Przechowywanie obiektów w pamięci programu
Część I (obiekty typu String)

(wg http://java.sun.com/docs/books/tutorial/collections/ **)**

1.Obiekty typu String

Elementy typu *String* są przystosowane do przechowywania w pojemnikach – mogą być wyszukiwane, sortowane, usuwane itp. dzięki przedefiniowaniu metod *equals* oraz *hashCode,* dziedziczonych po klasie *Object* i implementacji metody *compareTo* z interfejsu *Comparable.*

```
public final class String   extends Object
                             implements Serializable,
                                        Comparable<String>,
                                        CharSequence
```

```
public interface Comparable<T>
      {      public int compareTo(T o);   }
```

```
public class Object
 {  //..............
```

boolean	**equals**(Object obj) Indicates whether some other object is "equal to" this one.
int	**hashCode**() Returns a hash code value for the object.

```
}
```

Klasa *String* ma przedefiniowane metody *equals* oraz *hashCode* oraz zaimplementowaną metodę *compareTo.*

int	**compareTo**(String anotherString) Compares two strings lexicographically.
int	**hashCode**() Returns a hash code for this string.
boolean	**equals**(Object anObject) Compares this string to the specified object.

2. Tablice

Klasa usługowa ***Arrays*** pozwala między innymi wyszukiwać i sortować tablice wypełnione typami obiektowymi i nie obiektowymi.
public class **Arrays** extends Object

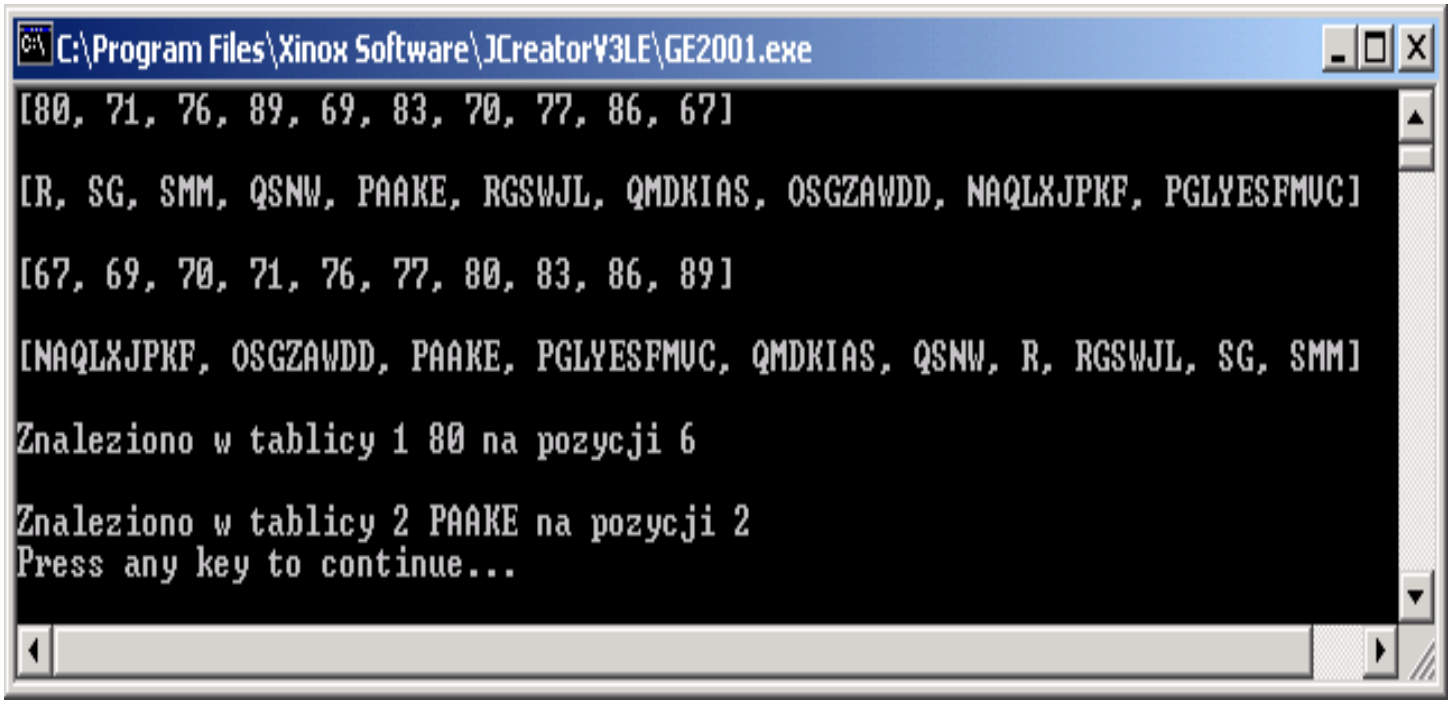
Method Summary	
static <T> List<T>	**asList**(T... a) Returns a fixed-size list backed by the specified array.
static int	**binarySearch**(byte[] a, byte key) Searches the specified array of bytes for the specified value using the binary search algorithm.
static int	**binarySearch**(char[] a, char key) Searches the specified array of chars for the specified value using the binary search algorithm.
static int	**binarySearch**(double[] a, double key) Searches the specified array of doubles for the specified value using the binary search algorithm.
static int	**binarySearch**(float[] a, float key) Searches the specified array of floats for the specified value using the binary search algorithm.
static int	**binarySearch**(int[] a, int key) Searches the specified array of ints for the specified value using the binary search algorithm.
static int	**binarySearch**(long[] a, long key) Searches the specified array of longs for the specified value using the binary search algorithm.
static int	**binarySearch**(Object[] a, Object key) Searches the specified array for the specified object using the binary search algorithm.
static int	**binarySearch**(short[] a, short key) Searches the specified array of shorts for the specified value using the binary search algorithm.
static<T> int	**binarySearch**(T[] a, T key, Comparator<? super T> c) Searches the specified array for the specified object using the binary search algorithm.
static boolean	**deepEquals**(Object[] a1, Object[] a2) Returns true if the two specified arrays are *deeply equal* to one another.
static int	**deepHashCode**(Object[] a) Returns a hash code based on the "deep contents" of the specified array.
static String	**deepToString**(Object[] a) Returns a string representation of the "deep contents" of the specified array.
static boolean	**equals**(boolean[] a, boolean[] a2) Returns true if the two specified arrays of booleans are *equal* to one another.
static boolean	**equals**(byte[] a, byte[] a2) Returns true if the two specified arrays of bytes are *equal* to one another.
static boolean	**equals**(char[] a, char[] a2) Returns true if the two specified arrays of chars are *equal* to one another.
static boolean	**equals**(double[] a, double[] a2) Returns true if the two specified arrays of doubles are *equal* to one another.
static boolean	**equals**(float[] a, float[] a2) Returns true if the two specified arrays of floats are *equal* to one another.
static boolean	**equals**(int[] a, int[] a2) Returns true if the two specified arrays of ints are *equal* to one another.

