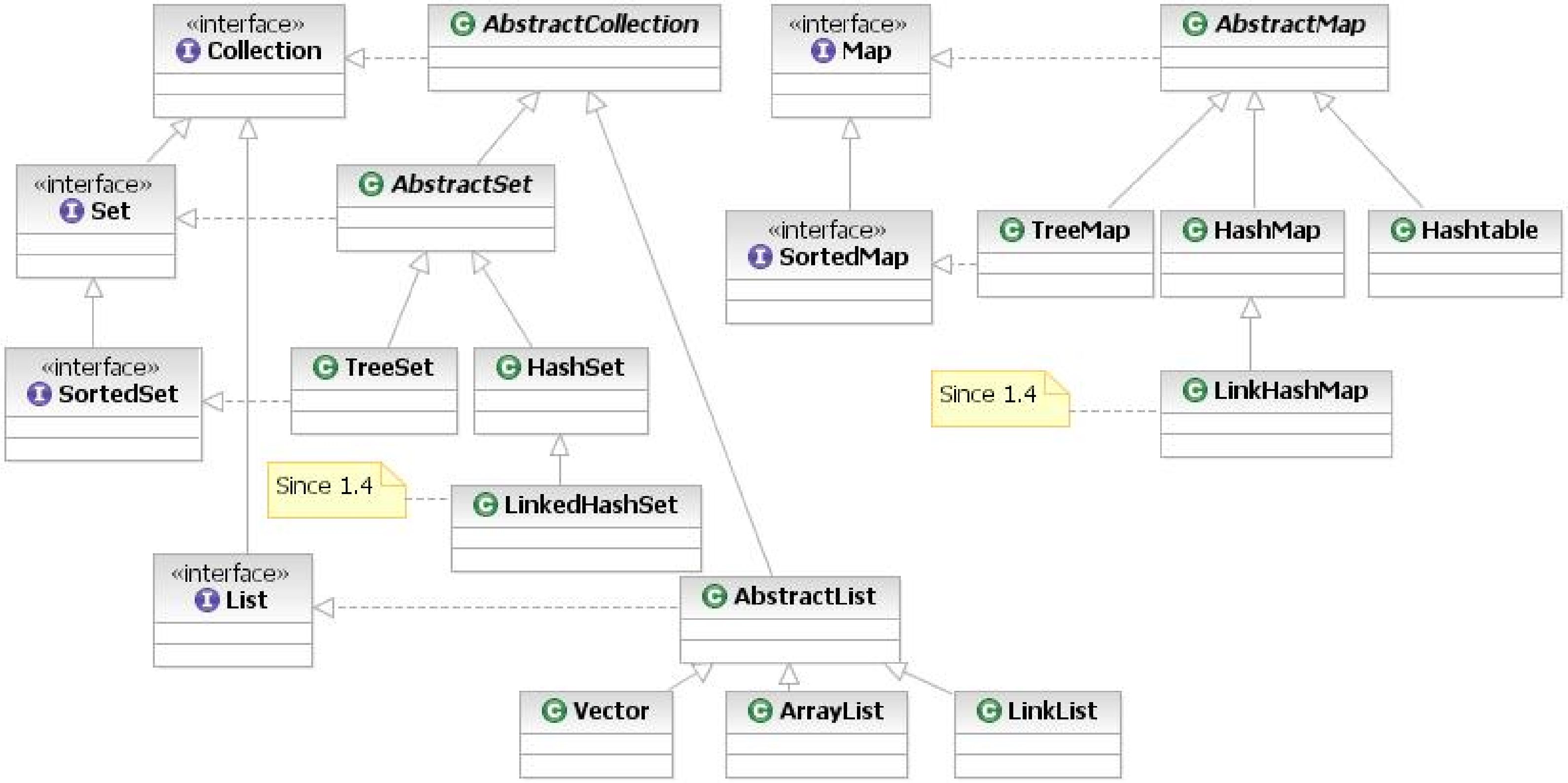
Overview of the Java Collections Framework

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software
Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Lesson

