Wykład II

Podstawowe elementy języka

Typy danych w Javie

- Java jest językiem ze ścisłą kontrola typów, w którym rozmiar i postać danych są określone bardzo precyzyjnie.
- Typy danych w Javie można podzielić na typy proste i typy referencyjne (klasy, interfejsy i tablice).
- Do przechowywania liczb całkowitych przeznaczone są cztery typy:
  - byte (8), short (16), int (32) oraz long (64).

Typ	Rozmiar	Zakres przechowywanych danych
byte	8 bitów	-128 do 127
short	16 bitów	-32768 do 32767
int	32 bity	-2147483648 do 2147483647
long	64 bity	-9223372036854775808 do 9223372036854775807

- Rzeczywiste typy liczbowe to: float (32) i double (64).

Typ	Rozmiar	Zakres przechowywanych danych
float	32 bity	1.4E-45 do 3.4E+38
double	64 bity	4.9E-324 do 1.7E+308

- Dane znakowe zapisywane są zgodnie ze standardem Unicode - są to 16-bitowe liczby całkowite bez znaku. Do ich przechowywania służy typ char.
- Typ boolean (1 bit) umożliwia przechowywanie wartości logicznych. Może on przyjmować tylko dwie wartości: true i false
- Typ referencyjny – nazwy typu referencyjnego pochodzą od nazwy klasy lub interfejsu. Wartością zmiennej typu referencyjnego jest referencja (odniesienie) do obiektu.

Literały

Literał to napis reprezentujący w sposób bezpośredni wartość danej. W Javie wyróżniamy literały liczbowe, znakowe, łańcuchowe i logiczne.

Literał liczbowy to bezpośredni zapis konkretnej liczby.

Liczby całkowite mogą być zapisywane w systemie:

- dziesiętnym, w naturalny sposób np. 3 lub 121
- ósemkowym, poprzez poprzedzenie liczby zerem np. 03
- szesnastkowym, poprzez poprzedzenie liczby znakami 0x lub 0X np. 0xFF

(cyfry szesnastkowe powyżej 9 mogą być zapisywane wielkimi lub małymi literami).

Każda liczba całkowita zapisana literalnie (np. 100) jest traktowana jako liczba typu int.

Liczby całkowite typu long są zapisywane z przyrostkiem L lub l np. 300L.

Literały rzeczywiste

- Przy zapisie liczb rzeczywistych jako separator miejsc dziesiętnych jest stosowana kropka (a nie przecinek) np. 3.1415
- Liczby rzeczywiste mogą być zapisywane w notacji naukowej z wykorzystaniem litery e lub E np. 9.3e-9 (9.3 pomnożone przez 10 do potęgi -9)
- Każda liczba rzeczywista zapisana literalnie (z kropka dziesiętną lub w notacji naukowej) jest traktowana jako liczba typu double.
- Liczby rzeczywiste typu float są zapisywane z przyrostkiem F lub f np. 3.7f.

```
public class Literaly {

	public static void main(String[] args) {
		int i ;
		long j;
		//literał typu integer
		i=100;
		System.out.println(i);
		//literał typu ósemkowego
		i=077;
		System.out.println(i);
		//literał typu szesnastkowego
		i=0xFF;
		System.out.println(i);
		//literał typu Long
		j=100L;
		System.out.println(j);
		//literały rzeczywiste
		float y;
		double x;
		//literał typu rzeczywistego double
		x= 3.14;
		System.out.println(x);
		//literał typu rzeczywistego float
		y=678.67f;
		System.out.println(y);
		//literał typu rzeczywistego double w notacji naukowej
		x=34.56e3;
		System.out.println(x);
		}
}
```

Literał znakowy określa jeden znak zapisany bezpośrednio w programie.

Literały znakowe typu char zapisujemy jako:

- pojedyncze znaki w apostrofach np. 'A' , '+',
- znaki specjalne ujęte w apostrofy np. '\n' , '\t',
- jako kod szesnastkowy Unicode ujęty w apostrofy np. '\u006E' (litera n), '\u001B' (znak Esc),

Bezpośrednich kodów Unikodu nie wolno stosować dla znaków specjalnych LF ('\u000a') i CR ('\000d'). Zamiast tego należy stosować znaki '\n' i '\r'.

Literał łańcuchowy to bezpośrednio zapisane ciągi znaków (napisy), które są traktowane jako teksty.

