Collection Framework

- **Characteristics**
  - Since JDK 1.2
  - Small API, not comparable with STL (C++)
    - Few interfaces
    - Few methods per interface
  - JDK 1.5 (Java 5): Generics
  - JDK 1.8 (Java 8): Support for Lambda expressions

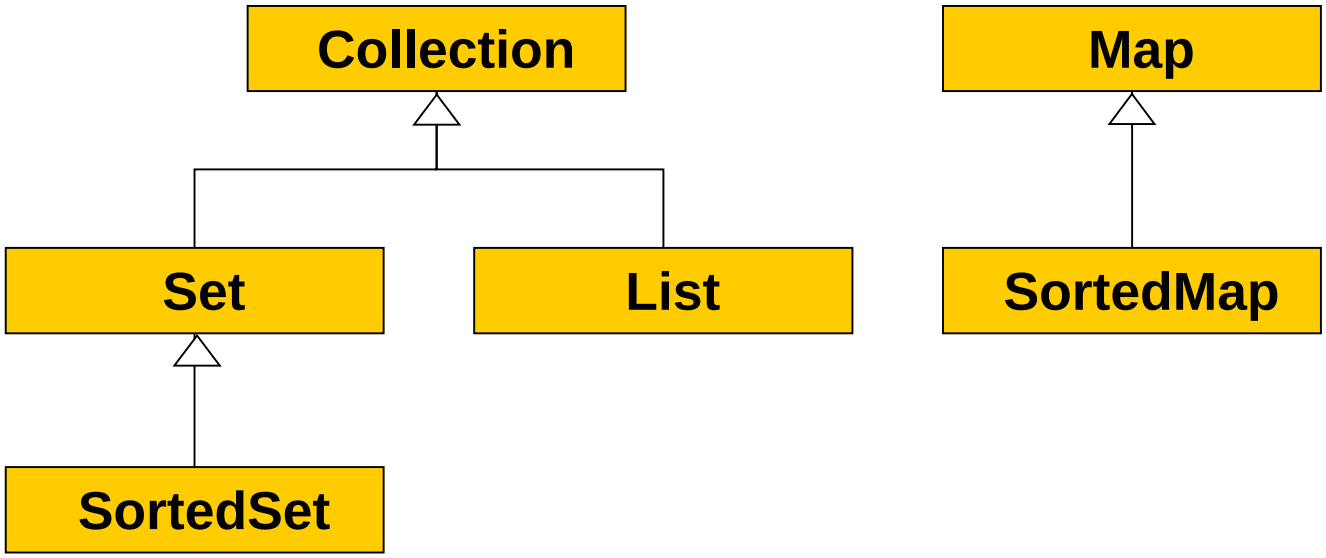
- **Documentation**
  - http://docs.oracle.com/javase/8/docs/technotes/guides/collections/
  - Tutorial: http://docs.oracle.com/javase/tutorial/collections/
  - API Docu: https://docs.oracle.com/javase/10/docs/api/

Collection Framework

- **Definitions**
  - **Collection** =
    - Object that groups multiple elements into a single unit (sometimes called a container)
    - Collections are used to store, retrieve and manipulate data, and to transmit data from one method to another
  - **Framework** =
    - Set of interfaces *user view*
      - Set<E>, List<E>, Map<K, V>
    - Concrete implementations of the interfaces *implementation view*
      - TreeSet<E>, ArrayList<E>, LinkedList<E>, HashMap<K, V>
    - Algorithms working on the interfaces *generic algorithms*
      - Collections.sort, Collections.binarySearch

Interfaces

- **Small set of interfaces**
  - No special interfaces for immutable collections (add/remove disabled)
  - No special interfaces for extensible only collections (remove disabled)
  - No special interfaces for collections which accept null objects



Collection

```
interface Collection<E> extends Iterable<E> {
   int          size();
   boolean      isEmpty();
   boolean      contains(Object x);
   boolean      containsAll(Collection<?> c);
   boolean      add(E x);                              // optional
   boolean      addAll(Collection<? extends E> c);     // optional
   boolean      remove(Object x);                      // optional
   boolean      removeAll(Collection<?> c);            // optional
   boolean      retainAll(Collection<?> c);            // optional
   void         clear();                               // optional
   Object[]     toArray();
   <T> T[]      toArray(T[] a);
   Iterator<E> iterator();
}
```

