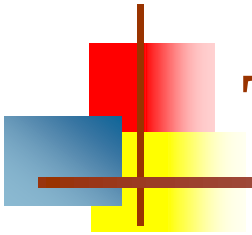
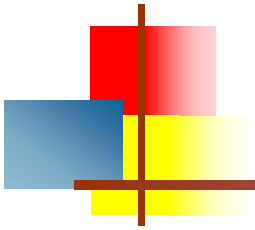


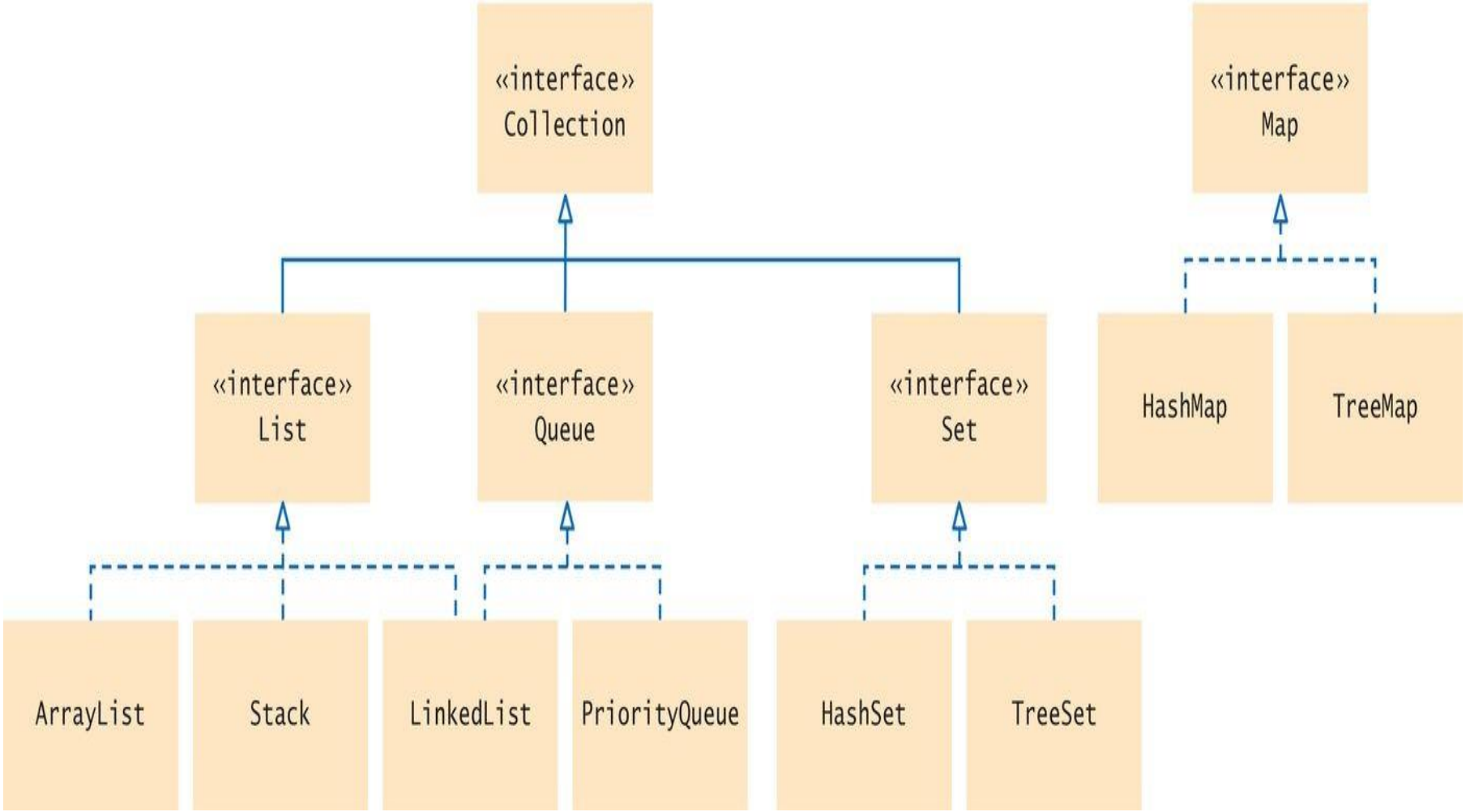
The Java Collections Framework

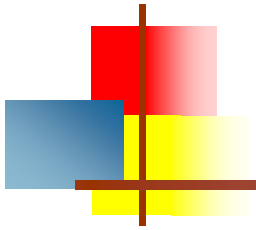


Goals

- To learn how to use the collection classes supplied in the Java library
- To use iterators to traverse collections
- To choose appropriate collections for solving programming problems
- To study applications of stacks and queues

