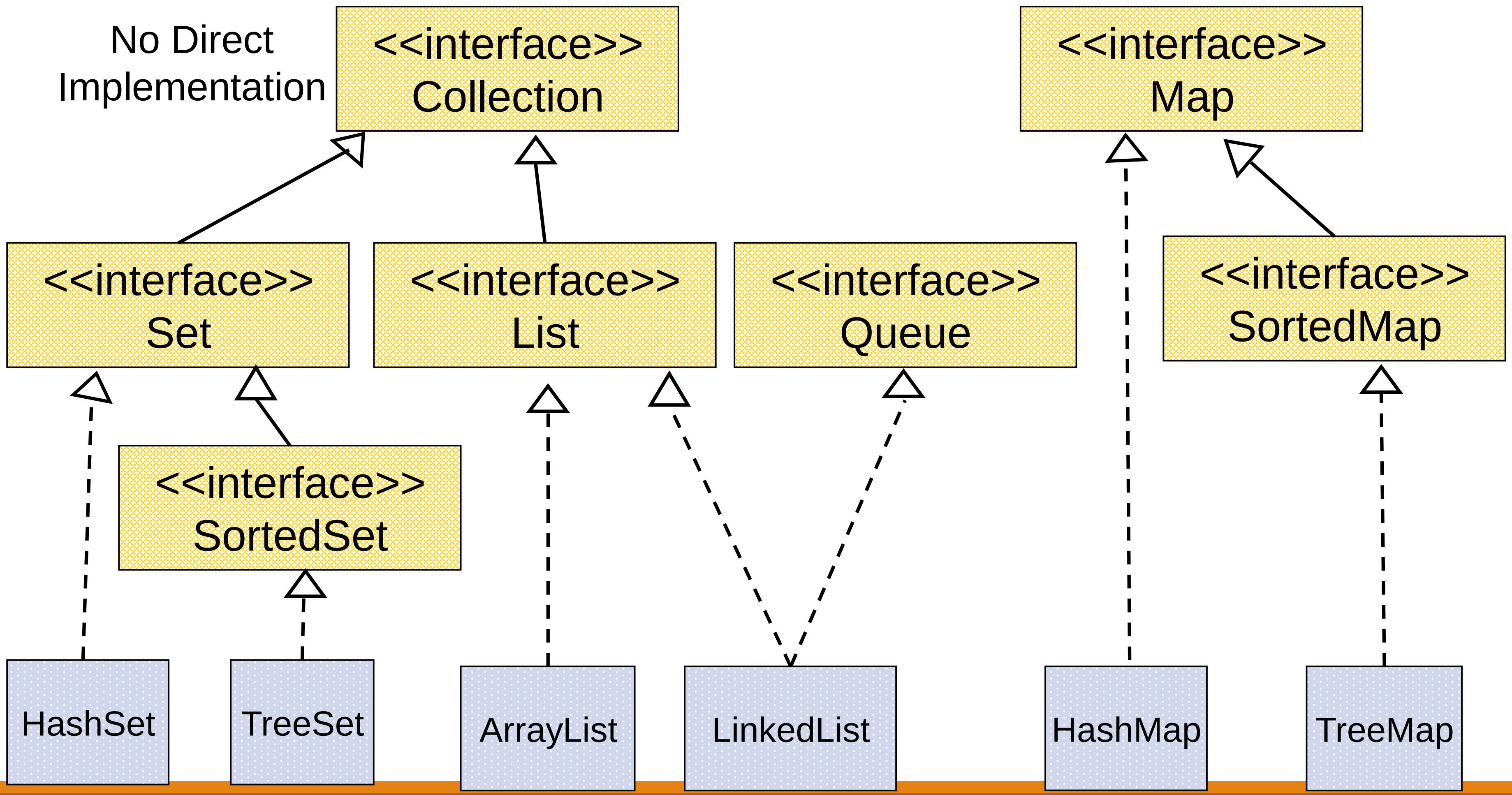
Map duomenų tipai

Collection Implementations

Class Name Convention: <Data structure> <Interface>

General Purpose Implementations		Data Structures			
		Hash Table	Resizable Array	Balanced Tree	Linked List
Interfaces	Set	HashSet		TreeSet (SortedSet)	
	Queue		ArrayDeque		LinkedList
	List		ArrayList		LinkedList
	Map	HashMap		TreeMap (SortedMap)	