- optional means, that these methods may throw a `UnsupprtedOperationException`

Iterator

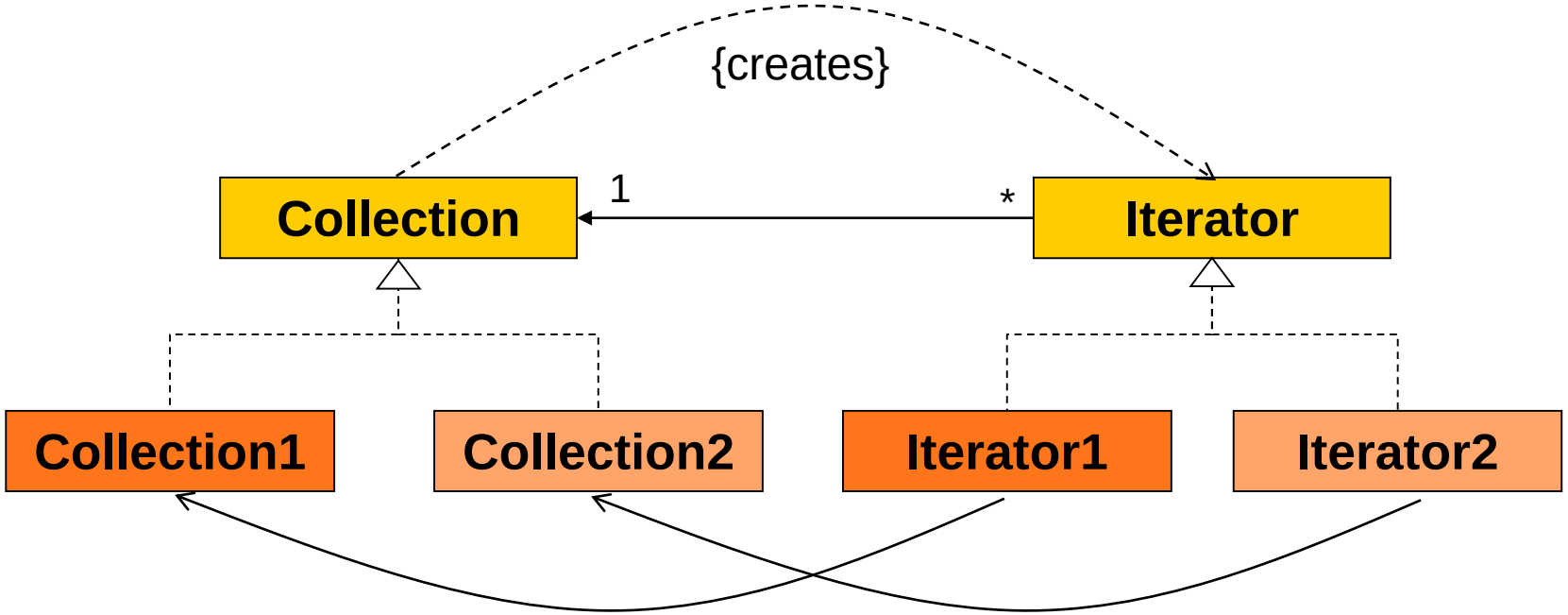
```
interface Iterator<E> {
   boolean      hasNext();
   E            next();       // may throw a NoSuchElementException
   default void remove() {    // optional
      throw new UnsupportedOperationException("remove");
   }
}
```

- **Pros**
  - Several access paths in a collection
  - Iterator can be specialized (e.g. Strings only)

- **Cons**
  - For every concrete collection implementation an iterator must be provided

Iterator

Every collection has a specific iterator



Generic Algorithms

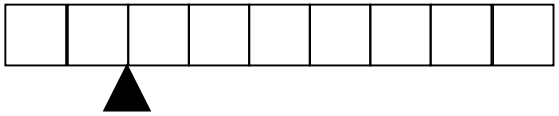
- **Example: Printing of a Collection**

```
public static void print(Collection<?> c) {
   Iterator<?> it = c.iterator();
   System.out.print("[");
   while(it.hasNext()) {
      System.out.print(it.next());
      if(it.hasNext()) System.out.print(", ");
   }
   System.out.print("]");
}
```

=> works with all Collections!

