

Podstawy Java SE



Hello

Tomasz Lisowski

Software developer, JIT Solutions

IT trainer

Agenda

- powtórka
- instrukcje sterujące
- pobranie z klawiatury
- równość obiektów
- operacje na typach tekstowych
- OOP



Wprowadzenie

klasa

- podstawowy element składowy aplikacji
- typ danych
- szablon - konkretna definicja pewnego 'bytu'
- zawiera pola i metody

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("infoShareAcademy - Java SE");  
    }  
}
```

klasa - pole

- dana cecha naszej klasy
- może reprezentować dowolny typ (klasę)
- może być ich wiele lub wcale

```
public class Car {  
    public String name;  
    public int maxSpeed;  
}
```

modyfikator dostępu →

typ danych →

nazwa (dowolna) →

klasa - metoda

- funkcjonalność naszej klasy (tu logika - nie piszemy kodu poza metodami!)
- zwracają jakiś typ danych (lub nic - wtedy 'zwracamy' *void*)
- mogą przyjmować parametry

modyfikator
dostępu

```
public void method1() {  
    System.out.println("Ta metoda nie zwraca nic!");  
}  
  
public int getNumberTwo() {  
    return 2; //ta metoda zwraca liczbę całkowitą 2  
}  
  
public int sum(int a, int b) {  
    return a + b; //ta metoda zwraca sumę dwóch parametrów  
}
```

typ danych

nazwa
(dowolna)

klasa - obiekt

wywołanie pól i metod

```
Car myCar = new Car();  
//gdy pole name jest public  
System.out.println(myCar.name);  
//gdy pole name jest private  
System.out.println(myCar.getName());  
//wywołanie metody printName() z klasy Car, na obiekcie myCar  
myCar.printName();
```


klasa vs obiekt



obiekt

- instancja klasy
- konkretny obiekt na podstawie definicji klasy

```
public class Car {  
    public String name;  
    public int maxSpeed;  
}
```

```
Car myCar = new Car();
```

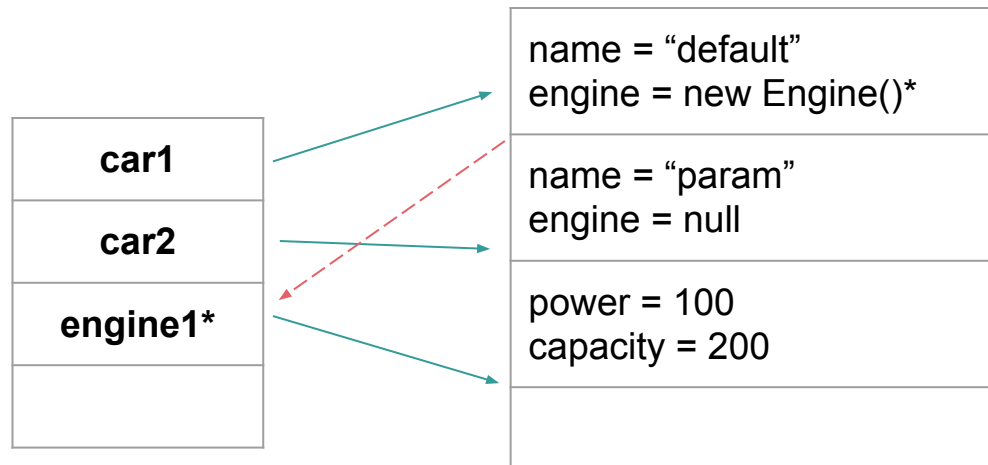
konstruktor

przypisanie do pola name
(pole klasy) wartości
parametru

```
public class Car {  
    public String name;  
    public int maxSpeed;  
  
    public Car() {  
        name = "default name";  
        maxSpeed = 150;  
    }  
  
    public Car(String name) {  
        this.name = name;  
    }  
}
```

pamięć

```
Car car1 = new Car();  
Car car2 = new Car("param");
```



metody statyczne

- oznaczone słowem kluczowym ***static***
- można je wywołać bezpośrednio na klasie
- nie wymagają stworzenia instancji obiektu

getter i setter