General Purpose Implementations



Maps

map: an unordered collection that associates a collection of element values with a set of keys so that elements they can be found very quickly

Each key can appear at most once (no duplicate keys)

A key maps to at most one value

the main operations:

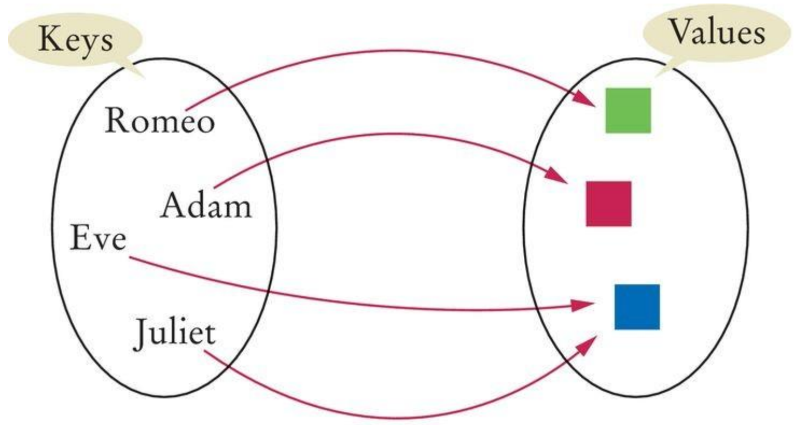
put(key, value) "Map this key to that value."

get(key) "What value, if any, does this key map to?"

Maps

A map allows you to associate elements from a key set with elements from a value collection.

Use a map when you want to look up objects by using a key.



Two implementations of the Map interface:

HashMap

TreeMap

Maps implementation

Store the reference to the map object in a Map reference:

```
Map<String, Color> favoriteColors = new HashMap<>();


Map<String, Color> favoriteColors = new TreeMap<>();
```

Maps

Use the put method to add an association:

```
favoriteColors.put("Juliet", Color.RED);
```

You can change the value of an existing association by calling put again:

```
favoriteColors.put("Juliet", Color.BLUE);
```

Maps

The get method returns the value associated with a key:

```
Color julietsFavoriteColor = favoriteColors.get("Juliet");
```

If you ask for a key that isn't associated with any values, the get method returns null.

To remove an association, call the remove method with the key:

```
favoriteColors.remove("Juliet");
```

