Collection与迭代器

杨亮

数组

声明

```
dataType[] arrayRefVar;    // 首选的方法
dataType arrayRefVar[];  // 效果相同，但不是首选方法
```

创建

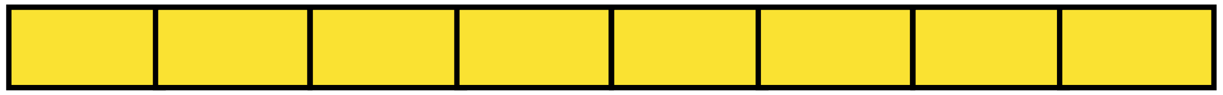
```
arrayRefVar = new dataType[arraySize];
```

声明 + 创建

```
dataType[] arrayRefVar = new dataType[arraySize];
dataType[] arrayRefVar = {value0, value1, ..., valuek};
```

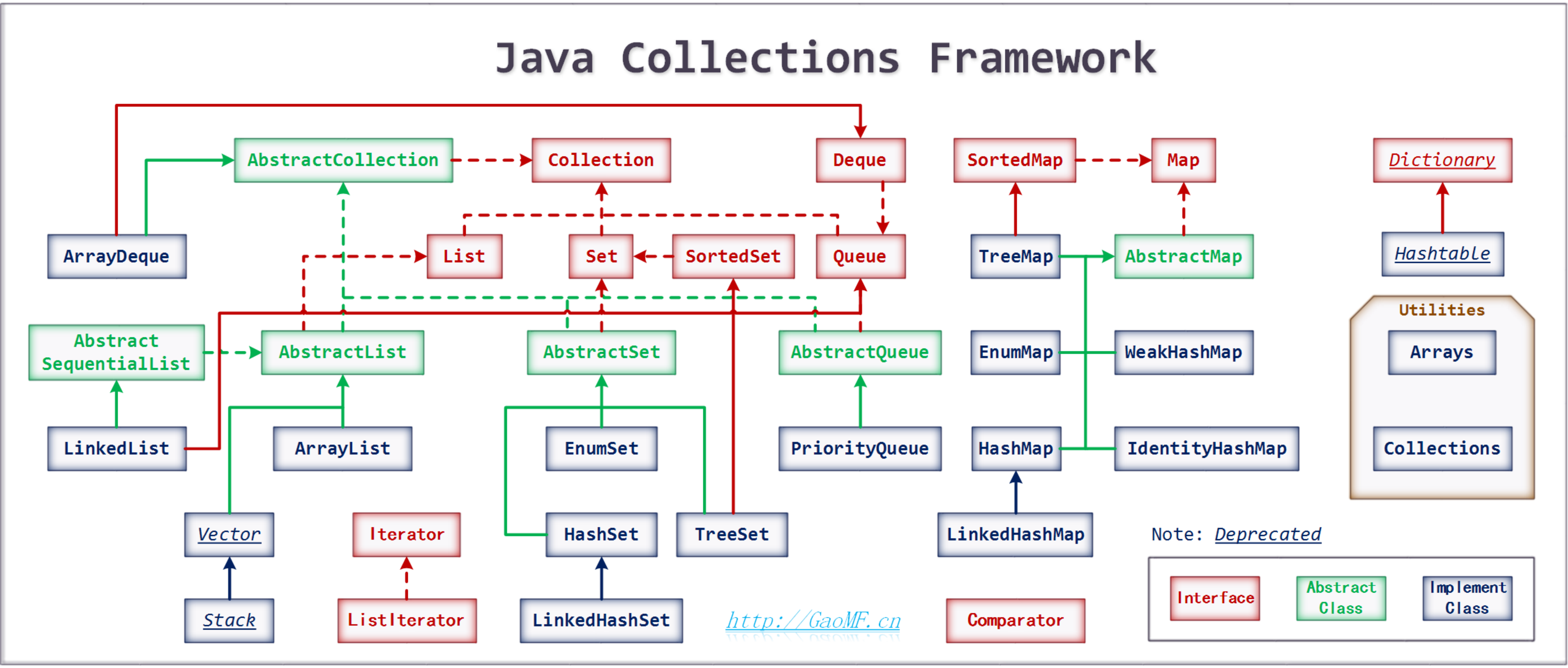
遍历

```
public static void main(String[] args) {
    double[] myList = {1.9, 2.9, 3.4, 3.5};

    // 打印所有数组元素
    for (double element: myList) {
        System.out.println(element);
    }
}
```



