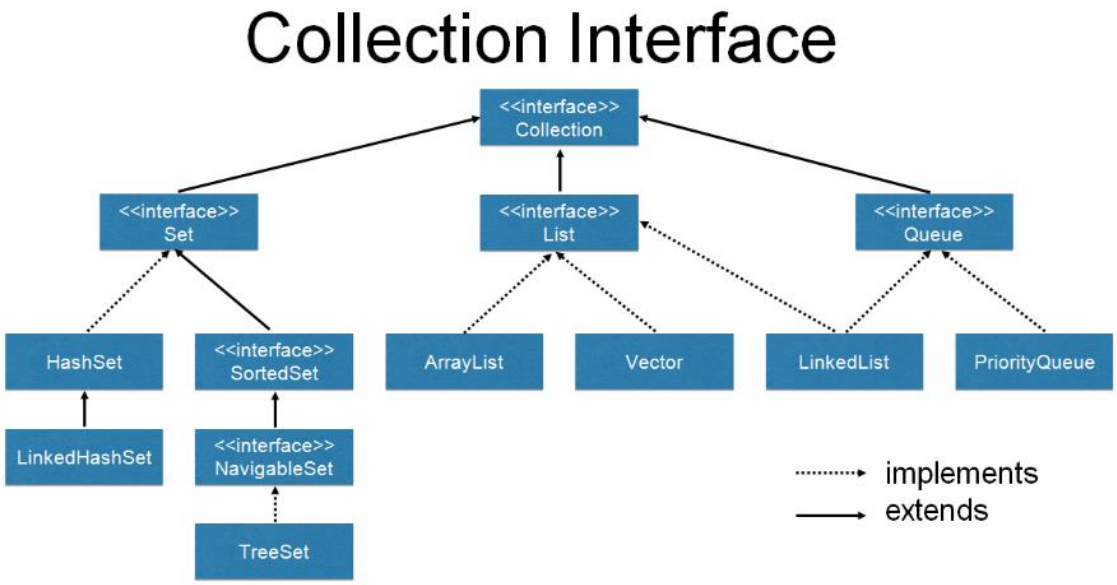
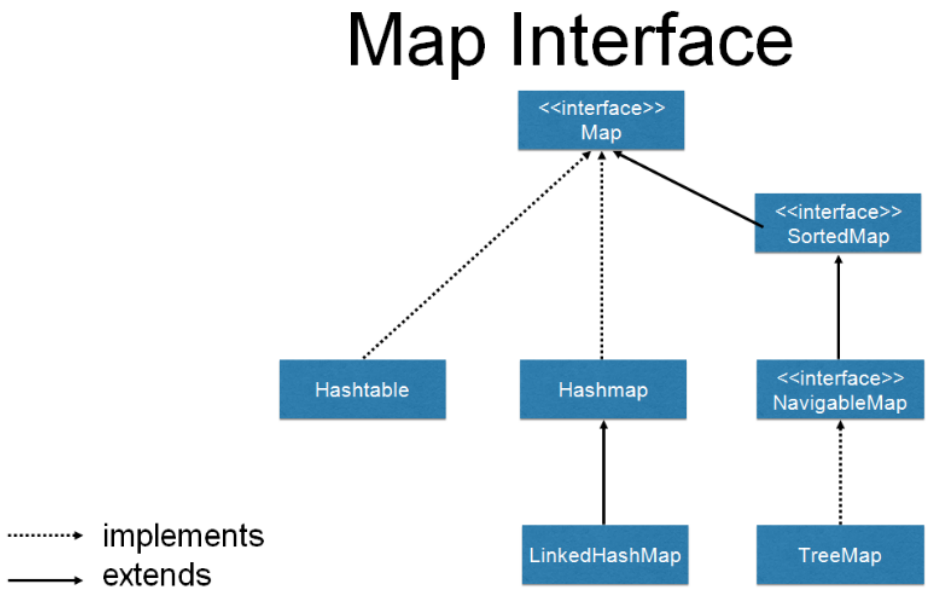
Kolekcje

Kolekcje przechowują dane tego samego typu (podobnie jak tablice). Jednak ich rozmiar może się zmieniać dynamicznie. Kolekcja to sposób grupowania obiektów

Source code of examples

Hierarchia dziedziczenia





Interfejs List

Rozmiar listy jest automatycznie powiększany wraz z dodawaniem nowych elementów. Listy z definicji są kolekcjami dla których **kolejność elementów jest istotna**, **mogą przechowywać ten sam obiekt po kilka razy**.

Przykładowe implementacje to **java.util.LinkedList** oraz **java.util.ArrayList**.

LinkedList najlepiej stosować gdy często usuwamy elementy, a **ArrayList** stosujemy gdy często pobieramy losowe elementy

Najczęściej używane metody

add – dodaje element do listy,

addAll – dodaje wszystkie elementy z kolekcji przekazanej jako parametr

contains – sprawdza, czy dany element znajduje się na liście

isEmpty – czy lista jest pusta,

size – rozmiar listy,

indexOf – indeks pierwszego wystąpienia elementu przekazanego jako parametr

lastIndexOf – indeks ostatniego wystąpienia elementu przekazanego jako parametr

Iterowanie po listach

```
public static void main(String[] args) {
   List<String> list= Arrays.asList("Kasia","Tomasz","Jacek");

//fori
   for (int i = 0; i < list.size(); i++) {
      System.out.println(list.get(i));
   }

   //foreach
   for (String string:list
       ) {
      System.out.println(string);
   }
}
```

Interfejs Set

Zbiór (ang. *set*) to kolekcja, która służy do przechowywania unikalnych elementów. Nie jest istotna kolejność dodawanych elementów.

