PODSTAWOWE ELEMENTY JĘZYKA JAVA – TYPY DANYCH, OPERATORY I INSTRUKCJE

1. Definicja klasy, dziedziczenie, implementowanie metod interfejsów

```
class nazwa_klasy
{
        //ciało klasy
}
```

Klasa:

✓ przed słowem class może wystąpić jeden ze specyfikatorów:

public – klasa dostępna publicznie

final, **public final** - klasa ta nie może mieć następcy

abstract, **public abstract** – klasy te nie mają wystąpień

Klasa abstrakcyjna może zawierać metody abstrakcyjne, poprzedzone słowem kluczowym **abstract;** w miejscu ciała metody abstrakcyjnej występuje średnik; każda jej podklasa musi podawać implementacje tych metod. Każda klasa, która odziedziczy metodę abstrakcyjną, ale jej nie implementuje, staje się klasą abstrakcyjną

✓ brak specyfikatora – klasa dostępna tylko dla klas zdefiniowanych w tym samym pakiecie

✓ **dziedziczenie -** po nazwie klasy wystąpią słowa: **extends** nazwa_superklasy

(czyli klasa dziedziczy zawsze publicznie i tylko od jednej od klasy nazwa_superklasy)

Każda klasa dziedziczy od predefiniowanej klasy Object. Jeżeli w definicji klasy nie występuje słowo **extends,** to oznacza to niejawne wystąpienie w tej definicji słów **extends** Object

✓ i**mplementowanie**- po nazwie klasy wystąpią słowa: **implements** nazwy_interfejsów

(czyli w danej klasie zostaną zdefiniowane metody, zadeklarowane w implementowanych interfejsach. Jeżeli dana klasa implementuje więcej niż jeden interfejs, wtedy nazwy kolejnych interfejsów oddziela się przecinkami. Implementowanie metod kilku interfejsów odpowiada dziedziczeniu wielobazowe w C++

Ciało klasy:

- ✓ zamknięte w nawiasy klamrowe
- ✓ może zawierać zmienne składowe (to jest pola lub może zawierać zmienne instancji)
- ✓ może zawierać zmienne klasowe (statyczne, tj. poprzedzone słowem kluczowym **static**)
- ✓ może zawierać konstruktory (metody o nazwie klasy bez zwracanego typu)
- ✓ może zawierać metody klasowe (nagłówek poprzedzony słowem kluczowym **static**)
- ✓ może zawierać metody zwykłe – można je wywołać, gdy utworzono obiekt
- ✓ nazwa każdej zmiennej składowej, zmiennej klasy, metody lub funkcji klasy musi być poprzedzona nazwą typu podstawowego (**byte**, **short**, **int, long, double, float, char, boolean, void**) lub klasowego
- ✓ przed nazwą typu składowej klasy może wystąpić jeden ze specyfikatorów dostępu:
  - o **private** (dostęp tylko dla elementów klasy - **private int** d;),
  - o **protected** (dostęp tylko w podklasie, nawet jeśli podklasa należy do innego pakietu; nie dotyczy zmiennych klasy)
  - o **public** (dostęp publiczny). Brak specyfikatora oznacza, że dany element jest dostępny tylko dla klas w tym samym pakiecie
- ✓ słowo **final** po specyfikatorze dostępu przed nazwą typu zmiennej wystąpienia lub zmiennej klasy deklaruje jej nie modyfikowalność

  np. **public static final int** stala1 = 10;

  final int stala2= 10;
- ✓ słowo **final** po specyfikatorze dostępu przed nazwą metody oznacza, że nie może ona być redefiniowana w klasie dziedziczącej

  np. **public final void** koncowa_wersja () {/* ... */ } – definicja publicznej metody koncowa_wersja może wystąpić tylko raz w rodzinie klas

2. Tworzenie obiektu

- Dostęp do zmiennych składowych klasy (statycznych) jest możliwy bez tworzenia obiektów tej klasy
  np. **System.out.println**("Dzien dobry, nazywam się Jan Kowalski\n");

- Klasy i interfejsy są typami referencyjnymi.
  Wartościami zmiennych tych typów są referencje (odnośniki) do wartości lub zbiorów wartości reprezentowanych przez te zmienne.
  np. instrukcja ***Random rand;*** jedynie powiadamia kompilator, że będzie używana zmienna ***rand***, której typem jest ***Random*** (brak przydzielonego miejsca w pamięci na taki obiekt)

- Do zmiennej ***rand*** można przypisać dowolny obiekt typu ***Random*** przydzielając mu pamięć za pomocą **new**
  np. **Random rand = new Random();**

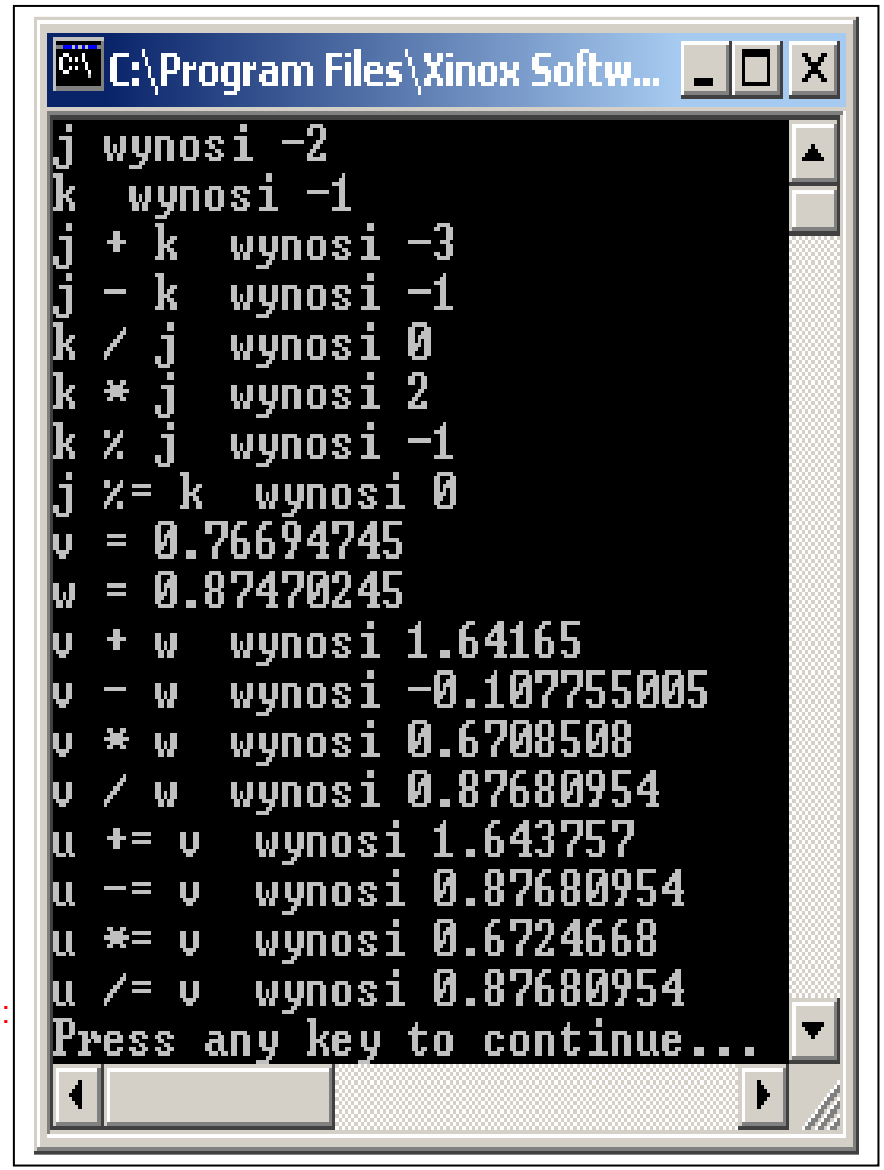
  Argumentem operatora **new** jest generowany przez kompilator konstruktor ***Random()***, który inicjuje obiekt utworzony przez operator **new**. Operator **new** zwraca referencję do tego obiektu, po czym przypisuje go do zmiennej ***rand***.
  np. **Random rand = new Random(20L);**
  Argumentem operatora **new** jest definiowany przez programistę konstruktor **Random(20L)**, który inicjuje obiekt utworzony przez operator **new**. Operator **new** zwraca referencję do tego obiektu, po czym przypisuje go do zmiennej ***rand***.