static boolean	**equals**(long[] a, long[] a2)Returns true if the two specified arrays of longs are *equal* to one another.
static boolean	**equals**(Object[] a, Object[] a2) Returns true if the two specified arrays of Objects are *equal* to one another.
static boolean	**equals**(short[] a, short[] a2)Returns true if the two specified arrays of shorts are *equal* to one another.
static void	**fill**(boolean[] a, boolean val) Assigns the specified boolean value to each element of the specified array of booleans.
static void	**fill**(boolean[] a, int fromIndex, int toIndex, boolean val) Assigns the specified boolean value to each element of the specified range of the specified array of booleans.
static void	**fill**(byte[] a, byte val) Assigns the specified byte value to each element of the specified array of bytes.
static void	**fill**(byte[] a, int fromIndex, int toIndex, byte val) Assigns the specified byte value to each element of the specified range of the specified array of bytes.
static void	**fill**(char[] a, char val)Assigns the specified char value to each element of the specified array of chars.
static void	**fill**(char[] a, int fromIndex, int toIndex, char val) Assigns the specified char value to each element of the specified range of the specified array of chars.
static void	**fill**(double[] a, double val) Assigns the specified double value to each element of the specified array of doubles.
static void	**fill**(double[] a, int fromIndex, int toIndex, double val) Assigns the specified double value to each element of the specified range of the specified array of doubles.
static void	**fill**(float[] a, float val)Assigns the specified float value to each element of the specified array of floats.
static void	**fill**(float[] a, int fromIndex, int toIndex, float val) Assigns the specified float value to each element of the specified range of the specified array of floats.
static void	**fill**(int[] a, int val) Assigns the specified int value to each element of the specified array of ints.
static void	**fill**(int[] a, int fromIndex, int toIndex, int val) Assigns the specified int value to each element of the specified range of the specified array of ints.
static void	**fill**(long[] a, int fromIndex, int toIndex, long val) Assigns the specified long value to each element of the specified range of the specified array of longs.
static void	**fill**(long[] a, long val) Assigns the specified long value to each element of the specified array of longs.
static void	**fill**(Object[] a, int fromIndex, int toIndex, Object val) Assigns the specified Object reference to each element of the specified range of the specified array of Objects.
static void	**fill**(Object[] a, Object val) Assigns the specified Object reference to each element of the specified array of Objects.
static void	**fill**(short[] a, int fromIndex, int toIndex, short val) Assigns the specified short value to each element of the specified range of the specified array of shorts.
static void	**fill**(short[] a, short val)Assigns the specified short value to each element of the specified array of shorts.
static int	**hashCode**(boolean[] a) Returns a hash code based on the contents of the specified array.
static int	**hashCode**(byte[] a) Returns a hash code based on the contents of the specified array.
static int	**hashCode**(char[] a) Returns a hash code based on the contents of the specified array.
static int	**hashCode**(double[] a) Returns a hash code based on the contents of the specified array.

static int	**hashCode**(float[] a) Returns a hash code based on the contents of the specified array.
static int	**hashCode**(int[] a) Returns a hash code based on the contents of the specified array.
static int	**hashCode**(long[] a) Returns a hash code based on the contents of the specified array.
static int	**hashCode**(Object[] a) Returns a hash code based on the contents of the specified array.
static int	**hashCode**(short[] a) Returns a hash code based on the contents of the specified array.
static void	**sort**(byte[] a) Sorts the specified array of bytes into ascending numerical order.
static void	**sort**(byte[] a, int fromIndex, int toIndex) Sorts the specified range of the specified array of bytes into ascending numerical order.
static void	**sort**(char[] a) Sorts the specified array of chars into ascending numerical order.
static void	**sort**(char[] a, int fromIndex, int toIndex) Sorts the specified range of the specified array of chars into ascending numerical order.
static void	**sort**(double[] a) Sorts the specified array of doubles into ascending numerical order.
static void	**sort**(double[] a, int fromIndex, int toIndex) Sorts the specified range of the specified array of doubles into ascending numerical order.
static void	**sort**(float[] a) Sorts the specified array of floats into ascending numerical order.
static void	**sort**(float[] a, int fromIndex, int toIndex) Sorts the specified range of the specified array of floats into ascending numerical order.
static void	**sort**(int[] a) Sorts the specified array of ints into ascending numerical order.
static void	**sort**(int[] a, int fromIndex, int toIndex) Sorts the specified range of the specified array of ints into ascending numerical order.
static void	**sort**(long[] a) Sorts the specified array of longs into ascending numerical order.
static void	**sort**(long[] a, int fromIndex, int toIndex) Sorts the specified range of the specified array of longs into ascending numerical order.
static void	**sort**(Object[] a) Sorts the specified array of objects into ascending order, according to the *natural ordering* of its elements.
static void	**sort**(Object[] a, int fromIndex, int toIndex) Sorts the specified range of the specified array of objects into ascending order, according to the *natural ordering* of its elements.
static void	**sort**(short[] a) Sorts the specified array of shorts into ascending numerical order.
static void	**sort**(short[] a, int fromIndex, int toIndex) Sorts the specified range of the specified array of shorts into ascending numerical order.
static<T> void	**sort**(T[] a, Comparator<? super T> c) Sorts the specified array of objects according to the order induced by the specified comparator.
static<T> void	**sort**(T[] a, int fromIndex, int toIndex, Comparator<? super T> c) Sorts the specified range of the specified array of objects according to the order induced by the specified comparator.
static String	**toString**(boolean[] a) Returns a string representation of the contents of the specified array.
static String	**toString**(byte[] a) Returns a string representation of the contents of the specified array.
static String	**toString**(char[] a) Returns a string representation of the contents of the specified array.

static String	**toString**(double[] a)	Returns a string representation of the contents of the specified array.
static String	**toString**(float[] a)	Returns a string representation of the contents of the specified array.
static String	**toString**(int[] a)	Returns a string representation of the contents of the specified array.
static String	**toString**(long[] a)	Returns a string representation of the contents of the specified array.
static String	**toString**(Object[] a)	Returns a string representation of the contents of the specified array.
static String	**toString**(short[] a)	Returns a string representation of the contents of the specified array.
Methods inherited from class java.lang.Object		
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait		