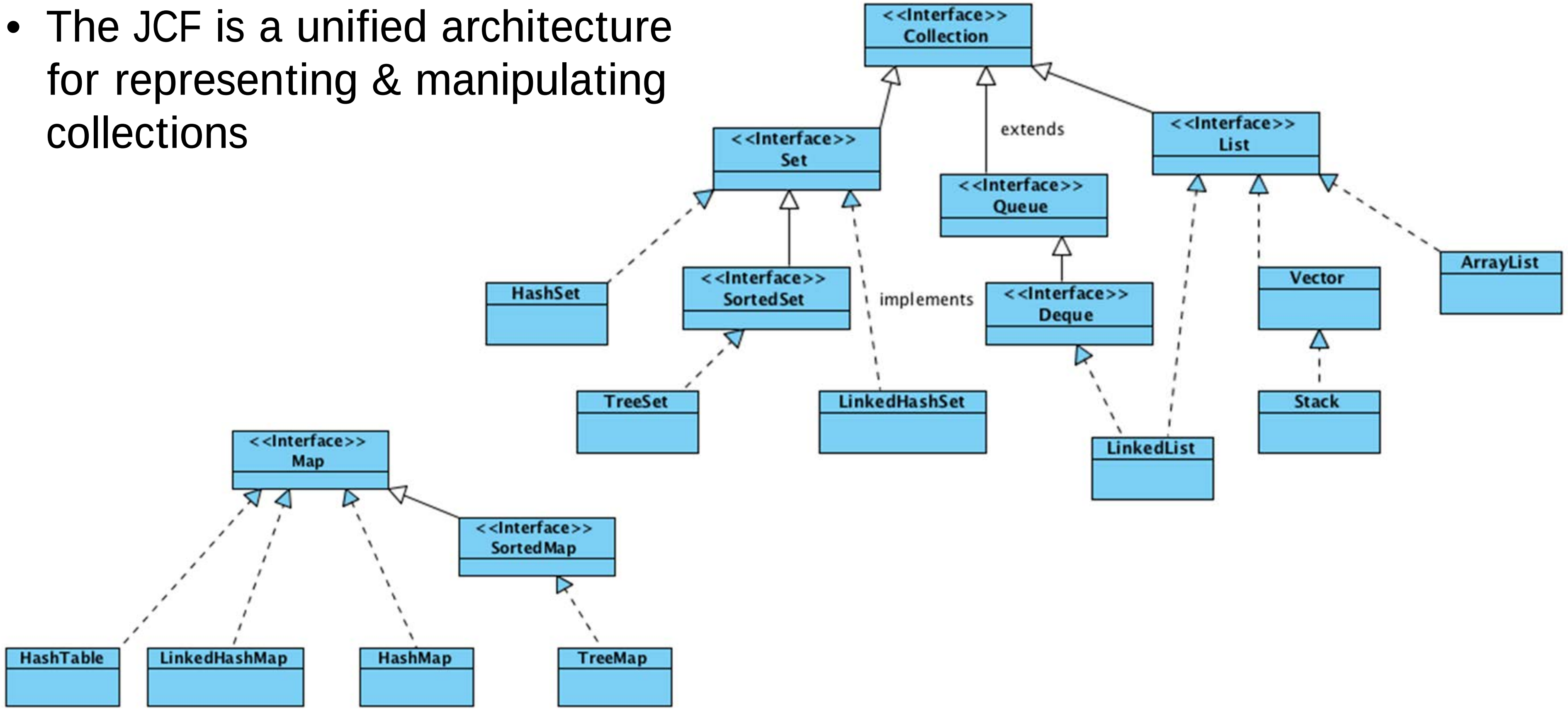
- Understand the Java Collections Framework (JCF)