```
private int number;

public int getNumber() {
    return number;
}

public void setNumber(int number) {
    this.number = number;
}
```

Typy danych

co to jest?

- *wszystko jest obiektem*
- typy reprezentują różne wartości, przechowywane w zmiennych
- np. tekstowe, liczbowe, zmiennoprzecinkowe, logiczne
- różne formaty dat
- każda klasa jest typem

Typy proste

- typy proste (*primitive*) nie są instancjami obiektów (wyjątek!)
- reprezentują podstawowe typy danych
- zawsze mają jakąś wartość

```
int liczbaCalkowita;  
long duzaLiczbaCalkowita;  
double liczbaZmiennoPrzecinkowa; //64bit  
float kolejnaLiczbaZmiennoPrzecinkowa; //32bit  
boolean prawdaFalsz;  
char znak;
```


Typy obiektowe

- klasy - możemy tworzyć swoje typy obiektowe
- mogą mieć dowolne zachowanie (metody)
- mogą nie mieć wartości -> NULL

```
Integer liczbaCalkowita;  
Long duzaLiczbaCalkowita;  
Double liczbaZmienniePrzecinkowa;  
Float kolejnaLiczbaZmienniePrzecinkowa;  
Boolean prawdaFalsz;  
String napis;
```

Autoboxing

- automatyczna zmiana typów prostych na obiektywne i odwrotnie

```
Integer integerFromInt = 12;  
Integer integerFromNew = new Integer( value: 12);
```

```
public void methodInt(int i) { }  
public void methodInteger(Integer i) { }
```

```
methodInt(integerFromNew);  
methodInteger(integerFromInt);
```

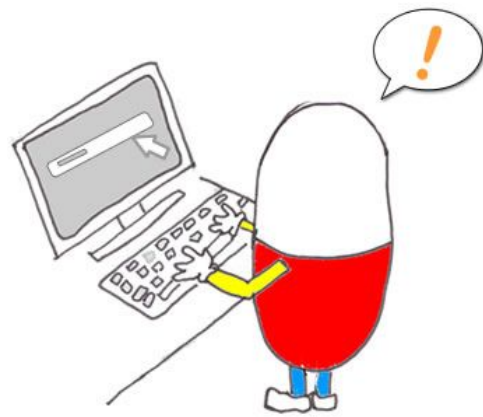
Rzutowanie

```
int liczbaA = 10;  
int liczbaB = 3;  
//wynik dzielenia nie jest liczbą całkowitą  
  
double wynikInt = liczbaA / liczbaB;  
// zmienna wynik = 3.0
```

Typy danych

ćwiczenie 7

- utwórz klasę *Calculator*
- utwórz w niej metodę *divide(Integer a, Integer b)*
- metoda powinna poprawnie podzielić liczbę *a* przez *b* i zwrócić wynik
- w metodzie *main()* wypisz wyniki kilku operacji *divide* dla różnych parametrów



Instrukcje sterujące

if else

- podstawowa operacja – instrukcja wyboru
- if = jeżeli
- *jeżeli warunek jest spełniony, to wykonaj instrukcje*

```
double wynik = liczbaA/liczbaB;  
  
if (wynik > 0) {  
    return "Liczba dodatnia";  
}
```

if else

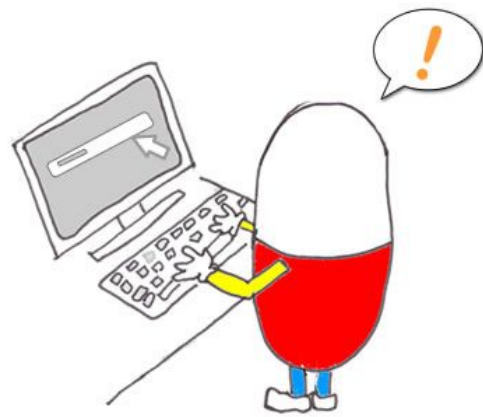
- warunek *if* można łączyć z *else*
- *else* wykonywane gdy pierwszy warunek nie jest spełniony
- można zagnieżdżać i/lub łączyć instrukcje *if* - *else*