- Dostęp do elementów klasy uzyskuje się za pomocą operatora kropkowego

1. Operatory arytmetyczne +, -, / ,*, %

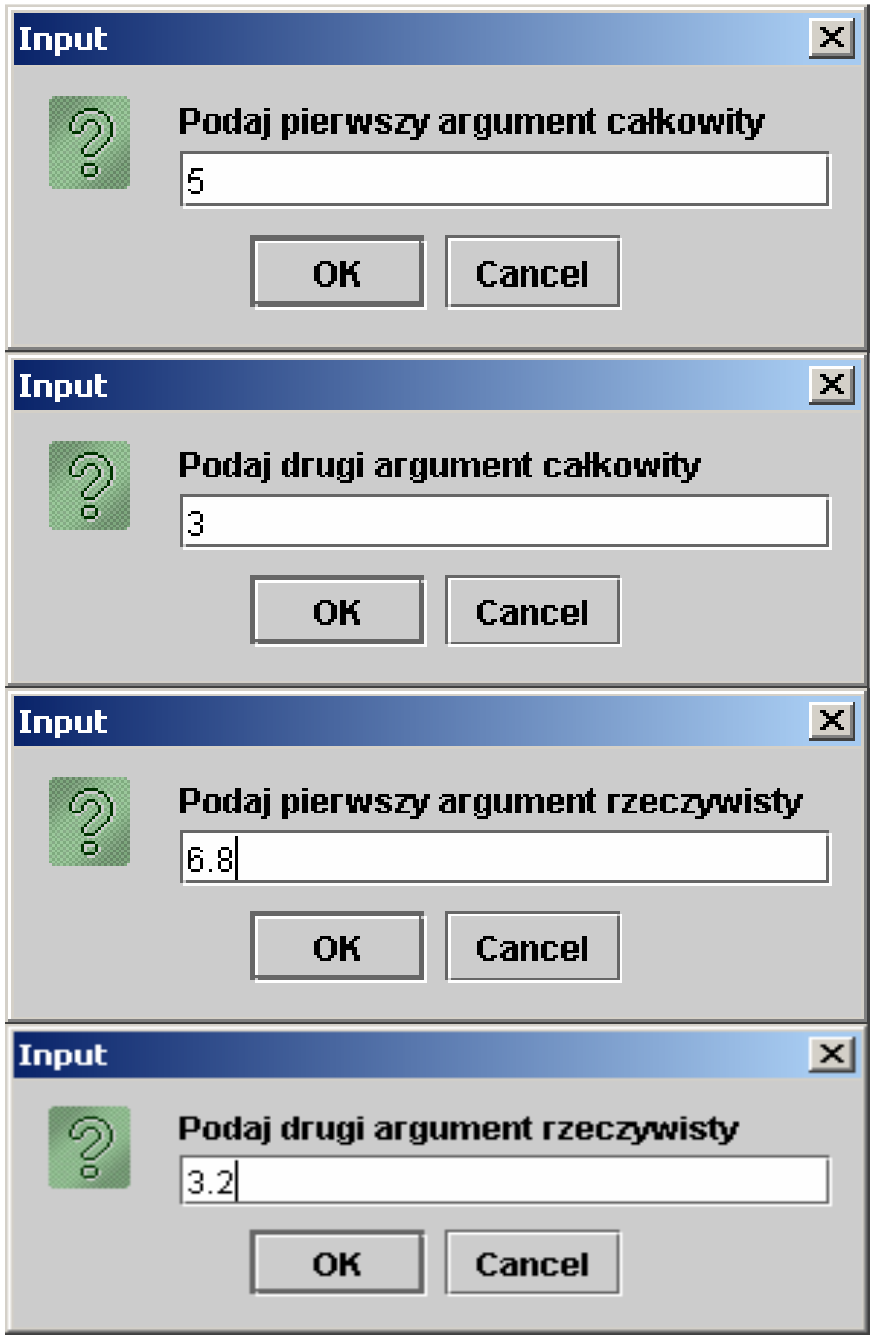
Przykład 1 programu z interfejsem konsolowym

```
import java.util.*;

public class Lab2_1
{  public static void main(String[] args)
   { //  Tworzy  generator liczb losowych, zainicjowany czasem systemowym
     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);
     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);
     j %= k;             s = "j %= k  wynosi " + j;        System.out.println(s);
    // Operacje na argumentach zmniennoprzecinkowych u,v,w
     float u, v, w;                    //losowanie kolejnej wartośći rzexczywistej
     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);
    //  następne wyrażenia są realizowane dla  char, byte, short, int, long,   and double:
     u += v;             s = "u += v  wynosi " + u;        System.out.println(s);
     u -= v;             s = "u -= v  wynosi " + u;        System.out.println(s);
     u *= v;             s = "u *= v  wynosi " + u;        System.out.println(s);
     u /= v;             s = "u /= v  wynosi " + u;        System.out.println(s); } }
```

