

'Zapiski uczestnika Bootcampu Zajavka' by Bartek Borowczyk aka Samuraj Programowania

Dopiski na czerwono od Karola Rogowskiego

Dzień 4 Bootcampu

Omawialiśmy IntelliJ IDEA - na teraz bez jakiś dodatkowych notatek. Ale trzeba zrobić listę jakiś podstawowych skrótów klawiszowych do funkcji IntelliJ i do generowania kodu. Karol podrzuci w przyszłym tygodniu pewnie :)

Proszę Link, tylko nie uczcie się na pamięć, to samo wejdzie do głowy

Dzień 5 Bootcampu

W tym dniu Karol mówił nam o tym, że

- program w Javie zaczyna się od **wywołania metody main** (metoda main jest punktem wejściowym aplikacji)
- czym są i przede wszystkim jak używać **komentarzy (mówił by nie nadużywać)**
- poznaliśmy wykaz **słów zastrzeżonych** przez Javę, czyli słów które coś w javie robią. Karol wspominał by nie uczyć się ich na pamięć bo one z czasem i tak nam wejdą do głowy jak będziemy ich wielokrotnie i na codziennie używać.
- **'przypominajki'** - todo and fixme - Karol ich nie lubi. Ale jest i niektórzy używają. **Nie lubię ich bo potem ludzie o nich zapominają i często wala się dużo zapomnianych w kodzie**

```
// plik Example.java
```

```
class Example {  
    public static void main(String[] args) {  
        // kod metody  
    }  
}
```

Na tym etapie jeszcze nie przejmuj się składnią: czym jest class, public, static, void, String[] i args dowiesz się już wkrótce (nawet trochę w tych notatkach, ale przede wszystkim w kolejnych dniach).

Zapewniam, że nie ma tu wielkiej filozofii i krok po kroku to zrozumiesz.

Na co na tym etapie zwrócić uwagę i co zapamiętać:

- **Nazwa pliku musi być taka sama jak nazwa klasy w pliku** np. Example.java i nazwa klasy w pliku (po słowie kluczowym class), również Example. Wielkość liter też ma znaczenie! Czyli jak mamy w pliku `class Main`, to plik nazwiemy Main.java
- **Metoda main jest wymagana!** Jest wymagana żeby program w jakikolwiek sposób uruchomić. Możesz napisać też program bez metody main, ale jak go wtedy uruchomić dowiesz się na dalszych etapach. Dlatego na dzień dzisiejszy przyjmijmy, że programu bez metody main nie uruchomisz ;)
- Od metody main rozpoczyna się wykonanie programu. Jest to tzw. **punkt wejściowy aplikacji**.

Popatrzmy jeszcze raz na nasz kod źródłowy, tym razem jest trochę bardziej rozbudowany. Przede wszystkim pojawia się zawartość metody.

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("Mam zajawkę na Javę!");  
    }  
}
```

Zawartość metody zostanie wykonana w chwili uruchomienia programu, ponieważ metoda main jest punktem startowym programu. W tym konkretnym wypadku wyświetlony zostanie (wydrukowany) tekst w naszej konsoli. Oczywiście najpierw kod zostanie skompilowany do kodu bajtowego (w IntelliJ pojawi się katalog **/out** a w nim plik Main.class), który potem będzie wykonany przez maszynę wirtualną javy.

Efekt w konsoli:

```
C:\jdk-11\bin\java.exe "-javaagent:C:\Program Files\
Mam zajawkę na Javę!