Łańcuchy znakowe zapisujemy jako ciągi znaków ujęte w cudzysłowy np. "Wpisany nowy tekst".

W łańcuchach można wstawiać znaki specjalne oraz znaki unikodu np. "trzy\nnowe\nwiersze i litera \u0068".

W Javie literały łańcuchowe są zawsze traktowane jako obiekty klasy String.

Literały logiczne są zapisywane wyłącznie za pomocą słów false i true.

Symbole zastępcze znaków specjalnych typu **char**

Znak	Interpretacja
\n	Nowy wiersz
\t	Tabulacja pozioma
\b	backspace
\r	Powrót karetki
\f	Wysunięcie papieru nowa strona
\\	Ukośnik lewy
\'	Znak apostrofu
\"	Znak cudzysłowu
\uNNNN	Znak w standardzie Unicode

```
public class Znaki_specjalne {

	public static void main(String[] args) {
		System.out.println('A');
		System.out.println("Litera h: \t" +'\u0068');
		System.out.println("Wstawianie ukośnika w tekście \t" +'\\');
		System.out.println("Przejście do nowej linii: \n" + "Nowa linia");
		System.out.println("Wstawienie znaku \t"  +"tabulacji");
		System.out.println("Wstawienie cudzysłowiu \"");
		System.out.print("Powrót karetki \r" +"1234");
		System.out.println("Wpisany nowy tekst ");
		System.out.println("Wprowadzenie w tekscie znaku ukośnika \\ ");
		System.out.println("Wprowadzenie w tekscie znaku cudzysłowiu \"
");
		System.out.println("Znak podany w Unikodzie \u0067 ");
		System.out.println(true);
	}
}
```

Zmienne

Deklaracje zmiennych

- Zmienne typów podstawowych Np. **int** a;
- Zmienne typu klasa

String nazwisko; //zmienna nazwisko typu referencja do obiektu

int x , y;

Punkt p;

Inicjacja zmiennych

nazwisko = "Tomaszewicz";

lub

String nazwisko = ”Tomaszewicz”;

p = **new** Punkt(); //p jest teraz odwołaniem do obiektu typu Punkt

lub

Punkt p = **new** Punkt();

- Zmiene ustalone

final int Init = 1; //nie można zmienić wartości zmiennej ustalonej Init

Punkt = **new** Punkt(Init, Init); //zastosowana do zainicjowania obiektu –

//może poprawić czytelność programu

Typy wyliczeniowe

Wyliczenia tworzy się za pomocą słowa kluczowego *enum*, np.:

```
enum Dzien_tygodnia
      { Poniedziałek, Wtorek, Środa, Czwartek, Piątek, Sobota, Niedziela
      }
```

Identyfikatory ***Poniedziałek, Wtorek, Środa, Czwartek, Piątek, Sobota, Niedziela*** nazywany stałymi wyliczeniowymi, są one publicznymi statycznymi składowymi wyliczenia i posiadają taki sam typ jak wyliczenie

W programie można deklarować zmienne wyliczeniowe, którym można przypisywać stałe wyliczenia, np.:

```
Dzien_tygodnia dzien;
dzien=Dzien_tygodnia.Wtorek.
```

- Stałe wyliczeniowe można wykorzystywać w instrukcji warunkowej *if* oraz w instrukcji wyboru *switch*, np.:

```
if (dzien==Dzien_tygodnia.Wtorek)

switch(dzien)
            {
            case Wtorek: System.out.print("Tuesday");  break;
            case Środa: System.out.print(" Wednesday"); break;
            case Sobota: System.out.print("Saturday"); break;
```

- Wszystkie wyliczenia automatycznie zawierają dwie predefiniowane metody:

public static typ-wyliczeniowy[] values()
public static typ-wyliczeniowy valueOf(String tekst)

Metoda *values()* zwraca tablice zawierającą listę stałych wyliczeniowych.
Metoda *valueOf()* zwraca stałą wyliczeniową, której odpowiada tekst przekazany jako argument.