```
if (wynik > 0) {  
    return "Liczba dodatnia";  
}  
else if (wynik == 0) {  
    return "Liczba 0";  
}  
else {  
    return "Liczba ujemna";  
}
```

Instrukcje sterujące

ćwiczenie 8

- stwórz obiekt *car1* z wartością *name* = “*car1*”, *maxSpeed* = 100;
- stwórz obiekt *car2* z wartością *name* = “*car2*”, *maxSpeed* = 200;
- stwórz warunek, który wypisze nazwę pojazdu o większej wartości pola *maxSpeed*
- *stwórz warunek, który obiektowi o większej wartości pola *maxSpeed* przypisze nową wartość pola *name* => “faster car”
- *wypisz wartość *name* obydwu obiektów



Potrójny operator *if*

- jednolinowa operacja zastępująca ***if-else***
- *(warunek logiczny) ? pierwsze_wyrażenie : drugie_wyrażenie*

```
int c = (a > b) ? a : b;
```

```
if (a > b) {  
    c = a;  
} else {  
    c = b;  
}
```

switch

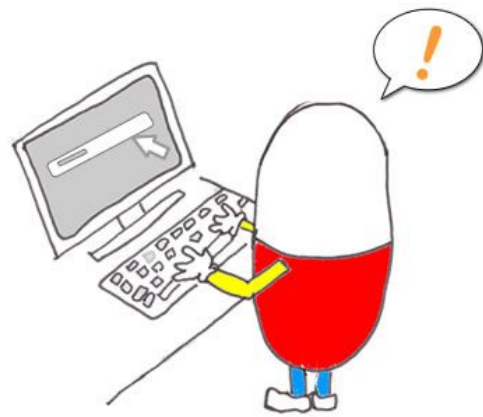
- “wielowarunkowy if”
- switch pobiera parametr i sprawdza dowolną liczbę warunków

```
switch(liczba){  
    case 1:  
        jakieś_instrukcje_1;  
        break;  
    case 2:  
        jakieś_instrukcje_2;  
        break;  
    ...  
    default:  
        instrukcje, gdy nie znaleziono żadnego pasującego przypadku  
}
```

Instrukcje sterujące

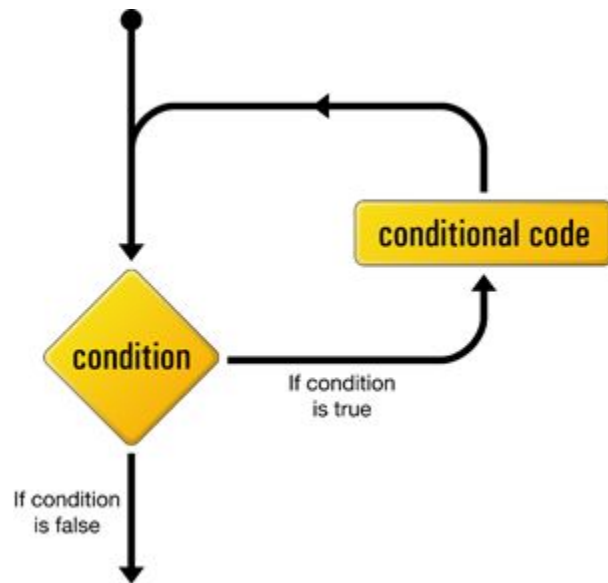
ćwiczenie 9

- stwórz metodę, która pobiera liczbę całkowitą
- wykorzystaj instrukcję *switch* do sprawdzenia, czy liczba z parametru jest parzysta



pętle

- podstawowa operacja – cykliczne wykonanie danych instrukcji
- niewiadoma ilość wykonań
- .. lub ściśle określona
- można przerwać lub pominąć dany obieg



while

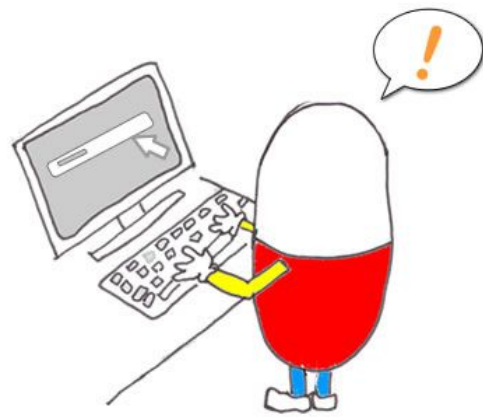
- wykorzystywana, gdy nie znamy ilość obiegów pętli
- .. ale znamy warunek jej zakończenia
- pętla while może wykonać się nieskończenie wiele razy
- albo wcale, gdy warunek już na starcie nie jest spełniony

