Programowanie obiektowe

Wykład 8:
Tablice i kolekcje obiektów.

Tablice

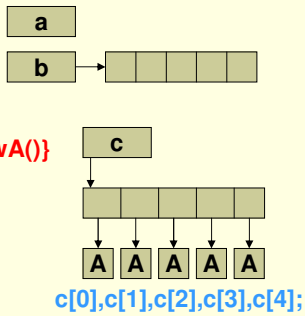
class A {...}

A [ ] a;

A [ ] b=new A[5];

A [ ] c = {new A(), new A(),
new A(), new A(), newA()}

c.length == 5



c[0],c[1],c[2],c[3],c[4];

Klasa Arrays (java.util) - statyczne metody: equals(), deepEquals(), binarySearch(), fill(), sort(), toString(), asList()
działające na tablicach dowolnych typów (też obiektów)

Operacje na tablicach

Wypełnianie tablic:

```
char [ ] a= new char[10];
Arrays.fill(a,'x');          - wypełnianie całej tablicy
Arrays.fill(a,4,8,'y');      - wypełnianie od a[4] do a[8]
```

Kopiowanie tablic:

```
Integer [ ] b= new Integer[5];
Integer [ ] c= new Integer[10];
System.arraycopy(a,0,b,5,a.length); od a[0] do b[5] a.length elementów
```

Porównywanie tablic:

```
String [ ] s1 = {"Test","Test","Test"};
String [ ] s2 = new String[3];
Arrays.fill(s2,"Test");
Arrays.equals(s1,s2);          - porównywanie płytkie (False)
Arrays.deepEquals(s1,s2);  - porównywanie głębokie (True)
```


Operacje na tablicach, cd.

Sortowanie tablic:

```
String [ ] s = new String[10];
…
Arrays.sort(s);          - sortowanie całej tablicy
Arrays.sort(a,4,8);      - sortowanie od a[4] do a[8]
```

```
Interface Comparable<T>;  =>  int compareTo(T o)
Interface Comparator<T>;  =>  int compare (T o1, T o2)
                                          -1  => o1<o2
                                           0  => o1==o2
                                           1  => o1>o2
```

```
public class MojComp implements Comparator<Object>;
 public int compare (Object o1, Object o2){
   String s1 = (String) o1;
   String s2 = (String o2;
   return s1.toLowerCase().compareTo(s2.toLowerCase());
 }
}
```

```
Arrays.sort(s, new MojComp());
Arrays.sort(s, 4, 8,
            new MojComp()
```

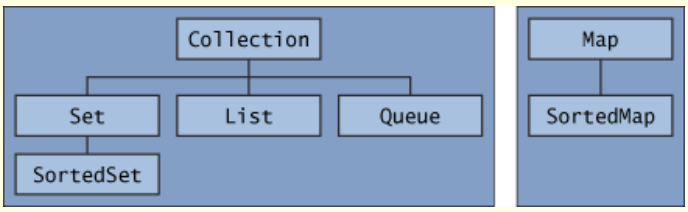
Operacje na tablicach, cd.

Przeszukiwanie tablic posortowanych:

```
String [ ] s = new String[10];
…
int index = Arrays.binarySearch(s,”Test”);
        0..length-1                 => pozycja znaleziona
        -pozycja wstawienia-1       => gdy nie znaleziono
```

```
int index = Arrays.binarySearch(s,”Test”, new MojComp());
```


Kolekcje obiektów



interfejsy kolekcji (java.util)

Interfejs Collection

```
public interface Collection <E> extends Iterable<E> {
        // Operacje podstawowe
int size();
boolean isEmpty();
boolean contains(Object element);
boolean add(E element);
boolean remove(Object element);
Iterator<E> iterator();
        // Operacje na całych kolekcjach
boolean containsAll(Collection c);
boolean addAll(Collection<? extends E> c);
boolean removeAll(Collection<?> c);
boolean retainAll(Collection<?> c);
void clear();
        // Operacje na tablicach
Object[ ] toArray();
<T> T[ ] toArray(T[ ] a);
}
```

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


Interfejs Set

```
public interface Set<E> extends Collection<E> {
}
```

Nie może zawierać zduplikowanych elementów.

