Języki i metody programowania – Java
INF302W
Wykład 3 (część 2)

Autor

Dr inż. Zofia Kruczkiewicz

STRUKTURA WYKŁADU

1. Systemowe struktury danych typu vector, stack, hashmap(R-1, EL-1)

1. Interfejsy – deklaracja metod abstrakcyjnych, typu default i static

Interfejs (interface): abstrakcyjny typ danych, tworzący hierarchię typów powiązanych **dziedziczeniem wielobazowym**, które deklarują abstrakcyjne operacje na elementach umieszczonych w pojemnikach, niezależnie od implementacji.

Atrybuty zadeklarowane w interfejsach: domyślnie są typu **public** i **final**

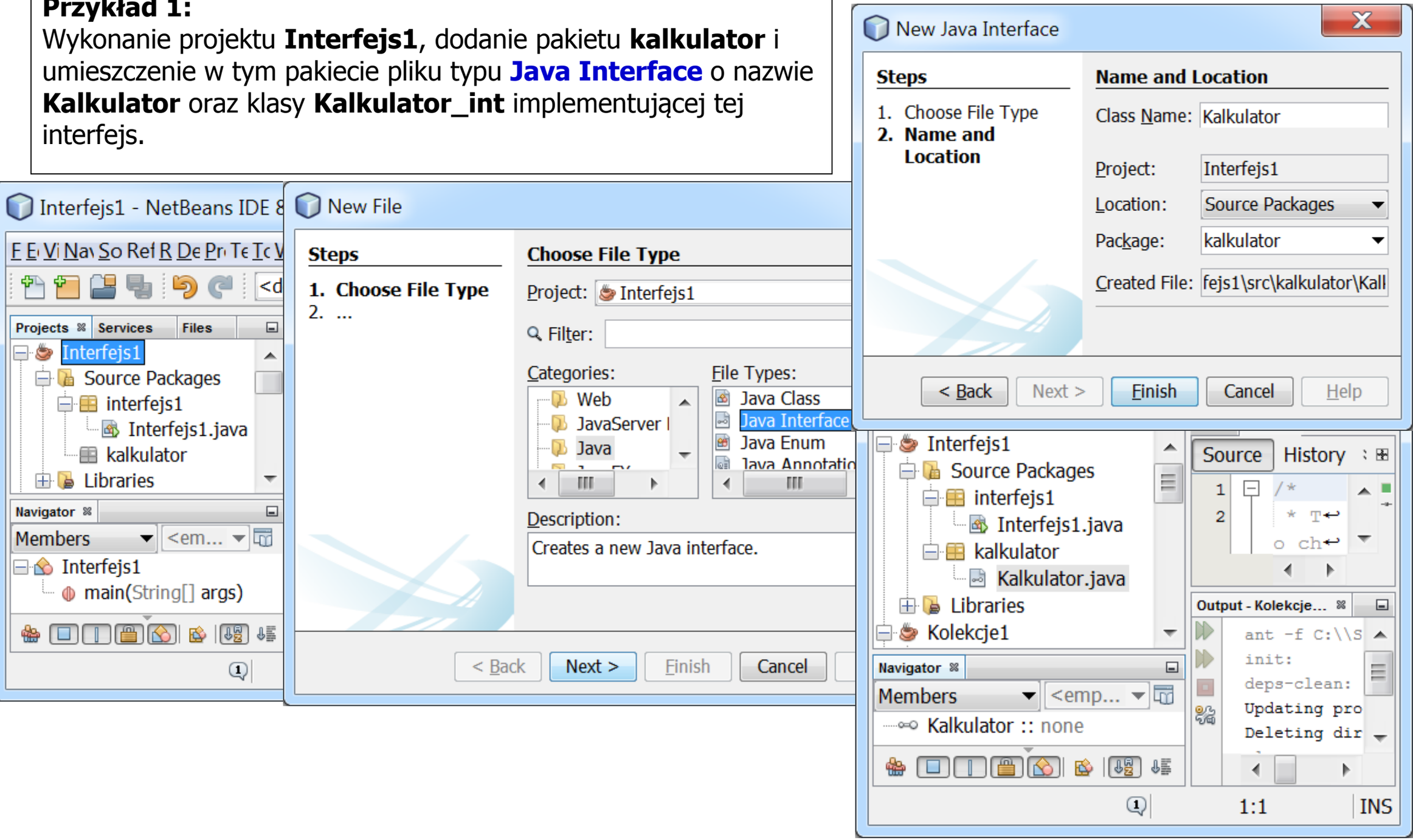
Metody zadeklarowane w interfejsach są domyslnie zadeklarowane jako public i mogą być:

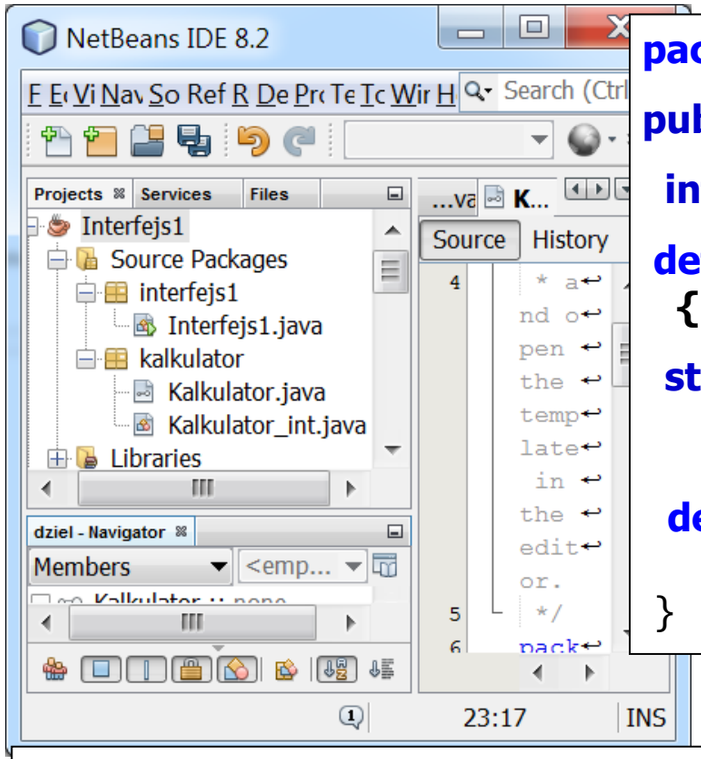
- **abstrakcyjne:** dziedziczone przez inne interfejsy i wymagają implementacji w klasach powiązanych implementacją (z wykorzystaniem słowa kluczowego **implements**). **Jedna klasa może implementować metody wielu interfejsów, których nazwy są podane w liście po słowie implements i oddzielone są przecinkami**
- **ze slowem kluczowym default**: metody te są gotowe do użycia przez klasy implementujące i mogą być przez nie przedefiniowane. Metody default podczas dziedziczenia przez inne interfejsy mogą być:

  ✓ dziedziczone,
  ✓ mogą być przedefiniowane jako abstrakcyjne
  ✓ mogą być przedefiniowane jako default (bez możliwości użycia słowa kluczowego super)

- **ze slowem kluczowym static**: metody te ułatwiają definicję metod typu default w innych, również niepowiązanych, interfejsach oraz klasach. Należy wtedy użyć nazwy metody **static** podając **nazwa_interfesu.nazwa_metody_static**.