长度固定，不支持动态增长

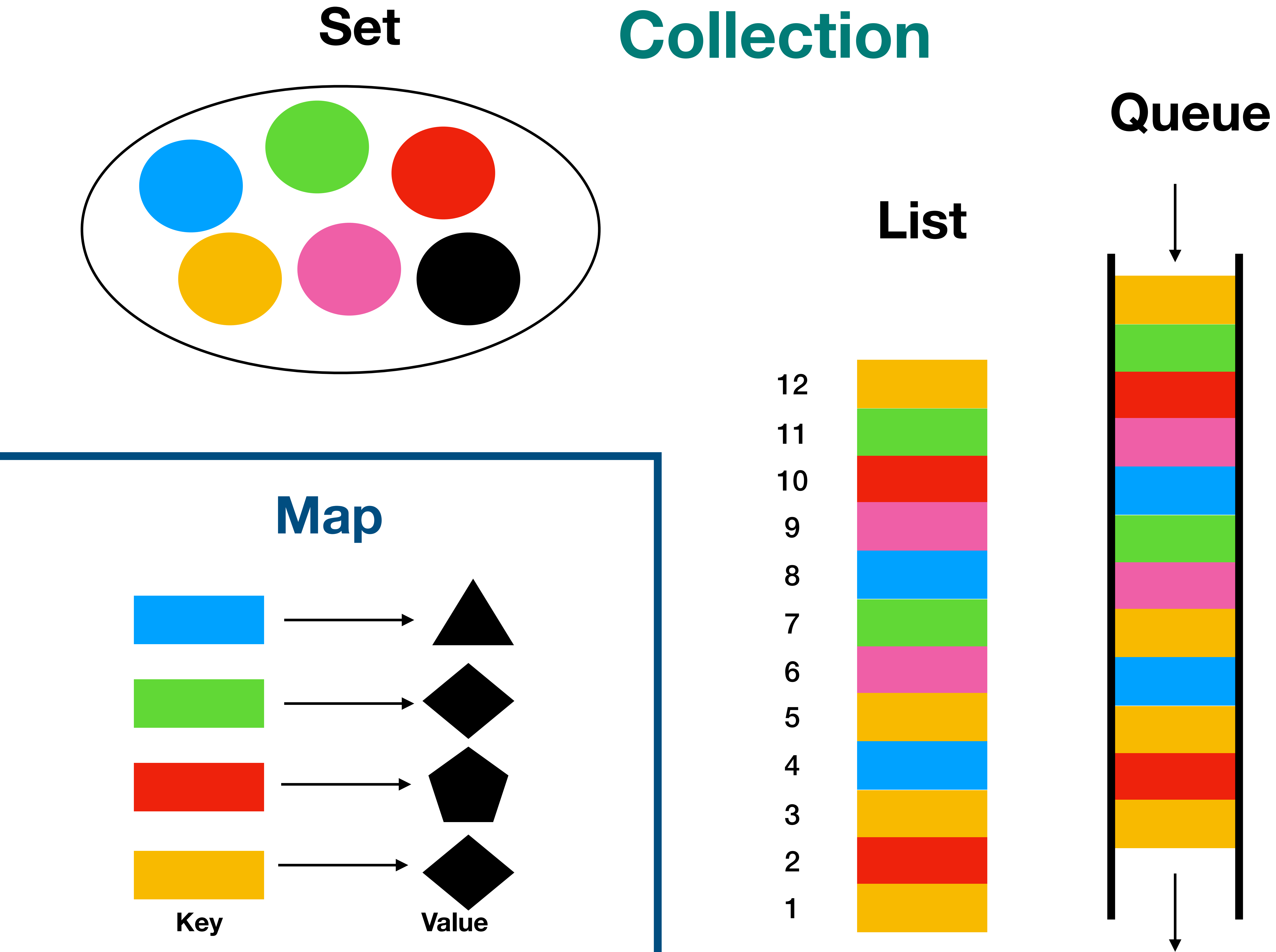
简单线性结构，不支持复杂操作



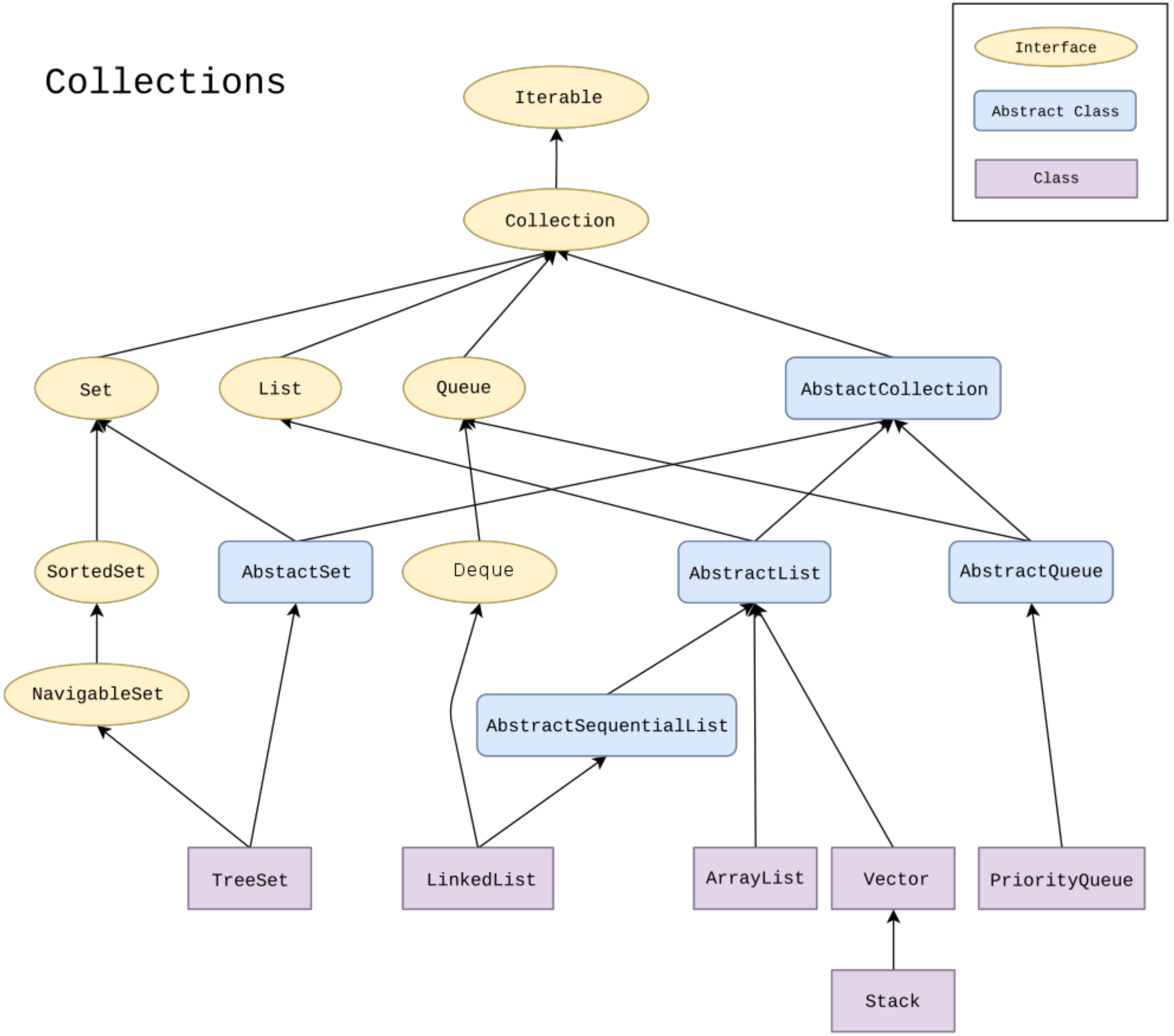
Interface	Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table + Linked List
Set	HashSet	–	TreeSet	–	LinkedHashSet
List	–	ArrayList	–	LinkedList	–
Deque	–	ArrayDeque	–	LinkedList	–
Map	HashMap	–	TreeMap	–	LinkedHashMap

