JAVA PROGRAMMING BASICS

Module 2: Java Object-oriented Programming

Training program

1. Classes and Instances
2. The Methods
3. The Constructors
4. Static Elements
5. Initialization sections
6. Package
7. Inheritance and Polymorphism
8. Abstract classes and Interfaces
9. String processing
10. Wrapper classes for primitive types
11. Exceptions and Assertions
12. Nested classes
13. Enums
14. Generics
15. **Collections**
16. Method overload resolution
17. Multithreads
18. Core Java classes
19. Object Oriented Design
20. Functional Programming

Module contents

- Collections
  - Java Collection Framework, Interfaces
  - The Collection Interface
  - The Iterators
  - The List Interface
  - The Queue and Deque Interfaces
  - The Set Interface
  - The Map Interface
  - The Collection Class

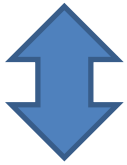
Module contents

- Collections
  - Java Collection Framework, Interfaces
  - The Collection Interface
  - The Iterators
  - The List Interface
  - The Queue and Deque Interfaces
  - The Set Interface
  - The Map Interface
  - The Collection Class

Java Collection Framework

We group instances of some class into a Collection
to get a convenient way to handle them during iteration.

Collections



Data Structures

Collections util methods

Algorithms

JDK	Features
1.0	Vector, HashTable
1.2	Collection Framework
5	Generics
6	Deque, NavigableSet, NavigableMap
8	Functional programming
9	Static methods for collection creating

Java Collection framework. Interfaces 1/3

- A collection, as its name implied, is simply an object that holds a collection (or a group, a container) of objects.
- Each item in a collection is called an element.
- A framework, by definition, is a set of interfaces that force you to adopt some design practices. A well-designed framework can improve your productivity and provide ease of maintenance.
- It also contains a set of classes that implements such interfaces
- It also contains classes with methods utilizing common algorithms (sort, search etc.)

Java Collection Framework Interfaces 2/3

There are four main interfaces in the Java Collections Framework:

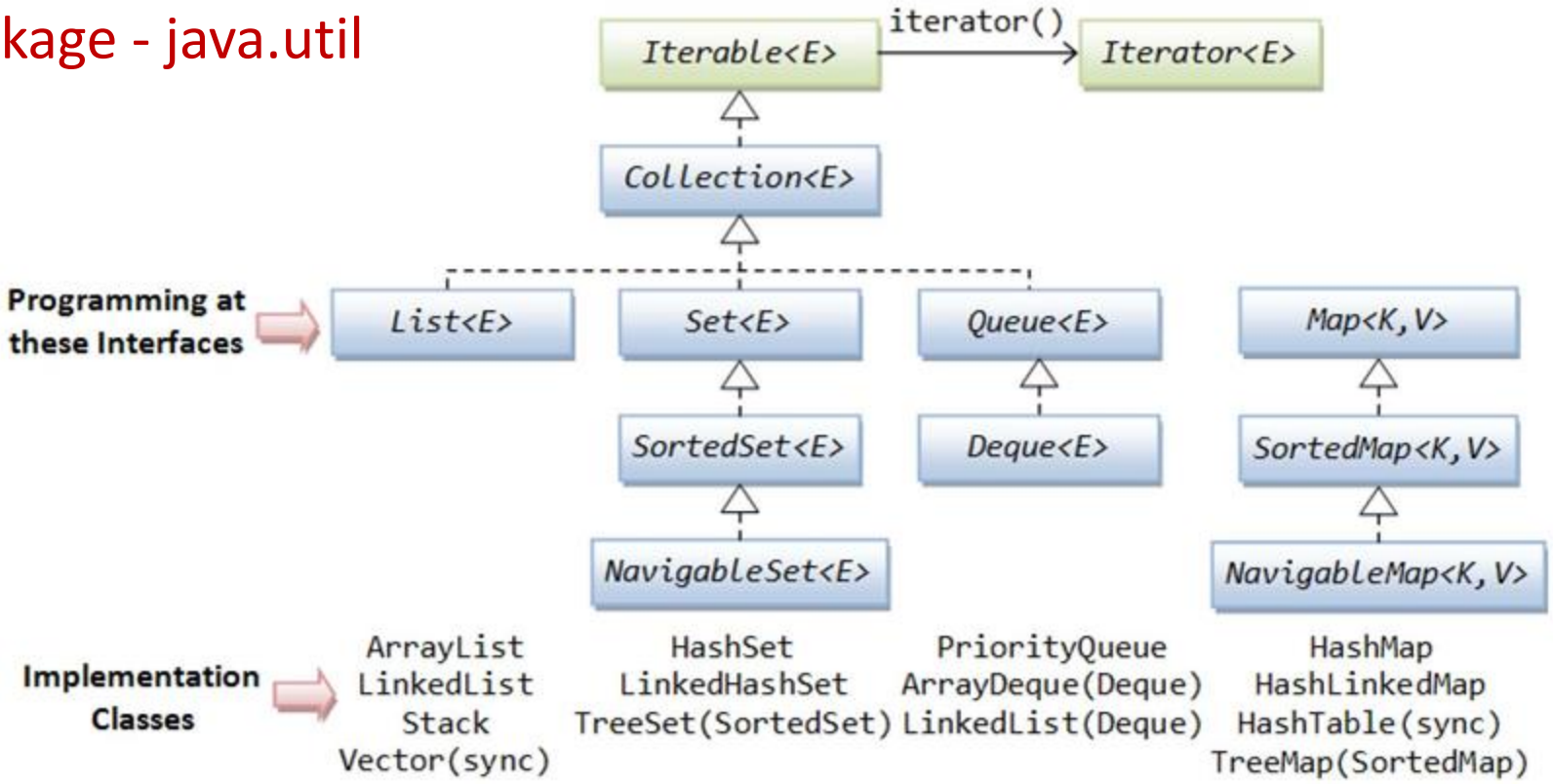
- **List:** A list is an ordered collection of elements that allows duplicate entries. Elements in a list can be accessed by an int index.
- **Set:** A set is a collection that does not allow duplicate entries.
- **Queue:** A queue is a collection that orders its elements in a specific order for processing. A Deque is a subinterface of Queue that allows access at both ends.
- **Map:** A map is a collection that maps keys to values, with no duplicate keys allowed. The elements in a map are key/value pairs.

since Java 1.2

Java Collection framework. Interfaces 3/3

- Collections

package - java.util



Module contents

- Collections
  - Java Collection Framework, Interfaces
  - The Collection Interface
  - The Iterators
  - The List Interface
  - The Queue and Deque Interfaces
  - The Set Interface
  - The Map Interface
  - The Collection Class

The Collection interface 1/5

- The **Collection** interface is used to pass around collections of objects where maximum generality is desired
- No DIRECT implementations
- Subinerfaces List<E>, Set<E>, Queue<E> are implemented

Collection
+add(element : Object) : boolean +addAll(collection : Collection) : boolean +clear() : void +contains (element : Object) : boolean +containsAll(collection : Collection) : boolean +equals(object : Object) : boolean +hashCode() : int +iterator() : Iterator +remove(element : Object) : boolean +removeAll(collection : Collection) : boolean +retainAll(collection : Collection) : boolean +size() : int +toArray() : Object[] +toArray(array : Object[]) : Object[]

The Collection interface 1/5

- The **Collection** interface is used to pass around collections of objects where maximum generality is desired

Collection
isEmpty() : boolean
parallelStream() : Stream<E>
removeIf(Predicate<? super E> filter) : boolean
spliterator() : Spliterator<E>
stream() : Stream<E>

The Collection interface 2/5

- *Basic operations* perform on an entire collection

```
1. Collection<String> myColl = new ArrayList<>();
2. myColl.add("aaa");
3. myColl.add("bbbb");
4. myColl.add("ccccc");
5. System.out.println(myColl);
6. int sizeOfColl = myColl.size();
7. System.out.println(sizeOfColl);
8. myColl.clear();
9. System.out.println(myColl);
10. myColl.add(null);
```

myCol has only Collection's methods, but not ArrayList!

Console output

```
[aaa, bbbb, ccccc]
3
[]
[null]
```

The Collection interface 3/5

```
1.  Collection<String> myColl =
2.        new ArrayList<>();
3.  myColl.add("aaa");
4.  myColl.add("bbbb");
5.  myColl.add("ccccc");
6.  System.out.println(myColl);
7.  System.out.println(myColl.remove("bbbb"));   //myColl changed
8.  System.out.println(myColl.remove("abcd"));//myColl unchanged
9.  System.out.println(myColl);
10. System.out.println(myColl.contains("aaa"));
11. System.out.println(myColl.contains("bbbb"));
```

Console output

```
[aaa, bbbb, ccccc]
true
false
[aaa, ccccc]
true
false
```

See more examples in Edu Project - CollectionBasicOpsRunner...

The Collection interface 4/5

- *Bulk operations* perform on an entire Collection

```
1. Collection<String> myColl1 = new ArrayList<>();
2. Collection<String> myColl2 = new ArrayList<>();
3. myColl1.add("aaa");  myColl1.add("bb");
4. myColl1.add("c");
5. myColl2.add("aaa");  myColl2.add("bb");
6. System.out.println(myColl1.containsAll(myColl2));
7. System.out.println(myColl1.removeAll(myColl2));    //myColl1
8. System.out.println(myColl1);                       //changed
```

Console output

```
true
true
[c]
```

See more examples in Edu Project - CollectionBulkOpsRunner...

The Collection interface 5/5

- Collection Interface Array Operations

```
1. Collection<String> myColl1 = new ArrayList<>();
2. myColl1.add("aaa"); myColl1.add("bb"); myColl1.add("c");
3. Object[] myArrObj = myColl1.toArray();
4. String[] myArrStr = myColl1.toArray(new String[3]);
5. System.out.println(Arrays.toString(myArrObj));
6. System.out.println(Arrays.toString(myArrStr));
```

Console output

```
[aaa, bb, c]
[aaa, bb, c]
```

See more examples in Edu Project - CollectionArrayOpsRunner...

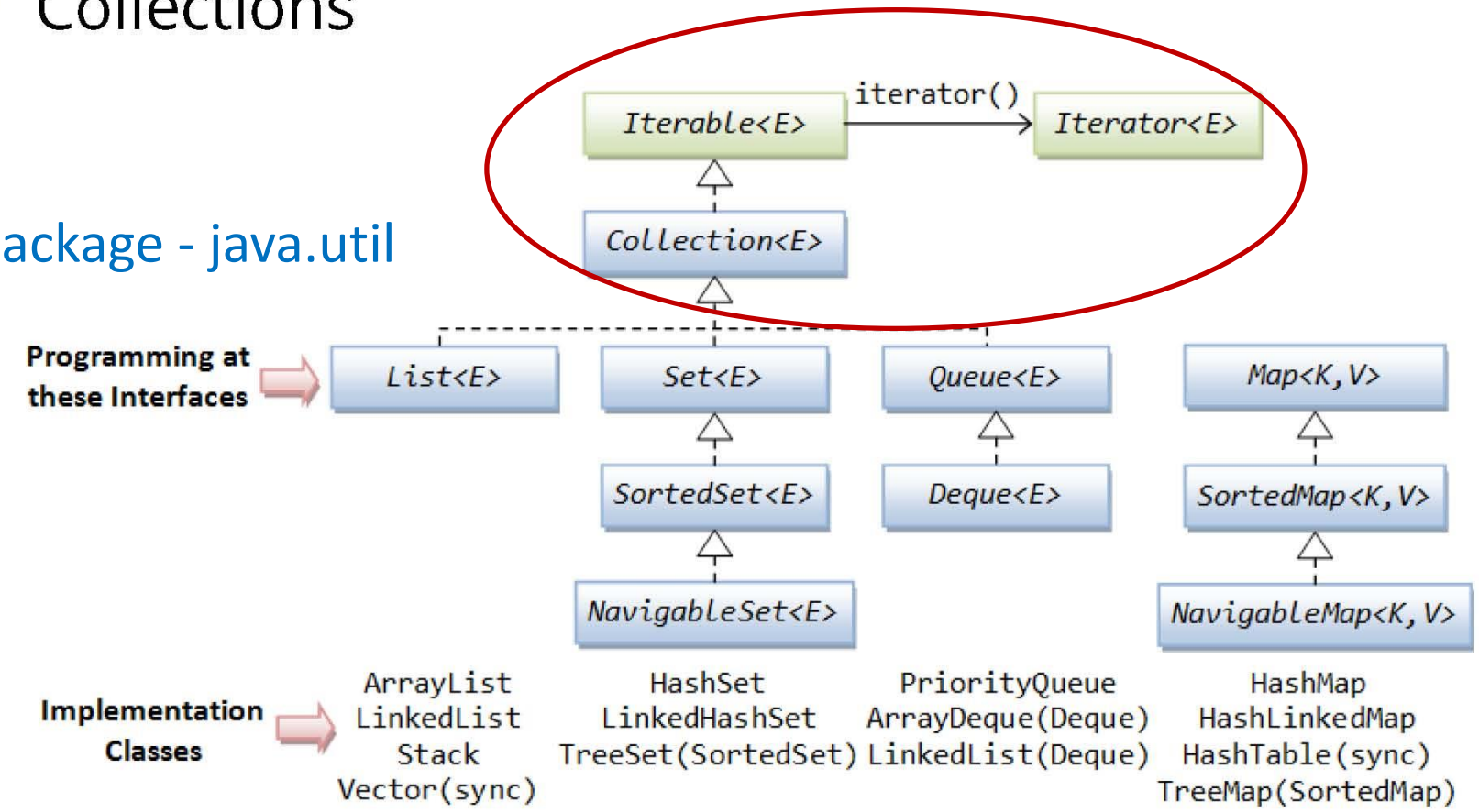
Module contents

- Collections
  - Java Collection Framework, Interfaces
  - The Collection Interface
  - The Iterators
  - The List Interface
  - The Queue and Deque Interfaces
  - The Set Interface
  - The Map Interface
  - The Collection Class

Java Collection framework. Interfaces 2/3

- Collections

package - java.util



public interface java.lang.Iterable<T>

public interface Collection<E> extends Iterable<E> { .. }

Modifier and Type	Method	Description
default void	forEach(Consumer <? super T> action)	Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.
Iterator<T>	iterator()	Returns an iterator over elements of type T.
default Spliterator<T>	spliterator()	Creates a Spliterator over the elements described by this Iterable.

Since Java 1.5

public interface java.util.Iterator<E>

Modifier and Type	Method	Description
default void	forEachRemaining(Consumer<? super E> action)	Performs the given action for each remaining element until all elements have been processed or the action throws an exception.
boolean	hasNext()	Returns true if the iteration has more elements.
E	next()	Returns the next element in the iteration.
default void	remove()	Removes from the underlying collection the last element returned by this iterator (optional operation).

