



Java - podstawy

Autor: Przemek Baranowski

Prawa do korzystania z materiałów posiada Software Development Academy

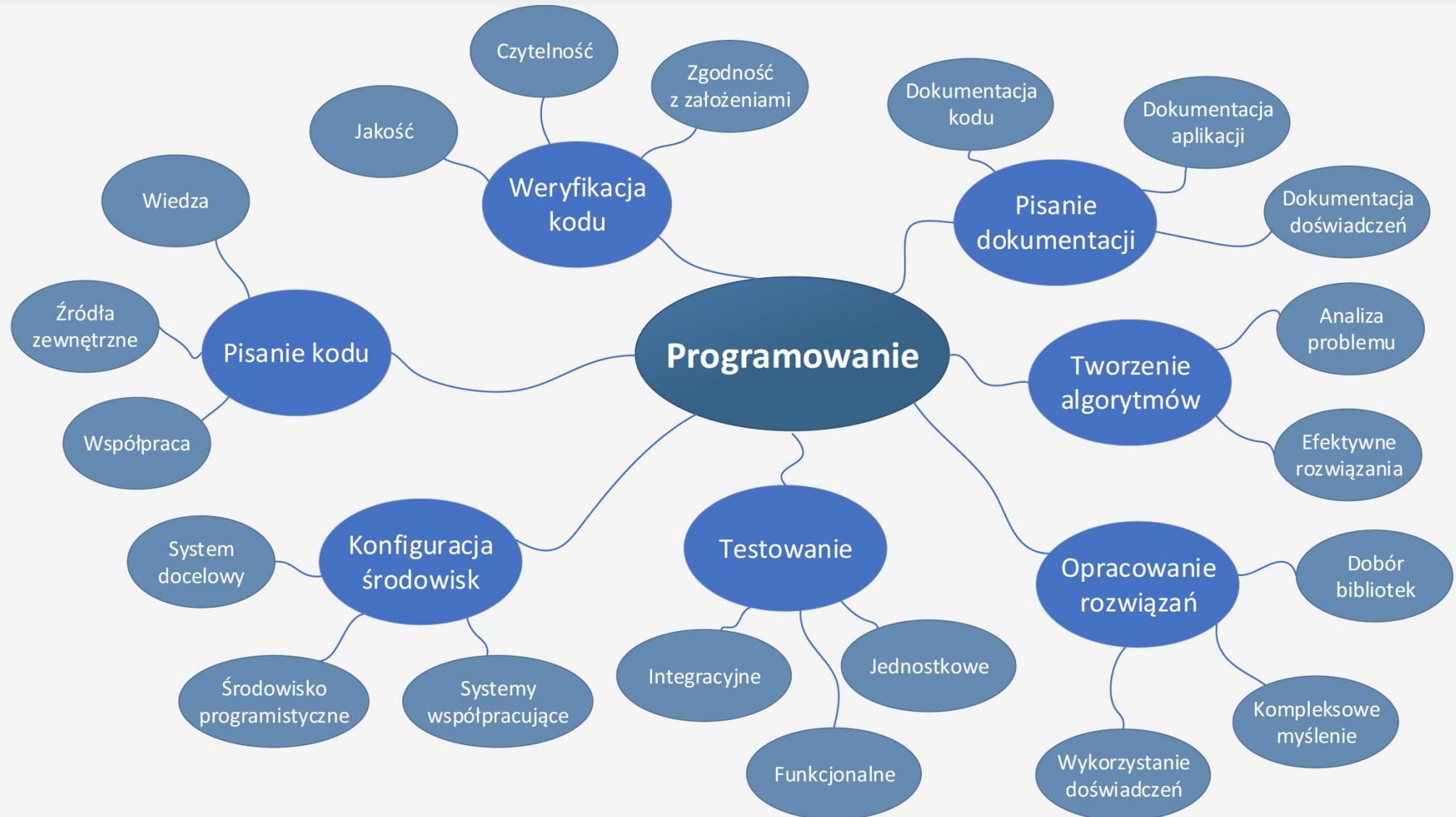
Java – wprowadzenie

- Programowanie
 - Java
- Środowisko programistyczne





Czym jest programowanie

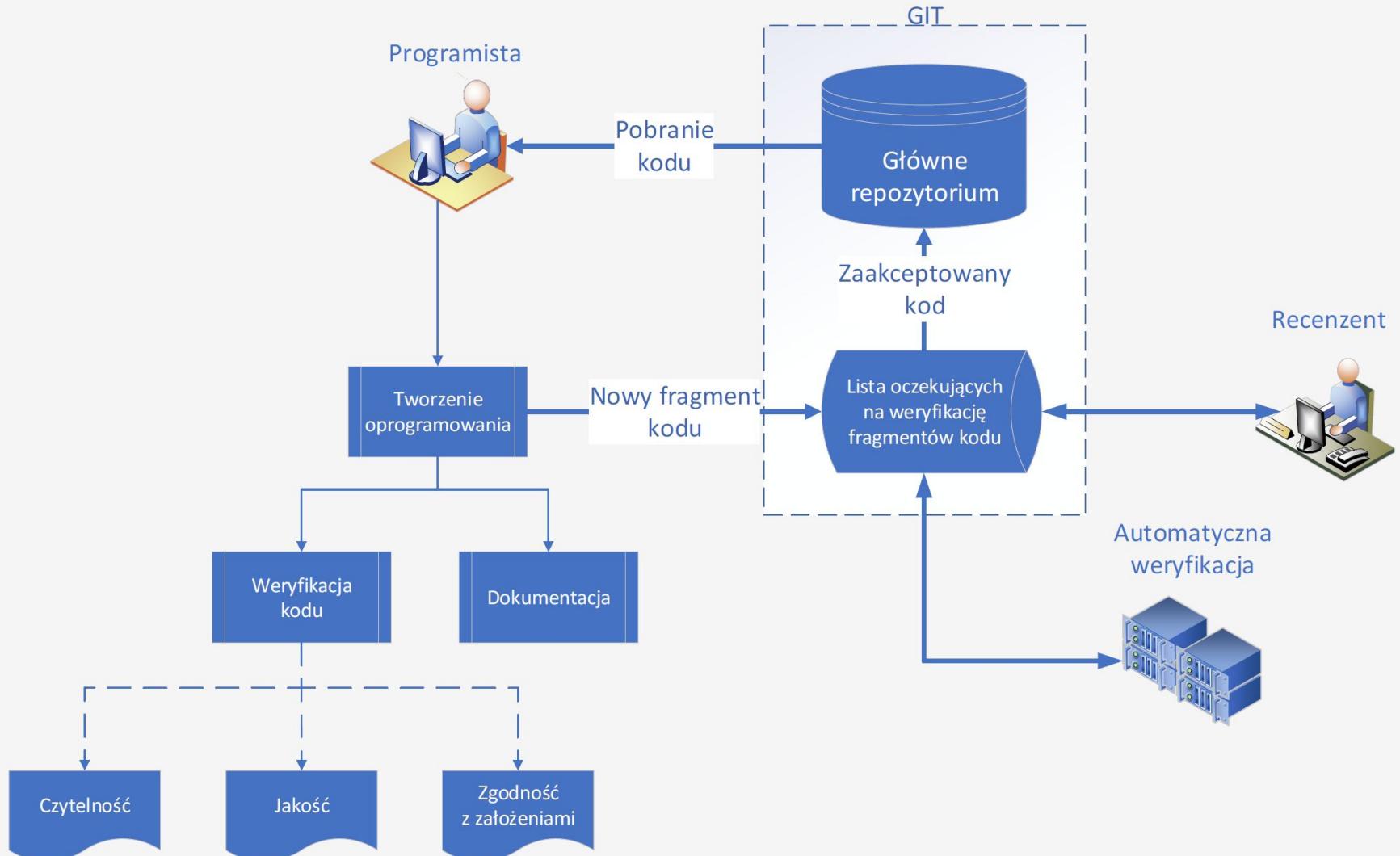


Autor: Marek Bobcow

Prawa do korzystania z materiałów posiada Software Development Academy



Praca programisty

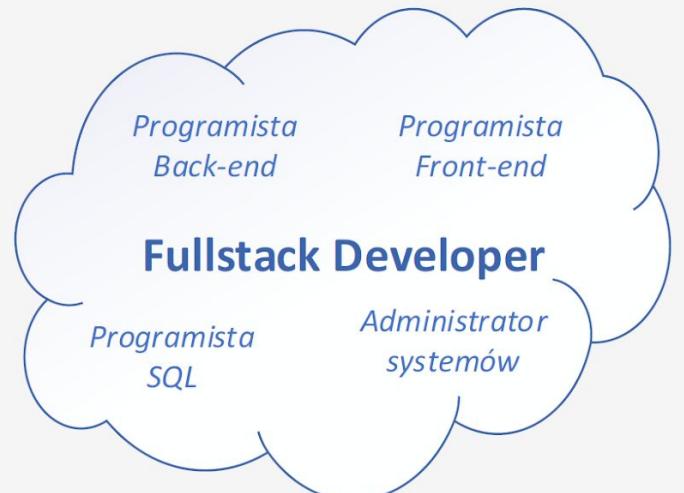


Autor: Marek Bobcow

Prawa do korzystania z materiałów posiada Software Development Academy



Ścieżki kariery





Co pomaga w programowaniu

- Cierpliwość
- Ciekawość i wnikliwość
- Planowanie przed wykonaniem (zawsze!)
- Spisywanie (pamiętanie) doświadczeń
- Analiza programów pisanych przez innych programistów
- Dokumentowanie (nikt tego nie lubi, ale każdy to docenia)*
- Współpraca
- Na etapie nauki ważne, aby działało „jakoś”, później zadbane o „ć”
- Podczas nauki możemy korzystać z języka polskiego
- Podczas tworzenia rozwiązań komercyjnych wyłącznie z angielskiego
- Stackoverflow

