

# Typy danych i zmienne



# Agenda

---

- Co to są dane i jakie typy danych wyróżniamy ?
- Co to są zmienne i do czego służą ?
- Jak korzystać ze zmiennych ?

# Dane

- Codziennie słyszymy dużo informacji o ochronie danych, danych firmowych, danych do przelewu... itd. Ale czym są *dane*?

Dane to informacje które przekazujemy innym osobom, zapisujemy na kartce czy mówimy lekarzowi... Wszystko co ma jakąś wartość.

Danymi są informacje o Twoim zatrudnieniu, stanie zdrowia, ilości pupili które posiadasz w domu czy temperaturze pieczenia kurczaka.

- *Dane w informatyce?*

Wszystkie aplikacje, bez względu w jakim języku są napisane, operują na *danych*. Przetwarzają je, zmieniają, przechowują, analizują...



# Typy danych

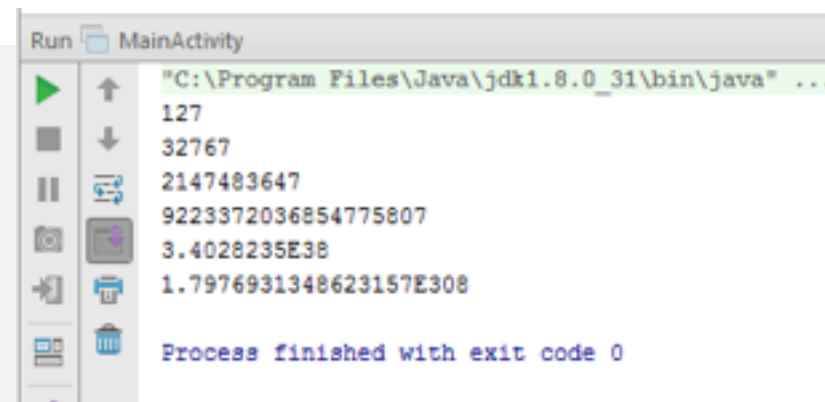
- W celu przechowywania informacji zostały zdefiniowane różne typy danych :
  - *Typy liczbowe* (np. 1, 10.99)
  - *Typ znakowy* (np. 'a')
  - *Typ String* („Ala ma kota”)
  - *Typ logiczny* (prawda/fałsz)
- Typy danych są zdefiniowane na poziomie języka programowania. Mogą one się nieznacznie różnić. Niektóre z nich czasami inaczej się nazywają
- Typy danych w Javie dzielą się na *typy proste* i *typy złożone*

# Typy liczbowe

- *short* (2 bajty pamięci) – może przechowywać liczby całkowite z zakresu od -32.768 do 32.767,
- *int* (4 bajty pamięci) – podobnie jak *short* przechowuje liczby całkowite z tym że ze znacznie większego zakresu od -2.147.483.648 do 2.147.483.647,
- *long* (8 bajtów pamięci) – największy typ całkowity mogący przechowywać liczby z zakresu od -9.223.372.036.854.775.808 do 9.223.372.036.854.775.807,
- *float* (4 bajty pamięci) – może przechowywać liczby zmiennoprzecinkowe (tj. ułamki) z zakresu  $1,40129846432481707e-45$  to  $3,40282346638528860e+38$ ,
- *double* (8 bajtów pamięci) – jest to rozszerzony odpowiednik typu *float* mogący przechowywać liczby z zakresu od  $4.94065645841246544e-324d$  do  $1.79769313486231570e+308d$ ,
- *byte* (1 bajt pamięci) –typ ten odpowiada jednemu bajtowi danych, może przyjmować liczby całkowite z zakresu od -128 do 127
- Wszystkie typy liczbowe tutaj wymienione są typami prostymi

```
byte byteNumber = Byte.MAX_VALUE;  
short shortNumber = Short.MAX_VALUE;  
int intNumber = Integer.MAX_VALUE;  
long longNumber = Long.MAX_VALUE;  
float floatNumber = Float.MAX_VALUE;  
double doubleNumber = Double.MAX_VALUE;
```

```
System.out.println(byteNumber);  
System.out.println(shortNumber);  
System.out.println(intNumber);  
System.out.println(longNumber);  
System.out.println(floatNumber);  
System.out.println(doubleNumber);
```



```
Run MainActivity  
"C:\Program Files\Java\jdk1.8.0_31\bin\java" ...  
127  
32767  
2147483647  
9223372036854775807  
3.4028235E38  
1.7976931348623157E308  
Process finished with exit code 0
```