Przykład 1–operacje na tablicach o elementach typu *byte* oraz *String*




```
import java.lang.*;
import java.util.*;

public class Tablice1
 { static byte tablica1[];
   static String tablica2[];

 static public void wypelnij1(int n)
       { tablica1=new byte[n];
          Random r=new Random(n);
          for (int i=0; i<tablica1.length;i++)
             tablica1[i]=(byte)(65+r.nextInt(26)); //generowanie kodów ASCCI dużych liter
       }
```

```
static byte klucz1(int n)
      { Random r=new Random(n);
        return (byte)(65+r.nextInt(26));
      }
static String klucz2(int n)
      { wypelnij1(n);
        return new String(tablica1);  }

static public void wypelnij2(int n)
    { tablica2=new String[n];
      for (int i=0; i<tablica2.length;i++)
       { wypelnij1(i+1);
         tablica2[i]=new String(tablica1); }      // String(byte[] bytes)
     }              // tworzy łańcuch przez konwertowanie tablicy bajtów jako domyślnych  kodów znaków

public static void main(String args[])
    {
      wypelnij2(10);
      wypelnij1(10);

      System.out.println("\n"+Arrays.toString(tablica1));
      System.out.println("\n"+Arrays.toString(tablica2));

      Arrays.sort(tablica1);
      Arrays.sort(tablica2);

      System.out.println("\n"+Arrays.toString(tablica1));
      System.out.println("\n"+Arrays.toString(tablica2));

      byte a1=klucz1(5);
      int b1=Arrays.binarySearch(tablica1,a1);
      System.out.println("\n"+"Znaleziono w tablicy 1 "+a1+" na pozycji "+b1);

      String a2=klucz2(5);
      int b2=Arrays.binarySearch(tablica2,a2);
      System.out.println("\n"+"Znaleziono w tablicy 2 "+a2+" na pozycji "+b2);
  }
}
```

3. Pojemniki na obiekty

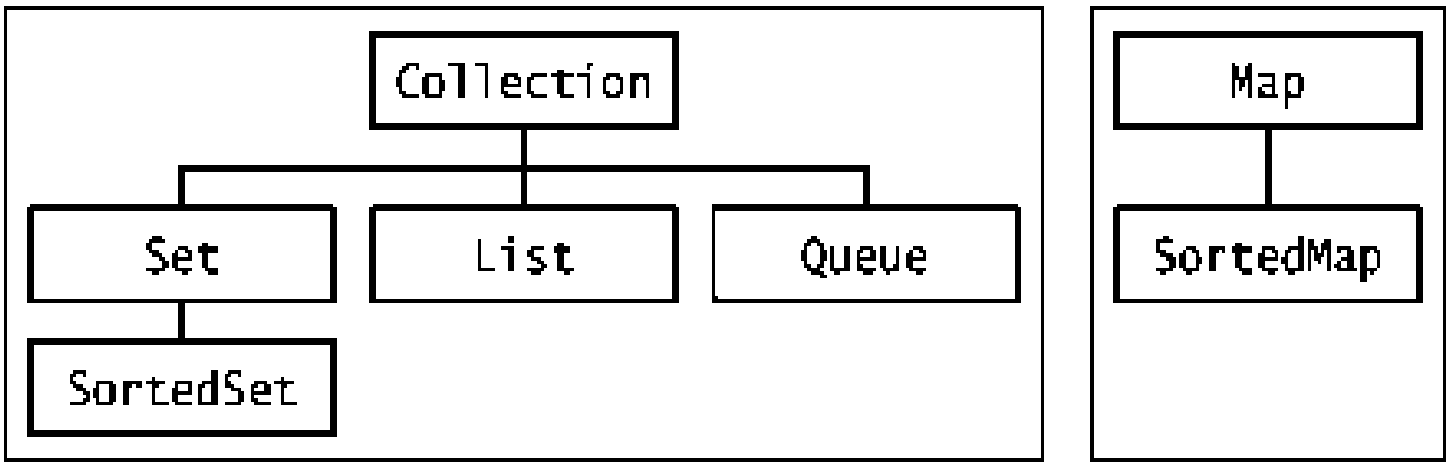
Właściwości pojemników:

- **Interfejsy (interfaces):** abstrakcyjne typy danych, które deklarują abstrakcyjne operacje na elementach umieszczonych w pojemnikach niezależnie od implementacji
- **Implementacje (Implementations):** Zdefiniowano klasy, które implementują 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
- 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



Dwa typy pojemników

1) **Kolekcje** (***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***) – wymaga imlementowania specjalnych operacji wstawiania, usuwania oraz przeszukiwania

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)***

Implementacje pojemników

General-purpose Implementations					
Interfaces	**Implementations**				
	Hash table	**Resizable array**	**Tree**	**Linked list**	**Hash table + Linked list**
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Quene					
Map	HashMap		TreeMap		LinkedHashMap

Funkcjonalność pojemników

1) Funkcjonalność pojemników typu Collection

```
public interface Collection<E> extends Iterable<E> {
    // Podstawowe operacje
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);                 // opcjonalne
    boolean remove(Object element);  // opcjonalne
    Iterator iterator();

    //operacje na zbiorach elementów
    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c);
    boolean removeAll(Collection<?> c);  // opcjonalne
    boolean retainAll(Collection<?> c);    // opcjonalne
    void clear();                                  // opcjonalne

    // operacje tablicowe
    Object[] toArray();
    <T> T[] toArray(T[] a);
}
```

Proste pobieranie elementów umieszczonych w pojemniku

```
public interface Iterator<E> {
    boolean hasNext();
    E next();
    void remove(); //opcjonalne
}
```

1.1) Funkcjonalność pojemników typu List

```
public interface List<E> extends Collection<E> {
        //dostęp pozycyjny
        E get(int index);
        E set(int index, E element);        //opcjonalne
        boolean add(E element);             //opcjonalne
        void add(int index, E element);     //opcjonalne
        E remove(int index);                //opcjonalne
        abstract boolean addAll(int index,
            Collection<? extends E> c);     //opcjonalne

       //wyszukiwanie
        int indexOf(Object o);
        int lastIndexOf(Object o);

       //Iteracje
        ListIterator<E> listIterator();
        ListIterator<E> listIterator(int index);

        //operacje na fragmencie listy
        List<E> subList(int from, int to);
    }


    public interface ListIterator<E> extends Iterator<E> {
        boolean hasNext();
        E next();
        boolean hasPrevious();
        E previous();
        int nextIndex();
        int previousIndex();
        void remove();              //opcjonalne
        void set(E o);              //opcjonalne
        void add(E o);              //opcjonalne
    }
```