Java Collections Framework

Overview of the Java Collections Framework

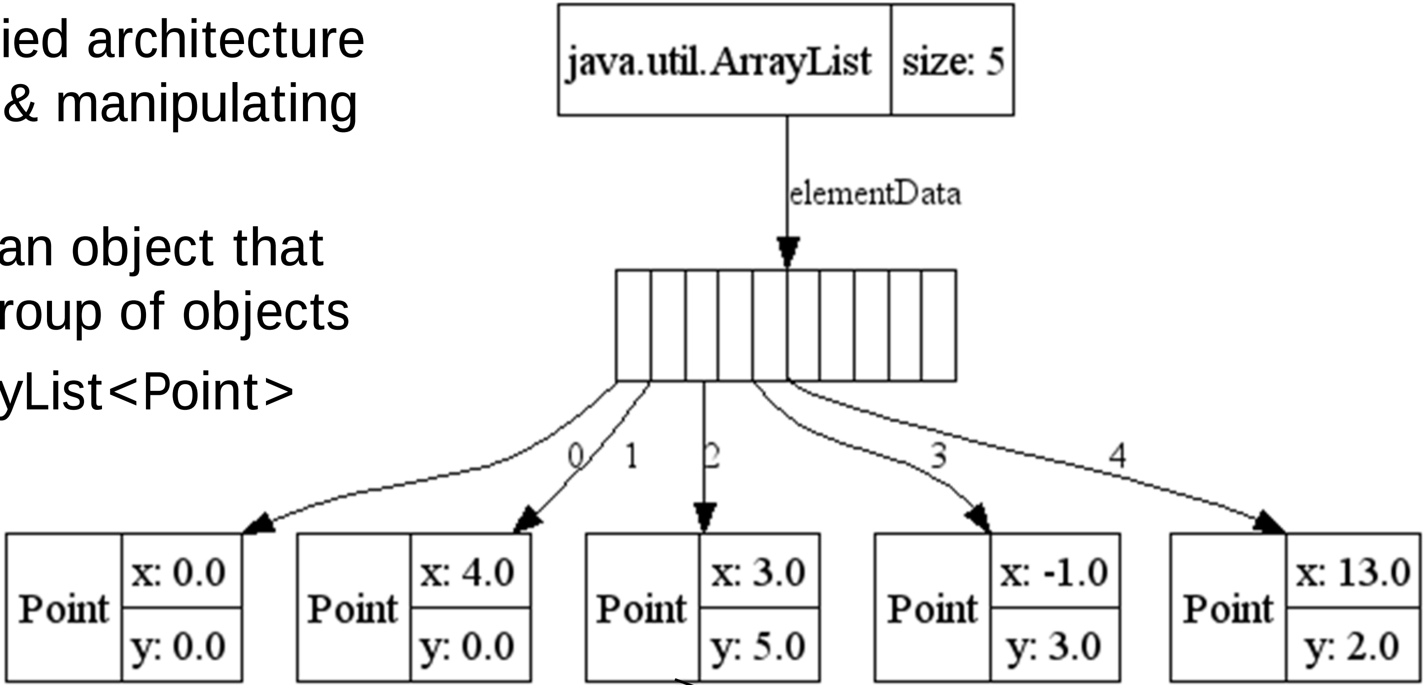
- The JCF is a unified architecture for representing & manipulating collections