*dobry kod jest sam w sobie komentarzem



Znaki rozpoznawcze – Duke i filiżanka jawy





Krótką historią języka JAVA

- za twórcę języka JAVA uważany jest James Gosling, który rozpoczął pracę nad pierwszą wersją języka w roku 1991 wspólnie z Mike'em Sheridan i Patrick'em Naughton'em
- język początkowo nazywał się Oak (dąb), później nazwę zmieniono na Green (zielony) by ostatecznie przyjąć nazwę JAVA (odmiana krzewu kawy)
- część założeń języka zaczerpnięto z języka Smalltalk, który rozwijał się w latach '70 i '80, składnia języka była wzorowana na wymyślonym w latach '80 języku C++
- pierwsza wersja języka została opublikowana przez Sun Microsystems w roku 1996
- od roku 2010 JAVA jest rozwijana głównie przez firmę Oracle Corporation



Podstawowe założenia JAVA

- write once, run anywhere (napisz raz, wykonuj gdzie chcesz)
- simple, object-oriented, and familiar (prosty, obiektowy, znany - podobny do C/C++)
- robust and secure (solidny i bezpieczny)
- architecture-neutral and portable (niezależny od architektury sprzętowej, przenośny)
- high performance (wydajny)
- interpreted, threaded, and dynamic (interpretowany, wątkowy, dynamiczny)



Jak działa język programowania

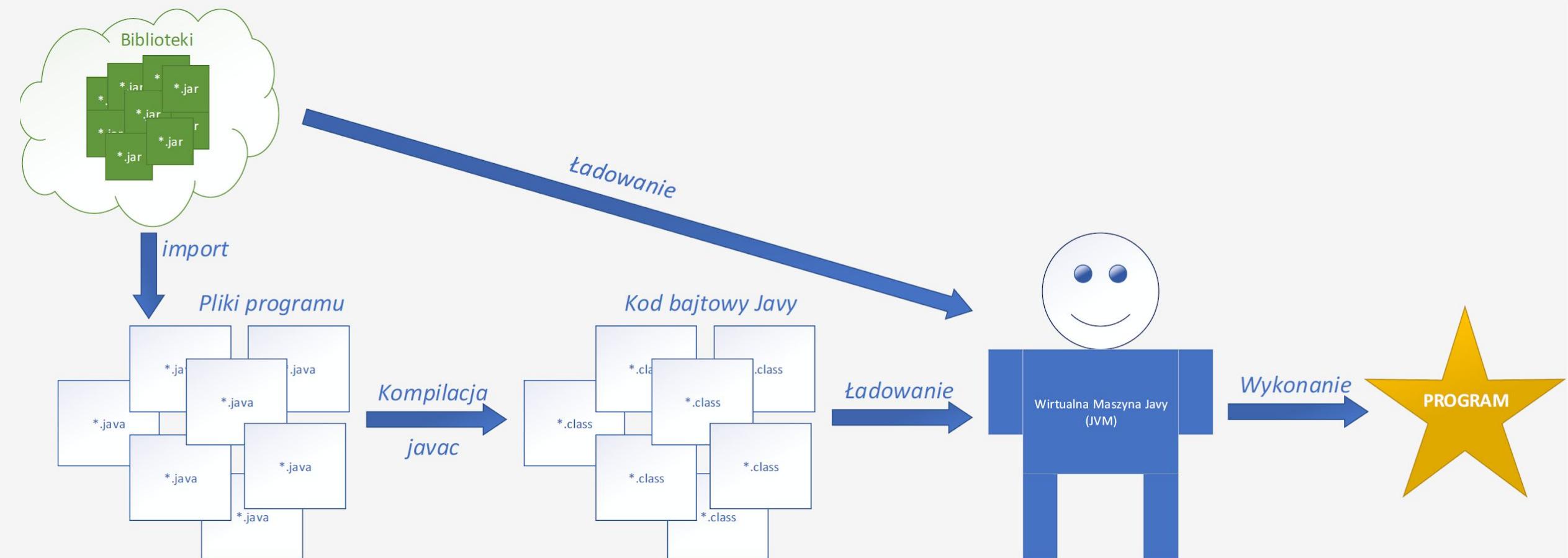
Kod źródłowy programu - zapis programu komputerowego przy pomocy określonego języka programowania, opisujący operacje jakie powinien wykonać komputer na zgromadzonych lub otrzymanych danych. Kod źródłowy jest wynikiem pracy programisty i pozwala wyrazić w czytelnej dla człowieka formie strukturę oraz działanie programu komputerowego.

Kompilator - program, który kod źródłowy zamienia w kod napisany w innym języku. Oprócz tego kompilator ma za zadanie odnaleźć błędy leksykalne i semantyczne oraz dokonać optymalizacji kodu.

Kod bajtowy - bytecode, wynik kompilacji programu napisanego w Javie, kod ten jest zrozumiały dla środowiska uruchomieniowego Java (JVM)



Jak działa Java



Autor: Marek Bobcow

Prawa do korzystania z materiałów posiada Software Development Academy



Środowisko - JVM, JRE, JDK

JDK

java, javac, jdb, appletviewer, javah, javaw
jar, rmi.....

JRE

Class Loader, Byte Code Verifier
Java API, Runtime Libraries

JVM

Java Interpreter
JIT
Garbage Collector
Thread Sync.....



IDE - Integrated Development Environment

- popularne obecnie
 - IntelliJ
 - Visual Studio
- popularne dawniej
 - Eclipse
 - Netbeans
- do innych zastosowań
 - Android Studio
 - XCode
 - Visual Studio Code
 - Atom



Środowisko - przygotowanie

1. Instalujemy:
 - Java - JDK 13
 - IntelliJ Community Edition
2. Sprawdź czy masz poprawnie zainstalowane JDK
W konsoli wpisz:
`java --version`
`javac --version`
3. Stwórz nowy projekt w IntelliJ



Ćwiczenie

1. W katalogu „src” utwórz plik o nazwie Hello.java
 - Prawy klik na „src” -> New class
 - Name: Hello -> OK
2. W pliku Hello.java umieść następującą treść:

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

3. Uruchom program zielonym trójkątem obok nazwy klasy lub klawiszami Crtl+Shift+F10

Jeśli wszystkie kroki zostały wykonane poprawnie na konsoli pojawi się napis: „Hello World”.



Podsumowanie

1. Z czym na co dzień pracuje programista?
2. Co robi kompilator?
3. Co to jest kod bajtowy (Bytecode)?
4. Czy kompilator Java jest częścią systemu operacyjnego?
5. Co to jest JVM?
6. Czym się różni JRE od JDK?
7. Co to jest IDE i do czego się go wykorzystuje?

Java – typy danych

- Typy danych
- zmienne, stałe





Zmienne

```
public class Zmienne1 {  
  
    int staloprzecinkowa = 10;  
    String ciagZnakow = "dowolny ciąg 12345..";  
    boolean warunek = true;  
    double stanPortfela = 8.79;  
  
}
```

Deklaracja - określamy typ i nazwę zmiennej

Inicjalizacja - nadanie wartości zmiennej

```
public class Zmienne{  
    public static void main(String[] args){  
        int liczba; // Deklaracja  
        liczba = 5; // Inicjalizacja  
    }  
}
```

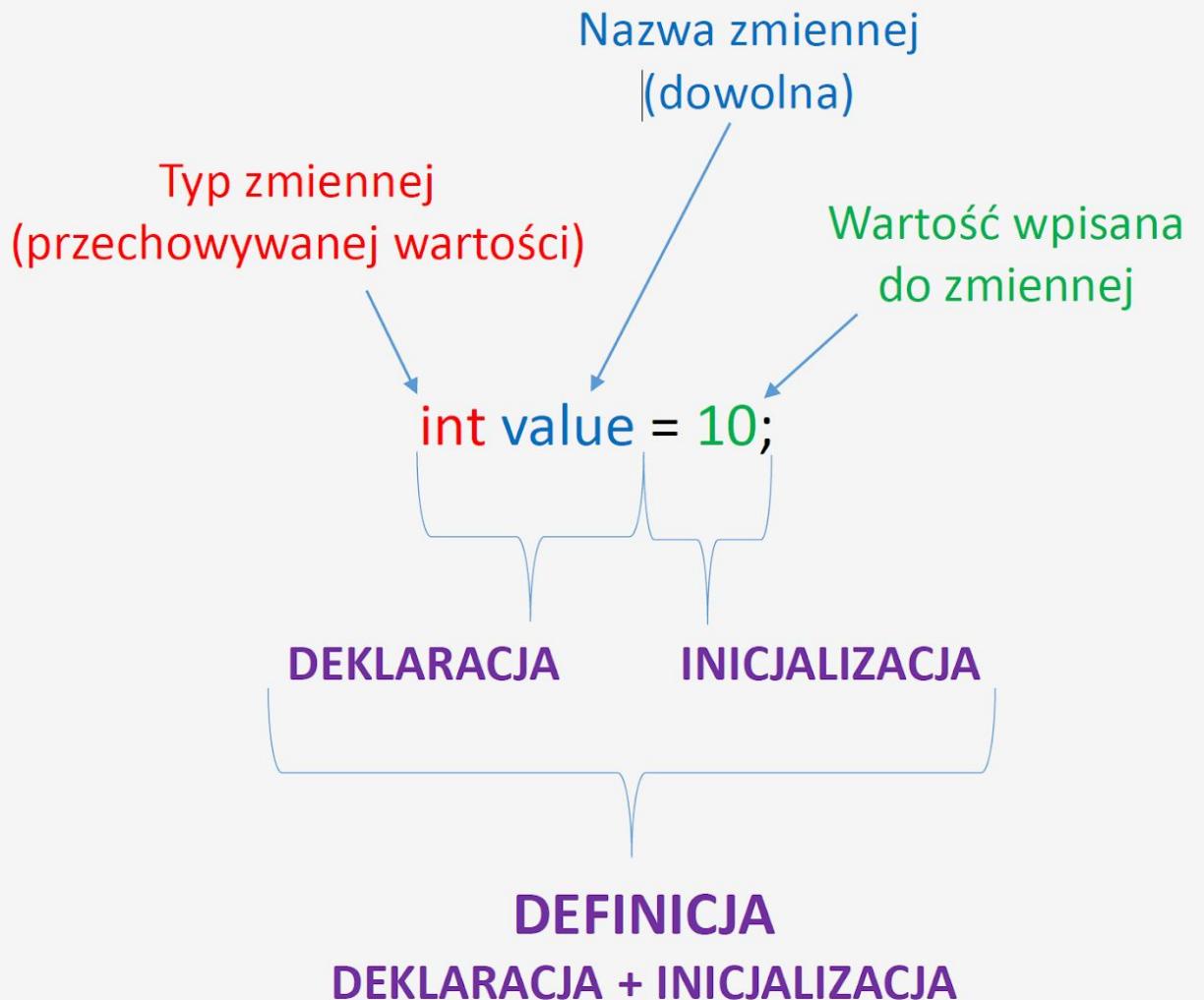


Tworzenie zmiennej

W celu utworzenia nowej zmiennej musimy:

- określić typ danych, jaki będzie przechowywała,
- wybrać nazwę,
- przypisać do niej określoną wartość (jeśli tego nie zrobimy, to zmienna otrzyma wartość domyślną).