Since Java 1.2

The Iterators 2/4

- By using this iterator object, you can access each element in the collection

```
1. Collection<String> myCol = new ArrayList<>();
2. myCol.add("aa"); myCol.add("bbb");
3. myCol.add("cccc"); myCol.add("ddddd");
4. Iterator<String> itr = myCol.iterator();
5. while(itr.hasNext()) {
6.     String element = itr.next();
7.     System.out.print(element + " ");
8. }
```

The Iterators

```
Collection<String> myColl = new ArrayList<>
                    (Arrays.asList("aa", "bbb", "cccc", "ddddd"));
iterator = myColl.iterator();
System.out.println(iterator.next());    //aa
System.out.println(iterator.next());    //bbb
System.out.println(iterator.next());    //cccc
System.out.println(iterator.next());    //ddddd
/*java.util.NoSuchElementException*/
System.out.println(iterator.next());
```

It need to re-get the iterator for the collection

The Iterators 3/4

```
1. Collection<String> myCol = new ArrayList<>();
2. myCol.add("aa"); myCol.add("bbb");
3. myCol.add("cccc"); myCol.add("ddddd");

4. for (String str : myCol) {
5.     myCol.remove(str);
6. }
```

no implicit iterator obtaining

Exception in thread "main"
java.util.ConcurrentModificationException

The Iterators 4/4

- Iterator.remove is the only safe way to modify a collection during iteration

```
1. Collection<String> myCol = new ArrayList<>();
2. myCol.add("aa"); myCol.add("bbb");
3. myCol.add("cccc"); myCol.add("ddddd");
4. Iterator<String> itr = myCol.iterator();
5. while (itr.hasNext()) {
6.     itr.next();
7.     itr.remove();
8. }
9. System.out.print(myCol);
```

```
while (itr.hasNext()) {
    itr.remove();
}
```

IllegalStateException

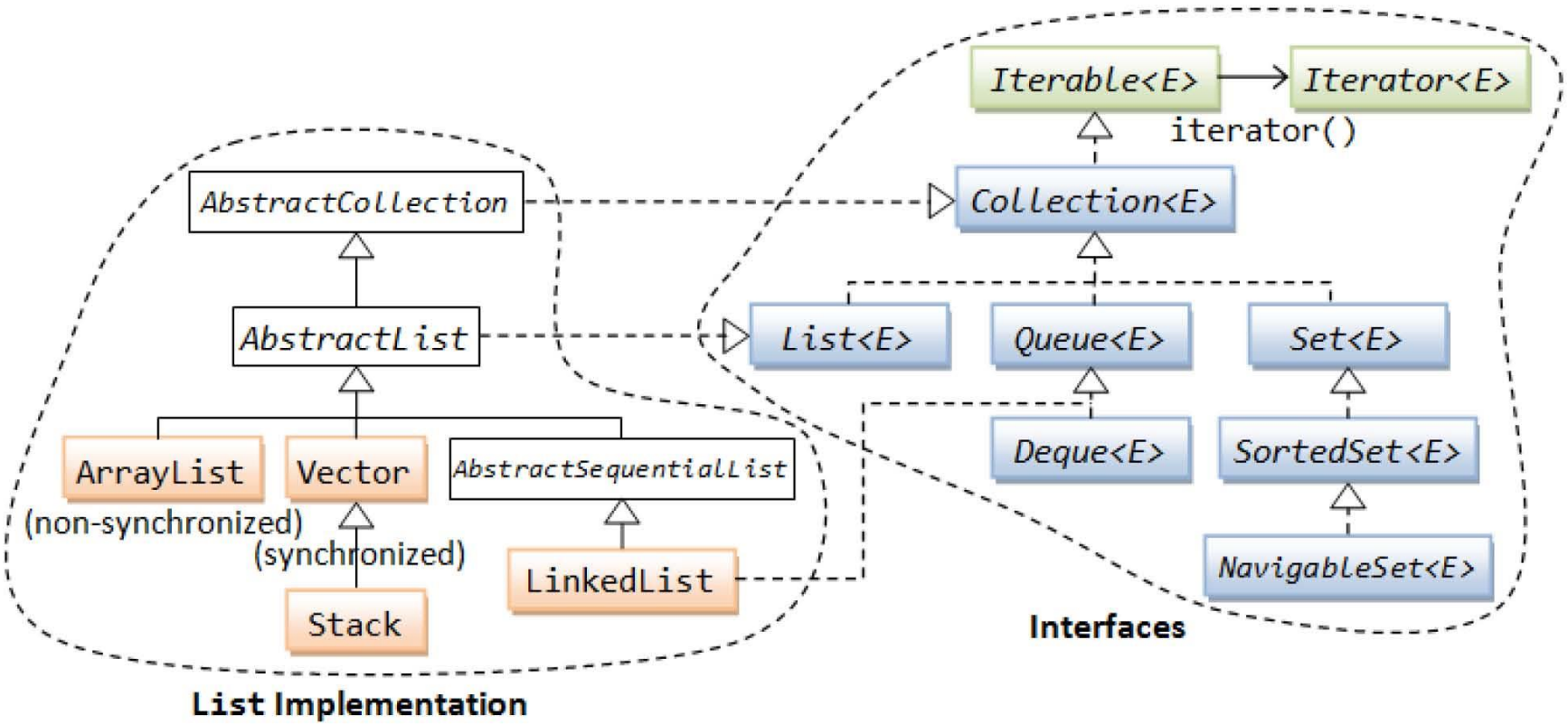
See more examples in Edu Project - IteratorRunner...

Module contents

- Collections
  - Java Collection Framework, Interfaces
  - The Collection Interface
  - The Iterators
  - The List Interface
  - The Queue and Deque Interfaces
  - The Set Interface
  - The Map Interface
  - The Collection Class

The List interface is a collection of ordered elements (for example, by the order of adding to the collection), which are accessed by index (positions in the collection).

Java Collection framework. Interfaces 3/3



List implements a mathematical abstraction of the list, can store the same elements

The List interface 1/12

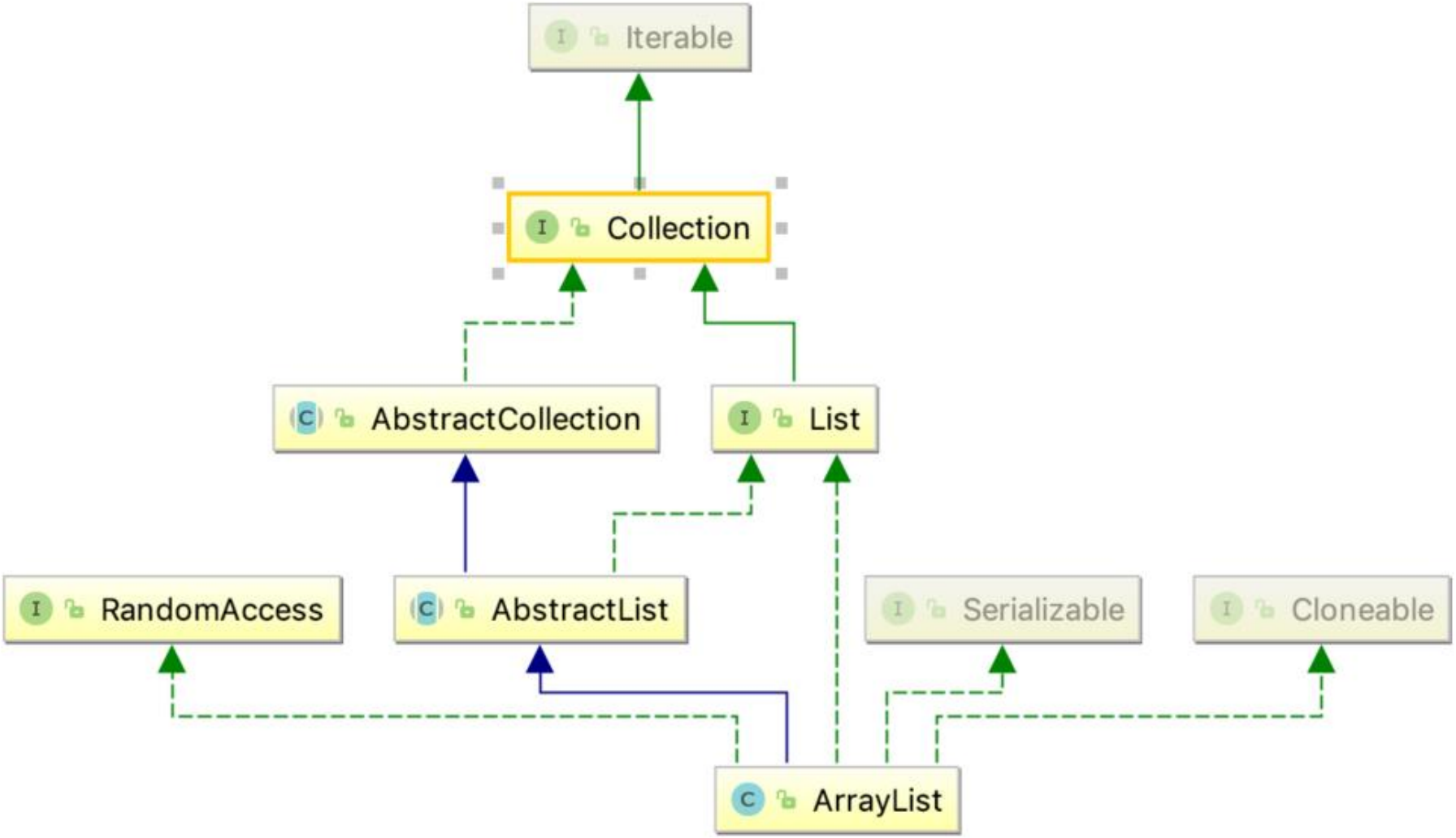
- The List Interface

+add(element : Object) : boolean
+add(index : int, element : Object) : void ✓
+addAll(collection : Collection) : boolean
+addAll(index : int, collection : Collection) : boolean ✓
+clear() : void
+contains (element : Object) : boolean
+containsAll(collection : Collection) : boolean
+equals(object : Object) : boolean
+get(index : int) : Object ✓
+hashCode() : int
+indexOf(element : Object) : int ✓
+iterator() : Iterator

✓ - added in List interface

+lastIndexOf(element : Object) : int ✓
+listIterator() : ListIterator ✓
+listIterator(startIndex : Int) : ListIterator ✓
+remove(element : Object) : boolean
+remove(index : int) : Object ✓
+removeAll(collection : Collection) : boolean
+retainAll(collection : Collection) : boolean
+set(index : int, element : Object) : Object ✓
+size() : int
+subList(fromIndex : int, toIndex : int) : List ✓
+toArray() : Object[]
+toArray(array : Object[]) : Object[]

List implementation as ArrayList




```
public class ArrayList<E> extends AbstractList<E>
        implements List<E>, RandomAccess, Cloneable, java.io.Serializable
```

The List interface 3/12
ArrayList creation

```
1. public static void main(String[] arr){
2.     List<String> lst = new ArrayList<String>();
3.     lst.add("aa");
4.     lst.add("bbb");
5.     lst.add("cccc");
6.     System.out.println(lst);     //[aa, bbb, cccc]
7. }
```

The List interface - ArrayList creation

```
List<String> list1 = Arrays.asList("aaa", "bbbb", "ccccc");
System.out.println(list1.get(0)); //aaa
list1.set(0, "ddd");
list1.add("somestr");   //UnsupportedOperationException
```

Returns a fixed-size list backed by the specified array

```
/*Since Java 9*/
List<String> list2 = List.of("aaa", "bbbb", "ccccc");
System.out.println(list2.get(0)); //aaa
list2.set(0, "ddd");  //UnsupportedOperationException
list2.add("somestr"); //UnsupportedOperationException
```

Returns an unmodifiable list

```
list1 = new ArrayList<>(Arrays.asList("aaa", "bbbb", "ccccc"));
list1.add("somestr");
list2 = new ArrayList<>(List.of("aaa", "bbbb", "ccccc"));
list2.add("somestr");
```

Creating a List with a Factory

Method	Description	Can add elements?	Can replace elements?	Can delete elements?
Arrays.asList(varargs)	Returns **fixed size** list backed by an array	No	Yes	No
List.of(varargs)	Returns **immutable** list	No	No	No
List.copyOf(collection)	Returns **immutable** list with copy of original collection's values	No	No	No

```
List<Integer> fabricList = List.of(1, 2, 3, 4, 5);
List<Integer> forCopyList = new ArrayList<>(fabricList);
forCopyList.add(6);
forCopyList.remove(0);
forCopyList.set(0, 7);
System.out.println(forCopyList);    //[7, 3, 4, 5, 6]
```

Wrapping immutable list while creating new list make new list mutable

The List interface methods

```
List<String> list = new ArrayList<>(List.of("aaa", "bbbb", "ccccc", "fff"))
list.add(1, "a1a1a1");
System.out.println(list);   //[aaa, a1a1a1, bbbb, ccccc, fff]
List<String> anotherList = new ArrayList<>();
anotherList.add(0, "eeee");
list.addAll(4, anotherList);
System.out.println(list);   //[aaa, a1a1a1, bbbb, ccccc, eeee, fff]
String s1 = list.get(3);
System.out.println(s1);  //ccccc
list.remove(1);
list.add("aaa");
System.out.println(list);   //[aaa, bbbb, ccccc, eeee, fff, aaa]
list.remove("aaa");
/*Remove first element*/
System.out.println(list);   //[bbbb, ccccc, eeee, fff, aaa]
...
```

The List interface methods

```
...     //[bbbb, ccccc, eeee, fff, aaa]
list.set(2, "bbbb");
System.out.println(list);   //[bbbb, ccccc, bbbb, fff, aaa]
int pos1 = list.indexOf("bbbb");
System.out.println(pos1);    //0
int pos2 = list.lastIndexOf("bbbb");
System.out.println(pos2);    //2
List<String> sublist = list.subList(1, 4);
System.out.println(sublist);       //[ccccc, bbbb, fff]
...
```

The List interface methods

```
... //[bbbb, ccccc, bbbb, fff, aaa]
list.replaceAll(new ListStringUnaryOperator());
System.out.println(list);        //[4bbbb4, ccccc, 4bbbb4, fff, aaa]

static class ListStringUnaryOperator implements UnaryOperator<String>{
    @Override
    public String apply(String t) {
        String s;
        if (t.length() == 4) {
            s = 4 + t + 4;
        } else {
            s = t;
        }
        return s;
    }
}
...
```

```
public void replaceAll(UnaryOperator<E> operator)
```

The List interface methods

```
... //[4bbbb4, ccccc, 4bbbb4, fff, aaa]
list.sort(null);
System.out.println(list);          //[4bbbb4, 4bbbb4, aaa, ccccc, fff]]
list.sort(new StringLengthComparator());
System.out.println(list);          //[aaa, fff, ccccc, 4bbbb4, 4bbbb4]

static class StringLengthComparator implements Comparator<String> {
      @Override
      public int compare(String o1, String o2) {
         return o1.length() - o2.length();
      }
   }
...
```

The ListIterator interface methods

```
... //[aaa, fff, ccccc, 4bbbb4, 4bbbb4]
ListIterator<String> listIterator = list.listIterator();
/*Iterator in foreach cycle*/                    alternative while cycle
for (listIterator = list.listIterator(); listIterator.hasNext();) {
    String element = listIterator.next();
    System.out.print(element + " ");     //aaa fff ccccc 4bbbb4 4bbbb4
}
System.out.println();

/*Iteration in reverse direction*/
listIterator = list.listIterator(list.size());
while (listIterator.hasPrevious()) {
    System.out.print(listIterator.previous() + " ");
}                                          //4bbbb4 4bbbb4 ccccc fff aaa
System.out.println();
```

public interface ListIterator<E> extends Iterator<E>

The ListIterator interface methods

```
List<String> list = new ArrayList<>(Arrays.asList("aaa", "bbbb", "ccccc" ,
                                                  "bbbb", "fff"));
ListIterator<String> listIterator = list.listIterator();
listIterator.add("ggg");
System.out.println(list);   //[ggg, aaa, bbbb, ccccc, bbbb, fff]
String s2 = listIterator.previous();    //ggg
//       s2 = listIterator.previous();    //NoSuchElementException
listIterator.remove();
System.out.println(list);   //[aaa, bbbb, ccccc, bbbb, fff]
int prevIdx = listIterator.previousIndex();
System.out.println(prevIdx);    //-1  - index out-of-bounds
int nextIdx = listIterator.nextIndex();
System.out.println(nextIdx);    //0  - beginning of the list
int nextIdx1 = listIterator.nextIndex();
System.out.println(nextIdx1);    //0
```

public interface ListIterator<E> extends Iterator<E>

The ListIterator interface methods

```
String s3 = listIterator.next();
System.out.println(s3);        //aaa
System.out.println(list);      //[aaa, bbbb, ccccc, bbbb, fff]
listIterator = list.listIterator(2);
String s7 = listIterator.next();
System.out.println(s7);        //ccccc
listIterator.set("eeee");
System.out.println(list);      //[aaa, bbbb, eeee, bbbb, fff]
String s8 = listIterator.next();
System.out.println(s8);        //bbbb
String s9 = listIterator.next();
System.out.println(s9);        //fff
int nextIdx2 = listIterator.nextIndex();    //index after the last element
System.out.println(nextIdx2);      //5 - returns the size of the list
int nextIdx3 = listIterator.nextIndex();    //fetching the index further
System.out.println(nextIdx3); //5 - returns the size of the list
```

The List inner work

```
public class ArrayList<E> extends AbstractList<E>
      implements List<E>, RandomAccess, Cloneable, java.io.Serializable {
...
private static final int DEFAULT_CAPACITY = 10;
...
/**
  * The array buffer into which the elements of the ArrayList are stored.
*/
transient Object[] elementData;
/**
 * The size of the ArrayList (the number of elements it contains).
*/
private int size;
```