See docs.oracle.com/javase/8/docs/technotes/guides/collections

Overview of the Java Collections Framework

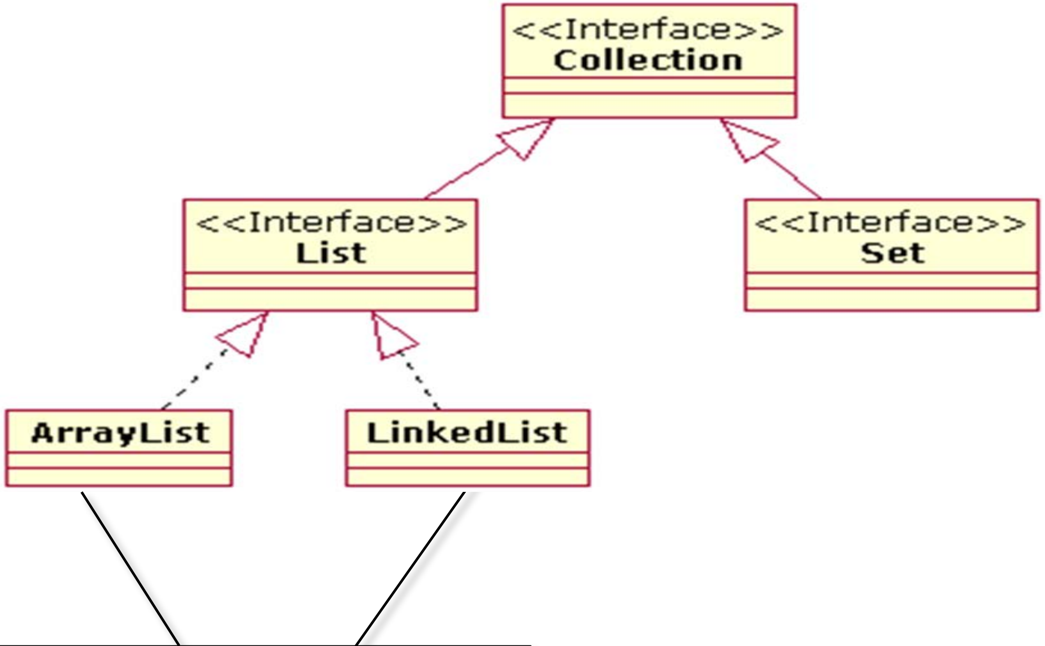
- The JCF is a unified architecture for representing & manipulating collections
  - A *collection* is an object that represents a group of objects
    - e.g., an ArrayList<Point>



Each element in the collection is actually a reference to another object

Overview of the Java Collections Framework

- The JCF is a unified architecture for representing & manipulating collections
  - A *collection* is an object that represents a group of objects
  - Collections can be accessed & manipulated independently of their representation



e.g., the List interface can be implemented as either a LinkedList or as an ArrayList

Overview of the Java Collections Framework

- JCF is based on more than a dozen collection interfaces

The collection interfaces contain two groups

- java.util.Collection

```
java.util.Set
java.util.SortedSet
java.util.NavigableSet
java.util.Queue
java.util.concurrent.BlockingQueue
java.util.concurrent.TransferQueue
java.util.Deque
java.util.concurrent.BlockingDeque
```

Overview of the Java Collections Framework

- JCF is based on more than a dozen collection interfaces

The collection interfaces contain two groups

- java.util.Collection
- java.util.Map

```
java.util.SortedMap
java.util.NavigableMap
java.util.concurrent.ConcurrentMap
java.util.concurrent.
          ConcurrentNavigableMap
```

Overview of the Java Collections Framework

- JCF is based on more than a dozen collection interfaces
  - Includes implementations of these interfaces & algorithms to manipulate them

Inter face	Hash Table	Resize Array	Balanced Tree	Linked List	Hash Table+ Linked List
Set	HashSet		Tree Set		Linked Hash Set
List		Array List		LinkedList	
Deque		Array Deque		LinkedList	
Map	HashMap		TreeMap		Linked Hash Map

JCF implementations use inheritance, polymorphism, & generics extensively

Overview of the Java Collections Framework

- JCF has several key benefits

Overview of the Java Collections Framework

- JCF has several key benefits
  - Reduces programming effort
    - By providing data structures & algorithms so developers don't need to write them

```
class ArrayList ... {
  ...
  public Object[] toArray() {
    return Arrays
      .copyOf(elementData,
                    size);
  }
  ...
```

Overview of the Java Collections Framework

- JCF has several key benefits
  - Reduces programming effort
  - Enables interoperability
    - e.g., gives a common way to pass collections

```
class Vector ... {
  ...
  boolean addAll(Collection<?
                        extends E> c) {
    Object[] a = c.toArray();
    int numNew = a.length;
    ensureCapacityHelper
      (elementCount + numNew);
    System.arraycopy(a, 0,
                     elementData,
                     elementCount,
                     numNew);
    elementCount += numNew;
    return numNew != 0;
  } ...
```

