Collections

Topics

1. What are collections?
2. Basics of Lists, Sets, and Maps
3. The Collection Interface
4. Iterators

Collections

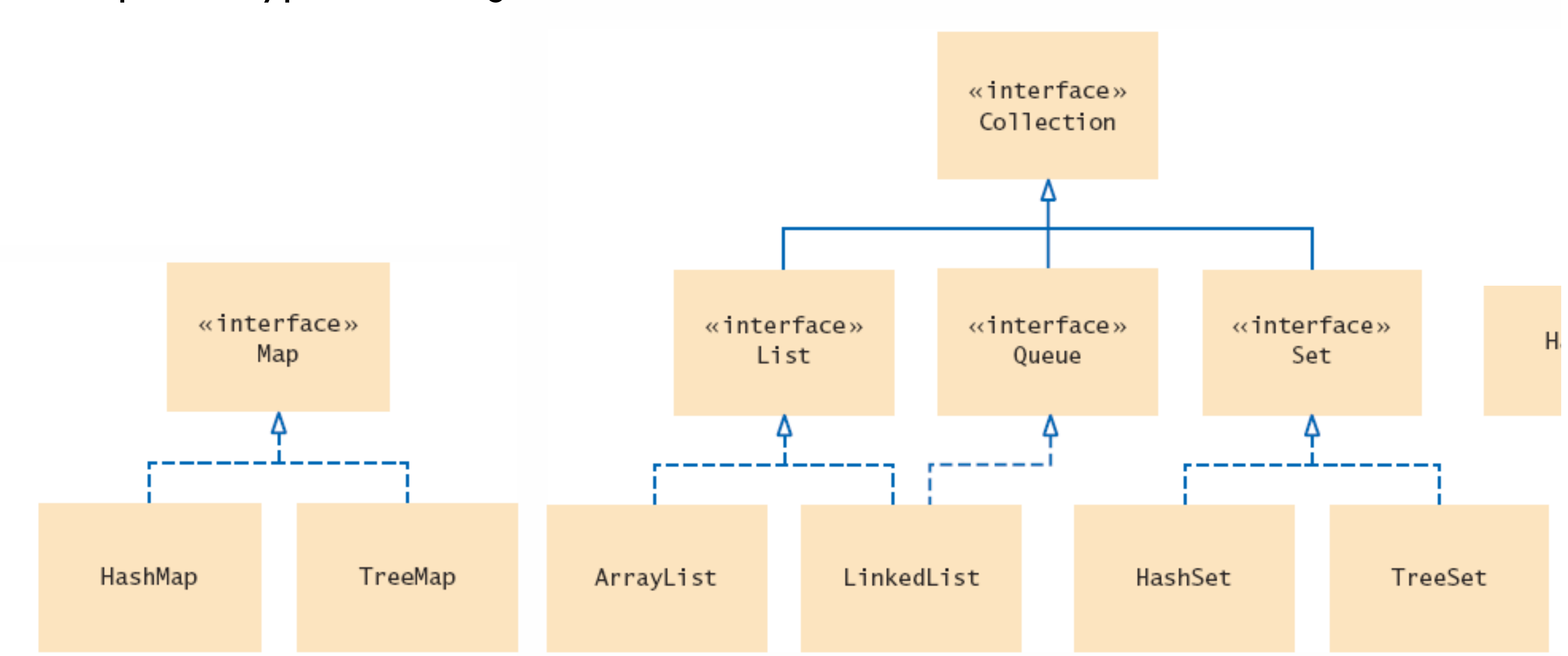
Java Collections Framework

- When you need to organize multiple objects in your program, you can place them into a ***collection***
- The `ArrayList` class that was introduced in Chapter 6 is one of many collection classes that the standard Java library supplies
- Each interface type is implemented by one or more classes

A collection groups together elements and allows them to be accessed and retrieved later

Collections Framework Diagram

- Each collection class implements an interface from a hierarchy
  - Each class is designed for a specific type of storage



Lists and Sets

A **list** is a collection that maintains the order of its elements.

- Ordered Lists

  - `ArrayList`
    - Stores a list of items in a dynamically sized array
    - Speedy access, but can spend time resizing
  - `LinkedList`
    - Allows speedy insertion and removal of items from the list
    - Can’t access “interior elements” as quickly

Lists and Sets

A **set** is an unordered collection of unique elements.