Dažniau naudojami Map metodai

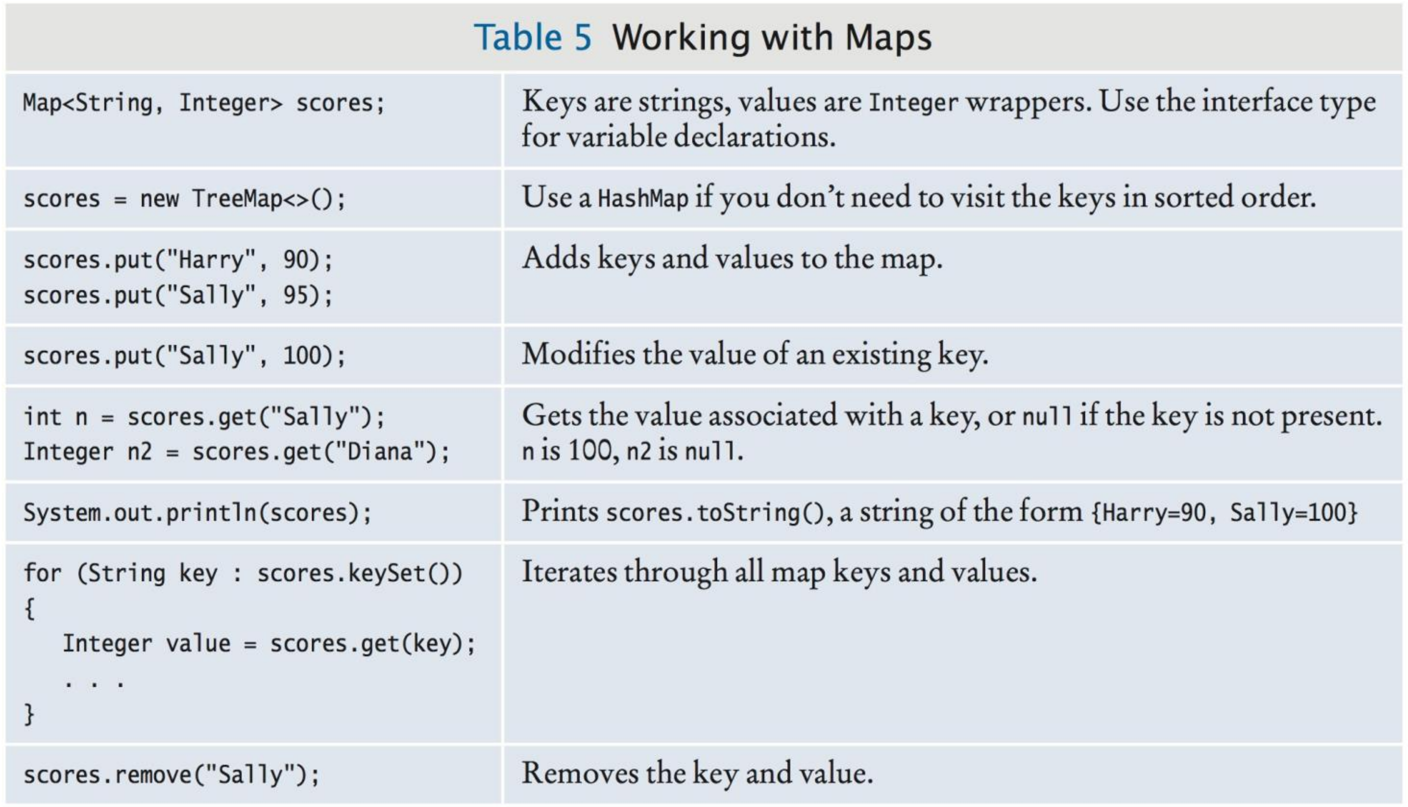


Table 5 Working with Maps

`Map<String, Integer> scores;`	Keys are strings, values are `Integer` wrappers. Use the interface type for variable declarations.
`scores = new TreeMap<>();`	Use a `HashMap` if you don't need to visit the keys in sorted order.

`scores.put("Harry", 90);`
`scores.put("Sally", 95);`
	Adds keys and values to the map.

`scores.put("Sally", 100);`	Modifies the value of an existing key.

`int n = scores.get("Sally");`
`Integer n2 = scores.get("Diana");`
	Gets the value associated with a key, or `null` if the key is not present. `n` is 100, `n2` is `null`.

`System.out.println(scores);`	Prints `scores.toString()`, a string of the form `{Harry=90, Sally=100}`

`for (String key : scores.keySet())`
`{`
`   Integer value = scores.get(key);`
`   . . .`
`}`
	Iterates through all map keys and values.

`scores.remove("Sally");`	Removes the key and value.

Iterators and Maps

Map interface has no iterator method; you can’t get an Iterator directly

must first call either

keySet()	returns a Set of all the keys in this Map
values()	returns a Collection of all the values in this Map

then call iterator() on the key set or values

Examples:

```
Iterator keyItr = grades.keySet().iterator();
Iterator elementItr = grades.values().iterator();
```

If you really want the keys or element values in a more familiar collection such as an ArrayList, use the ArrayList constructor that takes a Collection as its argument

```
ArrayList elements = new ArrayList(grades.values());
```

Iterating Over the Keys of a Map

```
Map<String,String> map = new
HashMap<String,String> ();
map.put("Dan", "03-9516743");
map.put("Rita", "09-5076452");
map.put("Leo", "08-5530098");
map.put("Rita", "06-8201124");

Iterator<String> iter= map.keySet().iterator();
while (iter.hasNext() ) {
		System.out.println(iter.next());
}
```

Output:

```
Leo
Dan
Rita
```

Iterating over a map

```
Map<String,String> map = new HashMap<String,String>();
map.put("Dan", "03-9516743");
map.put("Rita", "09-5076452");
map.put("Leo", "08-5530098");
map.put("Rita", "06-8201124");

Iterator<Map.Entry<String,String>> iter=
map.entrySet().iterator();
for (iter.hasNext(); ) {
  Map.Entry<String,String> entry = iter.next();
  System.out.println(entry.getKey() + ": " + entry.getValue());
}
```

Output:

```
Leo: 08-5530098
Dan: 03-9516743
Rita: 06-8201124
```

Iterating over a map

You can also ask the key set for an iterator and get all keys. For each key, you can find the associated value with the get method.