```
int liczba = -5;  
while(liczba < 0) {  
    liczba++; //liczba = liczba + 1;  
}
```

Instrukcje sterujące

ćwiczenie 10

- stwórz zmienną liczbową o wartości ujemnej
- stwórz pętlę while, która “kręci się” dopóki powyższy parametr jest mniejszy od 0
- wewnątrz pętli wypisz wartość zmiennej, a następnie zwiększ ją o 1



do..while

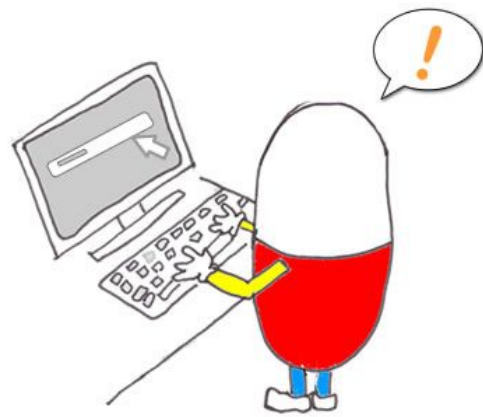
- inna wersja pętli *while*
- pętla *do..while* zawsze wykona się co najmniej jeden raz

```
int liczba = 5;  
do {  
    liczba++; //liczba = liczba + 1;  
} while(liczba < 0);
```

Instrukcje sterujące

ćwiczenie 11

- zmodyfikuj poprzednie zadanie:
 - odwróć warunek (pętla “kręci się” dopóki parametr jest większy od 0)
 - zastosuj pętlę do..while



for

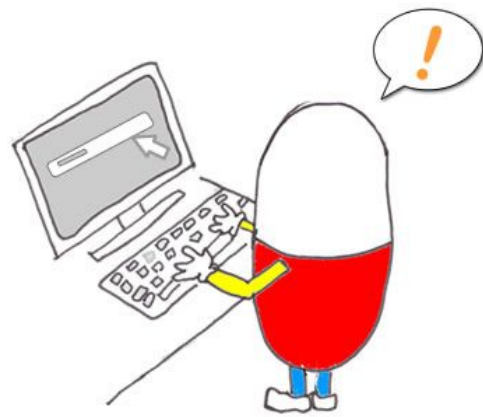
- zazwyczaj znamy liczbę iteracji w pętli
- 3 parametry:
 - wyrażenie początkowe → np. **int i = 0**
 - warunek → np. **i < 5**
 - modyfikator → np. **i++**

```
for (int i = 0; i < 5; i++) {  
    System.out.println("i: " + i);  
}
```

Instrukcje sterujące

ćwiczenie 12

- stwórz zmienną liczbową o wartości 10
- stwórz pętlę *for*, którą “kręci się” od 0
- warunkiem jest licznik pętli mniejszy od utworzonej zmiennej
- z każdym obiegiem pętli zwiększ licznik pętli o 1
- wypisz wartość licznika pętli w każdym obiegu



break - continue

- instrukcje manipulujące działaniem pętli
- **break**
 - przerwanie pętli
- **continue**
 - pominięcie danej iteracji

```
int liczba = -5;
while(liczba < 0) {
    if (liczba == 2) {
        continue;
    }

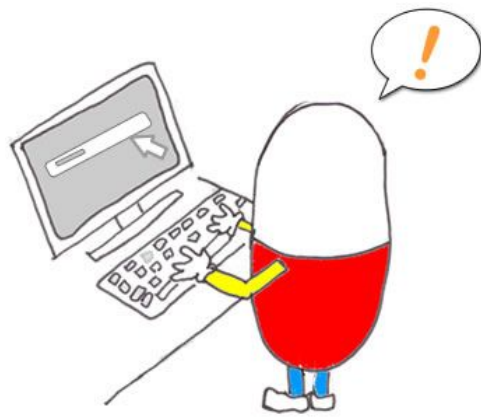
    if (liczba == 3) {
        break;
    }

    liczba++; //liczba = liczba + 1;
}
```

Instrukcje sterujące

ćwiczenie 13

- napisz metodę, przyjmującą jeden parametr typu int
- w metodzie napisz pętlę iterującą od 0 do wartości tego parametru
- wypisz na konsolę każdą liczbę nieparzystą
- pomiń każdą liczbę parzystą
- przerwij pętlę, jeśli liczba jest podzielna bez reszty przez 11



pętle



```
//Grab odd numbers from array
for(int i=0;i<Array.Length;i++)
{
    if(i == 1){
        Console.Write(i);
    }
    if(i == 3){
        Console.Write(i);
    }
    if(i == 5){
        Console.Write(i);
    }
    if(i == 7){
        Console.Write(i);
    }
    if(i == 9){
        Console.Write(i);
    }
    if(i == 11){
        Console.Write(i);
    }
    if(i == 13){
        Console.Write(i);
    }
    if(i == 15){
        Console.Write(i);
    }
}
```

Pobranie danych z klawiatury

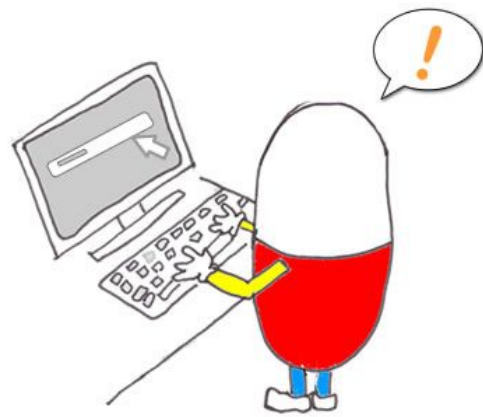
Scanner

- podstawowe pobranie danych od użytkownika
- obiekt korzysta ze strumienia wejściowego:
- **Scanner scanner = new Scanner(System.in);**
- popularne metody:
 - ***nextLine()*** - zwraca *String*
 - ***nextInt()*** - zwraca *int*
 - ***nextDouble()*** - zwraca *double*

Pobranie danych z klawiatury

ćwiczenie 14

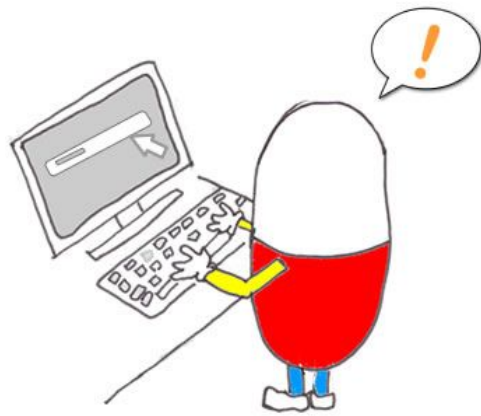
- stwórz dwie zmienne typu Double
 - pobierz je za pomocą klasy Scanner
 - dodaj je do siebie
 - wyświetl wynik
-
- ile wynosi $0,1 + 0,2$?



Pobranie danych z klawiatury

ćwiczenie 15

- pobierz za pomocą klasy Scanner dwie wartości:
 - tekst (*name*)
 - liczbę (*engine.power*)
- przypisz je do pól obiektu klasy Car
- wypisz wartości tego obiektu



Czy obiekty są równe?

== VS equals

- instrukcje porównania
- == porównuje **referencję** (przestrzeń pamięci)
- *equals()* porównuje **wartość** dwóch pól
- *domyślna implementacja *equals()* z klasy *Object*

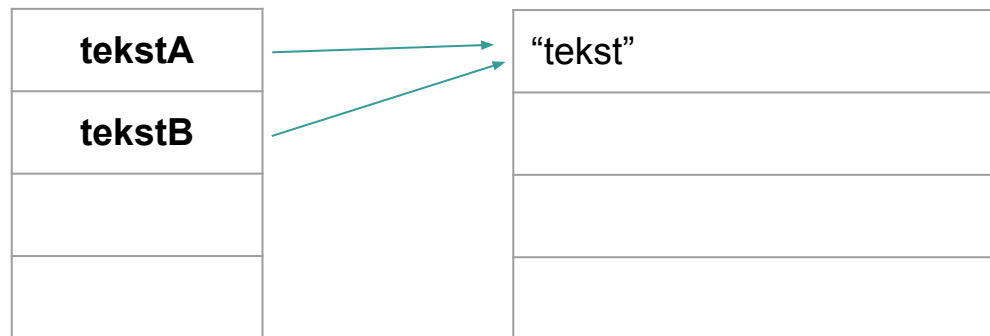
== VS equals

```
String tekstA = "tekst";  
String tekstB = "tekst";
```

```
if (tekstA == tekstB) {  
    System.out.println("warunek == prawdziwy");  
}
```

```
if (tekstA.equals(tekstB)) {  
    System.out.println("warunek equals prawdziwy");  
}
```

== VS equals



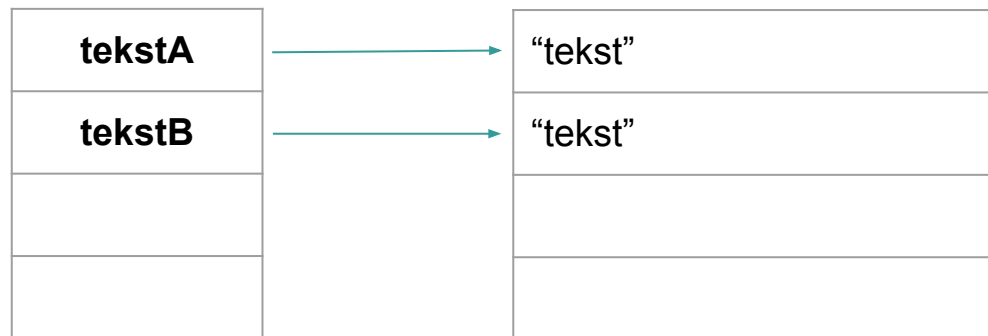
== VS equals

```
String tekstA = new String( original: "tekst");  
String tekstB = new String( original: "tekst");
```

```
if (tekstA == tekstB) {  
    System.out.println("warunek == prawdziwy");  
}
```