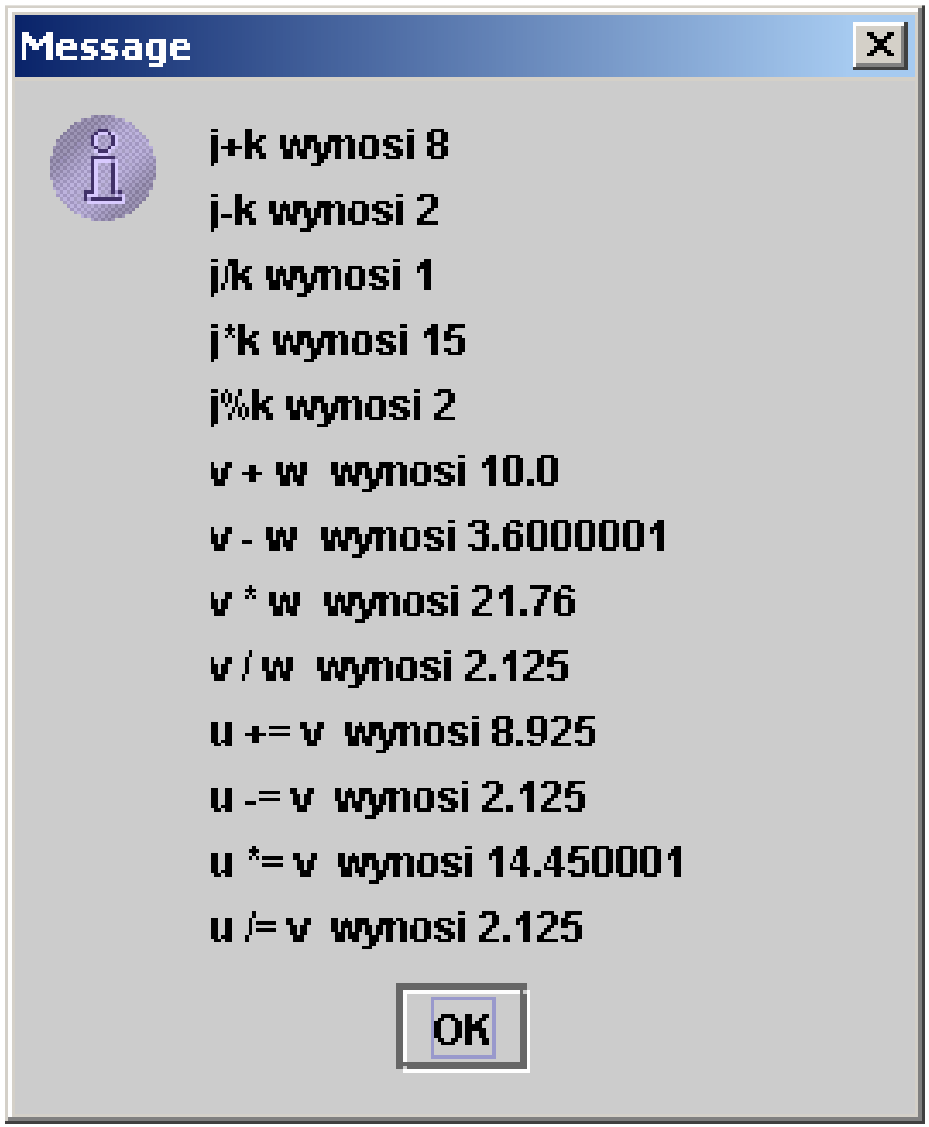


Przykład 1 programu z graficznym interfejsem użytkownika

```
import javax.swing.*;
import java.util.*;

public class Lab2_2
{
  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 5
   s = JOptionPane.showInputDialog(null,
                "Podaj pierwszy argument całkowity");
  //zamiana łąncucha 5 na liczbę 5
    j = Integer.parseInt(s);
  //  pobranie z okienka dialogowego łańcucha 3
   s = JOptionPane.showInputDialog(null,
                "Podaj drugi argument całkowity");
  //zamiana łąncucha 3 na liczbę 3
  k = Integer.parseInt(s);
//definicja zmiennych rzeczywistych u, v w
float u, v, w;
    //  pobranie z okienka dialogowego łańcucha 6.8
s=JOptionPane.showInputDialog(null,
                "Podaj pierwszy argument rzeczywisty");
 //zamiana łąncucha 6.8 na liczbę
 v = Float.parseFloat(s);
   //  pobranie z okienka dialogowego łańcucha 3.2
   s = JOptionPane.showInputDialog(null,
                "Podaj drugi argument rzeczywisty");
//zamiana łąncucha 3.2 na liczbę
 w = Float.parseFloat(s);
```



```
   i = j+k;          s="j+k wynosi "+ i + "\n";
 // dodanie do łańcucha s nowego łąncucha 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);
  }
}
```

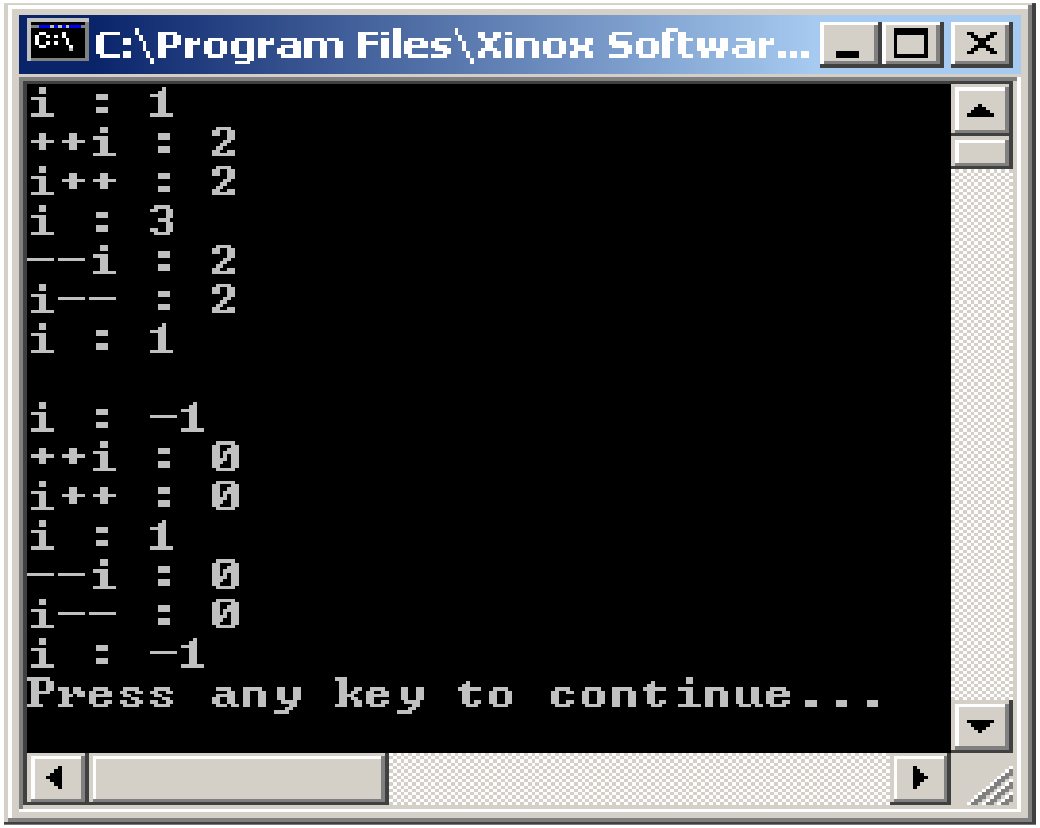


2. Operatory jednoargumentowe + ,-. Operatory inkrementacji przedrostkowej i przyrostkowej

Operatory jednoargumentowe – i plus służą do określenia wartości dodatniej lub ujemnej.
Operatory inkrementacji i dekrementacji przedrostkowej np. ++i lub --i wykonują się najpierw, zanim wykona się wyrażenie, w którym użyto te operatory.
Operatory inkrementacji i dekrementacji przyrostkowej np. i++ lub i-- wykonują się po wykonaniu wyrażenia, w którym użyto te operatory.