Algorytmy klasy Collections (spełnia taką samą rolę dla pojemników jak klasa Arrays dla tablic)

- sort — sortuje listę elementów za pomocą algorytmu sortowania przez łączenie
- shuffle— losowo przemieszcza elementy w tablicy
- reverse — odwraca położenie elementów w pojemniku
- rotate — zmienia położenie elementów wg ustalonej odległości
- swap — zamienia elementy miejscami na wyznaczonych pozycjach
- replaceAll — zmienia wszystkie wystąpienia podanej wartości na inną
- fill — wypełnia pojemnik elemetami o wyznaczonej wartości
- copy — kopiuje wyznaczoną kolekcję elementów do drugiej kolekcji
- binarySearch — wyszukuje element w posortowanej kolekcji za pomocą algorytmu wyszukiwania połówkowego
- indexOfSubList — zwraca indeks pierwszej podlisty, która jest równa podanej
- lastIndexOfSubList — zwraca indeks ostatniej podlisty, która jest równa podanej

- public class **ArrayList<E>** extends AbstractList<E>
  implements List<E>, RandomAccess, Cloneable, Serializable

ArrayList – Interfejs *List* jest implementowany jako tablica o szybkim, (swobodnym) dostępie do elementów. Charakteryzuje się wolnym usuwaniem i wstawianiem elementów.

- public class **LinkedList<E>** extends AbstractSequentialList<E>
  implements List<E>, Queue<E>, Cloneable, Serializable

LinkedList – reprezentuje optymalny sekwencyjny dostęp do elementów i wolniejszy niż w ArrayList dostęp swobodny do elementów.
Dodatkowe metody: *addFirst( ), addLast( ), getFirst( ), getLast( ), removeFirst( ), removeLast( )*

Oba typy pojemników pozwalają na tworzenie kopii elementów

1.2) Funkcjonalność pojemników typu SET

```
public interface Set<E> extends Collection<E> {
   // podstawowe operacje
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);              //opcjonalne
    boolean remove(Object element); //opcjonalne
    Iterator iterator();

    //operacje na zbiorach elementów
    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c);
    boolean removeAll(Collection<?> c);     //opcjonalne
    boolean retainAll(Collection<?> c);     //opcjonalne
    void clear();                           //opcjonalne

    // operacje na tablicach
    Object[] toArray();
    <T> T[] toArray(T[] a);
}


public interface SortedSet<E> extends Set<E> {
   // działanie w wyznaczonych podzbiorach elementów
    SortedSet<E> subSet(E fromElement, E toElement);
    SortedSet<E> headSet(E toElement);
    SortedSet<E> tailSet(E fromElement);

   //punkty końcowe
    E first();
    E last();

    //porównywanie
   Comparator<? super E> comparator();
}
```

- public interface **Set<E>** extends Collection<E>

Nie pozwala ma tworzenie kopii elementów

- public class **HashSet<E>** extends AbstractSet<E>
  implements Set<E>, Cloneable, Serializable

HashSet – szybkie przeszukiwanie kolekcji. *HashSet* przy przetwarzaniu elementów korzysta z ich metod *hashCode* i *equals*

- public class **TreeSet<E>** extends AbstractSet<E>
  implements SortedSet<E>, Cloneable, Serializable

TreeSet – można tworzyć uporządkowany ciąg elementów, ponieważ elementy są umieszczane w strukturze typu drzewo czerwono-czarne. *TreeSet* przy przetwarzaniu elementów korzysta z ich metody *compareTo.*

1.3) Funkcjonalność pojemników typu Quene

```
public interface Queue<E> extends Collection<E> {
    E element();
    boolean offer(E o);
    E peek();
    E poll();
    E remove();
}
```

Queue Interface Structure		
	Throws exception	**Returns special value**
Insert	add(e)	offer(e)
Remove	remove()	poll()
Examine	element()	peek()

2) Funkcjonalność pojemników typu Map

Elementy są umieszczane w kolekcji jako pary: klucz (*K*) oraz wartość (*V*)

```
public interface Map {
   //podstawowe operacje
    V put(K key, V value);
    V get(Object key);
    V remove(Object key);
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    int size();
    boolean isEmpty();

   //operacje na zbiorach elementów
    void putAll(Map<? extends K,? extends V> t);
    void clear();

   // podejście typu kolekcja
    public Set<K> keySet();
    public Collection<V> values();
    public Set<Map.Entry<K,V>> entrySet();

   // interfejs dla dostępu do elementów za pomocą kluczy
    public interface Map.Entry {
        K getKey();
        V getValue();
        V setValue(V value);
        int hashCode();
        V setValue(V value);
        boolean equals(Object o) //opcjonalne
    }
}

public interface SortedMap<K, V> extends Map<K, V>{
    Comparator<? super K>comparator();
    SortedMap<K, V> subMap(K fromKey, K toKey);
    SortedMap<K, V> headMap(K toKey);
    SortedMap<K, V> tailMap(K fromKey);
    K firstKey();
    K lastKey();
}
```

- public class **HashMap<K,V>** extends AbstractMap<K,V>
  implements Map<K,V>, Cloneable, Serializable

HashMap – implementacja oparta na na tablicy haszowej. Zapewnia stały czas wstawiania i wyszukiwania elementów w kolekcji. Podczas tworzenia kolekcji można określić efektywny czas wstawiania (*load factor)* i powiększania pojemności kolekcji *(capacity)*. HasMap przy przetwarzaniu elementów korzysta z ich metod *hashCode* i *equals.*

- public class **TreeMap<K,V>** extends AbstractMap<K,V>
  implements SortedMap<K,V>, Cloneable, Serializable

TreeMap – implementacja oparta na drzewie czerwono-czarnym. Elementy mogą być zwracane w sposób uporządkowany wg ich metody *compareTo.*