# Typ znakowy i typ String

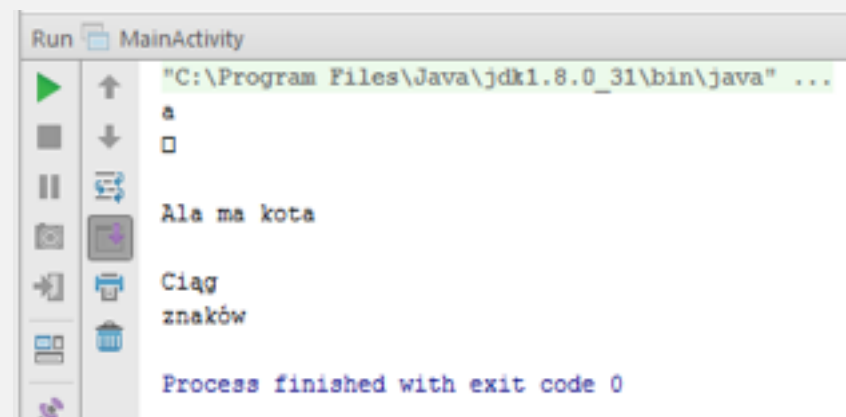
- *char* – odpowiada jednemu znakowi (np. literze), może przechowywać liczby całkowite z zakresu od 0 do 65.535. Char jest typem prostym.
- *String* – jest typem obiekowym. Przechowuje on *ciągi znaków*, mówiąc inaczej tekst. Każdy ciąg znaków ujęty w znaki „” jest rozpoznawany przez Java jako typ String.

Zauważ że typy znakowe w przeciwieństwie do String używają pojedynczego nawiasu *'* a nie podwójnego *“”*

- Typami znakowymi mogą być białe znaki takie jak: znaki nowej linii, spacja, wcięcie w tekście

```
char sign= 'a';
char sign1 = 1;
char sign2 = 65535;
String text = "Ala ma kota";
String text1 = "";
String text2 = "Ciąg\nznaków";
```

```
System.out.println(sign);
System.out.println(sign1);
System.out.println(sign2);
System.out.println(text);
System.out.println(text1);
System.out.println(text2);
```



```
Run MainActivity
"C:\Program Files\Java\jdk1.8.0_31\bin\java" ...
a
1
65535
Ala ma kota

Ciąg
znaków

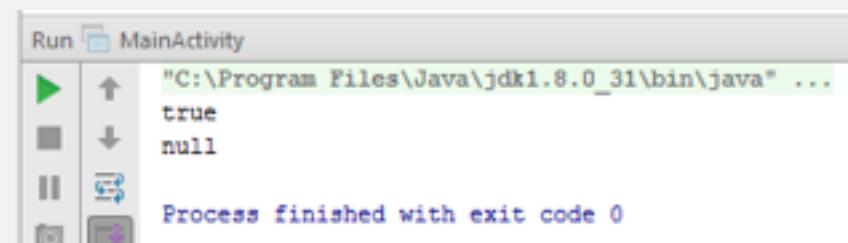
Process finished with exit code 0
```

