НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Object-Oriented Programming in the Java language

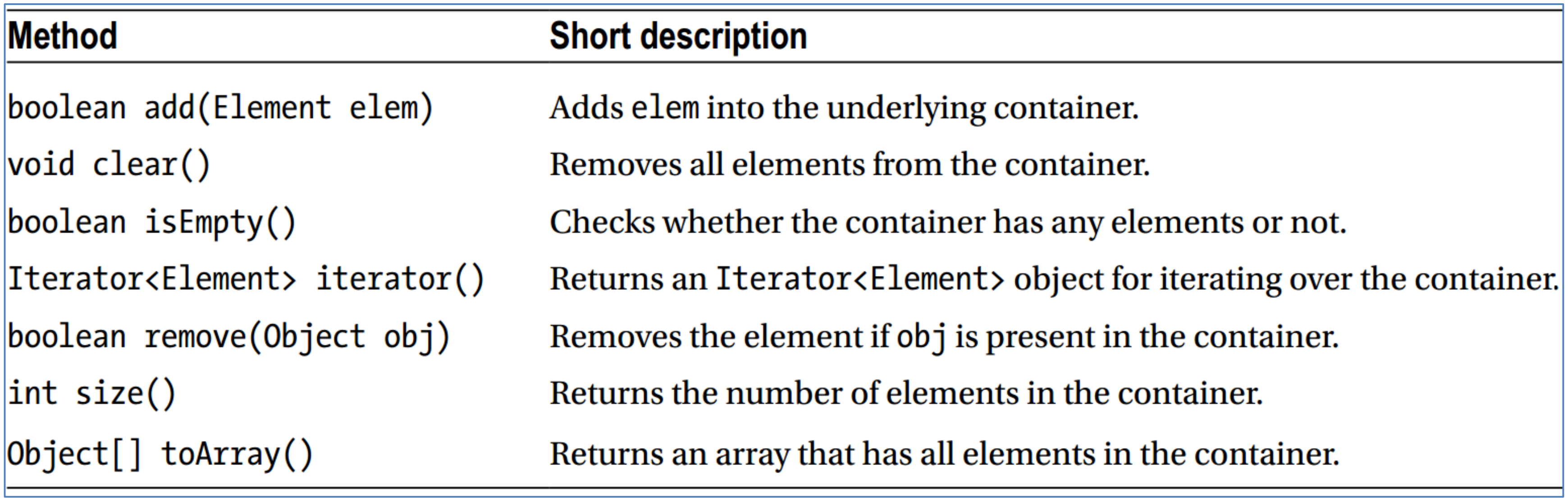
Part 7. Collections(2/2)

Yevhen Berkunskyi, NUoS
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Collection interface

The Collection interface provides methods such as **add()** and **remove()** that are common to all containers:

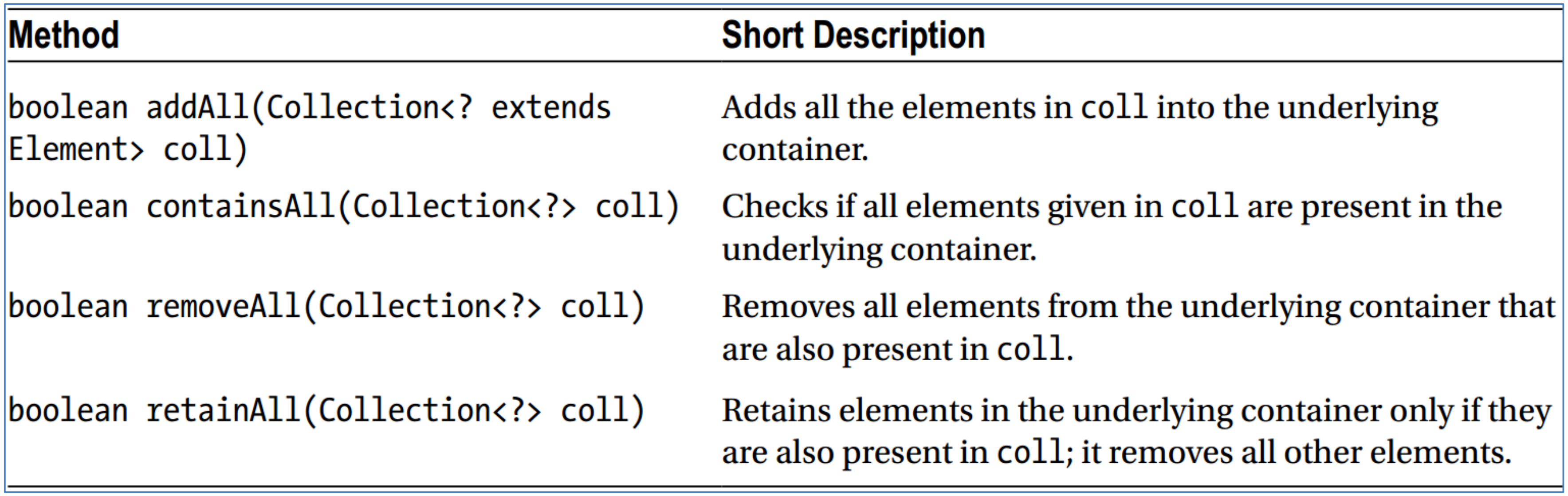


Method	Short description
`boolean add(Element elem)`	Adds `elem` into the underlying container.
`void clear()`	Removes all elements from the container.
`boolean isEmpty()`	Checks whether the container has any elements or not.
`Iterator<Element> iterator()`	Returns an `Iterator<Element>` object for iterating over the container.
`boolean remove(Object obj)`	Removes the element if `obj` is present in the container.
`int size()`	Returns the number of elements in the container.
`Object[] toArray()`	Returns an array that has all elements in the container.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Collection interface

Methods in the Collection Interface that apply to multiple elements:

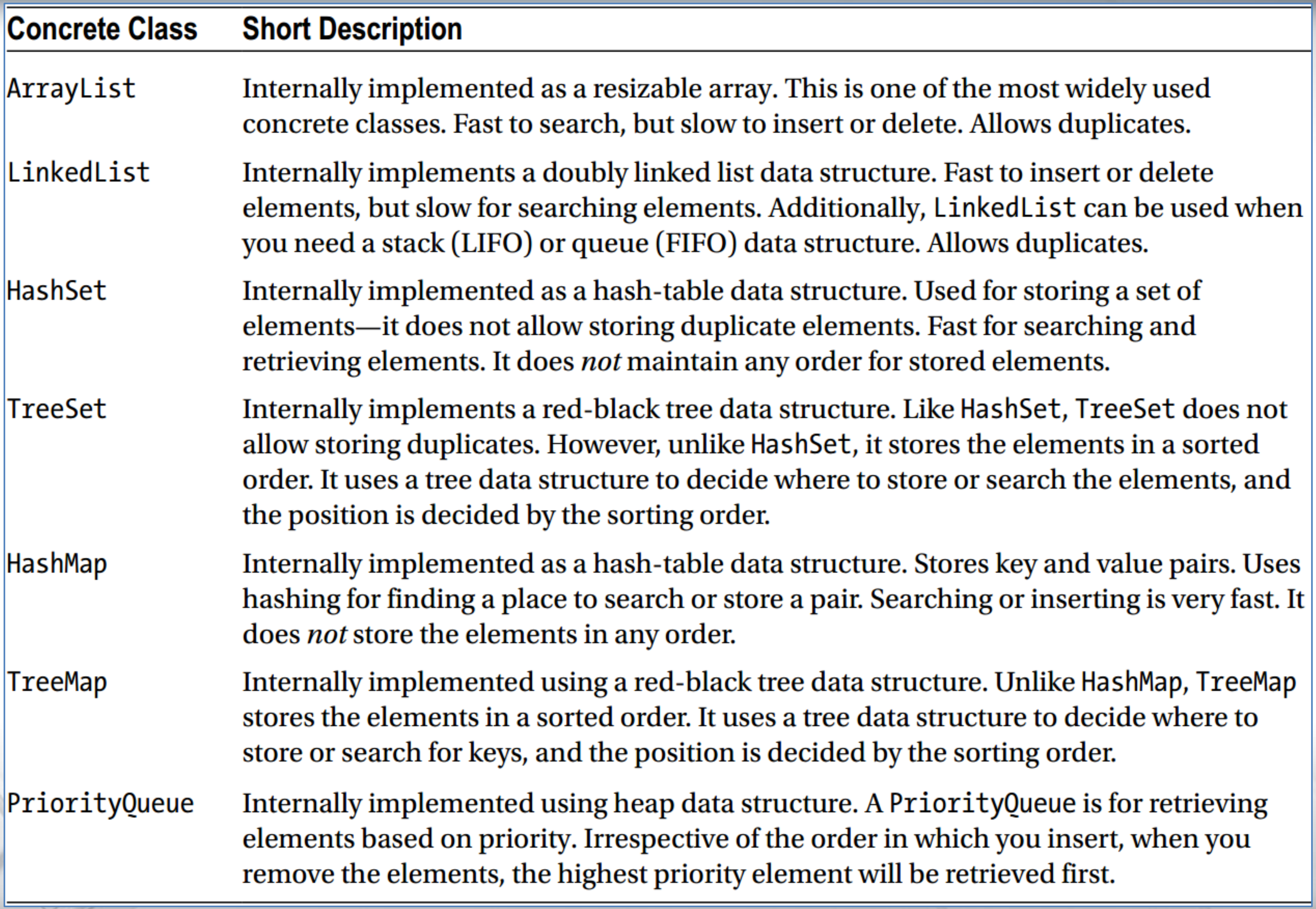


Method	Short Description
`boolean addAll(Collection<? extends Element> coll)`	Adds all the elements in `coll` into the underlying container.
`boolean containsAll(Collection<?> coll)`	Checks if all elements given in `coll` are present in the underlying container.
`boolean removeAll(Collection<?> coll)`	Removes all elements from the underlying container that are also present in `coll`.
`boolean retainAll(Collection<?> coll)`	Retains elements in the underlying container only if they are also present in `coll`; it removes all other elements.

Concrete Classes

- Numerous interfaces and abstract classes in the Collection hierarchy provide the common methods that specific concrete classes implement/extend.
- The concrete classes provide the actual functionality

Important Concrete Classes in Collection Framework



Concrete Class	Short Description
`ArrayList`	Internally implemented as a resizable array. This is one of the most widely used concrete classes. Fast to search, but slow to insert or delete. Allows duplicates.
`LinkedList`	Internally implements a doubly linked list data structure. Fast to insert or delete elements, but slow for searching elements. Additionally, `LinkedList` can be used when you need a stack (LIFO) or queue (FIFO) data structure. Allows duplicates.
`HashSet`	Internally implemented as a hash-table data structure. Used for storing a set of elements—it does not allow storing duplicate elements. Fast for searching and retrieving elements. It does *not* maintain any order for stored elements.
`TreeSet`	Internally implements a red-black tree data structure. Like `HashSet`, `TreeSet` does not allow storing duplicates. However, unlike `HashSet`, it stores the elements in a sorted order. It uses a tree data structure to decide where to store or search the elements, and the position is decided by the sorting order.
`HashMap`	Internally implemented as a hash-table data structure. Stores key and value pairs. Uses hashing for finding a place to search or store a pair. Searching or inserting is very fast. It does *not* store the elements in any order.
`TreeMap`	Internally implemented using a red-black tree data structure. Unlike `HashMap`, `TreeMap` stores the elements in a sorted order. It uses a tree data structure to decide where to store or search for keys, and the position is decided by the sorting order.
`PriorityQueue`	Internally implemented using heap data structure. A `PriorityQueue` is for retrieving elements based on priority. Irrespective of the order in which you insert, when you remove the elements, the highest priority element will be retrieved first.

List Classes

ArrayList implements a resizable array.

- When you create a native array (say, new String[10];), the size of the array is known (fixed) at the time of creation.

However, ArrayList is a dynamic array:

- it can grow in size as required. Internally, an ArrayList allocates a block of memory and grows it as required.
- So, accessing array elements is very fast in an ArrayList.

LinkedList implements interface List with data structure of linked list.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Set Classes

- Set, as we studied, contains no duplicates.
- Unlike List, a Set doesn’t remember where you inserted the element (i.e., it doesn’t remember the insertion order).

There are two important concrete classes for Set:

- HashSet and TreeSet.
- A HashSet is for quickly inserting and retrieving elements; it does *not* maintain any sorting order for the elements it holds.
- A TreeSet stores the elements in a sorted order (and it implements the SortedSet interface).

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Comparable and Comparator Interfaces

- Comparable and Comparator interfaces are used to compare similar objects (for example, while performing searching or sorting).
- Assume that you have a container containing a list of Person object.
- Now, how do you compare two Person objects?

Comparable

The Comparable interface has only one method compareTo() , which is declared as follows:

```
    int compareTo(Element that)
```

Since you are implementing the compareTo() method in a class, you have this reference available. You can compare the current element with the passed Element and return an int value.