```
if (tekstA.equals(tekstB)) {  
    System.out.println("warunek equals prawdziwy");  
}
```

== VS equals



== VS equals

- **equals()** to metoda klasy Object

jeśli obiekty są równe to muszą mieć ten sam hashCode

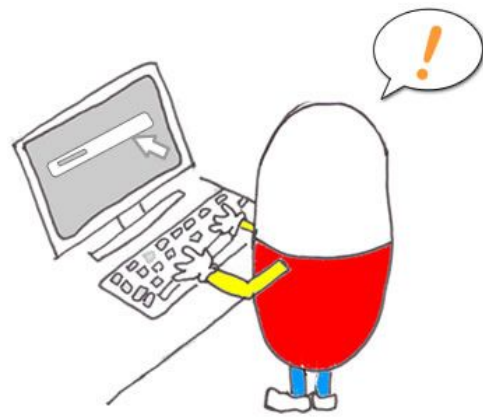
jeśli obiekty mają ten sam hashCode to nie muszą być równe

- nadpisanie metody **hashCode()**
- kontrakt **hashCode()** ↔ **equals()**

Równość obiektów

ćwiczenie 16

- stwórz 2 stringi o takiej samej wartości
- porównaj je za pomocą instrukcji if i operatorów:
 - ==
 - equals()
- wypisz ich hashCode



Operacje na typach tekstowych

String

tworzenie

```
//przypisanie wartości (by literal)  
String s1 = "java";
```

```
//tablica pojedynczych znaków  
char[] chars = {'i', 's', 'a'};
```

```
//zamiana tablicy znaków na String  
String s2 = new String(chars);
```

```
//użycie słowa kluczowego new  
String s3 = new String( original: "infoShare");
```

String

porównanie

```
String s1 = "INFOShare";  
String s2 = new String( original: "infoShare");  
  
System.out.println(s1.equals(s2));  
System.out.println(s1.equalsIgnoreCase(s2));  
System.out.println(s1 == s2);  
System.out.println(s1.compareTo(s2));  
System.out.println(s1.compareToIgnoreCase(s2));  
System.out.println(s2.compareTo(s1));
```

StringBuilder

- specjalny builder do tworzenia Stringów
- kolejne Stringi dodajemy za pomocą metody *append()*
- udostępnia metody do manipulacji budowanego tekstu
- wynik możemy zapisać do zmiennej String (metoda *toString()*)

StringBuilder

```
String s1 = "info" + "Share" + "Academy";
```

```
StringBuilder stringBuilder = new StringBuilder();  
stringBuilder.append("info");  
stringBuilder.append("Share");  
stringBuilder.append("Academy");
```

```
String s2 = stringBuilder.toString();  
String s3 = stringBuilder.reverse().toString();
```

String

metody

```
String s = " InfoShare ";

s.toUpperCase(); // INFOSHARE
s.toLowerCase(); // infoSHARE
s.trim(); //InfoShare
s.startsWith("In"); //false
s.endsWith("are"); //false
s.charAt(5); //o
s.length(); //13
String.valueOf(10); //10
s.replace(target: "Share", replacement: "Academy"); // InfoAcademy
```

String

metody

```
String s = "string:separate:by:colons";  
String[] sArray = s.split(regex: ":");  
  