Przykład 1:
Wykonanie projektu **Interfejs1**, dodanie pakietu **kalkulator** i umieszczenie w tym pakiecie pliku typu **Java Interface** o nazwie **Kalkulator** oraz klasy **Kalkulator_int** implementującej tej interfejs.






```java
package kalkulator;

public interface Kalkulator {

 int dodaj(int a, int b);

 default int odejmij(int a, int b)
  { return a - b; }

 static int podziel(int a, int b) {
      return a / b; }

 default int dziel(int a, int b) {
     return podziel(a, b);  }
}
```

```java
package kalkulator;

public class Kalkulator_int implements Kalkulator
{
    int wynik;

   @Override
    public int dodaj(int a, int b) {
       return wynik = a + b;   }

    public int podziel(int a, int b) {
      return wynik = Kalkulator.podziel(a, b) / 2;
    }
}
```

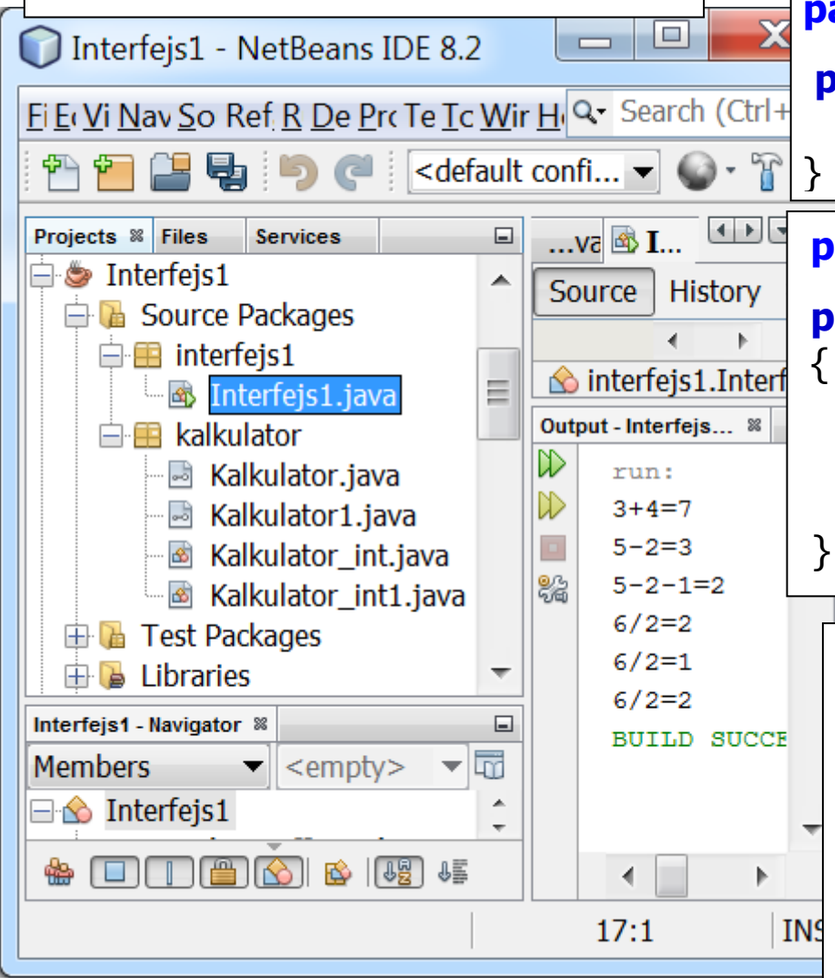
```java
package interfejs1;

import kalkulator.Kalkulator;
import kalkulator.Kalkulator_int;

public class Interfejs1 {
 public static void main(String[] args) {
   Kalkulator kalkulator = new Kalkulator_int();
   System.out.println("3+4=" + kalkulator.dodaj(3, 4));
   System.out.println("5-2=" + kalkulator.odejmij(5, 2));
   System.out.println("6/3=" + kalkulator.dziel(6, 3));
   System.out.println("(6/3)/2=" + ((Kalkulator_int) kalkulator).podziel(6,3));
   System.out.println("6/3=" + Kalkulator.podziel(6, 3));
  }
}
```

```
run:
3+4=7
5-2=3
6/3=2
(6/3)/2=1
6/3=2
BUILD SUCCESSFUL (total time: 0 seconds)
```

Przykład 2: cd przykładu 1




```
run:
3+4=7
5-2=3
5-2-1=2
6/3=2
(6/3)/2=1
6/3=2
BUILD SUCCESSFUL (total time: 0 seconds)
```

```java
package kalkulator;

public interface Kalkulator1 extends Kalkulator{
  int odejmij_(int a, int b);
}
```

```java
package kalkulator;

public class Kalkulator_int1 extends Kalkulator_int implements Kalkulator1
{
  @Override
  public int odejmij_(int a, int b)  {
    return odejmij(a,b)-1; }
}
```

kalkulator.Kalkulator1

```
public int odejmij_(int a, int b)
```

Ctrl+Alt+Click Navigates to Implementing Methods

kalkulator.Kalkulator

```
public int odejmij(int a, int b)
```

Ctrl+Alt+Click Navigates to Overriding Methods

```java
package interfejs1;

import kalkulator.Kalkulator;
import kalkulator.Kalkulator1;
import kalkulator.Kalkulator_int1;

 public class Interfejs1 {

  public static void main(String[] args) {
   Kalkulator1 kalkulator = new Kalkulator_int1();
   System.out.println("3+4=" + kalkulator.dodaj(3, 4));
   System.out.println("5-2=" + kalkulator.odejmij(5, 2));
   System.out.println("5-2-1=" + kalkulator.odejmij_(5, 2));
   System.out.println("6/3=" + kalkulator.dziel(6, 3));
   System.out.println("(6/3)/2=" + ((Kalkulator_int1) kalkulator).podziel(6, 3));
   System.out.println("6/3=" + Kalkulator.podziel(6, 3));  }
}
```

2. Przechowywanie obiektów w pamięci programu - pojemniki

Pojemnik (kolekcja, kontener) jest obiektem, który przechowuje wiele elementów typu klasowego w jednej jednostce.

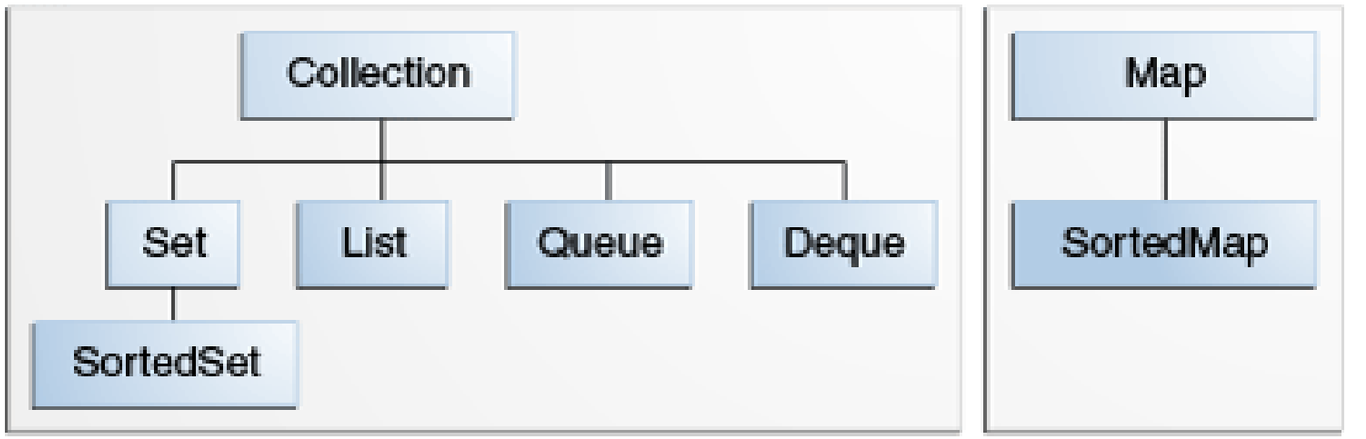
Definicja środowiska do przetwarzania pojemników (Collection Framework):

- **Interfejsy (Interfaces):** abstrakcyjne typy danych, tworzące hierarchię typów powiązanych dziedziczeniem, które deklarują abstrakcyjne operacje na elementach umieszczonych w pojemnikach niezależnie od implementacji.
- **Implementacje (Implementations):** Zdefiniowano klasy (abstrakcyjne i zwykłe), które implementują w różny sposób metody abstrakcyjnych pojemników (interfejsów)
- **Algorytmy (Algorithms):** Zastosowano wydajne algorytmy wyszukiwania, sortowania itp. do operacji na danych umieszczonych w różnych typach zaimplementowanych pojemników. Zróżnicowanie algorytmów osiągnięto za pomocą **polimorfizmu**.

