Collection Classes

An ADT in which each object contains a collection of elements

What is a Collection Class

- A collection class = an ADT in which is object can hold a group of items

  Example: grades for an exam, a grocery list, etc.

  In Java, Collection classes can be implemented as a class, along with methods to add, remove, and examine items

Example: A bag

- Consider a bag object that can hold numbered cards
  - Initially empty
  - Can hold many cards – numbers could appear multiple times

Operations:
- Add a card (only one at a time)
- Remove a card (only one at a time)
- Count/state how many numbers are in the bag
- Count how many of a particular card there is in the bag

We can implement such a bag using a Java class!

Other Kinds of Bags…

- In this example, we have implemented a bag containing **integers**.
- But we could have had a bag of **float numbers**, a bag of **characters**, a bag of **Strings** . . .

  - *Suppose you wanted one of these other bags. How much would you need to change in the implementation ?*

Implementation of Collection Classes

- Collection classes can be implemented in many ways.
- The simplest way to implement a collection class is to use an `array`

`Arrays` in Java

- Using an array of integers:
  - `int[] scores = new int[10];`
  - `scores[2] = 10;`
  - `int y = scores[5];`

- Printing an array:
  - You have to use a for loop to print each item separately.

The `Arrays` utility class

- The `Arrays` Java library includes static methods that are intended to help a programmer to manipulate arrays.

- Common methods:
  - `Arrays.copyof(inArray,3)`
    - Copies only the first 3 items of `inArray`
  - `Arrays.copyOfRange(inArray,1,3)`
  - `Arrays.fill(inArray,100)`
    - Fills all the elements of `inArray` with the value 100
  - `System.arraycopy(array1,0,array2,0,10)`
    - Copies `array1` into `array2`
    - Starts from index 0 in `array1` and index 0 in `array2`
    - Copies only 10 elements

- To access these methods,
  - `import java.util.Arrays;`

Example of a Java Collection Class?