Wydajność operacji pojemników

wg [Thinking in Java, Second Edition Bruce Eckel**]**

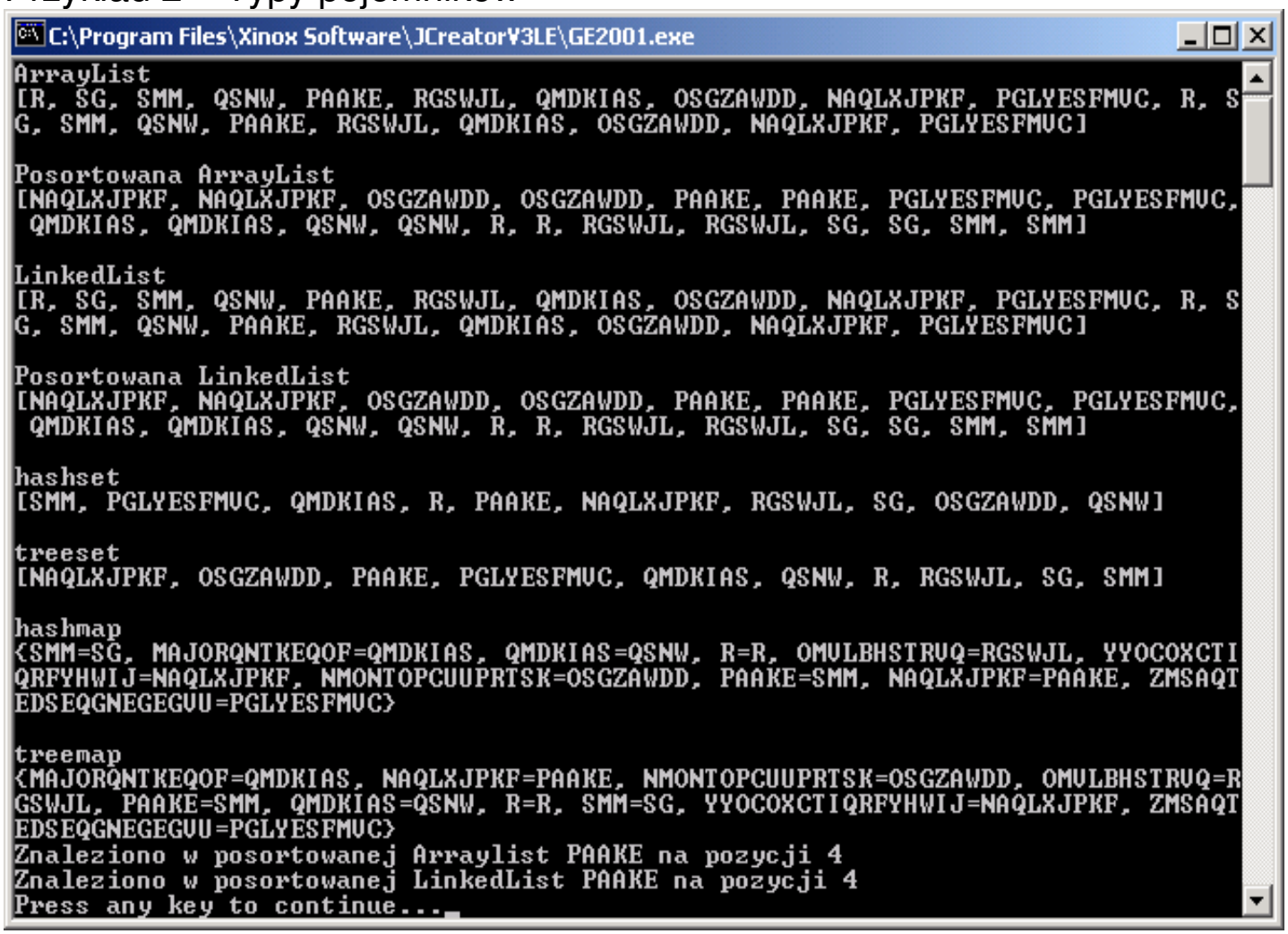
Type	**Get**	**Iteration**	**Insert**	**Remove**
array	1430	3850	na	na
ArrayList	3070	12200	500	46850
LinkedList	16320	9110	110	60
Vector	4890	16250	550	46850