# Typ logiczny

- *boolean* - typ logiczny – inaczej nazywany typem boolowskim. Może przyjąć jedną z wartości **true** (oznaczającą prawdę) lub **false** (oznaczającą fałsz). Jest typem prostym
- Czasami jednak nie wszystko jest albo prawdą albo fałszem, czasami nie mamy takiej informacji. Wtedy z pomocą przychodzi typ złożony (obiektowy) *Boolean*. Jest to klasa która przyjmuje trzy wartości:
  - *True*
  - *False*
  - *Null*

```
boolean bool = true;  
Boolean bool1 = null;
```

```
System.out.println(bool);  
System.out.println(bool1);
```



# Typy proste vs. Typy złożone

- Typ złożony może być również nazywany typem obiektowym albo referencyjnym

Typ referencyjny ponieważ każdy z elementów posiada własną referencję do pamięci gdzie przechowywana jest wartość

- Typ złożony jak sama nazwa wskazuje składa się z kilku typów prostych. Staje się wtedy obiektem
- Przykładem typu złożonego jest klasa `String`. W językach obiektowych można tworzyć własne typy złożone. Przykładem może być klasa `User` zawierająca w sobie kilka zmiennych zarówno prostych jak i złożonych

```
public class User{  
    String firstName;  
    String secondName;  
    int age;  
    Child child;  
}  
  
public class Child{}
```



# Zmienne

- Przez pojęcie zmiennej w Javie rozumiemy referencję, wskazanie, do określonego miejsca w pamięci komputera. Zmienne reprezentują dane i pozwalają za swoim pośrednictwem nimi manipulować
- Nazwy zmiennych mogą być dowolnymi ciągami znaków, mogą zawierać cyfry przy czym cyfra nie może być pierwszym znakiem. Znak podkreślenia również jest dozwolony
- Zmienne mogą mieć przypisaną wartość, ale nie muszą

```
<typ_danych> <nazwa_zmiennej> = <wartość_zmiennej>;
```

typ\_danych - typ prosty lub złożony,  
nazwa\_zmiennej - wymyślona przez programistę nazwa np. "mojNapis",  
wartosc\_zmiennej - tutaj podajemy wartość np. dla **char** 'a'

```
int youWillNeverKnowWhyIExist;  
String someText = "Mój napis";
```

Warto pamiętać, aby nadawać zmiennym nazwy, które świadczą o celu jej wykorzystania. Kod, który jest czytelny jest wartościowy !

Good job!



# Deklaracja i inicjalizacja

- Deklaracja

W pierwszych dwóch liniach kodu zmienne *age* i *nameAndSurname* zostały zadeklarowane. Oznacza to, że w pamięci alokowana jest przestrzeń o danym rozmiarze.

- Inicjalizacja

Powoduje nadanie odpowiednich wartości dla zmiennych. W przykładzie są to linie 4 i 5.

Deklaracja może być połączona z inicjalizacją. W linii 7 dla zmiennej stanowisko od razu przypisywana jest wartość (inicjalizowana).

Wartości dla zmiennych można zmieniać dowolną ilość razy.

```
1) int age;  
2) String nameAndSurname;  
3)  
4) age = 30;  
5) nameAndSurname = "Jan Kowalski";  
6)  
7) String position = „Programista”;
```

Jedynym wyjątkiem zmiennej której wartości nie można zmienić podczas działania programu są zmienne *final*.

Przy próbie zmiany wartości program nie skompiluje się i otrzymamy komunikat:

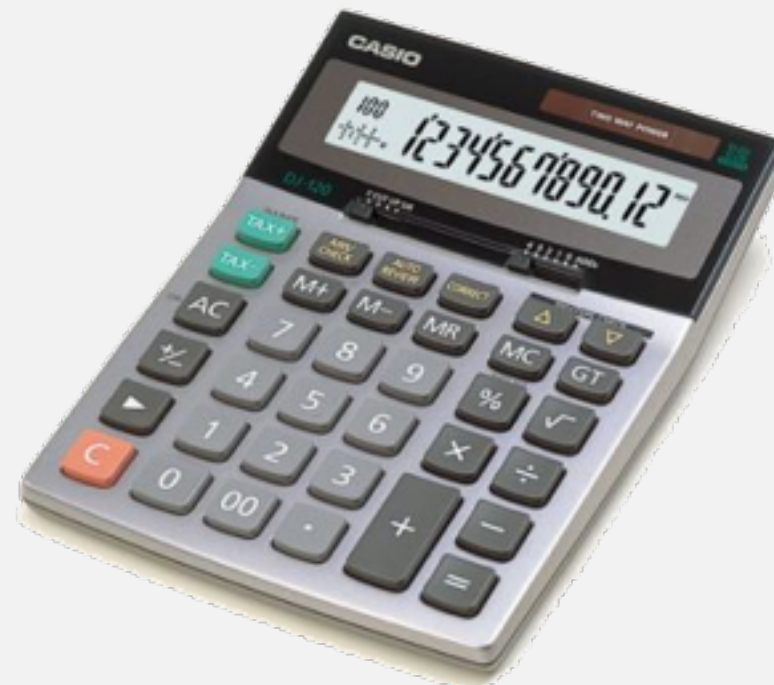
„ The final local variable dogName may already have been assigned”.

```
final String dogName= „Burek”;
```



# Operacje na zmiennych

- Podstawowymi operacjami na typach liczbowych jest na pewno dodawanie, odejmowanie, mnożenie, dzielenie i modulo (reszta z dzielenia)
- Bardziej zaawansowane operacje znajdziesz w klasie `java.lang.Math`
- Warto zapamiętać, że zmienne typu `String` są *immutable* co oznacza że modyfikacje na tych zmiennych powodują zawsze utworzenie nowego obiektu



# Przykłady operacji na zmiennych

W tym przykładzie zaskakujący jest wynik operacji dzielenia.

Dlaczego pojawiło się 2.0 a nie 2.5?

Zmienne *a* i *b* są zmiennymi całkowitymi, więc operacja dzielenia zwróci także liczbę całkowitą – 2. Następnie liczba całkowita 2 jest konwertowana na typ zmiennoprzecinkowy *double*. Stąd wynik wyświetlony to 2.0.

Aby uzyskać spodziewany wynik 2.5 przynajmniej jeden z argumentów dzielenia musi być zmiennoprzecinkowy, albo cała operacja musi być konwertowana na typ *double* tak jak w przykładzie drugim.

```
// Przykład 1
int a = 10;
int b = 4;
int sum = a + b; // dodawanie
int sub = a - b; // odejmowanie
int multi = a * b; // mnożenie
double div = a / b; // dzielenie całkowitoliczbowe
int mod = a % b;
```

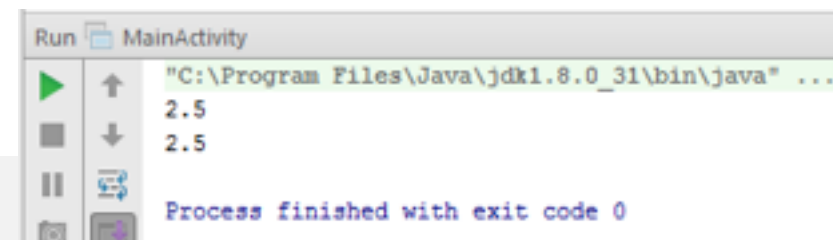
```
System.out.println(sum);
System.out.println(sub);
System.out.println(multi);
System.out.println(div);
System.out.println(mod);
```

```
// Przykład 2
int a = 10;
int b = 4;
double b2 = 4d;
double div = (double) a / b; // dzielenie całkowitoliczbowe
double div2 = a / b2; // dzielenie mieszane
System.out.println(div);
System.out.println(div2);
```



Run MainActivity

```
"C:\Program Files\Java\jdk1.8.0_31\bin\java" ...
14
6
40
2.0
2
Process finished with exit code 0
```