Deklaracja i inicjalizacja zmiennej **nie musi** odbywać się w jednej linii (instrukcji).





Typy danych

W Javie podobnie jak w innych językach wyróżniamy wiele typów danych mogących przechowywać zarówno liczby stało i zmiennoprzecinkowe, znaki, ciągi znaków oraz typ logiczny. Java posiada ścisłą kontrolę typów, czyli mówiąc prościej każdy obiekt musi mieć określony typ.

byte - 1 bajt - zakres od -128 do 127

short - 2 bajty - zakres od -32 768 do 32 767

int - 4 bajty - zakres od -2 147 483 648 do 2 147 483 647

long - 8 bajtów - zakres od -2^{63} do $(2^{63})-1$ (posiadają przyrostek **L**, lub **l**)

float - 4 bajty - max ok 6-7 liczb po przecinku (posiadają przyrostek **F**, lub **f**)

double - 8 bajtów - max ok 15 cyfr po przecinku (posiadają przyrostek **D**, lub **d**)

char – odpowiada jednemu znakowi (np. literze), może przechowywać liczby całkowite z zakresu od 0 do 65 535.

boolean – wartość *logiczna*, może przyjąć jedną z wartości **true** (oznaczającą prawdę) lub **false** (oznaczającą fałsz).



Operatory

- Dodawanie

$x = x + 10$

$x += 10$

- Odejmowanie

$x = x - 10$

$x -= 10$

- Mnożenie

$x = x * 10$

$x *= 10$

- Dzielenie

$x = x / 10$

$x /= 10$

- Inkrementacja

post $\rightarrow x++$

pre $\rightarrow ++x$

- Dekrementacja

post $\rightarrow x--$

pre $\rightarrow --x$

- Jest równe

$x == y$

- Jest różne

$x != y$

- Oraz

$x \&& y$

- Lub

$x || y$

- Większy

$x > y$

- Mniejszy

$x < y$

- Większy lub równy

$x >= y$

- Mniejszy lub równy

$x <= y$



Zadanie

1. Wprowadź swoje imię do zmiennej typu String. Wypisz na konsolę „Witaj, <imię>” wykorzystując funkcję println
2. Wypisz dwa dowolnełańcuchy znaków jeden pod drugim z wykorzystaniem funkcji print
3. Wprowadź dowolną wartość z kilkoma cyframi po przecinku do zmiennej typu double .
Wyświetl podaną wartość z dokładnością do dwóch miejsc po przecinku.
4. *Wyświetl trzy dowolne ciągi znaków w jednej linii tak, aby były wyrównane do prawej krawędzi 15 znakowych bloków. Np. „ test t wprowadzenie”.
5. **Wyświetl wartości 192, 168, 1, 10 heksadecymalnie w formacie XX:XX:XX:XX, dla podanych wartości powinniśmy otrzymać: „C0:A8:01:0A”



Zadania ...

Przydatne rzeczy:

Pobranie tekstu od użytkownika:

```
import java.util.Scanner;

public class Wejscie1 {

    public static void main(String[] args) {
        Scanner wejście = new Scanner(System.in);
        String napisanaLinia = wejście.nextLine();
    }

}
```



Zadania

6. Wprowadź swoje imię oraz wiek z poziomu konsoli i zapisz je do zmiennej typu String i int. Wypisz na konsolę „Witaj, <imię>, masz <wiek> lat”.
7. Wprowadź z poziomu konsoli dwie wartości, dodaj je do siebie i wypisz ich sumę.

Instrukcje warunkowe

- if-else
- switch





Instrukcje warunkowe - if-else

```
Jeśli (warunek lub warunki) {  
    Instrukcje do wykonania  
} W przeciwnym wypadku jeśli (warunek lub warunki) {  
    Instrukcje do wykonania  
} W pozostałych wypadkach {  
    Instrukcje do wykonania  
}
```

- Warunki sprawdzamy od góry do dołu.
- Jeśli którykolwiek z warunków będzie spełniony, to wykonujemy instrukcje dla tego warunku. Pozostałych warunków nie sprawdzamy.
- Warunek może być pojedynczy, np. „`x == 10`” lub możemy sprawdzić kilka warunków jednocześnie, np. „`x == 10 && y >= 4`”.
- Liczba bloków „W przeciwnym wypadku jeśli ...” może być dowolna.



Instrukcje warunkowe - if-else

```
public class Conditions {  
    public static void main(String[] args) {  
  
        int number = 200;  
  
        if (number < 10) {  
            System.out.println("Liczba jest mniejsza niż 10");  
        } else if (number == 10) {  
            System.out.println("Liczba jest równa 10");  
        } else {  
            System.out.println("Liczba jest większa niż 10");  
        }  
    }  
}
```



Instrukcje warunkowe - if-else - Zadania

1. Zmodyfikuj przykładowy program tak, aby pobierana liczba pochodziła z konsoli. Przetestuj program dla każdego przypadku (liczba mniejsza, równa lub większa od 10).
Podpowiedź: skorzystaj z metody Scannera nextInt
2. Pobierz z konsoli wartość od 0 do 9. Na podstawie otrzymanej wartości wyświetl dowolny znak. Na przykład dla numeru 0 wyświetl „*”, dla 1 wyświetl „\$” (lub dowolny inny).
3. *Jak wyżej, ale zamiast wartości operuj na łańcuchach znaków (Stringach). Na przykład dla słowa „gwiazdka” wyświetl „*”.



Instrukcje warunkowe - switch

- Porównujemy zmienną do określonych z góry wartości (w przykładzie nasze wartości oznaczone są przez „...”).
- Musimy uwzględnić wszystkie możliwe wartości, jakie przyjmie nasza zmienna.
- Jeśli nasza zmienna nie jest zgodna z żadną ze zdefiniowanych wartości, to wykonają się instrukcje z bloku „domyślnego”.
- Każdy blok instrukcji musi zostać na końcu przerwany. W innym wypadku zostaną wykonane instrukcje dla każdej kolejnej wartości zmiennej

```
Porównaj zmienną (zmienna) {  
    wartość zmiennej to ...:  
        Instrukcje do wykonania  
        przerwij  
    wartość zmiennej to ...:  
        Instrukcje do wykonania  
        przerwij  
    domyślnie wykonaj:  
        Instrukcje do wykonania  
        przerwij  
}
```



Instrukcje warunkowe - switch

```
3 ►  public class ConditionsSwitch {  
4 ►  ↴    public static void main(String[] args) {  
5  
6      int number = 200;  
7  
8      switch (number) {  
9          case 10: {  
10             System.out.println("Liczba to 10");  
11             break;  
12         }  
13         case 100: {  
14             System.out.println("Liczba to 100");  
15             break;  
16         }  
17         default: {  
18             System.out.println("Liczba jest różna od 10 i 100");  
19             break;  
20         }  
21     }  
22 }  
23 }
```



Instrukcje warunkowe - switch - Zadania

1. Zmodyfikuj przykładowy program tak, aby pobierana liczba pochodziła z konsoli. Przetestuj program dla każdego przypadku (liczba mniejsza, równa lub większa od 10).
Podpowiedź: skorzystaj z metody Scannera nextInt
2. Pobierz z konsoli wartość od 0 do 9. Na podstawie otrzymanej wartości wyświetl dowolny znak. Na przykład dla numeru 0 wyświetl „*”, dla 1 wyświetl „\$” (lub dowolny inny).
3. Jak wyżej, ale zamiast wartości operuj na łańcuchach znaków (Stringach). Na przykład dla słowa „gwiazdka” wyświetl „*”.
4. *Jak wyżej, ale po wyświetleniu danego znaku wyświetl wszystkie kolejne możliwości (jeśli posiadamy case „gwiazdka”, a kolejny to „dolar” to po wpisaniu gwiazdki powinniśmy otrzymać zarówno gwiazdkę, jak i dolar).



Zadania

1. Napisz prosty kalkulator, który:
 - a) Przyjmuje dwie wartości x i y , zwraca ich sumę oraz różnicę. Wykorzystaj funkcję `printf` albo `String.format` tak, aby znak „=”, „+” oraz „-” były w tym samym miejscu jeden pod drugim. Nie używaj metody `nextInt`
 - b) j/w + podawanie informacji, czy chcemy, aby dokonać dodawania, czy odejmowania poprzez napisanie „suma” lub „różnica”. Wykorzystaj instrukcję warunkową `if` `else`
 - c) *j/w + użytkownik może zdecydować jaką operację chce wykonać uwzględniając mnożenie i dzielenie. Wykorzystaj instrukcję warunkową `switch` `case`
 - d) **j/w + skorzystaj z „ternary operator” zamiast standardowej instrukcji `if` `else`
 - e) **obsłuż sytuację, w której użytkownik poda wartość 0 podczas
2. Napisz program, który pozwoli określić, czy wprowadzona przez użytkownika liczba:
 - a) Jest parzysta
 - b) Jest podzielna przez drugą podaną liczbę