泛型 + 接口 + 抽象类 + 多态

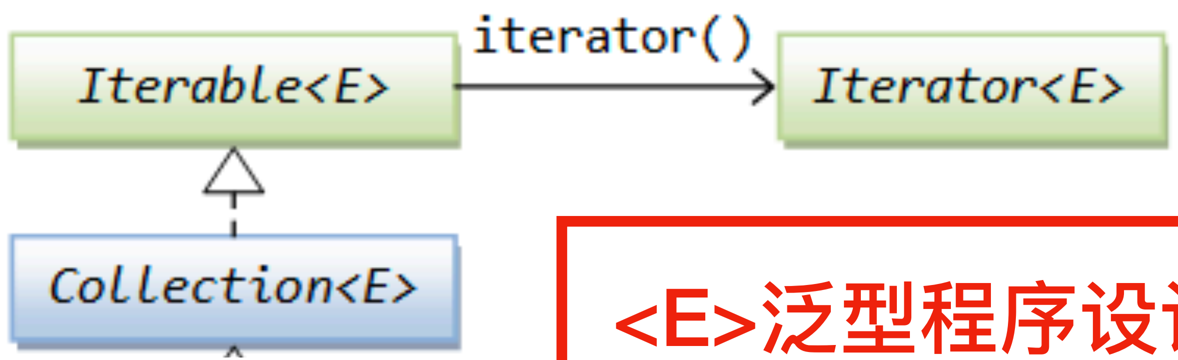
Collection



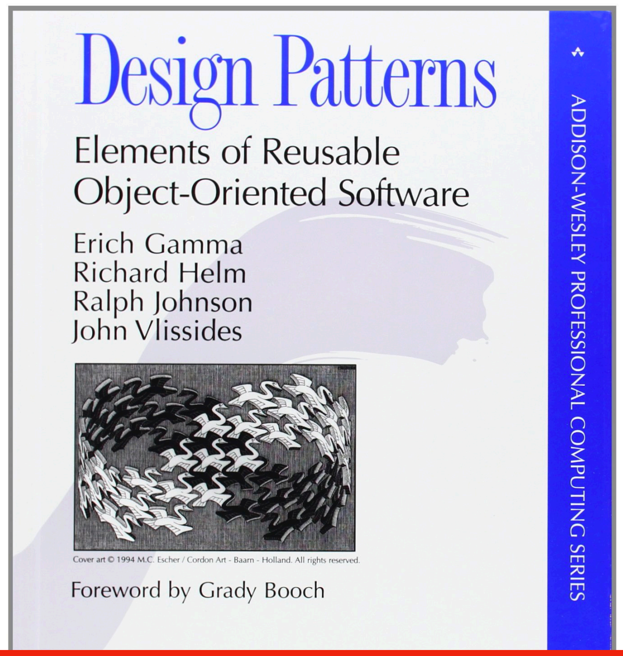
Collections



迭代器

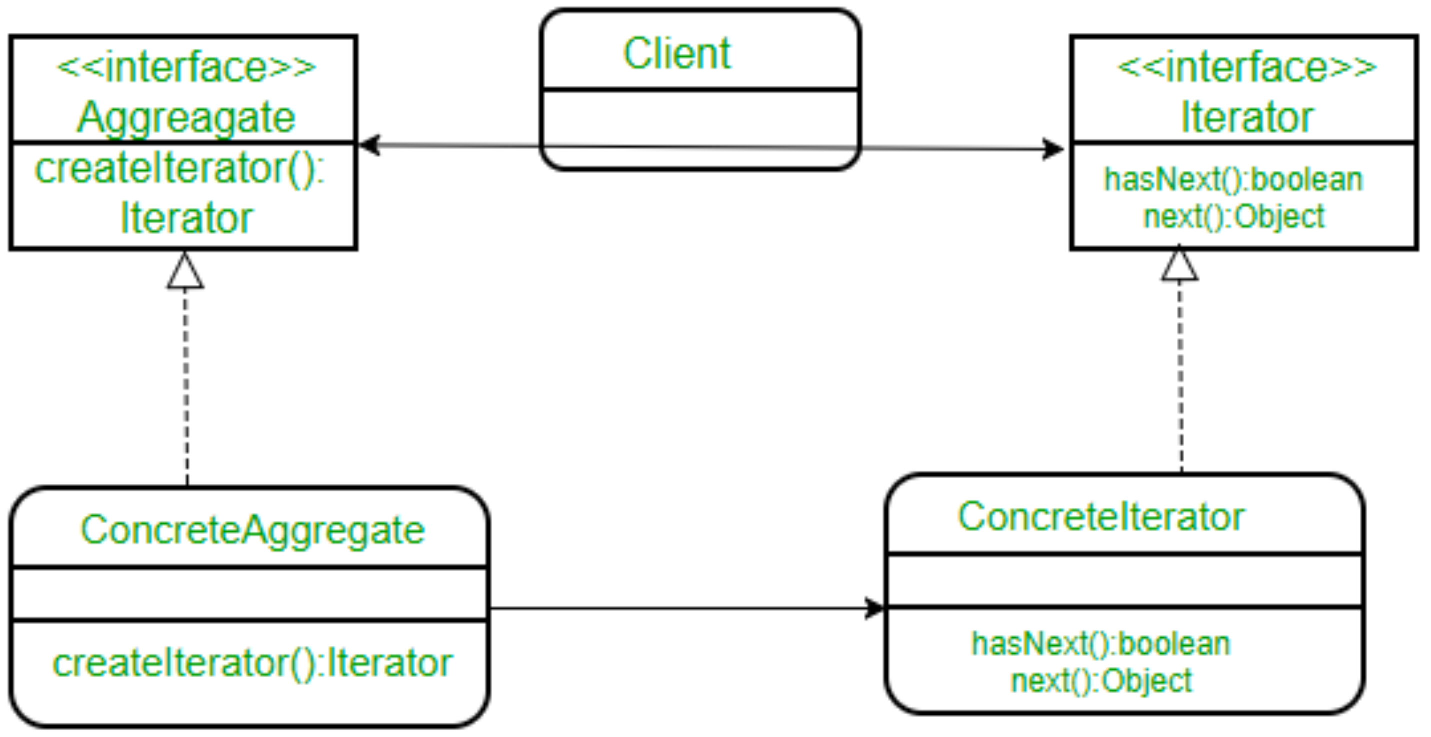


<E>泛型程序设计


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

迭代器模式

```
public interface Iterable<T> {
  Iterator<T> iterator();
}

public interface Iterator<E> {
  boolean hasNext();
  E next();
  void remove();
}
```



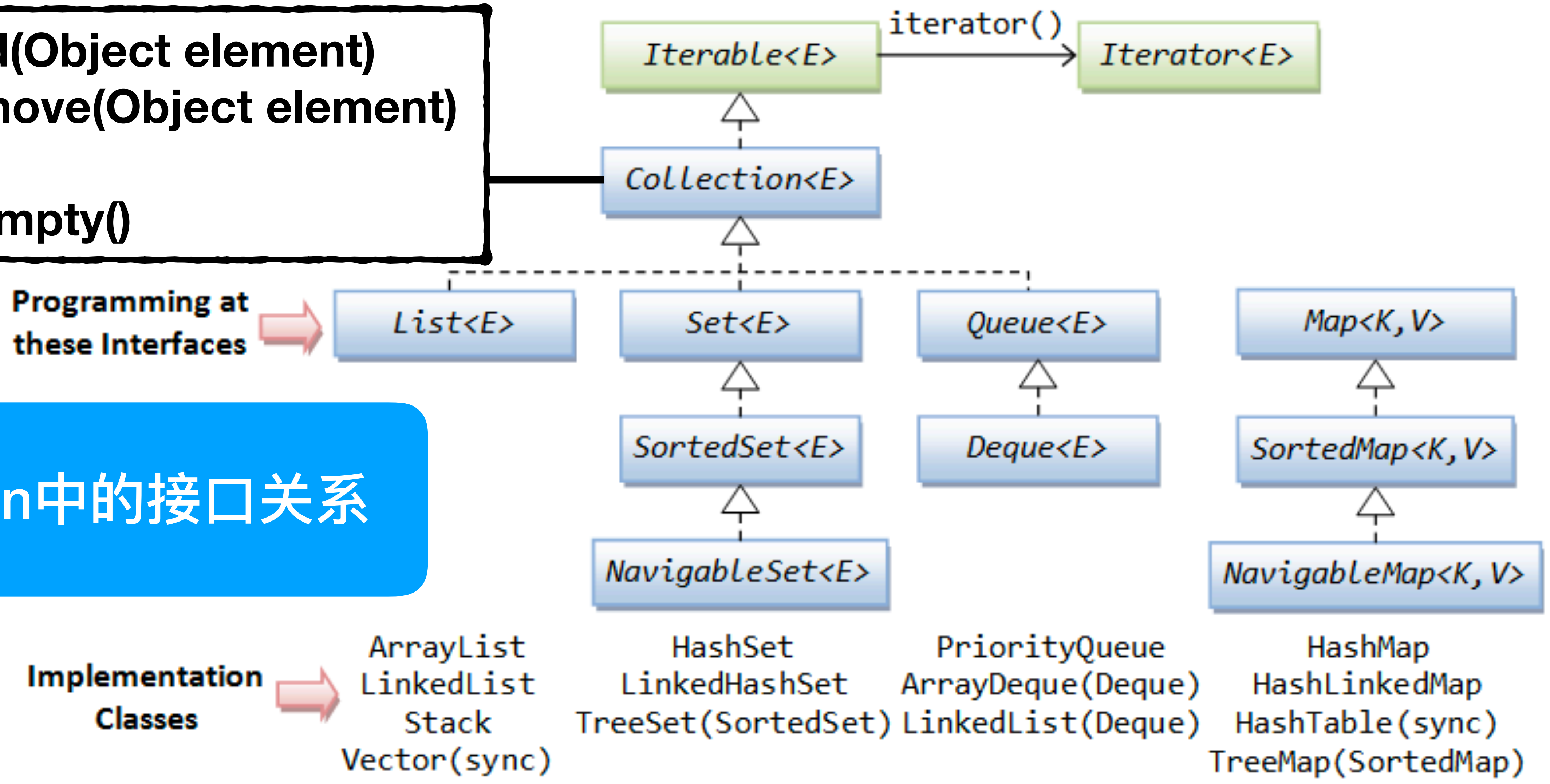
为什么要用两个接口?

```java
public interface Iterator<E> {
  boolean hasNext();
  E next();
  void remove();
}
```

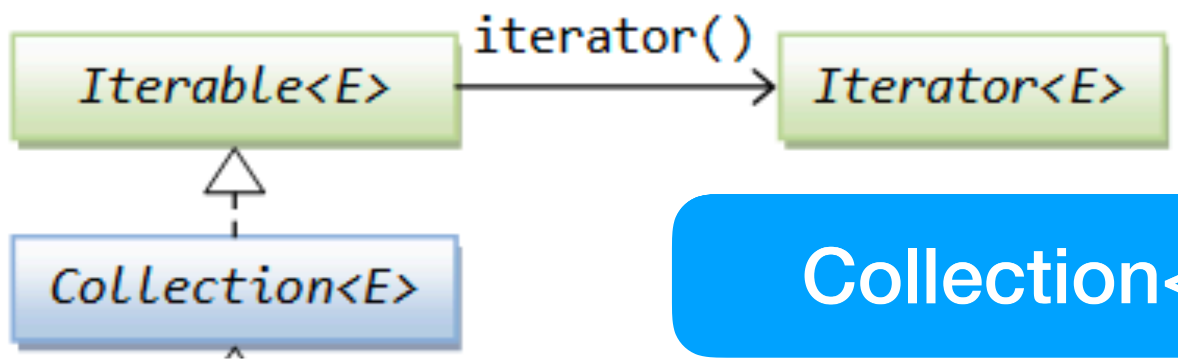
```java
public interface Iterable<T> {
  Iterator<T> iterator();
}
```