Interfejs List

```
public interface List<E> extends Collection<E> {
                    // Dostęp wg. pozycji
  E get(int index);
  E set(int index, E element);
  boolan add(E element);
  void add(int index, E element);
  E remove(int index);
  boolean addAll(int index, Collection<? extends E>
            c);
                    // Szukanie
  int indexOf(Object o);
  int lastIndexOf(Object o);
                    // Iteratory
  ListIterator<E> listIterator();
  ListIterator<E> listIterator(int index);
                    // Podlista (bez elementu „to”)
  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();
void set(E o);
void add(E o);
}
```


Interfejs Queue

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

Operacja	Sygnalizacja wyjątku	Wartość null lub false
Dodanie elementu	**add()**	**offer()**
Usunięcie elementu	**remove()**	**poll()**
Sprawdzenie	**element()**	**peek()**

Interfejs Map

```
public interface Map <K,V>{
        // Operacje podstawowe
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();
        // Operacja na mapie
void putAll(Map<? extends V> m);
void clear();
        // Operacja na kolekcjach
public Set<K> keySet();
public Collection<V> values();
public Set<Map.Entry<K,V>> entrySet();
```

```
// interfejs zbioru zwracanego przez
   metodę entrySet()
public interface Entry {
    K getKey();
    V getValue();
    V setValue(V value);
 }
}
```

Klucz	Wartość

…

Klucz	Wartość

Interfejs SortedSet

```
public interface SortedSet<E> extends Set<E> {
        //Zakresy
  SortedSet<E> subSet(E fromElement, E toElement);
  SortedSet<E> headSet(E toElement);  //bez „to”
  SortedSet<E> tailSet(E fromElement);
        // dostęp do skrajnych elementów
  E first();
  E last();
        // Komparator
  Comparator<? super E> comparator();
}
```

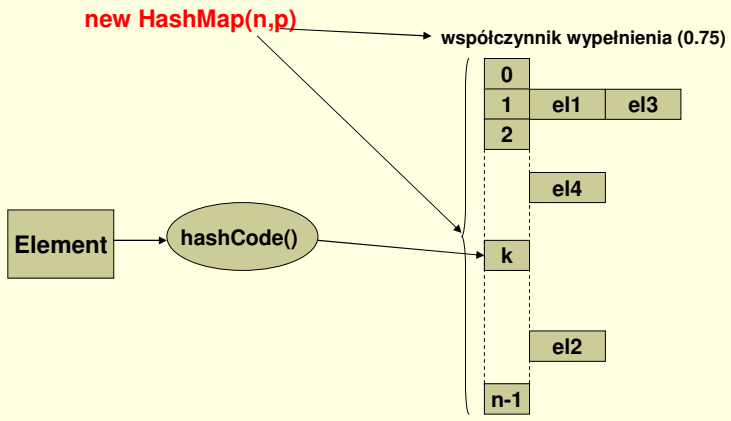
Interfejs SortedMap

```
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();
}
```


Podstawowe implementacje kolekcji

		Implementacja				
		Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table +Linked List
Interfejs	Set	HashSet		TreeSet		LinkedHashSet
	List		ArrayList		LinkedList	
	Queue				LinkedList	
	Map	HashMap		TreeMap		LinkedHashMap

Kolekcje z funkcją haszującą



Wyspecjalizowane implementacje kolekcji

- EnumSet, CopyOnWriteArraySet
- CopyOnWriteArrayList
- EnumMap, WeakHashMap, IdentityHashMap, ConcurrentHashMap
- PriorityQueue, LinkedBlockingQueue, ArrayBlockingQueue, PriorityBlockingQueue, DelayQueue, SynchronousQueue

Klasa Collections

Metody Statyczne:

sort()	**- sortowanie list**
binarySearch()	**- wyszukiwanie binarne na liście**
copy()	**- kopiowanie elementów z jednej listy do drugiej**
shuffle()	**- permutacja elementów na liście**
reverse()	**- odwrócenie listy**
max(), min()	**- najwiekszy i najmniejszy element w kolekcji**
unmodifiableMap()	**- utworzenie niemodyfikowalnych kolekcji**
unmodifiableSet()	
unmodifiableList()	
unmodifiableCollection()	

Instrukcja for-each

for (T zmienna : kolekcja_lub_tablica)
instrukcja;

- int[] numbers = {1,2,3,4,5,6,7,8,9,10};
  for (int item : numbers) {
  System.out.println("Count is: " + item); }

Przykład

```
import java.util.*;
public class PrintingContainers {
  static Collection<String> fill(Collection<String> c) {
    c.add("pies");
    c.add("pies");
    c.add("kot");
    return c;
  }
  static Map<String,String> fill(Map<String,String> m) {
    m.put("pies", "As");
    m.put("pies", "Pluto");
    m.put("kot", "Filemon");
    return m;
  }
  public static void main(String[] args) {
    System.out.println(fill(new ArrayList<String>()));
    System.out.println(fill(new HashSet<String>()));
    System.out.println(fill(new
            HashMap<String,String>()));
  }
}
```

Wynik:
[pies, pies, kot]
[pies,kot]
[kot=Filemon, pies=Pluto]

Pytania

1. Do czego służy i w jakich kolekcjach występuje iterator?
2. Różnice pomiędzy zbiorem a listą.
3. Jak przeglądać wszystkie elementy w mapie?

Koniec