Przechowywania unikalnych elementów pilnują metody **equals()** i **hashCode()**.

Najcześciej używane metody:

add – dodaje element do listy,

addAll – dodaje wszystkie elementy z kolekcji przekazanej jako parametr

contains – sprawdza, czy dany element znajduje się na liście

isEmpty – czy lista jest pusta,

size – rozmiar listy,

Iterowanie po set

```
public static void main(String[] args) {
   Set<String> set=Set.copyOf(Arrays.asList("Kasia","Tomasz","Jacek"));

//foreach
   for (String string:set
       ) {
      System.out.println(string);
   }
}
```

Interfejs Map

Map to struktura która przechowuje odwzorowanie zbioru kluczy na listę wartości. Klucze są unikalne. Kluczami mogą być obiekty typu **Immutable**.

Najczęściej używane metody:

put – wkłada klucz i wartość do mapy,

putAll – dodaje wszystkie elementy z mapy,

containsKey – sprawdza czy klucz istnieje,

containsValue – sprawdza cz wartość już istnieje,

isEmpty – czy mapa jest pusta,

size –rozmiar mapy,

remove – usuwa klucz z mapy,

get – zwraca wartość dla podanego klucza

Iterowanie po mapie

```
public static void main(String[] args) {
   Map<String,Integer> map=new HashMap<>();
   map.put("Kasia",22);
   map.put("Tomasz",36);
   map.put("Jacek",30);

//keys
   for (String key:map.keySet()
       ) {
      System.out.println(key);
   }
```

```
    //values
    for (Integer value:map.values()
        ) {
        System.out.println(value);
    }
    //key and values
    for (Map.Entry<String,Integer> element:map.entrySet()
        ) {
        System.out.println("Key :"+element.getKey());
        System.out.println("Value :"+element.getValue());
    }
}
```

Porównania

Collection class	Random access by index / key	Search / Contains	Insert
ArrayList	O(1)	O(n)	O(n)
HashSet	O(1)	O(1)	O(1)
HashMap	O(1)	O(1)	O(1)
TreeMap	O(log(n))	O(log(n))	O(log(n))

JAVA COLLECTIONS

Cheat Sheet

List	Add	Remove	Get	Contains	Next	Data Structure
ArrayList	O(1)	O(n)	O(1)	O(n)	O(1)	Array
LinkedList	O(1)	O(1)	O(n)	O(n)	O(1)	Linked List
CopyOnWriteArrayList	O(n)	O(n)	O(1)	O(n)	O(1)	Array

Set	Add	Remove	Contains	Next	Size	Data Structure
HashSet	O(1)	O(1)	O(1)	O(h/n)	O(1)	Hash Table
LinkedHashSet	O(1)	O(1)	O(1)	O(1)	O(1)	Hash Table + Linked List
EnumSet	O(1)	O(1)	O(1)	O(1)	O(1)	Bit Vector
TreeSet	O(log n)	O(log n)	O(log n)	O(log n)	O(1)	Redblack tree
CopyOnWriteArraySet	O(n)	O(n)	O(n)	O(1)	O(1)	Array
ConcurrentSkipListSet	O(log n)	O(log n)	O(log n)	O(1)	O(n)	Skip List

Map	Put	Remove	Get	ContainsKey	Next	Data Structure
HashMap	O(1)	O(1)	O(1)	O(1)	O(h / n)	Hash Table
LinkedHashMap	O(1)	O(1)	O(1)	O(1)	O(1)	Hash Table + Linked List
IdentityHashMap	O(1)	O(1)	O(1)	O(1)	O(h / n)	Array
WeakHashMap	O(1)	O(1)	O(1)	O(1)	O(h / n)	Hash Table
EnumMap	O(1)	O(1)	O(1)	O(1)	O(1)	Array
TreeMap	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)	Redblack tree
ConcurrentHashMap	O(1)	O(1)	O(1)	O(1)	O(h / n)	Hash Tables
ConcurrentSkipListMap	O(log n)	O(log n)	O(log n)	O(log n)	O(1)	Skip List

Queue	Offer	Peak	Poll	Remove	Size	Data Structure
PriorityQueue	O(log n)	O(1)	O(log n)	O(n)	O(1)	Priority Heap
LinkedList	O(1)	O(1)	O(1)	O(1)	O(1)	Array
ArrayDequeue	O(1)	O(1)	O(1)	O(n)	O(1)	Linked List
ConcurrentLinkedQueue	O(1)	O(1)	O(1)	O(n)	O(1)	Linked List
ArrayBlockingQueue	O(1)	O(1)	O(1)	O(n)	O(1)	Array
PrioriritыBlockingQueue	O(log n)	O(1)	O(log n)	O(n)	O(1)	Priority Heap
SynchronousQueue	O(1)	O(1)	O(1)	O(n)	O(1)	None
DelayQueue	O(log n)	O(1)	O(log n)	O(n)	O(1)	Priority Heap
LinkedBlockingQueue	O(1)	O(1)	O(1)	O(n)	O(1)	Linked List

Ciekawostki

Ciekawostka: Istnieją implementacje kolekcji, które pozwalają na używanie typów prymitywnych

Ciekawostka : Od Javy 1.7 nie jest wymagana nazwa klasy w drugim nawiasie <>

Ciekawostka : **Ekstencja klasy** to obiekt przechowujący wszystkie obiekty danej klasy