Autor: Marek Bobcow

Prawa do korzystania z materiałów posiada Software Development Academy

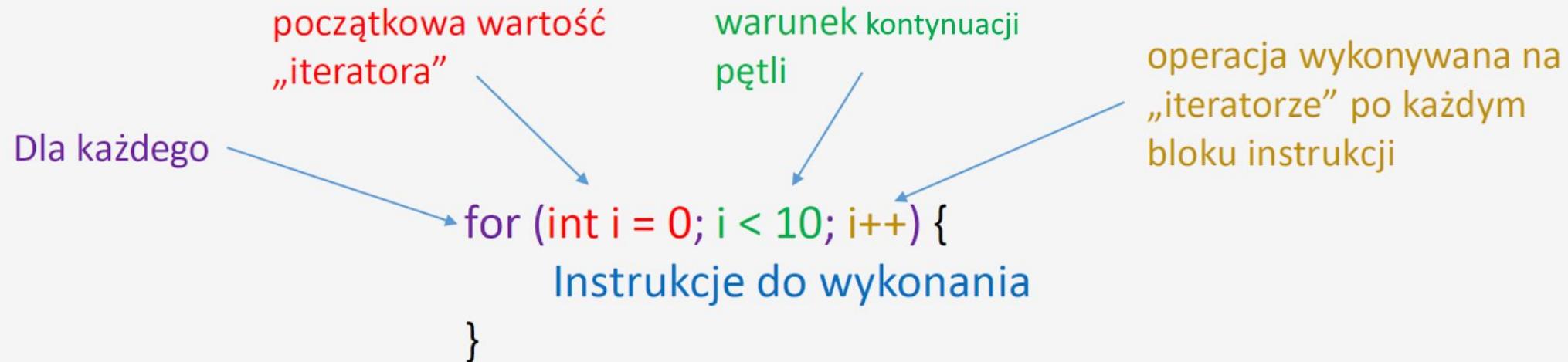
Java – pętle

- for
- while
- do-while





Pętle - for



- Pętla **for** pozwala na wykonanie „instrukcji do wykonania” określona liczbę razy.
- Zarówno początkowa wartość iteratora, jak i warunek zakończenia pętli może być dowolny.
- Jeśli zmienną iterującą tworzymy na potrzeby danej pętli („`int i = ...`”), to po wyjściu z pętli zmienna iterująca jest usuwana z pamięci.
- Zastosowanie: operacje na zbiorach danych (tablicach, obiektach), wypisywanie wartości



Pętle - for

```
3 ► public class LoopsFor {  
4 ► |   public static void main(String[] args) {}  
5 |   for (int i = 0; i < 5; i++) {  
6 |       System.out.println("Aktualna wartość: " + i);  
7 |   }  
8 | }  
9 }  
10 }
```

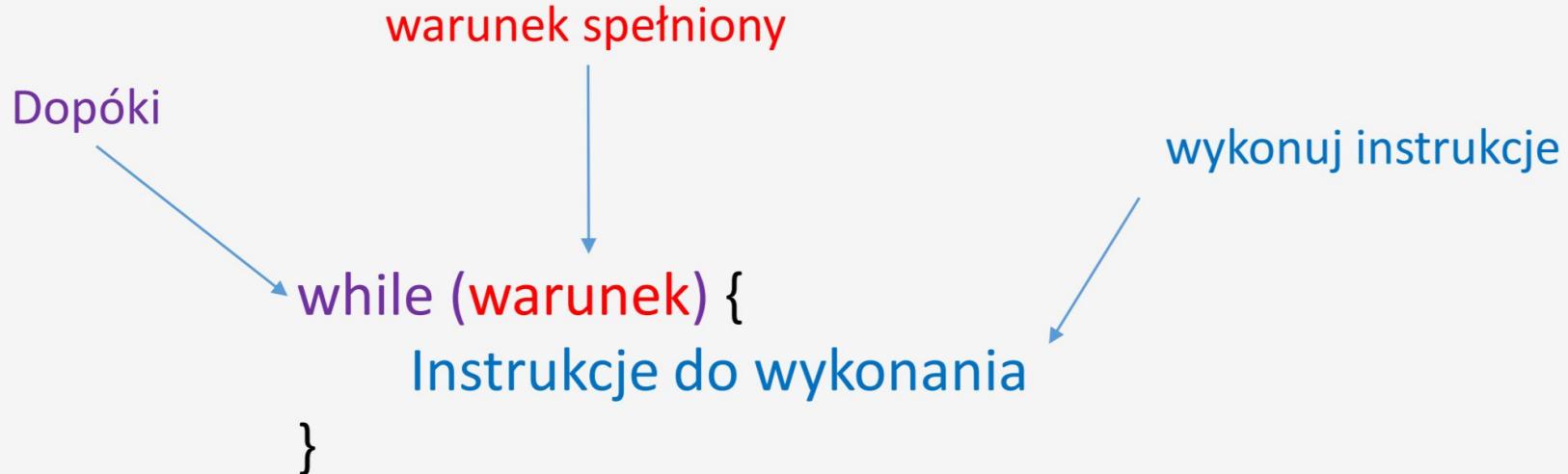


Pętle - for - Zadania

1. Wewnątrz pętli for (np. 10 razy) pobieraj produkt do kupienia i wyświetl go na ekran w postaci : „Dodałem do koszyka , to nasz produkt!”
2. Stwórz pętle for, która wykona 5 iteracji. Wewnątrz pętli pobieraj z konsoli dowolną wartość typu int. Po wyjściu z pętli zwróć sumę tych wartości.
3. *Stwórz pętle wewnątrz pętli (pamiętaj o różnych nazwach zmiennych iterujących!). Wyświetl wartości iteratorów w postaci: „ : ”.
4. **Wyświetl kwadrat składający się z samych gwiazdek „*”, których liczba (długość boku kwadratu) będzie równa podanej z konsoli wartości.
5. **Jak wyżej, ale znak, z którego składał się kwadrat, również pobierz z konsoli.



Pętle - while



- Pętla while pozwala na wykonywanie określonych instrukcji tak długo, jak długo warunek pętli jest spełniony.
- W przeciwieństwie do pętli for nie podajemy danych dotyczących iteratora (wartości początkowej, instrukcji wykonywanej po każdym bloku)
- Zastosowanie: pobieranie instrukcji od użytkownika aplikacji, do momentu otrzymania określonego rozkazu; czytanie zawartości pliku; usypianie programu do momentu spełnienia określonego warunku

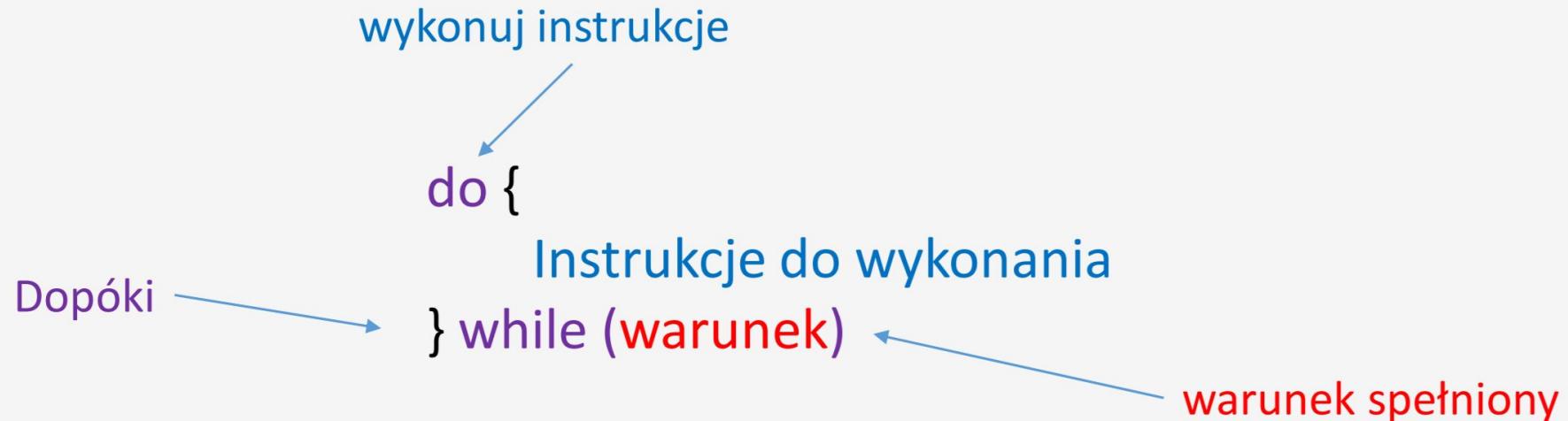


Pętle - while

```
5 ► public class LoopsWhile {  
6  
7 ►   public static void main(String[] args) {  
8     Random r = new Random();  
9     int i = -1;  
10    while (i % 5 != 0) {  
11      i = r.nextInt( bound: 100);  
12      System.out.println("Aktualna wartość: " + i);  
13    }  
14  }  
15 }
```



Pętle - do-while



- Pętla `do while` pozwala na wykonywanie określonych instrukcji tak długo, jak długo warunek pętli jest spełniony. W przeciwieństwie do pętli `while` wykona się zawsze chociaż jeden raz, nawet, jeśli warunek nie będzie spełniony.
- Zastosowanie: próba połączenia z inną aplikacją tak długo, aż nie otrzymamy informacji o prawidłowym połączeniu; zastosowania podobne do pętli `while`



Pętle - do-while

```
3 ► public class LoopsWhile {  
4  
5 ►   public static void main(String[] args) {  
6       int i = 0;  
7       do {  
8           System.out.println("Aktualna wartość: " + i);  
9           i++;  
10      } while (i < 5);  
11    }  
12 }
```



Pętle - while - Zadania

1. Wewnątrz pętli while (np. 10 razy) pobieraj produkt do kupienia i wyświetlaj go na ekran w postaci : „Dodałem do koszyka , to nasz produkt!”
2. Stwórz pętle while, która wykona 5 iteracji. Wewnątrz pętli pobieraj z konsoli dowolną wartość typu int. Po wyjściu z pętli zwróć sumę tych wartości.
3. Jak wyżej, ale zwróć sumę tylko tych wartości, które były większe od 10;
4. *Pobieraj i wyświetlaj dowolny ciąg znaków od użytkownika tak długo, aż nie napisze „koniec”.