Collection中的接口关系

boolean add(Object element)
boolean remove(Object element)
int size()
boolean isEmpty()



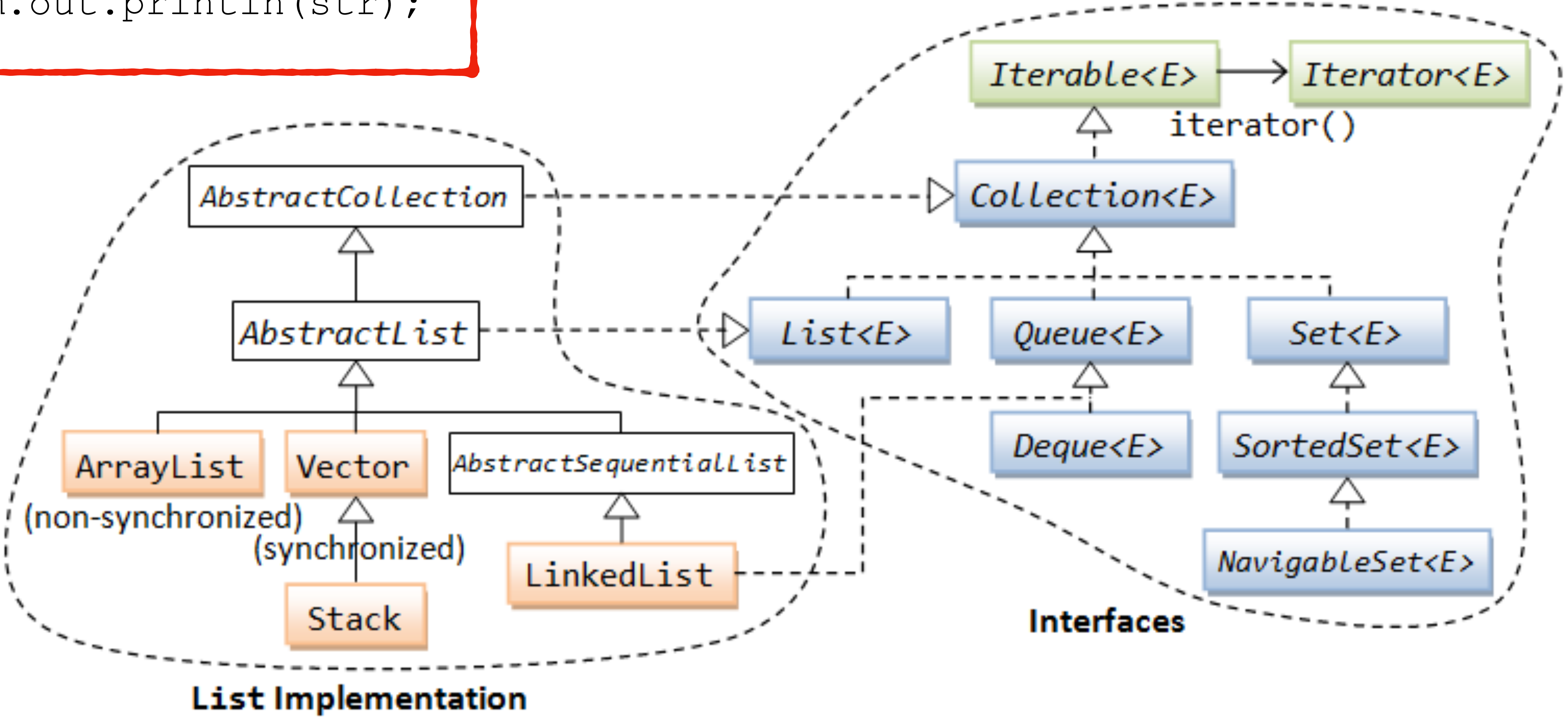
Interface	Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table + Linked List
Set	HashSet	–	TreeSet	–	LinkedHashSet
List	–	ArrayList	–	LinkedList	–
Deque	–	ArrayDeque	–	LinkedList	–
Map	HashMap	–	TreeMap	–	LinkedHashMap