Run MainActivity

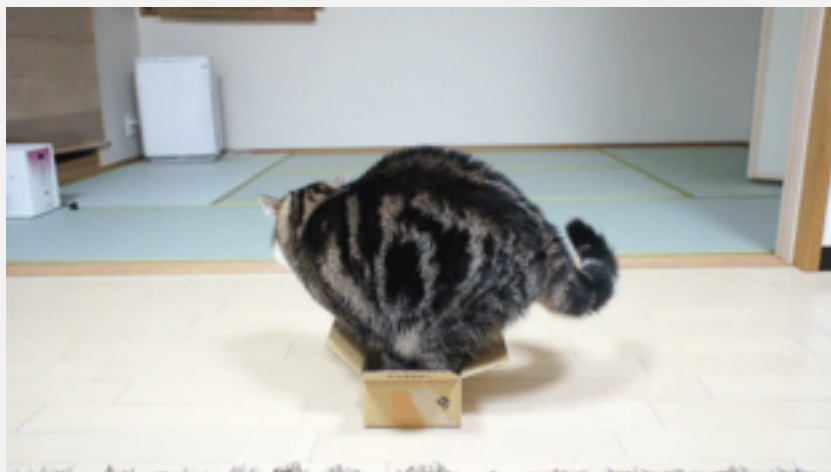
```
"C:\Program Files\Java\jdk1.8.0_31\bin\java" ...
2.5
2.5
Process finished with exit code 0
```

# Przykłady operacji na zmiennych – konwersja typu

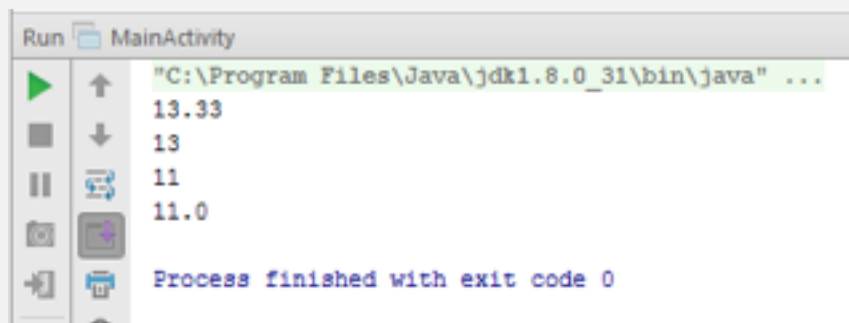
Konwersja typu – jest to zmiana typu danych na inny.

Wyróżnia się dwa typy konwersji:

- Zawężająca – typ o większym rozmiarze zamieniany jest na typ o mniejszym rozmiarze
- Rozszerzająca – typ o mniejszym rozmiarze zamieniany jest na typ o większym rozmiarze



```
double d = 13.33d;  
int a = 11;  
  
System.out.println(d);  
System.out.println((int) d); // konwersja zawężająca  
  
System.out.println(a);  
System.out.println((double) a); // konwersja rozszerzająca
```



```
Run MainActivity  
"C:\Program Files\Java\jdk1.8.0_31\bin\java" ...  
13.33  
13  
11  
11.0  
Process finished with exit code 0
```

# Przykłady operatorów skróconych

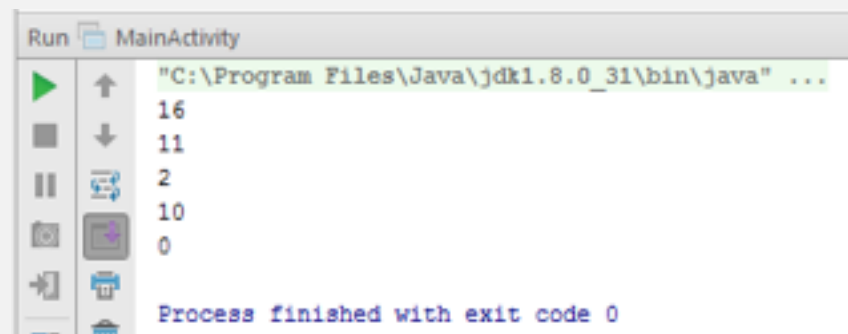
Operatory skrócone są to jak sama nazwa wskazuje krótką wersją zwykłych operatorów.