The List interface 4/12

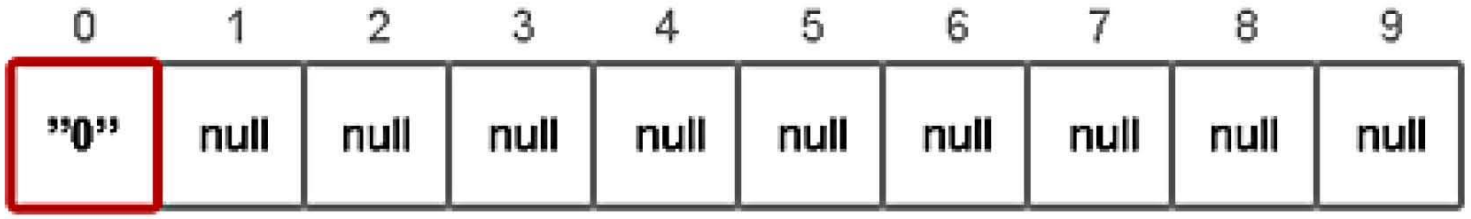
- ArrayList

1. List<String> myList = **new** ArrayList<>();

   elementData = {}, size=0, capacity=0

2. myList.add("**0**");

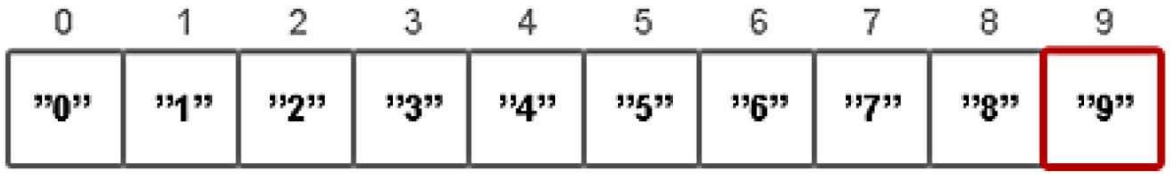
   size=1, capacity=10

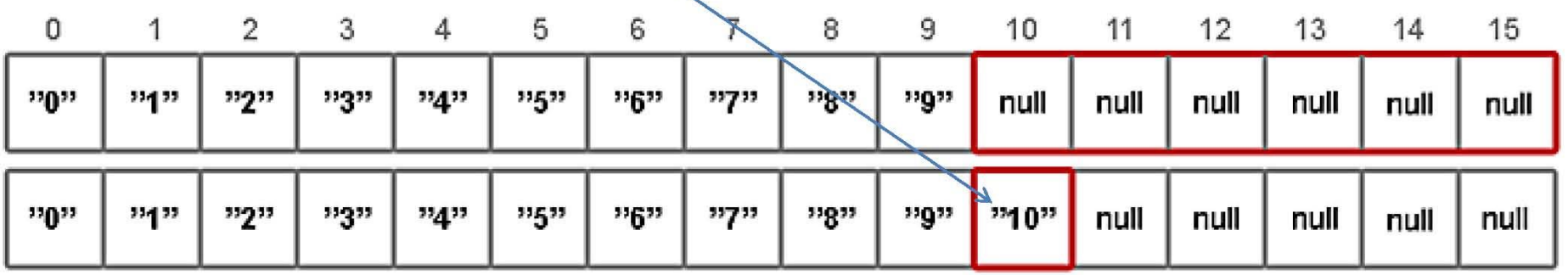


The List interface 5/12

- ArrayList

1. myList.add("1");
2. //...
3. myList.add("9");
4. myList.add("10");





newCapacity = oldCapacity + (oldCapacity >> 1)

15, 22, 33, ...

The List Interface methods implementation

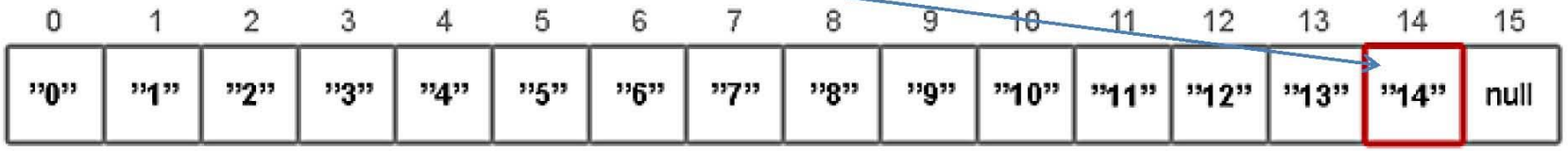
```
public boolean add(E e) {
    modCount++;
    add(e, elementData, size);
    return true;
}
private void add(E e, Object[] elementData, int s) {
    if (s == elementData.length)
        elementData = grow();
        elementData[s] = e;
        size = s + 1;
}
private Object[] grow() {
    return grow(size + 1);
}
```

The List Interface methods implementation

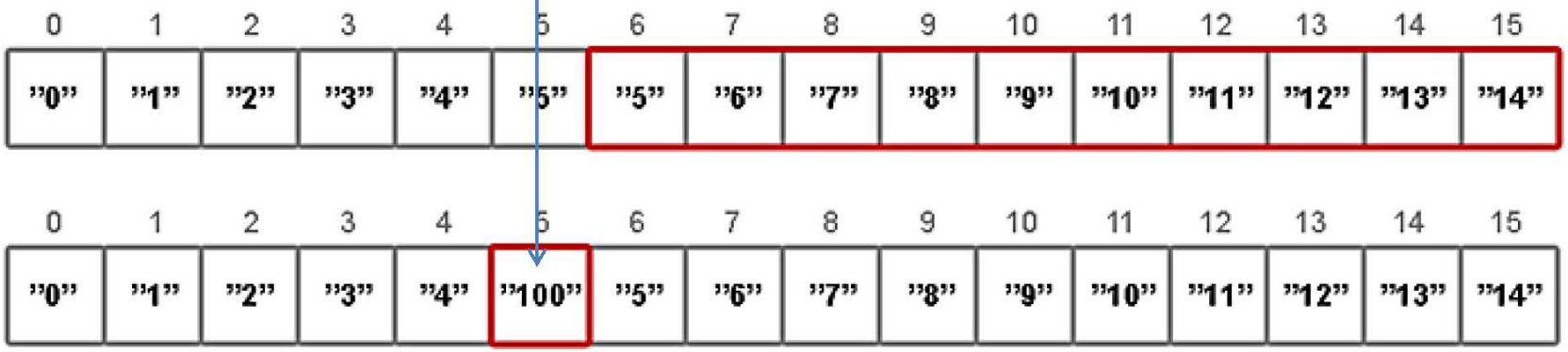
```
private Object[] grow(int minCapacity) {
   return elementData = Arrays.copyOf(elementData,
                                      newCapacity(minCapacity));
}
private int newCapacity(int minCapacity) {
   int oldCapacity = elementData.length;
   int newCapacity = oldCapacity + (oldCapacity >> 1);
      if (newCapacity - minCapacity <= 0) {
         if (elementData == DEFAULTCAPACITY_EMPTY_ELEMENTDATA)
            return Math.max(DEFAULT_CAPACITY, minCapacity);
         if (minCapacity < 0) // overflow
            throw new OutOfMemoryError();
         return minCapacity;
      }
      return (newCapacity - MAX_ARRAY_SIZE <= 0) ? newCapacity
               : hugeCapacity(minCapacity);          }
```

The List interface 6/12

- myList.add("14");



- myList.add(5, "100");

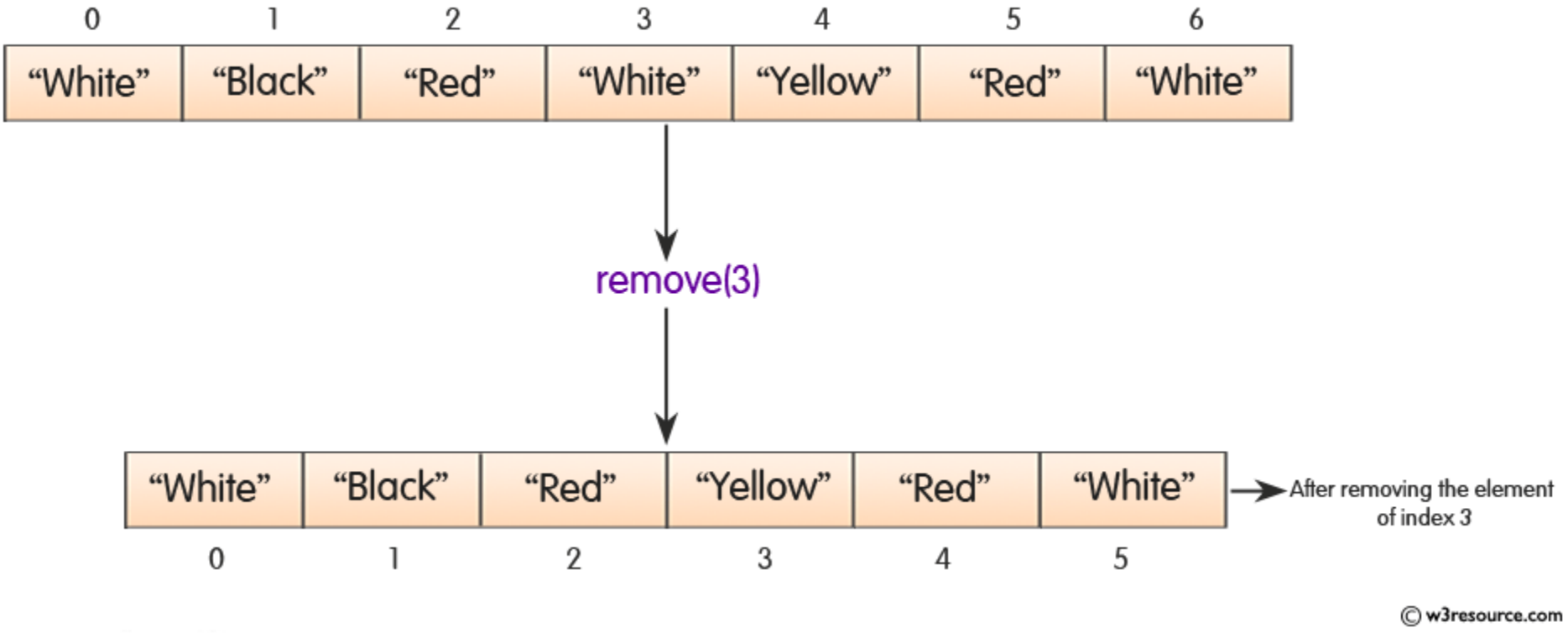


The List Interface methods implementation

```
public void add(int index, E element) {
    rangeCheckForAdd(index);
    modCount++;
    final int s;
    Object[] elementData;
    if ((s = size) == (elementData = this.elementData).length)
        elementData = grow();
        System.arraycopy(elementData, index,
                         elementData, index + 1,
                         s - index);
    elementData[index] = element;
    size = s + 1;
}
```

See Edu Project ArrayListInnerWork...

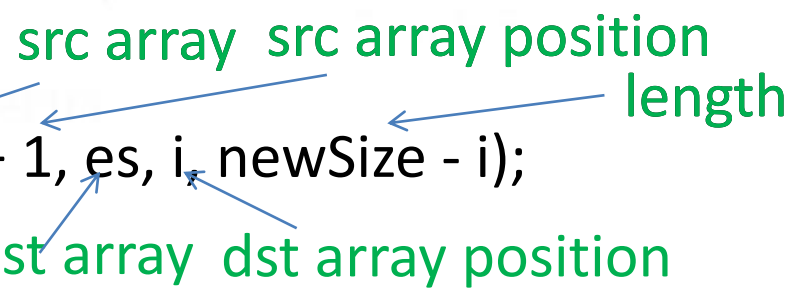
The List Interface



The List Interface methods implementation

```
public E remove(int index) {
        Objects.checkIndex(index, size);
        final Object[] es = elementData;
        @SuppressWarnings("unchecked")
        E oldValue = (E) es[index];
        fastRemove(es, index);
        return oldValue;
}

private void fastRemove(Object[] es, int i) {
        modCount++;
        final int newSize;
        if ((newSize = size - 1) > i)
            System.arraycopy(es, i + 1, es, i, newSize - i);
        es[size = newSize] = null;
}
```



See Edu Project ArrayListInnerWork...

The ArrayList class

```
ArrayList<String> list = new ArrayList<>();
        list.add("aaa");
        System.out.println("size= " + list.size() + ", capacty= " +
ArrayListInnerWork.getCapacity(list));
        list.trimToSize();
        System.out.println("size= " + list.size() + ", capacty= "
+ ArrayListInnerWork.getCapacity(list));
        list.ensureCapacity(3);
        System.out.println("size= " + list.size() + ", capacty= " +
ArrayListInnerWork.getCapacity(list));
```

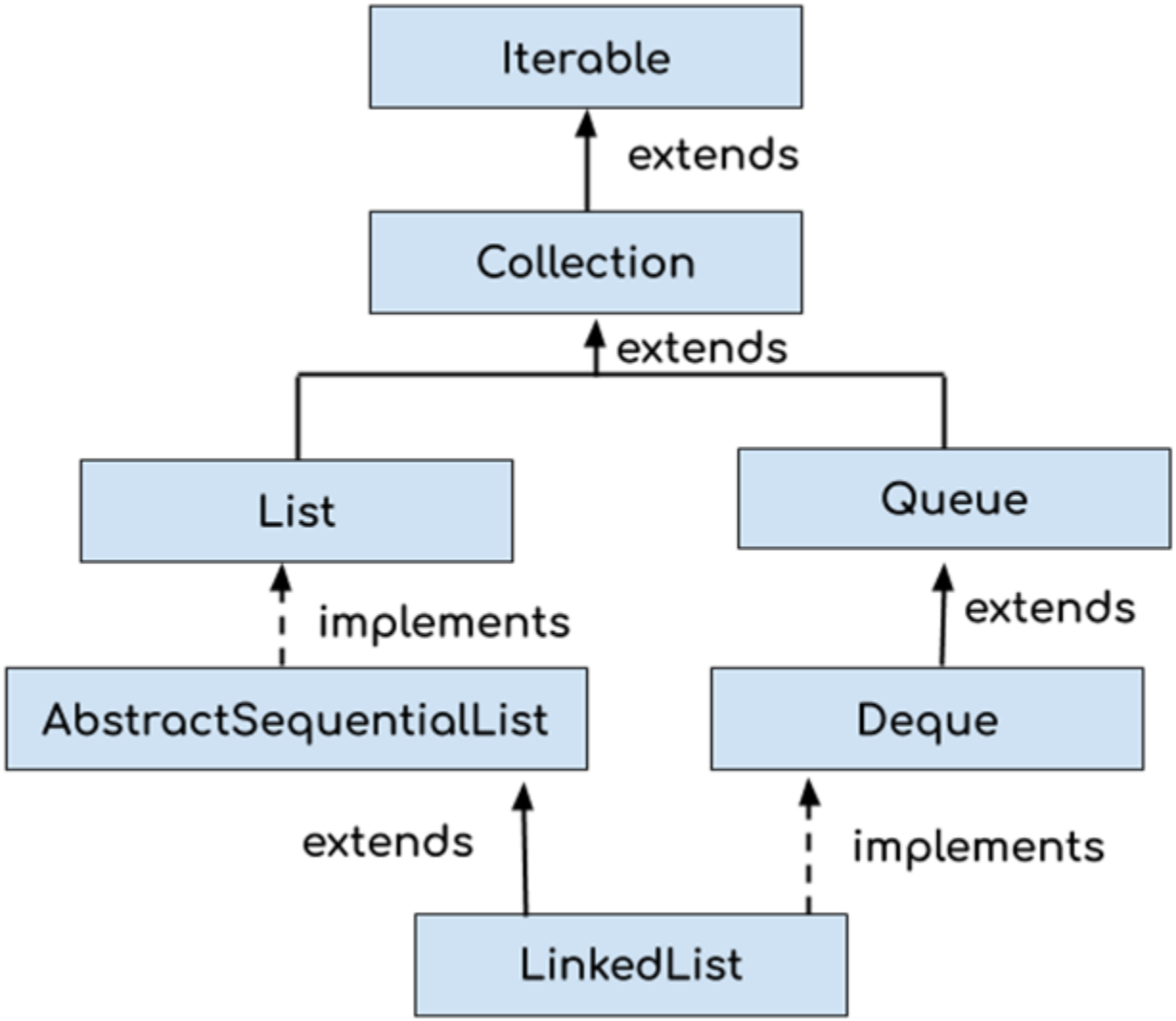
static int getCapacity(ArrayList<?> list) throws Exception is the ArrayListInnerWork Reflection method in Educational Project

See more examples in Edu Project - ArrayListRunner...