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

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

Przykłady zastosowania pojemników

Przykład 2 – Typy pojemników

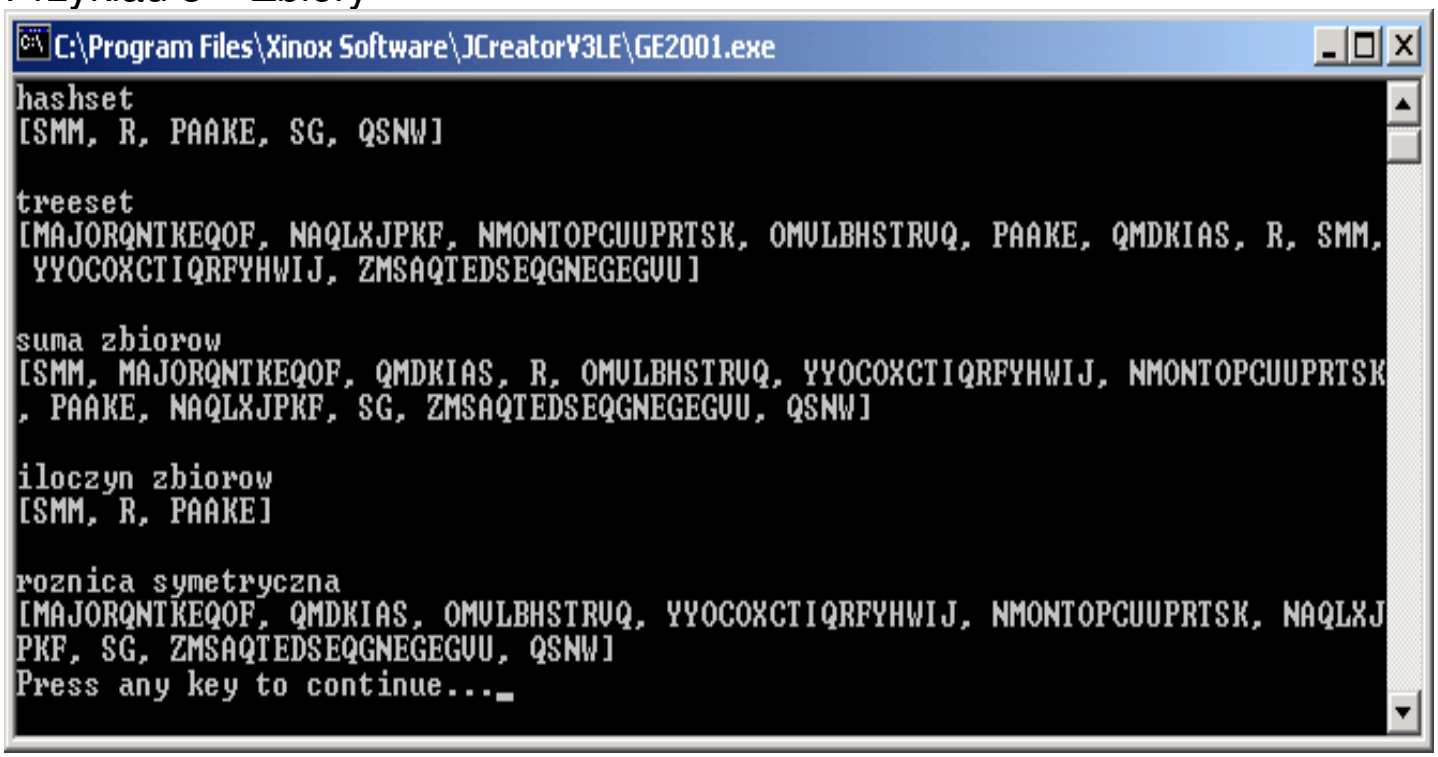



```
import java.lang.*;
import java.util.*;

public class Kolekcje2
 { static ArrayList <String> arraylist = new ArrayList<String>();
                                          //domyślna pojemność 10 elementów
   static LinkedList <String> linkedlist = new LinkedList<String>();
   static HashSet <String> hashset =new HashSet <String>();
   static TreeSet <String> treeset=new TreeSet<String>();
   static HashMap <String,String> hashmap=new HashMap<String,String>();
   static TreeMap <String,String> treemap=new TreeMap<String,String>();

  static public byte[] wypelnij(int n)
    { byte tablica1[]=new byte[n];
      Random r=new Random(n);
      for (int i=0; i<tablica1.length;i++)
        tablica1[i]=(byte)(65+r.nextInt(26));
        return tablica1; }
```

```
static String klucz2(int n)
 { return new String(wypelnij(n)); }

static public void wypelnij1(int n)
 { for (int i=0; i<n;i++)
    { String s=new String(wypelnij(i+1));
      arraylist.add(s);
    } }

static public void wypelnij2(int n)
 { for (int i=0; i<n;i++)
    { String s=new String(wypelnij(i+1));
       linkedlist.add(s);
    } }

 static public void wypelnij3(int n)
 { for (int i=0; i<n;i++)
    { String s=new String(wypelnij(i+1));
       hashset.add(s);
    } }

 static public void wypelnij4(int n)
 { for (int i=0; i<n;i++)
    { String s=new String(wypelnij(i+1));
       treeset.add(s);
    } }

 static public void wypelnij5(int n)
 { for (int i=0; i<n;i++)
    { String s1=new String(wypelnij(2*i+1));
      String s2=new String(wypelnij(i+1));
      hashmap.put(s1,s2);
    } }

 static public void wypelnij6(int n)
 { for (int i=0; i<n;i++)
    { String s1=new String(wypelnij(2*i+1));
      String s2=new String(wypelnij(i+1));
      treemap.put(s1,s2);
    } }
```

```
 public static void main(String args[])
   { wypelnij1(10);wypelnij1(10);
     wypelnij2(10);wypelnij2(10);
     wypelnij3(10);wypelnij3(10);
     wypelnij4(10);wypelnij4(10);
     wypelnij5(10);wypelnij5(10);
     wypelnij6(10);wypelnij6(10);
     System.out.println("\nArrayList\n"+arraylist.toString());
     Collections.sort(arraylist);
     System.out.println("\nPosortowana ArrayList\n"+arraylist.toString());
     System.out.println("\nLinkedList\n"+linkedlist.toString());
     Collections.sort(linkedlist);
     System.out.println("\nPosortowan LinkedList\n"+linkedlist.toString());
     System.out.println("\nhashset\n"+hashset.toString());
     System.out.println("\ntreeset\n"+treeset.toString());
     System.out.println("\nhashmap\n"+hashmap.toString());
     System.out.println("\ntreemap\n"+treemap.toString());
     String a=klucz2(5);
     int b1=Collections.binarySearch(arraylist,a);
     System.out.println("Znaleziono w posortowanej Arraylist "+a+" na pozycji "+b1);
     int b2=Collections.binarySearch(linkedlist,a);
     System.out.println("Znaleziono w posortowanej LinkedList "+a+" na pozycji "+b2);
    }
}
```