Java – tablice

- tablice jednowymiarowe
 - pętle for-each
- tablice wielowymiarowe
 - varargs





Tablice - tworzenie

Pamiętajmy, że do utworzenia tablicy potrzebny jest operator **new**

```
int[] tablica = new int[100];
```

Powyższa instrukcja tworzy i inicjuje tablicę, w której można zapisać 100 liczb całkowitych. Elementy tablicy są **numerowane od 0** (w przypadku wyżej od 0 do 99)

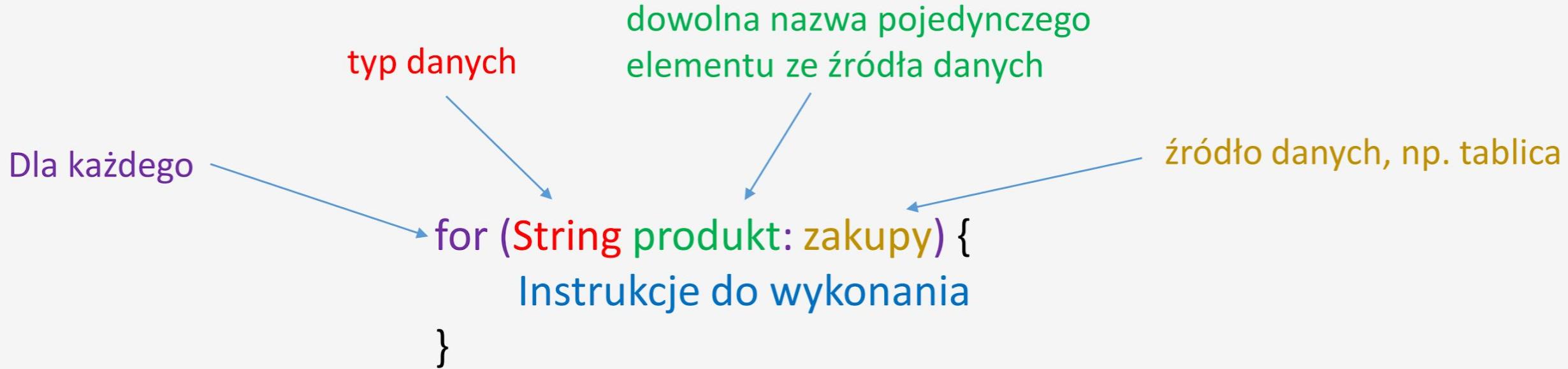


Tablice 1-wymiarowe

```
3 ► public class SingleDimensionArray {  
4 ►     □     public static void main(String[] args) {  
5         int[] array = {1, 5, 2, 6, 8, 3, 2, 7, 6, 5};  
6  
7         System.out.println("Tablica ma rozmiar: " + array.length);  
8  
9         int item5 = array[5];  
10        System.out.println("Element o indeksie 5 to: " + item5);  
11  
12  
13        System.out.println("Zawartość tablicy:");  
14        □     for (int item : array) {  
15            System.out.println(item);  
16        }  
17    }  
18 }
```



Pętle - for-each



- Pętla for each pozwala na wykonanie jednakowych instrukcji dla każdego elementu z danego zbioru danych. Jest to uproszczona forma pętli `for(int i=0; i <)` umożliwiająca ominięcie zmiennej iterującej.
- Źródło danych musi przechowywać elementy tego samego typu (np. tablica przechowująca zmienne typu String).



Tablice i foreach - Zadania

1. Utwórz tablicę przechowującą wartości typu int o rozmiarze podanym przez użytkownika. Wypełnij ją losowymi wartościami wewnątrz pętli **for**.
 - a) Zwróć sumę, średnią i medianę tych wartości.
 - b) Wypisz tylko liczby parzyste
2. Wylosuj liczbę z przedziału 1-10. Proś użytkownika o jej zgadnięcie, do momentu aż wprowadzi wylosowaną liczbę. (**while**)
3. Stwórz tablicę zawierającą 5 imion. Wewnątrz pętli **for-each** dopisz imiona do zmiennej typu String, oddzielając je przecinkami. Wyświetl utworzony łańcuch znaków. Np. „Małgorzata, Jan, Jakub”.
4. *Jak wyżej, ale dopisz tylko imiona, które składają się z mniej, niż 5 znaków.



Tablice wielowymiarowe

[0]	[1]	[2]	[3]	[4]	
[0]	0	0	0	0	0
[1]	0	0	0	0	0
[2]	0	0	0	0	0
[3]	0	0	0	0	0
[4]	0	0	0	0	0

matrix = new int[5][5];

(a)

[0]	[1]	[2]	[3]	[4]	
[0]	0	0	0	0	0
[1]	0	0	0	0	0
[2]	0	7	0	0	0
[3]	0	0	0	0	0
[4]	0	0	0	0	0

matrix[2][1] = 7;

(b)

[0]	[1]	[2]	
[0]	1	2	3
[1]	4	5	6
[2]	7	8	9
[3]	10	11	12

```
int[][] array = {  
    {1, 2, 3},  
    {4, 5, 6}, (c)  
    {7, 8, 9},  
    {10, 11, 12}  
};
```



Varargs

```
3 ► public class Varargs {  
4 ► @ [ ] public static void main(String[] args) {  
5 [ ] if (args.length > 0) {  
6 [ ] System.out.println("Przekazane argumenty:");  
7 [ ] for (String arg : args) {  
8 [ ] System.out.println(arg);  
9 [ ] }  
10 [ ] } else {  
11 [ ] System.out.println("Nie przekazano żadnych argumentów");  
12 [ ] }  
13 [ ] }  
14 [ ] }  
15 }
```



Tablice i varargs- Zadania

Użytkownik przekazuje 2 argumenty do programu: nazwę figury ('kwadrat', 'prostokąt' itd.) oraz znak z użyciem którego ma być narysowana (*, \$, #...). np.

Program arguments:

kwadrat #

Program powinien umieć narysować:

1. Kwadrat pusty w środku (wyłącznie krawędzie).
2. Prostokąt pusty w środku.
3. Literę „L”, o krawędziach równej długości.
4. *Trójkąt prostokątny.
5. *Jak punkt 1, ale najpierw wprowadź wszystkie elementy do tablicy dwuwymiarowej, a dopiero później je wyświetl.
6. *Kwadrat pusty w środku z jedną przekątną
7. **Kwadrat pusty w środku z dwiema przekątnymi.



Zadanie podsumowujące

Napisz aplikację, która:

- a) Będzie posiadała tablicę jednowymiarową składającą się z 3 elementów typu String: kombinerek, nożyczek i śrubokrętu.
- b) Pobierze imię użytkownika.
- c) Wypisze na ekran: „Witaj ! W naszej ofercie mamy: ” + lista elementów z tablicy produktów zdefiniowanych w podpunkcie a, każdy w nowej linii zaczynający się od myślnika + „Co chciałbyś kupić?”
- d) Za pośrednictwem pętli switch – case przeanalizuj wybór użytkownika, dla opcji default wypisz „Takiego produktu nie mamy w ofercie”
- e) Dla prawidłowego wyboru usuń dany element z tablicy i potwierdź użytkownikowi wybrany produkt.
- f) Potwierdź usunięcie elementu z listy poprzez jej ponowne wyświetlenie.
- g) *Imię użytkownika przekaż przez Varargs jako pierwszy parametr.
- h) *Produkty do kupienia również.
- i) W przypadku błędного wyboru produktu pozwól na ponowny wybór tak długo, aż użytkownik nie napisze „do widzenia”

Java – programowanie obiektowe

- klasy, obiekty, metody
 - pakiety, importy
 - zakresy widoczności

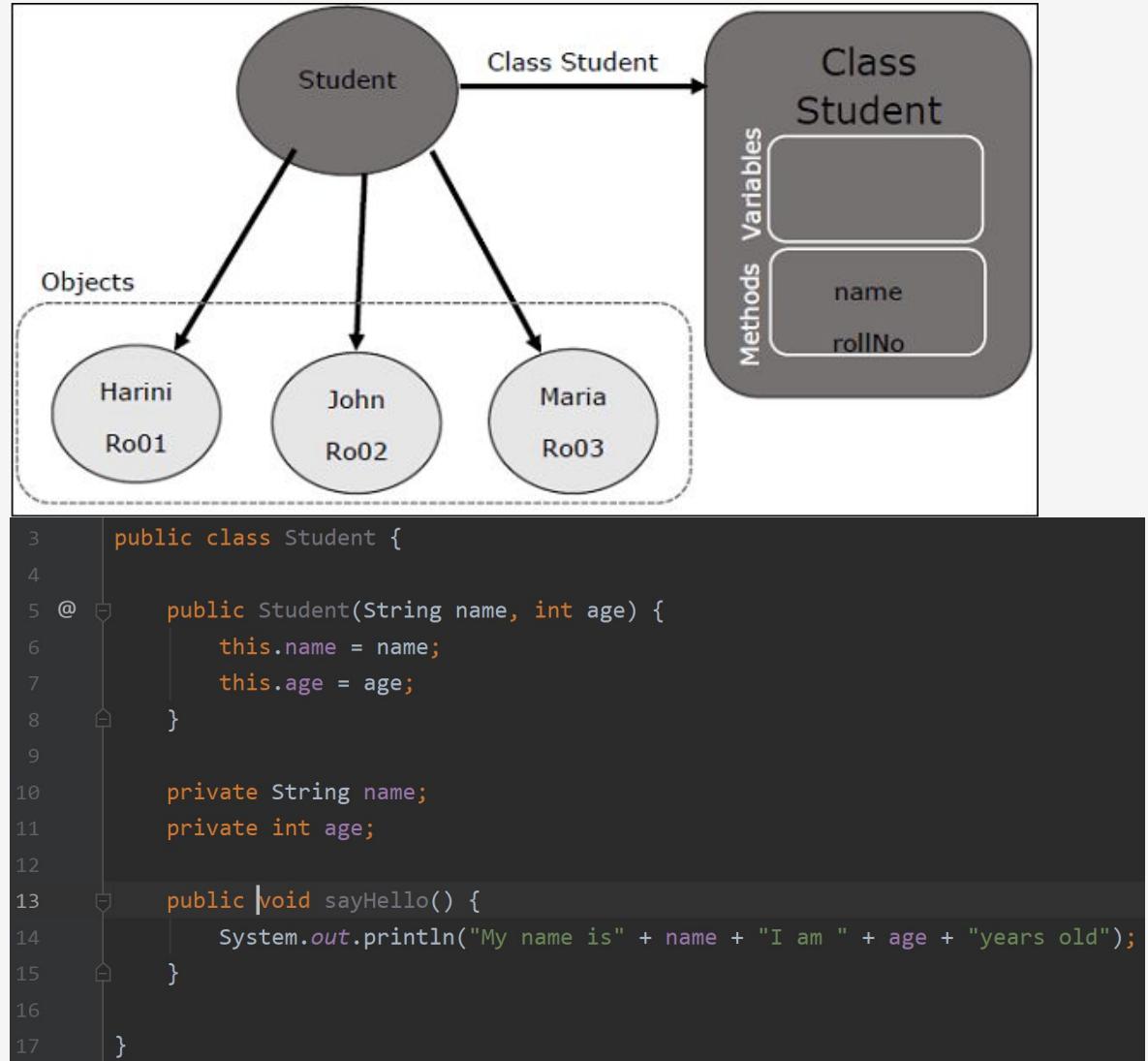




Klasa

Klasa – szablon, z którego tworzy się obiekty

Konstruując **obiekt**, tworzymy egzemplarz Klasy. Wszystko to, co piszemy w Javie, znajduje się w jakiejś klasie. Nazwa klasy odpowiada wycinkowi rzeczywistości, który reprezentuje