Overview of the Java Collections Framework

- JCF has several key benefits
  - Reduces programming effort
  - Enables interoperability
  - Increases performance
    - Highly optimized implementations of data structures & algorithms

```
class ConcurrentHashMap ... {
  ...
  public V get(Object key) {
    ...
    int h = spread(key
                     .hashCode());
    if ((tab = table) != null &&
        ((e = tabAt(tab, (n - 1)
          & h)) != null) {
      if (key.equals(ek)))
         return e.val;
    }
    ...
```

See www.ibm.com/developerworks/library/j-jtp08223

Overview of the Java Collections Framework

- JCF has several key benefits
  - Reduces programming effort
  - Enables interoperability
  - Increases performance
  - Reduces effort designing & learning new (non *ad hoc*) APIs

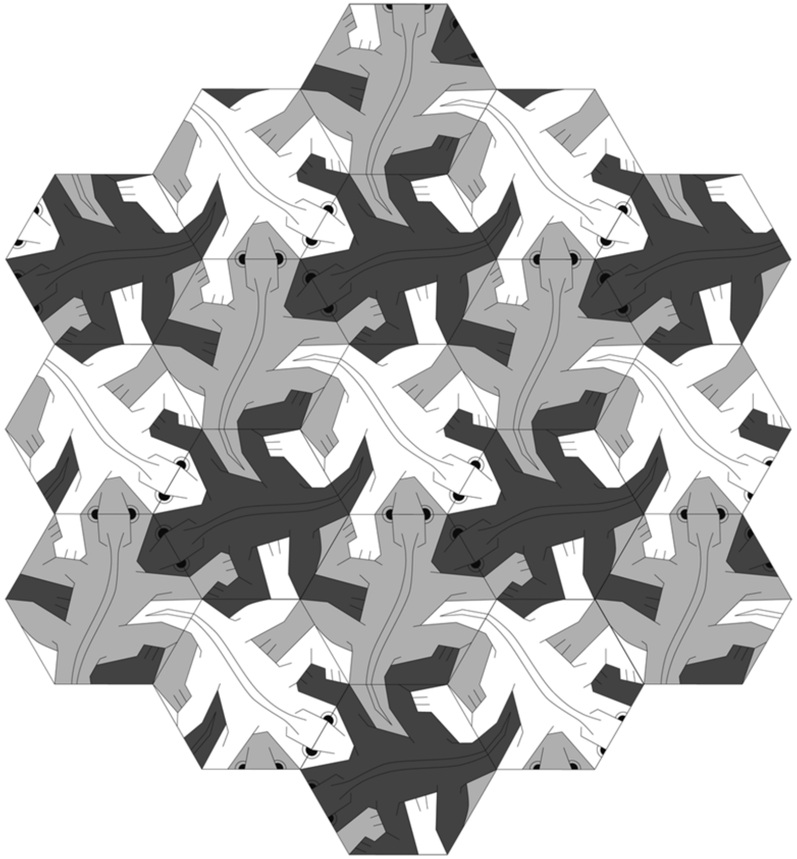
```
class AbstractList ... {
  ...
  public Iterator<E> iterator(){
    return new Itr();
  }

  private class Itr implements
                Iterator<E> {
    public boolean hasNext()
    { ... }

    public E next() { ... }
    ...
  }
  ...
```

Overview of the Java Collections Framework

- JCF has several key benefits
  - Reduces programming effort
  - Enables interoperability
  - Increases performance
  - Reduces effort designing & learning new (non *ad hoc*) APIs
  - Fosters software reuse
    - By providing standard interfaces for collections & algorithms that manipulate them



Overview of the Java Collections Framework

- Common JCF classes
  - An ArrayList is a variable-sized list of items similar to a built-in Java array

```
import java.util.ArrayList;
...
List<String> myList =
   new ArrayList<>();

myList.add("I");
myList.add("am");
myList.add("Ironman");

String itemOne = myList.get(0);


myList.remove(0);


...
```