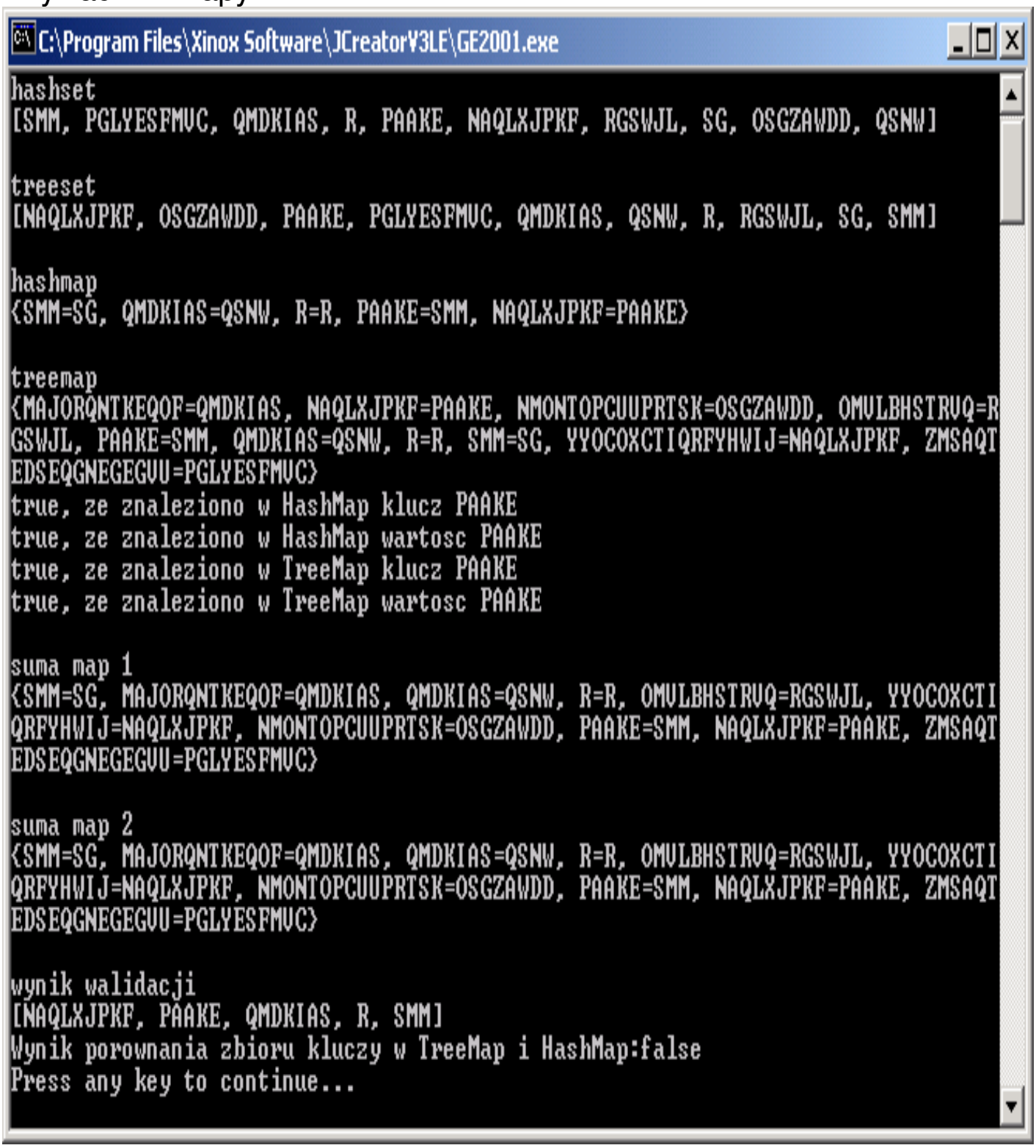
Przykład 3 – Zbiory




```
import java.lang.*;
import java.util.*;

public class Zbiory
 { static HashSet <String> hashset=new HashSet <String>();
   static TreeSet <String> treeset=new TreeSet<String>();

static  public byte[] wypelnij(int n)
  { byte tablica1[]=new byte[n];
    Random r=new Random(n);
    for (int i=0; i<tablica1.length;i++)
      tablica1[i]=(byte)(65+r.nextInt(26));
      return tablica1; }
   static public void wypelnij3(int n)
  { for (int i=0; i<n;i++)
     { String s=new String(wypelnij(i+1));
            hashset.add(s);
     } }
    static public void wypelnij4(int n)
  { for (int i=0; i<n;i++)
     { String s=new String(wypelnij(2*i+1));
            treeset.add(s);
     } }
 static <K> void roznicasymetryczna(Set<K> set1, Set<K> set2)
   {
    Set<K> roznicasym = new HashSet<K>(set1);
    roznicasym.addAll(set2);
    System.out.println("\nsuma zbiorow\n"+roznicasym.toString());
    Set<K> tmp = new HashSet<K>(set1);
    tmp.retainAll(set2);
    System.out.println("\niloczyn zbiorow\n"+tmp.toString());
    roznicasym.removeAll(tmp);
    System.out.println("\nroznica symetryczna\n"+roznicasym.toString());    }
```

```
public static void main(String args[])
   { wypelnij3(5);wypelnij3(5);
     wypelnij4(10);wypelnij4(10);
     System.out.println("\nhashset\n"+hashset.toString());
     System.out.println("\ntreeset\n"+treeset.toString());
     roznicasymetryczna(hashset,treeset);
    }
```