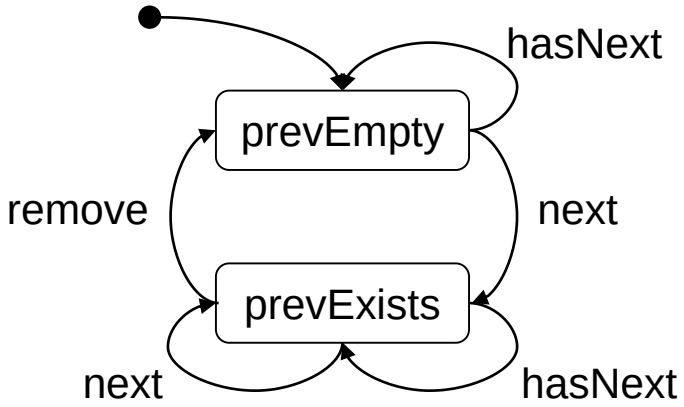
Iterator

- **Iterator Position**
  - Conceptionally inbetween two elements



in a collection with n elements
there exist n+1 iterator positions

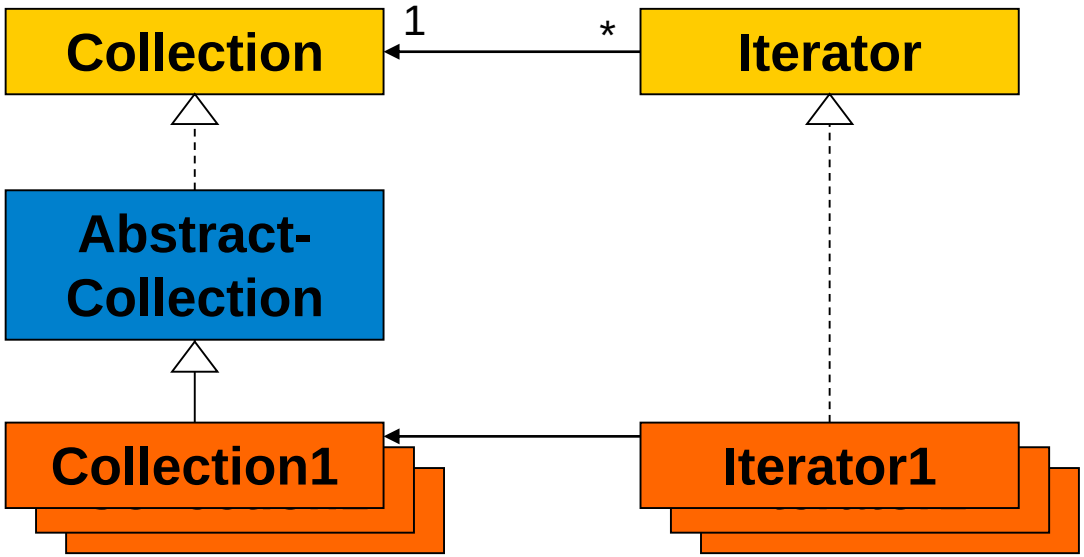
- hasNext() = there is an element which can be jumped over
- next() = returns the jumped over element
- remove() = removes the last element returned by next



otherwise an IllegalStateException is thrown

Implementation

- **Abstract Base Class**
  - In order to implement an interface, all methods have to be implemented
  - In an abstract base class some methods can be defined in terms of other methods

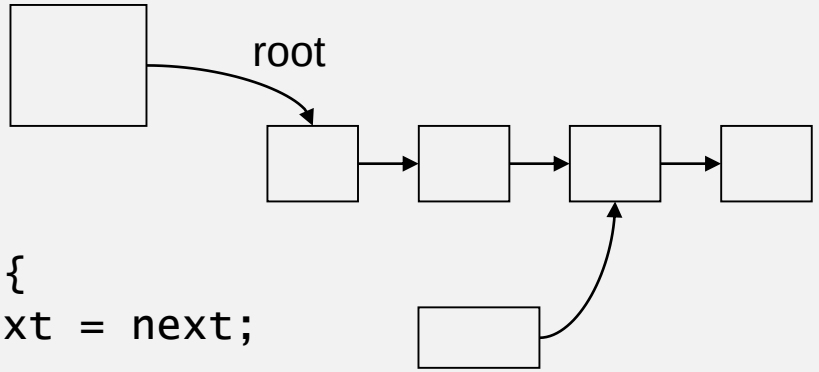


Abstract Collection

```
public interface Collection<E> extends Iterable<E> {
   public Iterator<E> iterator();
   public boolean     add(E x);
   public int         size();
   public boolean     isEmpty();
   public boolean     contains(Object x);
   public boolean     containsAll(Collection<?> c);
   public boolean     addAll(Collection<? extends E> c);
   public boolean     remove(Object x);
   public boolean     removeAll(Collection<?> c);
   public boolean     retainAll(Collection<?> c);
   public void        clear();
   public Object[]    toArray();
   public <T> T []    toArray(T[] a);
}
```

