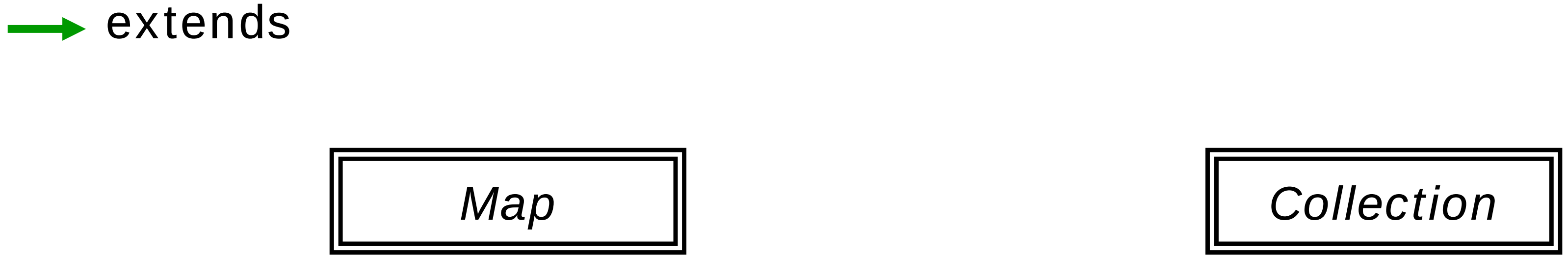
Collections Framework: Part 1

Lecture 17

Overview

- A framework of many classes and interfaces
- Part of the java.util package
  - See API Javadoc
  - See “Collections Framework” trail
- This framework provides *container* classes
  - Hold other objects
  - Defined as generic classes (recall Box<T>)
  - Allow efficient access to contents in useful ways
- Two basic kinds of containers:
  - Collection (List, Queue, Set)
  - Map

Map & Collection Hierarchies



Root Interface: Collection

- Generic

```
Collection<String> bag;
```

- Methods working with an individual collection

```
public int size()
public boolean isEmpty()
public boolean contains(Object target)
public boolean add(E element)
```

  - Danger: Client keeps reference (aliasing!)
  - Vague specification (eg are duplicates allowed?)

```
public boolean remove(Object target)
public Object[] toArray()
```

  - Returns a new array containing references to all the elements of the collection

```
public <T> T[] toArray(T[] dest)
```

  - What is returned depends on whether the elements in the collection fit in dest
  - If the type of dest is not compatible with the types of all elements in the collection, an exception is thrown