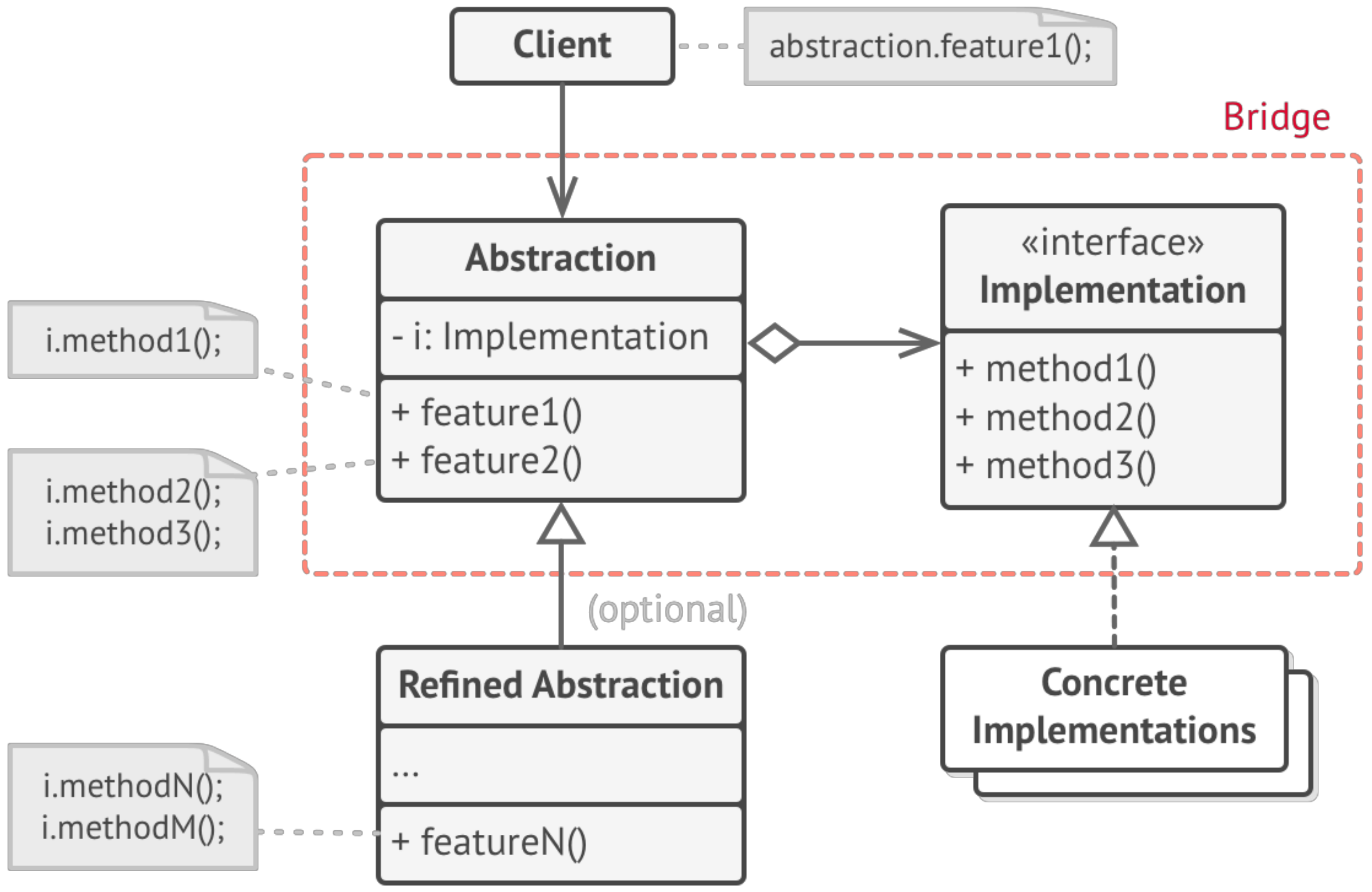


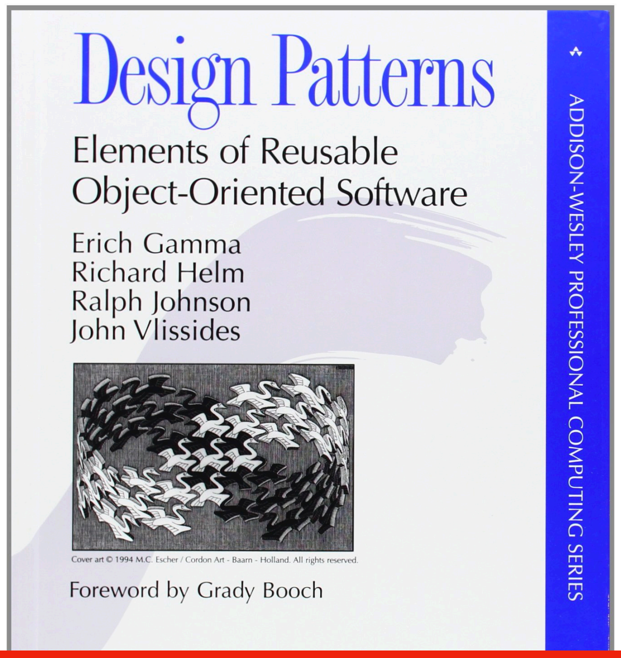
Collection<E>中定义的抽象函数

```
// Basic Operations
int size()                              // Returns the number of elements of this Collection
void clear()                            // Removes all the elements of this Collection
boolean isEmpty()                       // Returns true if there is no element in this Collection
boolean add(E element)                  // Ensures that this Collection contains the given element
boolean remove(Object element)          // Removes the given element, if present
boolean contains(Object element)        // Returns true if this Collection contains the given element

// Bulk Operations with another Collection
boolean containsAll(Collection<?> c)          // Collection of any "unknown" object
boolean addAll(Collection<? extends E> c)     // Collection of E or its sub-types
boolean removeAll(Collection<?> c)
boolean retainAll(Collection<?> c)

// Comparison - Objects that are equal shall have the same hashCode
boolean equals(Object o)
int hashCode()

// Array Operations
Object[] toArray()          // Convert to an Object array
<T> T[] toArray(T[] a)      // Convert to an array of the given type T
```

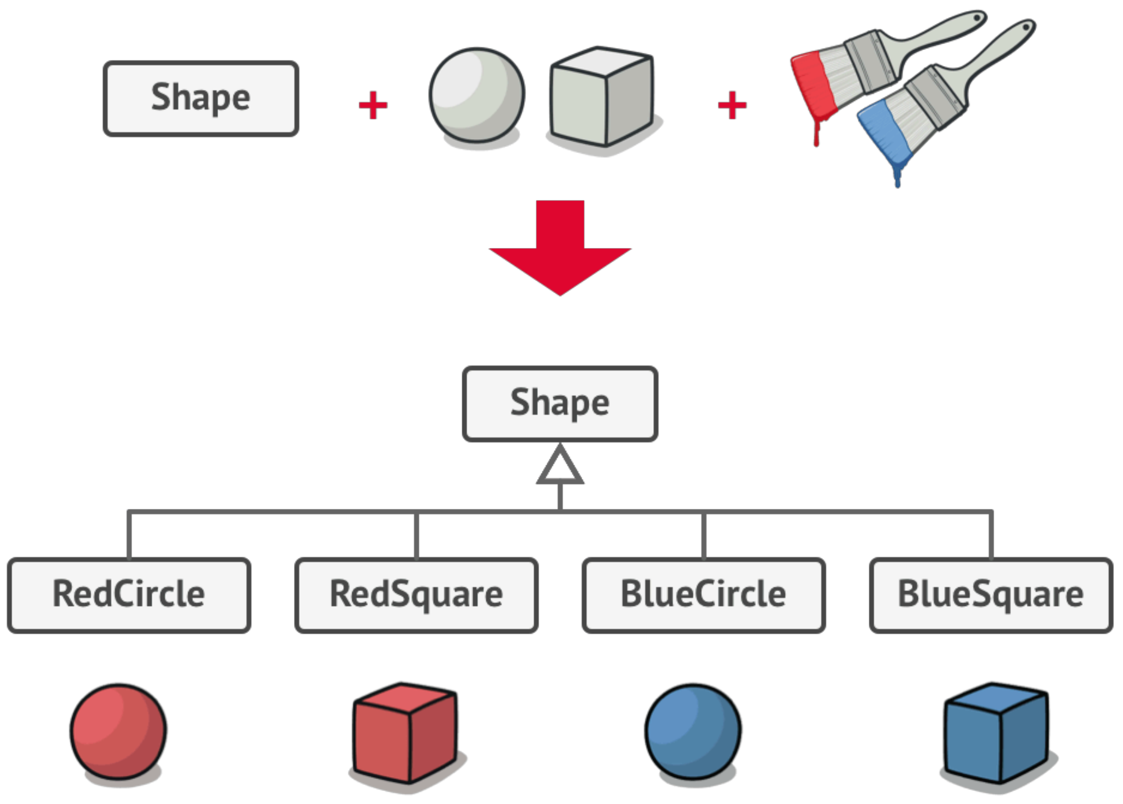
```
public class ArrayListPostJDK15Test {
   public static void main(String[] args) {
      List<String> lst = new ArrayList<String>();  // Inform compiler about the type
      lst.add("alpha");         // compiler checks if argument's type is String
      lst.add("beta");
      lst.add("charlie");
      System.out.println(lst);  // [alpha, beta, charlie]

      Iterator<String> iter = lst.iterator();   // Iterator of Strings
      while (iter.hasNext()) {
         String str = iter.next();  // compiler inserts downcast operator
         System.out.println(str);
      }

      for (String str : lst) {
         System.out.println(str);
      }
   }
}
```