- Unordered Sets

  - HashSet
    - Uses hash tables for fast finding, adding, and removing
  - TreeSet
    - Uses a binary tree for fast finding, adding, and removing
- Slower “access” than ordered sets, faster “find”

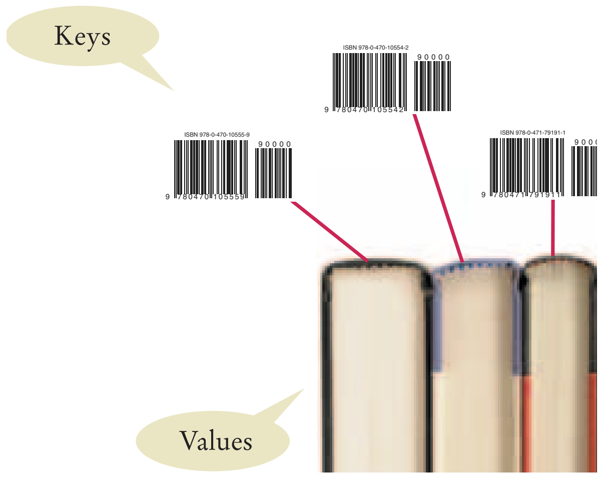
Stacks and Queues

- Another way of gaining efficiency in a collection reduce the number of operations available
- Two examples are:
  - Stack
    - Ordered, but can only add and remove elements at top
    - Like a “stack of heavy books”
  - Queue
    - Ordered, but can only add at end (tail) and remove from front (the head)
    - Example: A line of people waiting for a bank teller

Maps

A map keeps associations between key and value objects.

- A map stores keys, values, and the associations between them
  - Example:
  - Barcode keys and books



- Keys
  - Provides an easy way to represent an object
  - Like a numeric bar code
- Values
  - Actual object associated with the key

The Collection Interface (1)

- List, Queue and Set are specialized interfaces that inherit from the Collection interface
  - All share the following commonly used methods

Table 1 The Methods of the Collection Interface

`Collection<String> coll = new ArrayList<String>();`	The ArrayList class implements the Collection interface.
`coll = new TreeSet<String>()`	The TreeSet class (Section 15.3) also implements the Collection interface.
`int n = coll.size();`	Gets the size of the collection. n is now 0.
`coll.add("Harry");` `coll.add("Sally");`	Adds elements to the collection.
`String s = coll.toString();`	Returns a string with all elements in the collection. s is now "[Harry, Sally]"
`System.out.println(coll);`	Invokes the toString method and prints [Harry, Sally].

The Collection Interface (2)