The ArrayList class

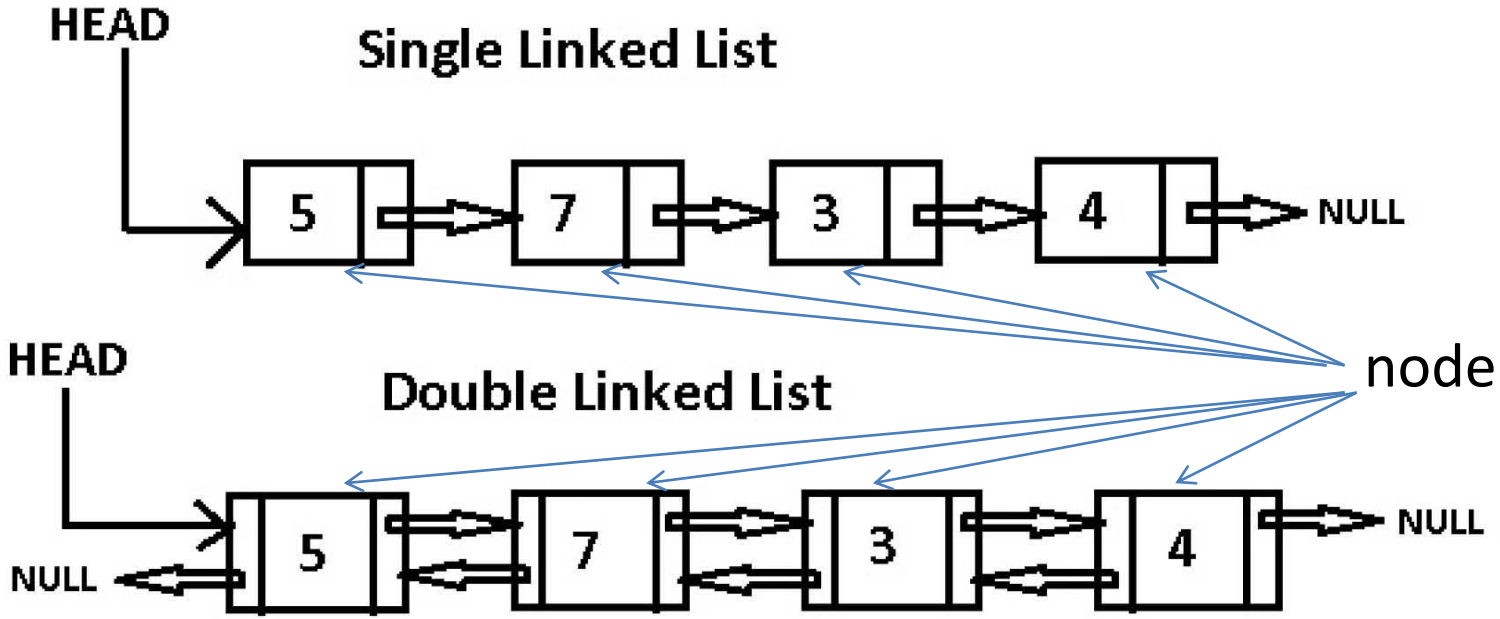
```
public void trimToSize() {
    modCount++;
    if (size < elementData.length) {
        elementData = (size == 0)
                ? EMPTY_ELEMENTDATA
                : Arrays.copyOf(elementData, size);
    }
}
```

List implementation as LinkedList



The List interface 7/12

- A **linked list** is a data structure in which data is stored in structures called nodes and every node has a pointer to the next node in the list



The List interface 8/12

```
1. List<String> lst = new LinkedList<String>();
2. lst.add("aa");
3. lst.add("bbb");
4. lst.add("cccc");
5. System.out.println(lst);
6. String str = lst.get(0);
7. System.out.println(str);
8. lst.add(1,"test");
9. System.out.println(lst);
```

Console output

```
[aa, bbb, cccc]
aa
[aa, test, bbb, cccc]
```

The LinkedList specific methods

```
LinkedList<String> linkedList = new LinkedList<>(Arrays
                    .asList("Item1", "Item2", "Item3", "Item4", "Item5"));
System.out.println(linkedList);     //[Item1, Item2, Item3, Item4, Item5]
linkedList.addFirst("First Item");
linkedList.addLast("Last Item");
System.out.println(linkedList);     //[First Item, Item1, Item2, Item3,
                                    //Item4, Item5, Last Item]
System.out.println(linkedList.getFirst()); //First Item
System.out.println(linkedList.getLast());  //Last Item
System.out.println(linkedList);     //[First Item, Item1, Item2, Item3,
                                    //Item4, Item5, Last Item]
System.out.println(linkedList.removeFirst()); //First Item
System.out.println(linkedList.removeLast());  //Last Item
System.out.println(linkedList);     //[Item1, Item2, Item3, Item4, Item5]
...
```

The LinkedList specific methods

```
...
Iterator<String> descIterator = linkedList.descendingIterator();
        while (descIterator.hasNext()) {
          String s = descIterator.next();
          System.out.print(s + " ");        //Item5 Item4 Item3 Item2 Item1
        }
        System.out.println();
//Additional methods for queues
String s1 = linkedList.element();   //does not remove == getFirst()
System.out.println(s1);              //Item1
boolean b = linkedList.offer("Item6");  //== add(e)
System.out.println("b= " + b + ", " + linkedList);  //b= true, [Item1,
                                        //Item2, Item3, Item4, Item5, Item 6]
boolean b1 = linkedList.offerFirst("Item0");  //==addFirst(e)
boolean b2 = linkedList.offerLast("Item7"); //==addLast(e)
System.out.println(linkedList); //[Item0, Item1, Item2, Item3, Item4,
                                  //Item5, Item6, Item7]
```

The LinkedList specific methods

```
...
System.out.println(linkedList.peek());     //gets 1st elem, doesn't remove
System.out.println(linkedList.peekFirst()); //Item0 == peek()
System.out.println(linkedList.peekLast()); //gets last elem, doesn't
                                           //remove
System.out.println(linkedList);  //[Item0, Item1, Item2, Item3, Item4,
                                 //Item5, Item6, Item7] - not remove
System.out.println(linkedList.poll());       //gets 1st elem, removes it
System.out.println(linkedList.pollFirst()); //Item1 == poll()
System.out.println(linkedList.pollLast()); //gets last elem, removes it
System.out.println(linkedList); //[Item2, Item3, Item4, Item5, Item6]
System.out.println(linkedList.pop());       //gets 1st elem, removes it
System.out.println(linkedList);      //[Item3, Item4, Item5, Item6]
linkedList.push("Item2");            //adds 1st elem
System.out.println(linkedList);      //[Item2, Item3, Item4, Item5, Item6]
```

The LinkedList specific methods

	Add element to start of list	Add element to end of list	Get element from start of list	Get element from end of list
Element does not remove from the list			throws NoSuchElementException for empty list	
	void addFirst(E e) (Deque)	void addLast(E e) (Deque)	E getFirst() (Deque) E element() (Queue)	E getLast() (Deque)
			returns null for empty list	
	boolean offerFirst(E e) (Deque) void push(E e) (Deque)	boolean offer(E e) (Queue) boolean offerLast(E e) (Deque)	E peek() (Queue) E peekFirst() (Deque)	E peekLast() (Deque)
Element removes from the list			throws NoSuchElementException for empty list	
			E remove() (Queue) E removeFirst() (Deque) E pop() (Deque)	E removeLast() (Deque)
			returns null for empty list	
			E poll() (Queue) E pollFirst() (Deque)	E pollLast() (Deque)

LinkedList inner work

```
public class LinkedList<E>
    extends AbstractSequentialList<E>
    implements List<E>, Deque<E>, Cloneable, java.io.Serializable {

    transient int size = 0;
    transient Node<E> first;
    transient Node<E> last;
...

    private static class Node<E> {
        E item;
        Node<E> next;
        Node<E> prev;
        Node(Node<E> prev, E element,
                Node<E> next) {...}
...}
}
```

first node

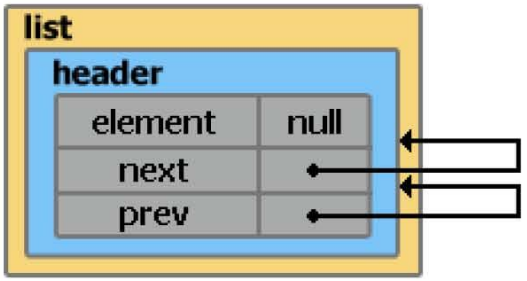
last node

next node

previous node

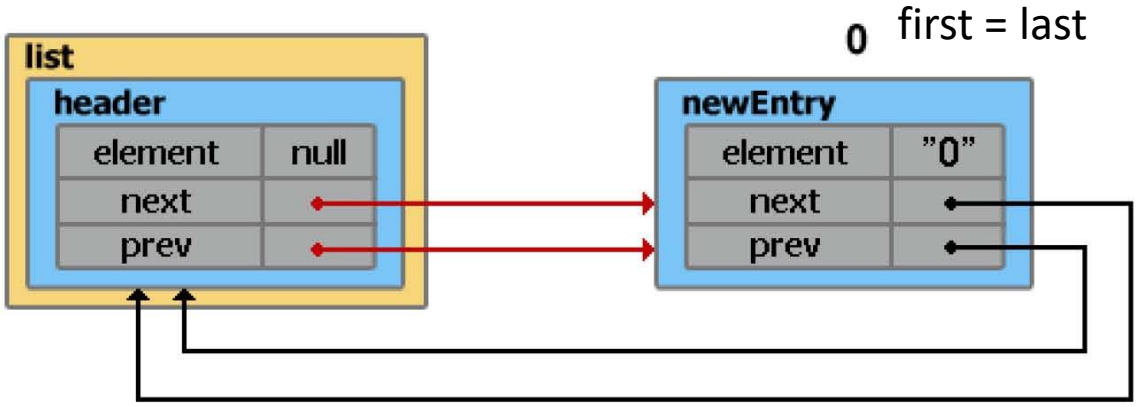
The List interface 9/12

- List<String> list = **new** LinkedList<>();



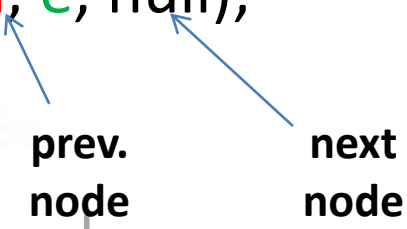

```
assert (size == 0)
    ? (first == null && last == null)
    : (first.prev == null && last.next == null);
```

- list.add("0");



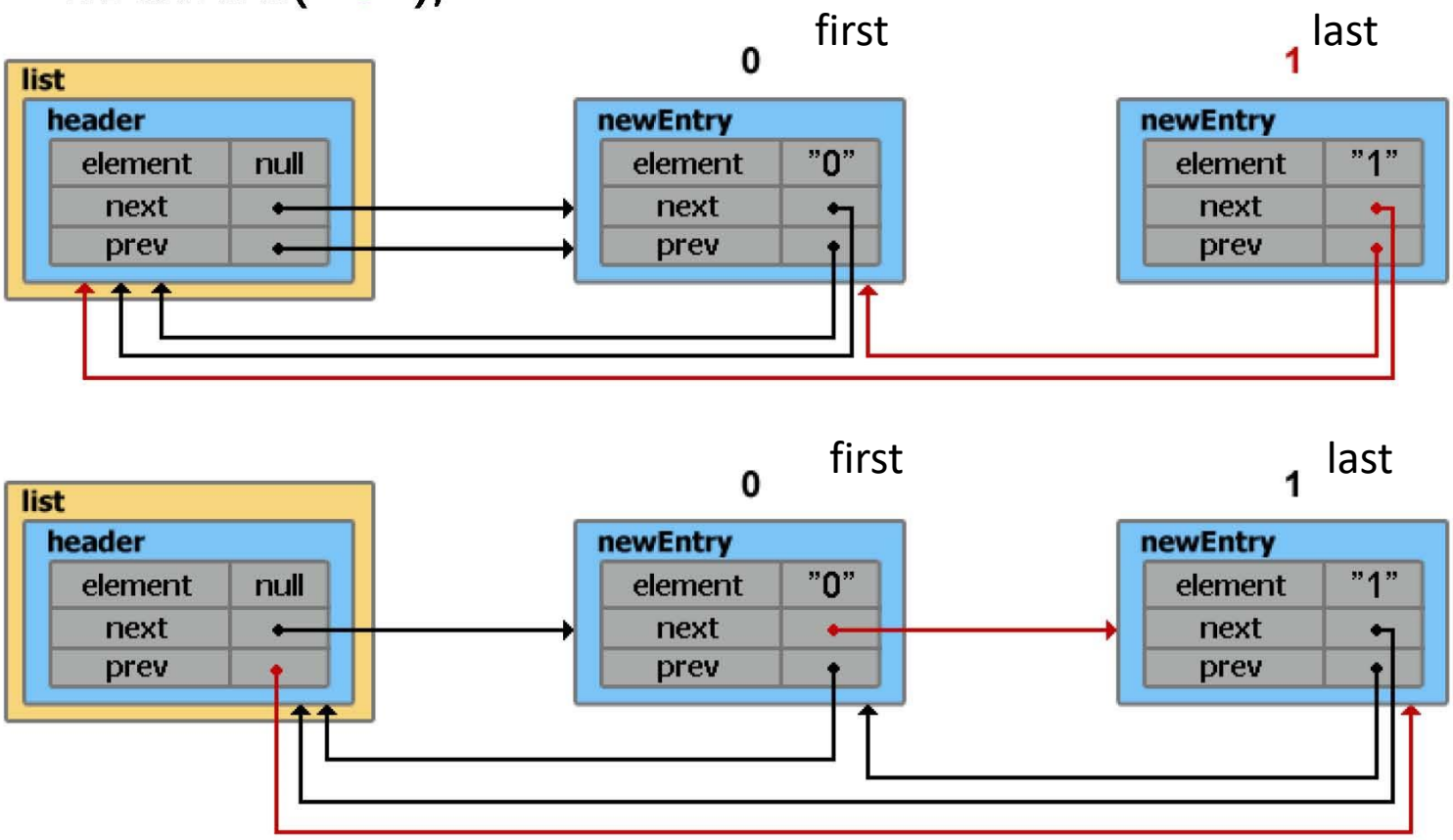
LinkedList inner work

```
public boolean add(E e) {
        linkLast(e);
        return true;
}
void linkLast(E e) {
        final Node<E> l = last; //take header.last value
        final Node<E> newNode = new Node<>(l, e, null);
        last = newNode; // set last to last node
        if (l == null) //if list empty
            first = newNode;       //set first to first node
        else
            l.next = newNode; //set next pointer of last node
        size++;
        modCount++;
}
```



The List interface 10/12

- list.add("1");



LinkedList inner work

```
public void add(int index, E element) {
      … if (index == size)
            linkLast(element);
      else
            linkBefore(element, node(index));
   }
void linkBefore(E e, Node<E> succ) {  // assert succ != null
      final Node<E> pred = succ.prev; //ук-ль на пред.
      final Node<E> newNode = new Node<>(pred, e, succ);
      succ.prev = newNode;
      if (pred == null)
         first = newNode;
      else
         pred.next = newNode;
      size++;        modCount++;     }
```