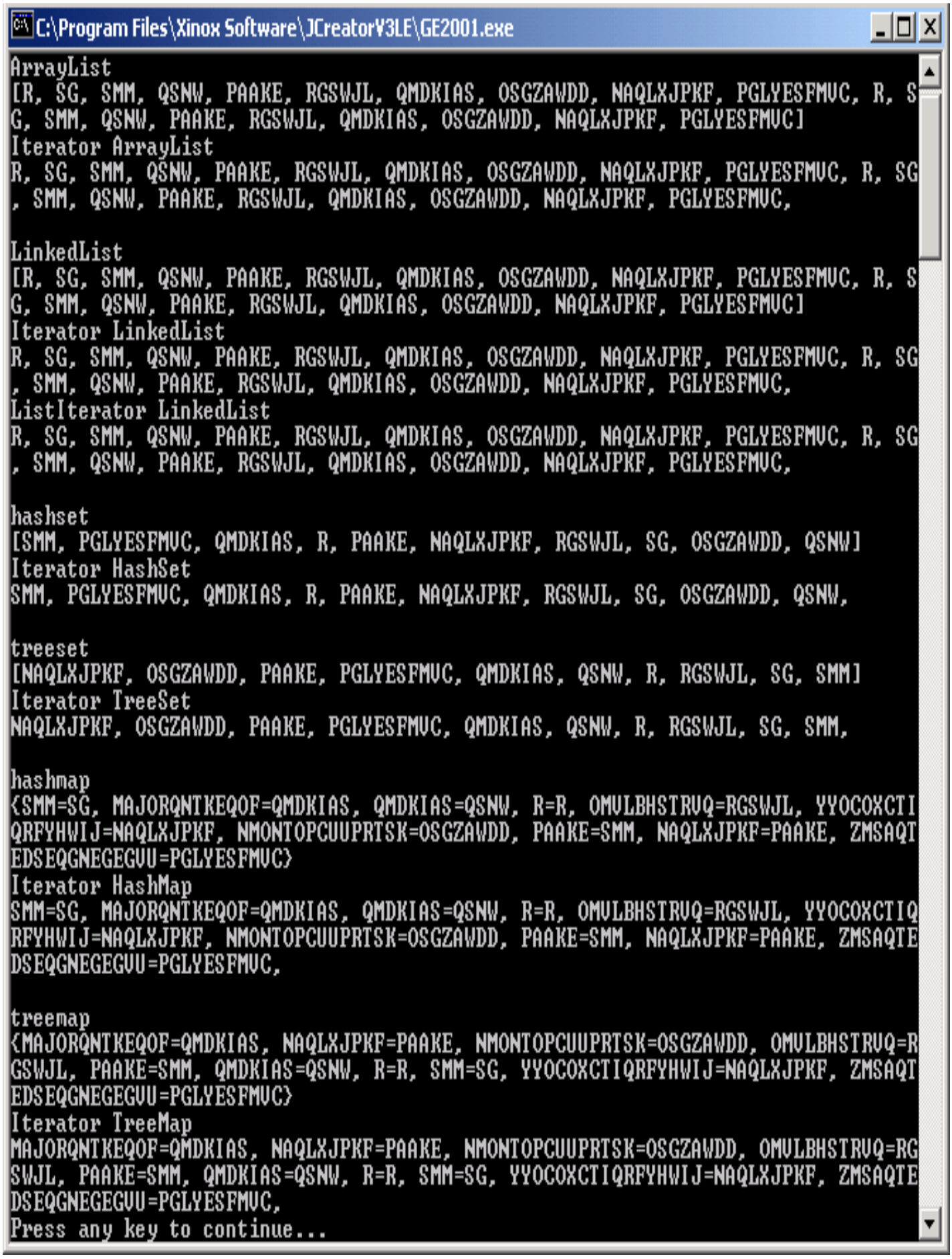
Przykład 4 – Mapy



```
import java.lang.*;
import java.util.*;

public class Mapy
 { static HashSet <String> hashset=new HashSet <String>();
   static TreeSet <String> treeset=new TreeSet<String>();
   static HashMap <String,String> hashmap=new HashMap<String,String>();
   static TreeMap <String,String> treemap=new TreeMap<String,String>();

 static public byte[] wypelnij(int n)
   { byte tablica1[]=new byte[n];
     Random r=new Random(n);
     for (int i=0; i<tablica1.length;i++)
       tablica1[i]=(byte)(65+r.nextInt(26));
       return tablica1;
   }

 static String klucz2(int n)
   { return new String(wypelnij(n)); }

 static public void wypelnij3(int n)
   { for (int i=0; i<n;i++)
     { String s=new String(wypelnij(i+1));
             hashset.add(s);
     } }

 static public void wypelnij4(int n)
   { for (int i=0; i<n;i++)
     { String s=new String(wypelnij(i+1));
             treeset.add(s);
     } }

 static public void wypelnij5(int n)
   { for (int i=0; i<n;i++)
     { String s1=new String(wypelnij(2*i+1));
       String s2=new String(wypelnij(i+1));
       hashmap.put(s1,s2);
     } }

 static public void wypelnij6(int n)
   { for (int i=0; i<n;i++)
     { String s1=new String(wypelnij(2*i+1));
       String s2=new String(wypelnij(i+1));
       treemap.put(s1,s2);
     } }

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

static <K, V> Set<K>  walidacja(Map<K, V> podstawowa, Set<K> wzorzec)
   {
     Set<K> zle = new TreeSet<K>(wzorzec);
     Set<K> klucze = podstawowa.keySet();
     If (!klucze.containsAll(wzorzec))
       { zle.retainAll(klucze)  }
     return zle;  }
```

```
public static void main(String args[])
   {
    wypelnij3(10);wypelnij3(10);
    wypelnij4(10);wypelnij4(10);
    wypelnij5(5);wypelnij5(5);
    wypelnij6(10);wypelnij6(10);
    System.out.println("\nhashset\n"+hashset.toString());
    System.out.println("\ntreeset\n"+treeset.toString());
    System.out.println("\nhashmap\n"+hashmap.toString());
    System.out.println("\ntreemap\n"+treemap.toString());

    String a=klucz2(5);
    boolean b1=hashmap.containsKey(a);
    System.out.println(b1+", ze znaleziono w HashMap klucz "+a);

    boolean b2=hashmap.containsValue(a);
    System.out.println(b2+", ze znaleziono w HashMap wartosc "+a);

    b1=treemap.containsKey(a);
    System.out.println(b1+", ze znaleziono w TreeMap klucz "+a);

    b2=treemap.containsValue(a);
    System.out.println(b2+", ze znaleziono w TreeMap wartosc "+a);

    Map <String, String> sumamap_ = sumamap(treemap,hashmap);
    System.out.println("\nsuma map 1\n"+sumamap_.toString());

    sumamap_ = sumamap(hashmap,treemap);
    System.out.println("\nsuma map 2\n"+sumamap_.toString());

    Set <String>klucze_walidacji=walidacja(treemap,hashset);
    // w wyniku walidacji zostaną podane te klucze z treemap, które wystąpiły w hashset
    System.out.println("\nwynik walidacji\n"+klucze_walidacji.toString());
   System.out.println("Wynik porownania zbioru kluczy w TreeMap i  HashMap:"
                      +treemap.keySet().equals(hashmap.keySet()));
   }
}
```

Przykład 5 – Iteratory



```
import java.lang.*;
import java.util.*;

public class Kolekcje3                                  //domyślna pojemność 10 elementów
 { static ArrayList <String> arraylist= new ArrayList<String>();
   static LinkedList <String> linkedlist= new LinkedList<String>();
   static HashSet <String> hashset=new HashSet <String>();
   static TreeSet <String> treeset=new TreeSet<String>();
   static HashMap <String,String> hashmap=new HashMap<String,String>();
   static TreeMap <String,String> treemap=new TreeMap<String,String>();

static public byte[] wypelnij(int n)
    { byte tablica1[]=new byte[n];
      Random r=new Random(n);
      for (int i=0; i<tablica1.length;i++)
        tablica1[i]=(byte)(65+r.nextInt(26));
        return tablica1;}

   static public void wypelnij1(int n)
    { for (int i=0; i<n;i++)
      { String s=new String(wypelnij(i+1));
            arraylist.add(s);
      } }

   static public void wypelnij2(int n)
    { for (int i=0; i<n;i++)
      { String s=new String(wypelnij(i+1));
            linkedlist.add(s);
      } }

    static public void wypelnij3(int n)
    { for (int i=0; i<n;i++)
      { String s=new String(wypelnij(i+1));
            hashset.add(s);
      } }

    static public void wypelnij4(int n)
    { for (int i=0; i<n;i++)
      { String s=new String(wypelnij(i+1));
            treeset.add(s);
      } }

    static public void wypelnij5(int n)
    { for (int i=0; i<n;i++)
      { String s1=new String(wypelnij(2*i+1));
        String s2=new String(wypelnij(i+1));
        hashmap.put(s1,s2);
      } }

    static public void wypelnij6(int n)
    { for (int i=0; i<n;i++)
      { String s1=new String(wypelnij(2*i+1));
        String s2=new String(wypelnij(i+1));
        treemap.put(s1,s2);
      } }
```

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

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


 public static void main(String args[])
   {
     wypelnij1(10);wypelnij1(10);
     wypelnij2(10);wypelnij2(10);
     wypelnij3(10);wypelnij3(10);
     wypelnij4(10);wypelnij4(10);
     wypelnij5(10);wypelnij5(10);
     wypelnij6(10);wypelnij6(10);
     System.out.println("\nArrayList\n"+arraylist.toString());
     wyswietlIterator("Iterator ArrayList",arraylist.iterator());

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

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

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

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

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