Przykład:

```
public class Dni_tygodnia {
      // deklaracja typu wyliczeniowego
      enum Dzien_tygodnia
      { Poniedziałek, Wtorek, Środa, Czwartek, Piątek, Sobota, Niedziela
      }

      public static void main(String[ ] args)
      { // deklaracja zmniennej typu wyliczeniowego
      Dzien_tygodnia dzien;
      // użycie metody values()
      System.out.println("Zdefiniowane wszystkie dni tygodnia");
      Dzien_tygodnia[] tab = Dzien_tygodnia.values();
      System.out.println(tab[0]);
      System.out.println(tab[1]);
      System.out.println(tab[2]);
      // użycie metody valueOf()
      dzien = Dzien_tygodnia.valueOf("Piątek");
      // użycie stałej wyliczeniowej w instrukcji if
if (dzien==Dzien_tygodnia.Poniedziałek) System.out.println("\n Monday \n");
if (dzien==Dzien_tygodnia.Wtorek) System.out.println("\n Tuesday \n");
if (dzien==Dzien_tygodnia.Piątek) System.out.println("\n Friday \n");
      }

}
```

Słowa kluczowe

Słowa kluczowe to słowa, które maja specjalne znaczenie (np. oznaczają instrukcje sterujące) i nie mogą być używane w innym kontekście niż określa to składnia języka.

abstract default if package synchronized assert do

implements private this boolean double import protected throw

break else instanceof public throws byte extends int return transient

case false interface short true

catch final long static try char finally native strictfp void class

float new super volatile const for null switch while continue goto

Uwagi:

słowa kluczowe goto i const, są zarezerwowane ale nie są używane.

słowa boolean, byte, char, double, float, int, long, short

są nazwami typów podstawowych.

słowa true, false i null są nazwami stałych.

Identyfikatory

Identyfikatory to tworzone przez programistę nazwy klas, pól i metod klasy oraz stałych i zmiennych.

Identyfikator musi zaczynać się od litery lub podkreślenia i może składać się z dowolnych znaków alfanumerycznych (liter i cyfr) oraz znaków podkreślenia.

Java rozróżnia wielkie i małe litery w identyfikatorach

Nazwa identyfikatora nie może być słowem kluczowym.

Zalecenia:

Nazwy klas: wszystkie słowa w nazwie rozpoczynać dużą literą,

np.: ObiektGraficzny

Nazwy metod i pól publicznych: pierwsze słowo rozpoczynać mała litera,

a kolejne wyrazy dużą litera, np.: rysujTlo, kolorWypelnienia.

Nazwy metod i pól prywatnych: należy pisać wyłącznie małymi literami,

a wyrazy łączyć podkreśleniem, np.: kierunek_ruchu.

Nazwy zmiennych niemodyfikowalnych (stałych): należy pisać wyłącznie dużymi literami, a wyrazy łączyć podkreśleniem, np.: ROZMIAR_TABLICY.

Operatory

Operatory są to specjalne symbole stosowane do wykonywania działań arytmetycznych, przypisań, porównań i innych operacji na danych.

Dane, na których są wykonywane operacje są nazywane argumentami.
Operatory są jedno, dwu lub trzyargumentowe.

Uwaga: Niektóre operatory mogą być stosowane zarówno jako jednoargumentowe jak i dwuargumentowe np. + .

Każdy operator może być stosowany wyłącznie do danych określonego typu.

Wynik działania operatora jest określonego typu.

Uwaga: Dla niektórych operatorów typ wyniku zależy od typu argumentów.

Wyrażenia tworzy sie za pomocą operatorów i nawiasów ze zmiennych, stałych, literałów oraz wywołań metod. Wyrażenia są opracowywane (wyliczane), a ich wyniki mogą być w różny sposób wykorzystane np. w przypisaniach, jako argumenty innych operatorów, w instrukcjach sterujących wykonaniem programu, w wywołaniach metod, itd.

Operatory arytmetyczne

Operatory arytmetyczne +, –, * i / do wykonywania operacji dodawania, odejmowania, mnożenia i dzielenia. Operator % jest wykorzystywany do dzielenia modulo czyli otrzymywania reszty z dzielenia.

Kolejność opracowywania (wyliczania) wyrażeń zależy od priorytetów i wiązań operatorów użytych w tych wyrażeniach.

Priorytety mówią o tym, w jakiej kolejności będą wykonywane różne operacje w tym samym wyrażeniu.

Przykład:

W wyrażeniu *a+b*c* najpierw będzie wykonane mnożenie, a potem dodawanie ponieważ operator * ma wyższy priorytet niż operator +.

Żeby odwrócić kolejność wykonywania działań trzeba użyć nawiasów: (a+b)*c

Wiązania określają kolejność wykonywania operacji o tym samym priorytecie tzn. czy są one wykonywane od lewej strony wyrażenia czy od prawej.