*Zmienna* += 5;

jest równoznaczne z:

*Zmienna* = *Zmienna* + 5;

```
int a = 11;  
a += 5;  
System.out.println(a);  
  
a -= 5;  
System.out.println(a);  
  
a /= 5;  
System.out.println(a);  
  
a *= 5;  
System.out.println(a);  
  
a &= 5;  
System.out.println(a);
```



```
Run MainActivity  
"C:\Program Files\Java\jdk1.8.0_31\bin\java" ...  
16  
11  
2  
10  
0  
Process finished with exit code 0
```



# Przykłady operatorów logicznych i porównania

## Operatory porównania:

- == porównanie równości,
- != różne,
- > większy,
- < mniejszy,
- >= większy równy,
- <= mniejszy równy,

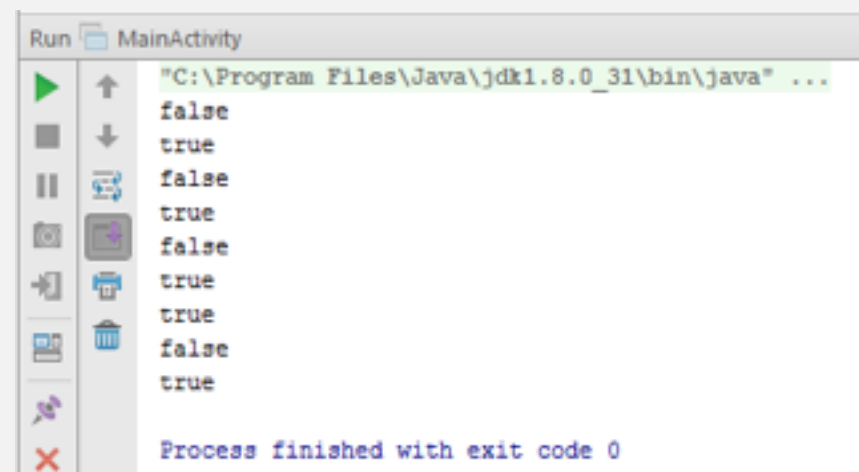
## Operatory logiczne:

- ! negacja,
- || alternatywa,
- && koniunkcja

```
int a = 3, b = 7;

System.out.println(a == b);
System.out.println(a != b);
System.out.println(a > b);
System.out.println(a < b);
System.out.println(a >= b);
System.out.println(a <= b);

System.out.println(a == b || a < b);
System.out.println(a == b && a < b);
System.out.println(!(a > b));
```



```
Run MainActivity
"C:\Program Files\Java\jdk1.8.0_31\bin\java" ...
false
true
false
true
false
true
true
false
true

Process finished with exit code 0
```

# Przykłady inkrementacji i dekrementacji

## Inkrementacja

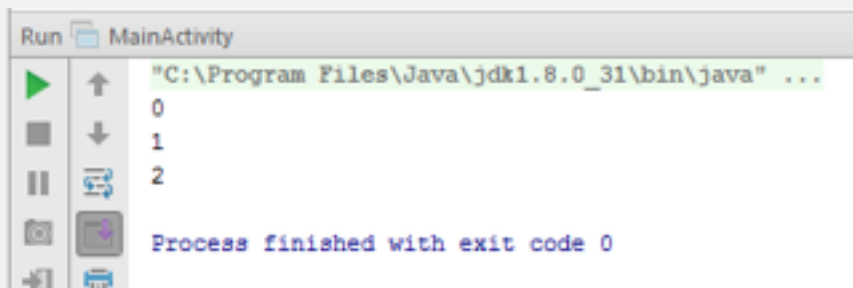
$i++$  równoznaczne z  $i = i + 1$

```
int i = 0;

// najpierw wyświetl później zwiększ
System.out.println(i++);

// wynik inkrementacji
System.out.println(i);

// najpierw zwiększ później wyświetl
System.out.println(++i);
```



```
Run MainActivity
"C:\Program Files\Java\jdk1.8.0_31\bin\java" ...
0
1
2
Process finished with exit code 0
```

