

Klasy lokalne i anonimowe

Czego się dowiesz

- Czym są klasy lokalne,
- czym są klasy anonimowe.

Klasy lokalne

Dowiedziałeś się już nieco o klasach wewnętrznych oraz zagnieżdżonych, jednak okazuje się, że rodzajów klas wewnętrznych jest jeszcze więcej. Można je zagnieżdżyć nawet w metodach i w takich sytuacjach nazywać będziemy je klasami lokalnymi. W odróżnieniu od klas wewnętrznych mają one jednak dużo więcej ograniczeń. Są one raczej niespotykane, jednak jest to jedno z zagadnień certyfikujących, więc dla samej świadomości ich istnienia wypada o tym wspomnieć.

- klasy lokalne nie mogą mieć specyfikatorów dostępu (public, private protected), nie mogą być statyczne (static). Mogą być jednak abstrakcyjne (abstract) i finalne (final), jednak nie mogą łączyć tych dwóch słów kluczowych w deklaracji
- klasy lokalne mogą korzystać z pól klasy otaczającej, ale jeśli korzystają ze zmiennych lokalnych metody to muszą one być oznaczone jako final. Od Javy w wersji 8 możliwe jest także stosowanie zmiennych efektywnie finalnych - tzn. zmiennych, które są raz zainicjowane i nie zmieniają później swojej wartości, a w szczególności nie zmieniają swojej wartości wewnątrz klasy lokalnej.

```
class Outer {  
    private int x;  
  
    public void doSomething() {  
        int localVar = 5;  
  
        class Inner {  
            public void innerMethod() {  
                x = 5;  
                System.out.println(localVar);  
            }  
        }  
    }  
}
```

Klasa *Inner* została stworzona wewnątrz metody *doSomething()* z klasy *Outer*. Możemy z metody w jej wnętrzu odwoływać się do pola *x* klasy opakowującej, możemy też wyświetlać wartość *localVar*, jednak tylko pod warunkiem, że nie zmienimy wartości tej zmiennej w dalszej części kodu. Zmienna taka nazywa się od Javy 8 efektywnie finalną - tzn. pomimo iż nie ma modyfikatora *final*, to kompilator jest w stanie stwierdzić, że faktycznie jest ona raz zainicjowana i później nie zmienia wartości.

Programując w Javie od kilkunastu lat nigdy nie spotkałem się z tym, żeby ktoś stosował klasy lokalne.

Klasy anonimowe

Klasy anonimowe są spotykane dużo częściej niż wspomniane powyżej klasy lokalne. Co ciekawe nie posiadają one swojej nazwy i pozornie pozwalają stworzyć instancję na bazie interfejsu lub klasy abstrakcyjnej. Pozornie ponieważ w rzeczywistości tworzona jest nowa klasa, w której metody takiego interfejsu musimy zaimplementować. Klasy anonimowe były w Javie szczególnie często wykorzystywane do tworzenia tzw. słuchaczy, czyli obiektów pozwalających przechwytywać informacje na temat interakcji użytkownika z aplikacją wykorzystującą technologię Swing. Innym zastosowaniem klas anonimowych są sytuacje, w których musimy zdefiniować klasę implementującą jakiś interfejs tylko w jednym miejscu kodu - można więc powiedzieć, że jest to "najbardziej lokalna" ze wszystkich klas wewnętrznych dostępnych w Javie.

Podobnie jak w przypadku klas lokalnych, klasy anonimowe również mogą odwoływać się do pól klasy opakowującej, w której ją definiujemy, a także do zmiennych lokalnych w metodach o ile zmienne te są finalne lub efektywnie finalne.

W Javie 8 wprowadzono pewne zmiany, które sprawiły, że klasy anonimowe da się zastąpić innymi konstrukcjami takimi jak wyrażenia lambda, o których powiemy nieco później, dlatego we współczesnym kodzie nie spotkasz ich już tak często jak kiedyś. Jeśli trafisz kiedyś na nieco starszy projekt, to warto mieć świadomość co to za dziwna konstrukcja.

Stwórzmy interfejs `BiggestNumber`, który posiada sygnaturę metody zwracającej największą z liczb w tablicy liczb typu `int`.

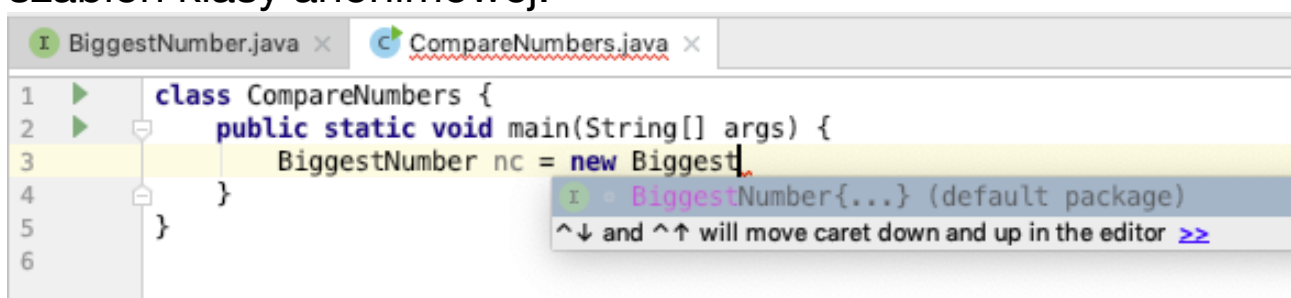
BiggestNumber.java

```
interface BiggestNumber {  
    int takeBiggest(int[] tab);  
}
```

Normalnie, aby móc z niego skorzystać musielibyśmy stworzyć klasę, która implementuje interfejs `BiggestNumber`, np.:

```
class BiggestNumberChooser implements BiggestNumber { ... }
```

Jeżeli zamierzamy skorzystać z implementacji tego interfejsu tylko w jednym miejscu naszego kodu, to wygodniej będzie wykorzystać do tego klasę anonimową. W celu jej utworzenia wpisz `BiggestNumber nazwaZmiennej = new Biggest` i wciśnij `Ctrl+Spacja` lub sam `Enter`, jeśli odpowiedź zdążyła się sama pojawić. Środowisko podpowie ci, że może wygenerować szablon klasy anonimowej.



Wybierz pierwszą opcję z listy. Przyjrzyjmy się wygenerowanemu kodowi anonimowej klasy:

```
class CompareNumbers {  
    public static void main(String[] args) {  
        BiggestNumber nc = new BiggestNumber() {  
            @Override  
            public int takeBiggest(int[] tab) {  
                return 0;  
            }  
        };  
    }  
}
```

Pozornie wygląda to tak, jakbyśmy utworzyli nowy obiekt typu `BiggestNumber`, ale przecież interfejsu nie można inicjalizować słowem `new`. W rzeczywistości pomiędzy nawiasami klamrowymi `{}` umieszczone jest ciało klasy, która nie posiada swojej nazwy (jest anonimowa). Ponieważ klasa ta implementuje interfejs `BiggestNumber`, to musi przesłaniać wszystkie metody abstrakcyjne z interfejsu, czyli te, które nie są oznaczone jako domyślne. Zwróć także uwagę na nietypowe zakończenie nawiasu klamrowego średnikiem, czego do tej pory jeszcze nie spotkaliśmy.

Metoda `takeBiggest()` ma zwrócić największą z liczb w tablicy, więc ją zaimplementujemy, a także zobaczymy jak wygląda wykorzystanie takiej klasy w użytkowym kodzie.

`CompareNumbers.java`

```
class CompareNumbers {
    public static void main(String[] args) {
        BiggestNumber nc = new BiggestNumber() {
            @Override
            public int takeBiggest(int[] tab) {
                if (tab == null)
                    throw new NullPointerException("argument cannot be null");
                if (tab.length == 0)
                    throw new IllegalArgumentException("array has to have at least 1 value");

                int biggestNumber = tab[0];
                for (int i = 1; i < tab.length; i++) {
                    if (biggestNumber < tab[i]) {
                        biggestNumber = tab[i];
                    }
                }
                return biggestNumber;
            }
        };

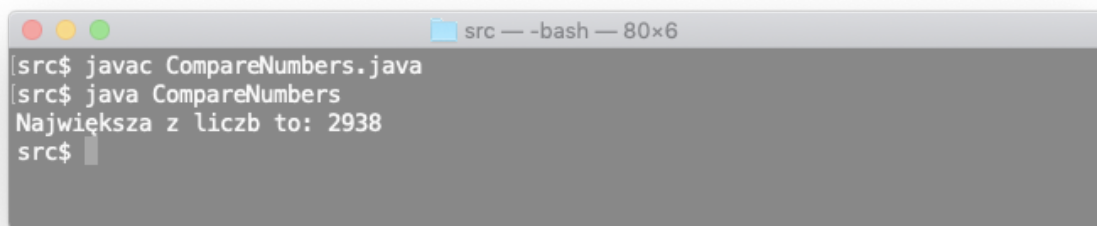
        int[] tab = {5, 10, -8, -23, 1009, 2938, 153, 24};
        System.out.println("Największa z liczb to: " + nc.takeBiggest(tab));
    }
}
```

W metodzie `takeBiggest()` sprawdzamy najpierw, czy przekazany parametr (tablica) nie jest wartością `null` oraz, czy zawiera chociaż 1 element. Jeśli nie, to rzucamy odpowiednio wyjątek `NullPointerException` lub `IllegalArgumentException`. Jeżeli tablica ma przynajmniej 1 element, to w dalszej części kodu do zmiennej `biggestNumber` przypisujemy pierwszy element tablicy, a następnie przechodzimy w pętli `for` przez jej wszystkie elementy. Jeżeli kolejny element jest większy od

wartości przypisanej do `biggestNumber`, to jest ona do tej zmiennej przypisywana.

Po skończonej iteracji zwracamy największą z liczb.

Dzięki temu, że zaimplementowaliśmy anonimową klasę wewnętrzną, w dalszej części możemy wywoływać jej metody, czyli np. `takeBiggest()`, która jak widać zwraca poprawną wartość:

A screenshot of a terminal window with a title bar that says "src — -bash — 80x6". The terminal shows the following commands and output:

```
[src$ javac CompareNumbers.java  
[src$ java CompareNumbers  
Największa z liczb to: 2938  
src$
```

Klasy anonimowe mogą być tworzone w taki sposób nie tylko na podstawie interfejsów, ale także na podstawie zwykłych i abstrakcyjnych klas.

Zapamiętaj

Klasy anonimowe nie posiadają swojej nazwy, mogą wykorzystywać zmienne lokalne metody, w której są zdefiniowane, ale zmienne te muszą być finalne lub efektywnie finalne (mogą być przypisane tylko raz), mogą korzystać ze składowych głównej klasy, w której się znajdują, nie mogą posiadać konstruktorów, mogą definiować dodatkowe pola i metody, których nie ma w interfejsie lub klasie na podstawie którego są tworzone.