Programowanie i projektowanie obiektowe

Kolekcje

Paweł Daniluk

Wydział Fizyki

Jesień 2012

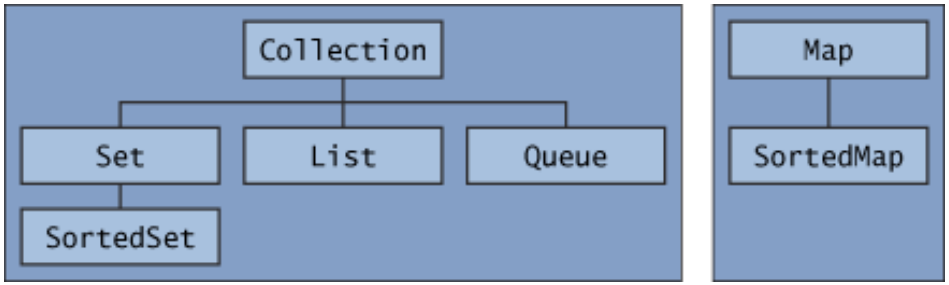
Kolekcje

- Interfejsy
- Implementacje
- Algorytmy

Marketspeak

- Mniejszy wysiłek programistyczny
- Wyższa jakość i wydajność programu
- Możliwość współpracy pomiędzy niezwiązanymi API
- Łatwiejsza nauka nowych API
- Łatwiej tworzyć nowe API
- Możliwość recyklingu istniejących programów

Kolekcje – interfejsy



Interfejsy generyczne np. `List<E>`.

- `Set` – Nie zawiera powtórzeń
- `List` – Uporządkowana, może zawierać powtórzenia
- `Queue` – Kolejka, może być FIFO, ale nie musi.
- `Map` – Zawiera parowania kluczy i wartości. Klucze nie mogą się powtarzać.

Iterable

Iterable

```
public interface Iterable<E> {
    Iterator<E> iterator();
}
```

Iterator

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

for-each

```
for (Object o : iterable)
    System.out.println(o);
```

Collection

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

    // Bulk operations
    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c); //optional
    boolean removeAll(Collection<?> c);        //optional
    boolean retainAll(Collection<?> c);        //optional
    void clear();                              //optional

    // Array operations
    Object[] toArray();
    <T> T[] toArray(T[] a);
}
```

Obchodzenie kolekcji

for-each

```
for (Object o : collection)
    System.out.println(o);
```

Przykład

```
static void filter(Collection<?> c) {
    for (Iterator<?> it = c.iterator(); it.hasNext(); )
        if (!cond(it.next()))
            it.remove();
}
```

Metoda `remove()` nie musi być zaimplementowana.

Set

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

    // Bulk operations
    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c); //optional
    boolean removeAll(Collection<?> c);        //optional
    boolean retainAll(Collection<?> c);        //optional
    void clear();                              //optional

    // Array Operations
    Object[] toArray();
    <T> T[] toArray(T[] a);
}
```

Set – przykłady

Usuwanie powtórzeń

```
public static <E> Set<E> removeDups(Collection<E> c) {
    return new LinkedHashSet<E>(c);
}
```

Set – przykłady

Usuwanie powtórzeń

```
public static <E> Set<E> removeDups(Collection<E> c) {
    return new LinkedHashSet<E>(c);
}
```

Powtórzenia jeszcze raz

```
public class FindDups {
    public static void main(String[] args) {
        Set<String> s = new HashSet<String>();
        for (String a : args)
            if (!s.add(a))
                System.out.println("Duplicate detected: " + a);

        System.out.println(s.size() + " distinct words: " + s);
    }
}
```

Set – przykłady c.d.

- `containsAll(Collection<?> c)` — `true` jeżeli zbiór zawiera wszystkie elementy `c`
- `addAll(Collection<?> c)` — dodaje wszystkie elementy z `c`
- `retainAll(Collection<?> c)` — usuwa wszystkie elementy, które nie należą do `c`
- `removeAll(Collection<?> c)` — usuwa wszystkie elementy, które należą do `c`

Set – przykłady c.d.

- `containsAll(Collection<?> c)` — `true` jeżeli zbiór zawiera wszystkie elementy `c`
- `addAll(Collection<?> c)` — dodaje wszystkie elementy z `c`
- `retainAll(Collection<?> c)` — usuwa wszystkie elementy, które nie należą do `c`
- `removeAll(Collection<?> c)` — usuwa wszystkie elementy, które należą do `c`

```
Set<Type> union = new HashSet<Type>(s1);
union.addAll(s2);

Set<Type> intersection = new HashSet<Type>(s1);
intersection.retainAll(s2);

Set<Type> difference = new HashSet<Type>(s1);
difference.removeAll(s2);
```

Set – przykłady c.d.

