# Składnia języka Java



## Typy proste

- Liczby, znaki i wartości logiczne w Javie nie są obiektami (w odróżnieniu od "czysto obiektowych języków")
- Java oferuje 8 typów prostych:
  - **boolean** (*true* lub *false*)
  - **char** (16-bitowy Unicode)
  - **byte** (8-bitowy typ całkowity, ze znakiem, U2)
  - **short** (16-bitowy typ całkowity, ze znakiem, U2)
  - **int** (32-bitowy typ całkowity, ze znakiem, U2)
  - **long** (64-bitowy typ całkowity, ze znakiem, U2)
  - **float** (32-bitowy typ zmiennoprzecinkowy, IEEE 754)
  - **double** (64-bitowy typ zmiennoprzecinkowy, IEEE 754)
- W przypadku gdy istnieje konieczność traktowania wartości prostych jak obiektów należy skorzystać z tzw. klas opakowujących (ang. wrapper)



## Komentarze

```
/*
  Komentarz wielowierszowy
  (jak w C i C++).
*/
```

```
// Komentarz jednowierszowy (jak w C++)
```

```
/**
  Komentarz wykorzystywany przez narzędzie javadoc
  do automatycznego generowania dokumentacji.
*/
```



## Zmienne

- Każda zmienna posiada typ
  - typ prosty (int, float, boolean, ...)
  - typ obiektowy
  - typ tablicowy
- Zmienne muszą być deklarowane przed użyciem
- Deklaracja może być połączona z nadaniem wartości
- Java jest językiem o ścisłej kontroli typów

```
int age;
double pi = 3.14;
boolean prawda = true, falsz = false;
```

# Nazwy zmiennych

- Zaczynają się od litery, znaku podkreślenia lub $
- Na kolejnych pozycjach mogą występować również cyfry
- Nazwy zmiennych nie mogą pokrywać się z zastrzeżonymi słowami kluczowymi języka Java:
  - `boolean, byte, char, double, float, int, long, short, void`
  - `false, null, true`
  - `abstract, final, native, private, protected, public, static, synchronized, transient, volatile`
  - `break, case, catch, continue, default, do, else, finally, for, if, return, switch, throw, try, while`
  - `class, extends, implements, interface, throws`
  - `import, package, instanceof, new, super, this`



# Literały

- Całkowitoliczbowe
  - 12, -12, 0123, 0x12f, 0X7A3, 15L
- Zmiennoprzecinkowe
  - 8.31, 3.00e+8, 8.31F , 3.00e+8f
- Logiczne
  - true, false
- Znakowe
  - 'a', '\n', '\u00ff', '\077'
- Tekstowe
  - "Hello\n"



# Operator przypisania =

- Operator o wiązaniu prawostronnym
- Przypisania można łączyć (instrukcja przypisania zwraca wartość)

```
int myAge, yourAge;
double pi;
boolean prawda;

pi = 3.14;
prawda = true;
myAge = yourAge = 28;
```



# Jawna i niejawna konwersja typów

- Niejawna konwersja dokonywana jest z mniejszych do większych typów całkowitoliczbowych (byte -> short -> int -> long)
- W innych przypadkach konieczna jest jawna konwersja przez operację rzutowania

```
int i = 1;
short s = 3;
byte b;

i = s; // OK
b = s; // B■ąd
b = (byte) s; // OK, ale możliwość utraty informacji
```

# Operatory (1)

- Arytmetyczne
  - +, -, *, /, % (modulo)
- Inkrementacja, dekrementacja
  - ++, --
  - 2 warianty: prefiks, postfix
- Porównania
  - >, <, >=, <=, ==, !=
  - ich wynikiem jest wartość typu *boolean*
- Logiczne
  - &&, & (**and** z/bez krótkiego wartościowania)
  - ||, | (**or** z/bez krótkiego wartościowania)
  - ^ (**xor**)
  - ! **(not)**