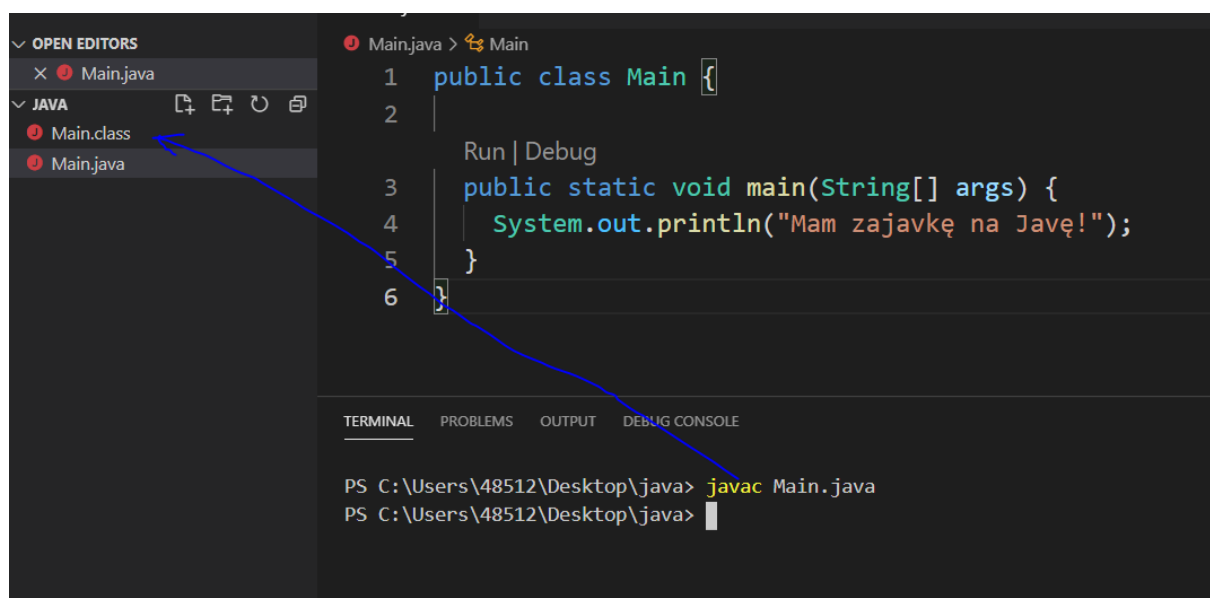
Process finished with exit code 0
|
```

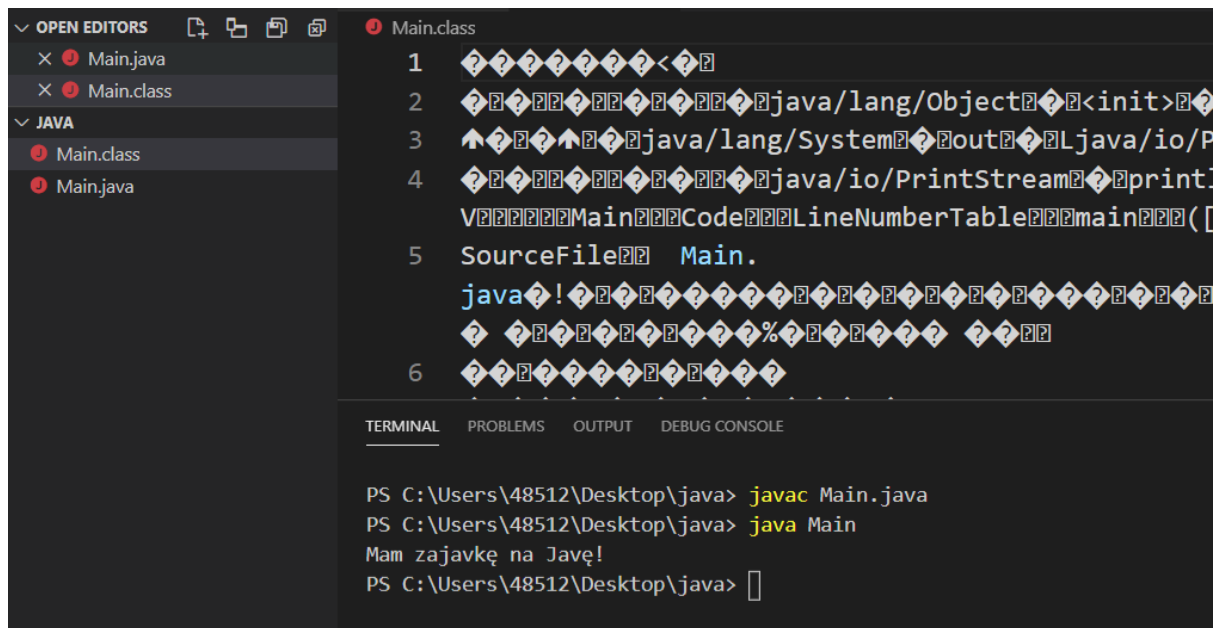
Przypomnę tylko że ten sam efekt uzyskamy gdy wpiszę w wierszu poleceń (oczywiście we właściwej ścieżce prowadzącej do danego pliku)