## Dekrementacja

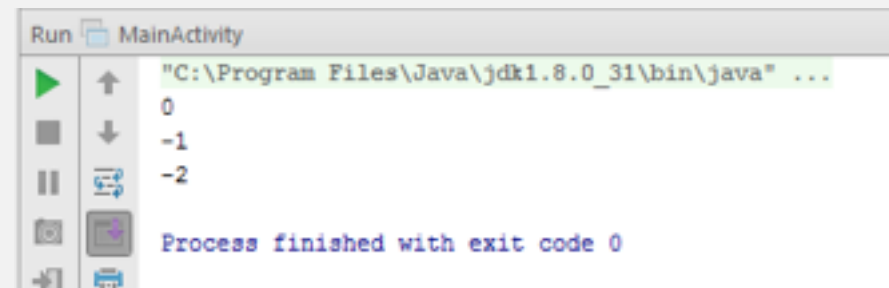
$i--$  równoznaczne z  $i = i - 1$

```
int i = 0;

// najpierw wyświetl później zmniejsz
System.out.println(i--);

// wynik dekrementacji
System.out.println(i);

// najpierw zmniejsz później wyświetl
System.out.println(--i);
```

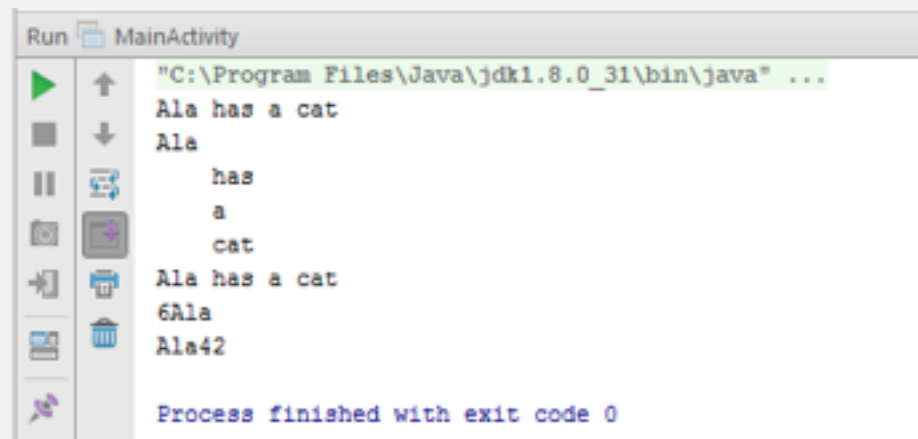


```
Run MainActivity
"C:\Program Files\Java\jdk1.8.0_31\bin\java" ...
0
-1
-2
Process finished with exit code 0
```

# Przykład konkatencji

- Napisy można w bardzo prosty sposób łączyć ze sobą za pomocą znaku „+”
- Kolejność dodawanych elementów ma znaczenie w przypadku wyświetlania liczb

```
String ala = "Ala";  
String has = "has";  
String a = "a";  
String cat = "cat";  
  
System.out.println(ala + " " + has + " " + a + " " + cat);  
System.out.println(ala + "\n\t" + has + "\n\t" + a + "\n\t" + cat);  
  
System.out.println("Ala h"  
    + "as a"  
    + " cat");  
  
System.out.println(4 + 2 + ala);  
System.out.println(ala + 4 + 2);
```



```
Run MainActivity  
"C:\Program Files\Java\jdk1.8.0_31\bin\java" ...  
Ala has a cat  
Ala  
    has  
    a  
    cat  
Ala has a cat  
6Ala  
Ala42  
  
Process finished with exit code 0
```