Example: Simple Collection

```
import java.util.*;
public class SimpleCollection<E> extends AbstractCollection<E> {
   private Node<E> root = null;
   public Iterator<E> iterator() {
      return new SCIterator<E>(root);
   }
   public boolean add(E e) {
      root = new Node<E>(e, root); return true;
   }
}

class Node<E> {
   E val;
   Node<E> next;
   Node(E val, Node<E> next) {
      this.val = val; this.next = next;
   }
}
```



Example: Simple Collection

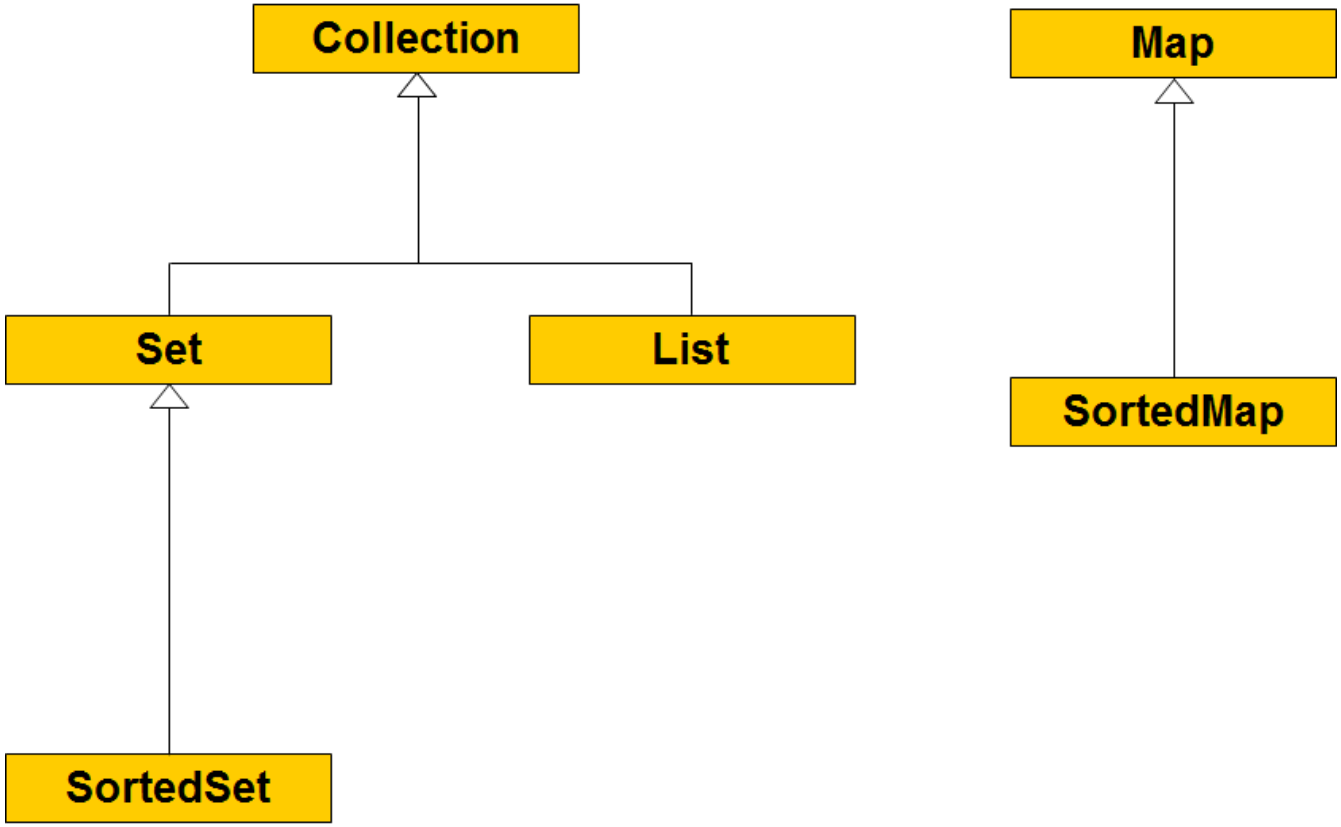
```
class SCIterator<E> implements Iterator<E> {
   private Node<E> current; // current is next elt to be returned

   SCIterator(Node<E> root) { current = root; }

   public boolean hasNext() { return current != null; }
   public E next() {
      if (current == null) { // end of collection reached
         throw new NoSuchElementException();
      }
      Node<E> res = current;
      current = current.next;
      return res.val;
   }
   public void remove() {
      throw new UnsupportedOperationException();
   }
}
```