# Operatory (2)

- Przypisanie złożone
  - op=, gdzie *op* jest operatorem dwuargumentowym np. +=, -=
- Konkatenacja łańcuchów znaków (obiektów klasy `String`)
  - + (jedyny przeciążony operator w Javie)
- Priorytet operatorów

| Priorytet | Operatory | Wiązanie |
|---|---|---|
| 1 | ++ -- + - ~ ! (typ) | R |
| 2 | * / % | L |
| 3 | + - + | L |
| 4 | << >> >>> | L |
| 5 | < > <= >= instanceof | L |
| 6 | == != | L |
| 7 | & | L |
| 8 | ^ | L |
| 9 | \| | L |
| 10 | && | L |
| 11 | \|\| | L |
| 12 | ?: | L |
| 13 | = op= | R |



# Blok kodu

- Sekwencja instrukcji ograniczona nawiasami klamrowymi
- Może wystąpić w instrukcjach warunkowych i pętlach zamiast pojedynczej instrukcji

```
{
  int i = 1;
  short s = 3;
  byte b;

  i = s;
  b = (byte) s;
}
```



# Instrukcja *if*

```
if ( boolean_expr )
    statement1;
else
    statement2;
```

# Operator ?:

```
boolean_expr ? expr1 : expr2
```

# Instrukcja *switch*

```
switch ( integer_expr ) {

    case constant_expr1:
        statement1;
        break;
    case constant_expr2:
        statement2;
        break;
    default:
        statement3;
        break;
}
```



# Pętla *while*

```
while ( boolean_expr )
    statement;
```

# Pętla *do-while*

```
do
    statement;
while ( boolean_expr );
```

# Pętla *for*

```
for ( init_expr; boolean_expr; update_expr )
    statement;
```



# Instrukcje *break* i *continue*

- **break** służy do opuszczenia pętli lub instrukcji *switch*
- **continue** służy do przejścia do następnej iteracji pętli
- Powyższe instrukcje domyślnie "wyskakują" z najbardziej zagnieżdżonej pętli
- Powyższe instrukcje mogą posiadać etykiety
  - możliwość wyskoku z kilku zagnieżdżonych pętli
  - kompensacja braku instrukcji *goto*

```
outer_loop:
for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 5; j++) {
        System.out.println(i, j);
        if (i + j > 7)
            break outer_loop;
    }
}
```



# Klasy

```
[ModyfikatorKlasy] class NazwaKlasy [extends Nadklasa]
     [implements ListaInterfejsów] {
  // Lista metod i pól
}
```

- ModyfikatorKlasy może być kombinacją wyrażeń:
  - abstract - klasa zawiera metody abstrakcyjne
  - final - nie może posiadać podklas
  - public - może być używana w kodzie poza klasą, jedna klasa publiczna w pliku (nazwa pliku = nazwa klasy)
  - private - dostępna tylko w obrębie pliku
  - <puste> - dostęp w ramach pakietu, w którym występuje

# Metody

```
[ModyfikatorMetody] TypZwrotny Nazwa (Typ arg1, ...) {
  // implementacja metody
}
```

- ModyfikatorMetody może być kombinacją wyrażeń:
  - modyfikator widzialności
    - public - dostępna dla metod spoza klasy
    - protected - dostępna w klasach z pakietu i wszystkich podklasach
    - <puste> - dostępna w klasach z pakietu, w którym występuje
    - private - dostępna tylko dla metod z tej samej klasy
  - final - metoda nie może zostać przesłonięta w podklasie
  - static - wspólna dla wszystkich wystąpień obiektu
  - synchronized - blokuje dostęp do obiektu na czas wykonywania
  - native - zaimplementowana w innym języku
  - abstract - metoda bez implementacji



# Pola

```
[ModyfikatorPola] Typ Nazwa [ = wartość];
```

- ModyfikatorPola może być kombinacją wyrażeń:
  - modyfikator widzialności
    - public - dostępna dla metod spoza klasy
    - protected - dostępna w klasach z pakietu i wszystkich podklasach
    - <puste> - dostępna w klasach z pakietu, w którym występuje
    - private - dostępna tylko dla metod z tej samej klasy
  - static - wspólne dla wszystkich wystąpień obiektu
  - final - stała, musi być zainicjalizowana



# Definiowanie klasy - przykład