- Java's `ArrayList`

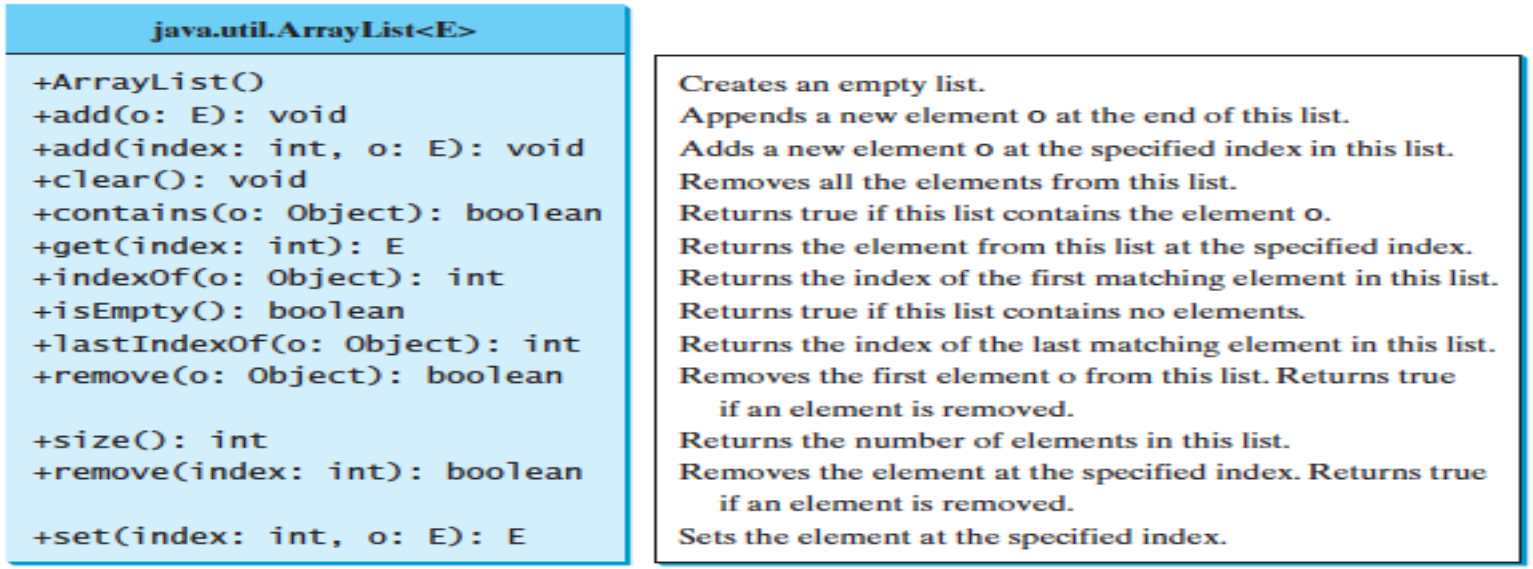


FIGURE 11.3 An `ArrayList` stores an unlimited number of objects.

Example 1 on using `ArrayList`

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

```
ArrayList myArray = new ArrayList();

myArray.add("New York");
myArray.add("Minneapolis");
myArray.add("St. Paul");
myArray.add("Seattle");


//Printing the array
System.out.println(myArray);

//deleting an item from the array
myArray.remove("St. Paul");
System.out.println(myArray);
```

Output:

```
[New York, Minneapolis, St. Paul, Seattle]



[New York, Minneapolis, Seattle]
```

Example 2 on using `ArrayList`

```
Line l1 = new Line(10,10,20,20);
Line l2 = new Line(10,10,30,30);

ArrayList lineArray = new ArrayList();

lineArray.add(l1);
lineArray.add(l2);

for (int i=0; i < lineArray.size(); i++){
      double length = ((Line)(lineArray.get(i))).length();
      System.out.println(length);
}
```

Output:

```
14.142135623730951
28.284271247461902
```

TABLE 11.1 Differences and Similarities between 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();`

Using an `ArrayList` is easier than using an Array

- The size of an `ArrayList` is flexible so you don't have to specify its size in advance.
  - On the other hand, the size of an `Array` has to be specified at creating time.
    - `int[] a = new int[10]`

- `ArrayList` supports many useful functionalities that are harder to perform when using an `Array`, for example:
  - `contains`: to test whether an element is in the list.
  - `remove`: to delete an element from the list

Java Collections Framework

- `Java Collections Framework` provides several data structures that can be used to organize and manipulate data efficiently.
  - A data structure is a collection of data organized in some fashion.