Zalety pojemników

- **Proste zastosowanie** w programach dla różnych typów elementów umieszczanych w pojemnikach dzięki zastosowaniu polimorfizmu narzucającemu cechy przechowywanych elementów
- **Poprawiają szybkość działania programów i ich jakość** (odpwiednio dobrane struktury i algorytmy danych)
- **Wprowadzają standard** w obsłudze różnych typów pojemników (rola interfejsów)
- **Ograniczają wysiłek przy poznawaniu** kolejnych pojemników (rola interfejsów)
- **Ograniczają wysiłek przy tworzeniu** nowych pojemników dzięki wprowadzeniu systemu interfejsów
- **Wprowadzają wieloużywalność** oprogramowania

Rodzina interfejsów określających typy pojemników



1) Interfejsy kolekcji (***Collection***) –gromadzą elementy obiektowe

1.1) (***List***) **- z** możliwością powtarzania wartości elementów

1.2) (***Set***) - bez możliwości powtarzania wartości elementów i z możliwością sortowania elementów ***(SortedSet)***

1.3) (***Quene***) – oprócz operacji dziedziczonych od interfesu Collection wprowadza dodatkowe operacje wstawiania (np w porządku FIFO), usuwania oraz przeszukiwania

1.4) (**Deque**) – oprócz operacji dziedziczonych od interfesu Collection wprowadza dodatkowe operacje wstawiania (np w porządku FIFO lub LIFO), usuwania i wyszukiwania na obu końcach kolekcji.

2) Mapy (*Map*) – gromadzą dane jako pary: klucz i odpowiadający mu element (obiektowy). Klucz nie może się powtarzać. Mapy mogą być wielowymiarowe, podobnie jak tablice i mogą sortować elementy ***(SortedMap)***

3. Implementacje pojemników

Interfaces	Hash table Implementations	Resizable array Implementations	Tree Implementations	Linked list Implementations	Hash table + Linked list Implementations
Set	**HashSet**		TreeSet		LinkedHashSet
List		**ArrayList**		LinkedList	
Queue				**LinkedList**	
Deque		**ArrayDeque**			
Map	**HashMap**		TreeMap		LinkedHashMap

Framework kolekcji Javy zawiera najczęściej używane implementacje interfejsów pojemników:

- interface Set - **HashSet**
- interface List - **ArrayList**
- interface Map - **HashMap**
- interface Queue - **LinkedList**
- interface Deque - **ArrayDeque**

Obiekty typów definiowanych przez użytkownika muszą być przystosowane do przechowywania w pojemnikach:

- wtedy mogą być wyszukiwane, sortowane, usuwane itp. dzięki przedefiniowaniu metod ***equals*** oraz ***hashCode***, dziedziczonych domyślnie po klasie ***Object***
- i implementacji metody ***compareTo*** z interfejsu ***Comparable***.

```
class Klasa_uzytkownika  extends Object  implements  Comparable<Klasa_uzytkownika>

 public interface Comparable<T>
     {
          public int compareTo(T o);
     }
 public class Object
 {  //..............
```

boolean	**equals**(Object obj)
int	**hashCode**()

```
 }
```

int	**compareTo**(Klasa_uzytkownika innyobiekt)	Porównuje dwa obiekty wg. koncepcji programisty.
int	**hashCode**()	Zwraca kod charakteryzujący dany typ użytkownika.
boolean	**equals**(Object anObject)	Porównuje atrybuty obiektu wywołującego metodę z atrybutami obiektu przekazanego do metody.

Wymagania stawiane programistom dotyczące implementacji interfejsów pojemników

1. Zapewnienie utrwalania danych w zewnętrznej bazie danych
2. Konieczność dopasowania spsosobu przechowywania danych w pojemnikach do potrzeb programu
3. Wybór wydajności przetwarzania danych – każdy z pojemników musi być zastosowany w dobranym kontekście użycia wynikającym np czy kluczową czynnością jest wyszukiwanie, usuwanie lub dodawanie

wg [Thinking in Java, Second Edition Bruce Eckel]

Type	**Get**	**Iteration**	**Insert**	**Remove**
array	1430	3850		
ArrayList	3070	12200	500	46850
LinkedList	16320	9110	110	60
Vector	4890	16250	550	46850

Type	**Test size**	**Add**	**Contains**	**Iteration**
	10	138.0	115.0	187.0
TreeSet	100	189.5	151.1	206.5
	1000	150.6	177.4	40.04
	10	55.0	82.0	192.0
HashSet	100	45.6	90.0	202.2
	1000	36.14	106.5	39.39

Type	Test size	Put	Get	Iteration
	10	143.0	110.0	186.0
TreeMap	100	201.1	188.4	280.1
	1000	222.8	205.2	40.7
	10	66.0	83.0	197.0
HashMap	100	80.7	135.7	278.5
	1000	48.2	105.7	41.4
	10	61.0	93.0	302.0
Hashtable	100	90.6	143.3	329.0
	1000	54.1	110.95	47.3

4. Należy wykorzystać otwarcia na implementację nowych sposobów przetwarzania
5. Należy uwzględnić możliwość przenoszenia danych w pojemnikach

Proces implementacji interfejsów i klas abstrakcyjnych

1. **Należy wybrać właściwy typ klasy abstrakcyjnej** reprezentujacej właściwy typ pojemnika
2. **Należy zaimplementować metody abstrakcyjne wybranej klasy abstrakcyjnej** (podane w dokumentacji). Jeżeli algorytm przetwarzania realizowany przez metody klasy abstrakcyjnej musi byc zmodyfikowany, nalezy przedefiniowac te metody.
3. **Należy przetestować i jeśli to konieczne, należy debagować** wykonany kod.
4. **Należy skoncentrować się na wydajności przetwarzania danych w wykonanej kolekcji**. W dokumentacji klasy abstrakcyjnej umieszczona jest informacja dotycząca wydajności zdefiniowanych metod w tej klasie i dziedziczone w klasie implementującej. Jeśli są zbyt wolne, nalezy je przedefiniowac i porównac wydajnosc metody przed przedefiniowaniem i po przedefiniowaniu. Wysilek zwiazany z przedefiniowaniem i eksperymentami powinien wynikac z potrzeb program.
5. **Algorytmy:**

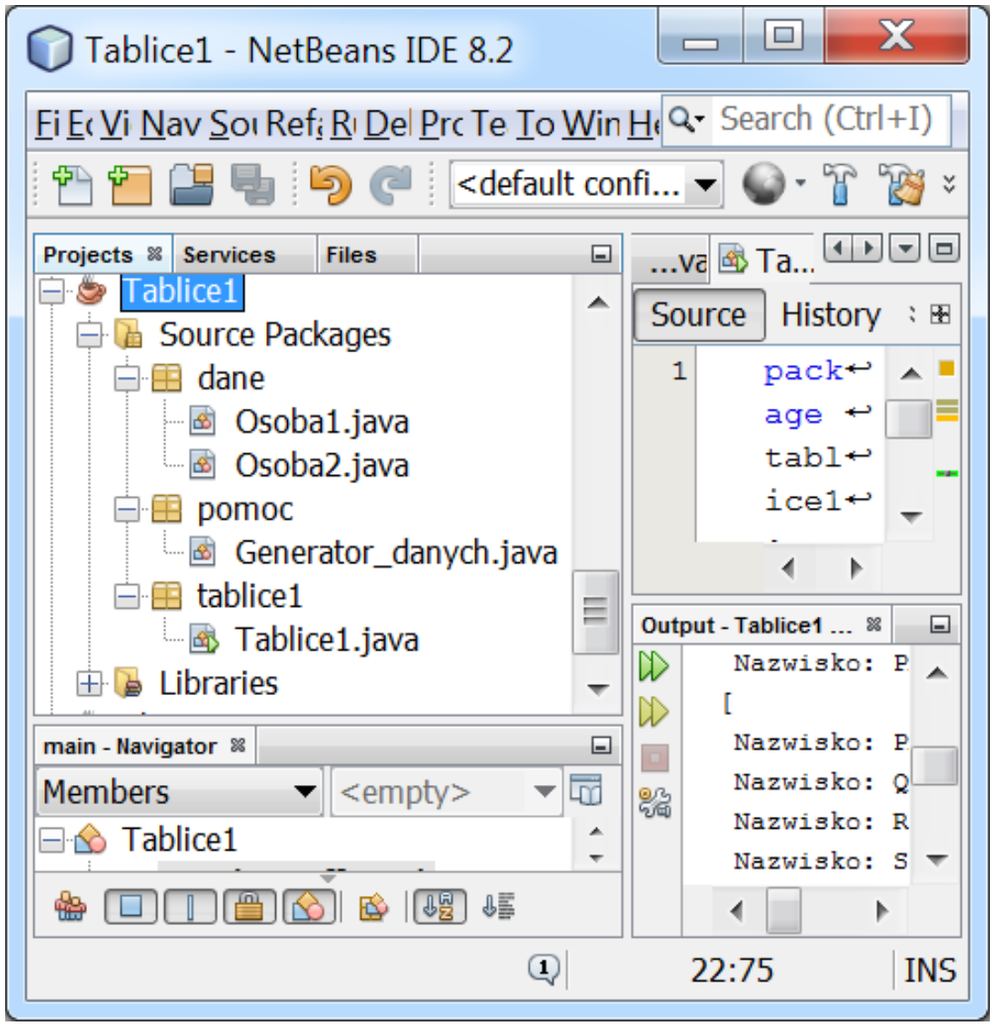
   - Sortowanie
   - Tasowanie
   - Rutynowe manipulowanie danymi (reverse, swap, adAll)
   - Wyszukiwanie danych (search, binarySearch)
   - Badanie danych w pojemniku: częstość wystąpienia tej samej danej oraz porównaniu zawartości dwóch pojemniku
   - Wyszukiwanie danych ekstremalnych (min i max)