```
> javac Main.java
> java Main
```

Tak jak to widać na dwóch screenach poniżej.

Na screenie pierwszym kod w pliku Main.java + użycie kompilatora javac (w terminalu), który stworzył plik Main.class (kod bajtowy) - do którego zaznaczyłem niebieską linię. Na screenie drugim wywołujemy już ten plik (klasę) za pomocą polecenia **java**. Przy okazji zobaczcie na drugim screenie jak edytor widzi kod bajtowy - otworzyłem bowiem plik Main.class. Efektem wywołania Main.class jest wywołanie metody main a w niej wywołanie **metody println**, która drukuje przekazany tekst w konsoli/terminalu.





Powróćmy ponownie do tego kodu:

```

public class Main {

    public static void main(String[] args) {
        System.out.println("Mam zajawkę na Javę!");
    }
}

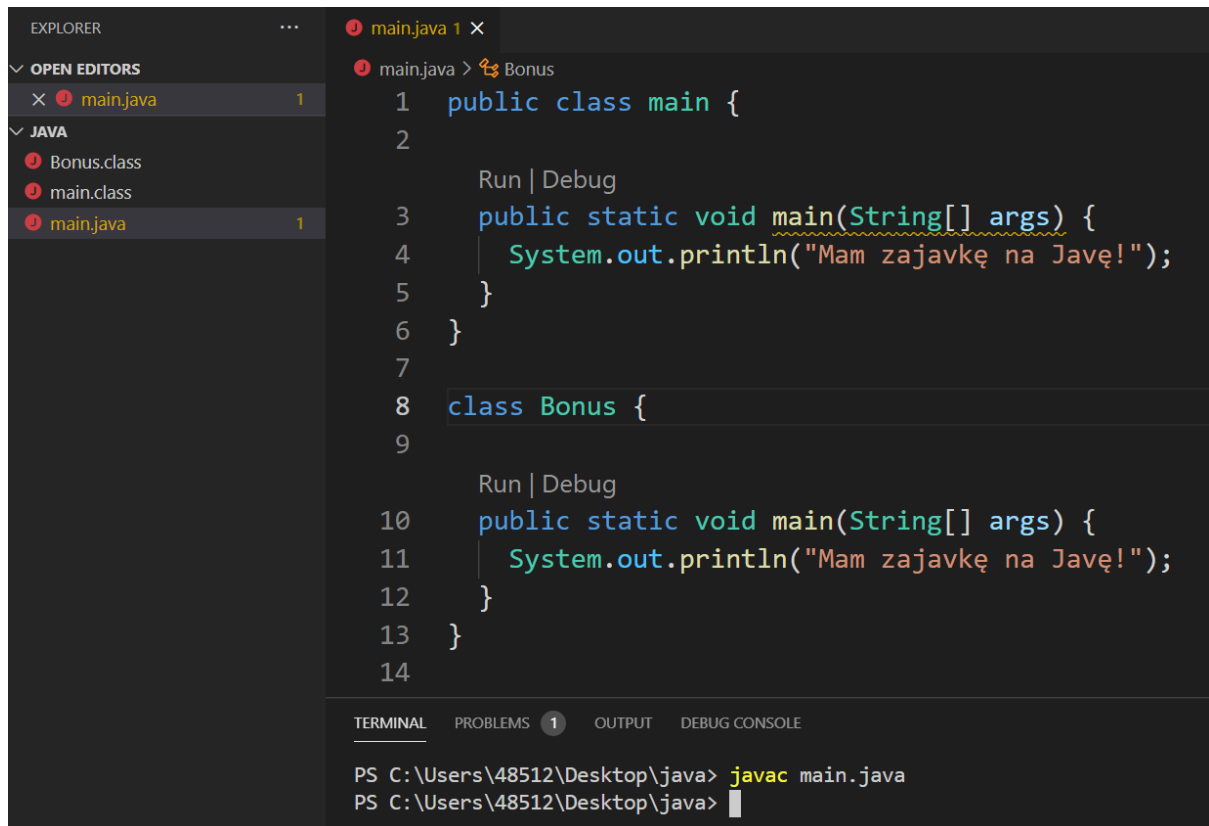
```

Na tym etapie zwróć uwagę, że mamy tutaj coś takiego jak klasa (słowo kluczowe class). W Javie cały kod musi znajdować się w klasach (w dużych projektach mogą być ich setki a nawet tysiące).

Zwróćmy uwagę że plik wynikowy (czyli ten tworzony w wyniku kompilacji z kodem bajtowym i rozszerzeniem .class) jest też tworzony z nazwą klasy (i jednocześnie pliku). Tak więc w pliku wynikowym kompilacji **każda klasa uzyskuje swój plik** z rozszerzeniem .class.

Widać to dobrze w przykładzie poniżej gdzie mamy dwie niemal identyczne klasy ale oczywiście z różnymi nazwami (nie można mieć dwóch klas o tych samych nazwach oczywiście).

Kompilując plik main.java (celowo plik i klasa małą literą byśmy zobaczyli wynik), który zawiera dwie klasy (najczęściej najlepszym rozwiązaniem jest jeden plik jedna klasa - ale teraz to tylko przykład), uzyskaliśmy dwa pliki wyjściowe. W tym przykładzie main.class (mała litera na początku, bo taka była nazwa klasy) i Bonus.class (wielka litera na początku, bo taka była nazwa klasy).