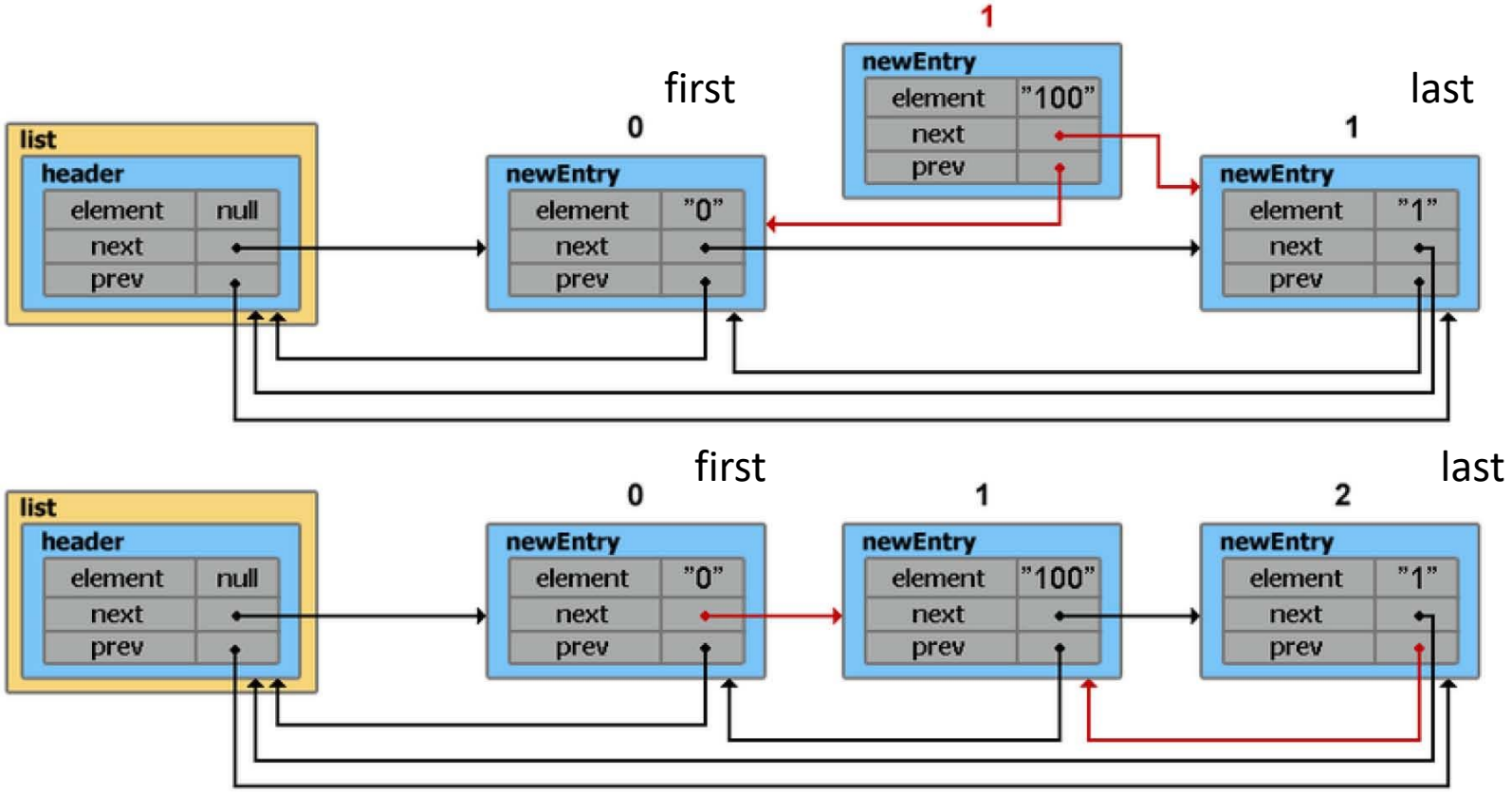
узел, после которого вставка

пред. узел

след. узел

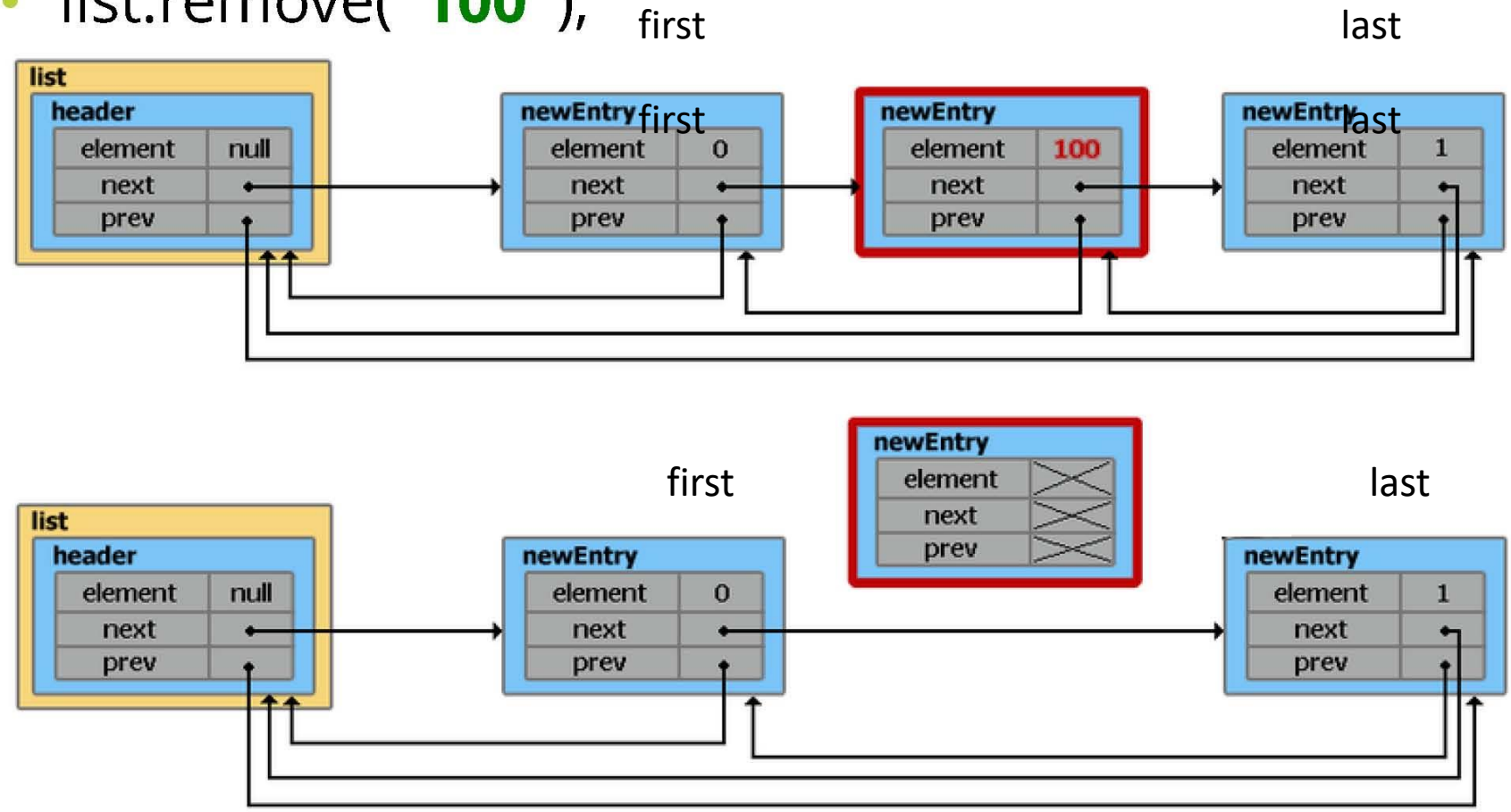
The List interface 11/12

- list.add(1, **"100"**);



The List interface 12/12

- list.remove("100");



LinkedList inner work

```
public boolean remove(Object o) {
    if (o == null) {
        for (Node<E> x = first; x != null; x = x.next) {
            if (x.item == null) {
                unlink(x);
                return true;
            }
        }
    } else {
        for (Node<E> x = first; x != null; x = x.next) {
            if (o.equals(x.item)) {
                unlink(x);
                return true;
            }
        }
    }
    return false;     }
```

LinkedList inner work

```
E unlink(Node<E> x) {
    // assert x != null;
    final E element = x.item;
    final Node<E> next = x.next;
    final Node<E> prev = x.prev;
    if (prev == null) {
        first = next;
    } else {
        prev.next = next;
        x.prev = null;
    }
    if (next == null) {
        last = prev;
    } else {
        next.prev = prev;
        x.next = null;
    } ...
```

```
...
        x.item = null;
        size--;
        modCount++;
        return element;
    }
```

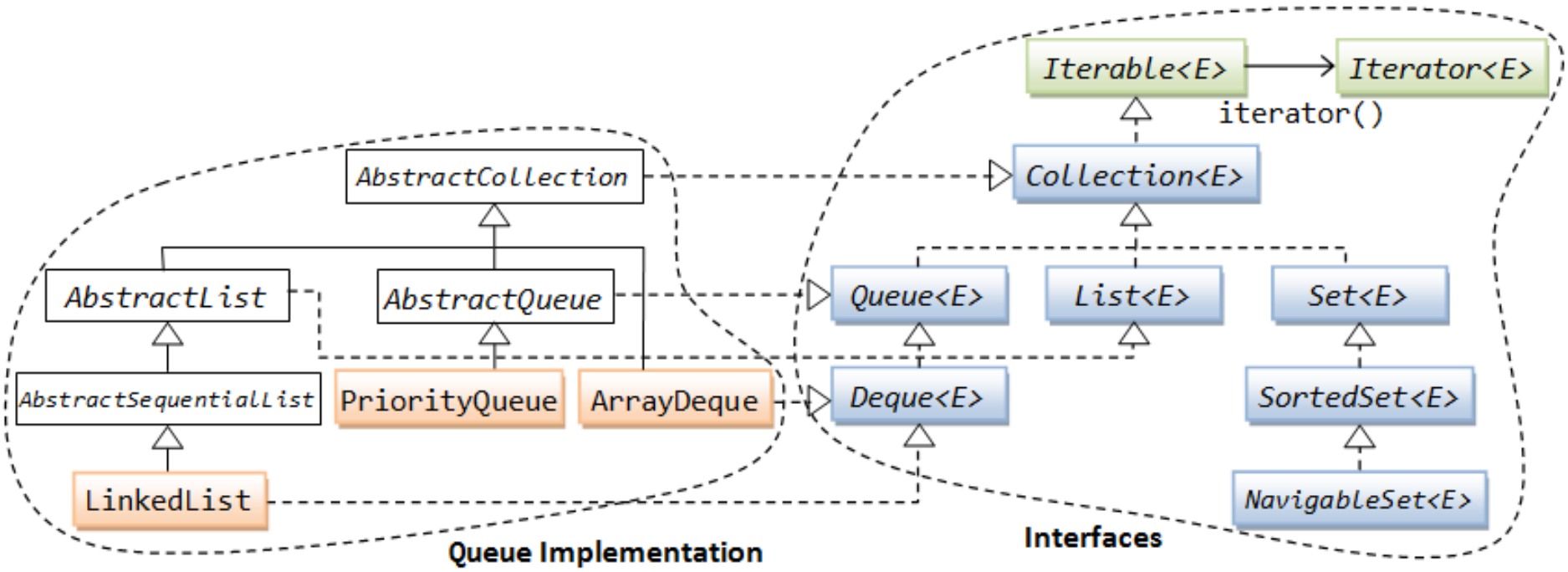
ArrayList and LinkedList Performance Comparison

	Start of the list	The list middle	End of the list	Homogeneous in the list
Insert	LinkedList	ArrayList	≈	ArrayList
Remove	LinkedList	ArrayList	ArrayList	ArrayList
Update	ArrayList	ArrayList	ArrayList	ArrayList
Get	ArrayList	ArrayList	ArrayList	ArrayList

Module contents

- Collections
  - Java Collection Framework, Interfaces
  - The Collection Interface
  - The Iterators
  - The List Interface
  - The Queue and Deque Interfaces
  - The Set Interface
  - The Map Interface
  - The Collection Class

The Queue Interface 1/6



The Queue interface 2/6

- The Queue
- The first element that is inserted is the first to be removed
- Often referred to as First-in First-out (FIFO) collection

- The Deque is a linear collection that supports element insertion and removal at both ends

Queue vs Deque

E peek()
E element()
gets, doesn't remove

boolean offer(E e)

E poll()
E remove()
gets, remove

E peekFirst()
E getFirst()
gets, doesn't remove

E peekLast()
E getLast()
gets, doesn't remove

void addFirst(E e)
boolean offerFirst(E e)
void push(E e)

void addLast(E e)
boolean offerLast(E e)

E pollFirst()
E removeFirst()
E pop()
gets, remove

E pollLast()
E removeFLast()
gets, remove

The Queue interface methods

LinkedList implementation

```
boolean add(E e)        -> linkLast(e);
                           return true;
boolean offer(E e);     -> return add(e);

E element()             -> return getFirst();
E peek();               -> final Node<E> f = first;
                           return (f == null) ? null : f.item;

E poll();               -> final Node<E> f = first;
                           return (f == null) ? null : unlinkFirst(f);
E remove();             ->return removeFirst();
```

remain in the queue

removed from the queue

```
public E getFirst() {
   final Node<E> f = first;
   if (f == null)
      throw
      new NoSuchElementException();
   return f.item; }
```

```
public E removeFirst() {
      final Node<E> f = first;
      if (f == null)
         throw
   new NoSuchElementException();
      return unlinkFirst(f);    }
```

The Queue interface methods

LinkedList implementation

```
private E unlinkFirst(Node<E> f) {
    // assert f == first && f != null;
    final E element = f.item;
    final Node<E> next = f.next;
    f.item = null;
    f.next = null; // help GC
    first = next;
    if (next == null)
        last = null;
    else
        next.prev = null;
    size--;
    modCount++;
    return element;
}
```

The Queue interface 3/6

- ```
  Queue<String> q = new LinkedList<>();
  q.add("aa"); q.add("bbb");
  q.add("cccc"); q.add("dddddd");
  System.out.println(q);
  System.out.println(q.remove());
  System.out.println(q.remove());
  System.out.println(q.remove());
  System.out.println(q);
  ```

  ```
  q.remove();
  ```

  q.remove();

  NoSuchElementException!!!

Console output

```
[aa, bbb, cccc, dddddd]
aa
bbb
cccc
[dddddd]
```
```

The Queue interface 4/6

- Queue<String> q = **new** LinkedList<>();
- q.offer(**"aa"**); q.offer(**"bbb"**);
  q.offer(**"cccc"**); q.offer(**"ddddddd"**);
  System.***out***.println(q);
  System.***out***.println(q.poll());
  System.***out***.println(q.poll());
  System.***out***.println(q.poll());
  System.***out***.println(q);
  q.poll();
  System.***out***.println(q.poll());

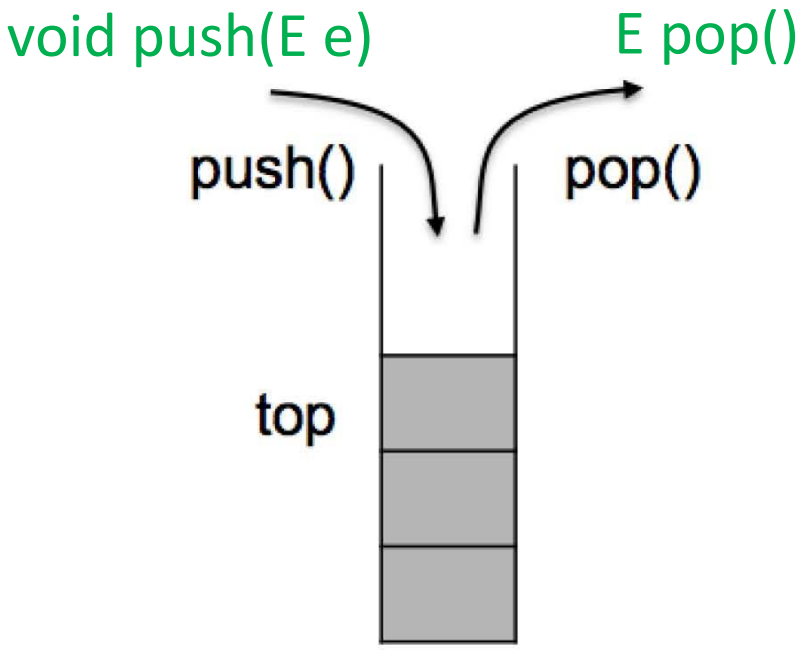
Console output

[aa, bbb, cccc, ddddddd]
aa
bbb
cccc
[ddddddd] null

See Edu Project QueueRunner...

The Deque interface 5/6

- Stack
- Elements are stored in order of insertion and the elements are removed in reverse order
- Often referred to as a Last-in First-out (LIFO) collection



The Deque interface - stack implementation