Lista klas abstrakcyjnych implementujacych interfejsy pojemników (slajd 8):

- AbstractCollection — pojemnik, który nie jest typu **Set** i **List**. Należy zdefiniować metody pobrania iteratora (***iterator***) oraz rozmiaru kolekcji (***size***).
- AbstractSet — reprezentuje typ **Set**; sposób użycia identyczny jak **AbstractCollection**.
- AbstractList — reprezentuje typ **List** przygotowany do przechowywania danych w tablicy z dostępem losowym. Należy zdefiniować metody (***get*** i, opcjonalnie: ***set***, ***remove***, ***add***) oraz metodę ***size***. Klasa ta jest obsługiwana przez iteratory za pomocą metod ***listIterator*** (i ***iterator***).
- AbstractSequentialList — reprezentuje typ **List** za pomocą sekwencyjnego dostępu do przechowywanych danych, w postaci listy powiązanych elementów. Jako minimum należy zdefiniować metody ***listIterator*** i ***size***. Ważna jest definicja metod pozycyjnego dostępu do danych (przeciwnie do metod dostępu do elementów przez klasę **AbstractList**.)
- AbstractQueue — jako minimum należy zdefiniowac metody ***offer***, ***peek***, ***poll***, i ***size*** oraz ***iterator*** wspierający usuwanie danych.
- AbstractMap — reprezentuje typ **Map**. Jako minimum, nalezy zdefiniowac metodę ***entrySet***, zwracajaca zbior przechowywanych danych, imlementowana podobnie jak typ **AbstractSet**. Jesli typ **Map** ma być modyfikowany, należy zdefiniować metodę ***put***.

3.1 Tablice

Klasa usługowa ***Arrays*** pozwala między innymi **wyszukiwać i sortować** tablice wypełnione typami obiektowymi i nieobiektowymi.



```
package dane;

public class Osoba1 {
    String nazwisko;
    float srednia;
    String uwagi;
    static int numer = 0;

    public void Inicjuj() {
        numer++;  }

    public void Nadaj_dane(String[] dane) {
        Inicjuj();
        nazwisko = dane[0];
        uwagi = dane[2];
        srednia = Float.parseFloat(dane[1]); }

   @Override
   public toString() {
    String napis="\nNazwisko: " +nazwisko;
    napis += ", Średnia: " + srednia;
    napis += ", Uwagi: " + uwagi;
    napis += ", Liczba osób jest równa "
                  + numer;
    return napis;   }
```

```
package dane;

public class Osoba2 extends Osoba1
                implements Comparable <Osoba2>
{
  @Override
    public boolean equals(Object dana) {
        Osoba2 dana_ = (Osoba2) dana;
        return nazwisko.equals(dana_.nazwisko);
    }

 @Override
    public int hashCode() {
         return nazwisko.hashCode();
    }

 }
@Override
    public int compareTo(Osoba2 o) {
        return nazwisko.compareTo(o.nazwisko);
    }
}
```

```
package pomoc;

import dane.Osoba2;
import java.util.Collection;
import java.util.Random;

public class Generator_danych {
    static public byte[] wypelnij(int n, int m, int offset, int zakres) {
        byte tablica1[] = new byte[n];
        Random r = new Random(m);
        for (int i = 0; i < tablica1.length; i++)
            tablica1[i] = (byte) (offset + r.nextInt(zakres));
        return tablica1;
    }
    static public Osoba2 Wstaw(int i) {
        String nazwisko = new String(wypelnij(i + 1, i + 1, 65, 26), 0, i + 1);              //nazwisko z dużych liter
        String uwagi = new String(wypelnij(2 * i + 1, 2 * i + 1, 65, 26), 0, 2 * i + 1);    //uwagi z dużych liter
        String srednia = new String(wypelnij(1, i + 1, 48, 6), 0, 1) //
                                    + "." + new String(wypelnij(2, 3 * i + 1, 48, 10), 0, 2);
        String dane[] = {nazwisko, srednia, uwagi};
        Osoba2 os = new Osoba2();
        os.Nadaj_dane(dane);
        return os;
    }
    static public void wypelnij(int n, Collection<Osoba2> kol) {
        for (int i=0;i<n;i++)
            kol.add(Wstaw(i));  }}
```

```
package tablice1;

import dane.Osoba2;
import java.util.ArrayList;
import java.util.Arrays;
import pomoc.Generator_danych;


public class Tablice1 {

    static Osoba2 tablica[];
```

```
run:
[
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 5,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 5,
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 5,
 Nazwisko: QSNW, Średnia: 2.3, Uwagi: QMDKIAS, Liczba osób jest równa 5,
 Nazwisko: PAAKE, Średnia: 5.2, Uwagi: NAQLXJPKF, Liczba osób jest równa 5]
[
 Nazwisko: PAAKE, Średnia: 5.2, Uwagi: NAQLXJPKF, Liczba osób jest równa 5,
 Nazwisko: QSNW, Średnia: 2.3, Uwagi: QMDKIAS, Liczba osób jest równa 5,
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 5,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 5,
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 5]

 Znaleziono w posortowanej tablicy na pozycji: 1 element:
 Nazwisko: QSNW, Średnia: 2.3, Uwagi: QMDKIAS, Liczba osób jest równa 6
BUILD SUCCESSFUL (total time: 0 seconds)
```