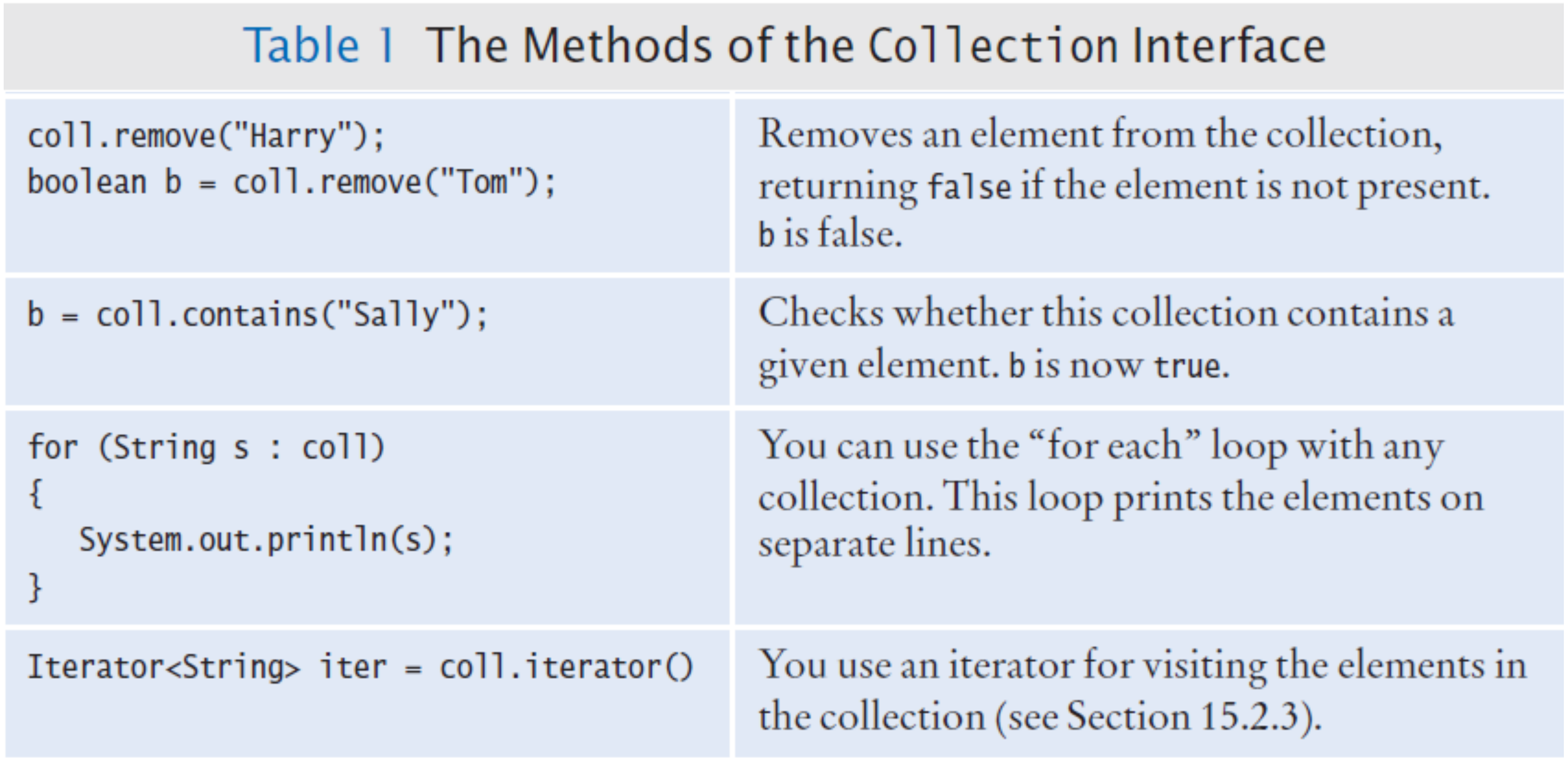


Table 1 The Methods of the Collection Interface

`coll.remove("Harry");`
`boolean b = coll.remove("Tom");`
	Removes an element from the collection, returning `false` if the element is not present. `b` is false.

`b = coll.contains("Sally");`	Checks whether this collection contains a given element. `b` is now `true`.

`for (String s : coll)`
`{`
`   System.out.println(s);`
`}`
	You can use the "for each" loop with any collection. This loop prints the elements on separate lines.

`Iterator<String> iter = coll.iterator()`	You use an iterator for visiting the elements in the collection (see Section 15.2.3).

Iterators

Iterators

- An **iterator** is an object for “iterating” or “traversing” a collection

- We’ve already used iterators

```
ArrayList<BankAccount> accounts = . . .;
double totalBalances = 0.0;
for (BankAccount a : accounts) {
   totalBalances += a.getBalance();
}
```

- This is only possible for collections
  - Because each has a method iterator()

Iterators

- Can use and access the iterator more directly
  - This was only way possible in older versions of Java

```
ArrayList<BankAccount> accounts = . . .;
double totalBalances = 0.0;
Iterator<BankAccount> iter = accounts.iterator();
while (iter.hasNext()) {
   BankAccount a = iter.next();
   totalBalances += a.getBalance();
}
```

- hasNext() checks if there are more elements

- next() returns the reference to the next object

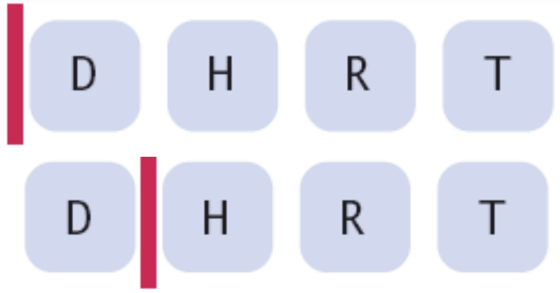
Using Iterators

- Think of an iterator as pointing **between** two elements

```
Iterator<String> iter = myList.iterator();
```

```
Initial iter location
```

```
String s = iter.next();
```



Iterator Interface

- An **iterator** is an interface
- Only requires three methods
  - hasNext() : “are there more elements”
  - next() : “return position to next element”
  - remove() : “remove the last element”

Topics

1. What are collections?
2. Basics of Lists, Sets, and Maps
3. The Collection Interface
4. Iterators