What should the int value be? Well, here are the rules for returning the integer value:

```
    return 1 if current object > passed object
    return 0 if current object == passed object
    return -1 if current object < passed object
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Natural Ordering

But, what does >, < or == mean for an Element? It is left to you to decide how to compare two objects!

But the meaning of comparison should be a natural one; in other words, the comparison should mean *natural ordering*.

Integers are comparing with each other, based on a *numeric order*

Strings are comparing with each other, based on a *lexicographic comparison*

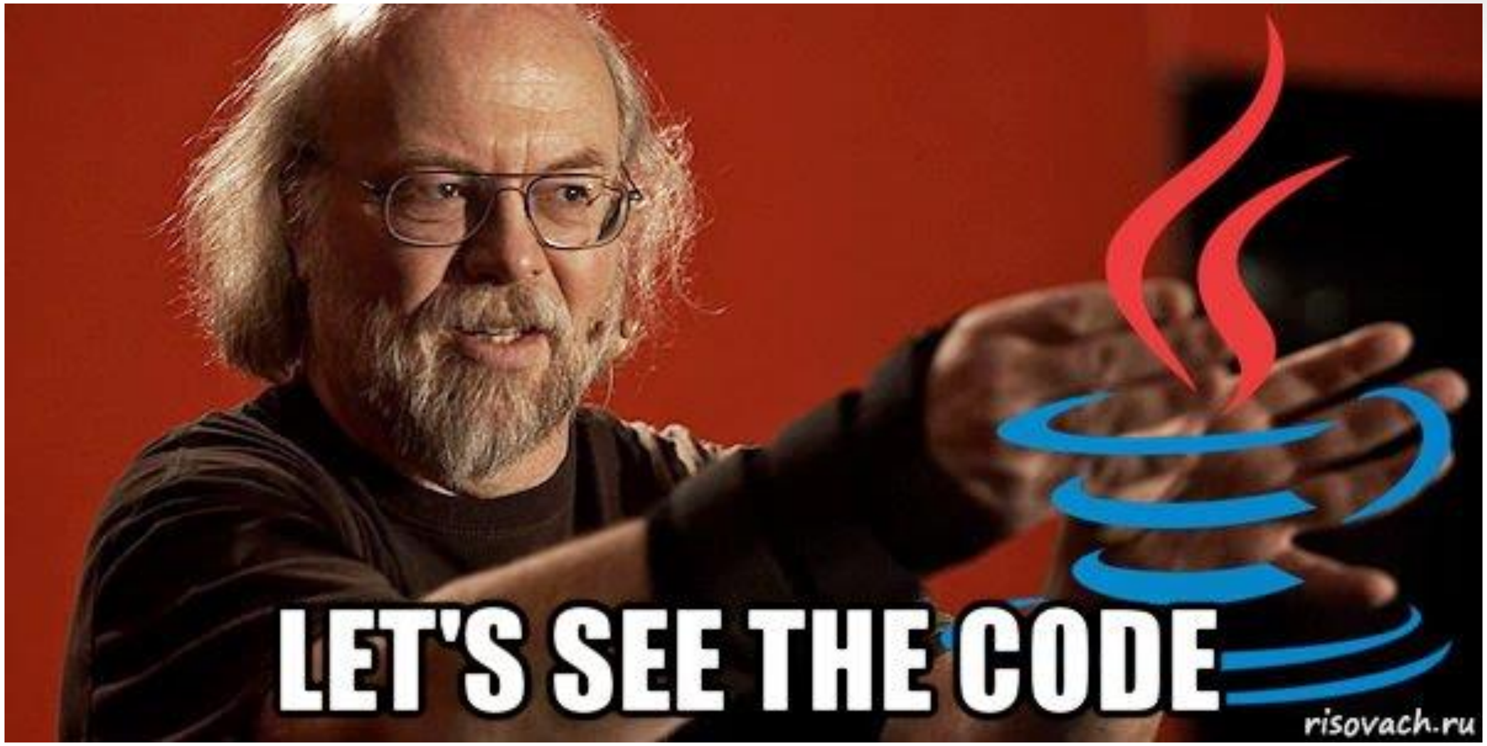
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Comparator

- If you need to implement two or more alternative ways to compare two similar objects, then you may implement the Comparator interface.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Example



The Map Interface

- A Map stores key and value pairs.
- The Map interface does *not* extend the Collection interface.