```
EXPLORER
...
main.java 1 x
main.java > Bonus
1 public class main {
2
3     Run | Debug
4     public static void main(String[] args) {
5         System.out.println("Mam zajawkę na Javę!");
6     }
7 }
8 class Bonus {
9
10    Run | Debug
11    public static void main(String[] args) {
12        System.out.println("Mam zajawkę na Javę!");
13    }
14 }

TERMINAL
PROBLEMS 1
OUTPUT
DEBUG CONSOLE
PS C:\Users\48512\Desktop\java> javac main.java
PS C:\Users\48512\Desktop\java>
```

!Pamiętaj: nazwy klas pisz wielkimi literami, podobnie jak nazwy plików. Staraj się by jeden plik zawierał jedną klasę.

Słowa kluczowe, które poznaliśmy w dzisiejszych zajęciach

public -- określa dostępność danego elementu dla innych elementów programu - w naszych przykładach widzimy już public przy klasie i przy metodzie. na teraz taka wiedza jest

wystarczająca, ale jeśli chcesz zapamiętać coś więcej, to to, że są to tzw. **modyfikatory dostępu** (jeden z kilku, mam też np. `private`, `protected` - na spokojnie je poznasz).

class -- słowo kluczowe, które służy do utworzenia klasy. Klasa jest podstawowym elementem każdego popularnego języka programowania zorientowanego obiektowo (będziemy się o programowaniu zorientowanym obiektowo uczyć w 4 i 5 tygodniu). Java wręcz wymusza na nasz używanie klas i to jest świetny sposób na uczenie się programowania zorientowanego obiektowo (skrót z angielskiego **OOP - Object Oriented Programming**), które jest fundamentem współczesnego programowania.

Zwróć uwagę, że po słowie `class` mamy zawsze nazwę (**identyfikator klasy**). Dzięki temu, możemy odwołać się do tworzonej klasy. np. `class Dog`, `class Main`, `class User` czy `class ElectricCar` itd.

!!! Uspokajam, na tym etapie nawet nie mówimy co robi klasy i czym jest. Przejdziemy do tego i szczegółowo wytłumaczymy, tak byś zrozumiał(a). Obiecuję!

Identyfikator klasy (jej nazwę) zgodnie z przyjętą konwencją piszemy wielką literą, dlatego nazwa pliku, też będzie nazywany z użyciem wielkiej litery na początku, bo nazwa pliku musi być identyczna jak nazwa klasy. Wiem, że się powtarzam, już nie będę :)

{ } -- Kolejny element składni to nawiasy klamrowe, w kręgach programistów pamiętających lata 90-te zwane także wąsami. Nawiasy klamrowe oznaczają ciało (zawartość) zarówno klasy jak i metod czyli ich zawartość. Są wymagane w Javie.

class Identyfiaktor { } - podstawowa składnia klas

Przejdźmy teraz do wiersza z definicją metody `main`. Wygląda on następująco:

```
public static void main(String[] args) { }
```

Metoda to mini program, którego instrukcje będą wywołane w chwili wywołania metody. Metoda `main` jest wywoływana domyślnie.

public - to modyfikator dostępu. Na teraz warto wiedzieć, że takie określenie przy metodzie oznacza, że jest ona... publiczna :) Czyli dostępna także spoza klasy (co to dokładnie znaczy, jeszcze się dowiemy). Metoda `main` musi być metodą publiczną. Przekonamy się w

przyszłości (Karol mi mówił ;)). że znacznie częściej występuje przy metodach modyfikator `private`, który oznacza, że dana metoda nie może być wywołana poza klasą.

// Jeszcze raz uspokajam, że to bardzo powierzchowne tłumaczenie, podczas bootcampu dowiesz się szczegółowo i dobrze, co to oznacza i jak korzysta się z danych metod.

static - to słowo kluczowe, które umożliwia wywołanie metody bez tworzenia obiektu. O Panie, brzmi strasznie. Ale takie się nie okaże w praktyce. Klasy będą służyły nam do tworzenie obiektów. Trochę tak jak plan domu (klasa) pozwala zbudować wiele domów (obiektów). Sama klasa nie jest obiektem! Używając `static` mamy możliwość coś zrobić bez zbudowania domu, np. `static ZmierzPokoj` może być metodą która pozwala zmierzyć wymiary pokoju na podstawie projektu (klasy), bo obiektu jeszcze nie ma. gdyby `ZmierzPokoj` była metodą nie posiadającą słowa kluczowego `static`, to można by użyć tej funkcji tylko poprzez odwołanie się do istniejącego (na podstawie klasy) obiektu. Na teraz nie ma co bardziej kombinować ;)

void - jest słowem kluczowym które oznacza, że dana metoda nic nie zwraca. Bo o czym się przekonasz wkrótce metody bardzo często zwracają jakąś wartość. Wyobraź sobie dwie metody w pseudokodzie

```
metoda nicNieZwraca dodaj(a, b) {  
    wydrukujNaEkranie(a + b)  
}
```

**`nicNieZwraca` zastępuje tu `void` właśnie*

```
metoda zwracaLiczbe odejmij(a,b) {  
    wydrukujNaEkranie(a - b)  
    zwróć a - b  
}
```

**zamiast `void` wskazujemy co zostanie zwrócone np. liczba (`int`) - o typach dowiemy się wkrótce.*

Pierwsza z metod `nic` nie zwraca (więc używamy `void`) - wykonuje po prostu jakieś zadanie. Druga wykonuje jakieś zadanie, ale oprócz tego coś zwraca (nie użyjemy więc `void` a np. `int`).

Coś co zostanie zwrócone z metody może zostać przekazane do innej metody czy zapisane i użyte w innym miejscu programu.

(String[] args) - wreszcie w nawiasach metody wskazujemy jakie parametry przyjmie metoda w chwili wywołania. Na metodach z pseudokodem gdzie metody pozwoliły nam dodać i odejmować nazwaliśmy je (parametry) a i b. W przypadku metody main musimy użyć String[].

Oczywiście może się zdarzyć, że metoda nie przyjmie żadnych parametrów. Jeśli szukasz innego przykładu parametrów, to spójrz na metodę, której użyliśmy do wydrukowania w konsoli:

```
System.out.println("Mam zajawkę na Javę!");
```

W tym wypadku metoda println otrzymuje parametr, który jest tekstem, a implementacja metody (jest to metoda wbudowana, której tylko używamy, ale nie potrzebujemy wiedzieć jak to jest zaimplementowane), sprawia, że przekazana w parametrze wartość pojawia się w konsoli.

String[] args - które jest użyte w nawiasach oznacza tablicę obiektów typu String (wartości tekstowe), na teraz nie jest to istotne, dowiesz się o stringach i innych obiektach w kolejnych dniach. Natomiast args jest identyfikatorem tej tablicy, przy czym nazwa może być inna (nie musi to być args, ale to konwencja, nic nie zmieniamy).

Po co więc ta tablica w metodzie main. Jeśli byśmy chcieli przekazać do programu jakieś wartości w chwili wywołania, to moglibyśmy to zrobić, póki co nie jest ona nam do niczego potrzebna, ale nie możemy jej usuwać.

tłumacząc na ludzki ten zapis: `public static void main(String args[]) { }`

otrzymamy:

```
dostępnaPozaKlasą niepotrzebującaDoWykonaniaObiektu nieZwracająca  
metoda o identyfikatorze main (Przekazująca tablicę o nazwie args  
zawierającą potencjalnie wartości tekstowe) {  
    // tutaj kod metody;  
}
```

Pamiętaj tylko, że od tej metody zaczyna się wykonanie naszego programu. I tyle ;)

Komentarze w Javie

Po pierwsze pamiętajmy, że komentarze są ignorowane przez kompilator. Komentarze są dla nas i innych programistów.

```
/*
```

```
komentarz wielowierszowy
```

```
*/
```

```
// komentarz jednowierszowy
```