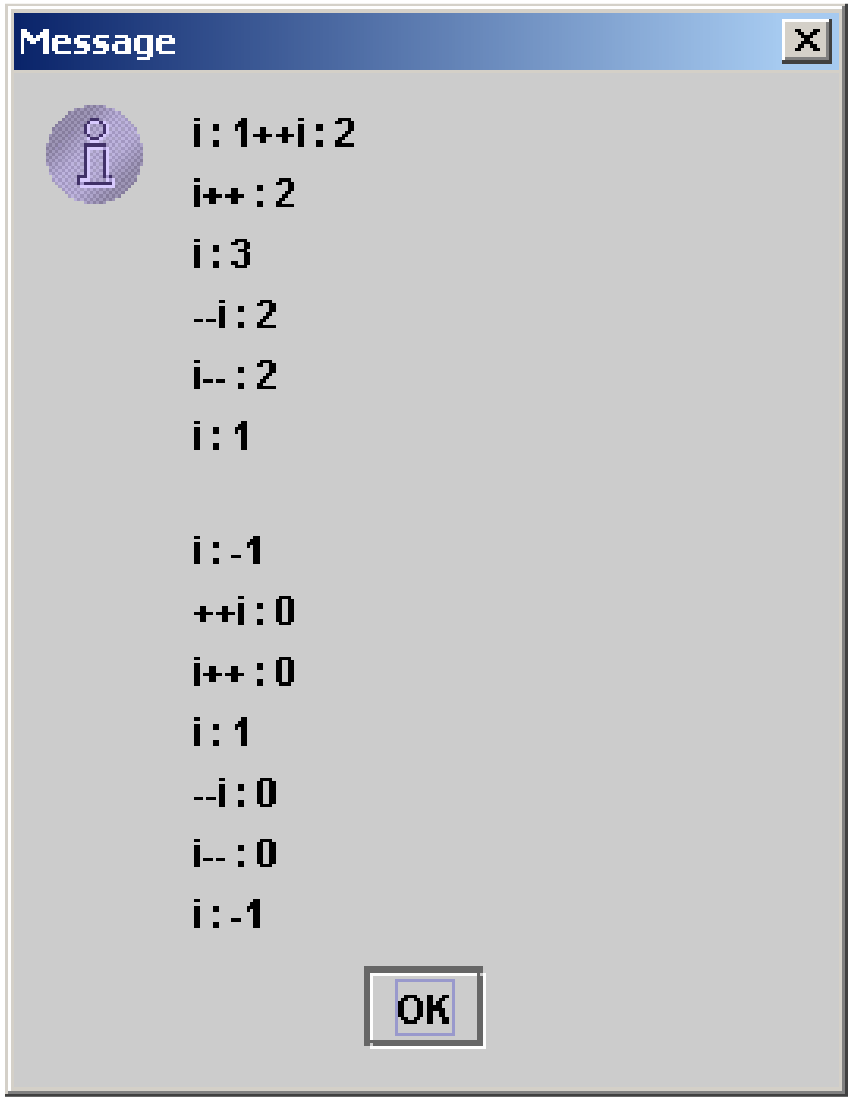
Przykład 2 z interfejsem konsolowym

```
public class  Lab2_3
 {
  public static void main(String[] args)
  {  int i = +1;
    System.out.println("i : " + i);
    System.out.println("++i : " + ++i); // Pre-increment
    System.out.println("i++ : " + i++); // Post-increment
    System.out.println("i : " + i);
    System.out.println("--i : " + --i); // Pre-decrement
    System.out.println("i-- : " + i--); // Post-decrement
    System.out.println("i : " + i);
    i = -1;
    System.out.println("\ni : " + i);
    System.out.println("++i : " + ++i); // Pre-increment
    System.out.println("i++ : " + i++); // Post-increment
    System.out.println("i : " + i);
    System.out.println("--i : " + --i); // Pre-decrement
    System.out.println("i-- : " + i--); // Post-decrement
    System.out.println("i : " + i);
  }
 }
```



Przykład 2 z interfejsem graficznym

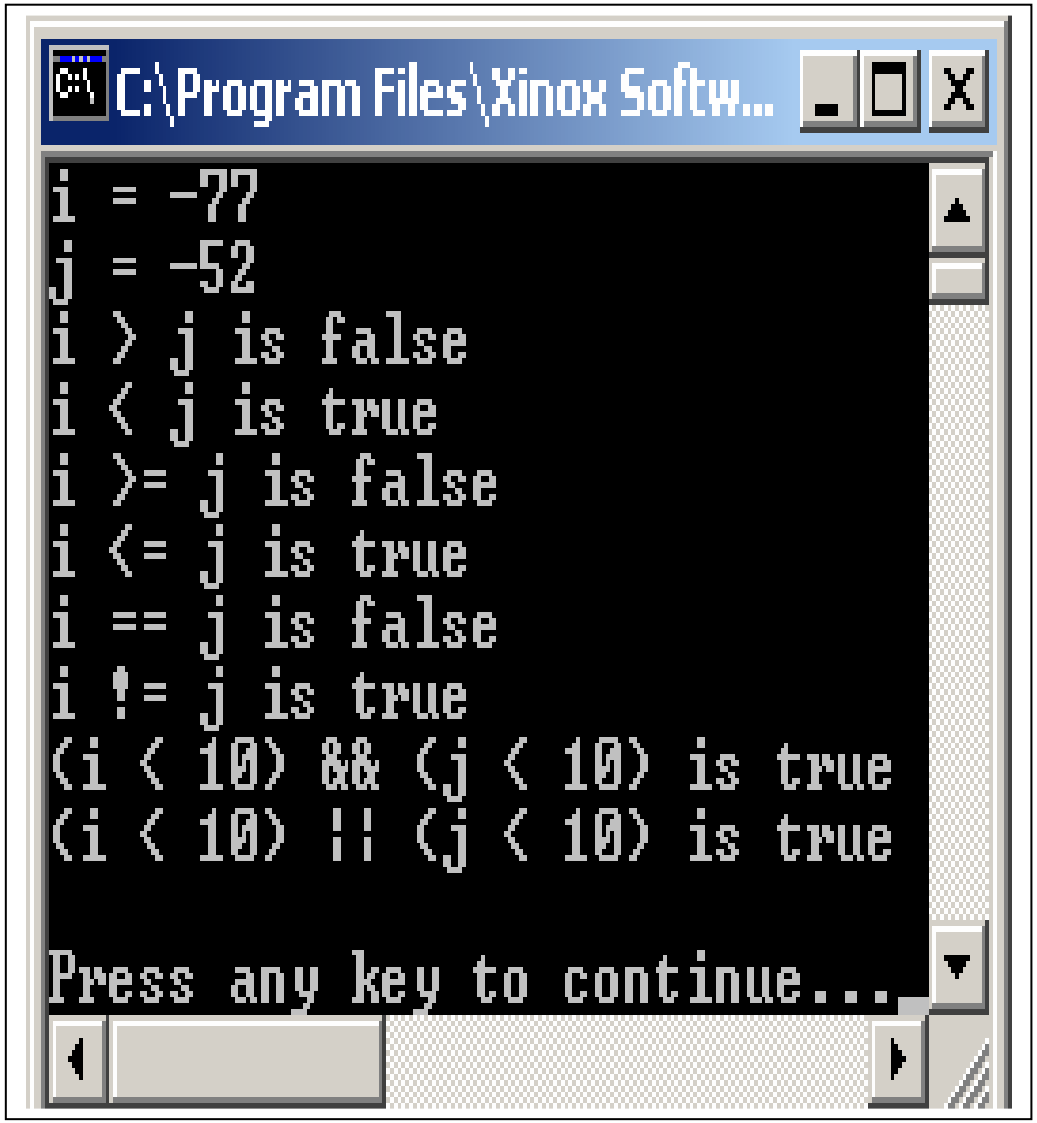
```
import javax.swing.*;
import java.util.*;
public class  Lab2_4
 {
  public static void main(String[] args)
  {
    int i = +1; String s;
    s="i : " + i;
    s+="++i : " + ++i +"\n"; // Pre-increment
    s+="i++ : " + i++ +"\n"; // Post-increment
    s+="i : " + i +"\n";
    s+="--i : " + --i +"\n"; // Pre-decrement
    s+="i-- : " + i-- +"\n"; // Post-decrement
    s+="i : " + i+"\n";
    i = -1;
    s+="\ni : " + i +"\n";
    s+="++i : " + ++i +"\n"; // Pre-increment
    s+="i++ : " + i++ +"\n"; // Post-increment
    s+="i : " + i +"\n";
    s+="--i : " + --i +"\n"; // Pre-decrement
    s+="i-- : " + i-- +"\n"; // Post-decrement
    s+="i : " + i +"\n";
    JOptionPane.showMessageDialog(null,s);
    System.exit(0);
  }
 }
```



3.Operatory relacyjne <>, >, <, >=, <=, ==, i logiczne AND (&&), OR (||) and NOT (!) - wynik działania operatorów jest równy wartości false lub true.