See docs.oracle.com/javase/8/docs/technotes/guides/collections

Overview of the Java Collections Framework

- Common JCF classes
  - An ArrayList is a variable-sized list of items similar to a built-in Java array

```
import java.util.ArrayList;
...
List<String> myList =
  new ArrayList<>();

myList.add("I");
myList.add("am");
myList.add("Ironman");

String itemOne = myList.get(0);


myList.remove(0);


...
```

List stores object of type java.lang.String, so no need to cast item back to String

See docs.oracle.com/javase/8/docs/technotes/guides/collections

Overview of the Java Collections Framework

- Common JCF classes
  - An ArrayList is a variable-sized list of items similar to a built-in Java array
  - A HashMap stores key/value pairs

```
import java.util.HashMap;
...
HashMap<String, Foo> myMap =
   new HashMap<>();

Foo f1 = new Foo();
Foo f2 = new Foo();
myMap.put("one", f1);
myMap.put("two", f2);

if (f2 == myMap.get("two"))
   ...
else if (f1 ==
           myMap.get("one"))
   ...
```

Overview of the Java Collections Framework

- Concurrent collections provide features that are frequently needed in concurrent programming

These are the concurrent-aware interfaces:

```
BlockingQueue
TransferQueue
BlockingDeque
ConcurrentMap
ConcurrentNavigableMap
```

Overview of the Java Collections Framework

- Concurrent collections provide features that are frequently needed in concurrent programming

Concurrent-aware classes include

```
LinkedBlockingQueue
ArrayBlockingQueue
PriorityBlockingQueue
DelayQueue
SynchronousQueue
LinkedBlockingDeque
LinkedTransferQueue
CopyOnWriteArrayList
CopyOnWriteArraySet
ConcurrentHashMap
```

Concurrent collections covered in CS 892 (www.dre.vanderbilt.edu/~schmidt/cs892)

Iterating Through Collections in Java

Iterating Through Collections in Java

- Java has several ways to loop through collections
  - The conventional for loop used in C/C++

```
List<String> myStrings =
  new ArrayList<>();

myStrings.add("a");
myStrings.add("b");
myStrings.add("c");

for(int i = 0;
        i < myStrings.size();
      i++)
    System.out.println
      (myStrings.get(i));
```

Venerable, but crufty...

Iterating Through Collections in Java

- Java has several ways to loop through collections
  - The conventional for loop used in C/C++

```
List<String> myStrings =
  new ArrayList<>();

myStrings.add("a");
myStrings.add("b");
myStrings.add("c");

Object[] array =
  myStrings.toArray();

for(int i = 0;
    i < array.length;
    i++)
  System.out.println(array[i]);
```

Useful in certain situations, but typically overkill...

Iterating Through Collections in Java

- Java has several ways to loop through collections
  - The conventional for loop used in C/C++
  - An enhanced for-each loop for iterating over collections

```
List<String> myStrings =
  new ArrayList<>();

myStrings.add("a");
myStrings.add("b");
myStrings.add("c");

for (String aString :
     myStrings)
  System.out.println(aString);
```

Very clean & concise

Iterating Through Collections in Java

- Java has several ways to loop through collections
  - The conventional for loop used in C/C++
  - An enhanced for-each loop for iterating over collections
  - An Iterable interface

```
List<String> myStrings =
  new ArrayList<>();

myStrings.add("a");
myStrings.add("b");
myStrings.add("c");

for (Iterator<String> it =
       myStrings.iterator();
     it.hasNext();
     )
   System.out.println
     (it.next());
```

Pattern-oriented, but overly verbose compared to for-each loop

Iterating Through Collections in Java

- Java has several ways to loop through collections
  - The conventional for loop used in C/C++
  - An enhanced for-each loop for iterating over collections
  - An Iterable interface
  - The forEach() method

```
List<String> myStrings =
  new ArrayList<>();

myStrings.add("a");
myStrings.add("b");
myStrings.add("c");

myStrings
  .stream()
  .forEach
    (aString ->
       System.out.println
         (aString));
```

Very powerful, but requires knowledge of Java lambda expressions & streams

End of Overvew of the Java Collections Framework