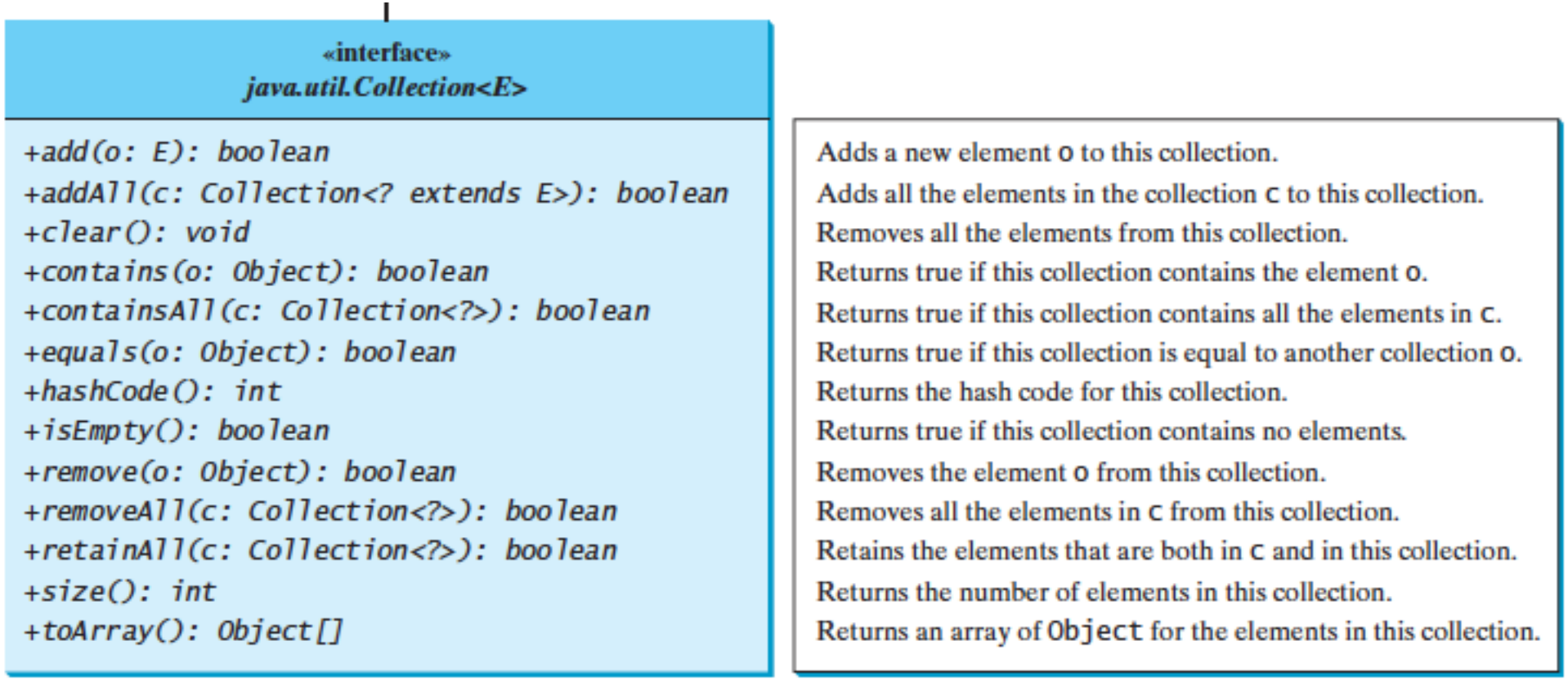
- To define a data structure is essentially to define a class. The class for a data structure should:
  - use data fields to store data, and
  - provide methods to support operations such as `search`, `insert`, and `delete`.

- In object-oriented thinking, a data structure, also known as a container or container object , is an object that stores other objects, referred to as data or elements.

Types of Java Collections

- **`Sets:`** store a group of non-duplicate elements.
- **`Lists:`** store an ordered collection of elements.
- **`Stacks:`** store objects that are processed in a last-in, first-out fashion.
- **`Queues:`** store objects that are processed in a first-in, first-out fashion.
- **`PriorityQueues:`** store objects that are processed in the order of their priorities.

- The `Collection` interface contains the methods for manipulating the elements in a collection



Example on using Java's built-in Collections

ArrayList

```
ArrayList<Integer> intArray = new ArrayList();


intArray.add(10);
intArray.add(5);
intArray.add(15);
intArray.add(5);
intArray.add(1);


System.out.println(intArray);
```

[10, 5, 15, 5, 1]

HashSet

```
HashSet<Integer> intHashSet = new HashSet();


intHashSet.add(10);
intHashSet.add(5);
intHashSet.add(15);
intHashSet.add(5);
intHashSet.add(1);


System.out.println(intHashSet);
```

[1, 5, 10, 15]

Comparing `ArrayList` and `HashSet`

- **Order of elements:**
  - `ArrayList` is **ordered**, which means that it stores element in the order they were added, while `Set` implementation doesn't provide such guarantee.
  - `HashSet` stores the elements in a **sorted** order according to their values.

- **Accessing elements:**
  - `ArrayList` provides random access which means you can access any element by index (`get(index)` and `remove(index)`)
  - `HashSet` does not provide `get` method. You can just iterate using an iterator (will be discussed later)

- **Duplicates:**
  - `HashSet` does not allow duplicates.