```
Deque<String> stack= new LinkedList<>();
stack.push("aaa");
stack.push("bbbb");
stack.push("ccccc");
Systen.out.println(stack);          //[ccccc, bbbb, aaa]
System.out.println(stack.pop());  //ccccc
System.out.println(stack.pop());  //bbbb
System.out.println(stack.pop());  //aaa
System.out.println(stack.pop());
```

java.util.NoSuchElementException

See Edu Project DequeRunner and StackRunner...

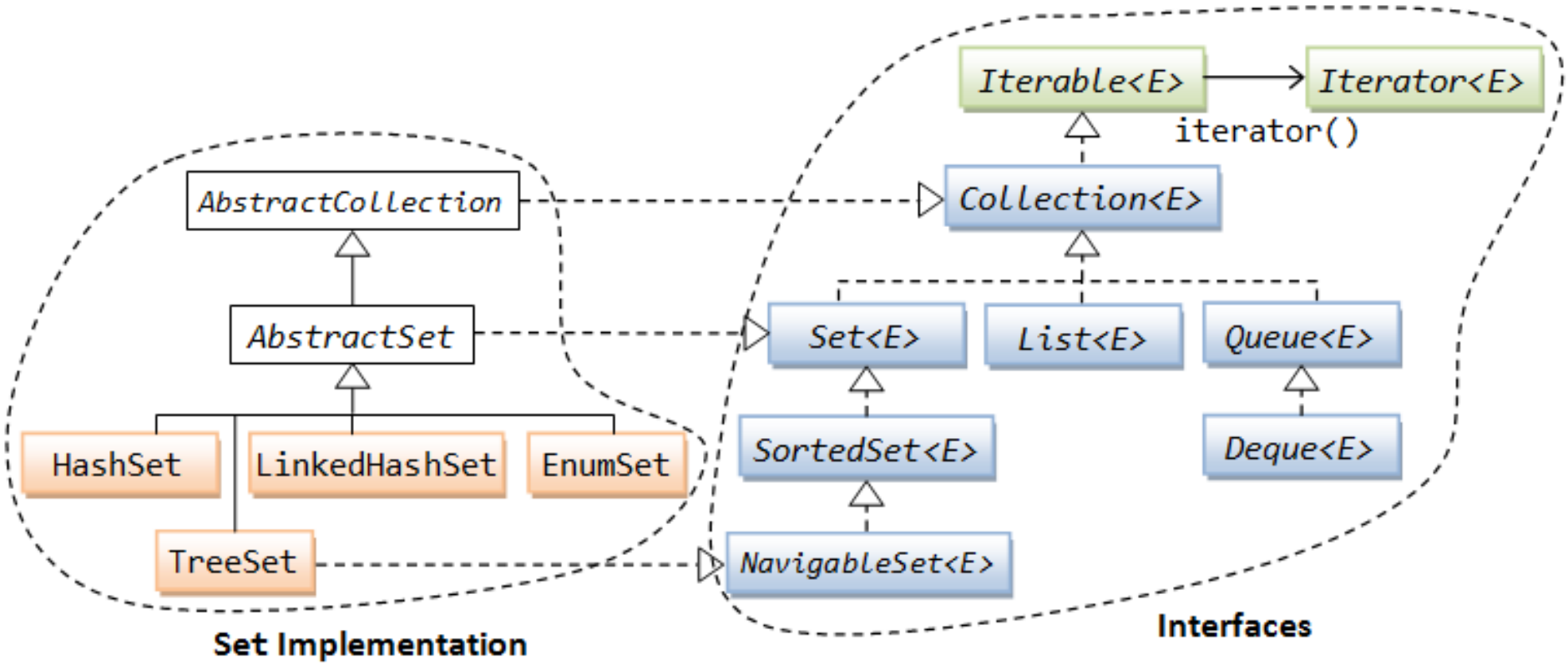
Module contents

- Collections
  - Java Collection Framework, Interfaces
  - The Collection Interface
  - The Iterators
  - The List Interface
  - The Queue and Deque Interfaces
  - The Set Interface
  - The Map Interface
  - The Collection Class

The Set interface 1/6

- A Set is a Collection that cannot contain duplicate elements
- Set interface models the mathematical set abstraction.
- The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.
- Set also adds a stronger contract on the behavior of the equals and hashCode operations, allowing Set instances to be compared meaningfully even if their implementation types differ.
- Two Set instances are equal if they contain the same elements.

The Set interface 2/6



The Set interface 3/6

- **HashSet** no guarantees concerning the order of iteration

```
1. Set<String> mySet = new HashSet<>();
2. mySet.add("aaa");      //returns true
3. mySet.add("bbbb");     //returns true
4. mySet.add("bbbb");     //returns false
5. mySet.add("aaa");      //returns false
6. mySet.add("ccccc");    //returns true
7. mySet.add("a");        //returns true
8. mySet.add("cc");       //returns true
9. System.out.println(mySet);
```

Console output

[bbbb, aaa, ccccc, a, cc]

See Edu Project HashSetRunner and Student...

Immutable and Mutable Sets

```
//      Set<Character> lett = Set.of('z', 'o', 'o');    //duplicate element: o
        Set<Character> letters = Set.of('z', 'o');
        System.out.println(letters);        //[z, o]
//      letters.add('!');                   //UnsupportedOperationException
//      letters.remove('z');                //UnsupportedOperationException

        Set<Character> copy = Set.copyOf(letters);
        System.out.println(copy);           //[z, o]
//      copy.add('!');                      //UnsupportedOperationException
//      copy.remove('z');                   //UnsupportedOperationException

        Set<Character> mutSet = new HashSet<>(letters);
        System.out.println(mutSet);     //[z, o]
        mutSet.add('!');
        mutSet.remove('z');
        System.out.println(mutSet);     //[!, o]
```

Immutable

Mutable

The Set interface - HashSet inner work

```
public class HashSet<E> extends AbstractSet<E> implements Set<E>,
Cloneable, java.io.Serializable {
    ...
    private transient HashMap<E,Object> map;
    // Dummy value to associate with an Object in the backing Map
    private static final Object PRESENT = new Object();
...
    public boolean add(E e) {
        return map.put(e, PRESENT)==null;
    }
...
public boolean remove(Object o) {
        return map.remove(o)==PRESENT;
    }
```

Inner work will be explored for Map<K,V>

The Set interface 4/6

- **TreeSet** orders its elements based on their values

```
1. Set<String> mySet = new TreeSet<>();
2. mySet.add("aaa");     //returns true
3. mySet.add("bbbb");  //returns true
4. mySet.add("bbbb");  //returns false
5. mySet.add("aaa");    //returns false
6. mySet.add("ccccc");//returns true
7. mySet.add("a");   //returns true
8. mySet.add("cc");//returns true
9. System.out.println(mySet);
```

Console output

[a, aaa, bbbb, cc, ccccc]

See Edu Project TreeSetRunner and Student...

The Set interface - TreeSet inner work

```
public class TreeSet<E> extends AbstractSet<E>
    implements NavigableSet<E>, Cloneable, java.io.Serializable {
...
    private transient NavigableMap<E,Object> m;
    // Dummy value to associate with an Object in the backing Map
    private static final Object PRESENT = new Object();
...
    public boolean add(E e) {
        return m.put(e, PRESENT)==null;
    }
...
public boolean remove(Object o) {
        return m.remove(o)==PRESENT;
    }
```

The SortedSet interface

```
public interface SortedSet<E> extends Set<E> {
    Comparator<? super E> comparator();
    SortedSet<E> headSet(E toElement);
    SortedSet<E> tailSet(E fromElement);
    SortedSet<E> subSet(E fromElement, E toElement);
    E first();
    E last();
    default Spliterator<E> spliterator() {...}
```

stores elements in sorted form (sort ascending)

See Edu Project SortedSetRunner...

The NavigableSet interface

```
public interface NavigableSet<E> extends SortedSet<E> {
   Iterator<E> descendingIterator();
   NavigableSet<E> descendingSet();
   NavigableSet<E> subSet(E fromElement, boolean fromInclusive,
                          E toElement,   boolean toInclusive);
   NavigableSet<E> headSet(E toElement, boolean inclusive);
   NavigableSet<E> tailSet(E fromElement, boolean inclusive);
   E floor(E e);
   E ceiling(E e);
   E lower(E e);
   E higher(E e);
   E pollFirst();
   E pollLast();
}
```

allows to retrieve elements based on their values

since Java 6

See Edu Project NavigableSetRunner...

The Set interface 5/6

- **LinkedHashSet** elements based on the order in which they were inserted

```
1. Set<String> mySet = new LinkedHashSet<>();
2. mySet.add("aaa");
3. mySet.add("bbbb");
4. mySet.add("bbbb");
5. mySet.add("aaa");
6. mySet.add("ccccc");
7. mySet.add("a");
8. mySet.add("cc");
9. System.out.println(mySet);
```

Console output

[aaa, bbbb, ccccc, a, cc]

See Edu Project LinkedHashSetRunner...

The Set interface - LinkedHashSet inner work

```
public class LinkedHashSet<E> extends HashSet<E>
    implements Set<E>, Cloneable, java.io.Serializable {
...
```

NO INNER WORK - EXTENDS HashSet

The Set interface 6/6

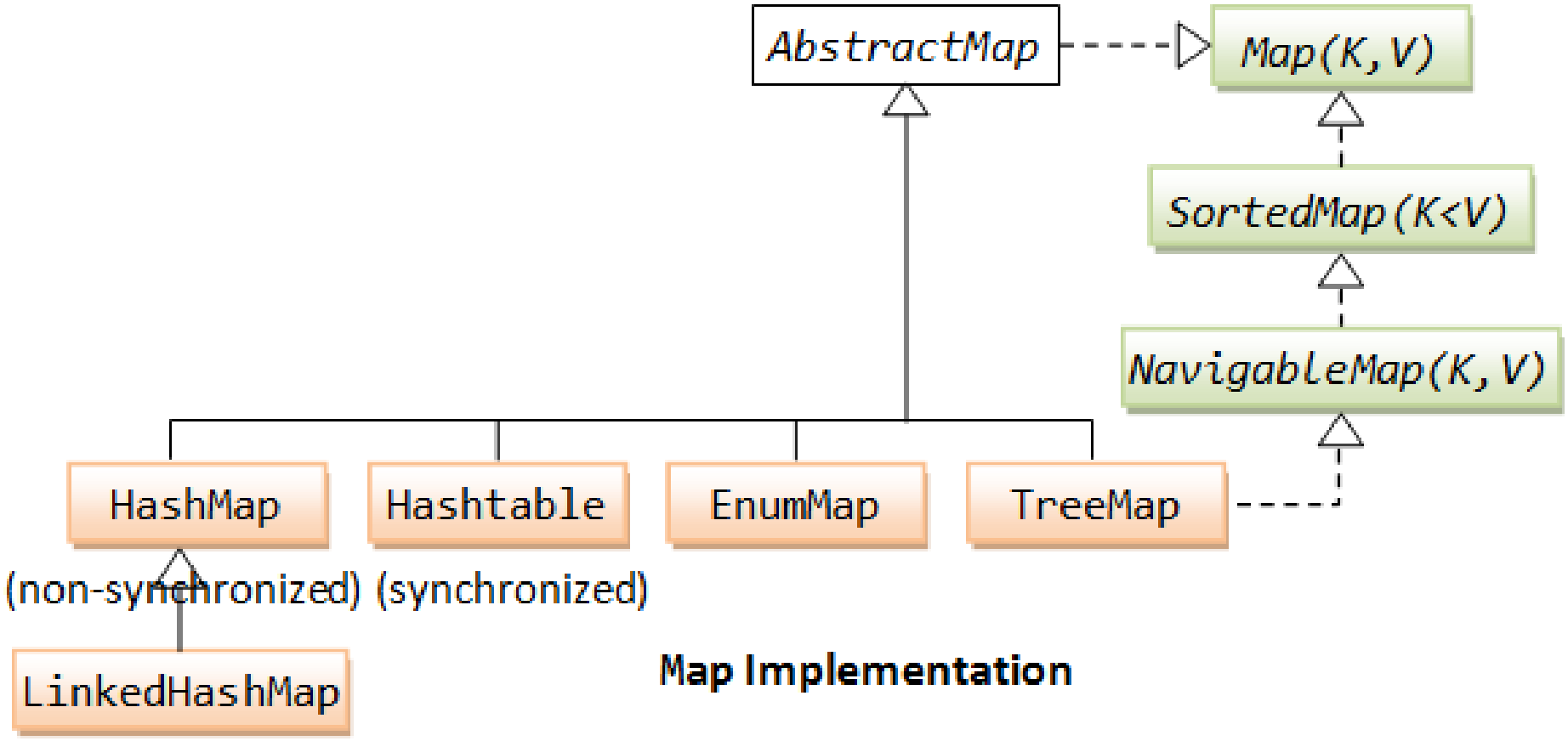
Constructs a new set containing the elements in the specified collection

```
1. List<String> myList = new ArrayList<>();
2. myList.add("aaa");
3. myList.add("bbbb");
4. myList.add("bbbb");
5. myList.add("aaa");
6. myList.add("ccccc");
7. Set<String> mySet = new HashSet<>(myList);
8. System.out.println(myList);
9. System.out.println(mySet);
```

Module contents

- Collections
  - Java Collection Framework, Interfaces
  - The Collection Interface
  - The Iterators
  - The List Interface
  - The Queue and Deque Interfaces
  - The Set Interface
  - The Map Interface
  - The Collection Class

The Map interface 1/13



Map Implementation

The Map interface abstract methods

```
public interface Map<K, V> {
   boolean containsKey(Object key);
   boolean containsValue(Object value);
   Set<Map.Entry<K, V>> entrySet();
   V get(Object key);
   Set<K> keySet();
   V put(K key, V value);
   void putAll(Map<? extends K, ? extends V> m);
   V remove(Object key);
   Collection<V> values();
... default and static methods
}
```

The Map interface 2/13

- **The HashMap**

```
1. Map<String, Integer> hm = new HashMap<>();
2. hm.put("one", 1);
3. hm.put("two", 2);
4. hm.put("three", 3);
5. System.out.println(hm);
6. int x = hm.get("two");
7. System.out.println(x);
```

console output

```
{two=2, one=1, three=3}
2
```

See Edu Project HashMapRunner...

Getting Values Safely

- The get() method returns null if the requested key is not in the map.
- Sometimes you prefer to have a different value returned.

public default V **getOrDefault**(Object key, V defaultValue)

```
Map<Character, String> map = new HashMap<>();
map.put('x', "spot");
System.out.println("X marks the " + map.get('x'));
System.out.println("X marks the " + map.getOrDefault('x', ""));
System.out.println("Y marks the " + map.get('y'));
System.out.println("Y marks the " + map.getOrDefault('y', ""));
```

Since Java 8

Factory methods to create Map

```
/*Inconvenient way - Passing keys and values is harder to read
   because you have to keep track of which parameter is which*/
Map<String, String> fabrMap = Map.of("key1", "value1", "key2",
                                                    "value2");
/*Convenient way*/
Map<String, String> fabricMap = Map.ofEntries(
      Map.entry("key1", "value1"),
      Map.entry("key2", "value2"));
System.out.println(fabricMap);     //{key1=value1, key2=value2}
//      fabricMap.put("key3", "value3");   //UnsupportedOperationException
//      fabricMap.remove("key1");          //UnsupportedOperationException
//      fabricMap.replace("key2", "abc");  //UnsupportedOperationException

Map<String, String> copyMap = Map.copyOf(fabricMap);
System.out.println(copyMap);       //{key1=value1, key2=value2}
//      copyMap.put("key3", "value3");   //UnsupportedOperationException
//      copyMap.remove("key1");          //UnsupportedOperationException
//      copyMap.replace("key2", "abc");  //UnsupportedOperationException
```

Immutable map

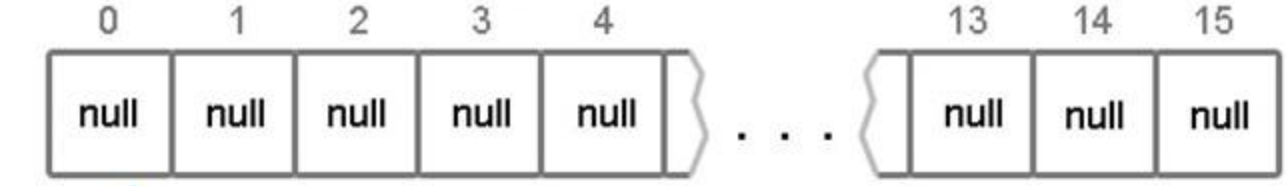
Factory methods to create Map

```
...
    Map<String, String> mutMap = new HashMap<>(fabricMap);
    System.out.println(mutMap);       //{key1=value1, key2=value2}
    mutMap.put("key3", "value3");
    mutMap.remove("key1");
    mutMap.replace("key2", "abc");
    System.out.println(mutMap);       //{key2=abc, key3=value3}
```

Mutable map

The Map interface 3/13

- Map<String, String> hashmap = **new** HashMap<>();



array of bucket

```
public class HashMap<K,V> extends AbstractMap<K,V>
    implements Map<K,V>, Cloneable, Serializable {
  transient Node<K, V>[] table;
  //...
  {
    static class Node<K, V> implements Map.Entry<K, V> {
      final K key;
      V value;
      Node<K, V> next;
      final int hash;
      //...
```

The Map interface

size - number of HashMap elements

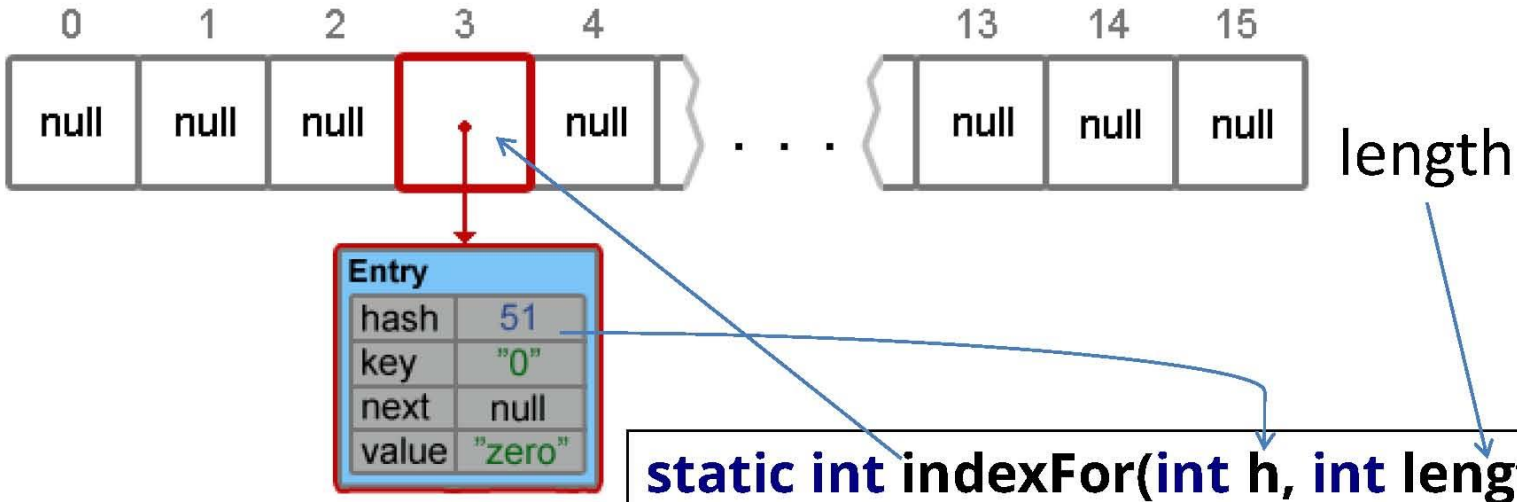
capacity - size of Node<K, V> array (initial = 16, max = 1 073 741 824);

loadFactor - the measure that decides when to increase the capacity of the Map, the default value is 0.75;

threshold - the maximum number of elements at which the Node<K, V> array size is doubled. It is calculated by the formula (capacity * loadFactor) - default is 16 * 0.75 = 12;

The Map interface 4/13

1. hashmap.put("0", "zero");

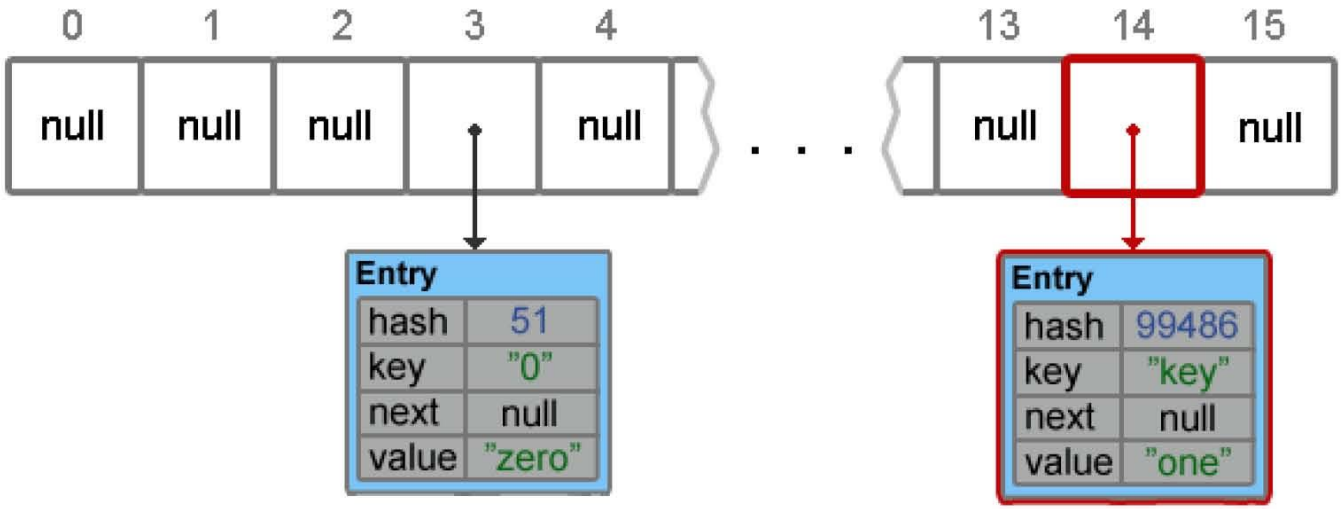



```
static int indexFor(int h, int length)
{
   return h & (length - 1);
}
```

i= 51 & 15= 3

The Map interface 5/13

- hashmap.put("**key**", "**one**");

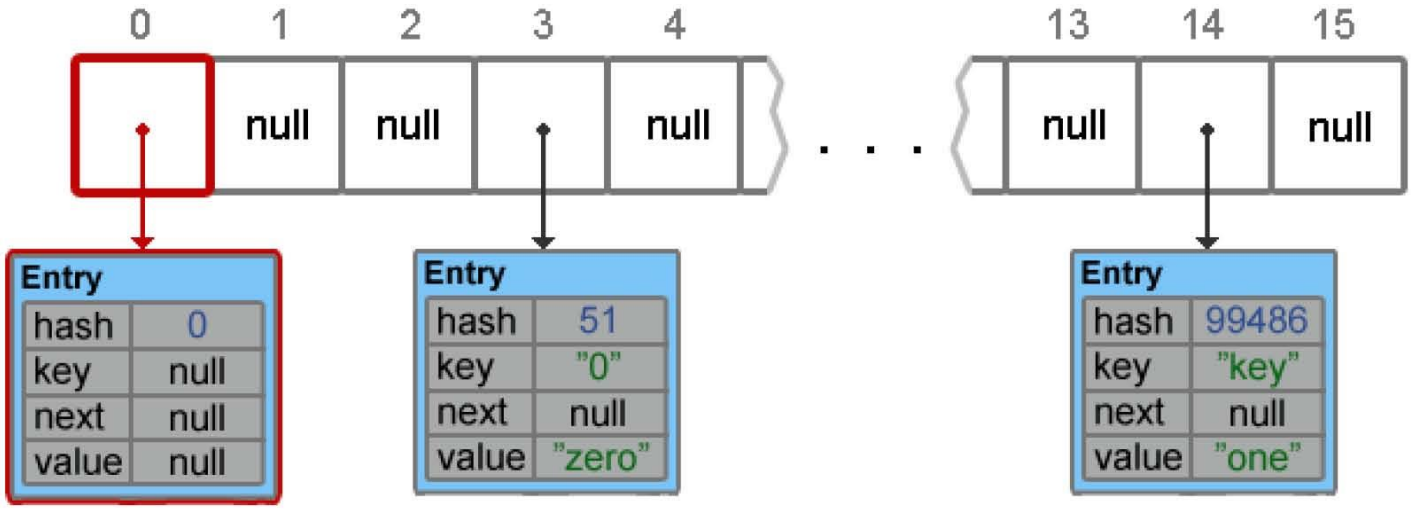


i= 99486 & 15= 14

The Map interface 6/13

value can be arbitrary

- hashmap.put(**null**, **null**);

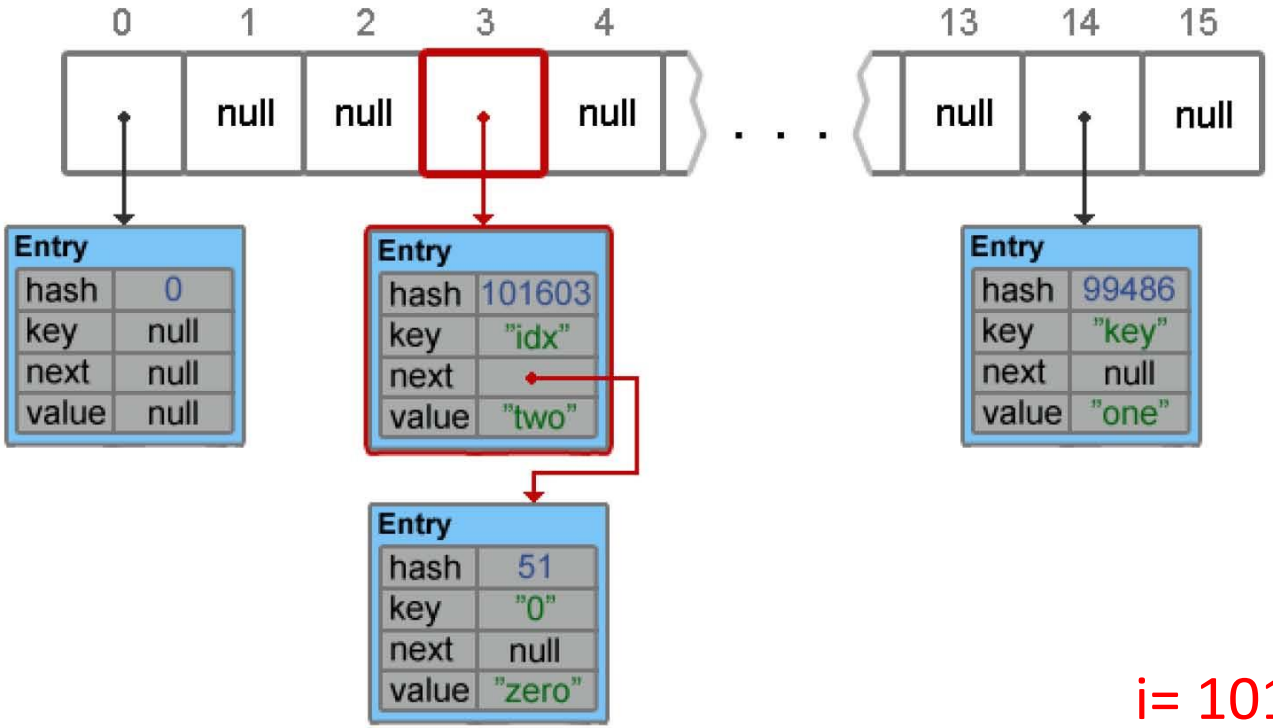



```
static final int hash(Object key) {
    int h;
    return (key == null) ? 0 : (h = key.hashCode()) ^ (h >>> 16);
}
```

i= 0 & 15= 0

The Map interface 7/13

- hashmap.put("idx", "two");

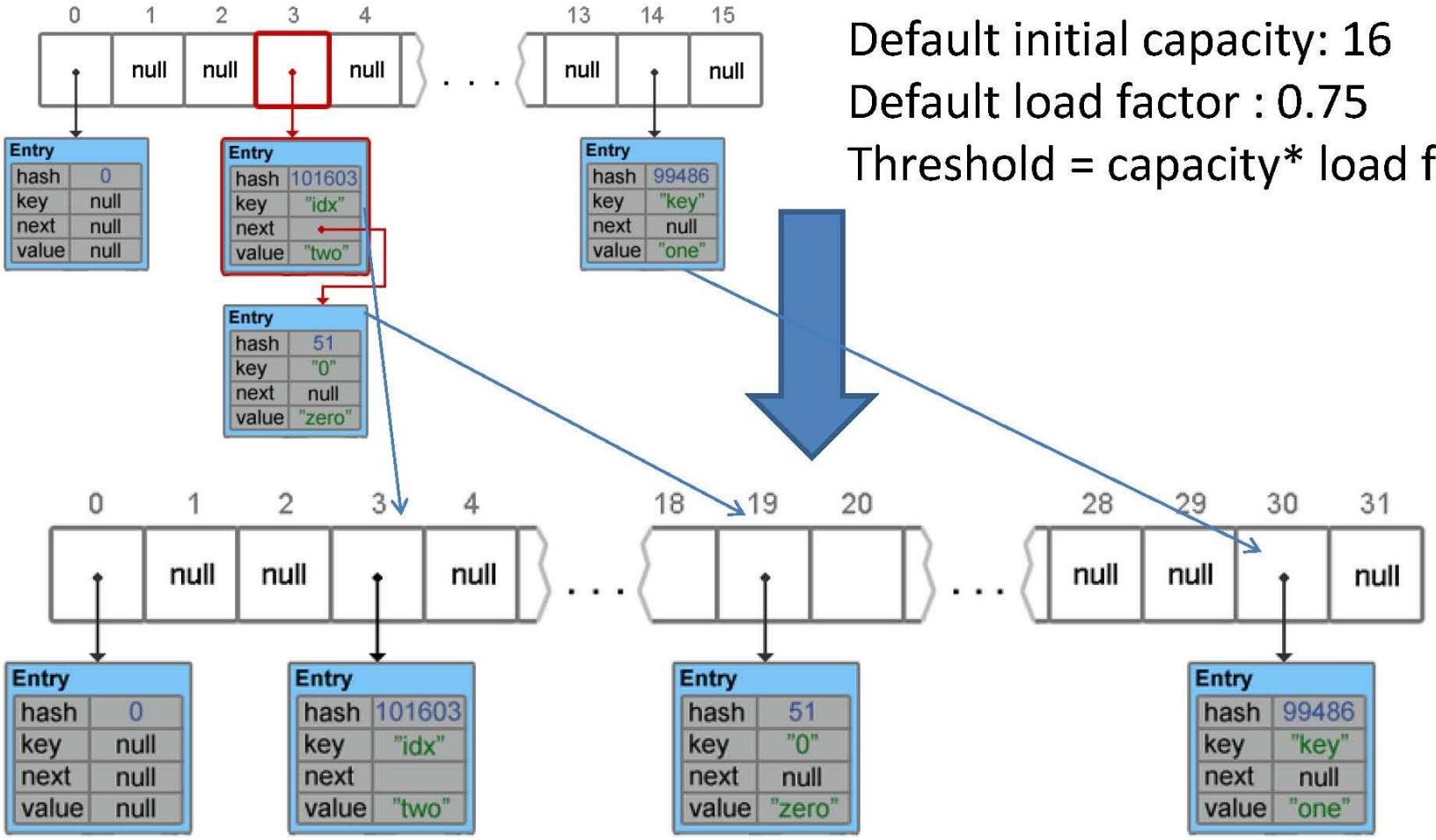


if (newKey != oldKey)

i= 101603 & 15= 3
- collision

else {replace oldValue by newValue}

The Map interface 8/13



Default initial capacity: 16
Default load factor : 0.75
Threshold = capacity* load factor

i=101603 & 31= 3

i=51 & 31= 19

i=98486 & 31= 30

The Map interface 11/13

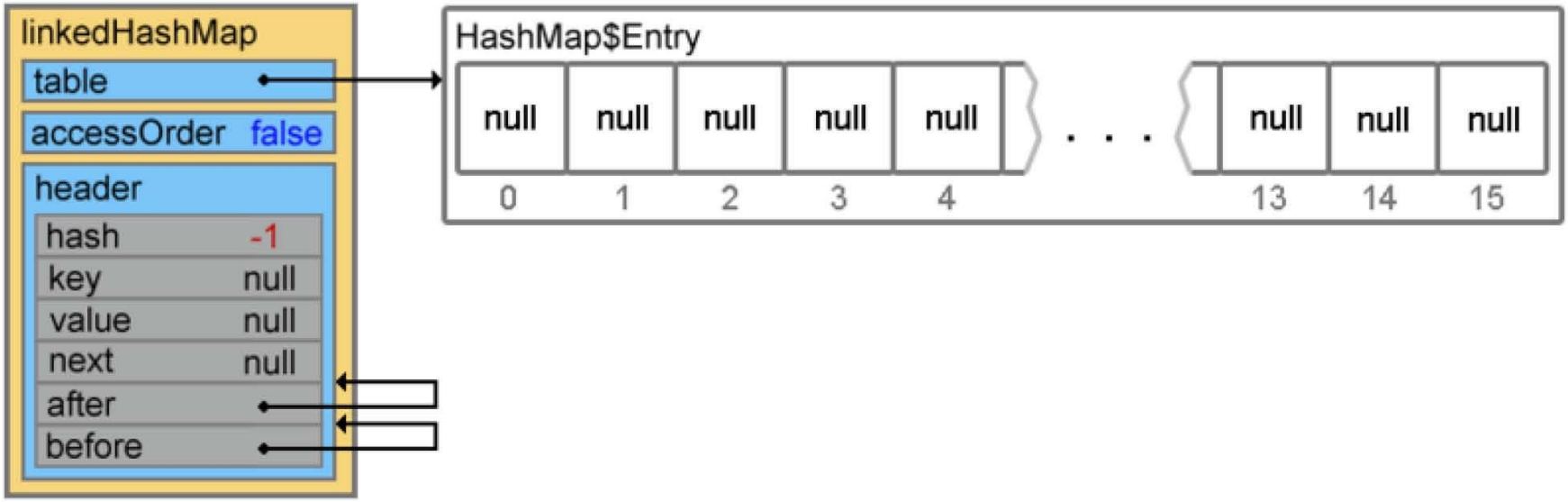
```
1.  LinkedHashMap<String, Integer> hm = new LinkedHashMap<>();
2.  hm.put("one", 1);
3.  hm.put("two", 2);
4.  hm.put("tree", 3);
5.  hm.put("four", 4);
6.  hm.put("five", 5);
7.  Iterator<Map.Entry<String, Integer>> itr1 =
8.  hm.entrySet().iterator();
9.  while (itr1.hasNext()) {
10.     Map.Entry<String,Integer> entry = itr1.next();
11.     System.out.println(entry.getKey() + " = " + entry.getValue());
12. }
```

Console output

```
one = 1
two = 2
tree = 3
four = 4
five = 5
```

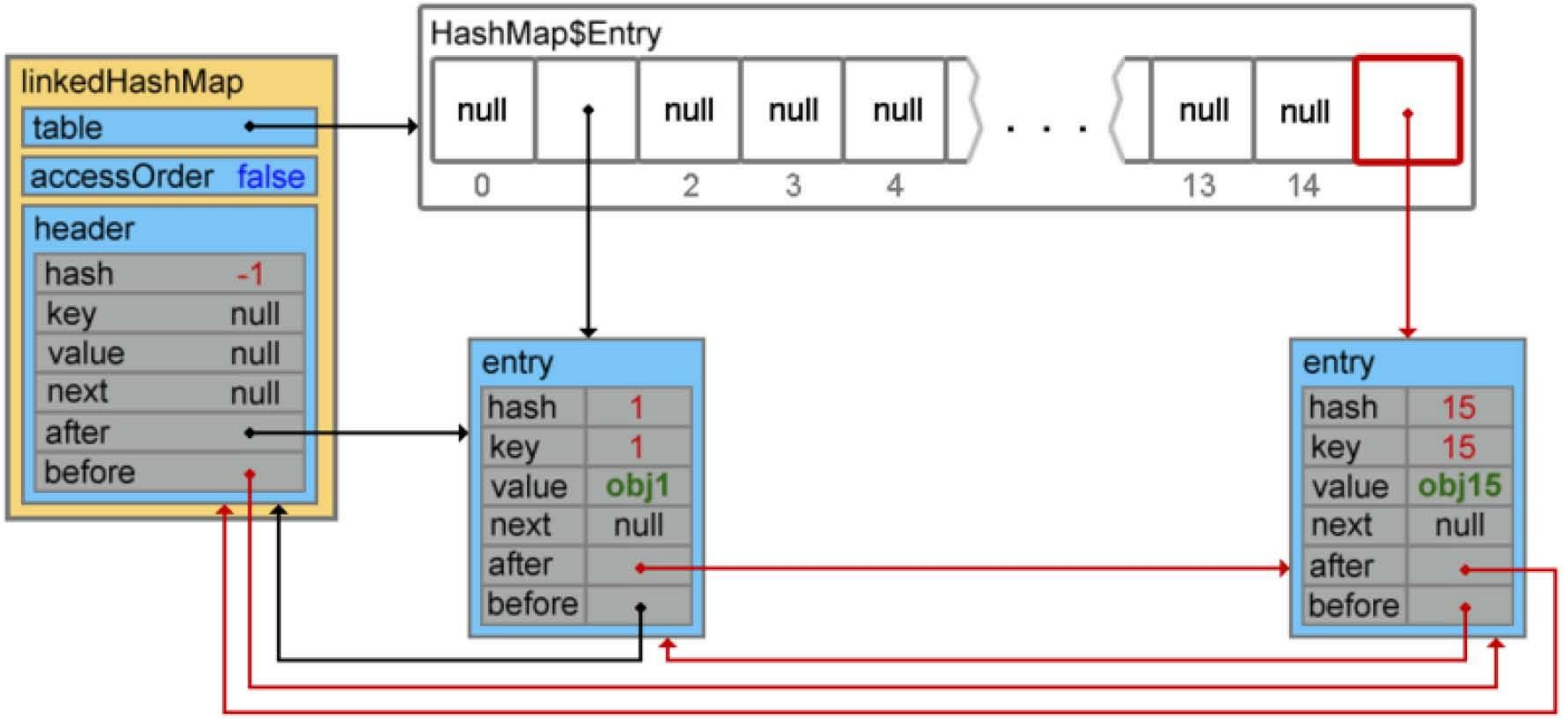
The Map interface 9/13

- LinkedHashMap - defines the iteration ordering, which is normally the order in which keys were inserted
- Map<Integer, String> hm = **new** LinkedHashMap<>();



The Map interface 10/13

1. linkedHashMap.put(1, **"obj1"**);
2. linkedHashMap.put(15, **"obj15"**);



The Map interface 13/13

```
1. Map<String, Integer> hm = new TreeMap<>();
2. hm.put("ee", 5);
3. hm.put("cc", 3);
4. hm.put("aa", 1);
5. hm.put("bb", 2);
6. hm.put("dd", 4);
7. hm.put("ff", 6);
8. System.out.println(hm);
9. int x = hm.get("dd");
10. System.out.println(x);
```

console output

```
{aa=1, bb=2, cc=3, dd=4, ee=5, ff=6}
4
```

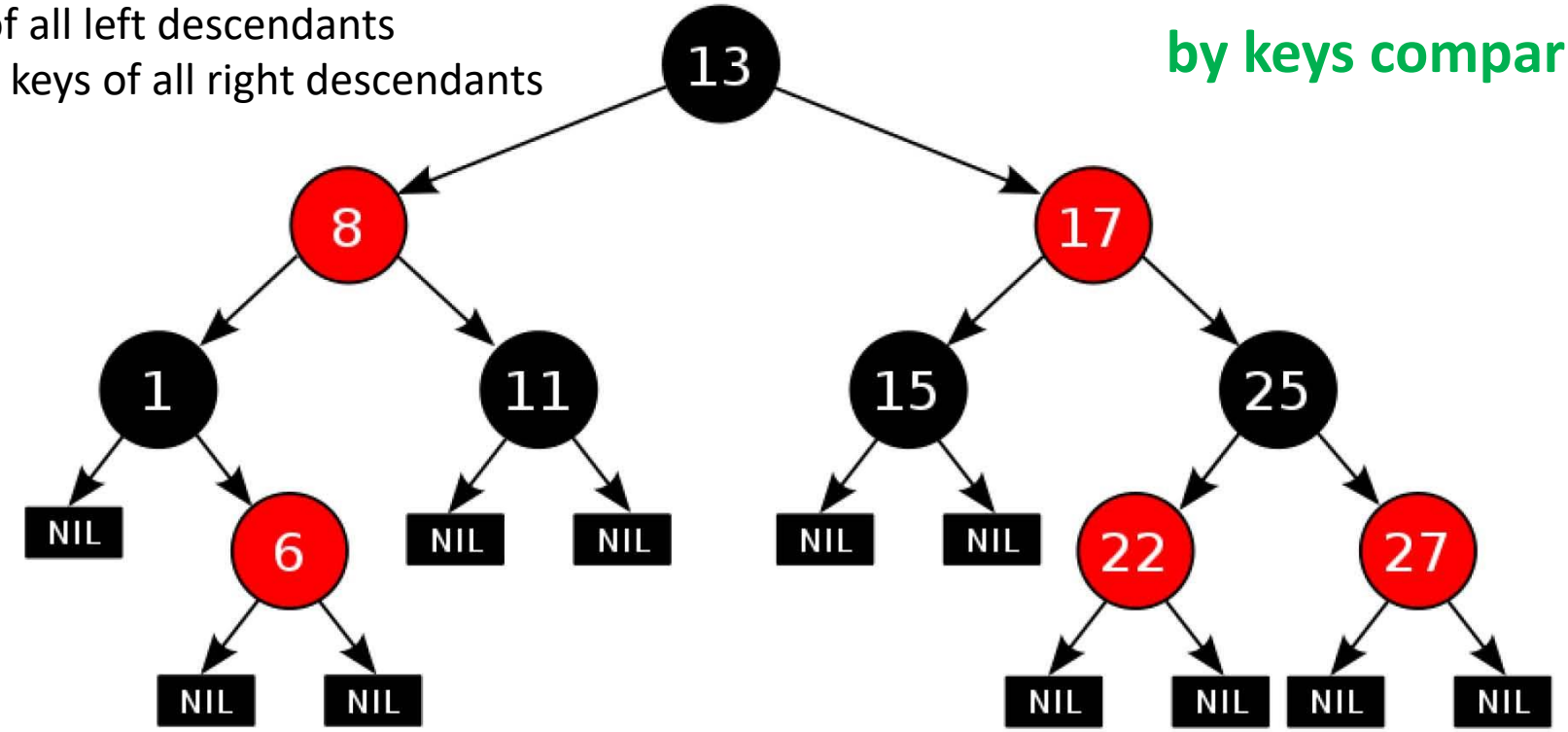
See Edu Project TreeMapRunner...

The Map interface 12/13

- TreeMap - red-black tree based Map implementation.

keys of all left descendants
<= k < keys of all right descendants

by keys comparing



TreeMap class

```
public class TreeMap<K,V> extends AbstractMap<K,V>
   implements NavigableMap<K,V>, Cloneable, java.io.Serializable  {
...
   static final class Entry<K,V> implements Map.Entry<K,V> {
      K key;
      V value;
      Entry<K,V> left;
      Entry<K,V> right;
      Entry<K,V> parent;
      boolean color = BLACK;   //true, RED=false
      ...
   }
....
}
```

Module contents

- Collections
  - Java Collection Framework, Interfaces
  - The Collection Interface
  - The Iterators
  - The List Interface
  - The Queue and Deque Interfaces
  - The Set Interface
  - The Map Interface
  - The Collection Class

The Collections class 1/6

- **The java.util.Collections** class consists exclusively of static methods that operate on or return collections
- It contains polymorphic algorithms that operate on collections, "wrappers", which return a new collection backed by a specified collection
- The methods of this class all throw a NullPointerException if the collections or class objects provided to them are null.

The Collections class 2/6

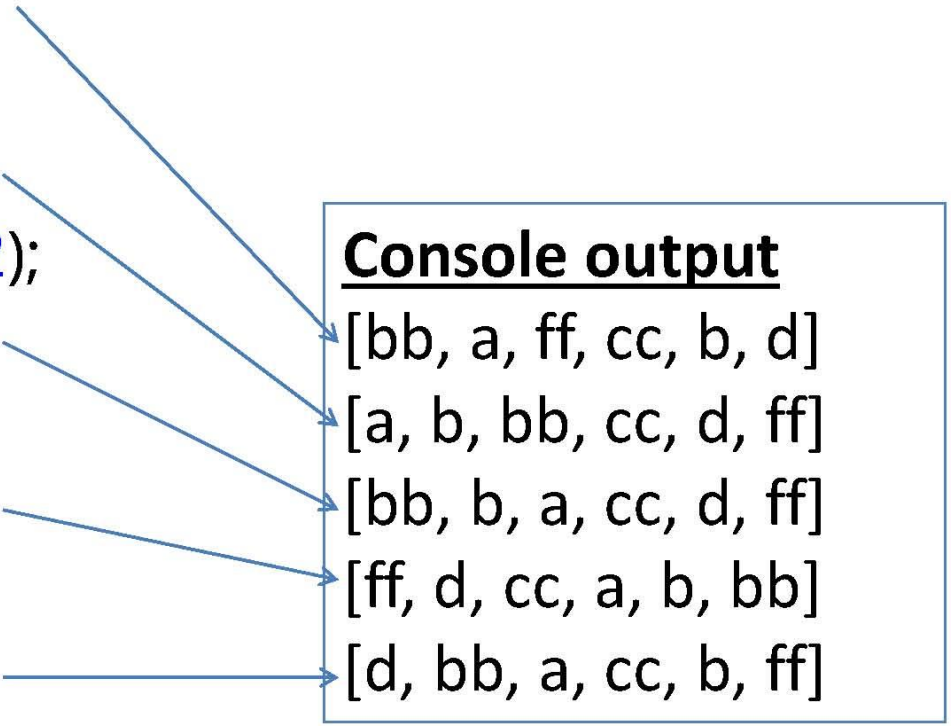
- Adds all of the specified elements to the specified collection
- *addAll(Collection<? super T> c, T... elements)*

- Sorts the specified list into ascending order
- sort(List<T> list)
- sort(List<T> list, Comparator<? super T> c)

- Swaps the elements at the specified positions in the specified list
- swap(List<?> list, int i, int j)

The Collections class 3/6

```
1. List<String> list = new ArrayList<String>();
2. Collections.addAll(list,"bb","a","ff","cc","b","d");
3. System.out.println(list);
4. Collections.sort(list);
5. System.out.println(list);
6. Collections.swap(list,0,2);
7. System.out.println(list);
8. Collections.reverse(list);
9. System.out.println(list);
10. Collections.shuffle(list);
11. System.out.println(list);
```



The Collections class 4/6

- Returns an unmodifiable view of the specified collection

- unmodifiableCollection(Collection<? extends T> c)
- unmodifiableList(List<? extends T> list)
- unmodifiableMap(Map<? extends K,? extends V> m)
- unmodifiableSet(Set<? extends T> s)
- unmodifiableSortedMap(SortedMap<K,? extends V> m)
- unmodifiableSortedSet(SortedSet<T> s)

The Collections class 5/6

```
1. List<String> myList = new ArrayList<String>();
2. myList.add("aaa");
3. myList.add("bbbb");
4. myList.add("ccccc");
5. List<String> readOnlyList =
6.         Collections.unmodifiableList(myList);
7. readOnlyList.add("fff");
```

```
Exception in thread "main" java.lang.UnsupportedOperationException
        at java.util.Collections$UnmodifiableCollection.add
```

The Collections class 6/6

- Returns a synchronized (thread-safe) collection backed by the specified collection

- synchronizedCollection(Collection<T> c)
- synchronizedList(List<T> list)
- synchronizedMap(Map<K,V> m)
- synchronizedSet(Set<T> s)
- synchronizedSortedMap(SortedMap<K,V> m)
- synchronizedSortedSet(SortedSet<T> s)

See Edu Project CollectionsRunner...