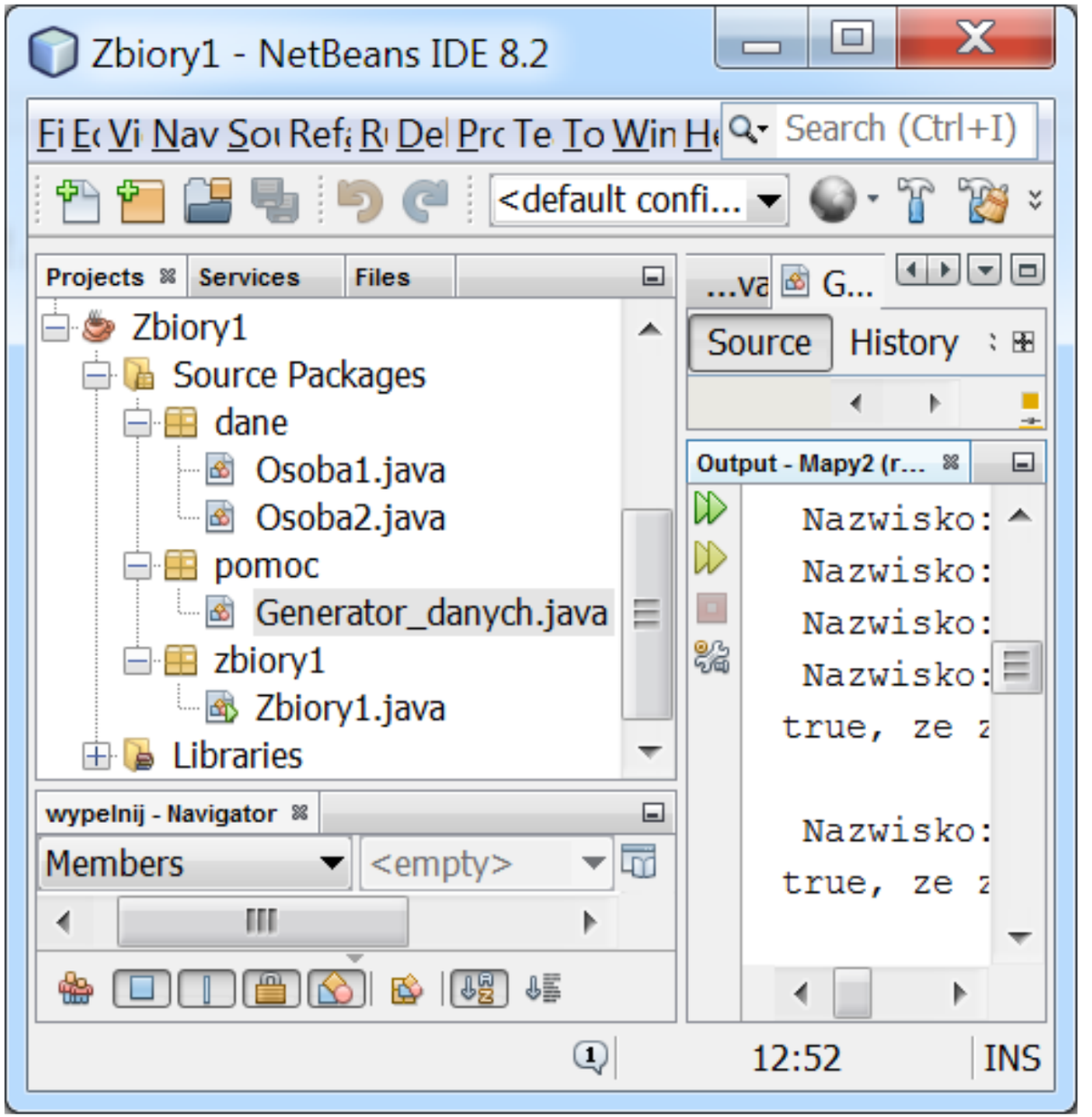
1.1-1.4

2.1-2.2

3.1-3.3

```
public static void main(String[] args) {
        static ArrayList<Osoba2> arraylist =  new ArrayList<>();          //1.1
       Generator_danych.wypelnij(5, arraylist);                           //1.2
        tablica=arraylist.toArray(new Osoba2[0]);                         //1.3
      System.out.println(Arrays.toString(tablica));                       //1.4
      Arrays.sort(tablica);                                               //2.1
      System.out.println( Arrays.toString(tablica));                      //2.2
       Osoba2 a = Generator_danych.Wstaw(3);                                        //3.1
       int b1 = Arrays.binarySearch(tablica, a);                                    //3.2
       System.out.println("\nZnaleziono w posortowanej tablicy na pozycji: " + b1 + " element:  " + a);   //3.3
   }
 }
```

3.2 Zbiory – wstawianie, wyszukiwanie, suma zbiorów, iloczyn zbiorów, różnica symetryczna zbiorów



```
package zbiory1;

import dane.Osoba2;
import java.util.*;
import pomoc.Generator_danych;

public class Zbiory1 {

    static HashSet<Osoba2> hashset = new HashSet<Osoba2>();
    static TreeSet<Osoba2> treeset = new TreeSet<Osoba2>();

    static <K> void roznicasymetryczna(Set<K> set1, Set<K> set2) {
        Set<K> roznicasym = new HashSet<>(set1);                          //3.1
        roznicasym.addAll(set2);                                          //3.2
        System.out.println("suma zbiorow\n" + roznicasym.toString());     //3.3
        Set<K> tmp = new HashSet<>(set1);                                 //3.4
        tmp.retainAll(set2);                                              //3.5
        System.out.println("iloczyn zbiorow\n" + tmp.toString());         //3.6
        roznicasym.removeAll(tmp);                                        //3.7
        System.out.println("roznica symetryczna\n" + roznicasym.toString());  //3.8
    }

    public static void main(String args[]) {
        Generator_danych.wypelnij(2, hashset);                            //1.1
        Generator_danych.wypelnij(2, hashset);                            //1.2
        Generator_danych.wypelnij(4, treeset);                            //1.3
        Generator_danych.wypelnij(4, treeset);                            //1.4
        System.out.println("hashset\n" + hashset.toString());     `       //1.5
        System.out.println("treeset\n" + treeset.toString());             //1.6
```

Liczba tworzonych obiektów typu Osoba 2

```
run:
hashset
[
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 12,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 12]
treeset
[
 Nazwisko: QSNW, Średnia: 2.3, Uwagi: QMDKIAS, Liczba osób jest równa 12,
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 12,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 12,
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 12]
```

1.1-1.2
1.5

1.3-1.4
1.6

```
    Osoba2 klucz = Generator_danych.Wstaw(3);                          //2.1
     boolean b = hashset.contains(klucz);                              //2.2
     System.out.println(b + " ze znaleziono " + klucz + "\n w hashset"); //2.3
     b = treeset.contains(klucz);                                      //2.4
     System.out.println(b + " ze znaleziono " + klucz + "\n w treeset"); //2.5
     roznicasymetryczna(hashset, treeset);                             //3.0
    }
}
```

```
false że znaleziono
 Nazwisko: QSNW, Średnia: 2.3, Uwagi: QMDKIAS, Liczba osób jest równa 13
 w hashset
true że znaleziono
 Nazwisko: QSNW, Średnia: 2.3, Uwagi: QMDKIAS, Liczba osób jest równa 13
 w treeset
suma zbiorow
[
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 13,
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 13,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 13,
 Nazwisko: QSNW, Średnia: 2.3, Uwagi: QMDKIAS, Liczba osób jest równa 13]
iloczyn zbiorow
[
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 13,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 13]
roznica symetryczna
[
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 13,
 Nazwisko: QSNW, Średnia: 2.3, Uwagi: QMDKIAS, Liczba osób jest równa 13]
BUILD SUCCESSFUL (total time: 0 seconds)
```

2.1, 2.2-2.3

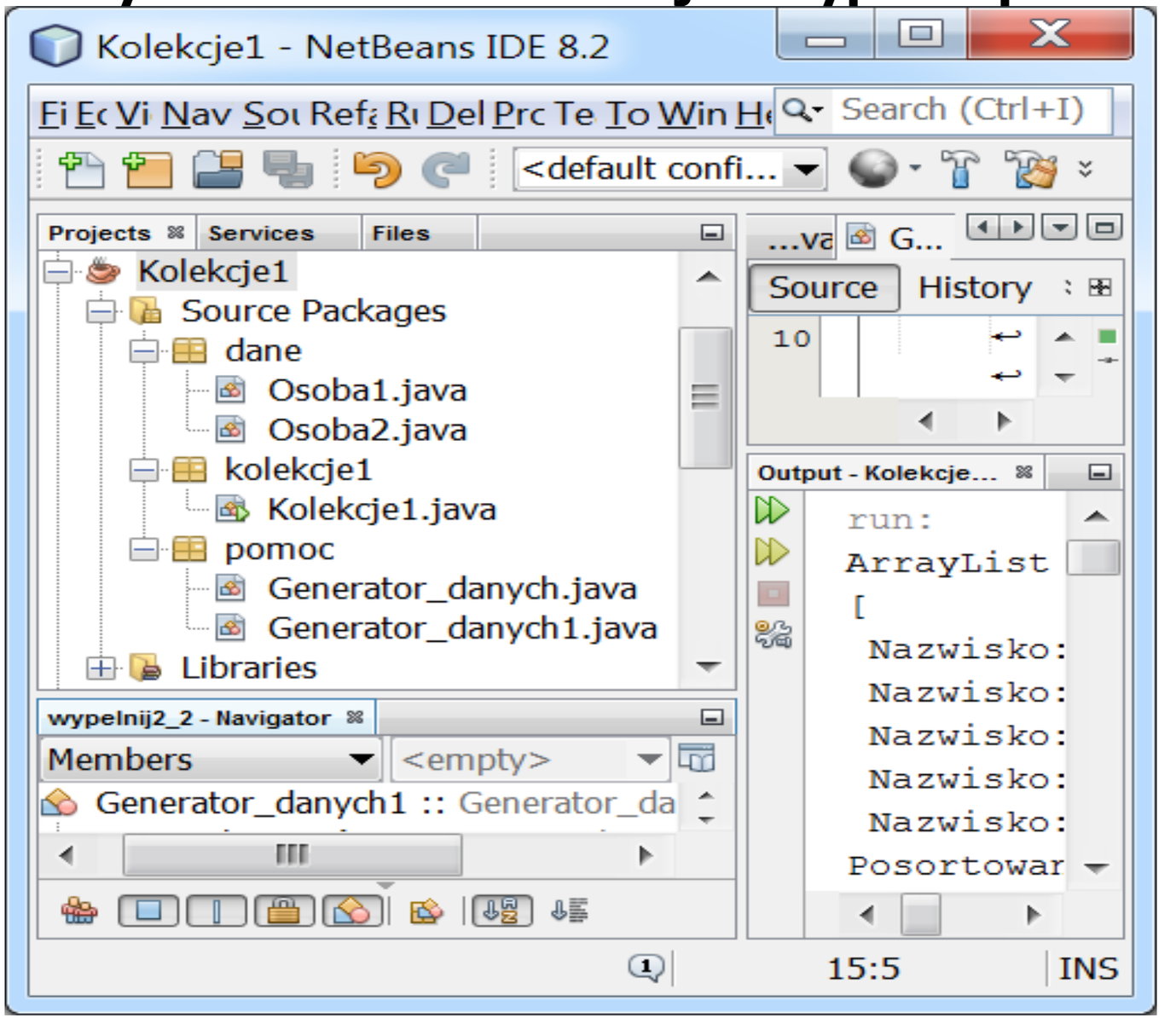
2.1, 2.4-2.5

3.0
3.1-3.3

3.0
3.4-3.6

3.0
3.7-3.8

3.3. Pojemniki na obiekty część1 – wstawianie, sortowanie, wyszukiwanie w posortowanych kolekcjach, wstawianie nowych elementów z tymi samymi kluczami w kolekcjach typu Map



```
package pomoc;

import dane.Osoba2;
import java.util.Map;

public class Generator_danych1 extends Generator_danych {

//wypełnianie kolekcji typu Map, gdzie klucz i dana są identyczne
   static public void wypelnij2_1(int n, Map<Osoba2, Osoba2> mapa) {
      for (int i = 0; i < n; i++) {
         Osoba2 dane = (Osoba2)Wstaw(i);
         mapa.put(dane, dane);
      }
   }

//wypełnianie kolekcji typu Map, gdzie klucz i dana różnią się
   static public void wypelnij2_2(int n, Map<Osoba2, Osoba2> mapa) {
      for (int i = 0; i < n; i++) {
         Osoba2 dane1 = (Osoba2)Wstaw(i);
         Osoba2 dane2 = (Osoba2)Wstaw(i + 3);
         mapa.put(dane1, dane2);
      }
   }
}
```

```
ArrayList
[
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 5,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 5,
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 5,
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 5,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 5]
Posortowana ArrayList
[
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 5,
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 5,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 5,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 5,
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 5]
```

1.1-1.3

1.4-1.5

```
package kolekcje1;
import dane.Osoba2;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.TreeMap;
import java.util.TreeSet;
import pomoc.Generator_danych;
import pomoc.Generator_danych1;

public class Kolekcje1 {

static ArrayList<Osoba2> arraylist = new ArrayList<Osoba2>(); //domyślna pojemność 10 elementów
static LinkedList<Osoba2> linkedlist = new LinkedList<Osoba2>();
static HashSet<Osoba2> hashset = new HashSet<Osoba2>();
static TreeSet<Osoba2> treeset = new TreeSet<Osoba2>();
static HashMap<Osoba2, Osoba2> hashmap = new HashMap<Osoba2, Osoba2>();
static TreeMap<Osoba2, Osoba2> treemap = new TreeMap<Osoba2, Osoba2>();
    public static void main(String[] args) {
        Generator_danych.wypelnij(3, arraylist);                    //1.1
        Generator_danych.wypelnij(2, arraylist);                    //1.2
        System.out.println("ArrayList\n" + arraylist);               //1.3
       arraylist.sort(null);                                        //1.4
        System.out.println("Posortowana ArrayList\n" + arraylist);   //1.5
```

```
Generator_danych.wypelnij(3, linkedlist);                    //2.1
Generator_danych.wypelnij(2, linkedlist);                    //2.2
System.out.println("LinkedList\n" + linkedlist);              //2.3
linkedlist.sort(null);                                       //2.4
System.out.println("Posortowana LinkedList\n" + linkedlist);  //2.5
```

```
LinkedList
[
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 10,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 10,
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 10,
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 10,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 10]
Posortowana LinkedList
[
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 10,
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 10,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 10,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 10,
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 10]
```

2.1-2.3

2.4-2.5

```
Osoba2 a = Generator_danych.Wstaw(1);                                          //3.1

int b1 = Collections.binarySearch(arraylist, a);                               //3.2
  System.out.println("\nWyszukano binarnie w posortowanej Arraylist na pozycji: " + b1 + " element:  " + a);    //3.3

int b2 = Collections.binarySearch(linkedlist, a);                              //3.4
  System.out.println("\nWyszukano binarnie w posortowanej LinkedList na pozycji: " + b2 + " element:  " + a);  //3.5

b1 = arraylist.indexOf(a);                                                     //3.6
System.out.println("\nWyszukano sekwencyjnie w Arraylist na pozycji: " + b1 + " element: " + a);              //3.7

b2 = linkedlist.indexOf(a);                                                    //3.8
System.out.println("\nWyszukano sekwencyjnie w LinkedList na pozycji: " + b2 + " element: " + a);            //3.9
```

Wyszukano binarnie w posortowanej Arraylist na pozycji: 2 element:
Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 11 — **3.1, 3.2-3.3**

Wyszukano binarnie w posortowanej LinkedList na pozycji: 2 element:
Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 11 — **3.1, 3.4-3.5**

Wyszukano sekwencyjnie w Arraylist na pozycji: 2 element:
Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 11 — **3.1, 3.6-3.7**

Wyszukano sekwencyjnie w LinkedList na pozycji: 2 element:
Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 11 — **3.1, 3.8-3.9**

```
Generator_danych.wypelnij(3, hashset);          //4.1
Generator_danych.wypelnij(3, hashset);          //4.2
System.out.println("hashset\n"+ hashset);       //4.3

Generator_danych.wypelnij(3, treeset);          //4.4
Generator_danych.wypelnij(3, treeset);          //4.5
System.out.println("treeset\n" + treeset);      //4.6
```

```
hashset
[
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 17,
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 17,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 17]
treeset
[
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 23,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 23,
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 23]
```

4.1-4.3

4.4-4.6

```
Generator_danych1.wypelnij2_1(3, hashmap);                                    //5.1
Generator_danych1.wypelnij2_1(3, hashmap);                                    //5.2
System.out.println("hashmap\n" + hashmap);                                    //5.3
Generator_danych1.wypelnij2_2(3, hashmap);          //te same klucze, a dane nowe!!!   5.4
System.out.println("hashmap po wstawieniu kolejnych innych danych z tymi samymi kluczami!!!\n"
                    + hashmap);                                               //5.5
```

```
hashmap
{
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 29=
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 29,
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 29=
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 29,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 29=
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 29}
hashmap po wstawieniu kolejnych innych danych z tymi samymi kluczami!!!
{
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 35=
 Nazwisko: QSNW, Średnia: 2.3, Uwagi: QMDKIAS, Liczba osób jest równa 35,
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 35=
 Nazwisko: RGSWJL, Średnia: 1.12, Uwagi: OMVLBHSTRVQ, Liczba osób jest równa 35,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 35=
 Nazwisko: PAAKE, Średnia: 5.2, Uwagi: NAQLXJPKF, Liczba osób jest równa 35}
```

Klucz
Dana

5.1-5.3

5.4-5.5

```
Generator_danych1.wypelnij2_1(3, treemap);                                    //5.6
Generator_danych1.wypelnij2_1(3, treemap);                                    //5.7
System.out.println("treemap\n" + treemap);                                    //5.8
Generator_danych1.wypelnij2_2(3, treemap);//te same klucze, a dane zastapione nowymi danymi!!! 5.9
System.out.println("treemap po wstawieniu kolejnych innych danych z tymi samymi kluczami!!!\n"
                        + treemap);                                           //5.10
                                                                    }}
```

```
treemap
{
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 41=
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 41,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 41=
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 41,
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 41=
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 41}
treemap po wstawieniu kolejnych innych danych z tymi samymi kluczami!!!
{
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 47=
 Nazwisko: QSNW, Średnia: 2.3, Uwagi: QMDKIAS, Liczba osób jest równa 47,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 47=
 Nazwisko: PAAKE, Średnia: 5.2, Uwagi: NAQLXJPKF, Liczba osób jest równa 47,
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 47=
 Nazwisko: RGSWJL, Średnia: 1.12, Uwagi: OMVLBHSTRVQ, Liczba osób jest równa 47}
BUILD SUCCESSFUL (total time: 0 seconds)
```

5.6-5.8

5.9-5.10

3.4. Pojemniki na obiekty część2

```
package mapy2;

import dane.Osoba2;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;
import java.util.TreeSet;
import pomoc.Generator_danych1;

public class Mapy2 {

    static HashSet<Osoba2> hashset = new HashSet<Osoba2>();
    static HashMap<Osoba2, Osoba2> hashmap = new HashMap<Osoba2, Osoba2>();
    static TreeMap<Osoba2, Osoba2> treemap = new TreeMap<Osoba2, Osoba2>();

    static <K, V> Map<K, V> sumamap(Map<K, V> pierwsza, Map<K, V> druga) {
        Map<K, V> sumamap_ = new HashMap<>(pierwsza);
        sumamap_.putAll(druga);
        return sumamap_;
    }
```

```
static <K, V> Set<K> walidacja(Map<K, V> podstawowa,  Set<K> wzorzec) {
        Set<K> zle = new TreeSet<>(wzorzec);
        Set<K> klucze = podstawowa.keySet();
        if (!klucze.containsAll(wzorzec))
            zle.retainAll(klucze);
        return zle;
    }
public static void main(String[] args) {
 Generator_danych1.wypelnij(5, hashset);
 Generator_danych1.wypelnij(5, hashset);
 Generator_danych1.wypelnij2_1(3, hashmap);
 Generator_danych1.wypelnij2_1(3, hashmap);
 Generator_danych1.wypelnij2_1(4, treemap);
 Generator_danych1.wypelnij2_1(4, treemap);

System.out.println("hashmap\n" +
                hashmap.toString());

 System.out.println("treemap\n" +
                treemap.toString());
```

```
hashmap
{
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 24=
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 24,
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 24=
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 24,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 24=
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 24}
treemap
{
 Nazwisko: QSNW, Średnia: 2.3, Uwagi: QMDKIAS, Liczba osób jest równa 24=
 Nazwisko: QSNW, Średnia: 2.3, Uwagi: QMDKIAS, Liczba osób jest równa 24,
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 24=
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 24,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 24=
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 24,
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 24=
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 24}
```

```
Osoba2 a = Generator_danych1.Wstaw(2);                          //1.1
boolean b1 = hashmap.containsKey(a);                            //1.2
System.out.println(b1 + ", ze znaleziono w HashMap klucz\n" + a);   //1.3
boolean b2 = hashmap.containsValue(a);                          //1.4
System.out.println(b2 + ", ze znaleziono w HashMap dane\n" + a);    //1.5
b1 = treemap.containsKey(a);                                    //1.6
System.out.println(b1 + ", ze znaleziono w TreeMap klucz\n" + a);   //1.7
b2 = treemap.containsValue(a);                                  //1.8
System.out.println(b2 + ", ze znaleziono w TreeMap dane\n" + a);    //1.9
```

```
true, ze znaleziono w HashMap klucz
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 25
true, ze znaleziono w HashMap dane
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 25
true, ze znaleziono w TreeMap klucz
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 25
true, ze znaleziono w TreeMap dane
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 25
```

1.1, 1.2 - 1.3

1.1, 1.4 - 1.5

1.1, 1.6 – 1.7

1.1, 1.8 – 1.9

Map<Osoba2, Osoba2> sumamap_ = sumamap(treemap, hashmap); //typu HashMap

System.out.println("suma map\n" + sumamap_.toString());

```
suma map
{
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 25=
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 25,
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 25=
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 25,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 25=
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 25,
 Nazwisko: QSNW, Średnia: 2.3, Uwagi: QMDKIAS, Liczba osób jest równa 25=
 Nazwisko: QSNW, Średnia: 2.3, Uwagi: QMDKIAS, Liczba osób jest równa 25}
```

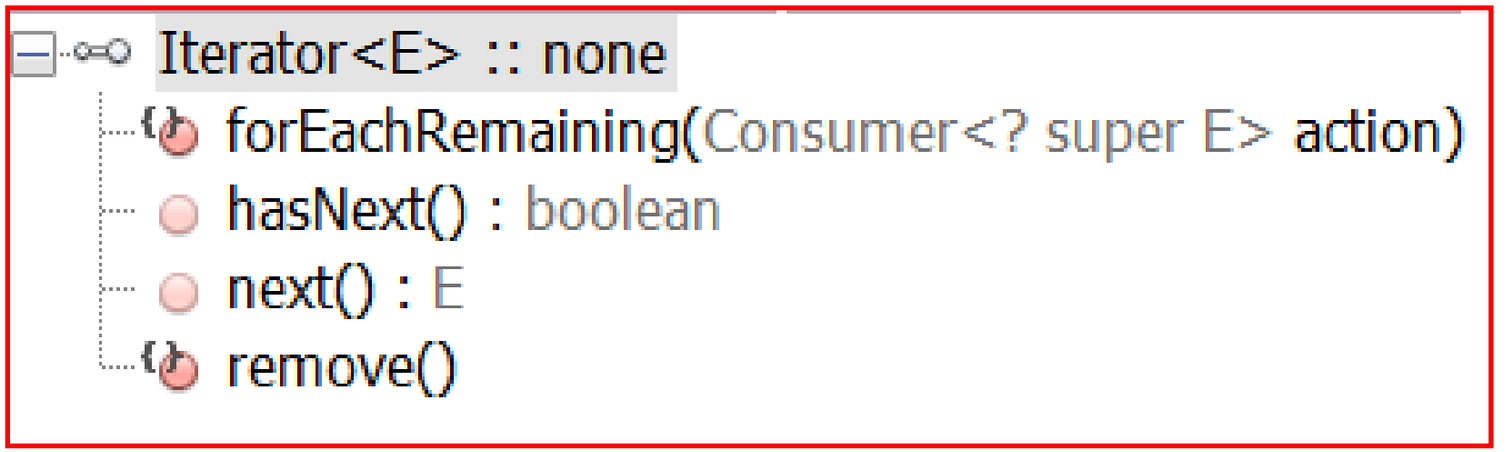
System.out.println("hashset\n" + hashset.toString());

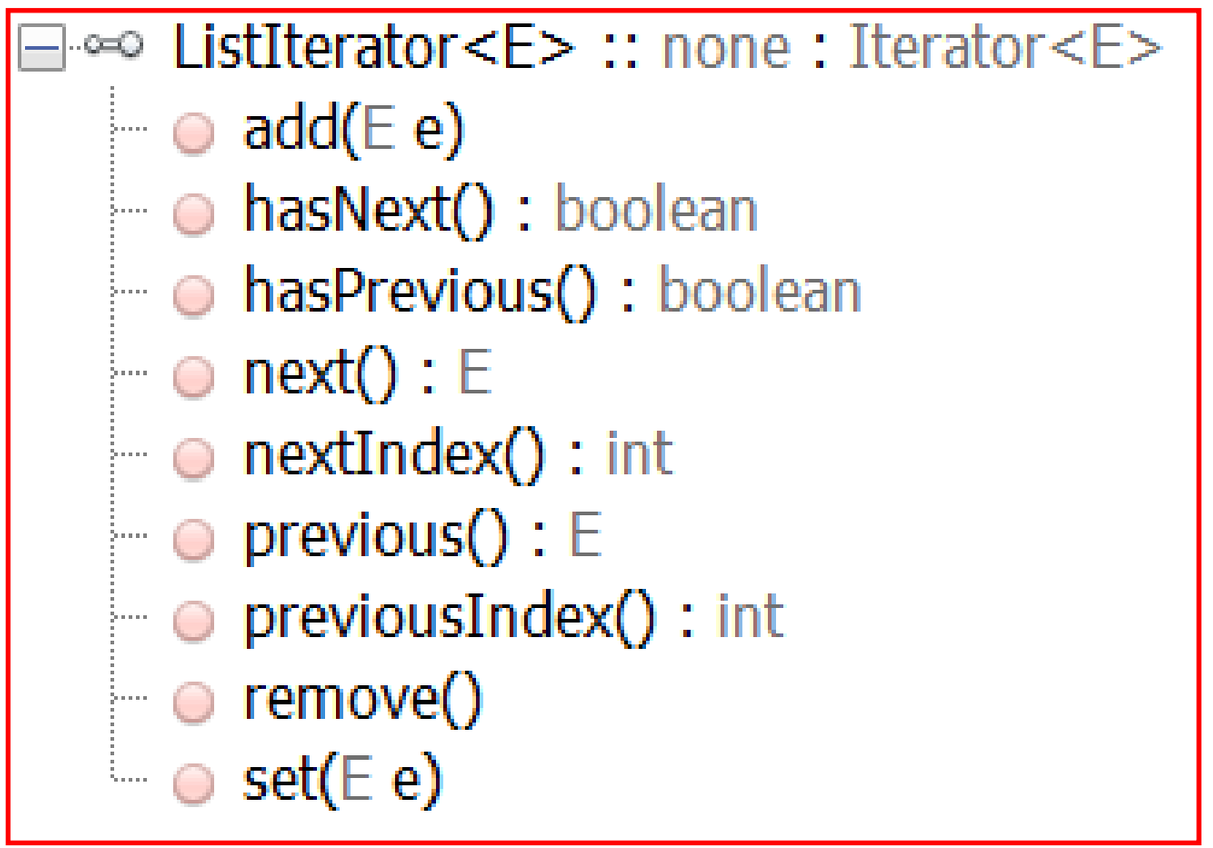
```
hashset
[
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 25,
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 25,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 25,
 Nazwisko: QSNW, Średnia: 2.3, Uwagi: QMDKIAS, Liczba osób jest równa 25,
 Nazwisko: PAAKE, Średnia: 5.2, Uwagi: NAQLXJPKF, Liczba osób jest równa 25]
```

```
Set<Osoba2> klucze_walidacji = walidacja(treemap, hashset);
System.out.println("wspolny zbior kluczy w treemap i hashset\n" + klucze_walidacji.toString());
System.out.println("Wynik porownania zbioru kluczy w TreeMap i HashMap: " +
                                                    treemap.keySet().equals(hashmap.keySet()));
                                                }
                                            }
```

```
wspolny zbior kluczy w treemap i hashset
[
 Nazwisko: QSNW, Średnia: 2.3, Uwagi: QMDKIAS, Liczba osób jest równa 25,
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 25,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 25,
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 25]
Wynik porownania zbioru kluczy w TreeMap i HashMap: false
BUILD SUCCESSFUL (total time: 0 seconds)
```

3.5. Iteratory





```
package iteratory1;

import dane.Osoba2;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.ListIterator;
import java.util.TreeMap;
import java.util.TreeSet;
import pomoc.Generator_danych1;


public class Iteratory1 {

    static ArrayList<Osoba2> arraylist = new ArrayList<Osoba2>(); //domyślna pojemność 10 elementów
    static LinkedList<Osoba2> linkedlist = new LinkedList<Osoba2>();
    static HashSet<Osoba2> hashset = new HashSet<Osoba2>();
    static TreeSet<Osoba2> treeset = new TreeSet<Osoba2>();
    static HashMap<Osoba2, Osoba2> hashmap = new HashMap<Osoba2, Osoba2>();
    static TreeMap<Osoba2, Osoba2> treemap = new TreeMap<Osoba2, Osoba2>();
```

```
static <K> void wyswietlIterator(String s, Iterator<K> it) {
    System.out.print(s);
    while (it.hasNext()) {
        K k = it.next();
        System.out.print(k);     }
}

static <K> void wyswietlIterator(String s, ListIterator<K> it) {
    System.out.print(s);
    while (it.hasNext()) {
        K k = it.next();
        System.out.print(k);   }
}

public static void main(String[] args) {
    Generator_danych1.wypelnij(2, arraylist);        Generator_danych1.wypelnij(2, arraylist);
    Generator_danych1.wypelnij(2, linkedlist);       Generator_danych1.wypelnij(2, linkedlist);
    Generator_danych1.wypelnij(3, hashset);          Generator_danych1.wypelnij(3, hashset);
    Generator_danych1.wypelnij(2, treeset);          Generator_danych1.wypelnij(2, treeset);
    Generator_danych1.wypelnij2_1(3, hashmap);   Generator_danych1.wypelnij2_1(3, hashmap);
    Generator_danych1.wypelnij2_1(2, treemap);   Generator_danych1.wypelnij2_1(2, treemap);
```

```
System.out.println("ArrayList\n" + arraylist);
```

```
ArrayList
[
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28,
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28]
```

```
wyswietlIterator("Iterator ArrayList", arraylist.iterator());
 wyswietlIterator("\nListIterator ArrayList", arraylist.listIterator());
```

```
Iterator ArrayList
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28
ListIterator ArrayList
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28
```

```
System.out.println("\nLinkedList\n" + linkedlist);
 wyswietlIterator("Iterator LinkedList", linkedlist.iterator());
 wyswietlIterator("\nListIterator LinkedList", linkedlist.listIterator());
```

```
LinkedList
[
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28,
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28]
Iterator LinkedList
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28
ListIterator LinkedList
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28
```

```
System.out.println("\nhashset\n" + hashset);
 wyswietlIterator("Iterator HashSet", hashset.iterator());
```

```
hashset
[
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28,
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 28,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28]
Iterator HashSet
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 28
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28
```

```
System.out.println("\ntreeset\n" + treeset);
 wyswietlIterator("Iterator TreeSet", treeset.iterator());
```

```
treeset
[
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28]
Iterator TreeSet
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28
```

```
System.out.println("\nhashmap\n" + hashmap);
 wyswietlIterator("Iterator HashMap", hashmap.entrySet().iterator());
```

```
hashmap
{
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28=
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28,
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 28=
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 28,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28=
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28}
Iterator HashMap
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28=
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 28=
 Nazwisko: SMM, Średnia: 2.64, Uwagi: PAAKE, Liczba osób jest równa 28
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28=
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28
```

```
        System.out.println("\ntreemap\n" + treemap);
        wyswietlIterator("Iterator TreeMap", treemap.entrySet().iterator());
        System.out.println();
    }
}
```

```
treemap
{
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28=
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28,
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28=
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28}
Iterator TreeMap
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28=
 Nazwisko: R, Średnia: 3.58, Uwagi: R, Liczba osób jest równa 28
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28=
 Nazwisko: SG, Średnia: 4.22, Uwagi: SMM, Liczba osób jest równa 28
BUILD SUCCESSFUL (total time: 0 seconds)
```