Przykład:

W wyrażeniu a-b+c najpierw będzie wykonane odejmowanie, a potem dodawanie bo wiązanie operatorów + i – jest lewostronne.

Żeby odwrócić kolejność wykonywania działań trzeba użyć nawiasów: a-(b+c)

Zestawienie operatorów dostępnych w Javie

wiązanie i priorytet		operator	sposób użycia	działanie
lewe	1	.	obiekt.składowa	wybór składowej klasy
		[ ]	tablica[wyrażenie]	indeks tablicy
		( )	metoda(lista wyrażeń)	wywołanie metody
prawe	2	++	zmienna++	przyrostkowe / przedrostkowe zwiększenie o 1
			++zmienna	
		--	zmienna--	przyrostkowe / przedrostkowe zmniejszenie o 1
			--zmienna	
		+	+wyrażenie	jednoargumentowy plus,
		-	-wyrażenie	jednoargumentowy minus
		!	!wyrażenie	negacja logiczna
		~	~wyrażenie	dopełnienie bitowe
		(typ)	(typ)wyrażenie	rzutowanie typu
		new	new typ	tworzenie obiektu
lewe	3	*	wyrażenie*wyrażenie	mnożenie,
		/	wyrażenie/wyrażenie	dzielenie,
		%	wyrażenie%wyrażenie	modulo

wiązanie i priorytet		operator	sposób użycia	działanie
lewe	4	+	wyrażenie+wyrażenie	dodawanie, łączenie łańcuchów,
		-	wyrażenie-wyrażenie	odejmowanie
lewe	5	<<	wyrażenie<<wyrażenie	przesunięcie bitowe w lewo
		>>	wyrażenie>>wyrażenie	przesunięcie bitowe w prawo
		>>>	wyrażenie>>>wyrażenie	przes. bitowe w prawo bez znaku
lewe	6	<	wyrażenie<wyrażenie	mniejsze,
		<=	wyrażenie<=wyrażenie	mniejsze lub równe,
		>	wyrażenie>wyrażenie	większe,
		>=	wyrażenie>=wyrażenie	większe lub równe
		instanceof	obiekt instanceof klasa	stwierdzenie typu obiektu
	7	==	wyrażenie==wyrażenie	równość,
		!=	wyrażenie!=wyrażenie	nierówność
lewe	8	&	wyrażenie&wyrażenie	bitowe AND
	9	^	wyrażenie^wyrażenie	bitiwe OR wyłączające
	10	|	wyrażenie|wyrażenie	bitiwe OR
	11	&&	wyrażenie&&wyrażenie	logiczne AND
	12	||	wyrażenie||wyrażenie	logiczne OR
	13	? :	wyraż ? wyraż : wyraż	operator warunku

wiązanie i priorytet		operator	sposób użycia	działanie
prawe	14	=	zmienna=wyrażenie	proste przypisanie
		*=	zmienna*=wyrażenie	pomnóż i przypisz
		/=	zmienna/=wyrażenie	podziel i przypisz
		%=	zmienna%=wyrażenie	oblicz modulo i przypisz
		+=	zmienna+=wyrażenie	dodaj i przypisz
		-=	zmienna-=wyrażenie	odejmij i przypisz
		<<=	zmienna<<=wyrażenie	przesuń w lewo i przypisz
		>>=	zmienna>>=wyrażenie	przesuń w prawo i przypisz
		>>>=	zmienna>>>=wyrażenie	przesuń w prawo bez znaku i przypisz
		&=	zmienna&=wyrażenie	koniunkcja bitowa i przypisz
		^=	zmienna^=wyrażenie	różnica bitowa i przypisz
		|=	zmienna|=wyrażenie	alternatywa bitowa i przypisz