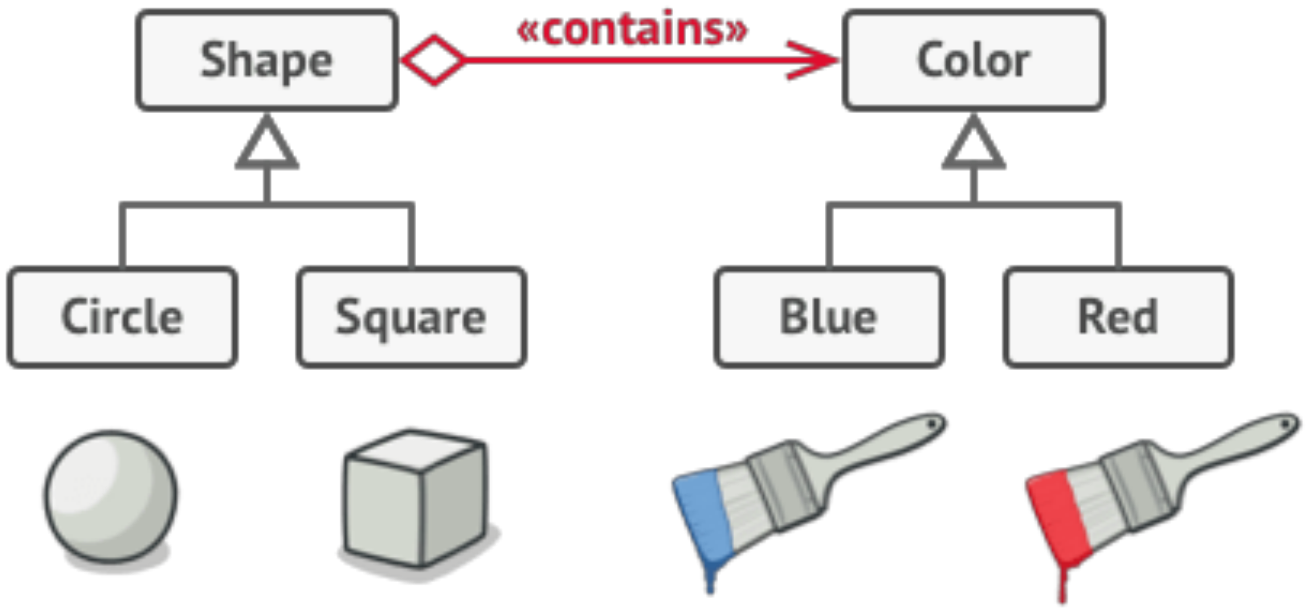


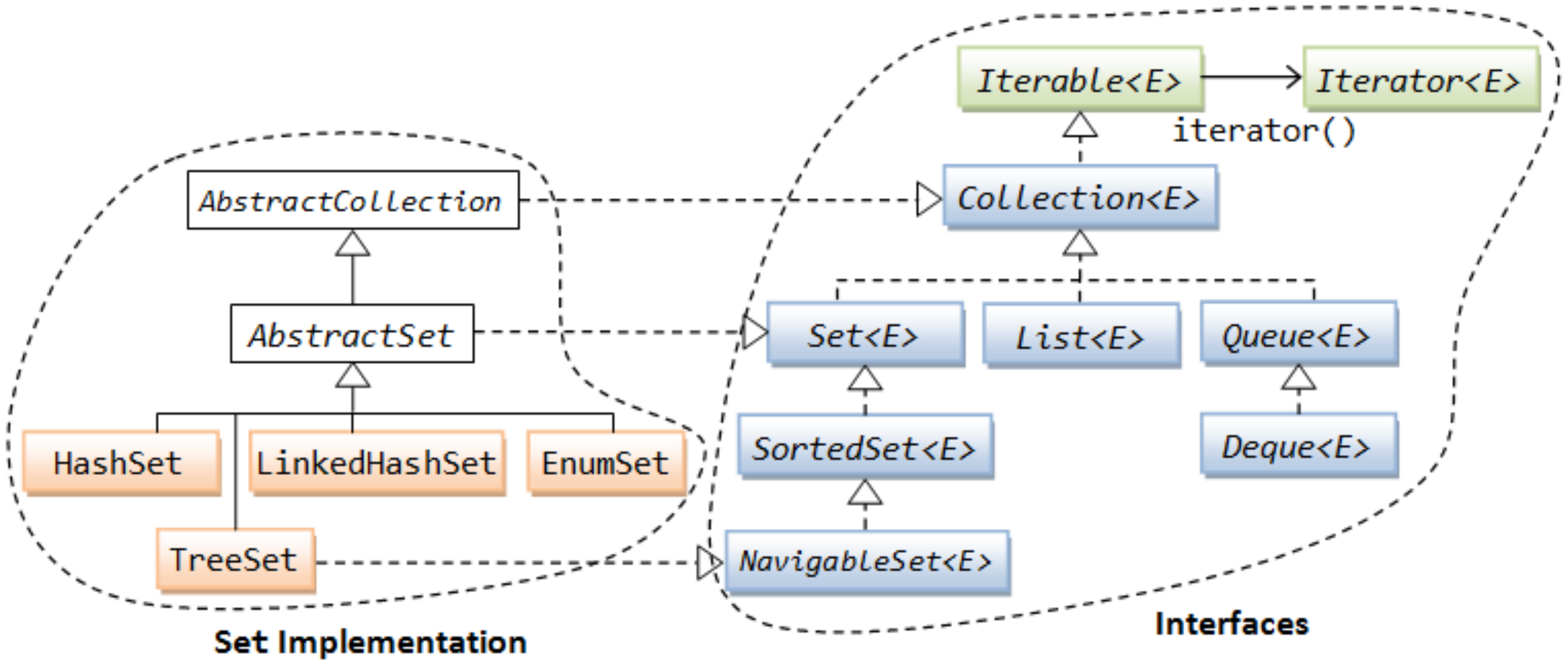

Client
abstraction.feature1();
Bridge
Abstraction
- i: Implementation
+ feature1()
+ feature2()
«interface»
Implementation
+ method1()
+ method2()
+ method3()
i.method1();
i.method2();
i.method3();
(optional)
Refined Abstraction
...
+ featureN()
Concrete Implementations
i.methodN();
i.methodM();