Autor: Przemek Baranowski

Prawa do korzystania z materiałów posiada Software Development Academy



Klasa

```
public class Person {  
    private String name;  
    private String gender;  
    private int age;  
  
    Person(String name, String gender, int age) {  
        this.name = name;  
        this.gender = gender;  
        this.age = age  
    }  
}
```

Modyfikator dostępu

Nazwa klasy

Typ zmiennych

Zmienne obiektowe

Zmienne przekazywane do konstruktora

Zawartość konstruktora

Konstruktor klasy
jednakowa nazwa, jak nazwa klasy



Obiekt

Obiekt – egzemplarz klasy Wszystkie obiekty charakteryzują się:

- Zachowaniem – co można z nimi zrobić, jakie metody można wywołać na ich rzecz?
- Stanem – jak obiekt reaguje na wywołane na jego rzecz metody?
- Tożsamością – jak odróżnić jeden obiekt od drugiego (tej samej klasy)?

Wszystkie obiekty tej samej klasy charakteryzują się jednakowym zachowaniem.

The screenshot shows an IDE interface with two main sections. The top section is a code editor displaying Java code. The bottom section is a terminal window showing the execution of the code and its output.

Code Editor (Top):

```
17 public static void main(String[] args) {  
18     Student student1 = new Student( name: "John", age: 23);  
19     Student student2 = new Student( name: "Andrew", age: 21);  
20     Student student3 = new Student( name: "Jane", age: 20);  
21  
22     student1.sayHello();  
23     student2.sayHello();  
24     student3.sayHello();  
25 }
```

Terminal Window (Bottom):

Run: **Student**

```
"C:\Program Files\Java\jdk-12.0.1\bin  
My name is John, I am 23 years old  
My name is Andrew, I am 21 years old  
My name is Jane, I am 20 years old  
Process finished with exit code 0
```





Metody

Metoda – procedura operująca na danych.

- zwraca informację o stanie obiektu,
- przeprowadza obliczenia na dostarczonych danych,
- zapisuje dostarczone dane...

```
17     public int getAge() {
18         return age;
19     }
20
21     public void setAge(int age) {
22         this.age = age;
23     }
24
25     public static void main(String[] args) {
26         Student student1 = new Student(name: "John", age: 23);
27         student1.sayHello();
28
29         student1.setAge(24);
30         student1.sayHello();
31     }

```

Run: Student ×

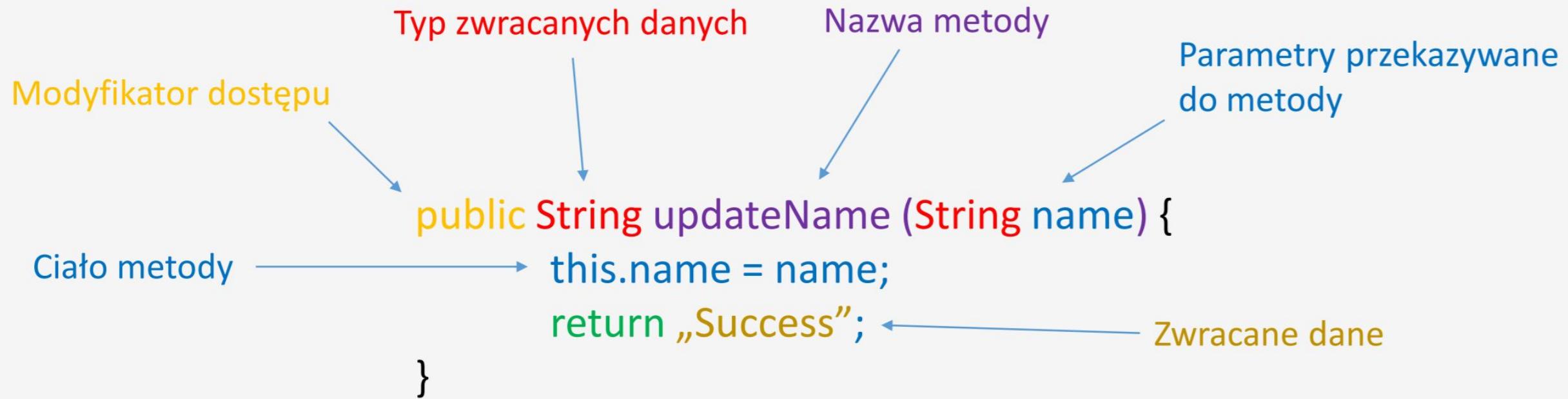
```
"C:\Program Files\Java\jdk-12.0.1\bin
My name is John, I am 23 years old
My name is John, I am 24 years old
Process finished with exit code 0
```

Autor: Przemek Baranowski

Prawa do korzystania z materiałów posiada Software Development Academy



Metody





Widoczność

Modyfikator	Klasa	Moduł	Subklasa	Świat
public	Tak	Tak	Tak	Tak
protected	Tak	Tak	Tak	Nie
<i>brak modyfikatora</i>	Tak	Tak	Nie	Nie
private	Tak	Nie	Nie	Nie

Modyfikatory dostępu określają czy inne klasy mogą używać poszczególnych pól lub wywoływać określone metody.

Wyróżniamy dwa poziomy dostępu:

- **top level** – public lub package-private (brak modyfikatora)
- **member level** – public, private, protected lub package-private (brak modyfikatora)

Podczas wyboru poziomu dostępu kierujemy się dwiema zasadami:

- używamy modyfikatora private chyba, że mamy naprawdę dobry powód, aby tego nie robić
- unikaj pól public za wyjątkiem stałych,



Zadanie

- 1) Stwórz klasę **Pies**.
 - a) Dodaj atrybuty rasa, wiek, płeć
 - b) Dodaj metody umożliwiające ustawienie wieku oraz pobranie wieku (getAge, setAge)
 - c) Dodaj konstruktor, który przyjmie wartości dla rasy i płci, a wiek ustawia na 0
 - d) Dodaj metodę powodującą wypisanie na ekran dźwięku wydawanego przez psa „Woof!”
 - e) Z poziomu metody main zaprezentuj działanie poszczególnych metod – utwórz obiekt klasy Pies, zmodyfikuj wiek, wyświetl parametry psa.
 - f) *Stwórz tablicę składającą się z 2 różnych psów, wypisz wartości ich atrybutów.



Zadanie

1) Stwórz klasę **Pokoj**.

- a) Dodaj atrybuty wysokość, szerokość, długość (typu double).
- b) Dodaj konstruktor, który przyjmie wszystkie wartości.
- c) Dodaj drugi konstruktor, który przyjmie szerokość i długość a wysokość ustawi na 2,4.
- d) Dodaj metody obliczające oraz zwracające pole powierzchni oraz objętość pokoju.
- e) Dodaj metody wyświetlające pole powierzchni oraz objętość pokoju.
- f) Z poziomu metody main zaprezentuj działanie poszczególnych metod.
- g) Stwórz tablicę pokoi. Wyświetl ich parametry

Zależności

- pakiety, importy
- do czego służy i jak działa maven
- tworzymy pierwszy projekt





Pakiet, importy

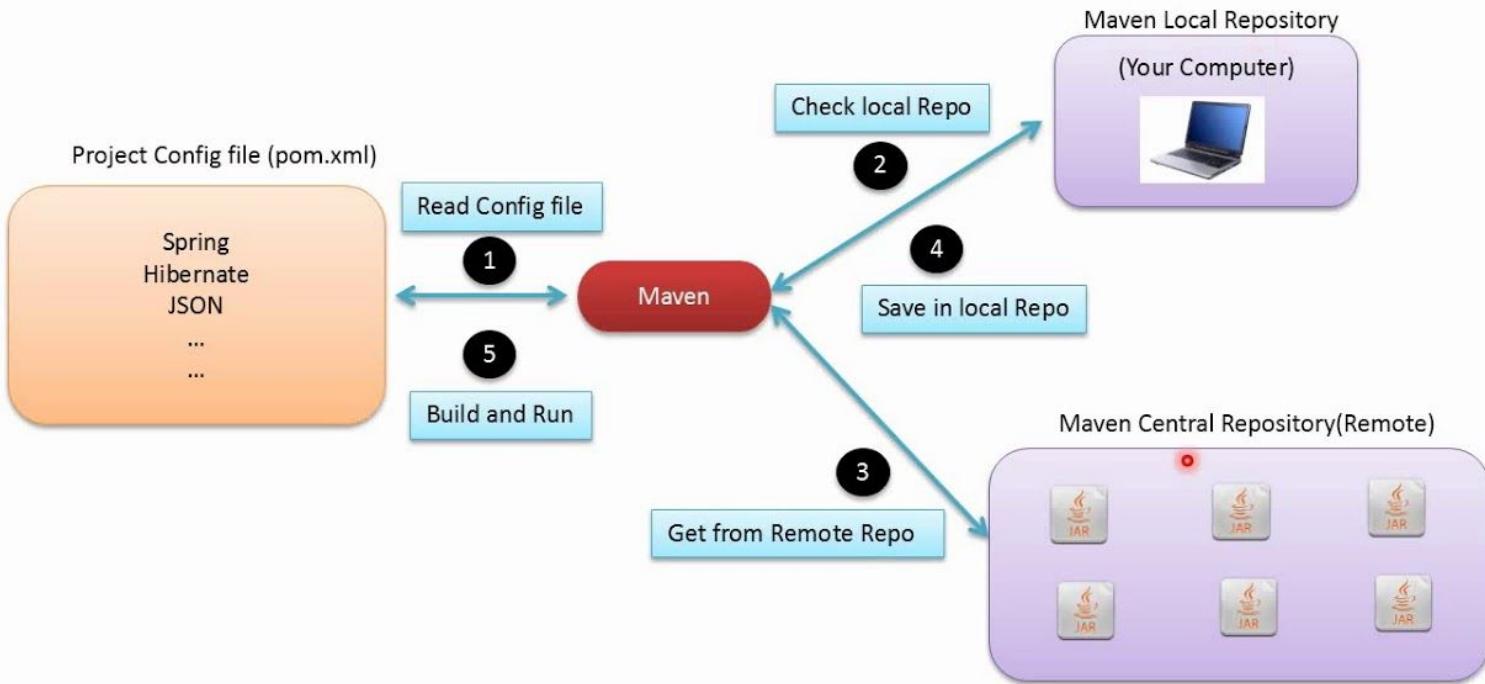
The screenshot shows a Java code editor with a dark theme. On the left, there is a file tree for a project named "javabasics [java-basics]". The tree includes a ".idea" folder, a "src" folder containing a "main" folder with a "java" folder. Inside "java" is a package named "pl.sda.javabasics.examples". This package contains several subfolders: "arrays", "conditions", "loops", and "objects". The "objects" folder contains a file named "Student.java". The "Student.java" file is currently selected in the editor. The code in the editor is:

```
1 package pl.sda.javabasics.examples.objects;
2
3 import java.util.ArrayList;
4 import java.util.Iterator;
5
6 public class Student {
7
8     @
9         public Student(String name, int age) {
10            this.name = name;
11            this.age = age;
12        }
13 }
```