Przykład 3 z interfejsem konsolowym

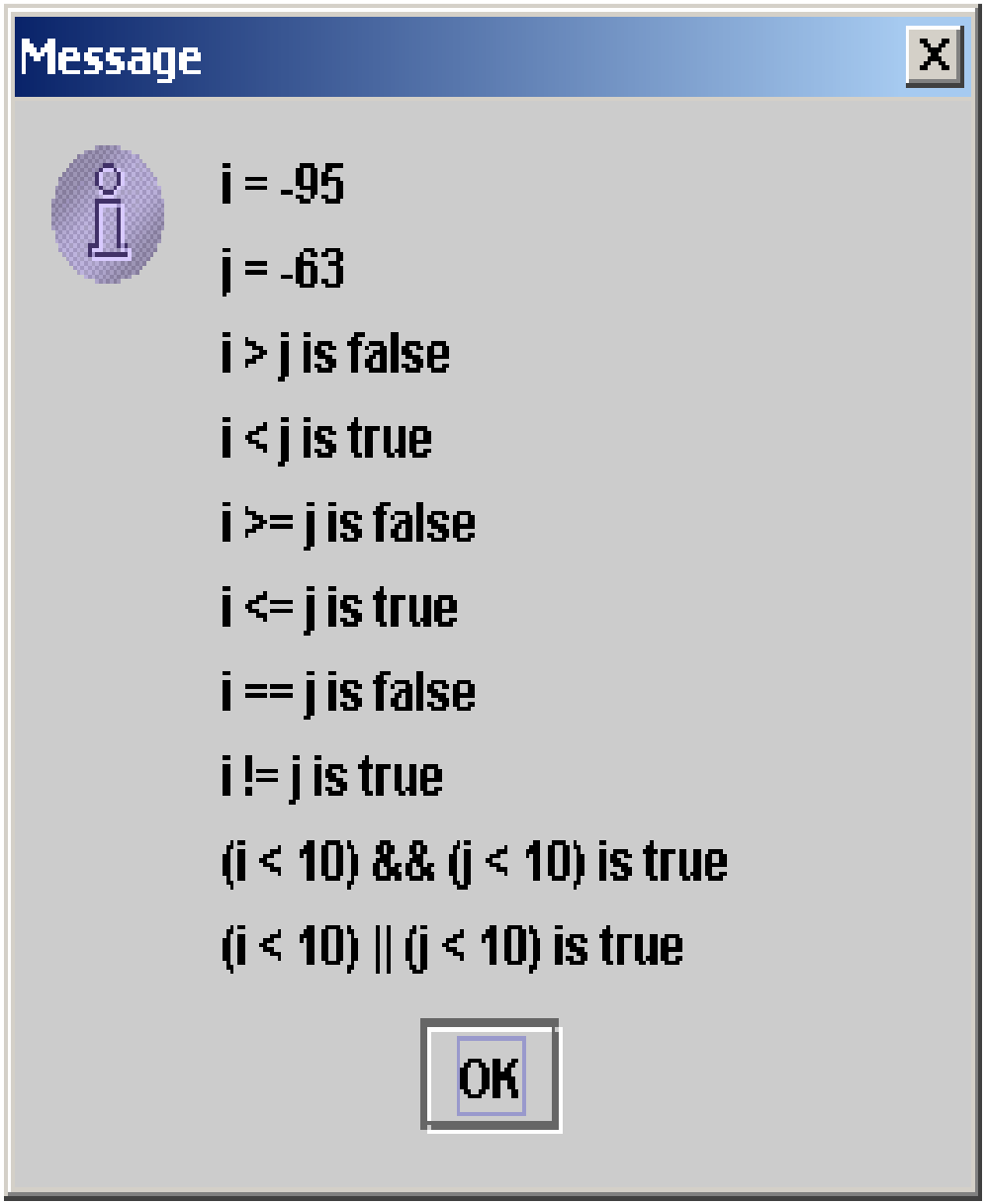
```
import java.util.*;

public class Lab2_5 {
  public static void main(String[] args)
  { Random rand = new Random();
    int i = rand.nextInt() % 100;
    int j = rand.nextInt() % 100;
    String s;
    s = "i = " + i + "\n";
    s+= "j = " + j + "\n";
    s+= "i > j is " + (i > j) + "\n";
    s+= "i < j is " + (i < j) + "\n";
    s+= "i >= j is " + (i >= j) + "\n";
    s+= "i <= j is " + (i <= j)+ "\n";
    s+= "i == j is " + (i == j)+ "\n";
    s+= "i != j is " + (i != j)+ "\n";

    // Wartość typu int nie jest wartością logiczną
    // w Javie
   //! System.out.println(s) ("i && j is " + (i && j));
   //! System.out.println ("i || j is " + (i || j));
   //! System.out.println ("!i is " + !i);

    s+= "(i < 10) && (j < 10) is " + ((i < 10) && (j < 10))+ "\n";
    s+="(i < 10) || (j < 10) is "  + ((i < 10) || (j < 10))+ "\n";
    System.out.println(s);
  }
}
```



Przy wyświetlaniu wartość typu logicznego jest równa true lub false.

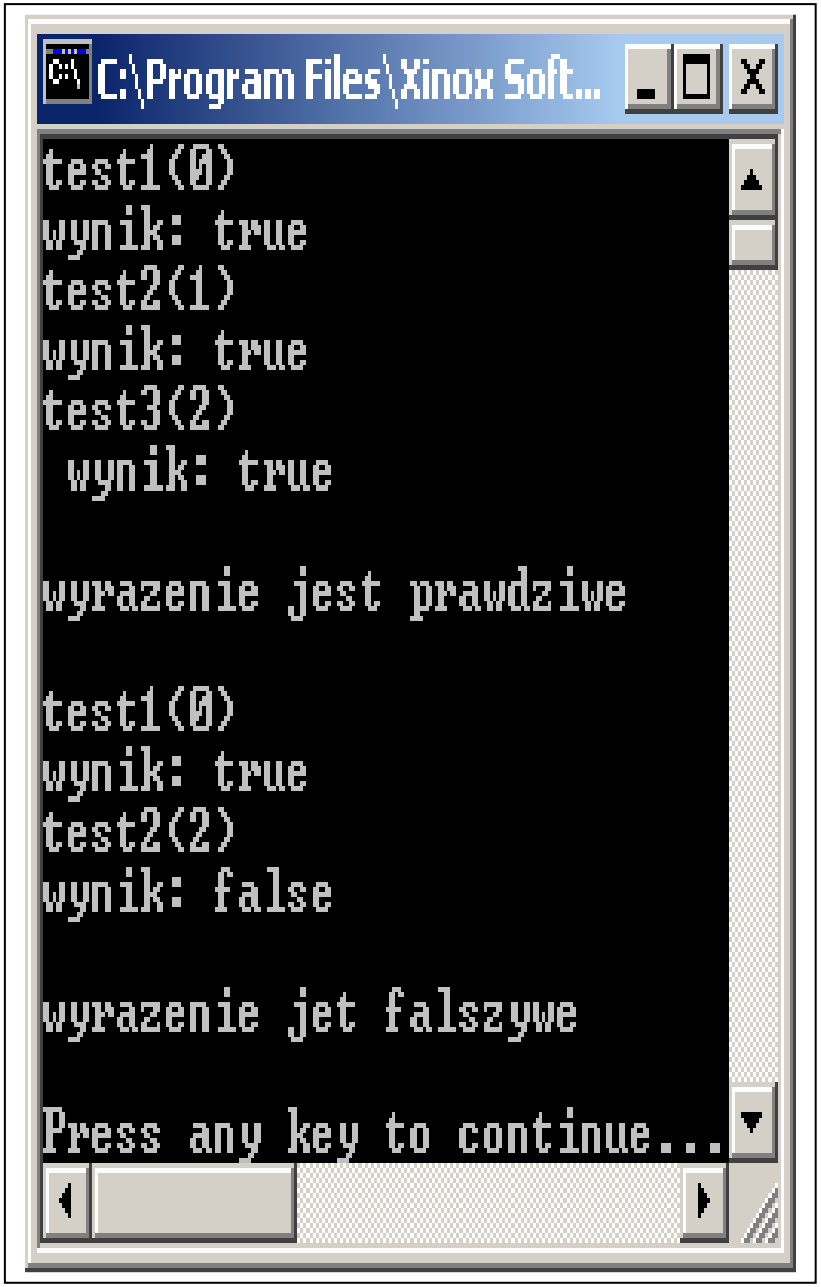
Przyklad 3 z interfejsem graficznym

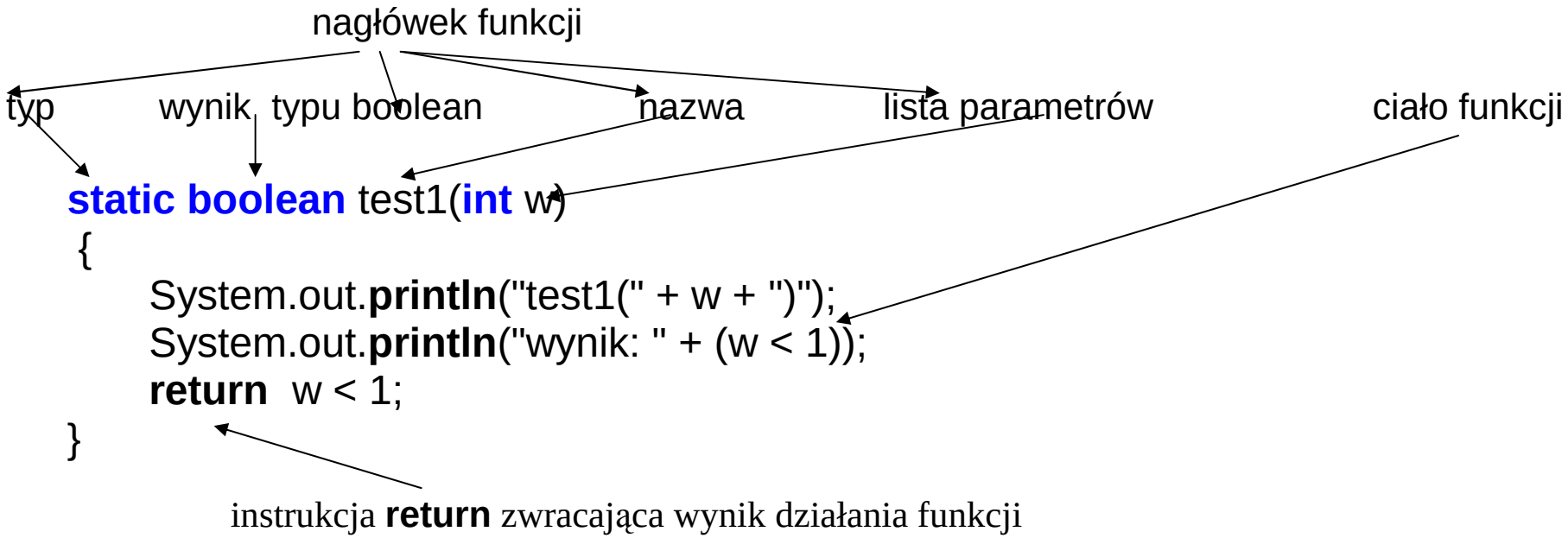
```
import java.util.*;
import javax.swing.*;

public class Lab2_6 {
  public static void main(String[] args)
  {
   Random rand = new Random();
    int i = rand.nextInt() % 100;
    int j = rand.nextInt() % 100;
    String s;
    s= "i = " + i + "\n";
    s+= "j = " + j + "\n";
    s+= "i > j is " + (i > j) + "\n";
    s+= "i < j is " + (i < j) + "\n";
    s+= "i >= j is " + (i >= j) + "\n";
    s+= "i <= j is " + (i <= j)+ "\n";
    s+= "i == j is " + (i == j)+ "\n";
    s+= "i != j is " + (i != j)+ "\n";


   s+="(i < 10) && (j < 10) is "+ ((i < 10) && (j < 10))+ "\n";
   s+="(i < 10) || (j < 10) is "   + ((i < 10) || (j < 10))+ "\n";

JOptionPane.showMessageDialog(null,s);
   System.exit(0);
  }
}
```

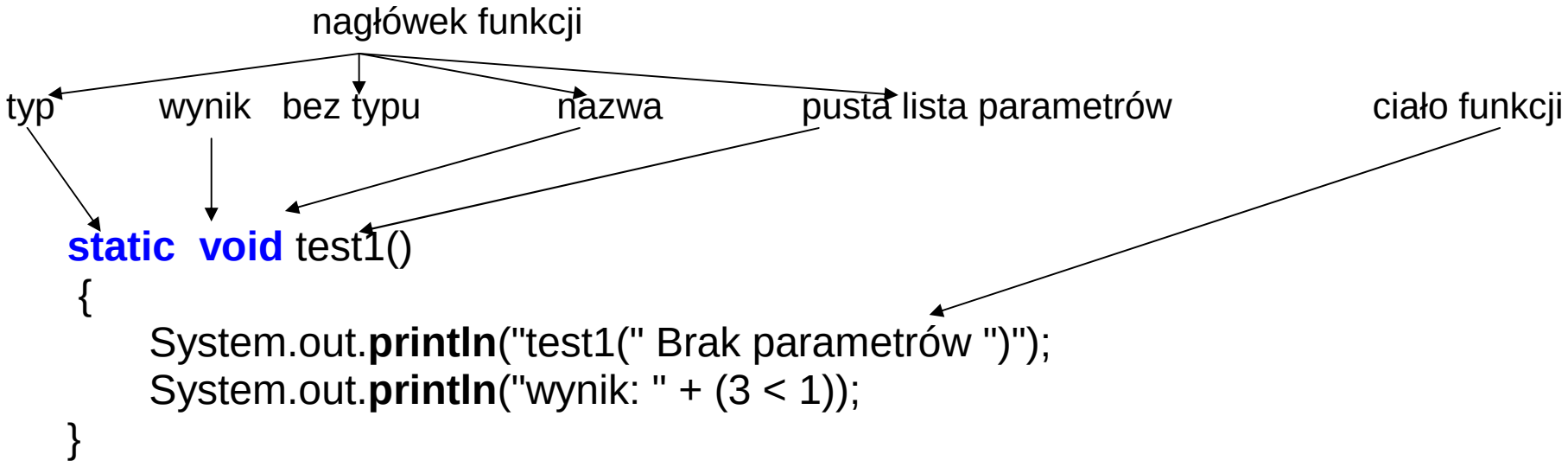


4. Funkcje statyczne, skrócone obliczanie wartości wyrażeń logicznych

```
public class Lab2_7
 {
  static boolean test1(int w)
  {    System.out.println("test1(" + w + ")");
        System.out.println("wynik: " + (w < 1));
        return  w < 1;
  }

  static boolean test2(int  w)
  {    System.out.println("test2(" +  w + ")");
       System.out.println("wynik: " + ( w < 2));
       return  w < 2;
  }

  static boolean test3(int w)
  {    System.out.println("test3(" + w  + ")");
       System.out.println(" wynik: " + ( w < 3));
       return w < 3;
  }

  public static void main(String[] args)
  { if(test1(0) && test2(1) && test3(2))
          System.out.println("\nwyrazenie jest prawdziwe\n");
    else
          System.out.println("\nwyrazenie jet falszywe\n");

    if(test1(0) && test2(2) && test3(2))
        System.out.println("\nwyrazenie jest prawdziwe\n");
    else
          System.out.println("\nwyrazenie jet falszywe\n");
  }}
```