Powtórzenia po raz trzeci

```
public class FindDups2 {
    public static void main(String[] args) {
        Set<String> uniques = new HashSet<String>();
        Set<String> dups    = new HashSet<String>();

        for (String a : args)
            if (!uniques.add(a))
                dups.add(a);

// Destructive set-difference
        uniques.removeAll(dups);

        System.out.println("Unique words:    " + uniques);
        System.out.println("Duplicate words: " + dups);
    }
}
```

List

```
public interface List<E> extends Collection<E> {
    // Positional access
    E get(int index);
    E set(int index, E element);    //optional
    boolean add(E element);         //optional
    void add(int index, E element); //optional
    E remove(int index);            //optional
    boolean addAll(int index,
        Collection<? extends E> c); //optional

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

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

    // Range-view
    List<E> subList(int from, int to);
}
```

List – przykłady

Zamiana elementów

```
public static <E> void swap(List<E> a, int i, int j) {
    E tmp = a.get(i);
    a.set(i, a.get(j));
    a.set(j, tmp);
}
```

Podzakres

```
subList(int fromIndex, int toIndex)
```

Podobnie do

```
for (int i = fromIndex; i < toIndex; i++) {
    ...
}
```

List – przykłady c.d.

ListIterator

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

Queue

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

Operacja	Rzuca wyjątek	Zwraca `null` lub `false`
dodawanie	`add(e)`	`offer(e)`
usuwanie	`remove(e)`	`poll()`
sprawdzanie	`element(e)`	`peek()`

Queue – przykłady

Odliczanie

```
import java.util.*;

public class Countdown {
    public static void main(String[] args)
            throws InterruptedException {
        int time = Integer.parseInt(args[0]);
        Queue<Integer> queue = new LinkedList<Integer>();
        for (int i = time; i >= 0; i--)
            queue.add(i);
        while (!queue.isEmpty()) {
            System.out.println(queue.remove());
            Thread.sleep(1000);
        }
    }
}
```

Queue – przykłady

Sortowanie – kolejki priorytetowe

```
static <E> List<E> heapSort(Collection<E> c) {
    Queue<E> queue = new PriorityQueue<E>(c);
    List<E> result = new ArrayList<E>();
    while (!queue.isEmpty())
        result.add(queue.remove());
    return result;
}
```

Map

```
public interface Map<K,V> {
    // Basic operations
    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();

    // Bulk operations
    void putAll(Map<? extends K, ? extends V> m);
    void clear();

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

    // Interface for entrySet elements
    public interface Entry {
        K getKey();
        V getValue();
        V setValue(V value);
    }
}
```

Map – przykłady

Odliczanie

```
import java.util.*;

public class Freq {
    public static void main(String[] args) {
        Map<String, Integer> m = new HashMap<String, Integer>();

        // Initialize frequency table from command line
        for (String a : args) {
            Integer freq = m.get(a);
            m.put(a, (freq == null) ? 1 : freq + 1);
        }

        System.out.println(m.size() + " distinct words:");
        System.out.println(m);
    }
}
```

Map – widoki

- keySet – zbiór kluczy
- values – zbiór wartości.
- entrySet – zbiór par klucz-wartość

```
// Filter a map based on some property of its keys.
for (Iterator<Type> it = m.keySet().iterator(); it.hasNext(); )
    if (it.next().isBogus())
        it.remove();
```

```
for (Map.Entry<KeyType, ValType> e : m.entrySet())
    System.out.println(e.getKey() + ": " + e.getValue());
```

Implementacje

- ogólnego stosowanie
- specjalne
- współbieżne
- owijające
- abstrakcyjne

Implementacje ogólnego stosowania

Interfejs	Tablice haszujące	Tablice	Drzewa	Łączone listy	Mieszane
`Set`	`HashSet`		`TreeSet`		`LinkedHashSet`
`List`		`ArrayList`		`LinkedList`	
`Map`	`HashMap`		`TreeMap`		`LinkedHashMap`

Sortowanie

```
public class Sort {
    public static void main(String[] args) {
        List<String> list = Arrays.asList(args);
        Collections.sort(list);
        System.out.println(list);
    }
}
```

Dowolny porządek

```
mySort(List<Integer> list) {
    Comparator<Integer> comp=new Comparator<Integer>() {
        public int compare(Integer a, Integer b) {
            return a-b;
        }
    }

    Collections.sort(list, comp);
}
```

Inne sztuki na elementach

- `shuffle`
- `reverse`
- `swap`
- `addAll`
- `frequency`
- `disjoint`
- `min`, `max`

Wyszukiwanie binarne

```
int pos = Collections.binarySearch(list, key);
```

Własne implementacje

```
private static class MyArrayList<T> extends AbstractList<T> {
    private final T[] a;

    MyArrayList(T[] array) {
        a = array;
    }

    public T get(int index) {
        return a[index];
    }

    public T set(int index, T element) {
        T oldValue = a[index];
        a[index] = element;
        return oldValue;
    }

    public int size() {
        return a.length;
    }
}
```

Zadanie 1 – Zastosowania kolekcji

Zadanie

Rozwiąż jeszcze raz zadania w poprzednich prac domowych stosując odpowiednie kolekcje.

Wskazówka

Chodzi o zadania: II.(6,7,8), III.1, V.1, IX.3

Zadanie 2 – Własna implementacja zbioru

Zadanie

Rozszerz klasę `AbstractSet`, aby implementowała interfejs `Set` dla elementów, które można porównywać (implementują interfejs `Comparable`). Użyj drzew wyszukiwania binarnego.