Maven

- Apache **Maven** - narzędzie automatyzujące budowę oprogramowania na platformę Java.
- Poszczególne funkcje Mavena realizowane są poprzez **wtyczki**, które są automatycznie pobierane przy ich pierwszym wykorzystaniu.
- Plik określający sposób budowy aplikacji nosi nazwę **POM-u** (ang. Project Object Model).





Maven - cykl życia

- **clean** - usuwa pliki powstałe w czasie budowania projektu
- **validate** - sprawdzenie, czy projekt jest poprawny i czy wszystkie niezbędne informacje zostały określone
- **compile** - kod źródłowy jest komplikowany
- **test** - przeprowadzane są testy jednostkowe
- **package** - budowana jest paczka dystrybucyjna
- **integration-test** - zbudowany projekt umieszczany jest w środowisku testowym, gdzie przeprowadzane są testy integracyjne
- **verify** - sprawdzenie, czy paczka jest poprawna
- **install** - paczka umieszczana jest w repozytorium lokalnym - może być używana przez inne projekty jako zależność
- **deploy** - paczka umieszczana jest w repozytorium zdalnym (opublikowana)



Maven - zadanie

Stwórz nowy, mavenowy projekt w IntelliJ.

Autor: Przemek Baranowski

Prawa do korzystania z materiałów posiada Software Development Academy

Java - static

- pola statyczne
- metody statyczne
- klasy statyczne





Pola statyczne

```
class A {  
    ...  
    Deklaracja zmiennej statycznej → static int value;  
    ...  
}
```

Zmienna statyczna jest atrybutem klasy, NIE atrybutem obiektu.

- Wszystkie obiekty danej klasy współdzielą zmienne statyczne – modyfikacja zmiennej statycznej w jednym obiekcie będzie widoczna w pozostałych obiektach danej klasy.
- W celu skorzystania ze zmiennej statycznej nie musimy tworzyć obiektu klasy, która ją zawiera.
- Do zmiennej statycznej odnosimy się poprzez nazwę klasy, nie nazwę obiektu. Np. `A.value = ...;` zamiast `A a = new A(); a.value = ...; .`

Autor: Marek Bobcow

Prawa do korzystania z materiałów posiada Software Development Academy



Pola statyczne

```
3  public class ChildWithAgeLimit {  
4      private static final int MAX_AGE = 18;  
5  
6      @  
7          public ChildWithAgeLimit(String name, int age) {  
8              this.name = name;  
9              this.age = age;  
10         }  
11  
12         private String name;  
13         private int age;  
14  
15         public void sayHello() {  
16             if (age <= MAX_AGE) {  
17                 System.out.println("My name is " + name + ", I am " + age + " years old");  
18             } else {  
19                 System.out.println("My name is " + name + ", I am not child anymore");  
20             }  
21         }  
22     }
```



Metody statyczne

```
class A {  
    ...  
    Definicja metody statycznej → static void sumator() {  
        ...  
        }  
        ...  
    }
```

Metoda statyczna jest metodą klasy, NIE metodą obiektu.

- W celu skorzystania z metody statycznej nie musimy tworzyć obiektu klasy, która ją zawiera.
- Metoda statyczna może odwoływać się wyłącznie do innych metod statycznych i korzystać wyłącznie z atrybutów statycznych.
- Do metody statycznej odnosimy się poprzez nazwę klasy, nie nazwę obiektu.



Metody statyczne

```
3 ►  public class Calculator {  
4 @   static int sum(int a, int b) {  
5     return a + b;  
6 }  
7  
8 @   static int multiply(int a, int b) {  
9     return a * b;  
10 }  
11  
12 ►  public static void main(String[] args) {  
13     //nie jest tworzony żaden obiekt  
14     System.out.println("Suma: " + Calculator.sum( a: 2, b: 5));  
15     System.out.println("Iloczyn: " + Calculator.multiply( a: 2, b: 5));  
16 }  
17 }  
18 }
```

Run: Calculator X

```
"C:\Program Files\Java\jdk-12.0.1'  
Suma: 7  
Iloczyn: 10  
Process finished with exit code 0
```



Metody i pola statyczne - zadanie

1. Stwórz kalkulator korzystając z metod statycznych:
 - a. Stwórz najbardziej popularne metody: dodawanie, odejmowanie, mnożenie, dzielenie, reszta z dzielenia wewnątrz klasy „Kalkulator”
 - b. Stwórz mechanizm wyboru przez użytkownika, która metoda zostanie wykonana. Do zapisania definicji komend wprowadzanych przez użytkownika użyj pól statycznych
 - c. Zaprezentuj działanie poszczególnych metod



Klasy statyczne

Definicja klasy statycznej →

```
class A {  
    ...  
    static class B() {  
        ...  
        }  
        ...  
    }  
}
```

- W celu utworzenia obiektu klasy statycznej nie musimy tworzyć obiektu klasy, która ją zawiera
- Obiekt klasy statycznej tworzymy poprzez nazwę klasy która ją zawiera, nie nazwę obiektu. Np. ... new A.B(); zamiast A a = new A(); new a.B();
- Klasa statyczna może być zdefiniowana wyłącznie wewnątrz innej (niestatycznej) klasy.
- Klasa statyczna może się odwoływać wyłącznie do pól statycznych klasy nadzędnej.
- Klasa statyczna może posiadać zarówno metody i pola statyczne, jak i niestatyczne.

Autor: Marek Bobcow



Klasy statyczne

```
6  public class ShoppingCart {  
7  
8      static class Item {  
9          @  
10         public Item(String name, double price) {  
11             this.name = name;  
12             this.price = price;  
13         }  
14  
15         String name;  
16         double price;  
17     }  
18  
19     List<Item> items = new ArrayList<>();  
20  
21     void addItem(Item item) {  
22         items.add(item);  
23     }  
24  
25     int getItemCount() {  
26         return items.size();  
27     }  
28 }
```

```
28 ►  public static void main(String[] args) {  
29  
30     ShoppingCart cart = new ShoppingCart();  
31     cart.addItem(new ShoppingCart.Item( name: "headphones", price: 210.4));  
32     cart.addItem(new ShoppingCart.Item( name: "battery", price: 5.2));  
33     System.out.println("Found " + cart getItemCount() + " items in the cart");  
34 }
```



Klasy statyczne - zadanie

Rozbuduj przykład koszyka zakupów o funkcjonalności:

- dodaj metodę `getCheckoutAmount()` która oblicza cenę wszystkich produktów w koszyku
- dodaj metodę `clear()` która usuwa wszystkie elementy z koszyka
- *dodaj metodę `getCheckoutAmountWithDiscount()` która policzy należność z uwzględnieniem zniżki. Procentowa wartość zniżki powinna być zapisana jako pole statyczne
- *dodaj metodę, która wpisze paragon - listę przedmiotów i cenę końcową.