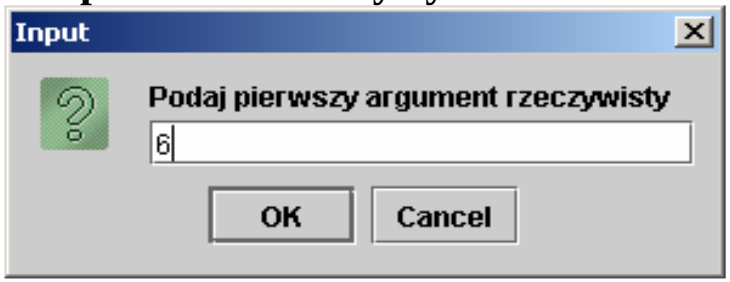


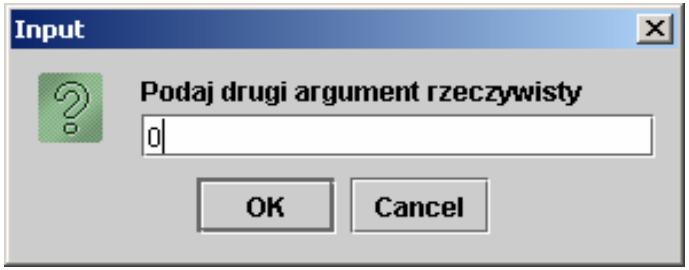


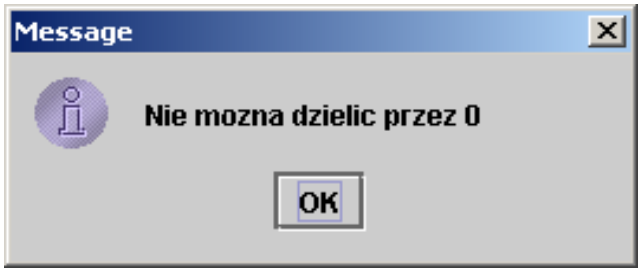
void – brak typu

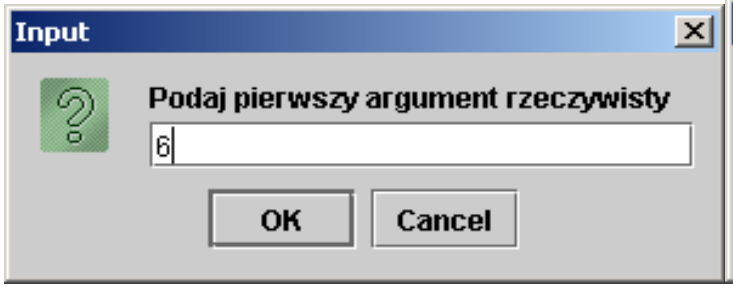
W przypadku obliczania iloczynu logicznego przerywa się obliczanie wartości jeśli jedno z podwyrażeń jest fałszywe.4.

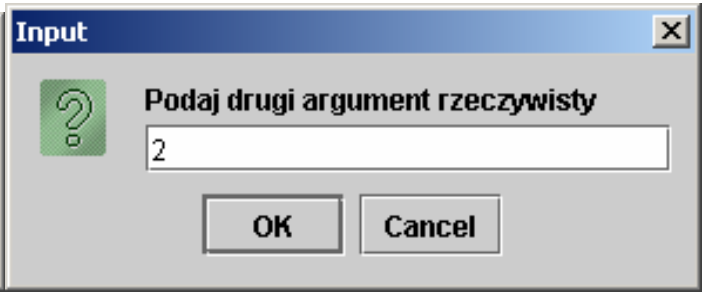
5. Operator alternatywy ? :

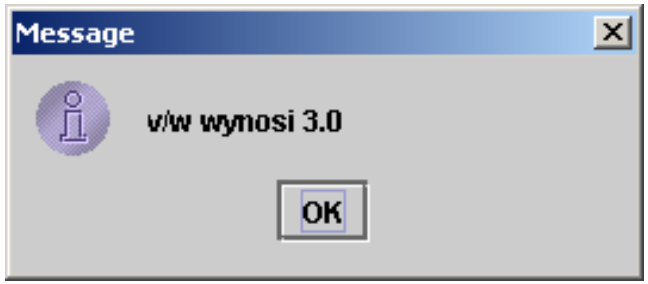













```
import javax.swing.*;
import java.util.*;
public class Lab2_8
 {
  public static void main(String[] args)
   {    float v,w; String s;
        s=JOptionPane.showInputDialog(null,
              "Podaj pierwszy argument rzeczywisty");
        v=Float.parseFloat(s);
        s=JOptionPane.showInputDialog(null,
              "Podaj drugi argument rzeczywisty");
        w=Float.parseFloat(s);
        s= w==0 ? " Nie mozna dzielic przez 0" ":
                    "v/w wynosi" + v/w;
        JOptionPane.showMessageDialog(null,s);
     System.exit(0);
   } }
```

```
import javax.swing.*;
import java.util.*;
public class Lab2_9
 {
  public static void main(String[] args)
   {    float v,w; String s;
        s=JOptionPane.showInputDialog(null,
              "Podaj pierwszy argument rzeczywisty");
        v=Float.parseFloat(s);
        s=JOptionPane.showInputDialog(null,
              "Podaj drugi argument rzeczywisty");
        w=Float.parseFloat(s);
        if (w==0)     s="Nie mozna dzielic przez 0";
        else    s="v/w wynosi" + v/w;
        JOptionPane.showMessageDialog(null,s);
        System.exit(0);
   } }
```

