorów, różniących się argumentami.
- Konstruktor nic nie zwraca, nawet typu `void`.
- Instrukcja `return` może pojawić się w jego ciele, ale zaraz po niej musi być średnik.
- Konstruktor może być wołany jawnie, ale często jest wołany niejawnie

Przypomnijmy, że jeśli klasa nie definiuje żadnego konstruktora, kompilator sam tworzy konstruktor postaci

```
nazwa_klasy()
```

Jest to tak zwany **konstruktor domniemany**.

Jeśli w klasie zdefiniowano choć jeden, dowolny konstruktor, kompilator nie utworzy sam konstruktora domniemanego.

Destruktor

Destruktor to funkcja składowa, której nazwa to nazwa klasy poprzedzona znakiem wężyka.

```
Konstruktor pełni rolę odwrotną do konstruktora.
samochod::~samochod(){
    ....
}
```

- Destruktor nie zwraca żadnego typu, nawet typu `void`.
- Klasa nie musi mieć obowiązkowo destruktora.
- Destruktor nie likwiduje obiektu, ani nie zwalnia obszaru pamięci, który obiekt zajmował - to robi kompilator
- Destruktor jest wywoływany bez argumentów, nie może więc być przeładowywany
- Destruktor przydaje się wtedy, gdy przed zniszczeniem obiektu trzeba jeszcze dokonać jakichś działań, na przykład posprzątać.

Przykładowe zastosowania destruktora

- Jeśli na przykład obiekt reprezentował okienko na ekranie, to możemy chcieć, by w momencie likwidacji tego obiektu okienko zostało zamknięte, a ekran wyglądał jak dawniej.
- Destruktor jest potrzebny, gdy konstruktor danej klasy dokonał na swój użytek rezerwacji dodatkowej pamięci (operatorem **new**). Wtedy w destruktorze umieszcza się instrukcję **delete** zwalniającą ten obszar pamięci.
- Destruktor może się też przydać, gdy liczymy obiekty danej klasy. W konstruktorze podwyższamy taki licznik o jeden, a w destruktorze zmniejszamy o jeden.
- Destruktor może się przydać po to, by obiekt mógł spisać na dysku (lub ekranie) swój testament.

Konstruowanie obiektów lokalnych automatycznych

Obiekt lokalny automatyczny to obiekt zdefiniowany wewnątrz bloku.

```
....
  {     // otwarcie bloku
    ...
    samochod kr30780;  // obiekt powołany do życia
    ...
  }     // zamknięcie bloku, obiekt przestaje istnieć
...
```

- Konstruktor takiego obiektu jest uruchamiany w momencie, gdy program napotyka jego definicję.
- Destruktor jest uruchamiany, gdy program opuszcza blok

Konstruowanie obiektów lokalnych statycznych i obiektów globalnych

Obiekt lokalny statyczny to obiekt zdefiniowany wewnątrz bloku, ale dodatkowo poprzedzony słowem **static**.

```
....
  {     // otwarcie bloku
    ...
    static samochod kr30780;
    ...
  }     // zamknięcie bloku,
...
```

Obiekt taki, choć niewidoczny poza blokiem, nie jest likwidowany przy wychodzeniu z bloku i jego wartość jest dostępna przy powtórnym wejściu do bloku.

Obiekt globalny to obiekt zdefiniowany poza wszystkimi funkcjami. Zakres ważności takiego obiektu to cały plik. Jest on dostępny we wszystkich funkcjach zdefiniowanych w tym pliku.

- Konstruktor obiektu globalnego jest uruchamiany przed rozpoczęciem bloku **main**.
- Destruktor obiektu globalnego jest uruchamiany po zakończeniu bloku **main**.
- Konstruktor obiektu lokalnego statycznego jest uruchamiany w momencie pierwszego napotkania deklaracji stosownej zmiennej lokalnej statycznej. Jeśli blok, w którym umieszczono deklarację takiego obiektu nie jest wykonywany, jego konstruktor nie jest wołany
- Destruktor obiektu lokalnego statycznego jest uruchamiany po zakończeniu bloku **main**, ale tylko wtedy gdy wcześniej był wołany konstruktor tego obiektu.

Konstruowanie obiektów tworzonych operatorem new

Obiekt może zostać powołany do życia poprzez użycie operatora **new**.

```
....
  samochod *s=new samochod("Wartburg",127000,12.);
...
```

Obiekt taki istnieje do czasu gdy nie zostanie zlikwidowany instrukcją **delete**

```
....
  delete s;
...
```

- Konstruktor obiektu utworzonego operatorem **new** na stercie jest uruchamiany natychmiast po udanym zarezerwowaniu pamięci dla tego obiektu na stercie.
- W przypadku, gdy pamięci nie uda się przydzielić, konstruktor nie jest uruchamiany i rzucany jest wyjątek systemowy **bad_alloc**.
- Destruktor jest uruchamiany bezpośrednio przed wykonaniem instrukcji **delete**, a więc bezpośrednio przed zwolnieniem pamięci, jaką ten obiekt zajmował na stercie.

Konstruowanie tablic obiektów operatorem new[]

Na stercie można też utowrzyć tablicę obiektów.

```
....
  samochod *s=new samochod[20];
...
```

Tablica taka istnieje do czasu gdy nie zostanie zlikwidowana instrukcją **delete**[]

```
....
  delete[] s;
...
```

- W przypadku operatora **new**[] pamięć rezerwowana jest dla tylu obiektów ile wskazano w liczbie umieszczonej pomiędzy nawiasami klamrowymi.
- Bezpośrednio po przydzieleniu pamięci wywoływany jest konstruktor domyślny na rzecz każdego elementu tworzonej tablicy
- Bezpośredno przed zwolnieniem pamięci dla tablicy przy pomocy operatora **delete**[] uruchomiane są destruktory na rzecz wszystkich obiektów tej tablicy.
- Użycie operatora **delete** w odniesieniu do wskaźnika pokazującego na umieszczoną na stercie tablicę zwolni całą zarezerwowaną pamięć, ale destruktor zostanie wywołany tylko na rzecz pierwszego elementu tablicy.

Jawne wywołanie konstruktora

Obiekt może zostać stworzony poprzez jawne wywołanie konstruktora.

```
....
  samochod("Wartburg",127000,12.);
...
```

W efekcie dostajemy obiekt, który nie ma nazwy, a czas jego życia ogranicza się do wyrażenia, w którym go użyto.

Konstruktor nie jest wywoływany na rzecz jakiegoś obiektu, bo dopiero sam go tworzy. Dlatego w wywołaniu konstruktora nie stosuje się notacji kropkowej.

Konstruktory kopiujące

Konstruktor kopiujący to konstruktor, który ma nagłówek

```
nazwa_klasy(nazwa_klasy &jakis_obiekt_klasy);
```

Na przykład

```
szyld(szyld &oryginalny_szyld);
```

```
Zadaniem konstruktora kopiującego jest stworzenie wiernej kopii oryginału
wysłanego jako wzór. Oczywiście to my tworząc taki konstruktor musimy zadbać,
by rzeczywiście tworzył wierną kopię.
```

Kopiowanie obiektów

Obok konstruktora domniemanego jest jeszcze drugi typ konstruktora, który zostanie wygenerowany automatycznie przez kompilator, jeśli my nie dostarczymy własnego.

Jeśli nie zdefiniujemy konstruktora kopiującego, kompilator zrobi go za nas, metodą "składnik po składniku". Zrobi to nawet wtedy, gdy w klasie zdefiniowane będą inne konstruktory, ale brak będzie konstruktora kopiującego.