Klasa String

- podstawowe metody klasy String





Klasa String

```
3 ► public class StringExamples {  
4 ►     public static void main(String[] args) {  
5         String testString = "Lorem Ipsum Dolor";  
6         String otherString = "Sit Amen";  
7  
8         System.out.println("Długość tekstu: " + testString.length());  
9         System.out.println("Litera o indeksie 4 to: " + testString.charAt(4));  
10        System.out.println("Połączone napisy: " + testString.concat(otherString));  
11        System.out.println("Napis małymi literami: " + testString.toLowerCase());  
12        System.out.println("Napis wielkimi literami: " + testString.toUpperCase());  
13        System.out.println("Napis powtórzony 3 razy: "+testString.repeat(3));  
14        System.out.println("Napis kończy się literą r?: " +testString.endsWith("r"));  
15        System.out.println("Napis z zamienionym m na X: " +testString.replace( target: "m", replacement: "X"));  
16        System.out.println("Napis podzielony na słowa:");  
17        for (String word : testString.split( regex: " " )) {  
18            System.out.println(word);  
19        }  
20    }  
21 }
```



Klasa String - zadanie

Sprawdź jakie jeszcze metody są zaimplementowane w klasie String

Data i czas

- Java Time
- Formattery
- Strefy czasowe





- W języku Java mamy obecnie 3 implementacje daty:
`java.util.Date`, `java.sql.Date` i **java.time.LocalDate**
- Ostatnia implementacja pochodzi z najnowszej wersji Javy i jest obowiązująca. (JSR-310)
- **LocalDate** reprezentuje datę, **LocalTime** reprezentuje czas,
LocalDateTime posiada jedno i drugie
- Instant - bez strefy czasowej



Data i czas

```
5 ► public class DateTimeExamples {  
6 ►     public static void main(String[] args) {  
7         LocalDateTime localDateTime = LocalDateTime.now();  
8         LocalDate localDate = LocalDate.now();  
9         LocalTime localTime = LocalTime.now();  
10        Instant instant = Instant.now();  
11  
12        Clock clock = Clock.system(ZoneId.of("America/Denver"));  
13        LocalDateTime localDateTimeOtherTz = LocalDateTime.now(clock);  
14        LocalDate localDateOtherTz = LocalDate.now(clock);  
15        LocalTime localTimeOtherTz = LocalTime.now(clock);  
16        Instant instantOtherTz = Instant.now(clock);  
17  
18        LocalDateTime createdDateTime = LocalDateTime.of( year: 2019, month: 12, dayOfMonth: 07, hour: 12, minute: 11, second: 43);  
19        LocalDate createdLocalDate = LocalDate.of( year: 2019, month: 12, dayOfMonth: 8);  
20        LocalTime createdTime = LocalTime.of( hour: 12, minute: 11, second: 32);  
21  
22        LocalDateTime parsedLocalDateTime = LocalDateTime.parse("2007-12-03T10:15:30");  
23        LocalDate parsedLocalDate = LocalDate.parse("2007-12-03");  
24        LocalTime parsedLocalTime = LocalTime.parse("10:15:30");  
25  
26  
27        System.out.println(localDateTime.toString());  
28        System.out.println(localTime.toString());|  
29    }  
30 }
```

Autor: Przemek Baranowski

Prawa do korzystania z materiałów posiada Software Development Academy



Data i czas - formatowanie

```
7 ►  public class DateTimeFormattersExample {  
8 ►    □  public static void main(String[] args) {  
9  
10       LocalDateTime now = LocalDateTime.now();  
11  
12       DateTimeFormatter dateFormatter = DateTimeFormatter.ISO_DATE;  
13       System.out.println("Std date formatter: " + dateFormatter.format(now));  
14       DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ISO_DATE_TIME;  
15       System.out.println("Std date time formatter: " + dateTimeFormatter.format(now));  
16  
17       DateTimeFormatter custom = DateTimeFormatter.ofPattern("yyyy-MM-dd");  
18       System.out.println("Custom formatter: " + custom.format(now));  
19  
20       DateTimeFormatter longStyle = DateTimeFormatter.ofLocalizedDate(FormatStyle.FULL);  
21       System.out.println("Full style date time formatter: " + longStyle.format(now));  
22       DateTimeFormatter shortStyle = DateTimeFormatter.ofLocalizedDate(FormatStyle.SHORT);  
23       System.out.println("Short style date time formatter: " + shortStyle.format(now));  
24  
25       System.out.println("formatted & parsed: " + shortStyle.parse(shortStyle.format(now)));  
26  
27    □ }  
28 }
```

Autor: Przemek Baranowski

Prawa do korzystania z materiałów posiada Software Development Academy



Data i czas - zadania

1. Wypisz bieżącą datę, czas oraz datę i czas korzystając z odpowiednich formatterów.
2. Punkt 1, ale dla strefy czasowej America/Detroit
3. * wczytaj datę podaną przez użytkownika w formacie 2019-12-07 i wyświetl jako 2019.12.07
4. * Dodaj do podanej przez użytkownika daty 10 dni i wyświetl.

Wyrażenia regularne

- zakresy
- grupy
- walidacja
- wyszukiwanie





Regex - Kwantyfikatory

Kwantyfikator	Znaczenie	Przykład	Przykład dopasowuje
*	Zero lub więcej wystąpień	a^*b	ab, b, aab, aaaaaab, aaab (i podobne)
+	Jedno lub więcej wystąpień	$a+b$	ab, aab, aaaaaaaab, aab (i podobne)
?	Zero lub jedno wystąpienie	$a?b$	ab, b
{n,m}	Co najmniej n i maksymalnie m wystąpień	$a\{1,4\}b$	ab, aab, aaab, aaaab
{n,}	Co najmniej n wystąpień	$a\{3,\}b$	aaab, aaaab aaaaab (i podobne)
{,n}	Maksymalnie n wystąpień	$a\{,3\}b$	b, ab, aab, aaab
{n}	Dokładnie n wystąpień	$a\{3\}b$	aaab



Regex - zakresy

Wyrażenie	Opis
[abcde]	Jedna z liter: a, b, c, d lub e
[a-zA-Z]	Jedna z liter od a do Z mała lub duża
[a-c3-5]	Litera od a do c lub cyfra od 3 do 5
[a-c14-7]	Litera od a do c lub cyfra 1 lub cyfra od 4 do 7
[abc\[\\]]	Litera a lub b lub c lub nawias kwadratowy (dlaczego dodaliśmy też odwrócone ukośniki, czytaj dalej)
[.]	Dowolna litera (czytaj dalej)



Regex - grupy

Wyrażenie	Opis
$a(bcd)^*$	litera a oraz ciąg bcd zero lub więcej razy
$a(b(cd)?)^+$	litera a, a następnie jedno lub więcej powtórzeń b lub bcd



REGULAR EXPRESSION

1 match, 215 steps (~1ms)

```
/ ([a-zA-Z]*)\s([a-zA-Z]*).*age:([0-9]*).*id:([0-9]*).*phone: /  
([0-9].*-?[0-9].*-?[0-9].*-?[0-9].*-?[0-9].*)
```

TEST STRING

[SWITCH TO UNIT TESTS ▶](#)

```
John Doe, age:27 id:123123456, phone:555-123-123
```



Regex - wyszukiwanie / parsowanie

```
6 ► public class RegexExample {  
7 ►     public static void main(String[] args) {  
8         String text = "John Doe, age:27 id:123123456, phone:555-123-123";  
9         Pattern pattern = Pattern.compile("( [a-zA-Z]* )\\s( [a-zA-Z]* ).*age:( [0-9]* ).*id:( [0-9]* )" +  
10            ".*phone:( [0-9].*-?[0-9].*-?[0-9].*-?[0-9].*-?[0-9].* )");  
11  
12         Matcher matcher = pattern.matcher(text);  
13         System.out.println("Found anything? " + matcher.find());  
14         System.out.println("Name: " + matcher.group(1));  
15         System.out.println("Surname: " + matcher.group(2));  
16         System.out.println("Age: " + matcher.group(3));  
17         System.out.println("ID: " + matcher.group(4));  
18         System.out.println("Phone no : " + matcher.group(5));  
19     }  
20 }  
21 }
```



Regex - walidacja

```
5 ► public class RegexValidationExample {  
6 ►     public static void main(String[] args) {  
7         Pattern emailPattern = Pattern.compile("[A-Z0-9a-zA-Z_.%+-]+@[A-Za-z0-9.-]+.[A-Za-z]{2,64}");  
8         validate(emailPattern, text: "valid@email.com");  
9         validate(emailPattern, text: "@nonvalidemail.com");  
10  
11        Pattern peselPattern = Pattern.compile("[0-9]{11}");  
12        validate(peselPattern, text: "90120812376");  
13        validate(peselPattern, text: "9012081237");  
14        validate(peselPattern, text: "9012081237a");  
15    }  
16  
17 @     private static void validate(Pattern pattern, String text) {  
18     System.out.println(text + " matches pattern: " + pattern.matcher(text).matches());  
19 }  
20 }  
21 }
```



Regex - zadania

- Napisz validator numeru NIP podanego przez użytkownika. założmy że “-” są opcjonalne.
- Z adresu podanego przez użytkownika wydziel nazwę ulicy i numer domu, np. ul.Graniczna 106

jShell

```
Microsoft Windows [Version 10.0.18362.476]
(c) 2019 Microsoft Corporation. Wszelkie prawa zastrzeżone.
```

```
C:\Users\P>jshell
| Welcome to JShell -- Version 12.0.1
| For an introduction type: /help intro
```

```
jshell>
```





Istnieje kilka dziedzin, w których **JShell** może nam pomóc:

- **nauka** – bardzo dobre narzędzie do nauki programowania dla juniorów
- **sprawdzanie corner case'ów** – jeśli zdarzyło wam się przygotowywać do certyfikacji z Javy, bardzo często pojawiają się tam językowe “smaczki”, które w łatwy sposób możemy wykonać w powłoce
- **prototypowanie** – do prototypowania nowych featurów
- **esperimentowanie** – na przykład z nowymi bibliotekami



Jshell - zmienne i metody

Wiersz polecenia - jshell

Microsoft Windows [Version 10.0.18362.476]
(c) 2019 Microsoft Corporation. Wszelkie prawa zastrzeżone.

```
C:\Users\P>jshell
| Welcome to JShell -- Version 12.0.1
| For an introduction type: /help intro
```

```
jshell> void printUpperCase(String text){
...>     System.out.println(text.toUpperCase());
...> }
| created method printUpperCase(String)
```

```
jshell> String napis = "Napis testowy";
napis ==> "Napis testowy"
```

```
jshell> printUpperCase(napis);
NAPIS TESTOWY
```

```
jshell>
```



jshell - klasy

```
Wiersz polecenia - jshell
Microsoft Windows [Version 10.0.18362.476]
(c) 2019 Microsoft Corporation. Wszelkie prawa zastrzeżone.

C:\Users\P>jshell
| Welcome to JShell -- Version 12.0.1
| For an introduction type: /help intro

jshell> public class Printer{
...>     private final String msg;
...>     public Printer(String text){
...>         this.msg = text;
...>     }
...>     public void print(){
...>         System.out.println(msg);
...>     }
...> }
| created class Printer

jshell> Printer p = new Printer("napis");
p ==> Printer@343f4d3d

jshell> p.print();
napis

jshell>
```



jshell - skrypt

```
script.jsh
1 System.out.println("Hello World")
2 /exit
```

```
C:\Users\P\Desktop>jshell script.jsh
Hello World
```

```
C:\Users\P\Desktop>
```



Napisz prosty kalkulator w Jshell (dodawanie i mnożenie) i sprawdź czy działa