ArrayList Interface

boolean add(Object o) This method appends the specified element to the end of this list.
void add(int index, Object element) This method inserts the specified element at the specified position in this list.
boolean contains(Object o) This method returns true if this list contains the specified element.
Object get(int index) This method returns the element at the specified position in this list.
int indexOf(Object o) This method returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
boolean isEmpty() This method returns true if this list contains no elements.
int lastIndexOf(Object o) This method returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
Object remove(int index) This method removes the element at the specified position in this list.
boolean remove(Object o) This method removes the first occurrence of the specified element from this list, if it is present.
E set(int index, E element) This method replaces the element at the specified position in this list with the specified element.

HashSet Interface

boolean add(Object o) Adds the specified element to this set if it is not already present.
boolean contains(Object o) Returns true if this set contains the specified element.
boolean isEmpty() Returns true if this set contains no elements.
Iterator iterator() Returns an iterator over the elements in this set.
boolean remove(Object o) Removes the specified element from this set if it is present.

How to decide which collection class to use?

- **Example 1:** write a program to read a text file and makes an alphabetical list of all words in that file.
  - Use a `HashSet`
  - Because we need to sort the words alphabetically
  - You can add more code to add the number of times each word appears
- **Example 2:** write a program to store student scores in a certain test. Your program should allow the user to add, delete and update the score of any student.
  - Use `ArrayList`
  - Each student will be stored at a given index in the array and hence we can access any student's information randomly.

Observations about using Collections' ADT

- The user can perform only the operations specific to the ADT that are declared by the interface.
- The user must adhere to the specifications (i.e., number and data type of input parameters) of the operations that the ADT provides.
- The user must understand how to use the operations.
- The client cannot access the data within the collection without using ADT operations (encapsulation).
- The client can use the collection even if the client does not know how the data is stored and retrieved.

Build Your Own

- In this class you will implement your own collection classes
  - **Abstract Data Type Interface (or specification):**
    - A description of how the class organizes data, and
    - What are the available methods
  - **Implementation**

`Bag` Collection class

- In this lecture we will implement a `Bag` collection class that can be used to store an unordered list of integers.

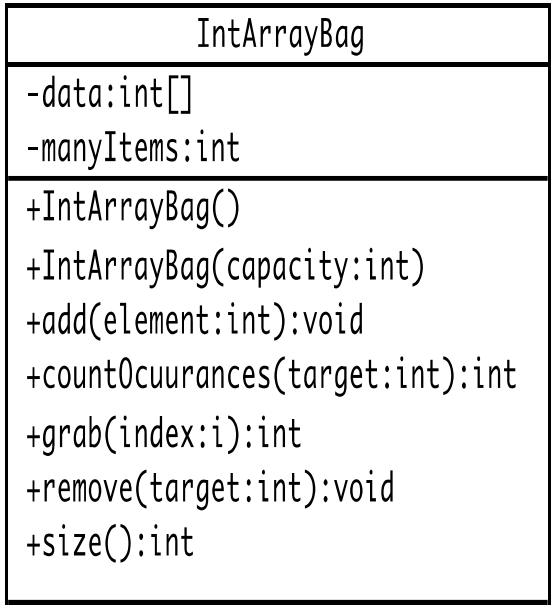
- The ADT specifications are given, and we will walk through how to implement the given specifications.

https://www.cs.colorado.edu/~main/docs/edu/colorado/collections/IntArrayBag.html

`Bag` ADT Specifications

- In order to use and/or implement the given ADTs, we need to define the following:
  - **Constructors:**
    - Specify how you can create a new collection.
  - **Accessor methods:**
    - Specify how you can `insert`, `delete`, and `retrieve` an element from the collections
  - **Capacity handling methods:**
    - Specify how to handle the capacity of the collection, for example:
      - How to insert an element in a full collection
      - How to reduce the memory usage of the collection is almost empty
  - **Handling multiple instances of the same collection:**
    - Specifies how to do `union`, `intersect`, and `difference` between two collections of the same type.

