Konstruktory

Streszczenie

Celem wykładu jest zaprezentowanie konstruktorów w Javie, syntaktyki oraz zalet ich stosowania.

Czas wykładu – 45 minut.

Rozpatrzmy przykład przedstawiający klasę `Prostokat`:

```
class Prostokat
{
    int wysokosc, szerokosc;
    void wypisz()
    {
        System.out.println(wysokosc + " " + szerokosc);
    }
}
```

Wykorzystajmy ją do wygenerowania prostokąta o bokach długości 7 i 3 oraz wypiszmy dane o tym prostokącie.

```
class MAIN
{
    public static void main(String args[])
    {
        Prostokat p = new Prostokat();
        p.wypisz();
    }
}
```

Oczywistym błędem jest brak inicjalizacji pól `wysokosc` oraz `szerokosc`, czego efektem jest wydruk `0 0`. Można łatwo poprawić program dopisując dwie instrukcje przypisania

```
class MAIN
{
    public static void main(String args[])
    {
        Prostokat p = new Prostokat();
        p.wysokosc = 7;
        p.szerokosc = 3;
        p.wypisz();
    }
}
```

Rozwiązanie to jednak nie uchroni nas przed ponownym zapomnieniem nadania wartości odpowiednim polom obiektu.

Bardziej uniwersalnym rozwiązaniem jest wykorzystanie mechanizmów programowania obiektowego—zastosowanie specjalnych metod nazywanych *konstruktorami*. Konstruktory wykonywane są w momencie generowania obiektów i dlatego idealnie nadają się do inicjalizowania wartości pól.

Jak definiuje się konstruktory?

Konstruktory są funkcjami o nazwach zgodnych z nazwami klas, w których dane konstruktory się definiuje. (Ta informacja dotyczy Javy i np. C++. W języku Pascal, konstruktory mogą mieć dowolne inne nazwy). Konstruktory nie zwracają żadnych wartości. Nie deklaruje się nawet typu `void`.

Powróćmy do przykładu z prostokątem. Stosując powyższe wytyczne, konstruktor tej klasy może wyglądać następująco:

```
Prostokat(int wys, int szer)
{
    wysokosc = wys;
    szerokosc = szer;
}
```

Zobaczmy jak zareaguje kompilator na dodanie takiego konstruktora do klasy:

```
javac Prostokat.java

Prostokat.java:21: cannot find symbol
symbol  : constructor Prostokat()
location: class Prostokat
Prostokat p = new Prostokat();
              ^
1 error
```

Okazuje się, że nie można utworzyć obiektu. Problem, który za chwilę okaże się zaletą, polega na tym, że tworząc obiekt uruchamiany jest konstruktor. Który? Ten przed chwilą dodany. Ponieważ posiada on dwa argumenty musimy przekazać mu dwie wartości—parametry tworzonego prostokąta. Ale o to właśnie nam chodziło, aby nie zapomnieć o nadaniu wartości polom klasy. Wykorzystanie konstruktora wygląda zatem następująco:

```
Prostokat p = new Prostokat(7,3);
```

Konstruktor domyślny

Powiedzieliśmy, że konstruktor jest funkcją wołaną automatycznie w momencie tworzenia obiektu. Co jednak się dzieje, gdy w klasie nie zdefiniowano żadnego konstruktora? Podobnie jak w C++, w takiej sytuacji kompilator generuje i dołącza do klasy bezargumentowy *konstruktor domyślny* umożliwiając generowanie obiektów. Jednak, w sytuacji gdy programista zdefiniuje własny konstruktor, ten domyślny nie jest generowany. Stąd pojawiły się problemy z kompilacją w przykładzie powyżej.

Zwróćmy jeszcze uwagę na syntaktykę tworzenia obiektu z wykorzystaniem konstruktora bezargumentowego:

```
Prostokat p = new Prostokat();
```

Operator `new` wymaga nawiasy `()`, inaczej niż w C++. Podkreśla to fakt, wołania konstruktora w momencie tworzenia obiektu.

Przeciążanie konstruktorów

Wykorzystajmy klasę `Prostokat` do utworzenia kwadratu o boku 4:

```
Prostokat k = new Prostokat(4,4);
```

Efekt uzyskujemy poprzez powtórzenie wielkości `4`. Jednak, gdyby klasa nie reprezentowała prostokąta na płaszczyźnie, ale wielokąt w wielowymiarowej przestrzeni, otrzymanie regularnej kostki wymagałoby wielokrotne powtórzenie tej samej wielkości, co może okazać się uciążliwe. Lepszym rozwiązaniem jest zdefiniowanie drugiego konstruktora, konstruktora jednoargumentowego, służącego do tworzenie regularnych kostek:

```
Prostokat(int a)
{
    wysokosc = szerokosc = a;
}
```

Przy przeciążaniu konstruktorów obowiązują ogólne zasady związane z przeciążaniem funkcji, tj. listy argumentów przeciążanych funkcji muszą być różne: albo mieć różne ilości argumentów, albo różnić się typem przynajmniej jednego argumentu.

Wzajemne wywoływanie się konstruktorów

Zauważmy, że oba powyższe konstruktory wykonują te same czynności—nadają odpowiednie wartości polom `wysokosc` i `szerokosc`. W ogólnym przypadku konstruktory mogą wykonywać inne, bardziej złożone zadania (np. w omawianym przykładzie obliczanie pola prostokąta). Jeśli założymy, że konstruktor danej klasy realizuje jakieś dodatkowe zadanie, to każdy konstruktor danej klasy powinien to zadanie realizować. Chcemy bowiem, aby funkcjonalność wszystkich konstruktorów danej klasy była dokładnie taka sama. Z drugiej strony nie chcemy implementować tego samego algorytmu wielokrotnie, w każdym konstruktorze osobno.

Rozwiązaniem jest implementacja żądanego algorytmu w najbardziej ogólnym konstruktorze, tym konstruktorze, który ma dostęp do największej ilości argumentów (dwuargumentowy w naszym przykładzie), a następnie uruchomienie tego algorytmu w innym konstruktorze. Java umożliwia bowiem wzajemne wywoływanie się konstruktorów. (W C++ podobne rozwiązanie wymagało zdefiniowania innej funkcji nie będącej konstruktorem i wywołania jej we wszystkich konstruktorach danej klasy.)

W jaki sposób wywołać konstruktor w innym konstruktorze tej samej klasy? W tym celu stosuje się konstrukcję `this()` z parametrami odpowiadającymi wywoływanemu konstruktorowi. Działa tu mechanizm przeciążania funkcji, z tą różnicą, że funkcję nie wywołuje się poprzez jej nazwę, ale poprzez słowo `this`. W naszym przykładzie podstawowym konstruktorem budowania prostokątów jest:

```
Prostokat(int wys, int szer)
{
    wysokosc = wys;
    szerokosc = szer;
}
```

Do generowania kwadratów, możemy go użyć jako:

```
Prostokat(int a)
{
    this(a,a);
}
```

Pewnym ograniczeniem wzajemnych wywołań konstruktorów jest fakt, że `this()` musi być pierwszym wywołaniem w ciele konstruktora. Wszystkie inne czynności muszą być realizowane później. Poprawnym jest zatem

```
Prostokat(int a)
{
    this(a,a);
    System.out.println("Konstruktor jednoargumentowy");
}
```

Należy pamiętać, aby nie doszło do zapętlenia się wywołań konstruktorów. Jest to kontrolowane przez kompilator Javy. Niedopuszczalne jest zatem:

```
Prostokat(int wys, int szer)
{
    this(wys);
}

Prostokat(int a)
{
    this(a,a);
}
```

Kompilator zgłasza błąd:

```
Prostokat.java:5: recursive constructor invocation
    Prostokat(int wys, int szer)
    ^
1 error
```

Kompletny przykład

```
class Prostokat
{
    int wysokosc, szerokosc;

    Prostokat(int wys, int szer)
    {
        wysokosc = wys;
        szerokosc = szer;
    }

    Prostokat(int a)
    {
        this(a,a);
    }

    void wypisz()
    {
        System.out.println(wysokosc + " " + szerokosc);
    }
}

class MAIN
{
    public static void main(String args[])
    {
        Prostokat p = new Prostokat(7,3);
        p.wypisz();
        Prostokat k = new Prostokat(4);
        k.wypisz();
    }
}
```

Podsumowanie

- Konstruktory są funkcjami o nazwach zgodnych z nazwą klasy.
- Konstruktory nie zwracają żadnej wartości.
- Konstruktory można przeciążać.
- Konstruktory są wywoływane w momencie generowania obiektów.
- Konstruktory idealnie nadają się do inicjalizowania wartości pól, w tym przydziału pamięci.
- Konstruktor domyślny nie jest generowany, gdy w klasie zdefiniowano własny konstruktor.

- Wywołanie `this()` musi być pierwszym wywołaniem w konstruktorze.
- Należy pilnować, aby użycie `this()` nie powodowało zapętlenia wywołań konstruktorów.