Root Interface: Collection cont'd

- Bulk methods using contents of another collection

  `public boolean containsAll(Collection c)`

  `public boolean addAll(Collection c)`

  - Returns true if any addition succeeds

  `public boolean removeAll(Collection c)`

  - Returns true if any removal succeeds

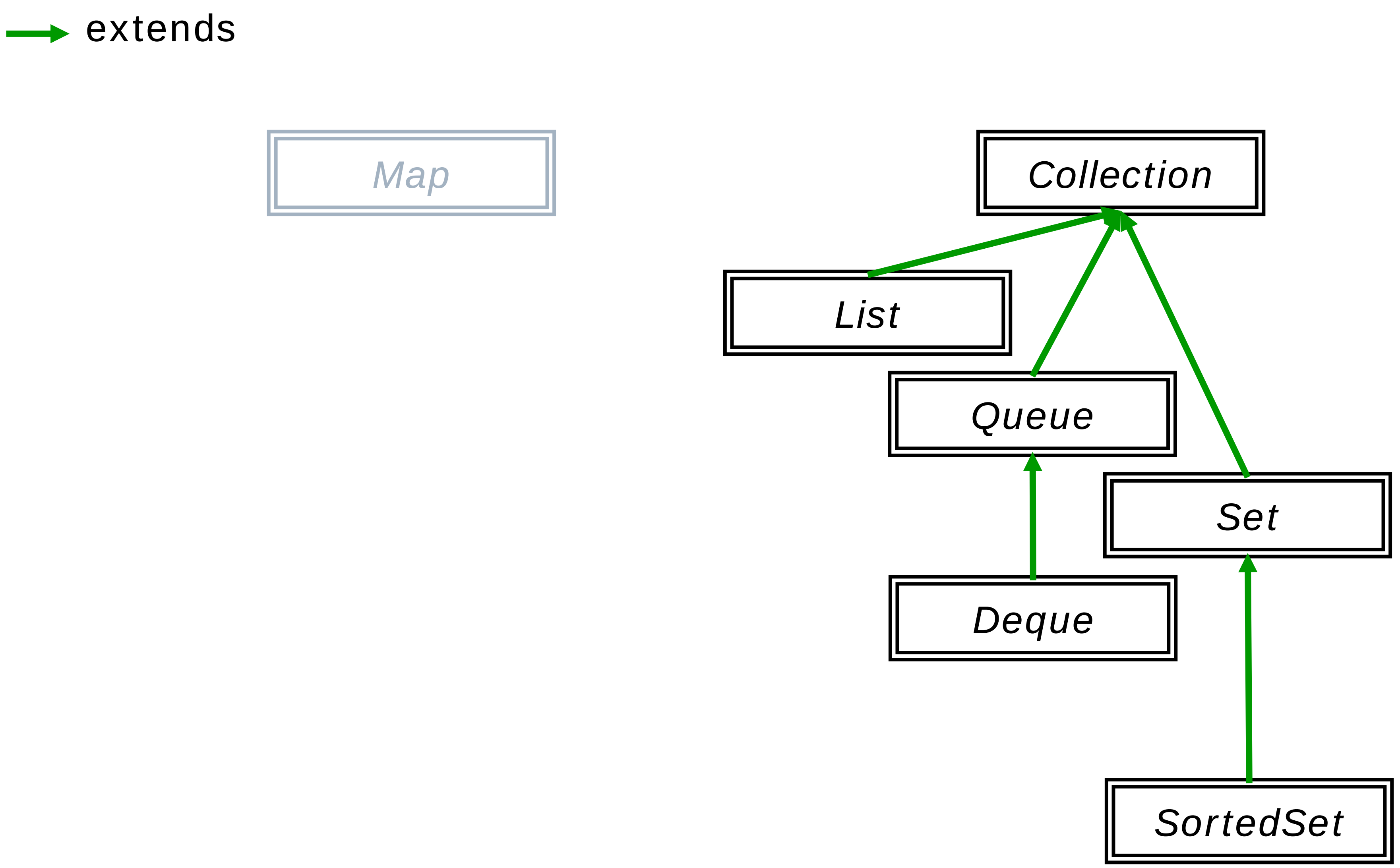
  `public boolean retainAll(Collection c)`

  - Removes from the collection all elements that are not elements of c

  `public void clear()`

  - Remove all elements from this collection
- No direct implementations of Collection in SDK
  - Useful for passing collections around and manipulating them where maximum generality is desired
    - Recall: "code to the interface"
  - Subinterfaces (List, Queue, Set) do have direct implementations

