Data Structures in Java

Lecture 3: ADTs in Java.

9/16/2015

Daniel Bauer

Today

- ADTs and Data Structures in Java (Generics, Interfaces etc., Java Standard Library)
- Linked List Implementation.
- Binary Search Example.

Recitation Sessions

- Tuesday 7:30pm, 413 Kent
- Thursday 7:30pm, 614 Schermerhorn
- **Friday 2:00pm, 603 Hamilton (NEW)**
- Also: One more TA, see schedule on website.

Homework 0 and 1

- Some of you still had trouble with HW 0. Had to make sure everyone got set up first.

- HW 1 out asap! New due date: Sun 9/27, 11:59pm

Homework Late Policy

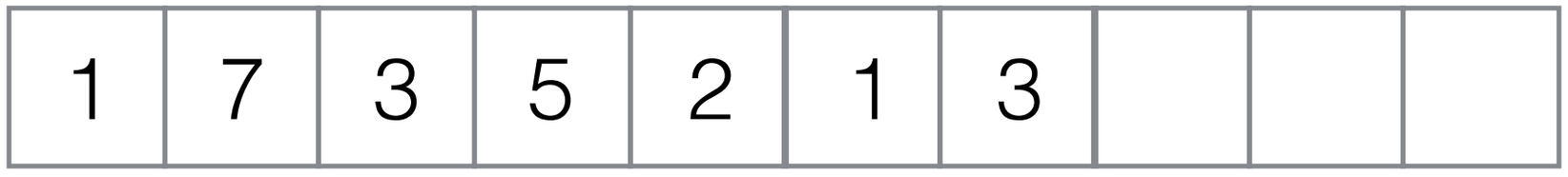
- You will lose 1% of the total homework score for every 6 minutes your homework is late.
- The latest pushed version of your homework will be graded.
- Homework submitted later than 10h after the official deadline will receive no credit.
- If you need to miss a homework assignment you need to talk to me in advance (except in emergencies… take care of the emergency first!)
- Document your code! Undocumented code will result in lower scores.

Outline

- Some Java features useful for implementing Data Structures.
    - Generics, Interfaces, Nested classes.
        - Iterator, Iterable from the Java API.
- Implementation of Linked List.
- Implementation of Binary Search.
- Lists in the Java Collections API.

Array Lists

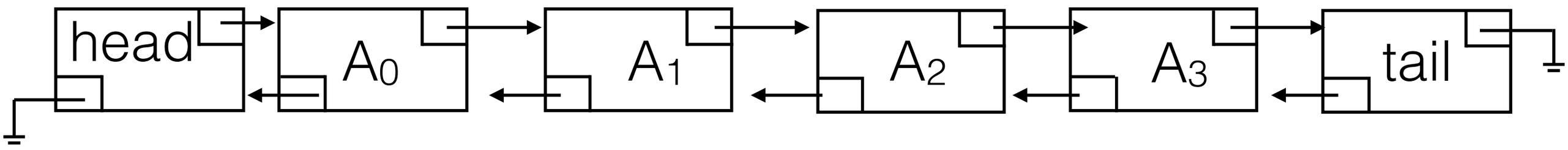
```
public class SimpleArrayList {

    public static final int DEFAULT_CAPACITY = 10;
    private int theSize;
    private Integer[] theItems;

}
```



Doubly Linked Lists

- Also maintain reference to previous node in the list.
- Speeds up append at end of list.

```
public class Node {
    public Integer data;
    public Node next;
    public Node prev;
    public Node(Integer d, Node n, Node p) {
        data = d; next = n; prev = n;
    }
}
```