Wykonamy przykładowy program w trybie konsoli

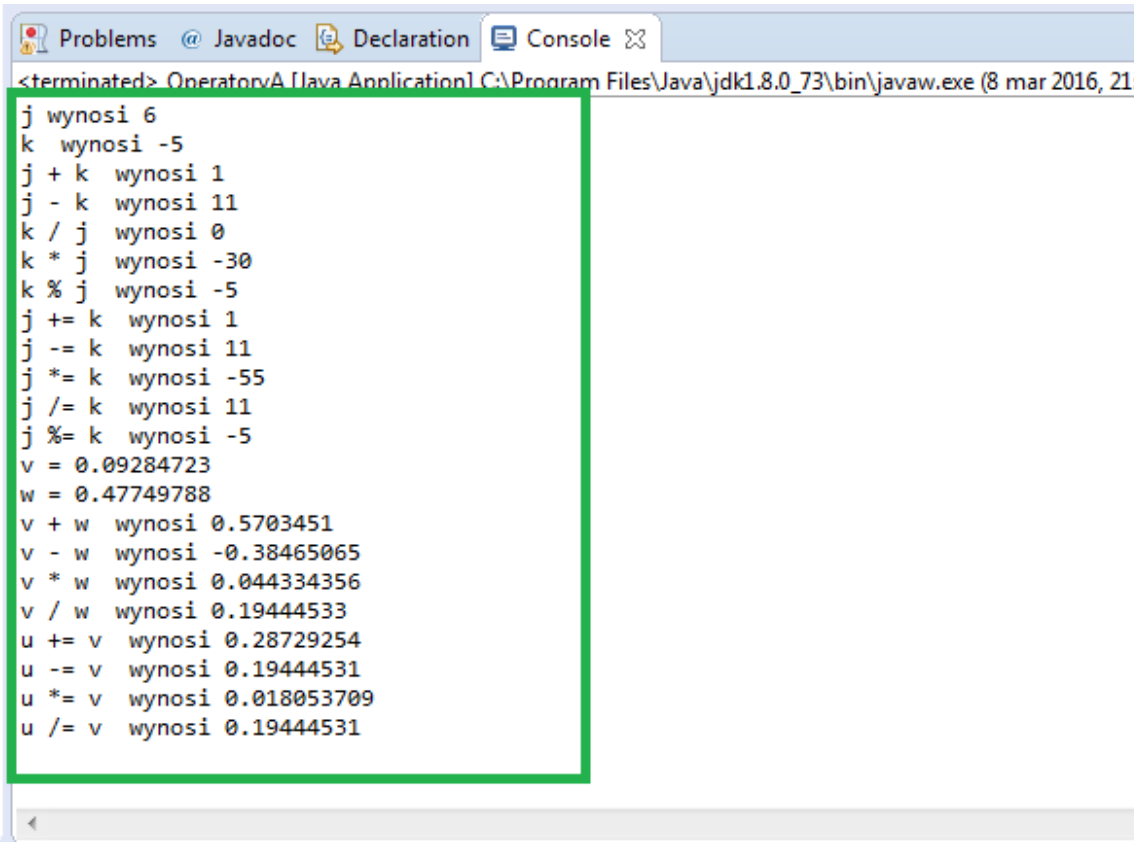
```
import java.util.*;
public class OperatoryA {

	public static void main(String[] args)
	{
		//  Tworzy  generator liczb losowych, zainicjowany
	    Random rand = new Random();
	    int i,  j, k;    String s;
	// '%' ogranicza wartość do 9: losowanie kolejnej wartości całkowitej
	    j = rand.nextInt() % 10;    k = rand.nextInt() % 10;
	    s = "j wynosi "+ j;
	System.out.println(s);
	    s = "k  wynosi "+ k;                                System.out.println(s);
	    // operacje dodawania odejmowania dzielenie mnożenia
	    i = j + k;            s = "j + k  wynosi " + i;
	System.out.println(s);
	    i = j - k;            s = "j - k  wynosi " + i;
	System.out.println(s);
	    i = k / j;            s = "k / j  wynosi " + i;
	System.out.println(s);
	    i = k * j;            s = "k * j  wynosi " + i;
	System.out.println(s);
	    i = k % j;      s = "k % j  wynosi " + i;
	System.out.println(s);
	    i=j;
	    // działanie j = j + k
	    j += k;               s = "j += k  wynosi " + j;
	System.out.println(s);

	    // działanie j = j - k
	    j -= k;         s = "j -= k  wynosi " + j;
	System.out.println(s);
	    // działanie j = j * k
	    j *= k;               s = "j *= k  wynosi " + j;
	System.out.println(s);
	    // działanie j = j /k
	    j /= k;               s = "j /= k  wynosi " + j;
	System.out.println(s);
	  //działanie k= k % j
	    k %= j;         s = "j %= k  wynosi " + k;
	System.out.println(s);
	    // Operacje na argumentach zmniennoprzecinkowych u,v,w
	    float u, v, w;
	//losowanie kolejnej wartośći rzeczywistej
	    v = rand.nextFloat();        w = rand.nextFloat();
	    s = "v = " + v;
	System.out.println(s);
	    s = "w = " + w;
	System.out.println(s);
	    u = v + w;      s = "v + w  wynosi " + u;
	System.out.println(s);
	    u = v - w;      s = "v - w  wynosi " + u; System.out.println(s);
	    u = v * w;      s = "v * w  wynosi " + u; System.out.println(s);
	    u = v / w;      s = "v / w  wynosi " + u; System.out.println(s);
	    //  działanie u = u + v
	    u += v;               s = "u += v  wynosi " + u;
	System.out.println(s);
	    //  działanie u = u - v
	    u -= v;         s = "u -= v  wynosi " + u;
	System.out.println(s);
	//  działanie u = u - v
```

```
        u *= v;                 s = "u *= v  wynosi " + u;
    System.out.println(s);
    //  działanie u = u / v
        u /= v;                 s = "u /= v  wynosi " + u;
    System.out.println(s);
    }

}
```