Collection Hierarchy



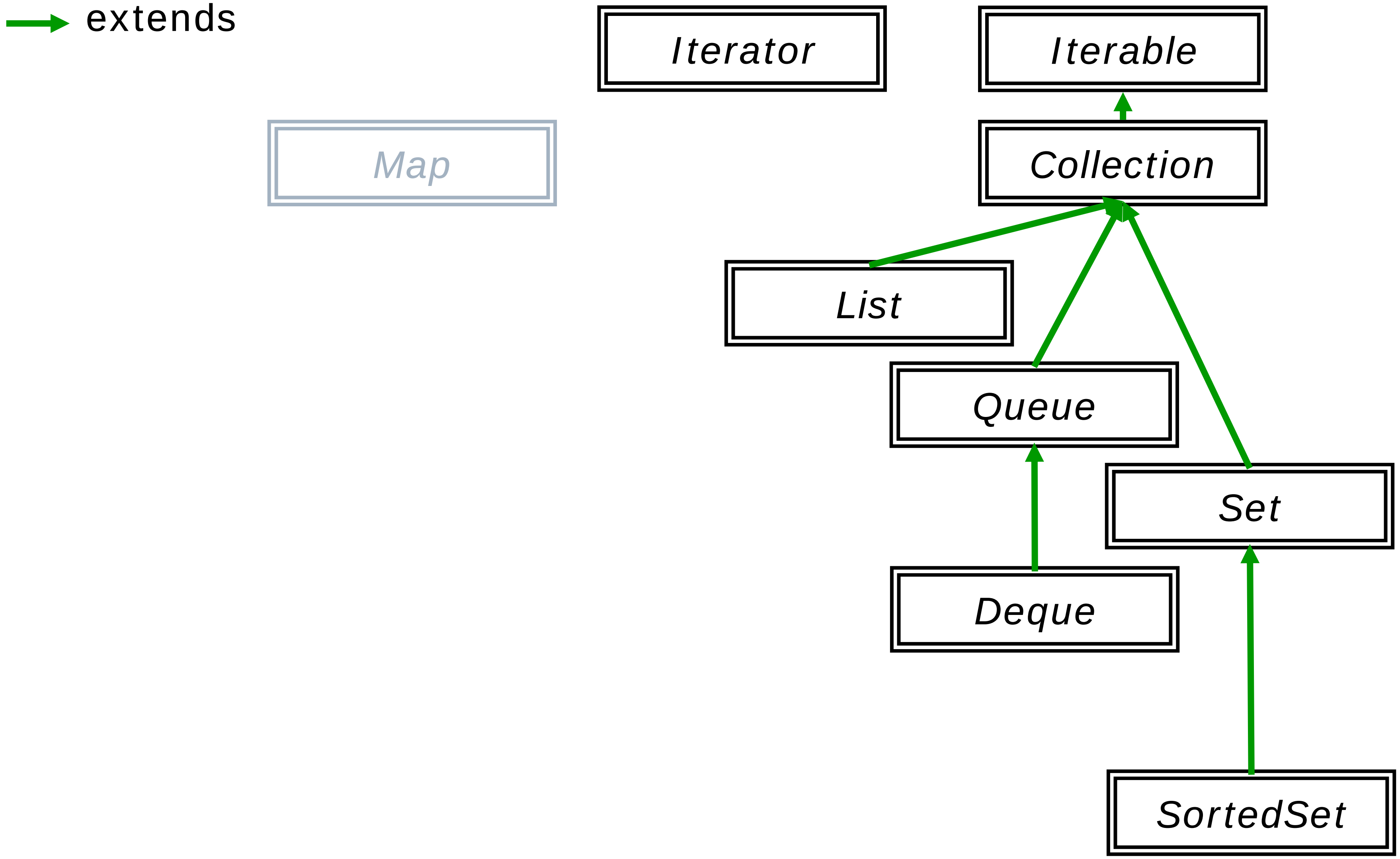
Subinterfaces

- List
  - Ordered sequence of elements
    - Indexed from 0 to list.size()-1
  - Client controls location of newly inserted element
  - Allows duplicate elements
  - New methods:
    - sublist (return a subsequence from index1 to index2)
- Queue
  - Ordered sequence of elements (LIFO, FIFO, priority)
  - Removals (and peeking) occur at the head
    - Subinterface Deque allows additions at head too
  - New methods:
    - offer (queue might be full)
    - peek (look at head without removing)
- Set
  - No duplicate elements (add is idempotent)
  - No guarantee of ordering
    - Subinterface SortedSet provides such a guarantee

Iteration

- To examine the contents of a collection, an *iterator* is used
  - Allows us to loop through contents, examining each element in turn
  - No guarantee of iteration order (for Collection)
  - Does not expose internal structure of collection
  - Declared type (an interface):
    ```
    interface Iterator<E> { ... }
    ```
- To obtain an iterator use collection method:
  ```
  public Iterator<E> iterator()
  ```
- Method is promised in the *Iterable* interface
  - Actually part of java.lang
  - Collection extends Iterable

Iterable Collection Hierarchy



Iterator Interface

- Three methods in Iterator interface

  `public boolean hasNext()`

  - Returns true iff the iteration has more elements

  `public E next()`

  - Returns the next element in the iteration
  - An exception will be thrown if there is no next element
  - Note use of generics in return type

  `public void remove()`

  - Remove from the collection the element last returned by the iteration
  - Can be called only *once per call of next*, otherwise an exception is thrown

Canonical Example

```
import java.util.Collection;
import java.util.Iterator;
. . .
public void removeLongStrings
       (Collection<String> c, int maxLen) {
  Iterator<String> it = c.iterator();
  while ( it.hasNext() ) {
    String str = it.next();
    if (str.length() > maxLen) {
      it.remove()
    }
  }
}
```

Special For-Loop Syntax (“for-each”)

- Syntactic shortcut for looping through something Iterable

  for (*Type loop-var* **:** *set-expression***)**

    statement

  - *Can not be used to remove elements from collection*
- Example

```
Collection<Student> roster = . . .
for (Student std : roster) {
  System.out.println(std.showInfo());
}
```

- Can be used with arrays as well

```
int[] values = . . .
double sum = 0.0;
for (int v : values) {
  sum += v;
}
```

ListIterator

- ListIterator interface extends Iterator interface
  - Provides ordering guarantee for iteration
  - Adds methods for moving forwards or *backwards*
- Methods

```
public boolean hasNext() / boolean hasPrevious()
public E next() / E previous()
public int nextIndex() / int previousIndex()
```

  - When at the end of the list, nextIndex() returns list.size()
  - When at the beginning of the list, previousIndex() returns -1

```
public void remove()
```

  - Remove the element last returned by next() or previous()