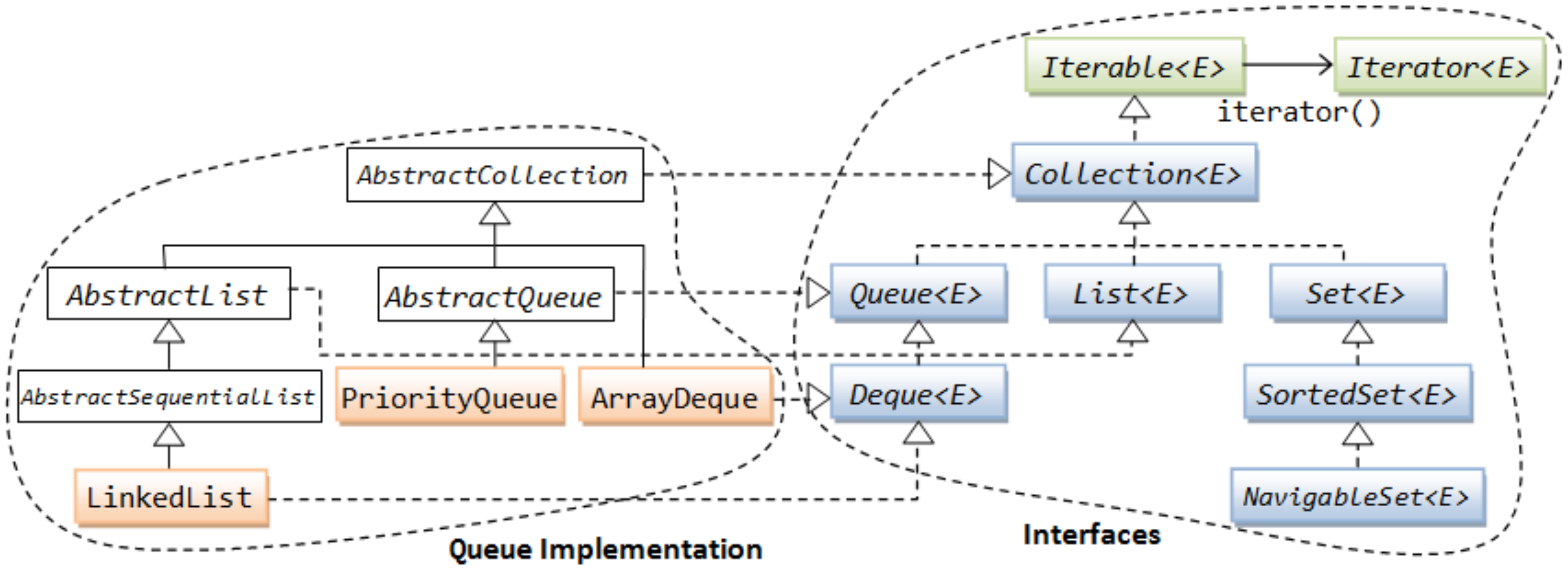

Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

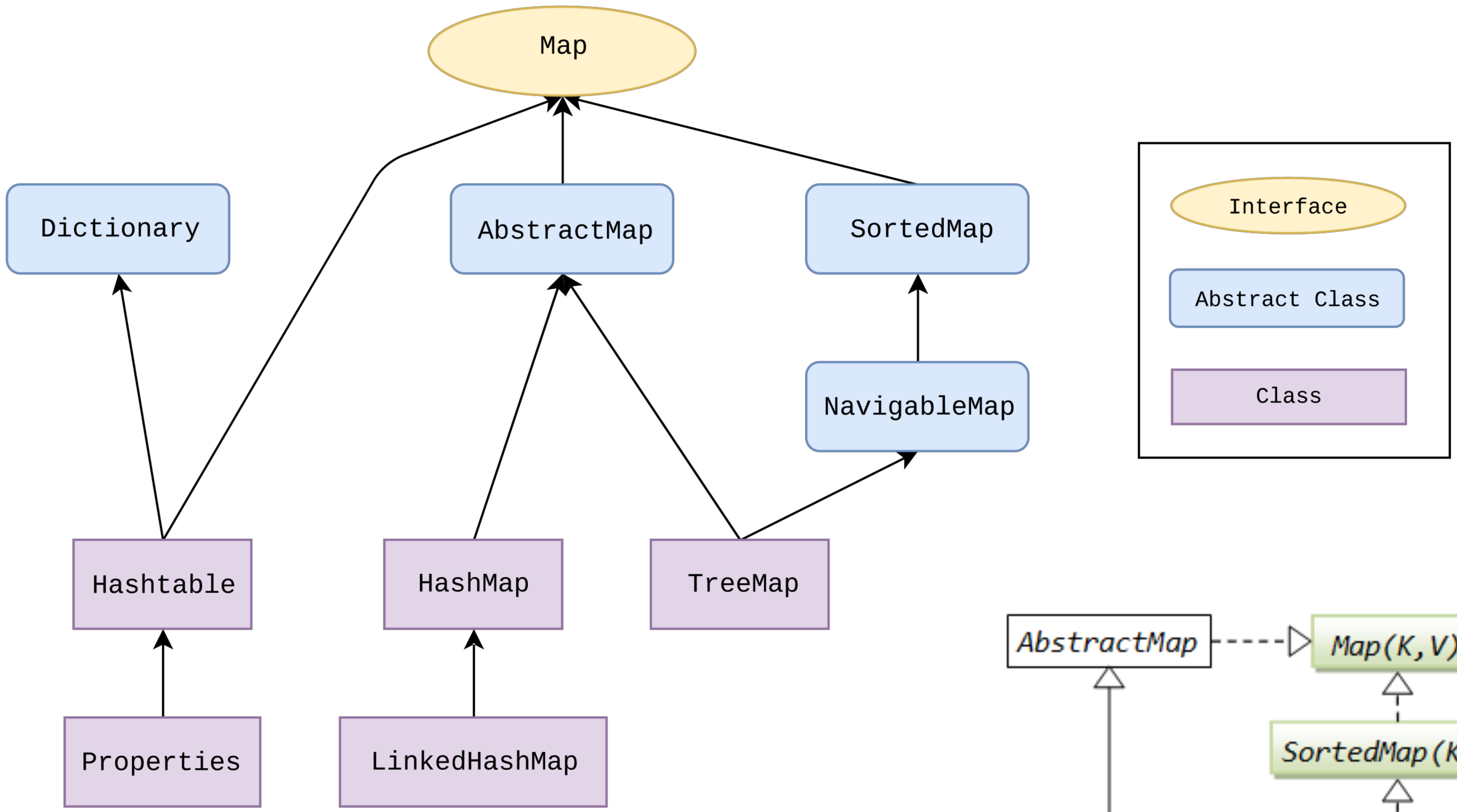
桥接模式

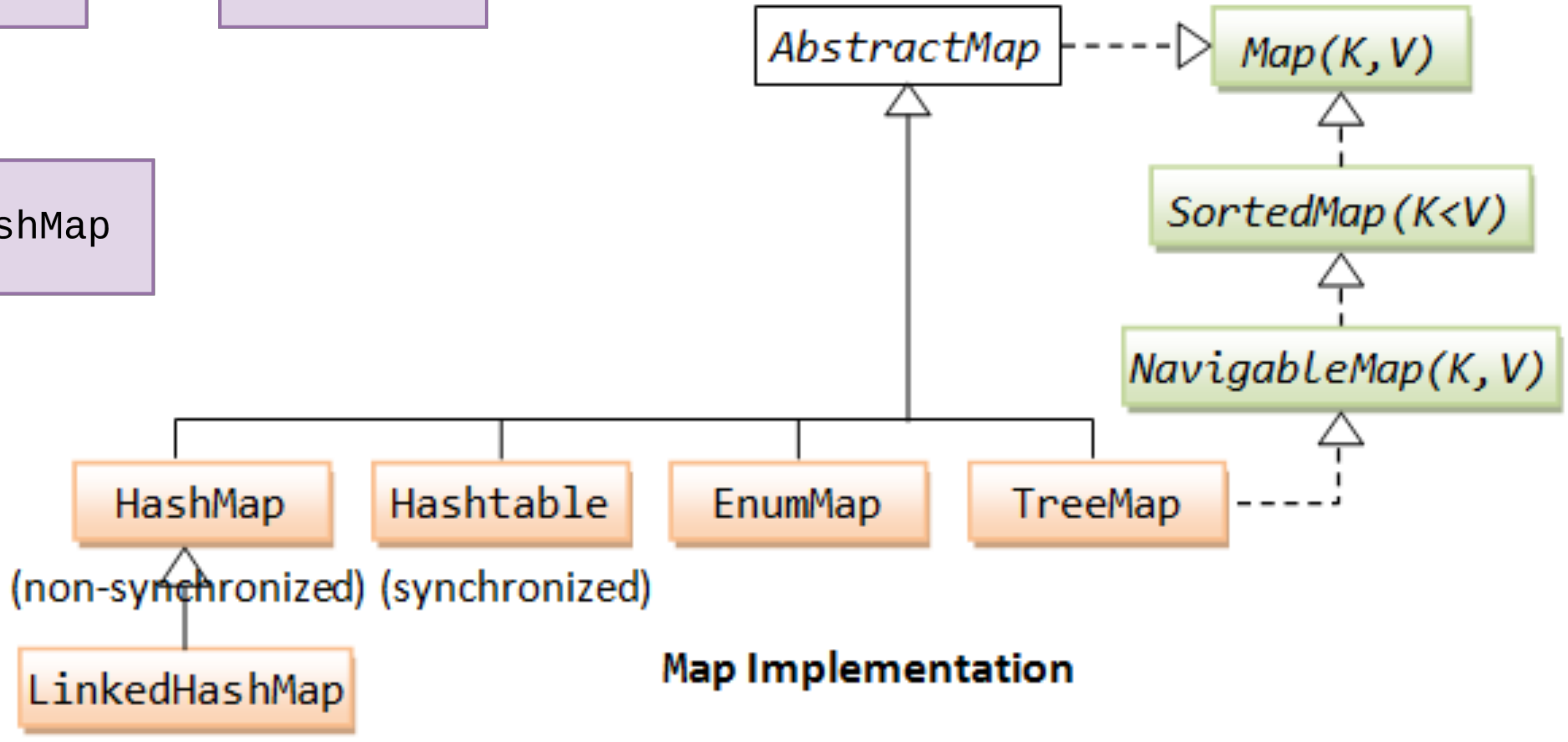

Shape
+
+
Shape
RedCircle
RedSquare
BlueCircle
BlueSquare


Shape
«contains»
Color
Circle
Square
Blue
Red

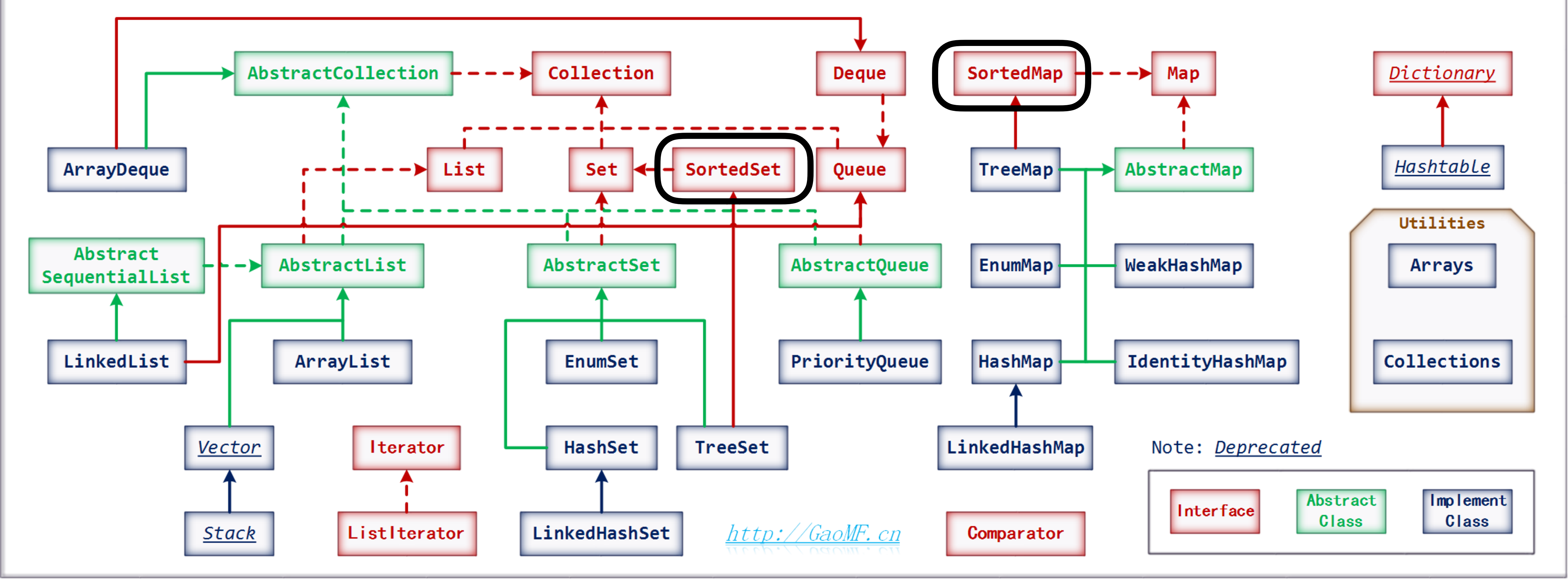

Iterable<E>
Iterator<E>
iterator()
Collection<E>
AbstractCollection
AbstractSet
Set<E>
List<E>
Queue<E>
SortedSet<E>
Deque<E>
NavigableSet<E>
HashSet
LinkedHashSet
EnumSet
TreeSet
Set Implementation
Interfaces


Iterable<E>
Iterator<E>
iterator()
AbstractCollection
Collection<E>
AbstractList
AbstractQueue
Queue<E>
List<E>
Set<E>
AbstractSequentialList
PriorityQueue
ArrayDeque
Deque<E>
SortedSet<E>
LinkedList
NavigableSet<E>
Queue Implementation
Interfaces





Map Implementation



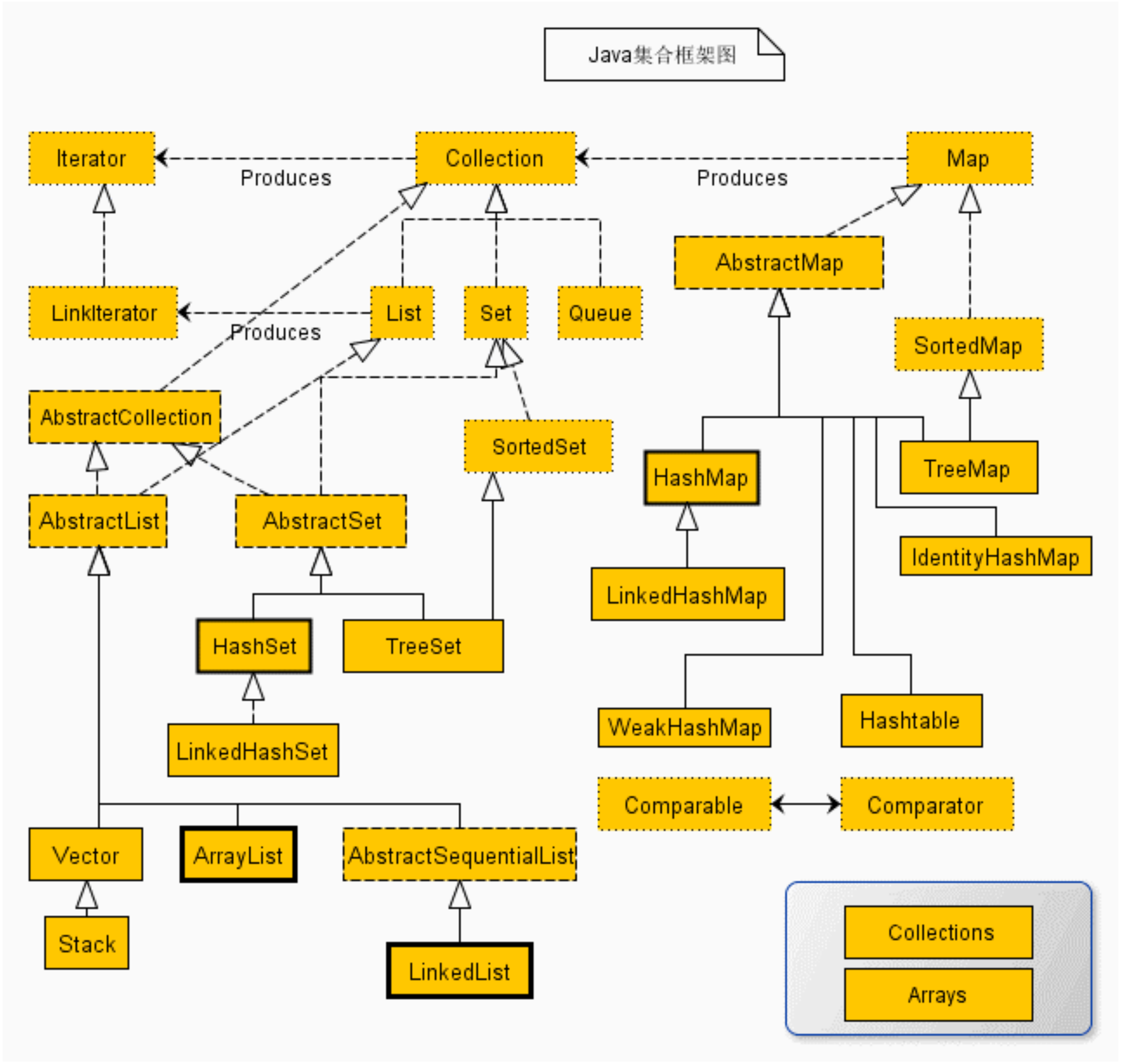
排序接口

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

排序器接口

```
public interface Comparator<T> {
    int compare(T o1, T o2);
    boolean equals(Object obj);
}
```

```java
private static class Person implements Comparable<Person>{
    int age;
    String name;
    @Override
    public int compareTo(Person person) {
        return name.compareTo(person.name);
    }
}

ArrayList<Person> list = new ArrayList<Person>();
list.add(new Person("ccc", 20));
list.add(new Person("AAA", 30));
list.add(new Person("bbb", 10));
list.add(new Person("ddd", 40));

Collections.sort(list);
private static class AscAgeComparator implements Comparator<Person> {
    @Override
    public int compare(Person p1, Person p2) {
        return p1.getAge() - p2.getAge();
    }
}

private static class DescAgeComparator implements Comparator<Person> {
    @Override
    public int compare(Person p1, Person p2) {
        return p2.getAge() - p1.getAge();
    }
}

Collections.sort(list, new AscAgeComparator());
Collections.sort(list, new DescAgeComparator());
```


Java集合框架图
Iterator
Collection
Map
Produces
Produces
LinkIterator
List
Set
Queue
AbstractMap
Produces
SortedMap
AbstractCollection
SortedSet
HashMap
TreeMap
AbstractList
AbstractSet
IdentityHashMap
LinkedHashMap
HashSet
TreeSet
LinkedHashSet
WeakHashMap
Hashtable
Comparable
Comparator
Vector
ArrayList
AbstractSequentialList
Collections
Stack
LinkedList
Arrays