# Computer Language

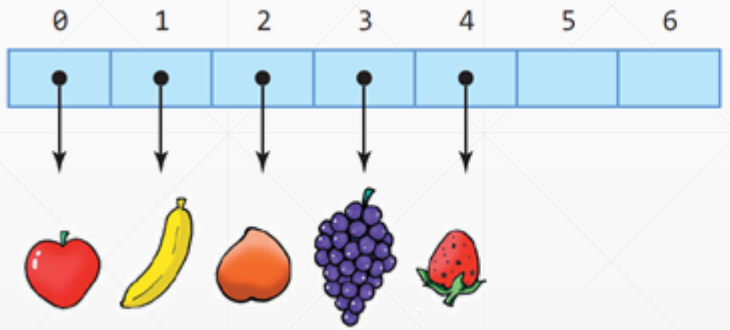
Generic & Collections

# Agenda

- Generic & Collection
- Collections
  - Vector
  - ArrayList
  - HashMap

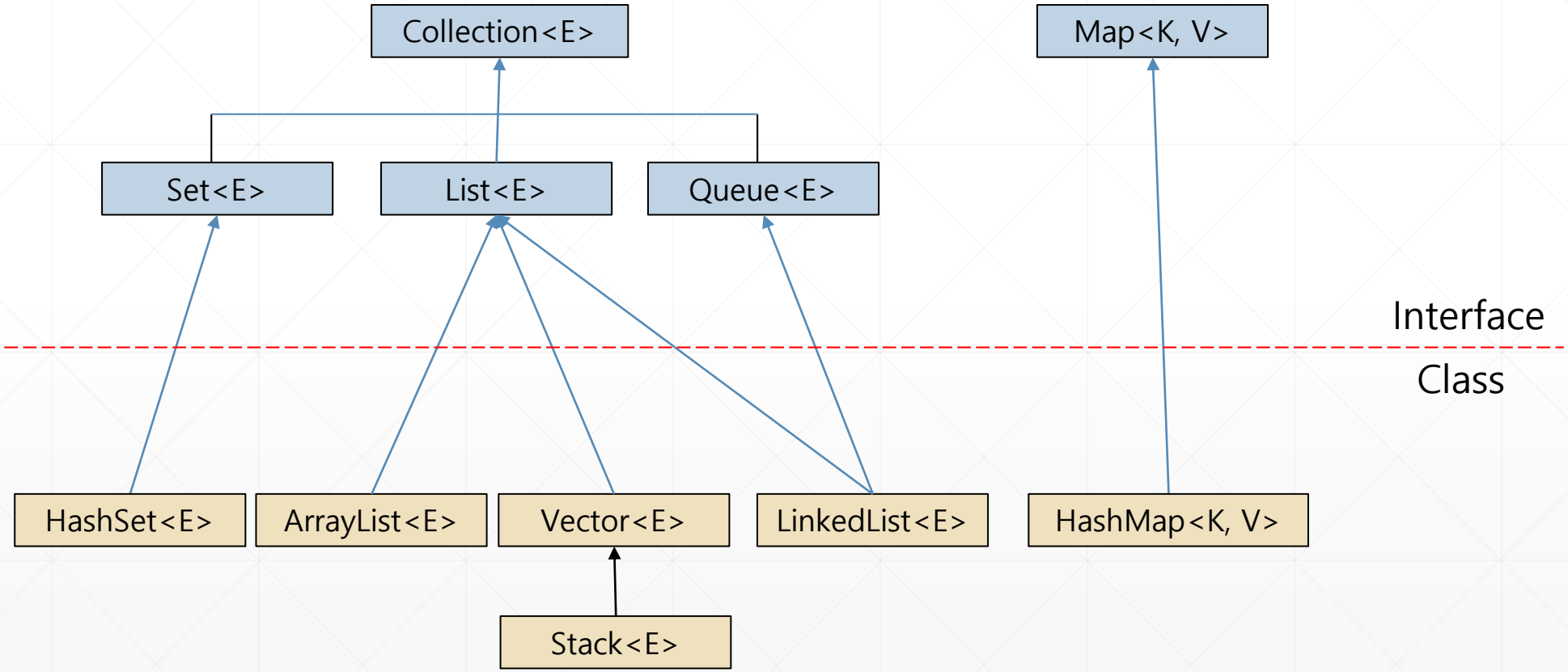
# Collection

- Storage of elements
  - Container of elements
  - Dynamically update the length based on the number of elements
  - Automatically update the position of elements according to the result of insert/delete operations
- Can overcome the limitation of a fixed length array
- Ease the Insertion, deletion, and search operations for various objects




# Collection (cont'd)

- Interface/class hierarchy



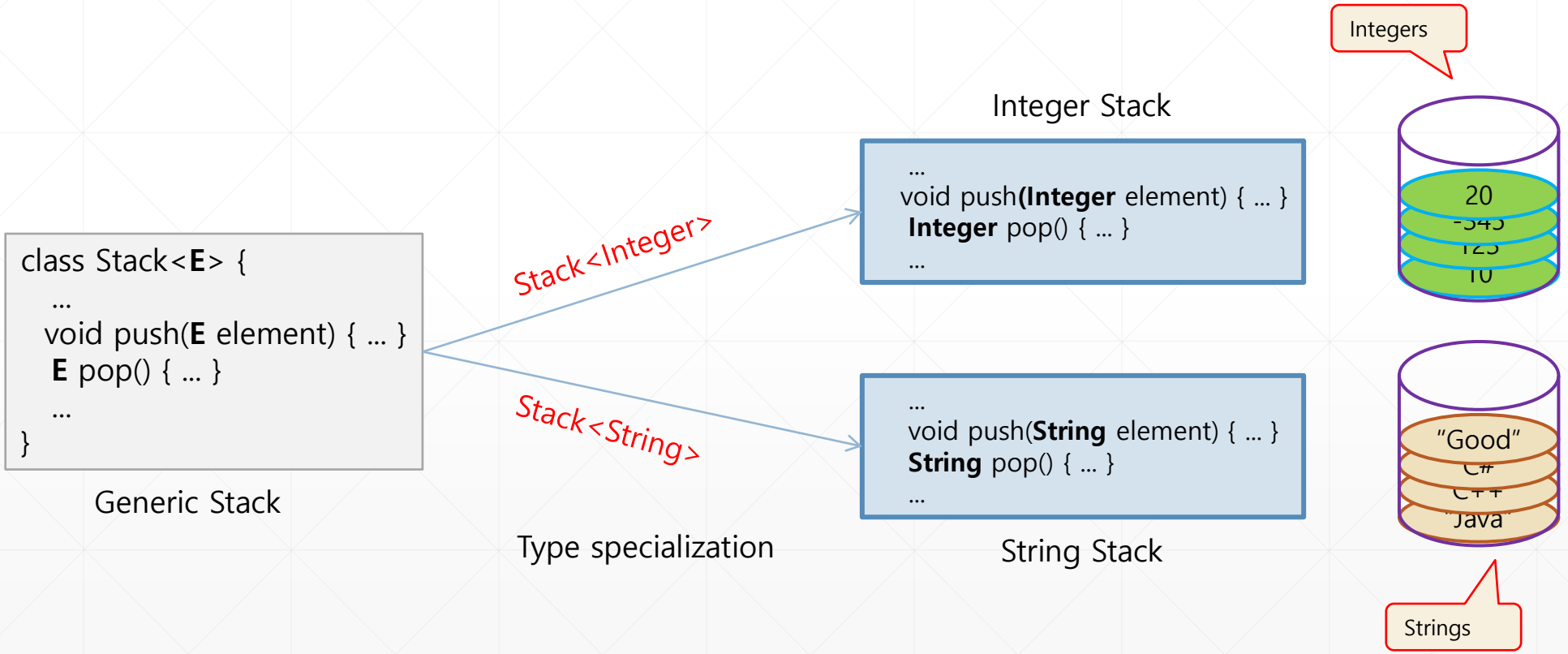
# Collection & Generic

- Collections are implemented based on Generics
  - Only objects can be elements of a collection
  - Primitive types cannot be used

- Generic
  - Enable *types* (classes and interfaces) to be parameters when defining classes, interfaces and methods
  - Type parameters provide a way to re-use the same code with different inputs

# Generic

- Provides a way to handle various types using generalized type parameters



# Generic (cont'd)

- Stack example

OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP Java SE 11 & JDK 11

ALL CLASSES SEARCH: Search

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

**Module** java.base
**Package** java.util

## Class Stack<E>

java.lang.Object
  java.util.AbstractCollection<E>
    java.util.AbstractList<E>
      java.util.Vector<E>
        java.util.Stack<E>

**All Implemented Interfaces:**

Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

---

```
public class Stack<E>
extends Vector<E>
```

The Stack class represents a last-in-first-out (LIFO) stack of objects. It extends class Vector with five operations that allow a vector to be treated as a stack. The usual push and pop operations are provided, as well as a method to peek at the top item on the stack, a method to test for whether the stack is empty, and a method to search the stack for an item and discover how far it is from the top.

When a stack is first created, it contains no items.

A more complete and consistent set of LIFO stack operations is provided by the Deque interface and its implementations, which should be used in preference to this class. For example:

```
Deque<Integer> stack = new ArrayDeque<Integer>();
```

**Since:**

1.0

# Generic (cont'd)

- Defining a generic class/interface

```
public class Box {
    private Object object;

    public void set(Object object) { this.object = object; }
    public Object get() { return object; }
}
```

Non-generic Box class

```
public class Box<T> {
    // T stands for "Type"
    private T t;

    public void set(T t) { this.t = t; }
    public T get() { return t; }
}
```

Generic version of Box class

# Generic (cont'd)

- Type parameter naming convention
  - By convention, type parameter names are single, uppercase letters
  - E - Element (used extensively by the Java Collections Framework)
  - K - Key
  - N - Number
  - T - Type
  - V - Value
  - S,U,V etc. - 2nd, 3rd, 4th types

# Generic: Specialization

- Defining a generic class/interface
  - Object instantiation of a generic class using specific type
  - Object instantiation of a generic class using primitive type is impossible

```
public class Box<T> {
    // T stands for "Type"
    private T t;

    public void set(T t) { this.t = t; }
    public T get() { return t; }
}
```

```
Box<String> s = new Box<String>(); // Setting String for generic type T
s.set("hello");
System.out.println(s.get()); // "hello"

Box<Integer> n = new Box<Integer>(); // Setting Integer for generic type T
n.set(5);
System.out.println(n.get()); //  5
```

# Generic: Specialization (cont'd)

- Defining a generic class/interface
  - What happens after specialization?

```
public class MyClass<T> {
   T val;
   void set(T a) {
      val = a;
   }
   T get() {
      return val;
   }
}
```

T to String

```
public class MyClass<String> {
   String val;
   void set(String a) {
      val = a;
   }
   String get() {
      return val;
   }
}
```

# Generic: Specialization (cont'd)

- Before Java 7

```
Box<Integer> v = new Box<Integer>();
```

- After Java 7

```
Box<Integer> v = new Box<>();
```

  - Type inference feature of compiler
  - Can skip type parameters in <> (diamond) as long as the compiler can determine the type arguments from the context

# Generic (cont'd)

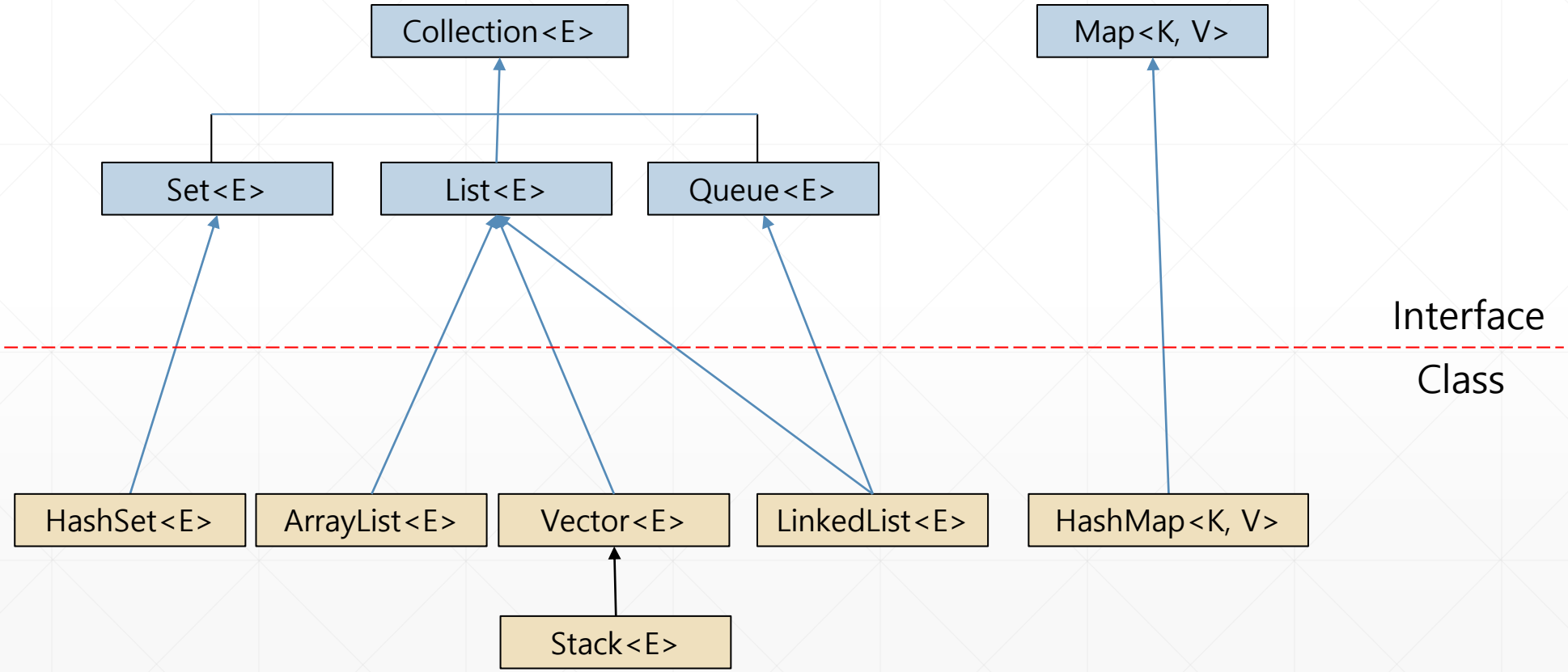
- Defining a generic class/interface with multiple type parameters

```
class Box<K, V> {
    private K k;
    private V v;

    public void set(K k, V v) {
        this.k = k;
        this.v = v;
    }

    public K getKey() {
        return k;
    }

    public V getValue() {
        return v;
    }

}
```

```
public class BoxEx{
    public static void main(String[] args) {
        Box<String, Integer> myBox = new Box<>();
        myBox.set("hey",5);
        System.out.println(myBox.getKey());
        System.out.println(myBox.getValue());

        Box<Double, Double> dBox = new Box<>();
        dBox.set(3.14, 3.14);
        System.out.println(dBox.getKey());
        System.out.println(dBox.getValue());
    }
}
```

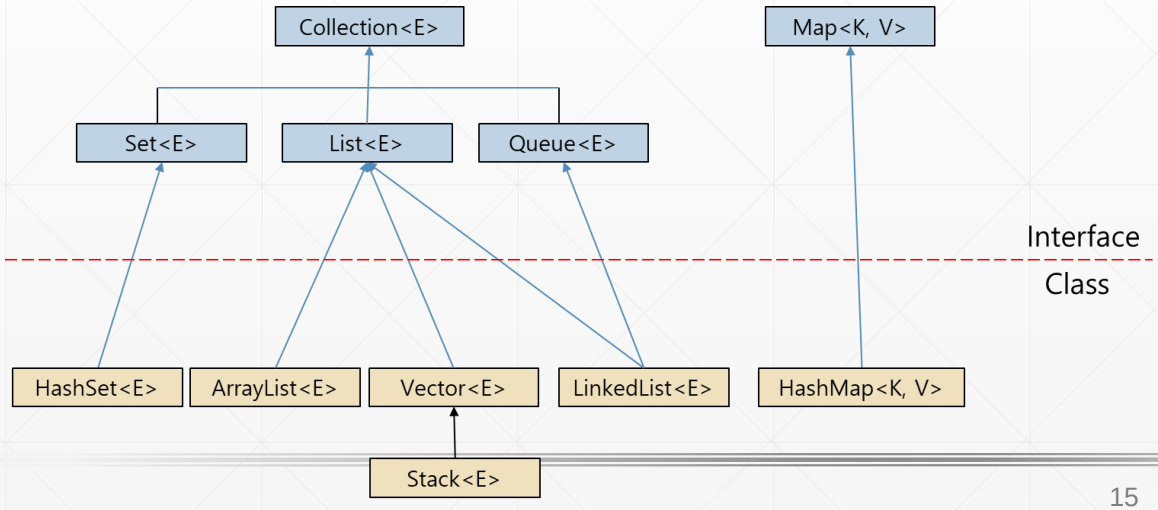
# Collections

- Interface/class hierarchy



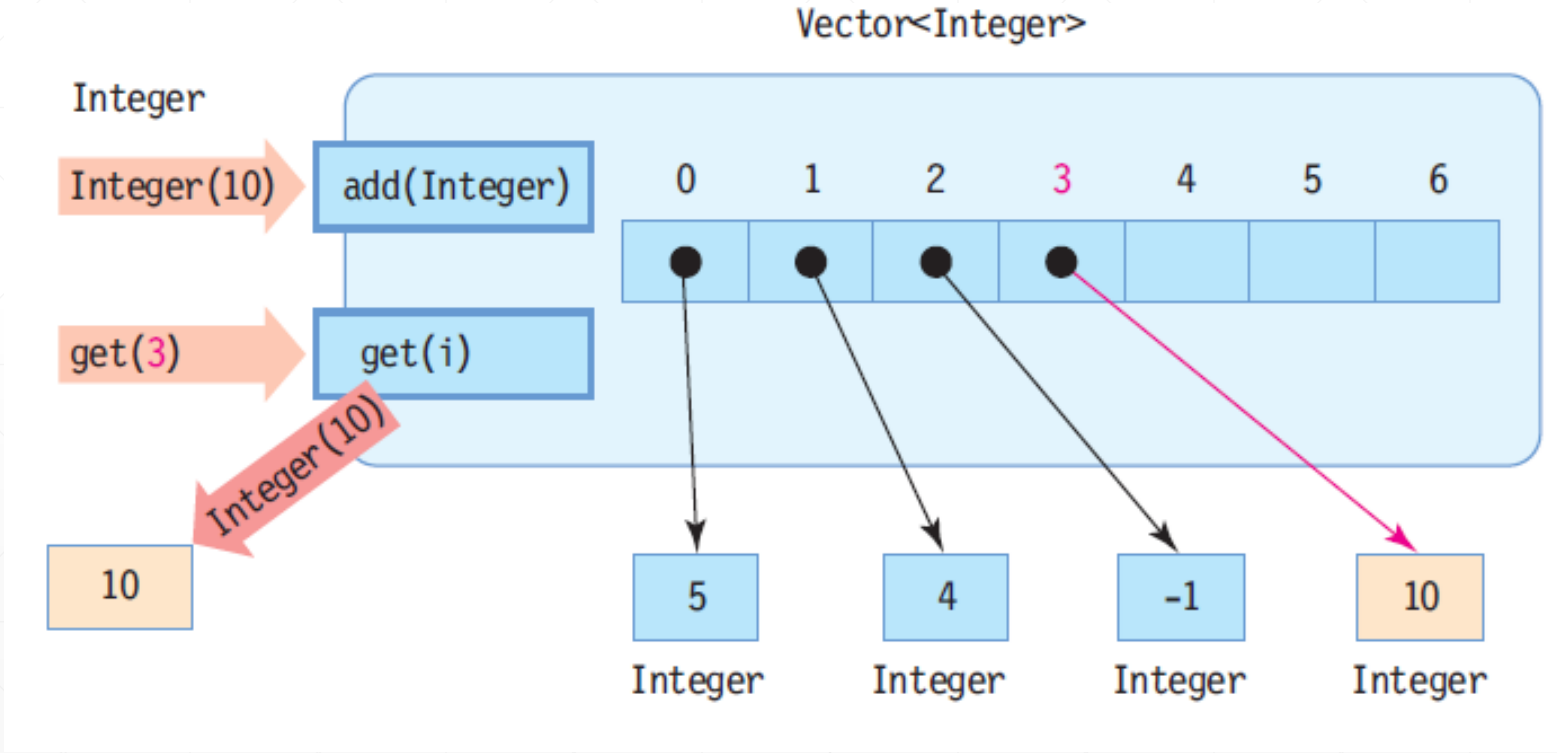
# Vector<E>

- Characteristics
  - java.util.vector
    - Can be specialized for <E>
  - Container class to insert, delete, search for multiple objects
    - Overcome the limitation of the fixed length of an array
    - Length is dynamically updated when overflow occurs
  - Vector can contain:
    - Object, null
    - Primitive types after boxing (i.e., wrapper class)
  - Support various collection features
    - insert/delete operations
    - contains() operation
    - Getters
    - …



# Vector<Integer>

```
Vector<Integer> v = new Vector<Integer>();
```



# Vector<E>: Methods

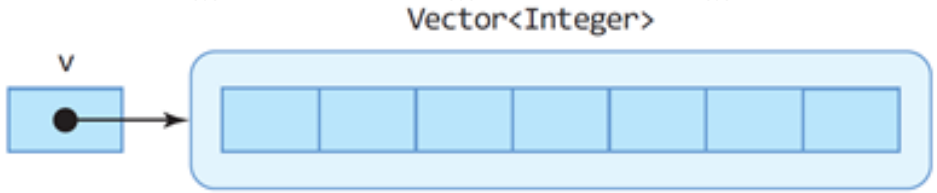
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Vector.html#method.summary

| Method | Description |
|---|---|
| `boolean add(E element)` | Appends the specified element to the end of this Vector |
| `void add(int index, E element)` | Inserts the specified element at the specified position in this Vector |
| `int capacity()` | Returns the current capacity of this Vector |
| `boolean addAll(Collection<? extends E> c)` | Appends all of the elements in the specified collection to the end of this Vector |
| `void clear()` | Removes all of the elements from this Vector |
| `boolean contains(Object o)` | Returns true if this Vector contains the specified element |
| `E elementAt(int index)` | Returns the component at the specified index |
| `E get(int index)` | Returns the element at the specified position in this Vector |
| `int indexOf(Object o)` | Returns the index of the first occurrence of the specified element in this Vector |
| `boolean isEmpty()` | Returns true if this Vector contains no elements |
| `E remove(int index)` | Removes the element at the specified position in this Vector |
| `boolean remove(Object o)` | Removes the first occurrence of the specified element from this Vector, if it is present |
| `void removeAllElements()` | Removes all components from this vector and sets its size to zero |
| `int size()` | Returns the number of elements in this Vector |
| `Object[] toArray()` | Returns an array containing all of the elements in this Vector in proper sequence |

# Vector<Integer>

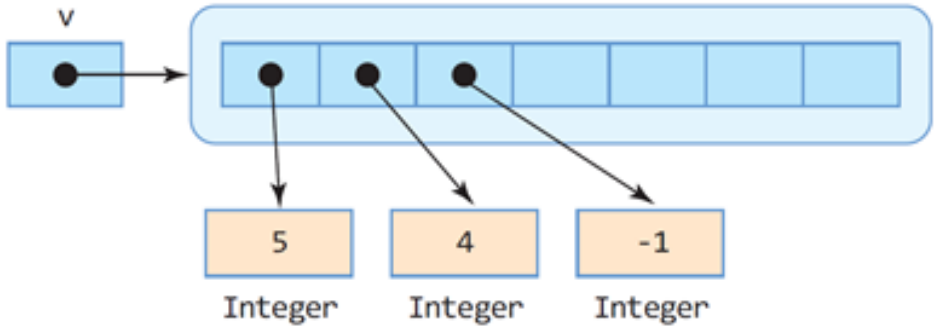
Create Vector

```
Vector<Integer> v = new Vector<Integer>(7);
```




Adding elements

```
v.add(5);
v.add(4);
v.add(-1);
```




Counting elements

```
int n = v.size();
int c = v.capacity();
```

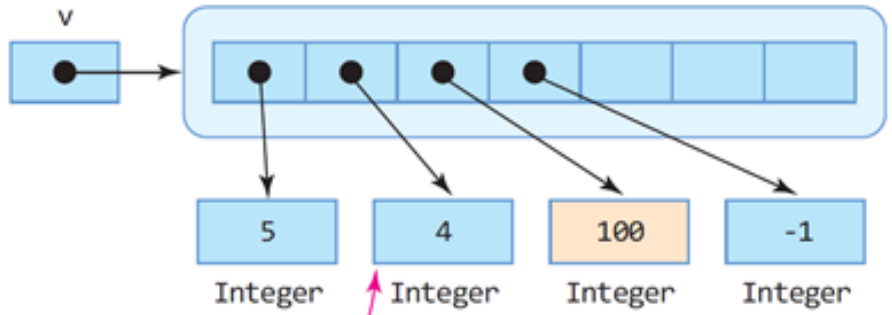
```
n = 3
c = 7
```

# Vector<Integer> (cont'd)

Adding elements

```
v.add(2, 100);
```

```
v.add(5, 100);
```




Getting element

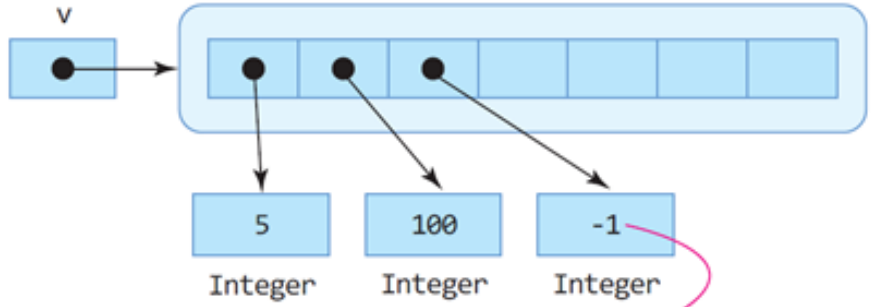
```
Integer obj = v.get(1);
int i = obj.intValue();
```

# Vector<Integer> (cont'd)

Removing elements

```
v.remove(1);
```

```
v.remove(4);
```

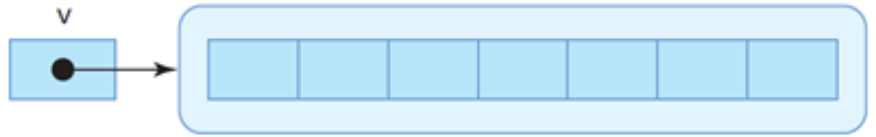



Removing all elements

```
int last = v.lastElement();
```

last = -1

```
v.removeAllElements();
```



# Vector<Integer> (cont'd)

- Auto boxing/unboxing
  - Boxing: primitive type → wrapper class
  - Unboxing: wrapper class → primitive type

```
Vector<Integer> v = new Vector<Integer> ();
v.add(4); // 4 → Integer.valueOf(4), auto boxing
int k = v.get(0); // Integer → int, auto unboxing (k = 4)
```

- Vector initialization with primitive type is impossible!

```
Vector<int> v = new Vector<int> (); // Error!
```

# Vector<Integer>: Example

- Basic usage of Vector<Integer>

```
import java.util.Vector;

public class VectorEx {
   public static void main(String[] args) {

      Vector<Integer> v = new Vector<Integer>();

      v.add(5);
      v.add(4);
      v.add(-1);

      // add element at specified index
      v.add(2, 100); // insert 100 between 4 and -1

      System.out.println("number of elements: " + v.size());
      System.out.println("current capacity: " + v.capacity());


      for(int i=0; i<v.size(); i++) {
         int n = v.get(i);
         System.out.println(n);
      }
      // sum all the number in the vector
      int sum = 0;
      for(int i=0; i<v.size(); i++) {
         int n = v.elementAt(i);
         sum += n;
      }
      System.out.println("sum of all integers in the vector: " + sum);
   }
}
```

# Vector<Integer>: Example (cont'd)

- Usage of Vector with a custom class

```
import java.util.Vector;

class Point {
   private int x, y;
   public Point(int x, int y) {
      this.x = x;
      this.y = y;
   }

   public String toString() {
      return "(" + x + "," + y + ")";
   }
}
```

```
public class PointVectorEx {
   public static void main(String[] args) {
      // Vector with Point class
      Vector<Point> v = new Vector<Point>();

      // adding 3 point instances
      v.add(new Point(2, 3));  // 0
      v.add(new Point(-5, 20)); // 1
      v.add(new Point(30, -8)); // 2

      v.remove(1); // remove a specific element

      //
      for(int i=0; i<v.size(); i++) {
         Point p = v.get(i); // getting i-th Point element
         System.out.println(p); //
      }
   }
}
```

# Vector<Integer>: Example (cont'd)

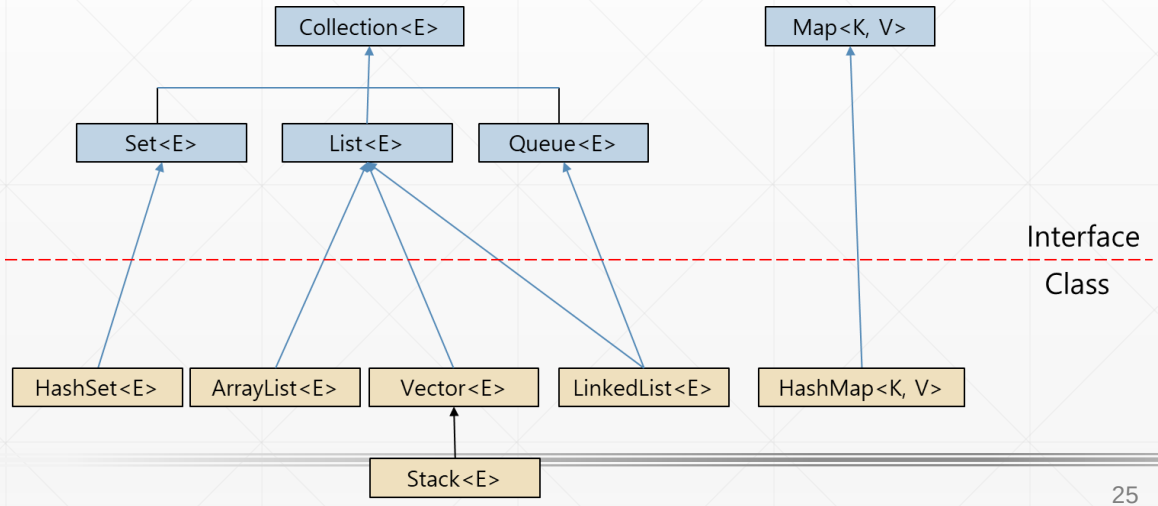
- Method which takes Collection as input
  - E.g.) public void printVector(Vector<Integer> v)

```
// takes Integer vector as input

public void printVector(Vector<Integer> v) {
   for(int i=0; i<v.size(); i++) {
      int n = v.get(i);
      System.out.println(n);
   }
}
```

```
Vector<Integer> v = new Vector<Integer>();
printVector(v); // invoke a method with a vector instance
```

# ArrayList<E>

- Characteristics
  - java.util.ArrayList, resizable-array implementation
    - Can be specialized for <E>
  - ArrayList can contain:
    - Object, null
    - Primitive types after boxing (i.e., wrapper class)
  - Support various collection features
    - Insert/delete operations
    - contains() operation
    - Getters
    - …



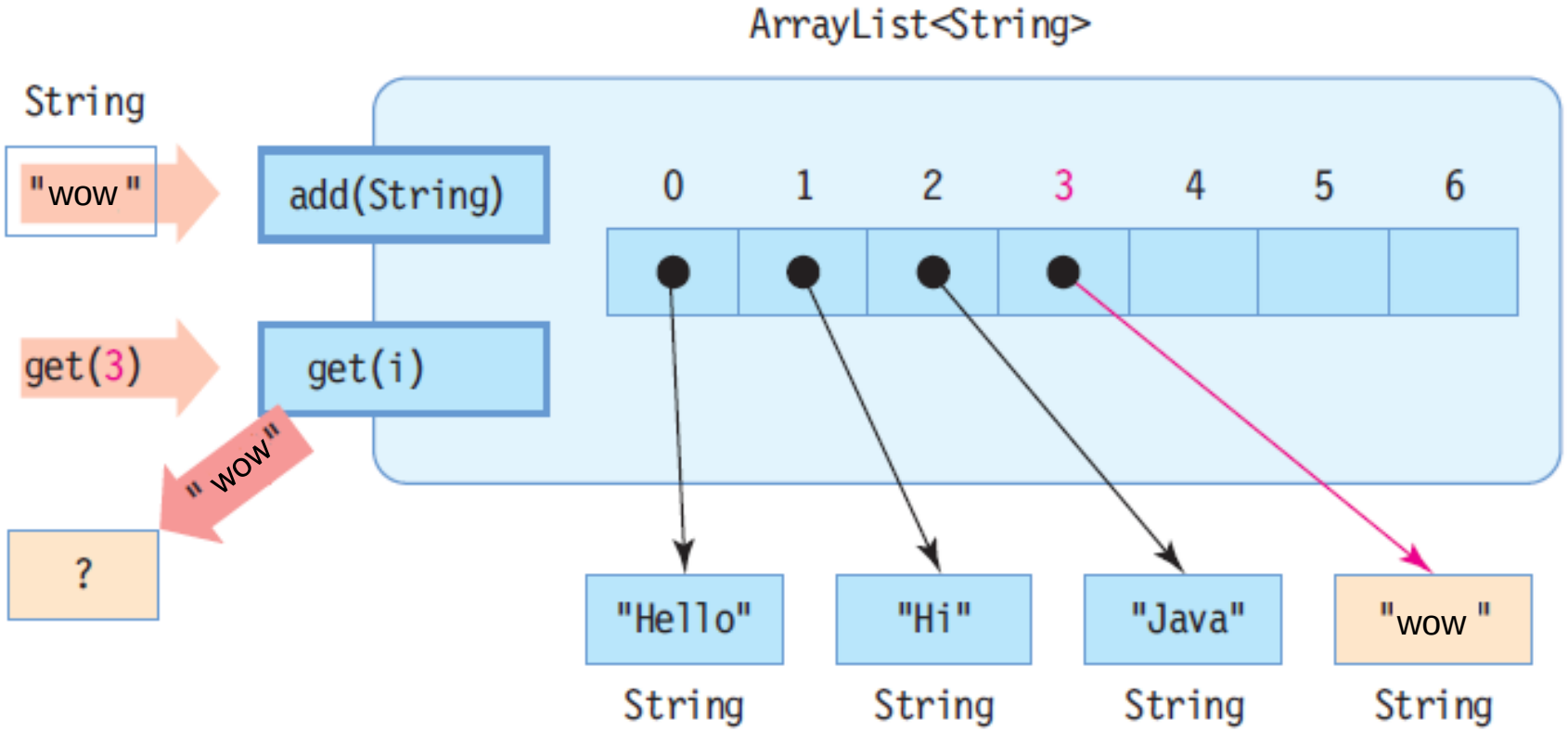
# ArrayList<E>: Methods

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/ArrayList.html#method.summary

| Method | Description |
|---|---|
| `boolean add(E element)` | 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 |
| `void clear()` | Removes all of the elements from this list |
| `boolean contains(Object o)` | Returns true if this list contains the specified element |
| `E get(int index)` | Returns the element at the specified position in this list |
| `int indexOf(Object o)` | Returns the index of the first occurrence of the specified element in this list |
| `boolean isEmpty()` | Returns true if this list contains no elements |
| `E remove(int index)` | Removes the element at the specified position in this list |
| `boolean remove(Object o)` | Removes the first occurrence of the specified element from this list, if it is present |
| `int size()` | Returns the number of elements in this list. |
| `Object[] toArray()` | Returns an array containing all of the elements in this list in proper sequence |

# ArrayList<String>

```
ArrayList<String> al = new ArrayList<String>();
```

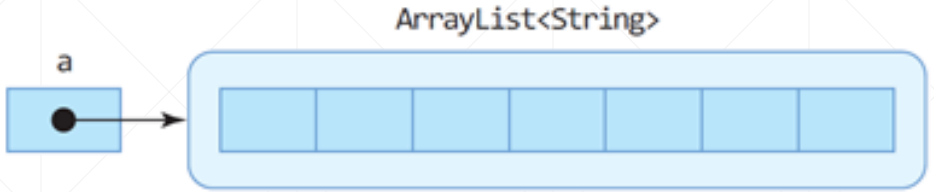


# ArrayList<String> (cont'd)

- Example)

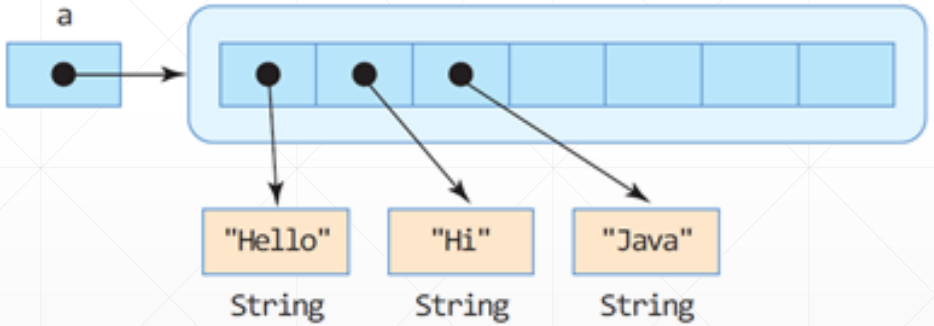
Create ArrayList

```
ArrayList<String> a = new ArrayList<String>(7);
```




Adding elements

```
a.add("Hello");
a.add("Hi");
a.add("Java");
```




Counting elements

```
int n = a.size();
int c = a.capacity();
```

n = 3

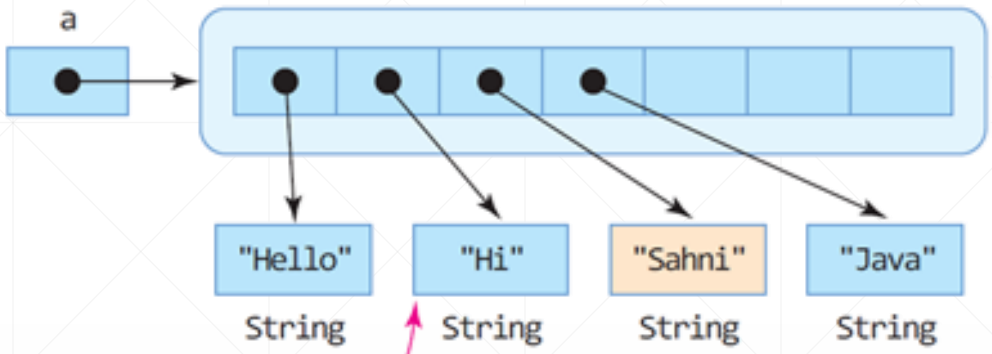
# ArrayList<String> (cont'd)

- Example)

Adding elements

```
a.add(2, "Sahni");
```

~~a.add(5, "Sahni");~~




Getting element

```
String str = a.get(1);
```

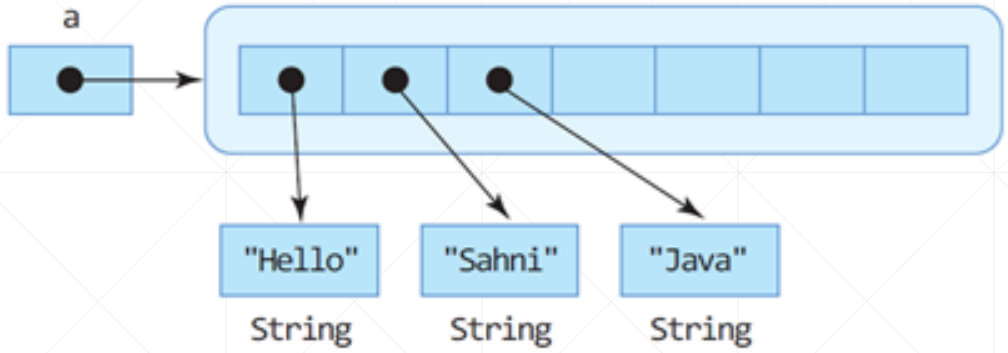
# ArrayList<String> (cont'd)

- Example)

Removing elements

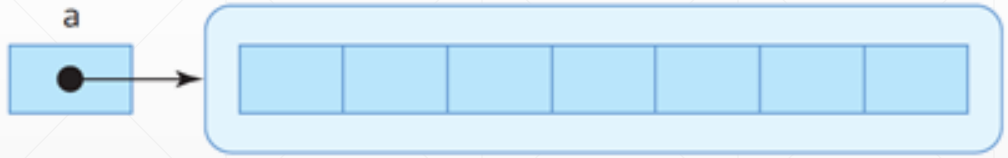
```
a.remove(1);
```

```
a.remove(4);
```




Removing all elements

```
a.clear();
```



# ArrayList<String> (cont'd)

- Example) take 4 names and store them into ArrayList. Then, print all the names and the longest one.

```
import java.util.*;

public class ArrayListEx {
    public static void main(String[] args) {

        ArrayList<String> a = new ArrayList<String>();
        Scanner scanner = new Scanner(System.in);

        for (int i = 0; i < 4; i++) {
            System.out.print("Input your name >> ");
            String s = scanner.next();
            a.add(s);
        }

        // printing all names
        for (int i = 0; i < a.size(); i++) {
            // Getting i-th element
            String name = a.get(i);
            System.out.print(name + " ");
        }
…
```

```
…
        // finding the longest name
        int longestIndex = 0;
        for (int i = 1; i < a.size(); i++) {
            if (a.get(longestIndex).length() < a.get(i).length())
                longestIndex = i;
        }
        System.out.println("\n the longest one is : " + a.get(longestIndex));
        scanner.close();
    }
}
```

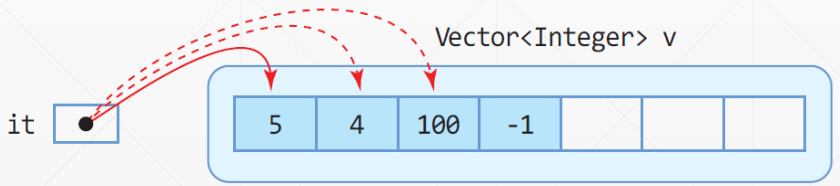
# Iterator

- Iterator<E> interface
  - Vector<E>, ArrayList<E>, LinkedList<E>
  - Declare methods to iteratively visit the elements in the list-type data structure
  - Methods

| Method | Description |
|---|---|
| `boolean hasNext()` | Returns true if the iteration has more elements |
| `E next()` | Returns the next element in the iteration |
| `void remove()` | Removes from the underlying collection the last element returned by this iterator (optional operation) |

  - iterator() method: returns an iterator instance
    - Can use this instance to iteratively visit each element in the collection

```
Vector<Integer> v = new Vector<Integer>();
Iterator<Integer> it = v.iterator();
while(it.hasNext()) { //
    int n = it.next(); //
    ...
}
```



# Iterator: Example

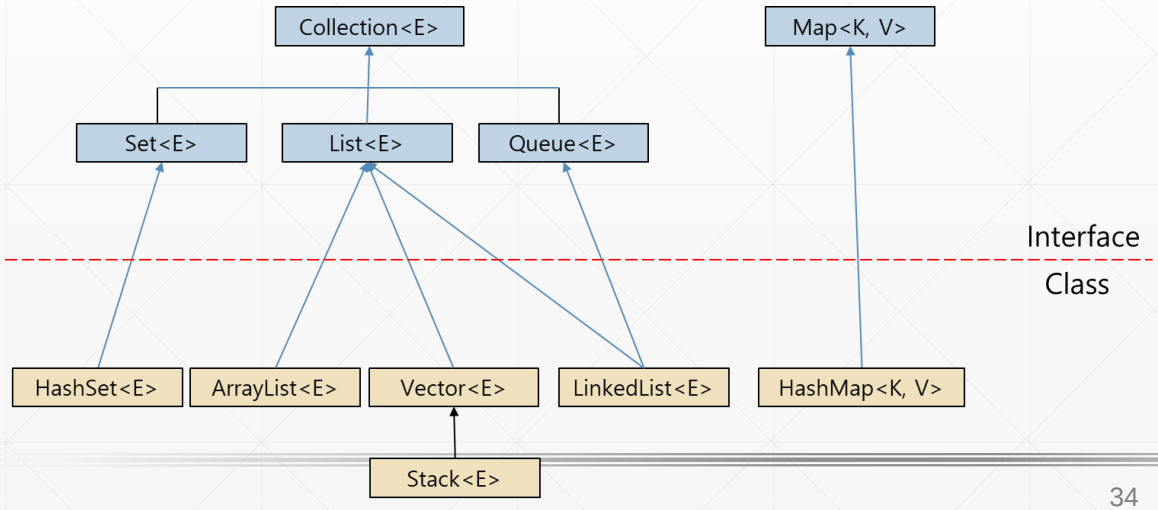
- Iterator<E> interface

```
import java.util.*;

public class IteratorEx {
   public static void main(String[] args) {

      Vector<Integer> v = new Vector<Integer>();
      v.add(5);
      v.add(4);
      v.add(-1);
      v.add(2, 100);

      // print all elements using Iterator

      Iterator<Integer> it = v.iterator();
      while(it.hasNext()) {
         int n = it.next();
         System.out.println(n);
      }
```

```
      // sum all the elements using Iterator

      int sum = 0;
      it = v.iterator();
      while(it.hasNext()) {
         int n = it.next();
         sum += n;
      }
      System.out.println("sum: " + sum);
   }
}
}
```

# HashMap<K,V>

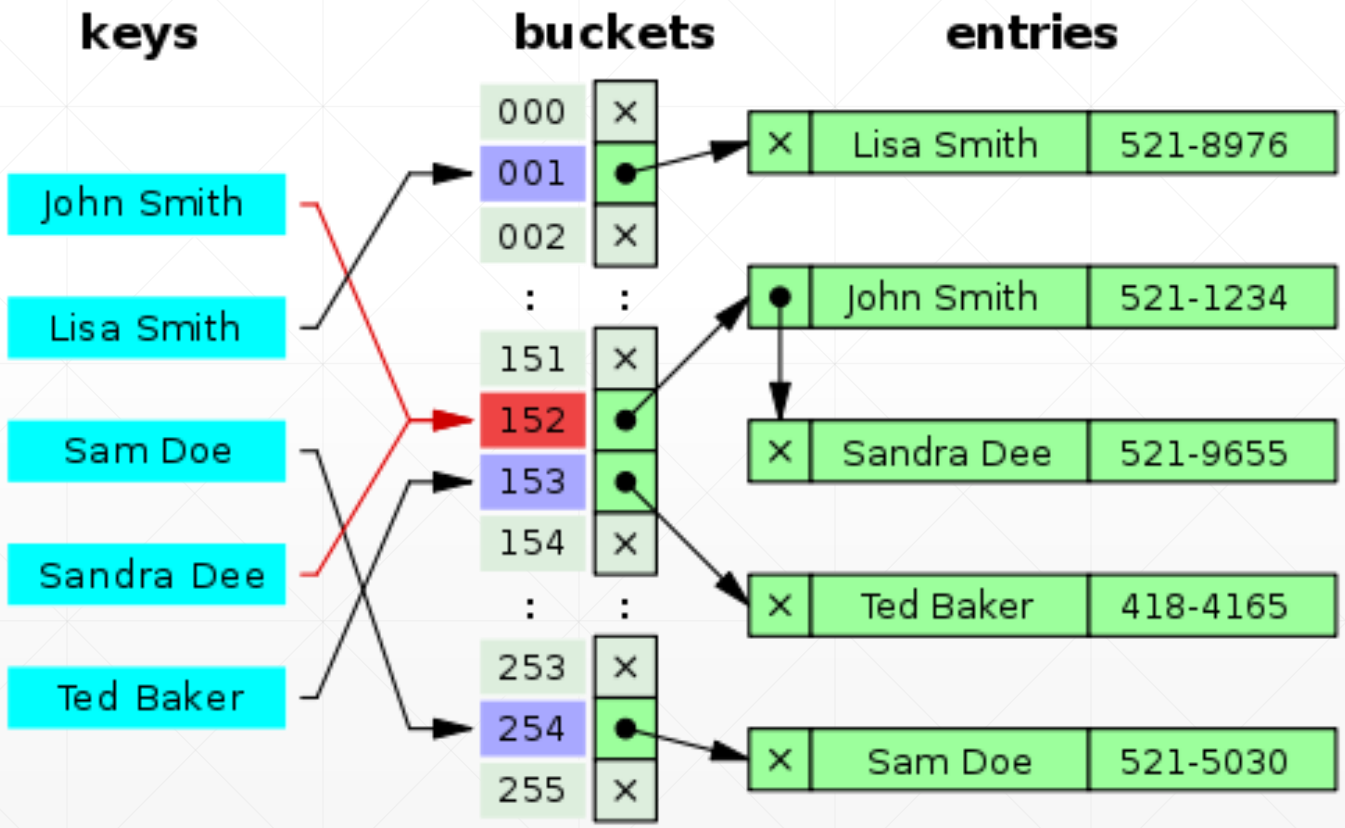
- Characteristics
  - java.util.HashMap
  - Container class to manage key-value pairs
    - K: type to be used for keys, V: type to be used for values
    - Key determines a position where the element is located (therefore, key must be unique)
    - Values can be searched based on the key
  - Support various collection features
    - Insert: put() method
    - Search: get() method
    - …



# HashMap<String,String>

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/HashMap.html#method.summary

```
HashMap<String, String> map = new HashMap<String, String>();
```



# HashMap<K,V>

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/HashMap.html#method.summary

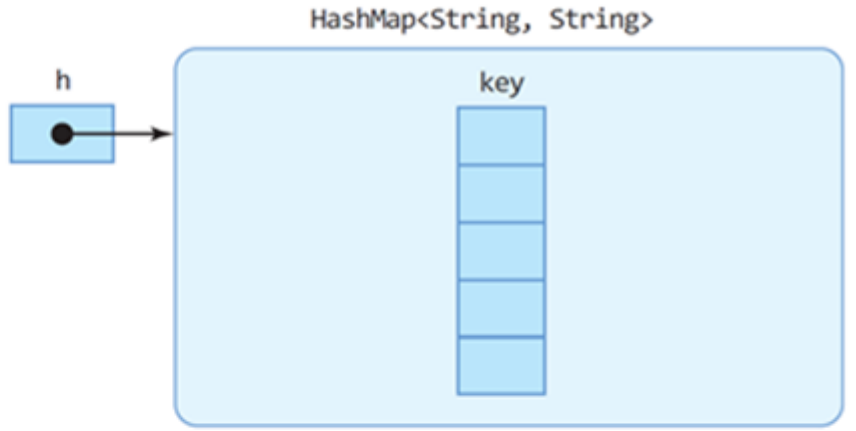
- Methods

| Method | Description |
|---|---|
| `void clear()` | Removes all of the mappings from this map |
| `boolean containsKey(Object key)` | Returns true if this map contains a mapping for the specified key |
| `boolean containsValue(Object value)` | Returns true if this map maps one or more keys to the specified value |
| `V get(Object key)` | Returns the value to which the specified key is mapped |
| `boolean isEmpty()` | Returns true if this map contains no key-value mappings |
| `Set<K> keySet()` | Returns a Set view of the keys contained in this map |
| `V put(K key, V value)` | Associates the specified value with the specified key in this map |
| `V remove(Object key)` | Removes the mapping for the specified key from this map if present |
| `int size()` | Returns the number of key-value mappings in this map |

# HashMap<String,String> (cont'd)

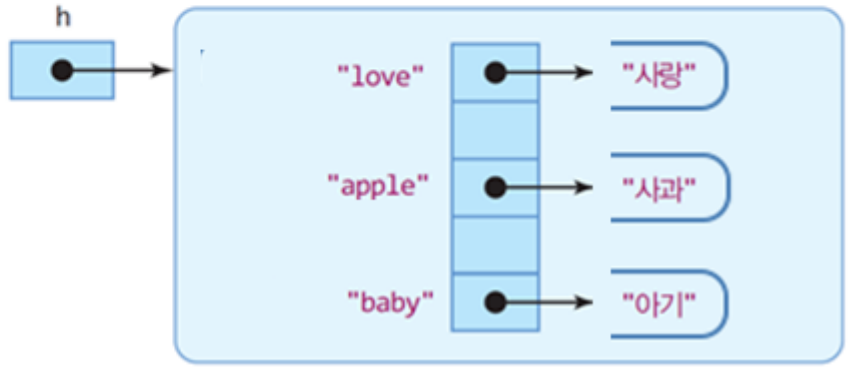
Create HashMap

```
HashMap<String, String> h =
new HashMap<String, String>();
```




Adding elements

```
h.put("baby", "아기");
h.put("love", "사랑");
h.put("apple", "사과");
```



# HashMap<String,String> (cont'd)

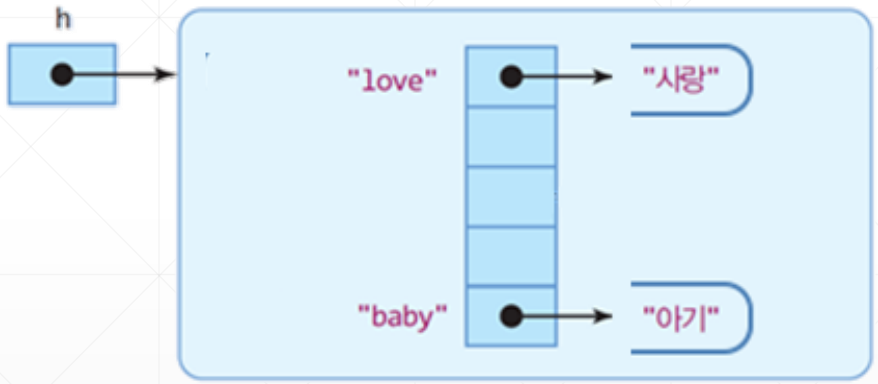
Getting elements

```
String kor = h.get("love");
```

kor = "사랑"

Removing elements

```
h.remove("apple");
```




Counting elements

```
int n = h.size();
```

n = 2

# HashMap<K,V>: Example

- Dictionary implementation

```
import java.util.*;

public class HashMapDicEx {
   public static void main(String[] args) {
      // HashMap for <String,String> pairs
      HashMap<String, String> dic = new HashMap<String, String>();

      // add 3 pairs
      dic.put("baby", "아기");
      dic.put("love", "사랑");
      dic.put("apple", "사과");

      // take English word and return its corresponding Korean word
      Scanner scanner = new Scanner(System.in);
      while(true) {
         System.out.print("which word?");
         String eng = scanner.next();
         if(eng.equals("exit")) {
            System.out.println("exit...");
            break;
         }
         String kor = dic.get(eng);
         if(kor == null)
            System.out.println(eng +
                  "does not exist.");
         else
            System.out.println(kor);
      }
      scanner.close();
   }
}
```

# HashMap<K,V>: Example

- ScoreTable implementation

```
public class HashMapScoreEx {
   public static void main(String[] args) {
      // HashMap for <String, Integer>
      HashMap<String, Integer> javaScore =
            new HashMap<String, Integer>();

      javaScore.put("jinwoo", 97);
      javaScore.put("jinhee", 88);
      javaScore.put("jinha", 98);
      javaScore.put("jinkoo", 70);
      javaScore.put("jindo", 99);

      System.out.println("HashMap's size :" + javaScore.size());

      // print all (key, value) pairs in javaScore HashMap
      // get Set collection containing all keys of HashMap
      Set<String> keys = javaScore.keySet();

      // get an Iterator for Set collection
      Iterator<String> it = keys.iterator();
      while(it.hasNext()) {
         String name = it.next();
         int score = javaScore.get(name);
         // get the value for that key from HashMap
         System.out.println(name + " : " + score);
      }
   }
}
```

# HashMap<K,V>: Example

- StudentTable implementation

```
class Student {
   int id;
   String tel;
   public Student(int id, String tel) {
      this.id = id; this.tel = tel;
   }
}
```

```
public class HashMapStudentEx {
   public static void main(String[] args) {
      // HashMap for <String, Student> pairs
      HashMap<String, Student> map = new HashMap<String, Student>();

      map.put("jinwoo", new Student(1, "010-111-1111"));
      map.put("jindo", new Student(2, "010-222-2222"));
      map.put("jinha", new Student(3, "010-333-3333"));

      Scanner scanner = new Scanner(System.in);
      while(true) {
         System.out.print("name?");
         String name = scanner.nextLine();
         if(name.equals("exit"))
            break; // exit the program
         Student student = map.get(name);
         if(student == null)
            System.out.println(name + "does not exist.");
         else
            System.out.println("id:" + student.getId() + ", tel:" + student.getTel());
      }
      scanner.close();
   }
}
```

# Collections

- Java.util.collections
  - Operates on collections and return collections
  - Only has static methods
- Methods
  - sort()
  - reverse()
  - min()/max()
  - …

# Collections: Example

- Usage of Collections class

```
import java.util.*;

public class CollectionsEx {
      static void printList(Vector<String> l) {
         Iterator<String> iterator = l.iterator();
         while (iterator.hasNext()) {
               String e = iterator.next();
               String separator;
               if (iterator.hasNext())
                     separator = "->";
               else
                     separator = "₩n";
               System.out.print(e+separator);
         }
      }
```

```
   public static void main(String[] args) {
         Vector<String> myList = new Vector<String>();
         myList.add("Transformer");
         myList.add("StarWars");
         myList.add("Matrix");
         myList.add(0,"Terminator");
         myList.add(2,"Avatar");

         Collections.sort(myList); // sorting elements
         printList(myList);

         Collections.reverse(myList); // reversing elements
         printList(myList);

         System.out.println(Collections.min(myList));
   }
}
```

# Q&A

- Next week (eClass video)
  - File IO
  - Exercises