- However, there are methods in the Map interface that you can use to get the objects classes that implement the Collection interface to work around this problem.
- Also, the method names in Map are very similar to the methods in Collection, so it is easy to understand and use Map.

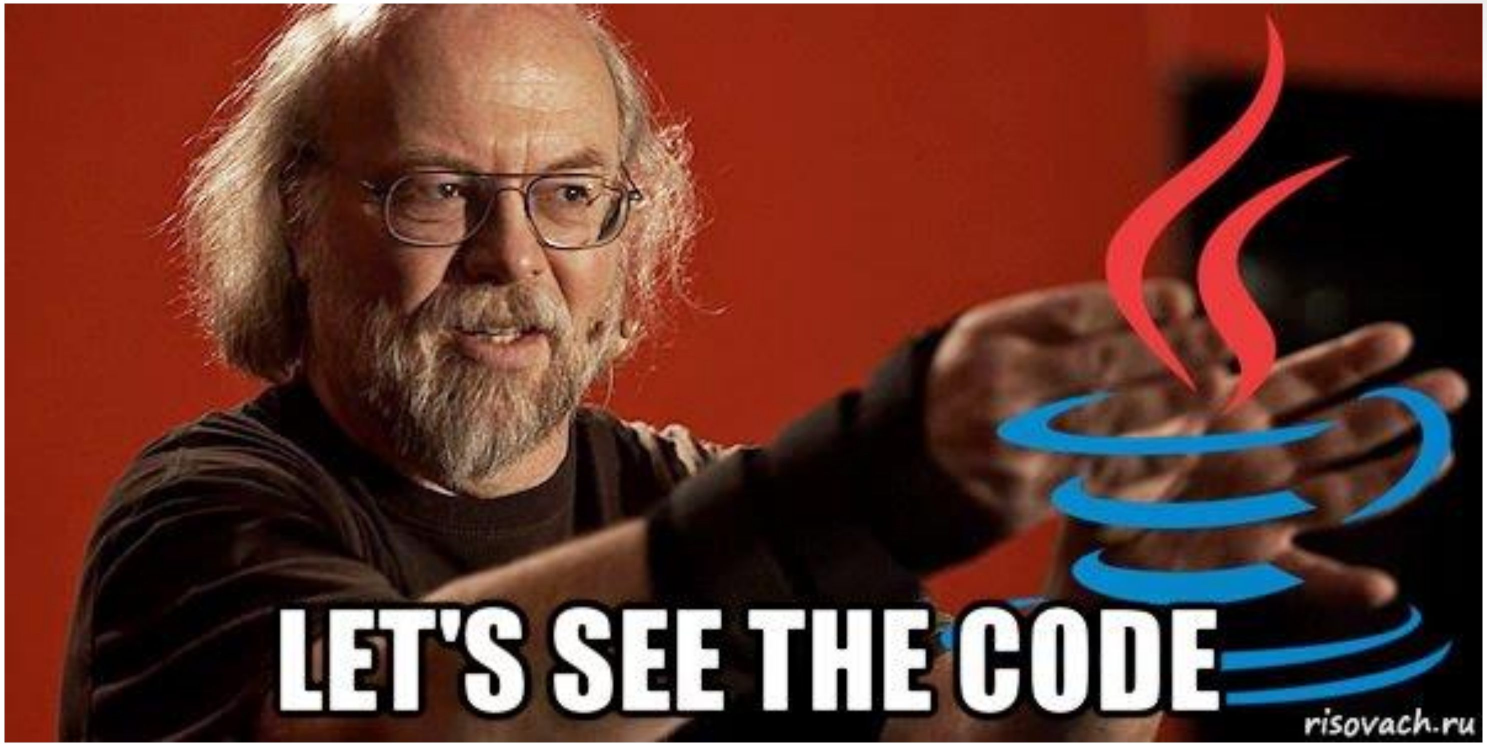
Maps

There are two important concrete classes of Map: HashMap and TreeMap.

- A HashMap uses a hash table data structure internally. In HashMap, searching (or looking up elements) is a fast operation. However, HashMap neither remembers the order in which you inserted elements nor keeps elements in any sorted order.
- A TreeMap uses a red-black tree data structure internally. Unlike HashMap, TreeMap keeps the elements in sorted order (i.e., sorted by its keys). So, searching or inserting is somewhat slower than the HashMap.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Example



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Questions?

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Object-Oriented Programming in the Java language

Part 7. Collections(2/2)

Yevhen Berkunskyi, NUoS
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua