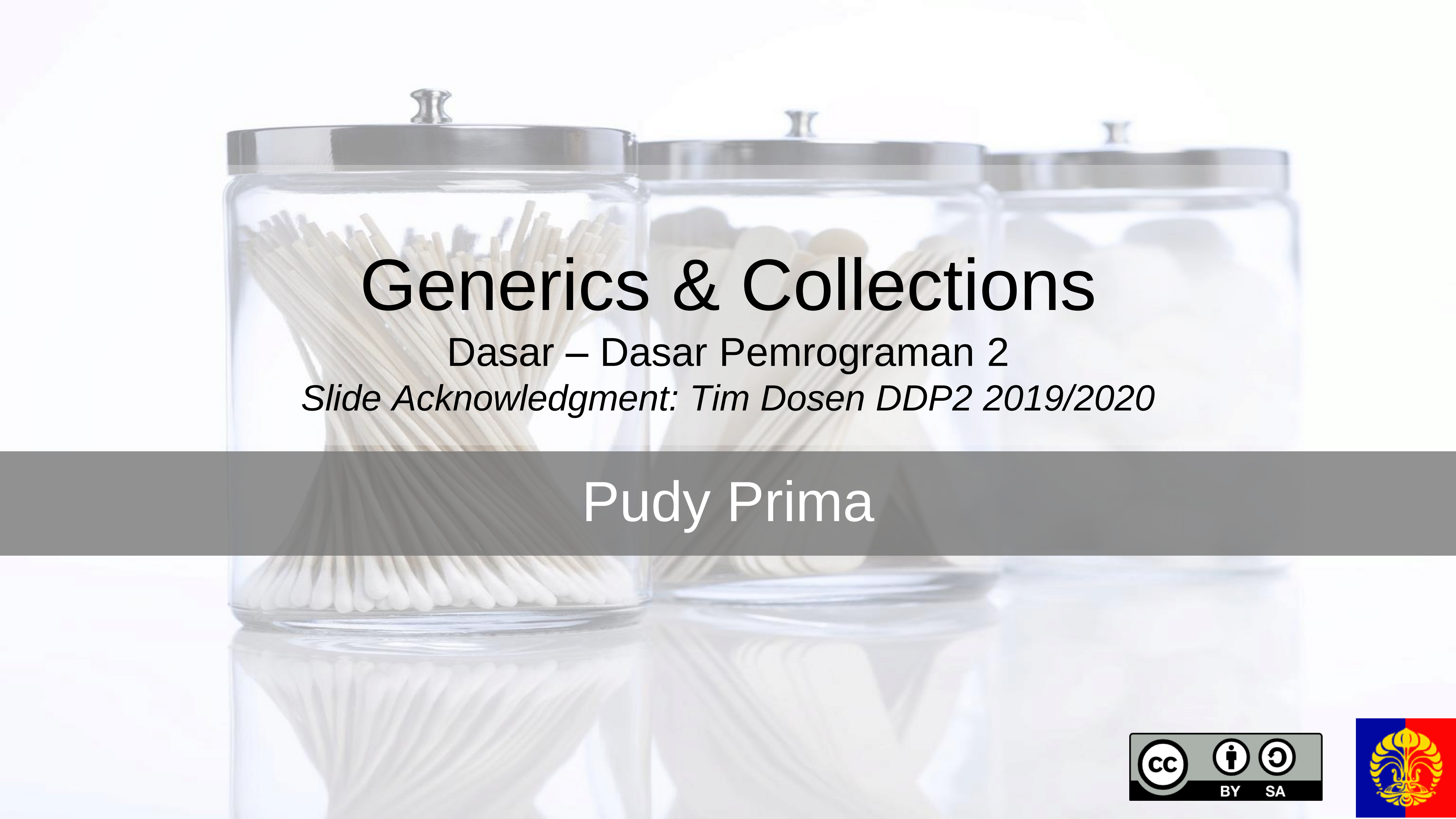


Generics & Collections

Dasar – Dasar Pemrograman 2

Slide Acknowledgment: Tim Dosen DDP2 2019/2020

Pudy Prima

Generics

Generics at a Glance

- The capability to parameterize types.
- You can define a class or a method with generic types
- Later the types can be substituted using concrete types by the compiler.
- For example,
    - A generic stack class stores the elements of a generic type.
    - From this generic stack class
        - A stack object for holding strings
        - A stack object for holding numbers.
    - Strings and numbers are concrete types

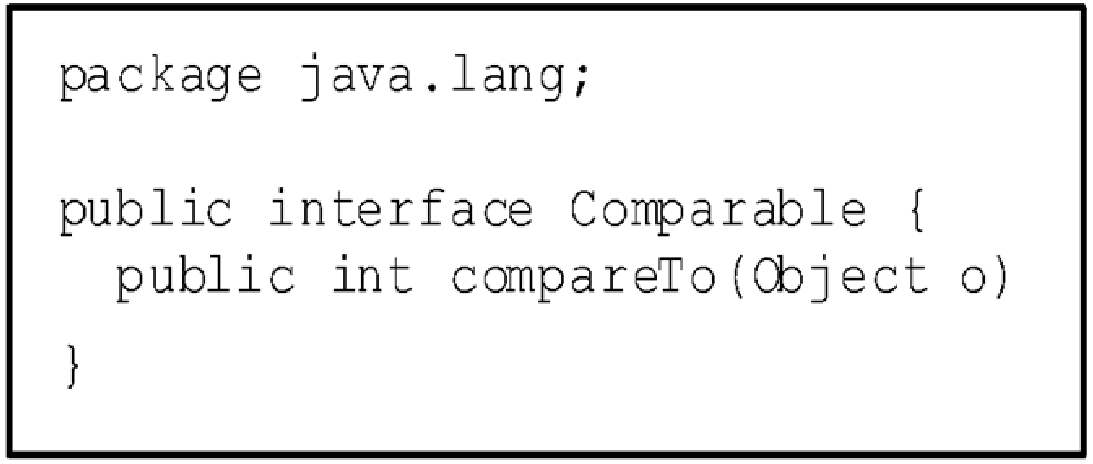
Why?

- Enable errors to be detected at compile time rather than at runtime.
    - Why is this better? Discuss.

- A generic class or method permits you to specify allowable types of objects that the class or method may work with.

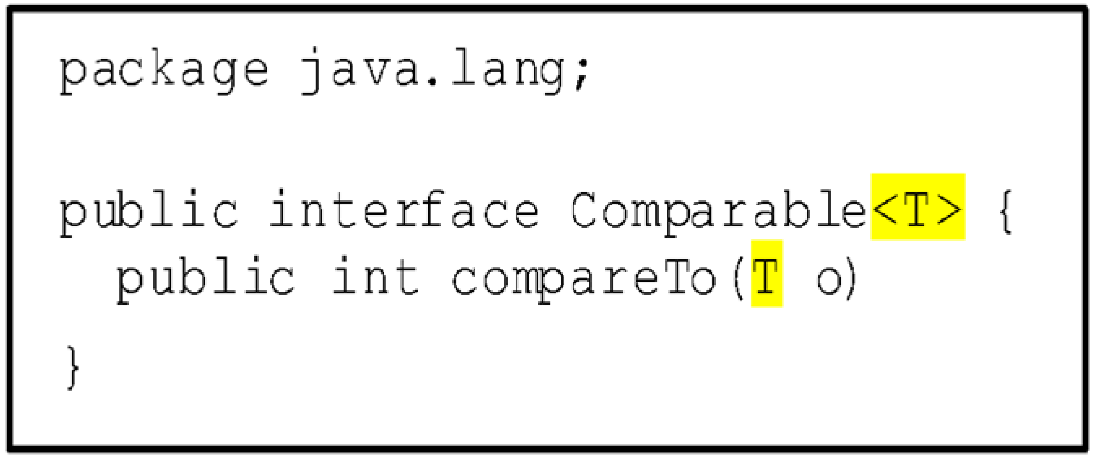
- If you attempt to use the class or method with an incompatible object, a compile error occurs.

Generic Type



```
package java.lang;

public interface Comparable {
  public int compareTo(Object o)
}
```

(a) Prior to JDK 1.5



```
package java.lang;

public interface Comparable<T> {
  public int compareTo(T o)
}
```

(b) JDK 1.5

Generic Instantiation

```
Comparable c = new Date();
System.out.println(c.compareTo("red"));
```

(a) Prior to JDK 1.5

Runtime error

```
Comparable<Date> c = new Date();
System.out.println(c.compareTo("red"));
```

(b) JDK 1.5

Compile error

- Why?

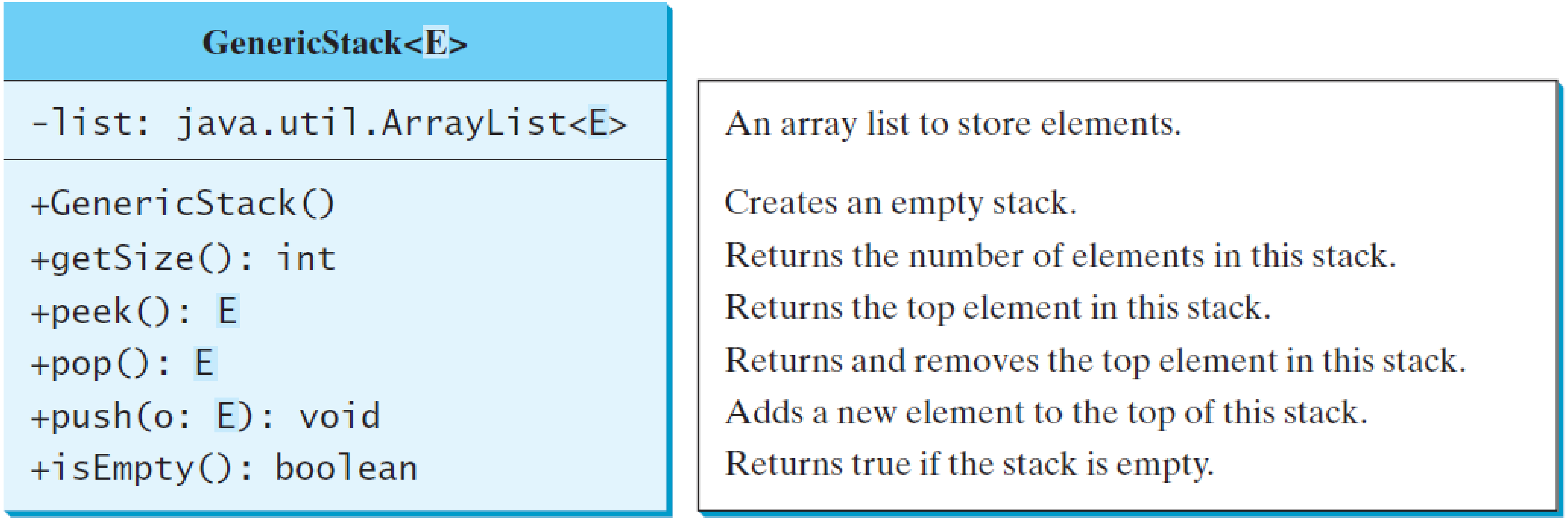
ArrayList is a Generic Class!

