

Składowe statyczne (static)

Czego się dowiesz

Co oznacza słowo static w Javie,
kiedy wykorzystywać elementy statyczne.

Wstęp

Już w lekcji, w której napisaliśmy pierwszy program pojawiło się słowo **static** - konkretnie przy metodzie main. Wtedy zostawiliśmy je na później, teraz czas wrócić do tego zagadnienia.

Do tej pory nie licząc metody main wykorzystywaliśmy składowe klas (pola, metody), do których co prawda mieliśmy dostęp, jednak dopiero po wcześniejszym utworzeniu obiektu za pomocą słowa *new*. Za pomocą słowa **static** możesz zdefiniować pola oraz metody do których możesz odwoływać się bez tworzenia nowego obiektu. Niesie to jednak ze sobą pewne ograniczenia - z metod oznaczonych jako static (np. main) nie można odwoływać się do niestatycznych elementów klasy. Jest to spowodowane tym, że skoro nie istnieje instancja danej klasy, to również składowe instancji jeszcze nie istnieją.

Początkowo słowo static powinieneś wykorzystywać głównie do definiowania **stałych**, czyli pól klasy oznaczonych słowami. Jeżeli jednak będziesz kontynuować swoją przygodę z programowaniem, to poznasz również wzorce projektowe, w których metody i pola statyczne bywają przydatne (wzorzec

projektowy to gotowe rozwiązanie powtarzalnego problemu, które jest ogólnie przyjęte jako dobra praktyka).

Zgodnie z konwencją nazewnictwa stałe klasy, czyli pola klasy oznaczone jako **final static** powinno nazywać się wyłącznie WIELKIMI literami, a jeśli nazwa składa się z kilku wyrazów, to kolejne słowa oddzielać znakiem podkreślenia, np:

```
class MyCalc {  
    public static final double PI = 3.14;  
}
```

albo:

```
class Calendar {  
    private final int NUMBER_OF_MONTHS = 12;  
    //dalszy kod  
}
```

Stałe mogą mieć dowolny specyfikator dostępu.

Przykład 1

Przykładem zastosowania pól statycznych mogłoby być zdefiniowanie prostej klasy, w której przechowywalibyśmy informacje dotyczące roku, tygodni oraz dni.

MyCalendar.java

```
class MyCalendar {  
    static final int WEEK_DAYS = 7;  
    static final int YEAR_DAYS = 365;  
}
```

Teraz nie musisz pamiętać już ile dni ma tydzień i rok, ale co ważniejsze możesz w ten sposób podnieść czytelność kodu. Zupełnie inaczej wygląda w skomplikowanym kodzie wykorzystanie zmiennej z nadaną nazwą, niż "magiczna liczba" 7, która osobie korzystającej z twojego kodu może niewiele mówić.

Tak jak napisano wcześniej, do wykorzystania składowych statycznych nie potrzebujesz instancji obiektu, możesz się do tych pól odwoływać bezpośrednio przez konstrukcję:

```
NazwaKlasy.nazwaElementuStatycznego;
```

czyli na przykład:

CalendarTest.java

```
class CalendarTest {  
    public static void main(String[] args) {  
        System.out.println("Tydzień ma " +  
MyCalendar.WEEK_DAYS + "dni");  
        System.out.println("Rok ma " + MyCalendar.YEAR_DAYS +  
"dni");  
    }  
}
```

Przykład 2

Pola statyczne oczywiście nie muszą być finalne. Ważne jest natomiast to, żeby zdawać sobie sprawę z tego, że jeżeli klasa posiada pole statyczne, to niezależnie od tego ile obiektów danej klasy utworzymy, to istnieje jeden i tylko jeden egzemplarz tego pola statycznego i jest on powiązany z klasą, a nie z żadną konkretną instancją.

Stwórzmy klasę *Human* (człowiek), w której przechowywać będziemy informacje takie jak imię i wzrost konkretnego człowieka, a także średni wzrost człowieka na świecie (dla uproszczenia zakładamy, że świat dopiero powstał i istnieją tylko Adam i Ewa).

Human.java

```
class Human {  
    private String name;  
    private double height;  
    public static double avgHeight;  
  
    public Human(String name, double height) {
```

```

        this.name = name;
        this.height = height;
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public double getHeight() {
        return height;
    }
    public void setHeight(double height) {
        this.height = height;
    }
}

```

Klasa *Human* przechowuje imię oraz wzrost osoby (pola prywatne). Dodatkowo zdefiniowane jest jednak statyczne pole typu `double` do przechowywania informacji o średnim wzroście człowieka. Do zmiennej takiej będziemy mieli dostęp bez potrzeby tworzenia konkretnej instancji obiektu, więc zadeklarowaliśmy je jako publiczne.

World.java

```

class World {
    public static void main(String[] args) {
        Human adam = new Human("Adam", 185.5);
        Human eve = new Human("Ewa", 167.5);

        double avgHeight = (adam.getHeight() +
eve.getHeight()) / 2;
        Human.avgHeight = avgHeight;

        System.out.println("Pierwsi ludzie na Ziemi: ");
        System.out.println(adam.getName() + " " +
adam.getHeight() + "cm");
        System.out.println(eve.getName() + " " +
eve.getHeight() + "cm");

        System.out.println("Średni wzrost: ");
    }
}

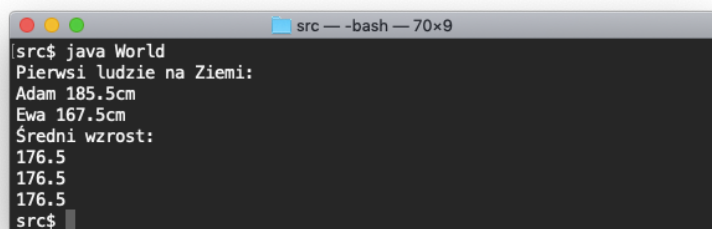
```

```

        System.out.println(adam.avgHeight);
        System.out.println(eve.avgHeight);
        System.out.println(Human.avgHeight);
    }
}

```

W klasie *World* (świat) utworzyliśmy dwóch ludzi - Adama i Ewę. Jak widać mają różny wzrost, więc poniżej wyliczyliśmy średni wzrost (*avgHeight*). Jak widzisz informację o średnim wzroście możemy uzyskać albo przez konkretne instancje (adam i eve) lub odwołując się do pola bezpośrednio przez klasę *Human*.



```

src$ java World
Pierwsi ludzie na Ziemi:
Adam 185.5cm
Ewa 167.5cm
Średni wzrost:
176.5
176.5
176.5
src$

```

W praktyce odwoływanie się do składowych statycznych poprzez instancje klasy jest złą praktyką, bo nie po to deklarujemy coś jako static, żeby mieć do tego elementu dostęp dopiero po utworzeniu obiekt. Możliwość zapisu typu *eve.avgHeight*, czyli odwoływanie się do składowych statycznych poprzez referencje jest dziś uznawane za jeden z błędów podjętych przez projektantów języka Java. Co więcej daje to możliwość do zapisania np. takiego kodu:

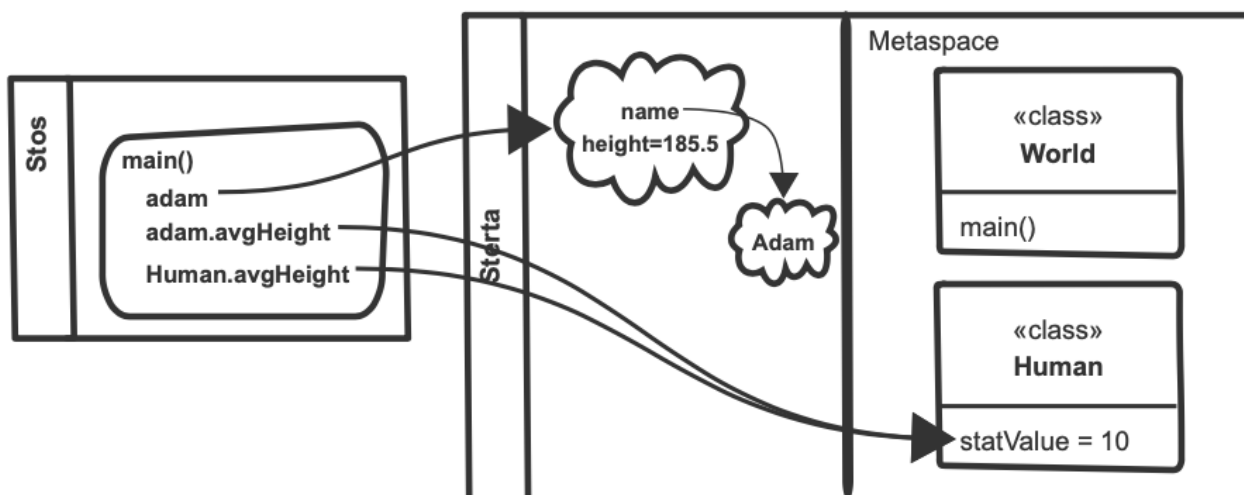
```

Human human = (Human)null;
System.out.println(human.avgHeight);

```

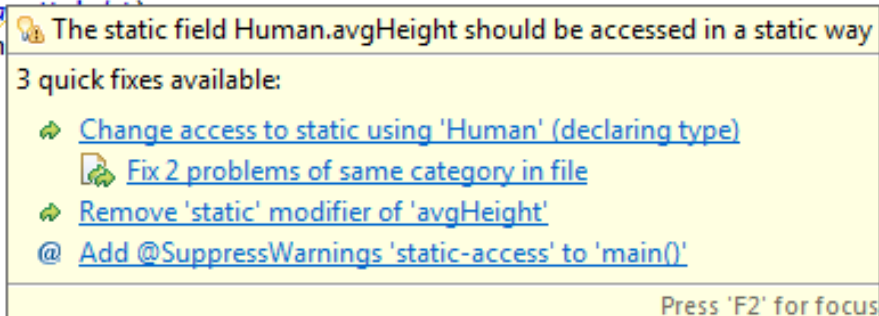
i jest to jak najbardziej "poprawne", choć można oczekiwać, że w takiej sytuacji zobaczymy błąd *NullPointerException*. Z powyższego kodu możemy

wywnioskować, że odwołując się do pola statycznego (a także metod statycznych) obiekt jest w rzeczywistości pomijany. W pamięci mamy więc w uproszczeniu coś takiego jak na poniższym diagramie.



Jeśli korzystasz ze środowiska takiego jak IntelliJ lub eclipse, to odwoływanie się do składowych statycznych poprzez obiekty będzie oznaczone jako ostrzeżenie. Zobaczysz komunikat, który można przetłumaczyć jako "hej, jeśli deklarujesz coś jako static, to odwołuj się do tego w sposób jak na static przystało, czyli bezpośrednio przez Human.avgHeight"

```
System.out.println("Średni wzrost: ");
System.out.println(adam.avgHeight);
System.out.println(eve.avgHeight);
System.out.println(Human.avgHeight);
```



Przykład 3

Jeżeli chcesz sobie ułatwić życie i na przykład przenieść powtarzalny kod z metody main do innej metody, to pamiętaj, że elementy do których będziesz się odwoływać z jej wnętrza, również muszą być oznaczone jako static. Poniżej krótki przykład programu, którego zadaniem jest wyświetlenie liczb z przedziału podanego przez użytkownika. Powtarzalną część kodu odpowiedzialną za wydruk na ekranie "wyrzuciliśmy" na zewnątrz.

```
import java.util.Scanner;

class PrintNumbers {

    private static final int EXIT = 0;

    private static void printNumbers(int start, int end) {
        for (int i = start; i <= end; i++) {
            System.out.print(i + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        int start, end;
        do {
            System.out.println("Pierwsza liczba: ");
            start = input.nextInt();
            System.out.println("Druga liczba: ");
            end = input.nextInt();
            printNumbers(start, end);

            System.out.println("Koniec programu, wprowadź 0");
            System.out.println("Kontynuuj, wprowadź 1");
        } while (input.nextInt() != EXIT);

        input.close();
    }
}
```

Oprócz metody `main`, która musi być statyczna stworzyliśmy również metodę `printNumbers()` odpowiedzialną za wydrukowanie liczb z przedziału podanego przez użytkownika. Ponieważ odwołujemy się do niej z wnętrza metody `main`, ją również musieliśmy oznaczyć jako *static*. Gdyby metoda `printNumbers()` nie była statyczna, to niezbędna byłaby taka konstrukcja:

```
PrintNumbers pn = new PrintNumbers();  
pn.printNumbers(start, end);
```

Dodatkowo mamy pole finalne `EXIT` (stałą), które pozwala nam ustalić wartość wyjścia z naszego programu. Do niej również odwołujemy się z metody statycznej, więc też musi być oznaczona jako *static*.

Zapamiętaj

Klasa może zawierać pola i metody statyczne.

Pola statyczne najczęściej będą połączone ze słowem **final** i będziemy je wtedy nazywali stałymi. Stałe nazywamy WIELKIMI_LITERAMI.

Pola statyczne istnieją tylko w jednym egzemplarzu w ramach całej aplikacji i powiązane są z klasą, niezależnie od tego ile obiektów danej klasy utworzysz.

Metody statyczne mogą się odwoływać tylko do innych składowych statycznych.