System.out.println(sArray.length);  
for (int i = 0; i < sArray.length; i++) {  
    System.out.println(sArray[i]);  
}
```

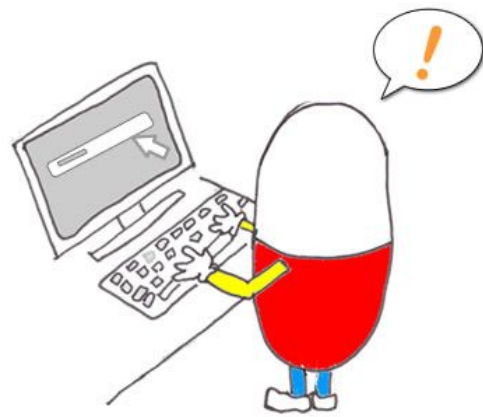

String

ćwiczenie 4* - z 20.03

- stwórz metodę `addString(String s)`
- parametr `s` jest tekstem w postaci `liczba;liczba`
- metoda powinna wywołać `split()` na obiekcie `s`
- następnie każdy z elementów zamienić na liczbę i dodać do siebie wszystkie wartości
- wypisz wynik takiej operacji

np.

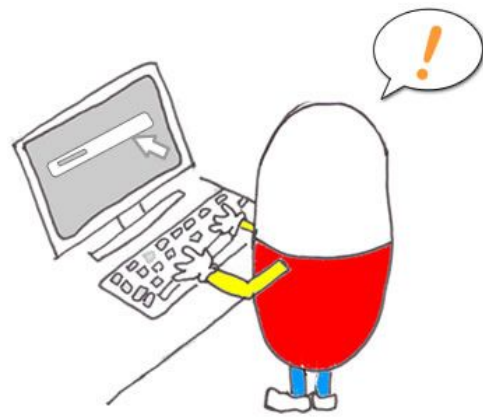
`addString("2;4;1")` -> zwraca 7



String

ćwiczenie 17

- pobierz z klawiatury dowolny tekst (jedna linia);
- w tekście mogą być kropki, ale nie muszą
- stwórz metodę, która zwróci ilość kropek w tekście
- wypisz na konsolę wyliczoną ilość kropek



Modyfikatory dostępu

pakiety

- klasy pogrupowane w pakiety
- struktura hierarchiczna
- pakiety → katalogi, klasy → pliki
- implementacja klasy znajduje się w jakimś pakiecie
- informuje o tym instrukcja package

np. klasa znajduje się w pakiecie ***infoshareacademy***, który znajduje się w pakiecie ***com***

```
package com.infoshareacademy;
```

modyfikatory dostępu

- słowa kluczowe określające poziom dostępności pól/metod innym klasom
- ***public*** – dostęp do elementu dla wszystkich klas
- ***protected*** – dostęp tylko dla klas dziedziczących lub z tego samego pakietu
- ***private*** – brak widoczności elementów poza klasą
- default – dostęp pakietowy, nie istnieje takie słowo kluczowe
package-private -> *publiczne w pakiecie, prywatne na zewnątrz*
- dobra praktyka – wszystkie pola prywatne

Enum

- typ wyliczeniowy
- konkretny (ograniczony) zbiór możliwych wartości

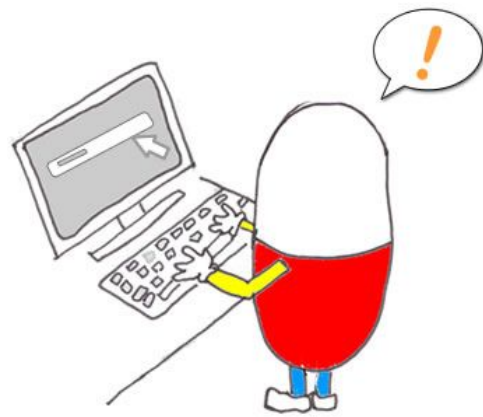
```
public enum Colour {  
    RED,  
    GREEN,  
    BLUE  
}
```

```
Colour myColour = Colour.GREEN;
```

Enum

ćwiczenie 18

- stwórz typ wyliczeniowy Colour
- stwórz w nim kilka wartości
- dodaj pole typu Colour w klasie Car
- przetestuj program



Enum - pola

- możliwość dodania parametru dla enum
- stworzenie pola
- oraz konstruktora

```
ACE("as"),  
KING("król"),  
QUEEN("dama"),  
JACK("walet");
```

```
private String name;
```

```
Rank(String name) {  
    this.name = name;  
}
```

```
public String getName() {  
    return name;  
}
```


Podstawy JSE

materiały

- <https://javastart.pl/baza-wiedzy/darmowy-kurs-java/podstawy-jezyka>
- <https://www.tutorialspoint.com/java/>
- <https://docs.oracle.com/javase/tutorial/>
- <https://www.samouczekprogramisty.pl/typ-wyliczeniowy-w-jezyku-java/>

Pytania?





Thanks!

Q&A

tomasz.lisowski@protonmail.ch