java.util.ArrayList<E>	
`+ArrayList()`	Creates an empty list
`+add(o: E) : void`	Appends a new element o at the end of this list.
`+add(index: int, o: E) : void`	Adds a new element o at the specified index in this list.
`+clear(): void`	Removes all the elements from this list.
`+contains(o: Object): boolean`	Returns true if this list contains the element o.
`+get(index: int) : E`	Returns the element from this list at the specified index.
`+indexOf(o: Object) : int`	Returns the index of the first matching element in this list.
`+isEmpty(): boolean`	Returns true if this list contains no elements.
`+lastIndexOf(o: Object) : int`	Returns the index of the last matching element in this list.
`+remove(o: Object): boolean`	Removes the element o from this list.
`+size(): int`	Returns the number of elements in this list.
`+remove(index: int) : boolean`	Removes the element at the specified index.
`+set(index: int, o: E) : E`	Sets the element at the specified index.

No Casting Needed!

```
ArrayList<Double> list = new ArrayList<>();

list.add(5.5); // 5.5 is automatically converted to new Double(5.5)
list.add(3.0); // 3.0 is automatically converted to new Double(3.0)

Double doubleObject = list.get(0); // No casting is needed

double d = list.get(1); // Automatically converted to double
```

Declaring Generic Classes and Interfaces



GenericStack<E>	
-list: java.util.ArrayList<E>	An array list to store elements.
+GenericStack()	Creates an empty stack.
+getSize(): int	Returns the number of elements in this stack.
+peek(): E	Returns the top element in this stack.
+pop(): E	Returns and removes the top element in this stack.
+push(o: E): void	Adds a new element to the top of this stack.
+isEmpty(): boolean	Returns true if the stack is empty.

Cek kode program GenericStack.java dan GenericStackDriver.java

Generic Methods

Cek kode program GenericMethodDemo.java

```
public static <E> void print(E[] list) {
   for (int i = 0; i < list.length; i++)
     System.out.print(list[i] + " ");
   System.out.println();
 }
```

```
public static void print(Object[] list) {
   for (int i = 0; i < list.length; i++)
     System.out.print(list[i] + " ");
   System.out.println();
 }
```

Bounded Generic Type

Cek kode program BoundedTypeDemo.java

```
public static void main(String[] args ) {
    Rectangle rectangle = new Rectangle(2, 2);
    Circle circle = new Circle (2);
    System.out.println("Same area? " + equalArea(rectangle, circle));
}
```

```
public static <E extends GeometricObject> boolean
       equalArea(E object1, E object2) {
    return object1.getArea() == object2.getArea();
}
```

Raw Type and Backward Compatibility

- Raw type:

```
ArrayList list = new ArrayList();
```

- This is *roughly* equivalent to

```
ArrayList<Object> list = new ArrayList<Object>();
```

Raw Type is Unsafe

```
// Max.java: Find a maximum object
public class Max {
  /** Return the maximum between two objects */
  public static Comparable max(Comparable o1, Comparable o2) {
    if (o1.compareTo(o2) > 0)
      return o1;
    else
      return o2;
  }
}
```

- What will happen if we run `Max.max("Welcome", 23);`

Runtime Error

Make it Safe

```
// Max1.java: Find a maximum object
public class Max1 {
  /** Return the maximum between two objects */
  public static <E extends Comparable<E>> E max(E o1, E o2) {
    if (o1.compareTo(o2) > 0)
      return o1;
    else
      return o2;
  }
}
```

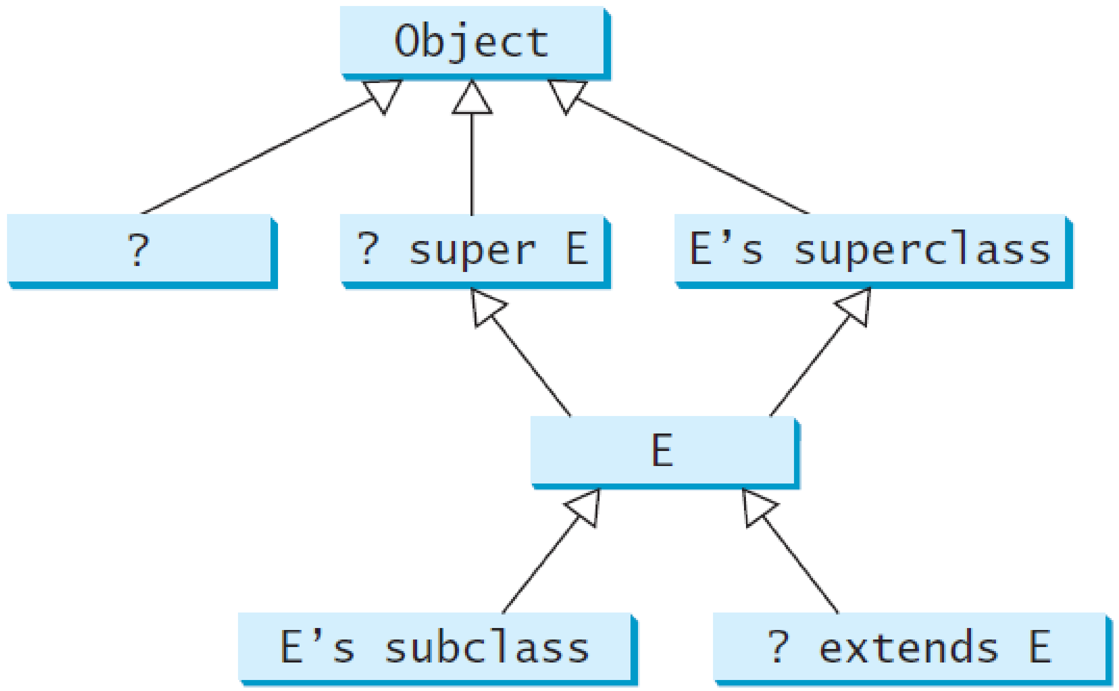
- What will happen if we run `Max.max("Welcome", 23);`

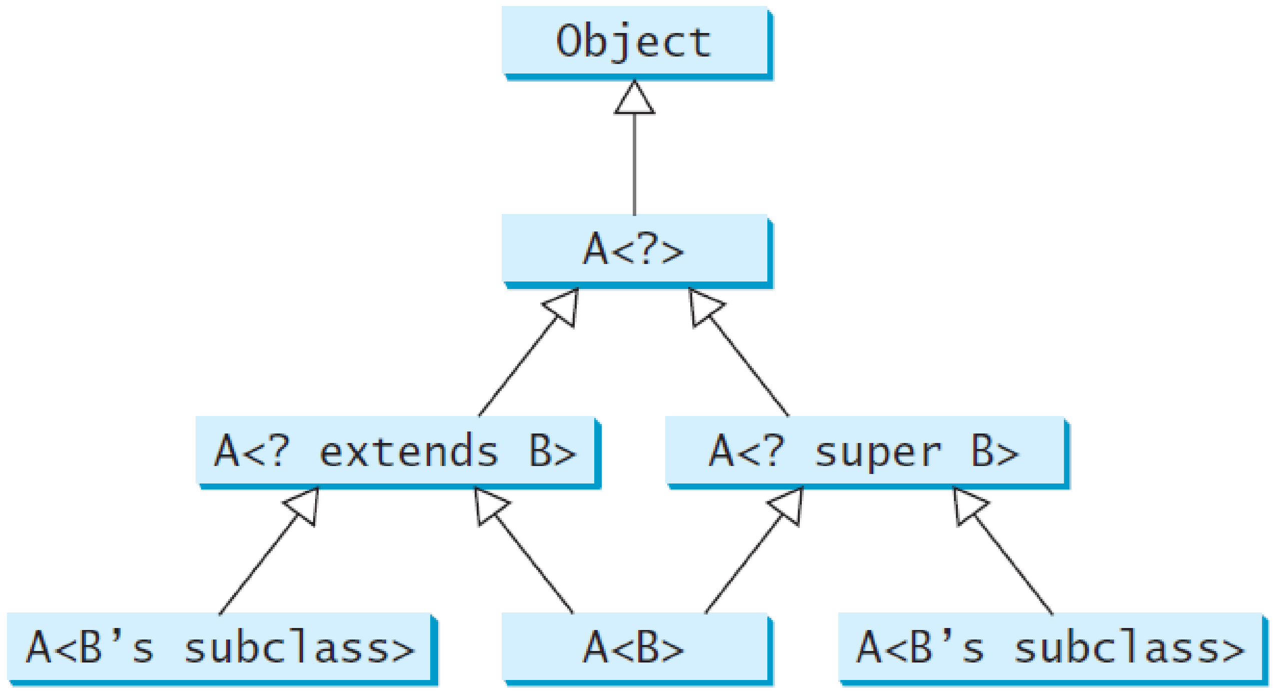
Wildcards

- Wildcard digunakan untuk membatasi class mana saja yang dapat menjadi tipe generic pada suatu generic class maupun generic method tertentu.

Cek kode program WildCardNeedDemo.java, AnyWildCardDemo.java, dan SuperWildCardDemo.java

Generic Types and Wildcard Types





Erasure and Restrictions on Generics

- Generics are implemented using an approach called *type erasure*.
- The compiler uses the generic type to compile the code, but erases it after.
- The generic information is **not available at run time**.
- This approach enables the generic code to be **backward-compatible** with the legacy code that uses raw types.

Compile Time Checking

- The compiler checks whether generics is used correctly for the following code in (a) and translates it into the equivalent code in (b) for runtime use. The code in (b) uses the raw type.

```
ArrayList<String> list = new ArrayList<>();
list.add("Oklahoma");
String state = list.get(0);
```

(a)

```
ArrayList list = new ArrayList();
list.add("Oklahoma");
String state = (String)(list.get(0));
```

(b)

Important Facts

- It is important to note that a generic class is shared by all its instances regardless of its actual generic type.

```
GenericStack<String> stack1 = new GenericStack<>();
GenericStack<Integer> stack2 = new GenericStack<>();
```

- Although GenericStack<String> and GenericStack<Integer> are two types, but there is only one class GenericStack loaded into the JVM.

Restrictions on Generics

- Restriction 1: Cannot Create an Instance of a Generic Type. (i.e., new E()).
- Restriction 2: Generic Array Creation is Not Allowed. (i.e., new E[100]).
- Restriction 3: A Generic Type Parameter of a Class Is Not Allowed in a Static Context.
- Restriction 4: Exception Classes Cannot be Generic.

Studi Kasus
Cek kode program
GenericMatrix.java,
IntegerMatrix.java,
RationalMatrix.java,
TestIntegerMatrix.java, dan
TestRationalMatrix.java

ArrayList

ArrayList

- The ArrayList class can be used to store an unlimited number of objects.
- Stores a collection of **objects** contiguously
- Can adjust the capacity as needed
- Provides methods for managing the array, e.g. add an element, remove an element

ArrayList is a Dynamic-size Array

Declaring an `ArrayList`

```
ArrayList<T> name = new ArrayList<T>();
```

- will store objects of type `T`

Compile error?

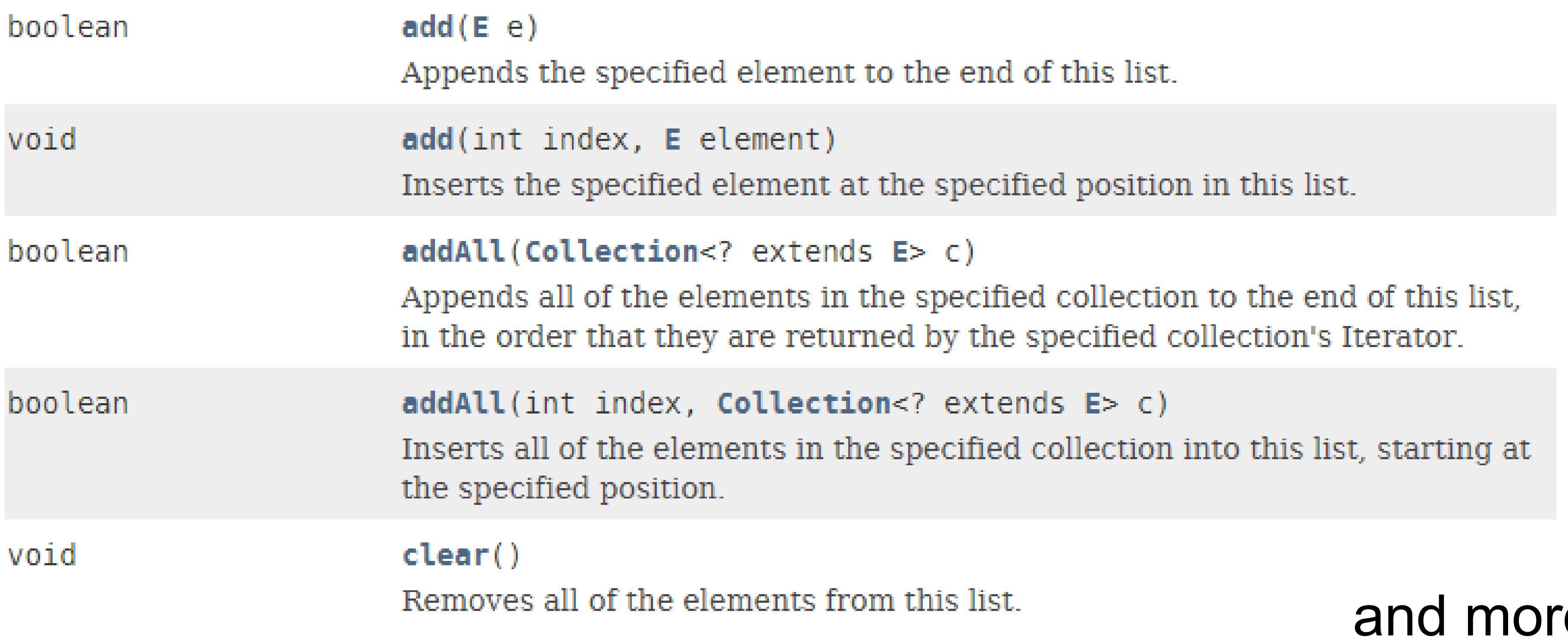
Hint: import the class

Declaring an `ArrayList`

```
import java.util.ArrayList;

public class TryArrayList{

    public static void main(String[] args) {
        ArrayList<String> names = new ArrayList<String>();
        names.add("Bona");
        names.add("Rong-Rong");
    }

}
```

The `ArrayList` Class

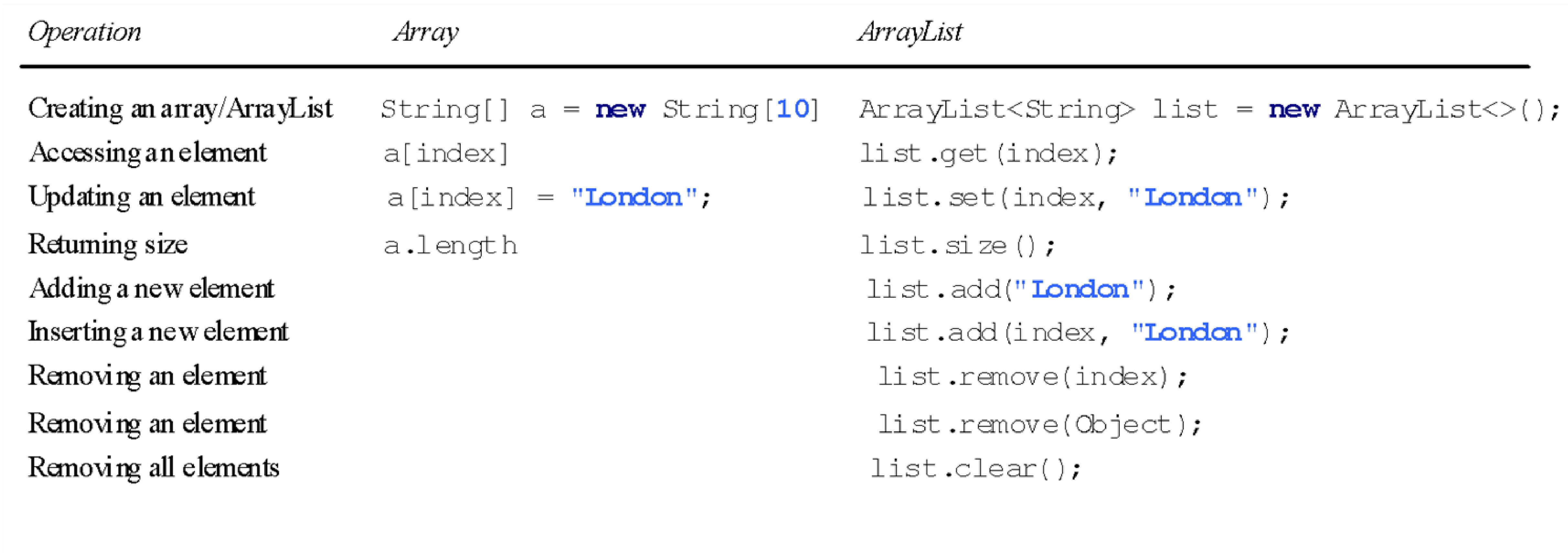
https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html



`boolean`	`add(E e)` Appends the specified element to the end of this list.
`void`	`add(int index, E element)` Inserts the specified element at the specified position in this list.
`boolean`	`addAll(Collection<? extends E> c)` Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's Iterator.
`boolean`	`addAll(int index, Collection<? extends E> c)` Inserts all of the elements in the specified collection into this list, starting at the specified position.
`void`	`clear()` Removes all of the elements from this list.

and more…

Arrays and ArrayList



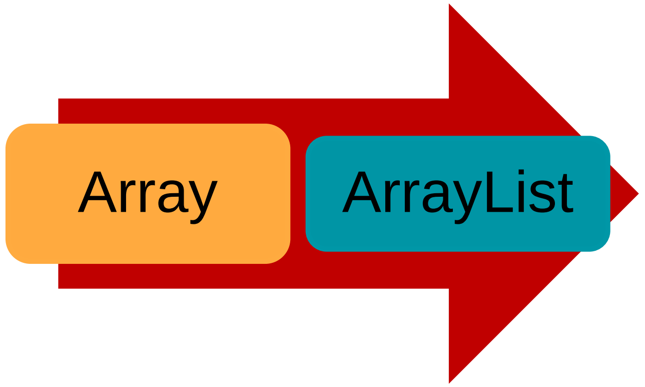
Operation	*Array*	*ArrayList*
Creating an array/ArrayList	`String[] a = new String[10]`	`ArrayList<String> list = new ArrayList<>();`
Accessing an element	`a[index]`	`list.get(index);`
Updating an element	`a[index] = "London";`	`list.set(index, "London");`
Returning size	`a.length`	`list.size();`
Adding a new element		`list.add("London");`
Inserting a new element		`list.add(index, "London");`
Removing an element		`list.remove(index);`
Removing an element		`list.remove(Object);`
Removing all elements		`list.clear();`

ArrayList Example

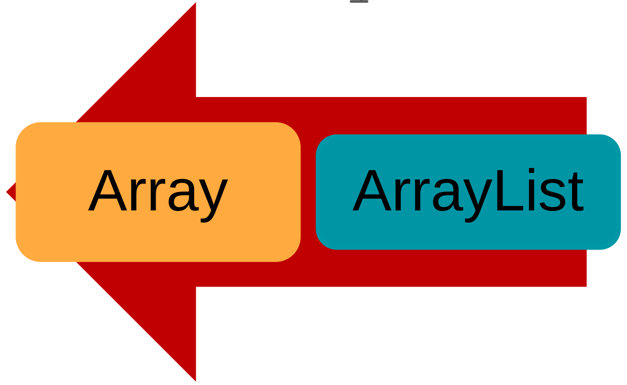
Cek kode program DistinctNumbers.java

```
import java.util.ArrayList;
import java.util.Scanner;

public class DistinctNumbers {
  public static void main(String[] args) {
    ArrayList<Integer> list = new ArrayList<>();
    Scanner input = new Scanner(System.in);
    System.out.print("Enter integers (input ends with 0): ");
    int value;

    do {
      value = input.nextInt(); // Read a value from the input
      if (!list.contains(value) && value != 0)
      list.add(value); // Add the value if it is not in the list
    } while (value != 0);

    // Display the distinct numbers
    for (int i = 0; i < list.size(); i++)
      System.out.print(list.get(i) + " ");
    input.close();
  }
}
```

Array Lists from/to Arrays



```
String[] array = {"red", "green", "blue"};
ArrayList<String> list = new ArrayList<>(Arrays.asList(array));
```



```
String[] array1 = new String[list.size()];
list.toArray(array1);
```

Access & Modify an Element

- Access element:

```
String name = names.get(2);
```

- Watch out for index out-of-bound!

- Modify an element:

```
names.set(2, "Carolyn");
```

Remove an Element

- Use remove() to remove an element at certain cell position

```
names.remove(1);
```

- Again, watch out for index out-of-bound!

Iterations

- Recall in arrays

```
double[] values = ...;
double sum = 0;
for (double element : values) {
    sum = sum + element;
}
```

Iterations in ArrayList

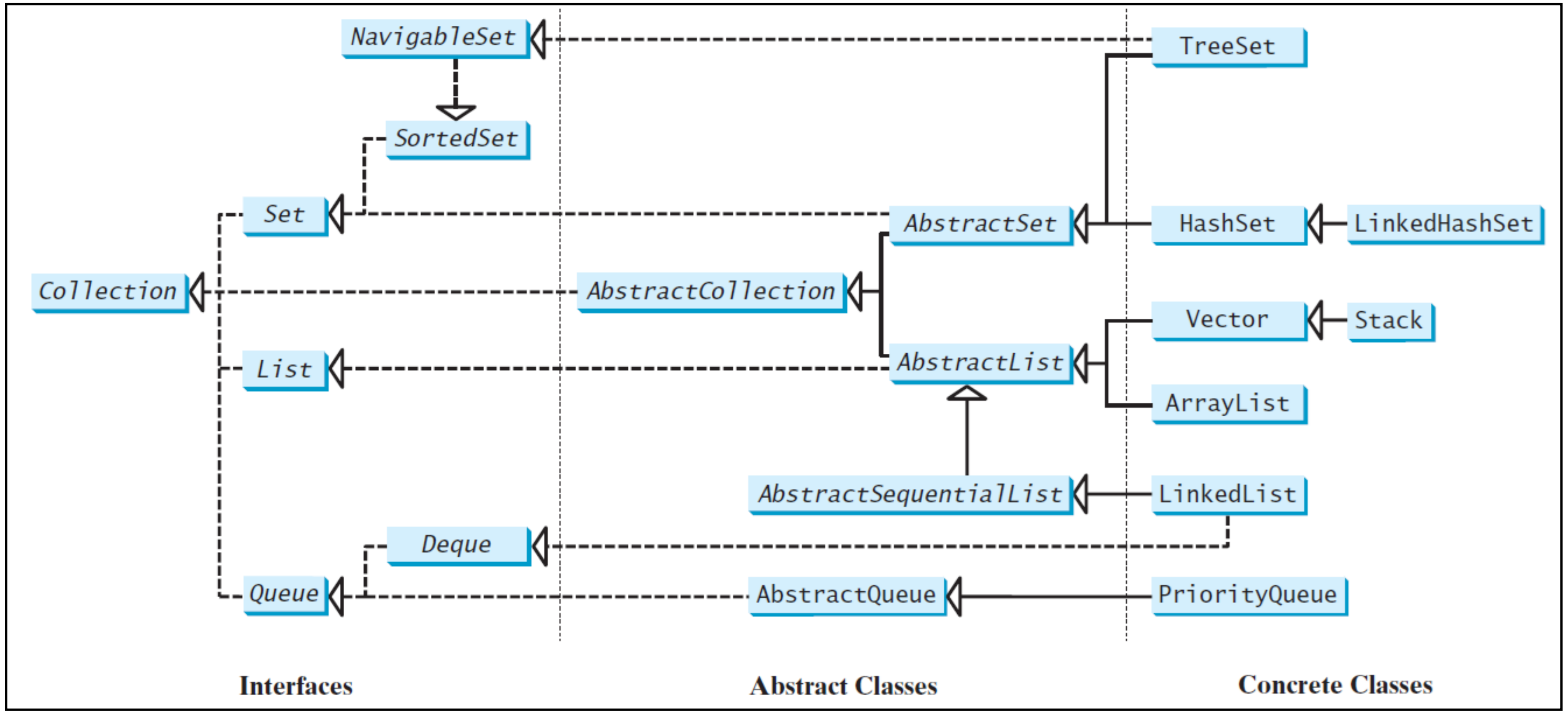
- For-each loop

```
ArrayList<BankAccount> accounts = ...;
double sum = 0;

for (BankAccount account : accounts) {
     sum = sum + account.getBalance();
}
```

Collections

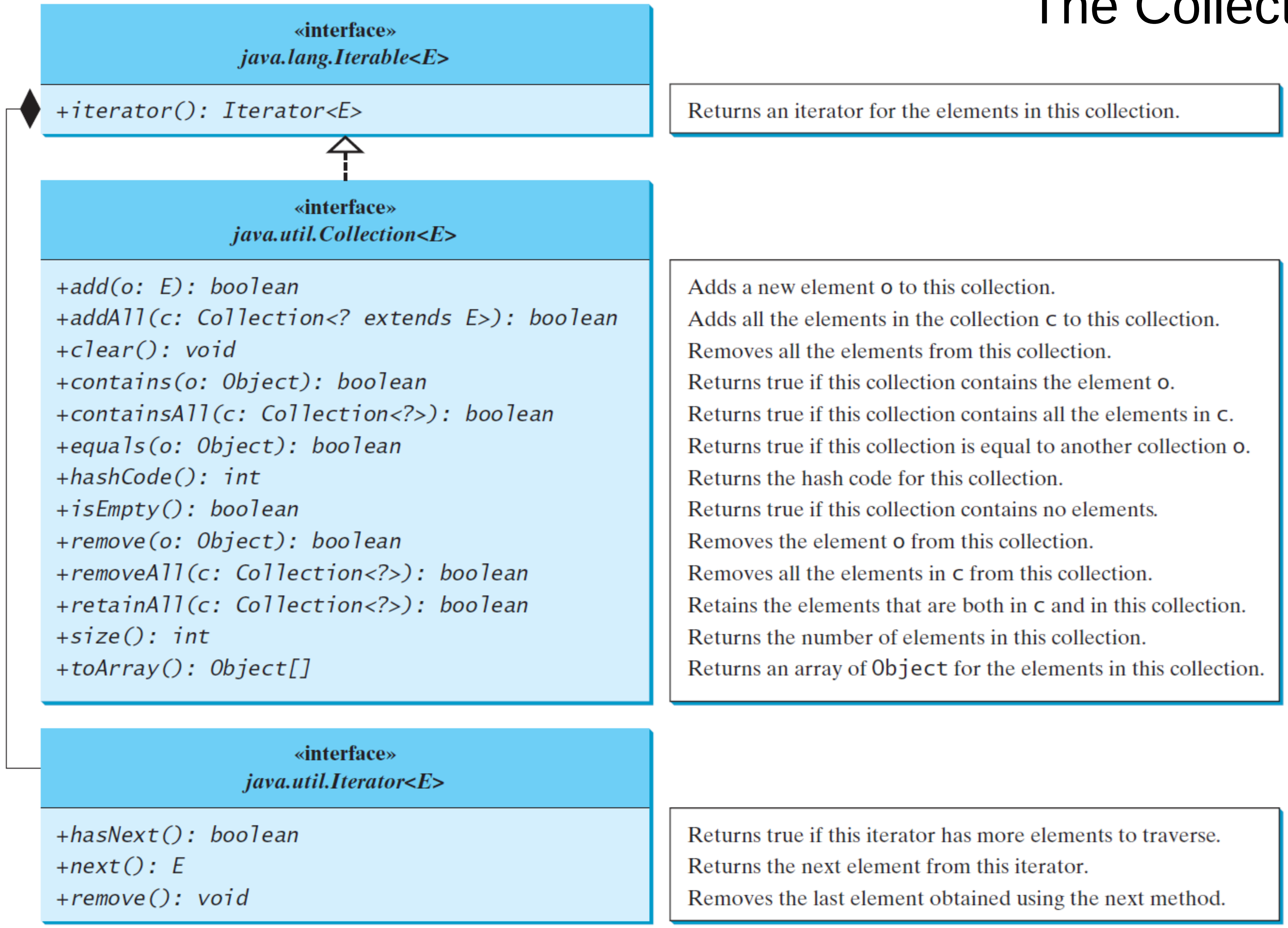
Java Collection Framework hierarchy

A *collection* is a container object that holds a group of objects, often referred to as *elements*. The Java Collections Framework supports three types of collections, named *lists, sets,* and *maps*.



Set and List are subinterfaces of Collection.

The Collection Interface



«interface» java.lang.Iterable<E>	
+iterator(): Iterator<E>	Returns an iterator for the elements in this collection.

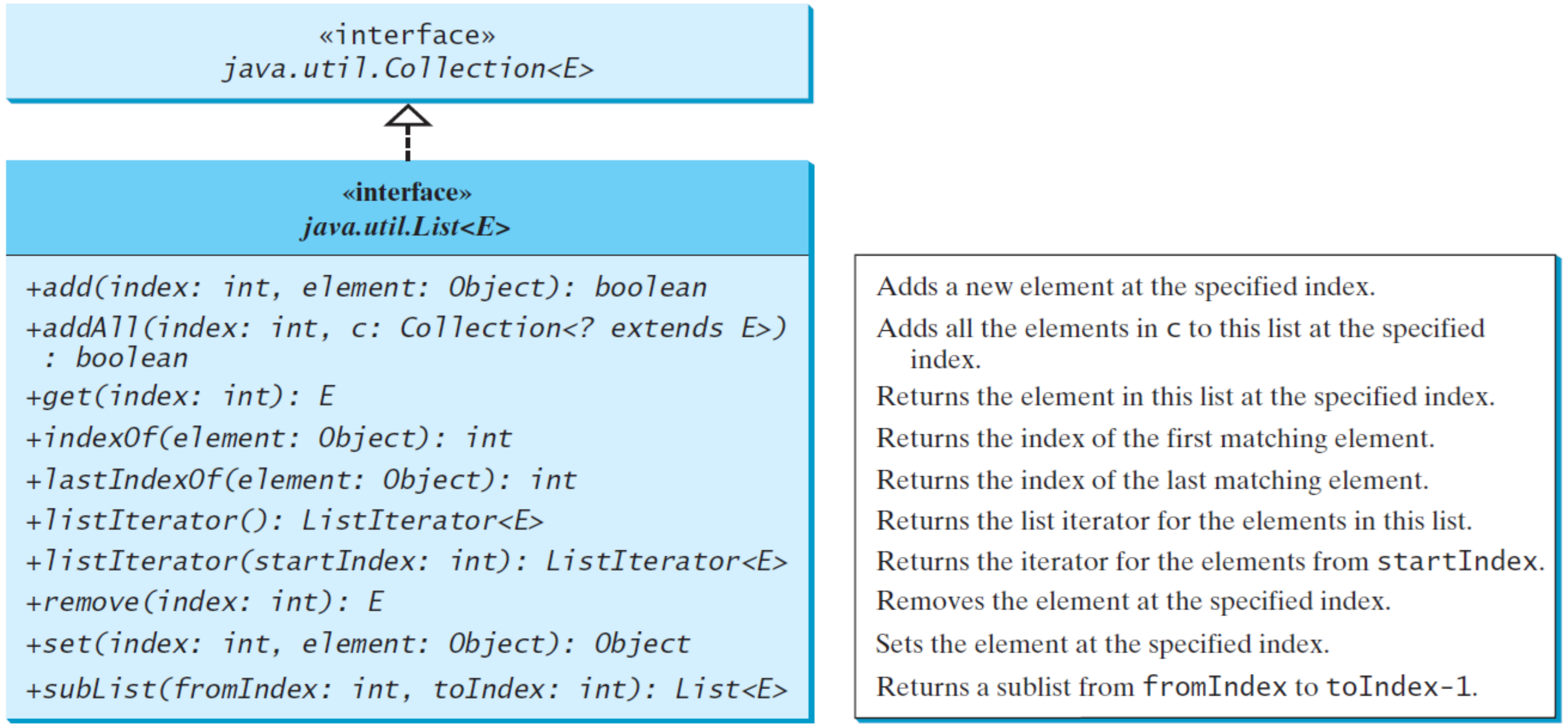
«interface» java.util.Collection<E>	
+add(o: E): boolean	Adds a new element o to this collection.
+addAll(c: Collection<? extends E>): boolean	Adds all the elements in the collection c to this collection.
+clear(): void	Removes all the elements from this collection.
+contains(o: Object): boolean	Returns true if this collection contains the element o.
+containsAll(c: Collection<?>): boolean	Returns true if this collection contains all the elements in c.
+equals(o: Object): boolean	Returns true if this collection is equal to another collection o.
+hashCode(): int	Returns the hash code for this collection.
+isEmpty(): boolean	Returns true if this collection contains no elements.
+remove(o: Object): boolean	Removes the element o from this collection.
+removeAll(c: Collection<?>): boolean	Removes all the elements in c from this collection.
+retainAll(c: Collection<?>): boolean	Retains the elements that are both in c and in this collection.
+size(): int	Returns the number of elements in this collection.
+toArray(): Object[]	Returns an array of Object for the elements in this collection.

«interface» java.util.Iterator<E>	
+hasNext(): boolean	Returns true if this iterator has more elements to traverse.
+next(): E	Returns the next element from this iterator.
+remove(): void	Removes the last element obtained using the next method.

The Collection interface is the root interface for manipulating a collection of objects.

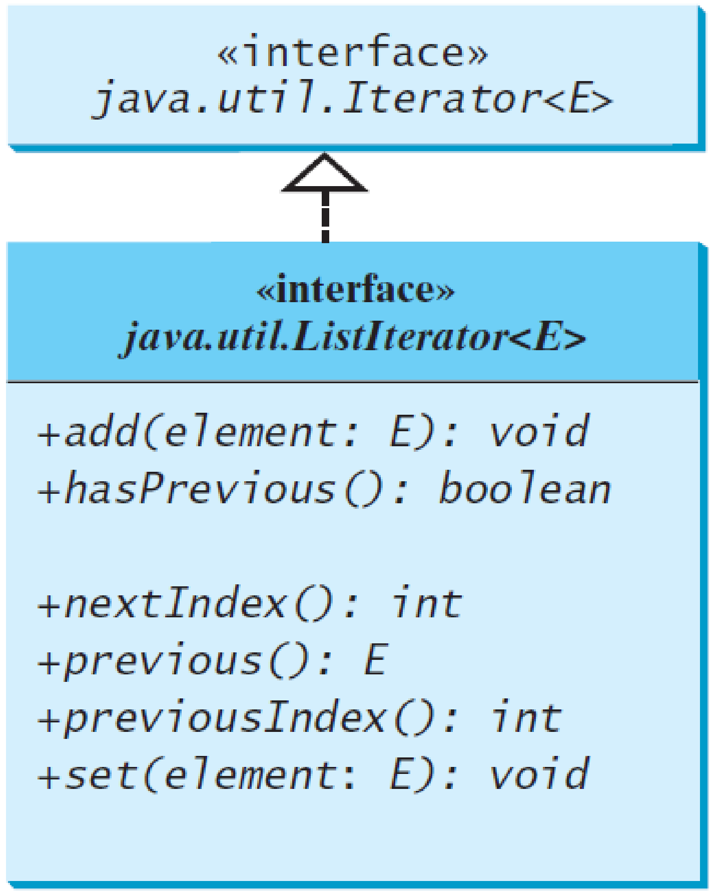
The List Interface

A list stores elements in a sequential order and allows the user to specify where the element is stored. The user can access the elements by index.



+add(index: int, element: Object): boolean	Adds a new element at the specified index.
+addAll(index: int, c: Collection<? extends E>) : boolean	Adds all the elements in c to this list at the specified index.
+get(index: int): E	Returns the element in this list at the specified index.
+indexOf(element: Object): int	Returns the index of the first matching element.
+lastIndexOf(element: Object): int	Returns the index of the last matching element.
+listIterator(): ListIterator<E>	Returns the list iterator for the elements in this list.
+listIterator(startIndex: int): ListIterator<E>	Returns the iterator for the elements from startIndex.
+remove(index: int): E	Removes the element at the specified index.
+set(index: int, element: Object): Object	Sets the element at the specified index.
+subList(fromIndex: int, toIndex: int): List<E>	Returns a sublist from fromIndex to toIndex-1.

The List Iterator



Adds the specified object to the list.
Returns true if this list iterator has more elements when traversing backward.
Returns the index of the next element.
Returns the previous element in this list iterator.
Returns the index of the previous element.
Replaces the last element returned by the previous or next method with the specified element.

ArrayList and LinkedList

The ArrayList class and the LinkedList class are concrete implementations of the List interface. Which of the two classes you use depends on your specific needs.

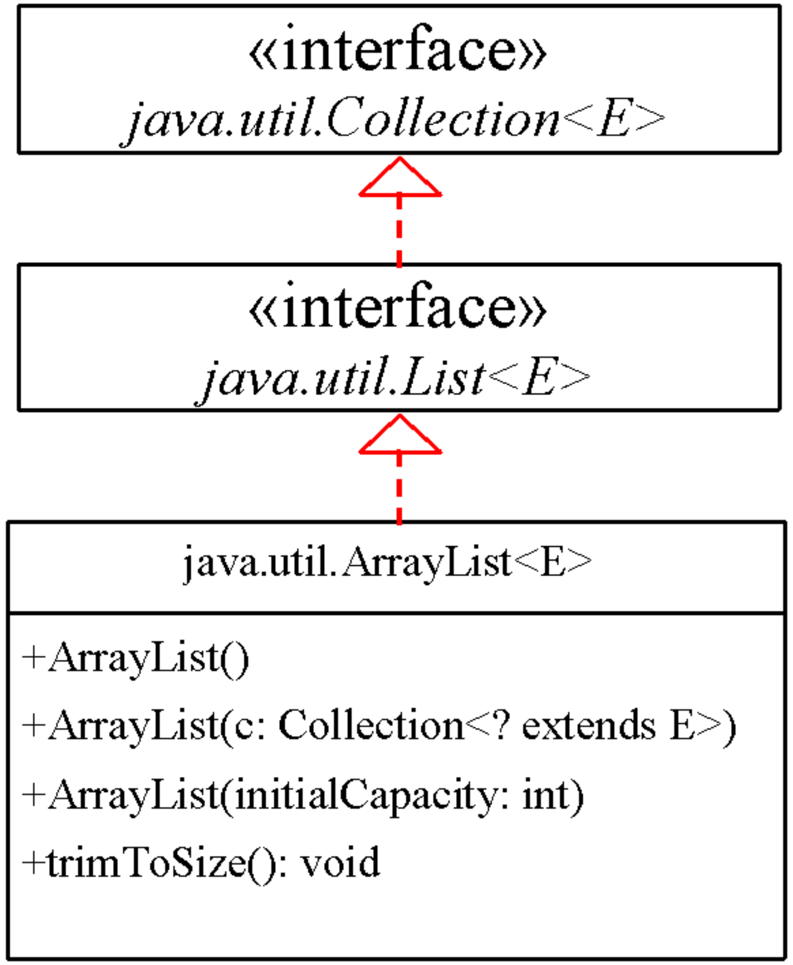
If you need to support random access through an index without inserting or removing elements from any place other than the end, ArrayList offers the most efficient collection.

If, however, your application requires the insertion or deletion of elements from any place in the list, you should choose LinkedList.

A list can grow or shrink dynamically. An array is fixed once it is created. If your application does not require insertion or deletion of elements, the most efficient data structure is the array.

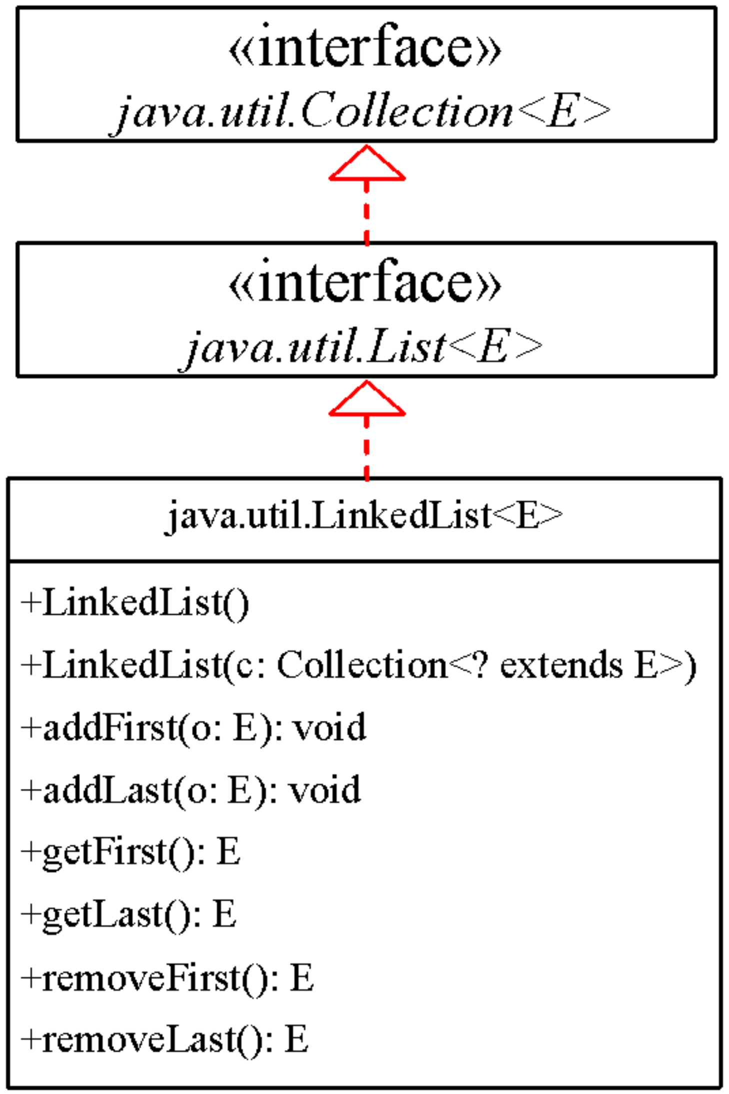
Cek kode program TestArrayAndLinkedList.java

java.util.ArrayList


«interface»
java.util.Collection<E>
«interface»
java.util.List<E>
java.util.ArrayList<E>
+ArrayList()
+ArrayList(c: Collection<? extends E>)
+ArrayList(initialCapacity: int)
+trimToSize(): void

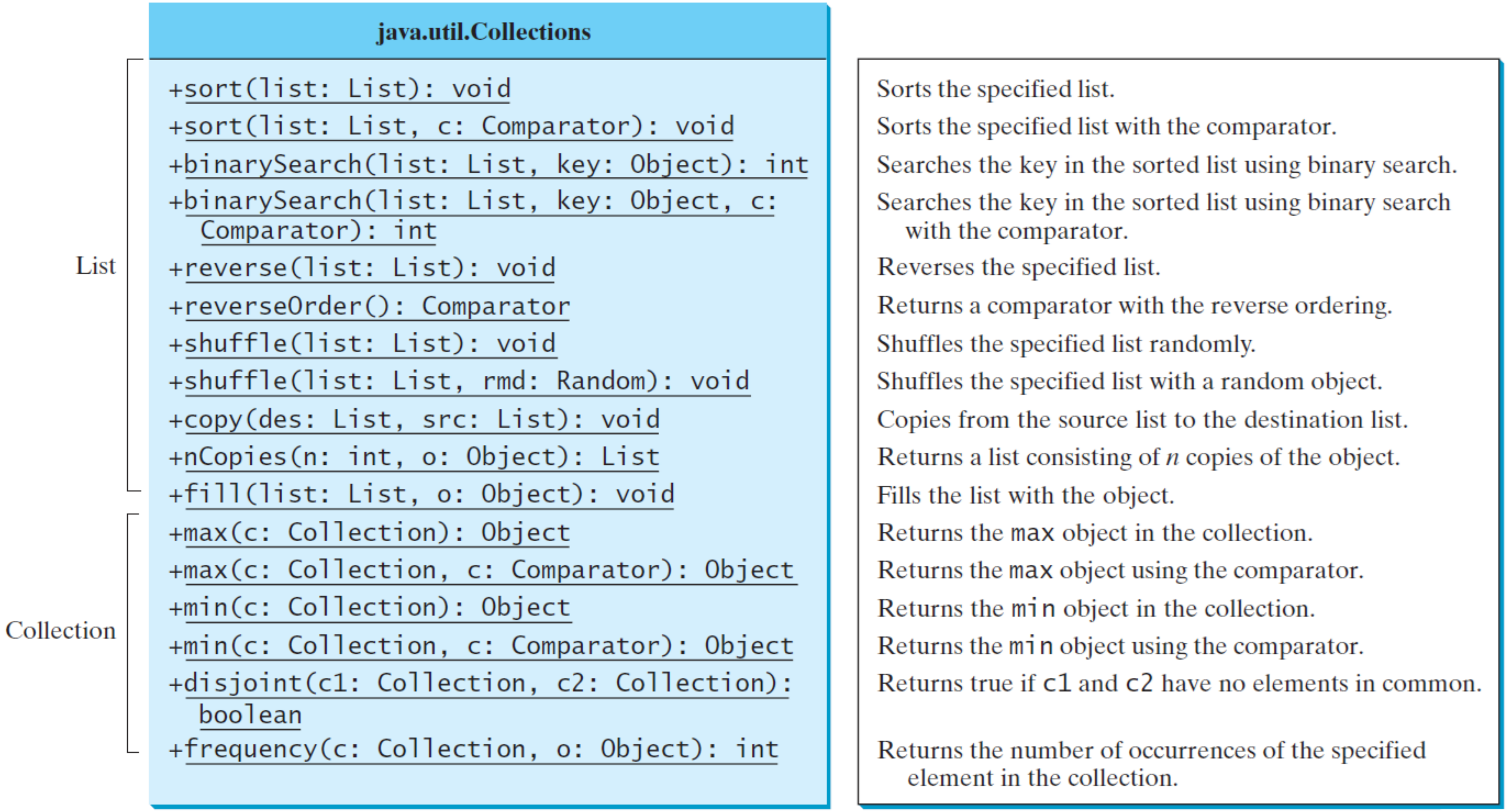
+ArrayList()	Creates an empty list with the default initial capacity.
+ArrayList(c: Collection<? extends E>)	Creates an array list from an existing collection.
+ArrayList(initialCapacity: int)	Creates an empty list with the specified initial capacity.
+trimToSize(): void	Trims the capacity of this ArrayList instance to be the list's current size.

java.util.LinkedList



+LinkedList()	Creates a default empty linked list.
+LinkedList(c: Collection<? extends E>)	Creates a linked list from an existing collection.
+addFirst(o: E): void	Adds the object to the head of this list.
+addLast(o: E): void	Adds the object to the tail of this list.
+getFirst(): E	Returns the first element from this list.
+getLast(): E	Returns the last element from this list.
+removeFirst(): E	Returns and removes the first element from this list.
+removeLast(): E	Returns and removes the last element from this list.

The Collections Class



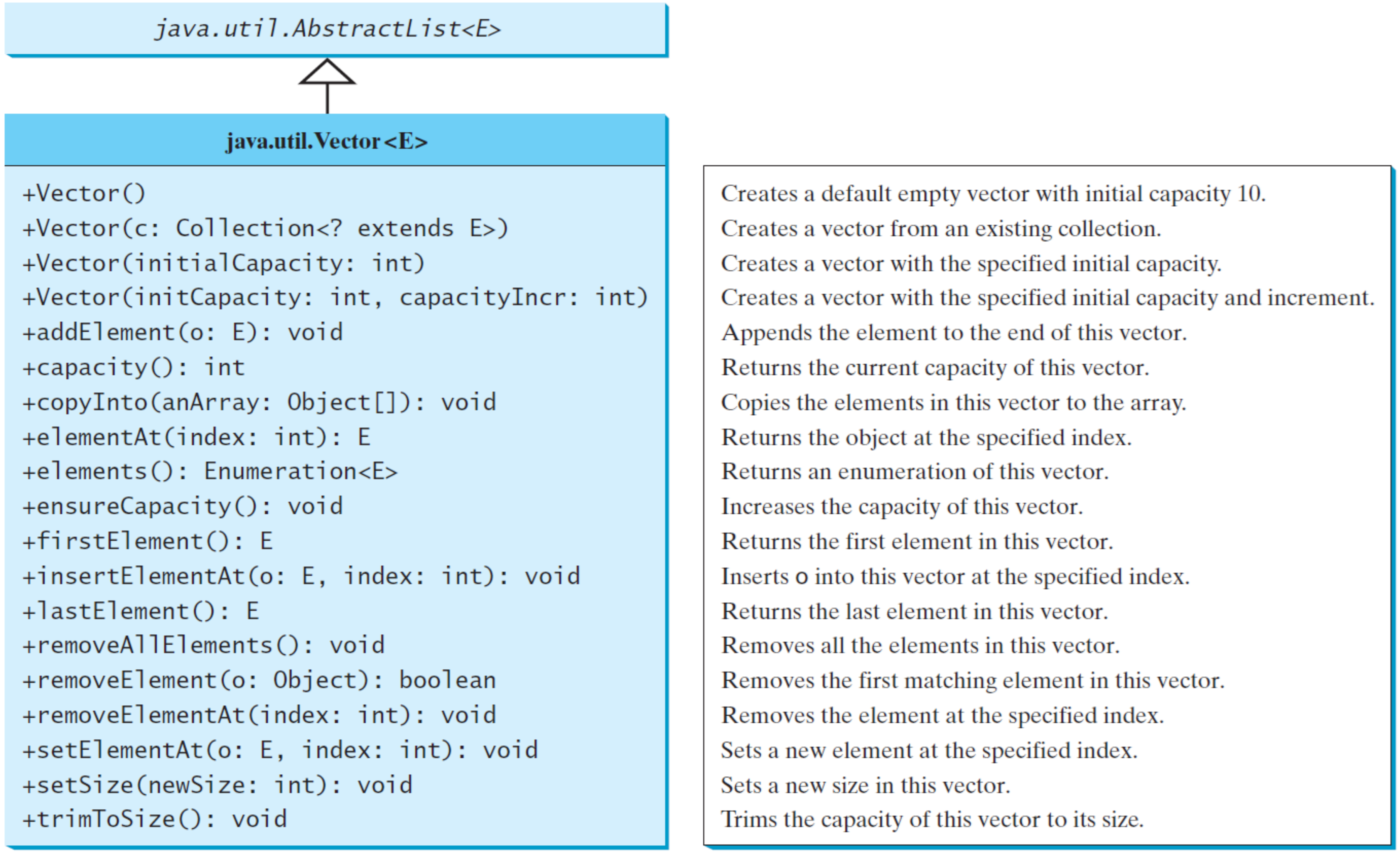
	java.util.Collections	
List	+sort(list: List): void	Sorts the specified list.
	+sort(list: List, c: Comparator): void	Sorts the specified list with the comparator.
	+binarySearch(list: List, key: Object): int	Searches the key in the sorted list using binary search.
	+binarySearch(list: List, key: Object, c: Comparator): int	Searches the key in the sorted list using binary search with the comparator.
	+reverse(list: List): void	Reverses the specified list.
	+reverseOrder(): Comparator	Returns a comparator with the reverse ordering.
	+shuffle(list: List): void	Shuffles the specified list randomly.
	+shuffle(list: List, rmd: Random): void	Shuffles the specified list with a random object.
	+copy(des: List, src: List): void	Copies from the source list to the destination list.
	+nCopies(n: int, o: Object): List	Returns a list consisting of *n* copies of the object.
	+fill(list: List, o: Object): void	Fills the list with the object.
Collection	+max(c: Collection): Object	Returns the `max` object in the collection.
	+max(c: Collection, c: Comparator): Object	Returns the `max` object using the comparator.
	+min(c: Collection): Object	Returns the `min` object in the collection.
	+min(c: Collection, c: Comparator): Object	Returns the `min` object using the comparator.
	+disjoint(c1: Collection, c2: Collection): boolean	Returns true if `c1` and `c2` have no elements in common.
	+frequency(c: Collection, o: Object): int	Returns the number of occurrences of the specified element in the collection.

The Collections class contains various static methods for operating on collections and maps, for creating synchronized collection classes, and for creating read-only collection classes.

The Vector and Stack Classes

The Java Collections Framework was introduced with Java 2. Several data structures were supported prior to Java 2. Among them are the Vector class and the Stack class. These classes were redesigned to fit into the Java Collections Framework, but their old-style methods are retained for compatibility.

The Vector Class



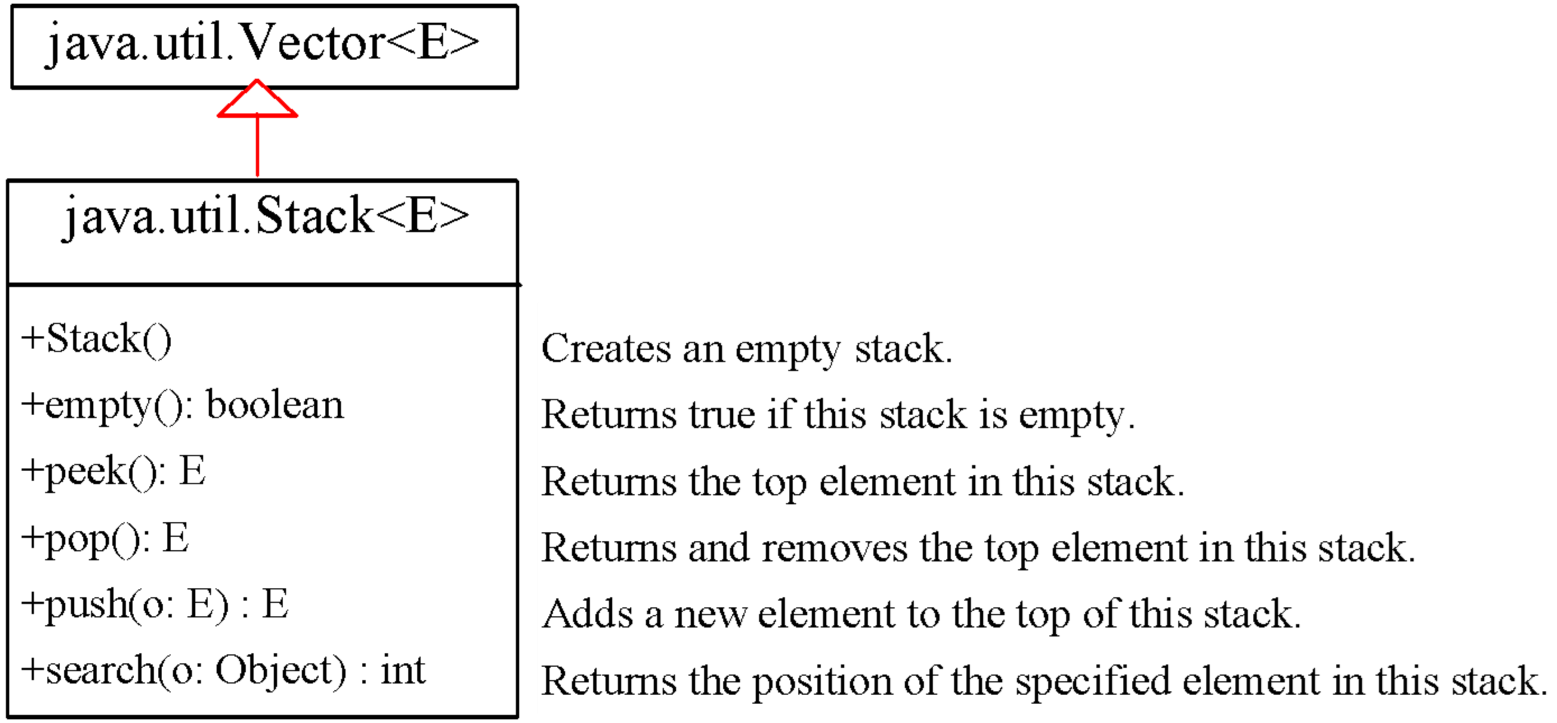
java.util.Vector<E>

+Vector()	Creates a default empty vector with initial capacity 10.
+Vector(c: Collection<? extends E>)	Creates a vector from an existing collection.
+Vector(initialCapacity: int)	Creates a vector with the specified initial capacity.
+Vector(initCapacity: int, capacityIncr: int)	Creates a vector with the specified initial capacity and increment.
+addElement(o: E): void	Appends the element to the end of this vector.
+capacity(): int	Returns the current capacity of this vector.
+copyInto(anArray: Object[]): void	Copies the elements in this vector to the array.
+elementAt(index: int): E	Returns the object at the specified index.
+elements(): Enumeration<E>	Returns an enumeration of this vector.
+ensureCapacity(): void	Increases the capacity of this vector.
+firstElement(): E	Returns the first element in this vector.
+insertElementAt(o: E, index: int): void	Inserts o into this vector at the specified index.
+lastElement(): E	Returns the last element in this vector.
+removeAllElements(): void	Removes all the elements in this vector.
+removeElement(o: Object): boolean	Removes the first matching element in this vector.
+removeElementAt(index: int): void	Removes the element at the specified index.
+setElementAt(o: E, index: int): void	Sets a new element at the specified index.
+setSize(newSize: int): void	Sets a new size in this vector.
+trimToSize(): void	Trims the capacity of this vector to its size.

In Java 2, Vector is the same as ArrayList, except that Vector contains the synchronized methods for accessing and modifying the vector.

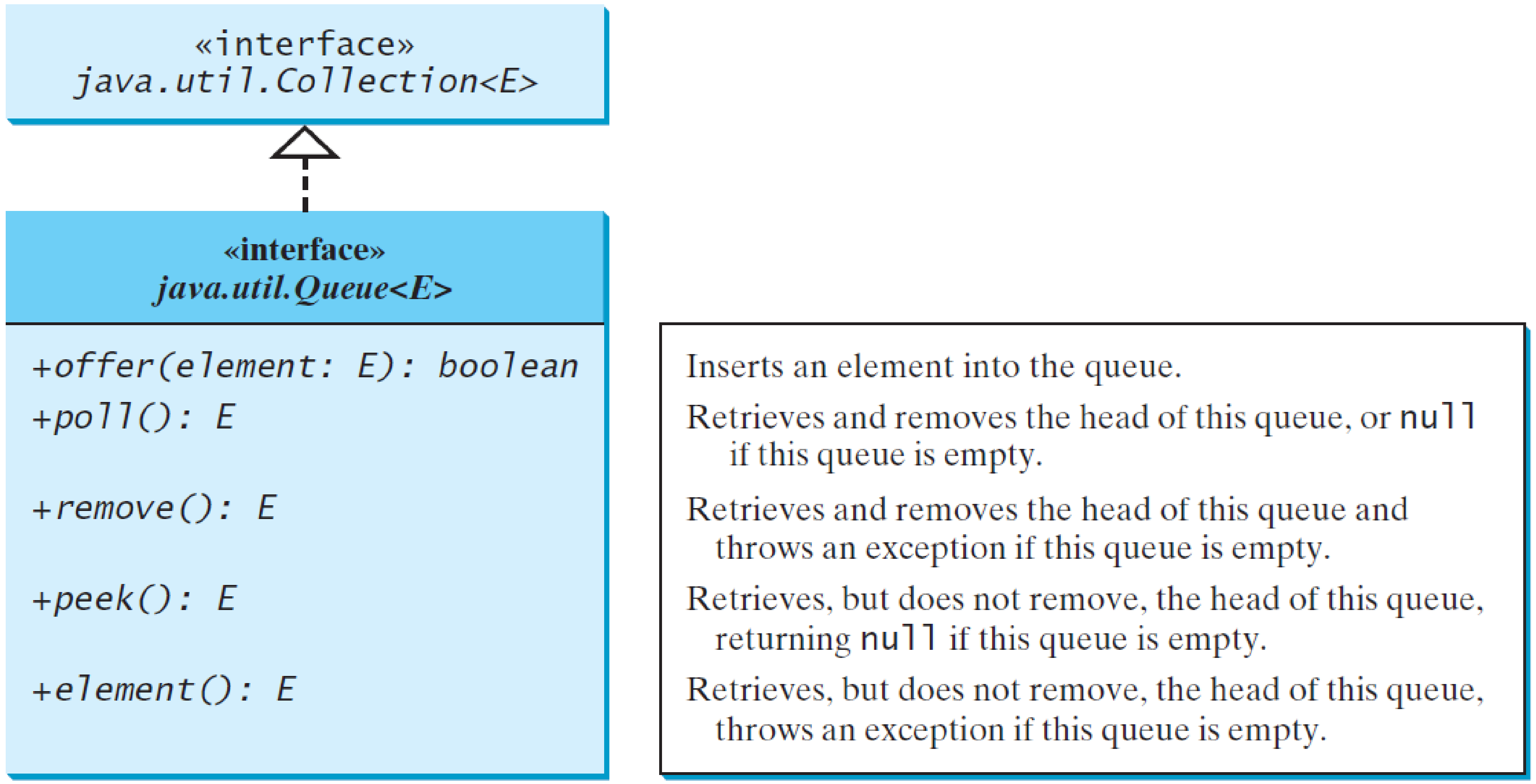
The Stack Class

The Stack class represents a last-in-first-out stack of objects. The elements are accessed only from the top of the stack. You can retrieve, insert, or remove an element from the top of the stack.


java.util.Vector<E>
java.util.Stack<E>
+Stack()
+empty(): boolean
+peek(): E
+pop(): E
+push(o: E) : E
+search(o: Object) : int
Creates an empty stack.
Returns true if this stack is empty.
Returns the top element in this stack.
Returns and removes the top element in this stack.
Adds a new element to the top of this stack.
Returns the position of the specified element in this stack.

Queues and Priority Queues

A queue is a first-in/first-out data structure. Elements are appended to the end of the queue and are removed from the beginning of the queue. In a priority queue, elements are assigned priorities. When accessing elements, the element with the highest priority is removed first.



Inserts an element into the queue.

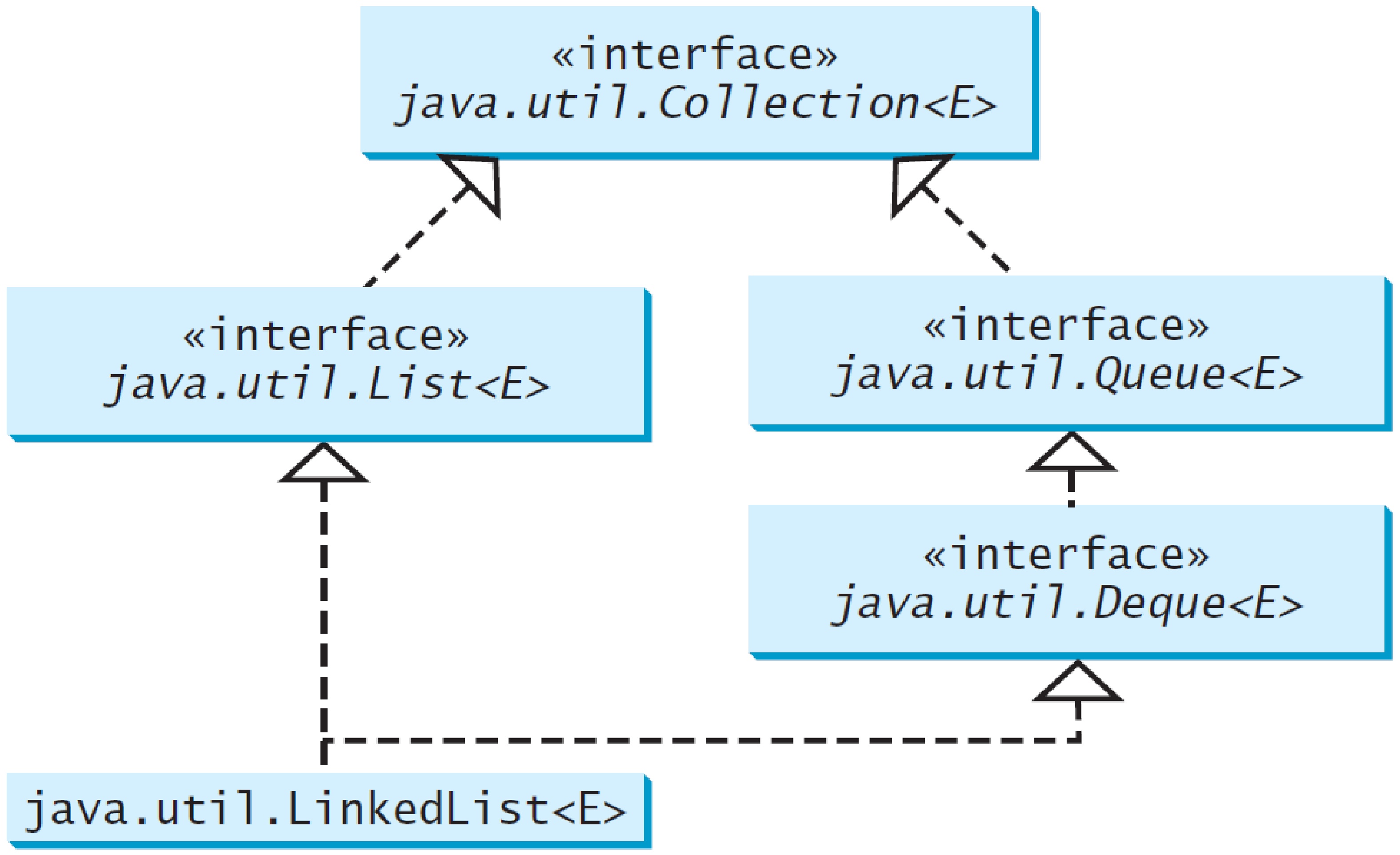
Retrieves and removes the head of this queue, or `null` if this queue is empty.

Retrieves and removes the head of this queue and throws an exception if this queue is empty.

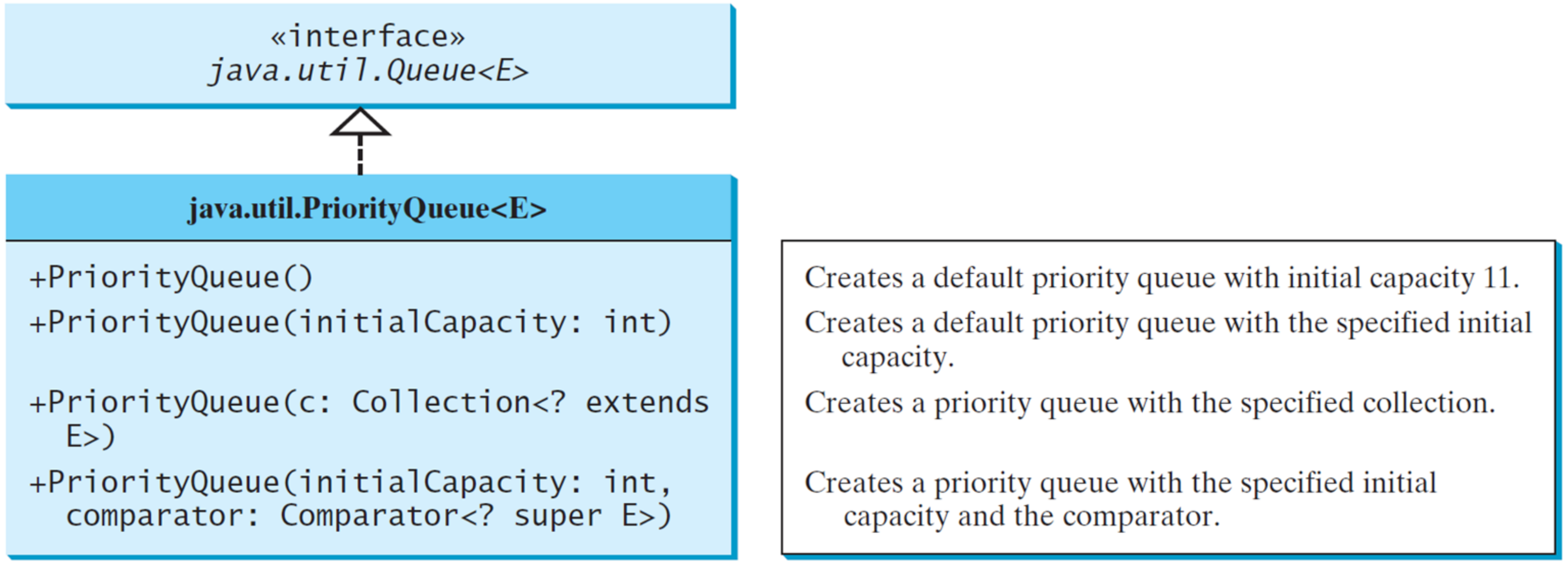
Retrieves, but does not remove, the head of this queue, returning `null` if this queue is empty.

Retrieves, but does not remove, the head of this queue, throws an exception if this queue is empty.

Using LinkedList for Queue



The PriorityQueue Class



«interface»
java.util.Queue<E>
java.util.PriorityQueue<E>
+PriorityQueue()
+PriorityQueue(initialCapacity: int)
+PriorityQueue(c: Collection<? extends E>)
+PriorityQueue(initialCapacity: int, comparator: Comparator<? super E>)
Creates a default priority queue with initial capacity 11.
Creates a default priority queue with the specified initial capacity.
Creates a priority queue with the specified collection.
Creates a priority queue with the specified initial capacity and the comparator.