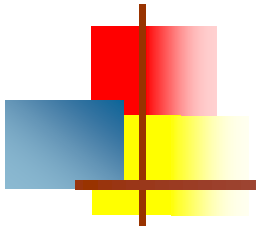
An Overview of the Collections Framework





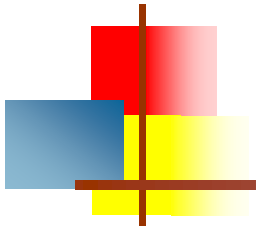
Collections Framework

- A collection groups together elements and allows them to be retrieved later.
- Java collections framework: a hierarchy of interface types and classes for collecting objects.
  - Each interface type is implemented by one or more classes
- The Collection interface is at the root
  - All Collection class implement this interface
  - So all have a common set of methods



Collections Framework

- Unified architecture for representing and manipulating collections.
- A collections framework contains three things
  - Interfaces
  - Implementations
  - Algorithms



Collection Interface

- Defines fundamental methods
  - `int size();`
  - `boolean isEmpty();`
  - `boolean contains(Object element);`
  - `boolean add(Object element);`
  - `boolean remove(Object element);`
  - `Iterator iterator();`
- These methods are enough to define the basic behavior of a collection
- Provides an Iterator to step through the elements in the Collection

Collection Interface

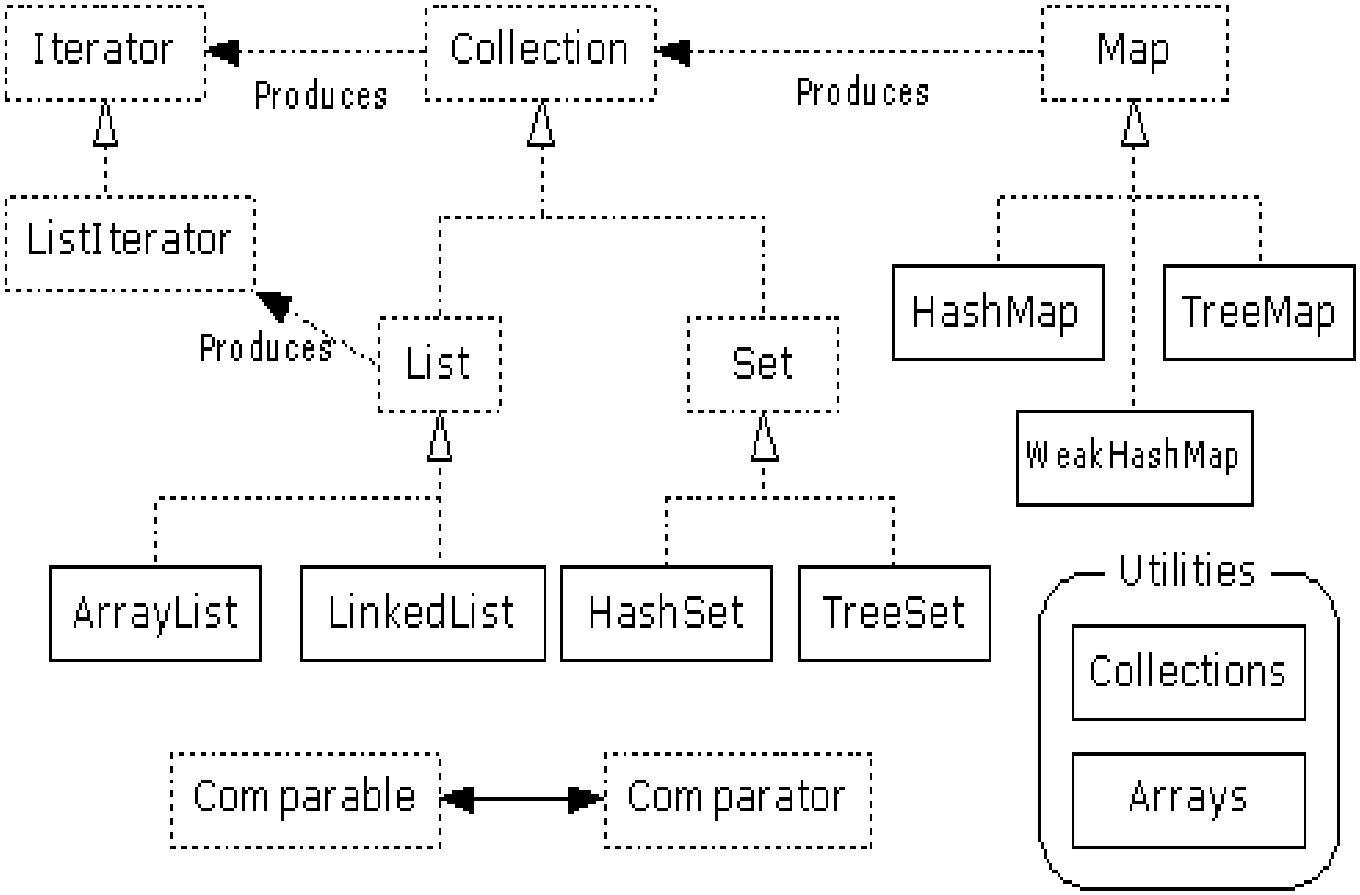
Table 1 The Methods of the `Collection` Interface

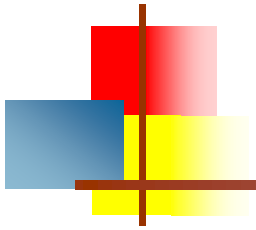
`Collection<String> coll = new ArrayList<>();`	The `ArrayList` class implements the `Collection` interface.
`coll = new TreeSet<>();`	The `TreeSet` class (Section 15.3) also implements the `Collection` interface.
`int n = coll.size();`	Gets the size of the collection. `n` is now `0`.
`coll.add("Harry");` `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]`.
`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).

Collection Interface





Iterator Interface

- Defines three fundamental methods
  - **`boolean hasNext()`** (returns true if the iterator has any more items)
  - **`Object next()`** (returns the next item in the iterator)
  - **`void remove()`**
- These three methods provide access to the contents of the collection
- An Iterator knows position within collection
- Each call to next() “reads” an element from the collection
  - Then you can use it or remove it

Iterator Interface

- Note that the next pointer is out-of-bounds after the last call to *next()*
  - another call to *next()* will result in exception
- Should always call *hasNext()* before calling *next()*

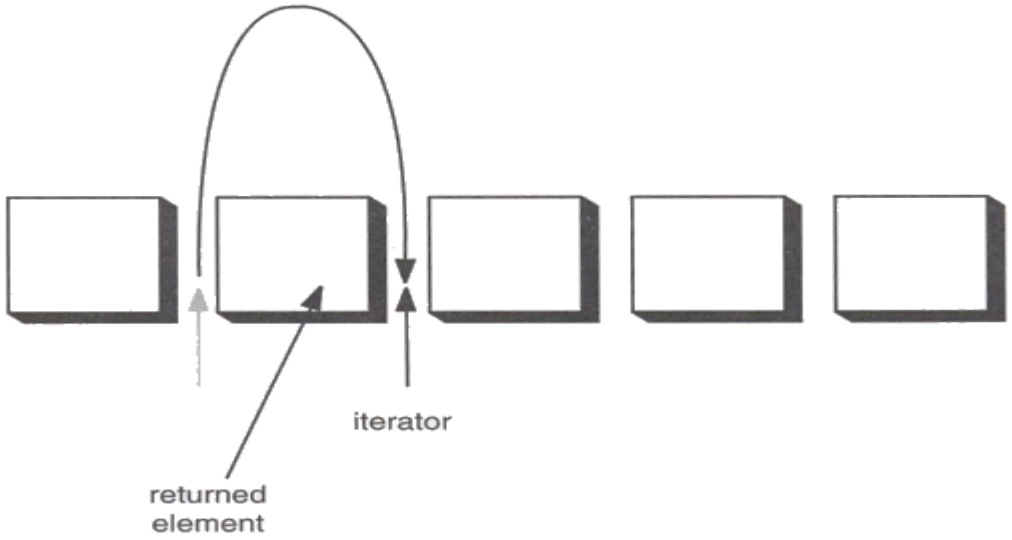
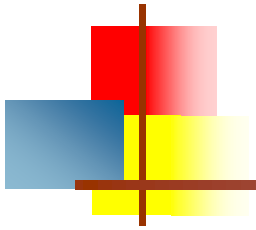


Figure 2–3: Advancing an iterator

Example - SimpleCollection

```
public class SimpleCollection  {
  public static void main(String[] args) {
    Collection c;
    c = new ArrayList();
    System.out.println(c.getClass().getName());
    for (int i=1; i <= 10; i++) {
           c.add(i + " * " + i + " = "+i*i);
    }
    Iterator iter = c.iterator();
    while (iter.hasNext())
           System.out.println(iter.next());
  }
}
```



Limitations of Iterator

- You can only move towards forward direction.
  - No previous
- Iterator can only perform read and remove operations.
  - No replacement