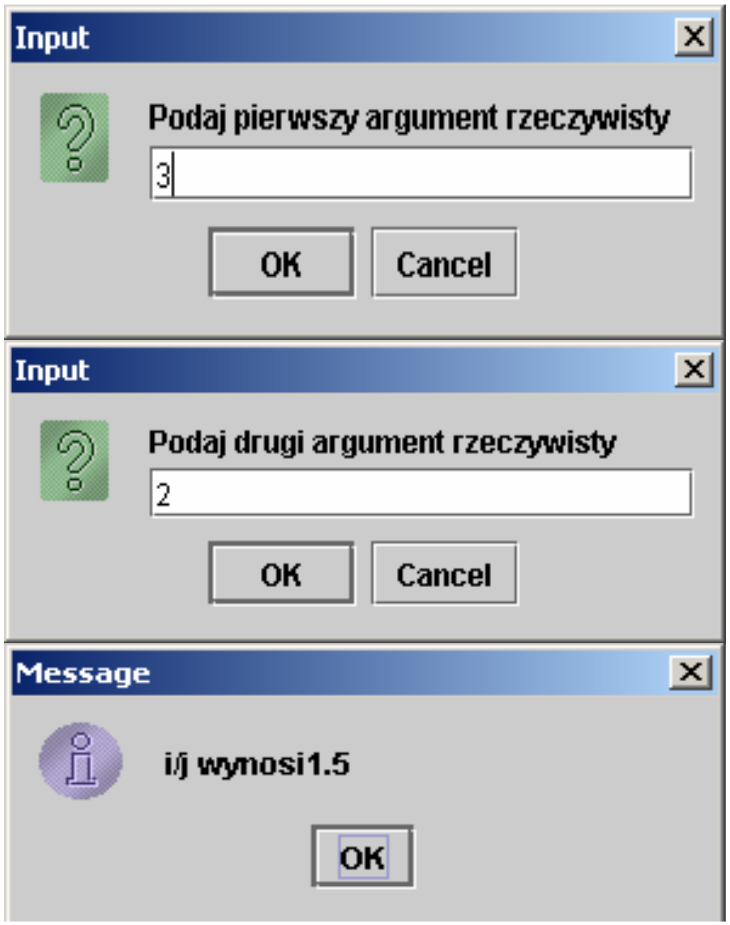
6. Operacje na łańcuchach – operator +, = oraz +=

Patrz przykłady. Zastosowanie w wyrażeniu podłańcucha wymusza konwersje pozostałych elementów na łańcuchy.

```
float v=3, w=2; String s;
        s="v/w wynosi" + v/w;
```

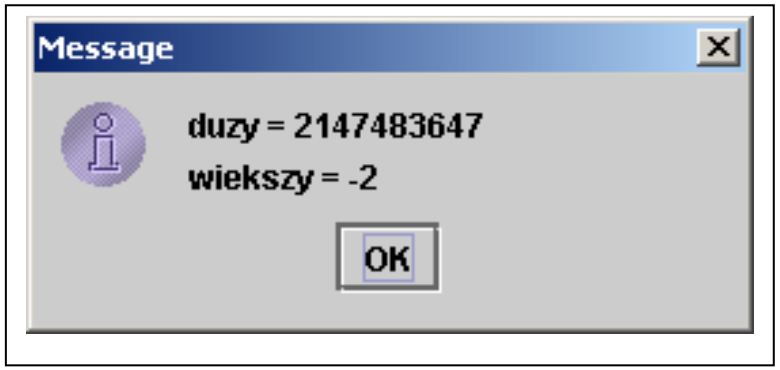
7. Rzutowanie

```
import javax.swing.*;
import java.util.*;
public class Lab2_10
 {
   public static void main(String[] args)
    {
      int i,j; String s;
          s=JOptionPane.showInputDialog(null,
                  "Podaj pierwszy argument rzeczywisty");
          i=Integer.parseInt(s);
          s=JOptionPane.showInputDialog(null,
                  "Podaj drugi argument rzeczywisty");
          j=Integer.parseInt(s);
          if (j==0)
           s = "Nie mozna dzielic przez 0";
          else //rzutowanie do wartośći rzeczywistej,
                //aby podzielić z resztą
            s = "i/j wynosi" + i/(float)j;

          JOptionPane.showMessageDialog(null,s);
      System.exit(0);
    }
 }
```



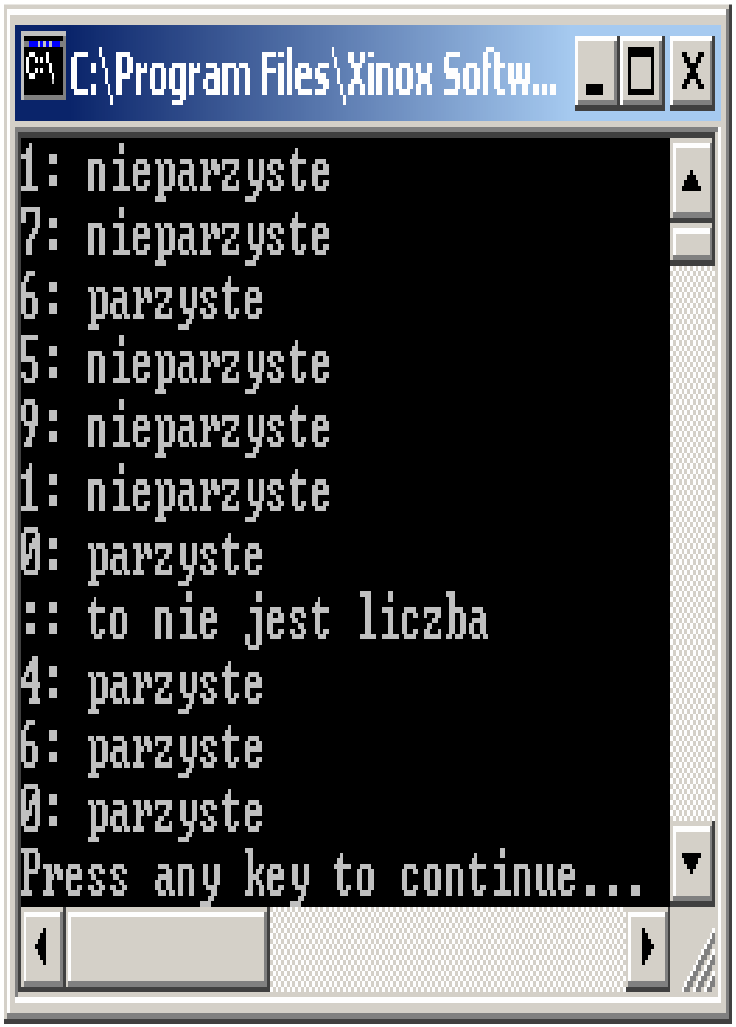
8. Przepełnienie

```
import javax.swing.*;
import java.util.*;

public class Lab2_11
 {
  public static void main(String[] args)
  {
    int duzy = 0x7fffffff;        //  maksymalna wartość int
    int  wiekszy =  duzy * 2;
    String s = "duzy = " + duzy + "\n";
    s += "wiekszy = " +  wiekszy + "\n";
    JOptionPane.showMessageDialog(null,s);

 /* Podczas kompilacji zgłaszany jest błąd przepełnienia  dla liczb rzeczywistych

    float  wielki = 3.4E+38;         //  maksymalna wartość float
    System.out.println("wielki = " + wielki);
    int   jeszcze_wiekszy=  wielki * 4;
    System.out.println (" jeszcze_wiekszy = " +  jeszcze_wiekszy);
 */
   System.exit(0);
  }
}
```



9. Instrukcje switch, do while

```
public class Lab2_12 {
  public static void main(String[] args)
  { int i=1;
    do  //losowanie kodu ASCCI cyfry
    { char c = (char)(Math.random() * 11 + '0');
      System.out.print(c + ": ");
      switch(c)
      {case '0': i++;
       case '2':
       case '4':
       case '6':
       case '8':
              System.out.println("parzyste");
              break;
       case '1':
       case '3':
       case '5':
       case '7':
       case '9':
              System.out.println("nieparzyste");
              break;
       default:
              System.out.println("to nie jest liczba");
      }
    }while (i!=3);
  }
}
```



- **break** przerywa instrukcję **switch**.
- w przypadku braku **break** instrukcja przechodzi do następnego **case** po wybranym wcześniej bez słowa **break**.
- **default** jest wybierane, gdy wartość zmiennej c nie jest równa żadnej wartości przy **case**