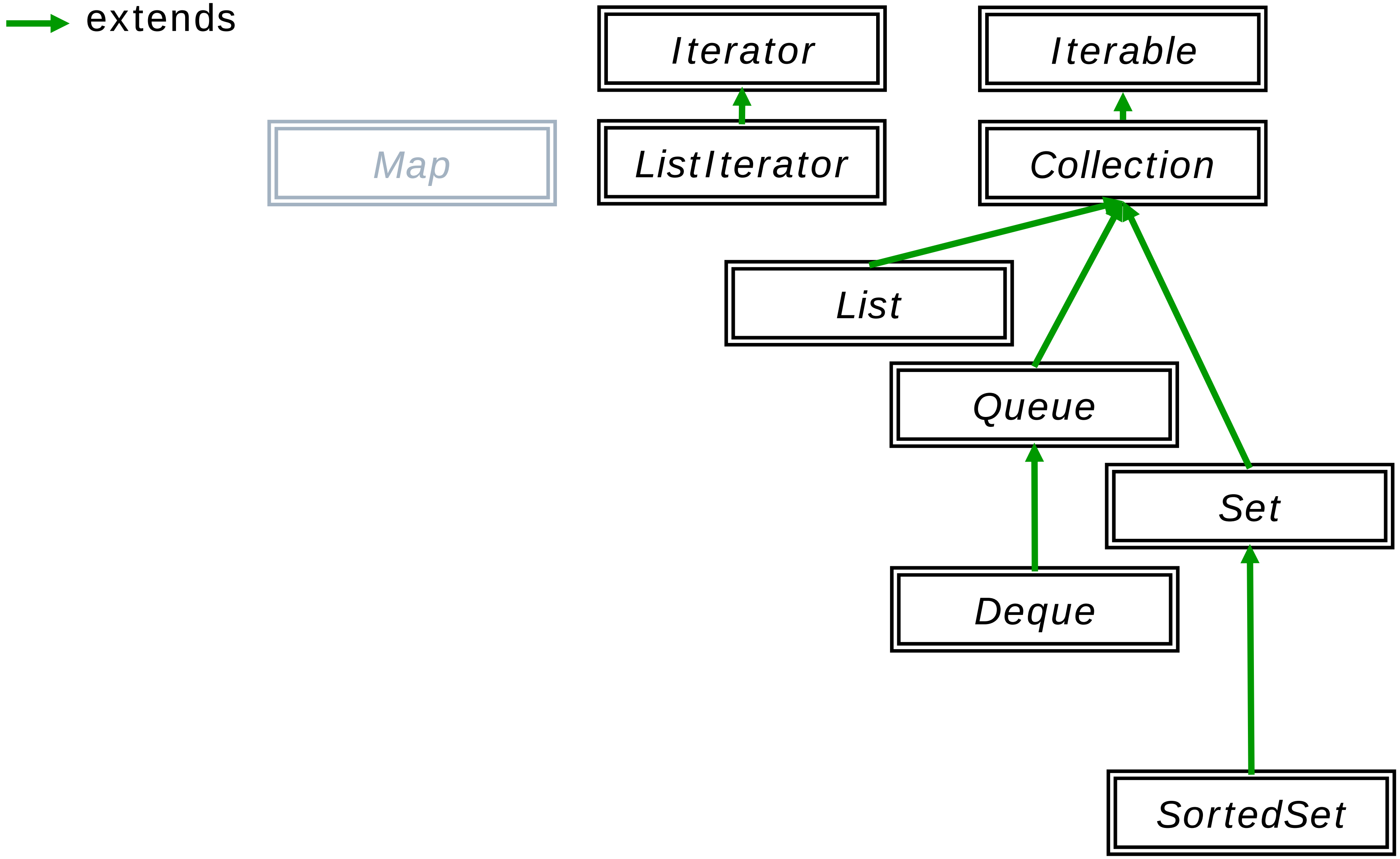
```
public void add(E elem)
```

  - Inserts elem into list in front of the element that would be returned by next(), or at the end if no next element exists

```
public void set(E elem)
```

  - Replace the element last returned by next() or previous() with elem

Iterable Collection Hierarchy



cf Resolve's Sequence

- Exercise for the reader:
  - Compare Java's ListIterator with Resolve's Sequence component
  - How does insertion point differ?
  - How does element removal differ?

Modifying a Collection

- While iterating through a collection, the *only* safe way to modify the collection is *through the iterator itself*
  - Use Iterator's remove() method, not Collection's remove(Object) method
- Many iterators in Java SDK try to detect a modification of the underlying collection and complain
  - An exception is thrown
  - Known as "fail-fast" behavior
  - Not guaranteed! Do not rely on this safety net!

Summary

- Collection Interface
  - Generic container classes
  - Subinterfaces: List, Queue, Set
- Iterators
  - Iterable interface for obtaining an iterator
  - Provides insertion/removal point for collection
  - “foreach” iteration syntax