Implementations

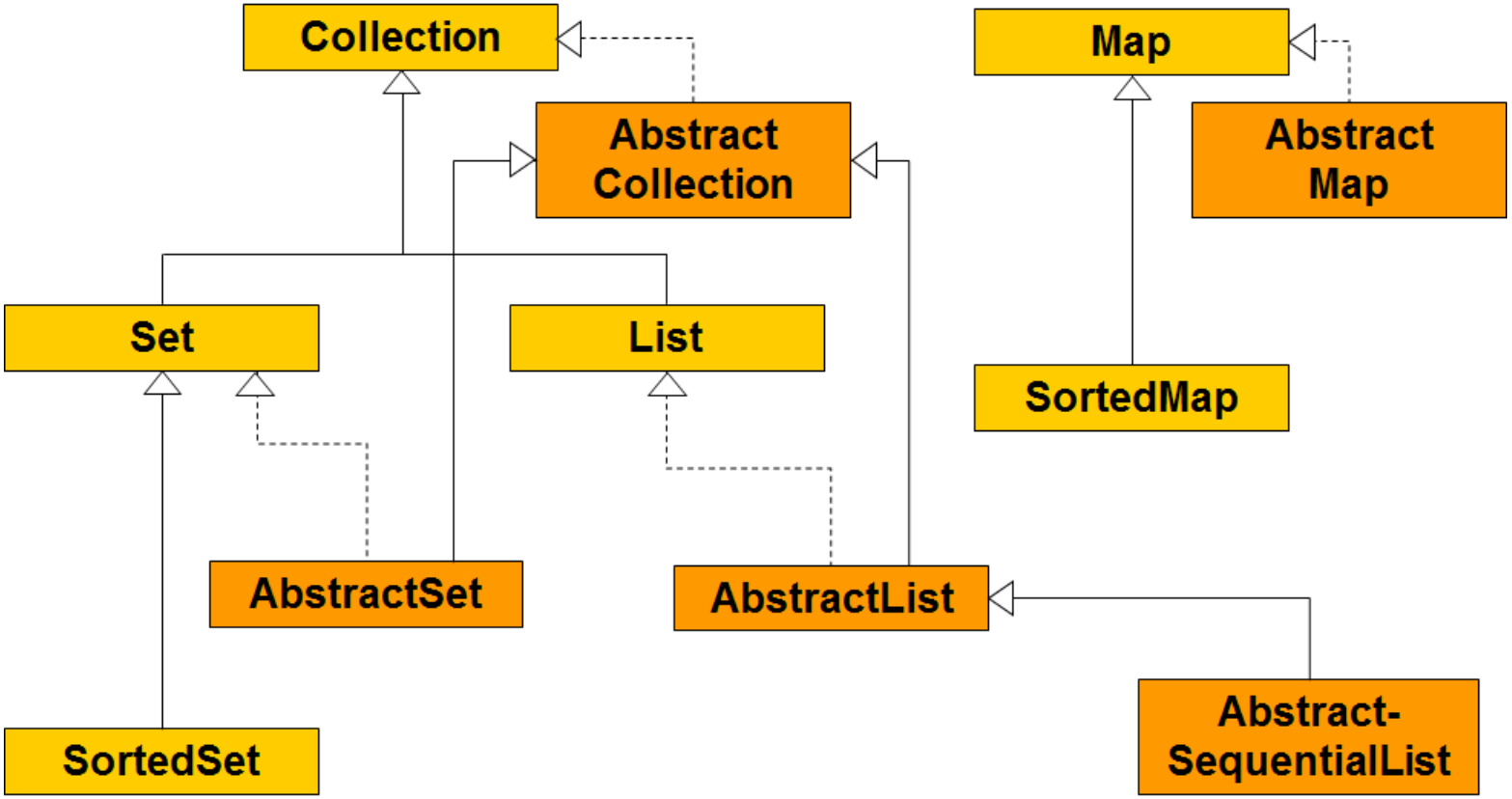
Interface	Implementation				Historical
Set	HashSet		TreeSet		
List		ArrayList		LinkedList	Vector Stack
Map	HashMap		TreeMap		Hashtable Properties

- **List**
  - ArrayList:
    - Array implementation (resizable)
    - Access O(1), insert/remove O(n)
  - LinkedList
    - Doubly linked list
    - Insert/remove fast, access O(n)
- **Set**
  - HashSet:
    - Implemented by hash table
    - Access O(1)
  - SortedSet
    - Implemented by red-black tree
    - Imposes ordering on its elements

Collection Framework Overview



Collection Framework Overview



Collection Framework Overview