IntArrayBag Interface

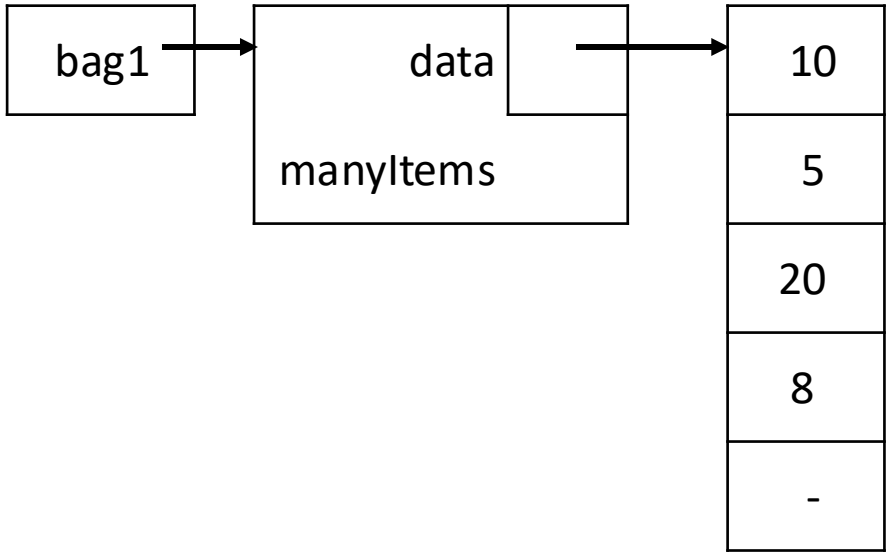


Examples on using `IntArrayBag`

```
IntArrayBag bag1 = new IntArrayBag(10);

bag1.add(10);
bag1.add(5);
bag1.add(20);
bag1.add(8);
System.out.println(bag1); //[10, 5, 20, 8, ]


bag1.remove(5);
System.out.println(bag1); //[10, 8, 20, ]
```



Bag ADT: constructor methods

IntArrayBag() Initialize an empty bag with an initial capacity of 10.
IntArrayBag(int initialCapacity) Initialize an empty bag with a specified initial capacity.

Bag ADT: accessor and mutator methods

void	**add(int element)** Add a new element to this bag.
boolean	**remove(int target)** Remove one copy of a specified element from this bag.
int	**size()** Determine the number of elements in this bag.
int	**countOccurrences(int target)** Accessor method to count the number of occurrences of a particular element in this bag.
String	**toString()** A method to return a string that is composed of all items in the collection (each item in a separate line).

ADT Specifications and Code

- Your textbook provides
  - An interface for `IntArrayBag` at https://www.cs.colorado.edu/~main/docs/edu/colorado/collections/IntArrayBag.html
  - Links to code for all the data structures in the book at https://www.cs.colorado.edu/~main/docs/edu/colorado/collections/

Invariants of an ADT

- An invariant is an explicit statement of the rules dictating how member variables are to be used.
- All methods may assume that the invariant is valid when they are called.
- Each method is responsible for ensuring continuing validity of the invariant when the method returns.

Invariants of `IntArrayBag` ADT

- The number of items contained in the bag is stored in an instance variable called `manyItems`

- The bag entries are stored in an instance array variable called `data`, from position `data[0]` to position `data[manyItems-1]`

Implementing Invariants

- Invariants are comments
- Place invariant comments at the top of each collection class
- Invariants are instructions or reminders to programmers
- Invariants are essential documentation for collection classes

Additional Methods

Handling multiple instances of Bag

void	**addMany(int...elements)** Add new elements to this bag.
void	**addAll(IntArrayBag addend)** Add the contents of another bag to this bag.
static IntArrayBag	**union(IntArrayBag b1, IntArrayBag b2)** Create a new bag that contains all the elements from two other bags.

Handling Capacity of Bag

int	**getCapacity()** Accessor method to get the current capacity of this bag.
void	**ensureCapacity(int minimumCapacity)** Change the current capacity of this bag.
void	**trimToSize()** Reduce the current capacity of this bag to its actual size (i.e., the number of elements it contains).

Summary

- A Collection class is a class that can hold a group of items.
- Collection classes can be implemented with a Java class.
- The author of the class should provide documentation that another programmer can read to use the class.
- Remember: If you are just **using** the Bag class, then you don't need to know how the operations are implemented.