To print all key/value pairs in a map m:

```
Set<String> keySet = m.keySet();
for (String key : keySet)
{
        Color value = m.get(key);
        System.out.println(key + "->" + value);
}
```

Collection Algorithms

Defined in the Collections class

Main algorithms:

- sort
- binarySearch
- reverse
- shuffle
- min
- max

Sorting

```
import java.util.*;

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

```
Arguments:  A C D B
Output:     [A, B, C, D]
```

Best Practice <with generics>

Specify an element type only when a collection is instantiated:

- `Set<String> s = new HashSet<String>();`

Interface

Implementation

Works, but…

- `public void foo(HashSet<String> s){…}`
- `public void foo(Set<String> s) {…}`
- `s.add()` invokes `HashSet.add()`

Better!

polymorphism

Compound collections

Collections can be nested to represent more complex data

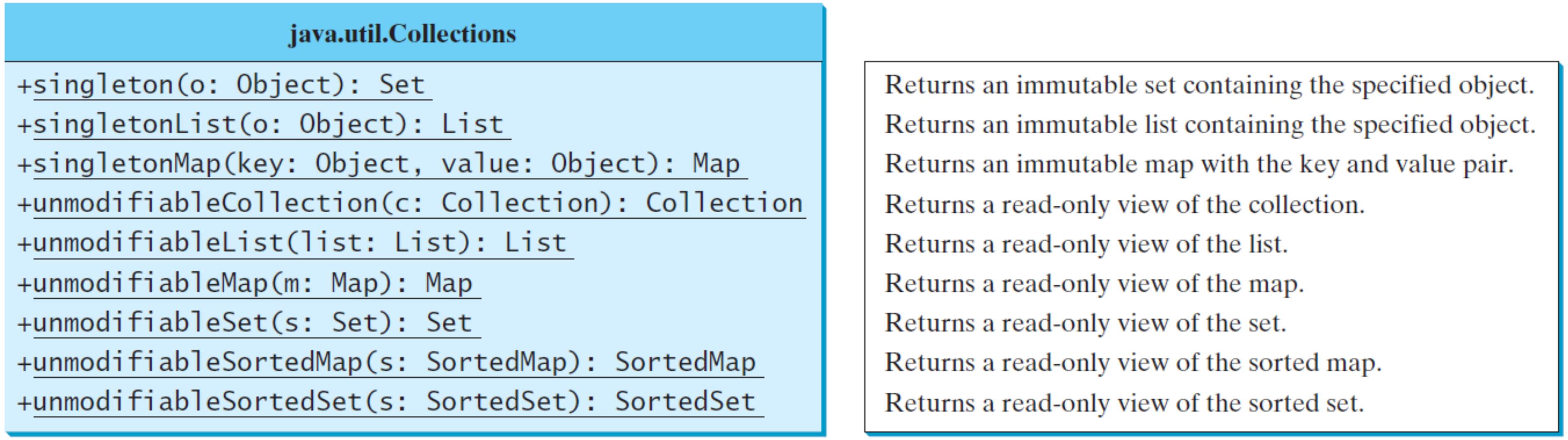
example: A person can have one or many phone numbers

want to be able to quickly find all of a person's phone numbers, given their name

We could implement this example as a HashMap of Lists

- keys are Strings (names)
- values are Lists (e.g ArrayList) of Strings, where each String is one phone number

The Singleton and Unmodiable Collections



java.util.Collections	
+singleton(o: Object): Set	Returns an immutable set containing the specified object.
+singletonList(o: Object): List	Returns an immutable list containing the specified object.
+singletonMap(key: Object, value: Object): Map	Returns an immutable map with the key and value pair.
+unmodifiableCollection(c: Collection): Collection	Returns a read-only view of the collection.
+unmodifiableList(list: List): List	Returns a read-only view of the list.
+unmodifiableMap(m: Map): Map	Returns a read-only view of the map.
+unmodifiableSet(s: Set): Set	Returns a read-only view of the set.
+unmodifiableSortedMap(s: SortedMap): SortedMap	Returns a read-only view of the sorted map.
+unmodifiableSortedSet(s: SortedSet): SortedSet	Returns a read-only view of the sorted set.