Wynik działania programu

Przykładowy program w trybie graficznym

```
import javax.swing.*;
import java.util.*;
public class OperatoryAG {

	public static void main(String[] args) {
		  //definicja zmiennych  całkowitych I, j, k oraz łańcucha s
		  int i, j, k; String s;
		  //  pobranie z okienka dialogowego łańcucha s
		 s = JOptionPane.showInputDialog(null,
		                  "Podaj pierwszy argument całkowity");
		 //zamiana łancucha s na liczbę
		  j = Integer.parseInt(s);
		 //  pobranie z okienka dialogowego łańcucha s
		 s = JOptionPane.showInputDialog(null,
		                  "Podaj drugi argument całkowity");
		  //zamiana łancucha s na liczbę 3
		 k = Integer.parseInt(s);
		//definicja zmiennych rzeczywistych u, v w
		float u, v, w;
		   //  pobranie z okienka dialogowego łańcucha s
		s=JOptionPane.showInputDialog(null,
		                  "Podaj pierwszy argument rzeczywisty");
		 //zamiana łancucha s na liczbę
		 v = Float.parseFloat(s);
		  //  pobranie z okienka dialogowego łańcucha s
		  s = JOptionPane.showInputDialog(null,
		                  "Podaj drugi argument rzeczywisty");
		//zamiana łancucha s na liczbę
		 w = Float.parseFloat(s);
+
		    i = j+k;              s="j+k wynosi "+ i + "\n";
		  // dodanie do łańcucha s nowego łancucha s+= czyli s=s+
		    i = j - k;            s+="j-k wynosi " + i + "\n";
		    i = j / k;            s+="j/k wynosi " + i + "\n";
		    i = j * k;            s+="j*k wynosi " + i + "\n";
		    i = j % k;            s+="j%k wynosi " + i + "\n";

		   // Operacje na argumentach zmniennoprzecinkowych
		    u = v + w;      s += "v + w  wynosi " + u +"\n";
		    u = v - w;      s += "v - w  wynosi " + u +"\n";
		    u = v * w;      s += "v * w  wynosi " + u +"\n";
		    u = v / w;      s += "v / w  wynosi " + u +"\n";

		  //  następne wyrażenia są realizowane dla
		    // char, byte, short, int, long i double:
		    u += v; s += "u += v  wynosi " + u +"\n";
		    u -= v; s += "u -= v  wynosi " + u +"\n";
		    u *= v; s += "u *= v  wynosi " + u +"\n";
		    u /= v; s += "u /= v  wynosi " + u +"\n";

		 //wyświetlenie łańcucha s
		   JOptionPane.showMessageDialog(null,s);
		    System.exit(0);

	}

}
```

Operatory inkrementacji i dekrementacji

Operator ++ zwiększa, a –-- zmniejsza o jeden wartość argumentu (zmiennej). Oba występują w dwóch postaciach:

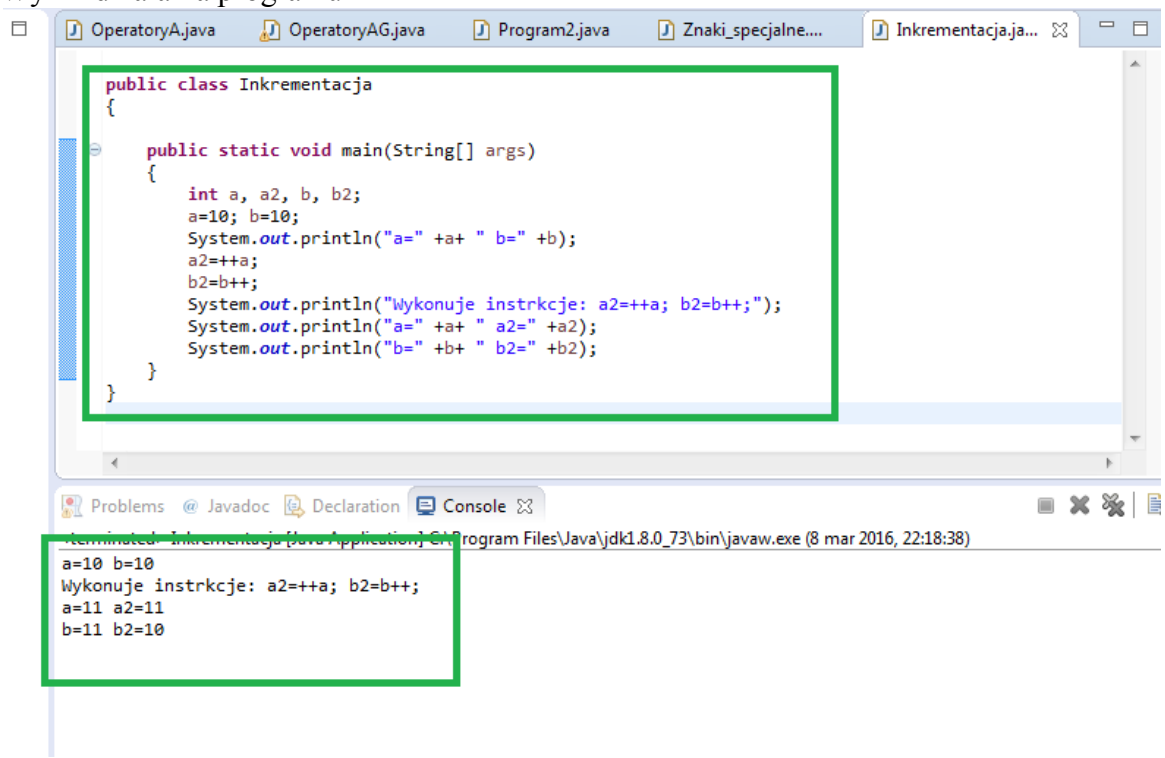
- przyrostkowej (operator po argumencie –- zmiennej),
- przedrostkowej (operator przed argumentem –- zmienną).

Przyrostkowa forma operatorów modyfikuje wartość argumentu po jej wykorzystaniu w wyrażeniu, przedrostkowa przed wykorzystaniem tej wartości.

Przykład:

```
public class Inkrementacja
{
	public static void main(String[] args)
	{
		int a, a2, b, b2;
		a=10; b=10;
		System.out.println("a=" +a+ " b=" +b);
		a2=++a;
		b2=b++;
		System.out.println("Wykonuje instrkcje: a2=++a; b2=b++;");
		System.out.println("a=" +a+ " a2=" +a2);
		System.out.println("b=" +b+ " b2=" +b2);
	}
}
```

Wynik działania programu



Rodzaje instrukcji w języku Java

Instrukcja pusta – nie powoduje wykonania żadnych działań np. ;

Instrukcje wyrażeniowe:

- przypisanie np. a = b;
- preinkrementacja np. ++a;
- predekrementacja np. --b;
- postinckrementacja np. a++;
- postdekrementacja np. b--;
- wywołanie metody np. x.metoda();
- wyrażenie new np. new Para();

Uwaga: instrukcja wyrażeniowa jest zawsze zakończona średnikiem.

Instrukcja grupująca – dowolne instrukcje i deklaracje zmiennych ujęte w nawiasy klamrowe np.

```
{ int a,b;
a = 2*a+b;
}
```

Instrukcja etykietowana – identyfikator i następujący po nim dwukropek wskazujący instrukcje sterującą switch, for, while lub do.

Instrukcja sterująca – umożliwia zmianę sekwencji (kolejności) wykonania innych instrukcji programu.

Rozróżniamy instrukcje:

- warunkowe: if, if ... else, switch
- iteracyjne: for, while, do ... while
- skoku: break, continue, return

Instrukcja throw – zgłaszanie wyjątku przerywającego normalny tok działania programu.

Instrukcja synchronized – wymuszanie synchronizacji przy współbieżnym wykonywaniu różnych wątków programu.