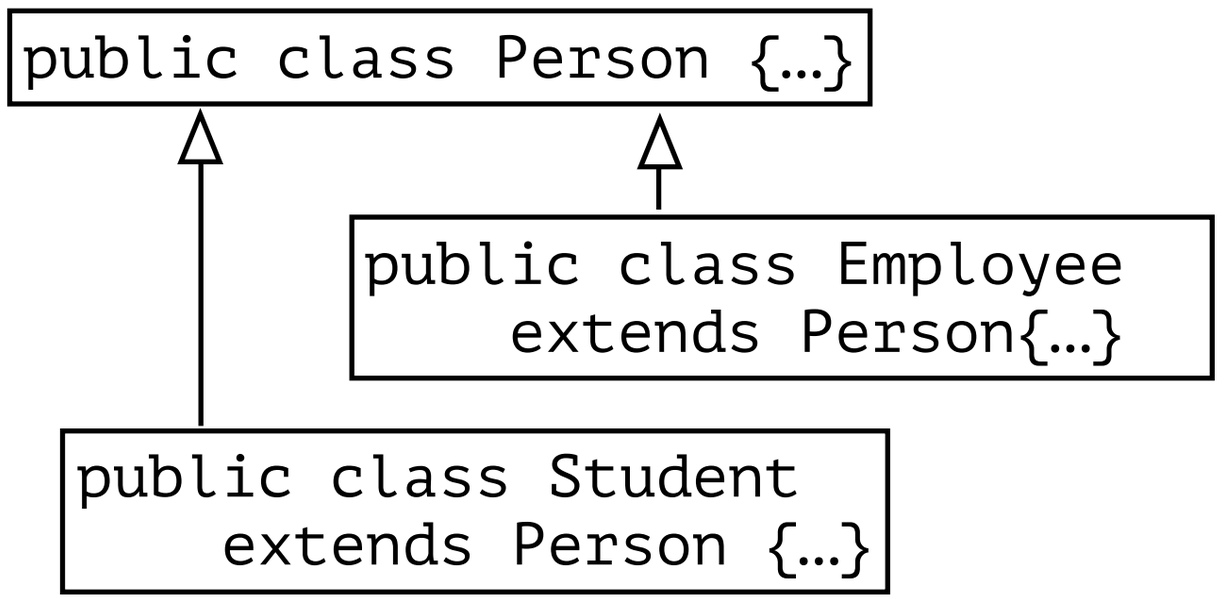


Making ADTs More General

- Problem: Our lists can only store Integers.
- Possible Solution: Polymorphism. Choose the *most general class* of items that you expect to see in the List.

```
Person[] arr = new Person[10];

arr[0] = new Employee(…);
arr[1] = new Student(…);
```

e.g. for an Array



Java Generics

- We don't normally know what kind of object to expect in a data structure.

- Java allows to add *type parameters (<> syntax)* to definitions of classes. Such classes are called *generic classes.*

```
public class MyArrayList<AnyType> {
    private AnyType[] theItems;
    ...
    public AnyType get(int idx) { ... }
    public boolean add(int idx, AnyType x) { ... }
}
```

Java Generics (2)

- Type parameters make it possible to create a new data structure for specific objects (and their sub-types) during runtime.

```
MyArrayList<Integer> l = new MyArrayList<Integer>();
```

- In Java 7 and 8, this can be simplified using the <> (Diamond) operator:

```
MyArrayList<Integer> l = new MyArrayList<>();
```

- Type of `l` is inferred automatically.

Nested Classes

- Usually every Java class is defined in its own `.java` file.
- Sometimes classes have a specific purpose (in relation to another class), e.g. `Node` is specific to `MyLinkedList`.

```
class OuterClass {
    ...
    static class StaticNestedClass {
        ...
    }
    class InnerClass {
        ...
    }
}
```

Static Nested Classes

```
public class MyLinkedList<AnyType> implements Iterable<AnyType>{

    private static class Node<AnyType> {
        public Node( AnyType d, Node<AnyType> p, Node<AnyType> n ) {
            data = d; prev = p; next = n;
        }
        public AnyType data;
        public Node<AnyType>   prev;
        public Node<AnyType>   next;
    }

    . . .
}
```

- Static nested classes cannot access any instance members of the outer class.
- They essentially behave like normal top-level classes.

Inner Classes

```
public class MyLinkedList<AnyType> implements Iterable<AnyType>{

    private int theSize;
    private Node<AnyType> beginMarker;
    private Node<AnyType> endMarker;

    public java.util.Iterator<AnyType> iterator( ) {
        return new LinkedListIterator( );
    }

    private class LinkedListIterator implements java.util.Iterator<AnyType> {
        private Node<AnyType> current = beginMarker.next;
       ...
    }
}
```

- Instances of inner classes can access instance members of the outer instance that created it.

For-Each Loops

- `Iterables` support special Java syntax

```
for (T item : someIterable) {
    System.out.println(item.toString());
}
```

- No need to explicitly get the `Iterator` and call `next()` repeatedly.

Java Iterators

```
package java.lang;

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

Our LinkedList implementation should be compatible with the Iterator interface.

http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html

The Iterable Interface

```
package java.lang;

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

- Using Iterables and Iterators:

```
Iterator<T> someIterator = someIterable.iterator()

while (someIterator.hasNext()) {
    T nextItem = someIterator.next();
    System.out.println(nextItem.toString());
}
```

- Don’t implement Iterable and Iterator in the same class!

http://docs.oracle.com/javase/7/docs/api/java/lang/Iterable.html

The Comparable Interface

```
package java.lang;

public interface Comparable<AnyType> {
    int compareTo(AnyType other);
}
```

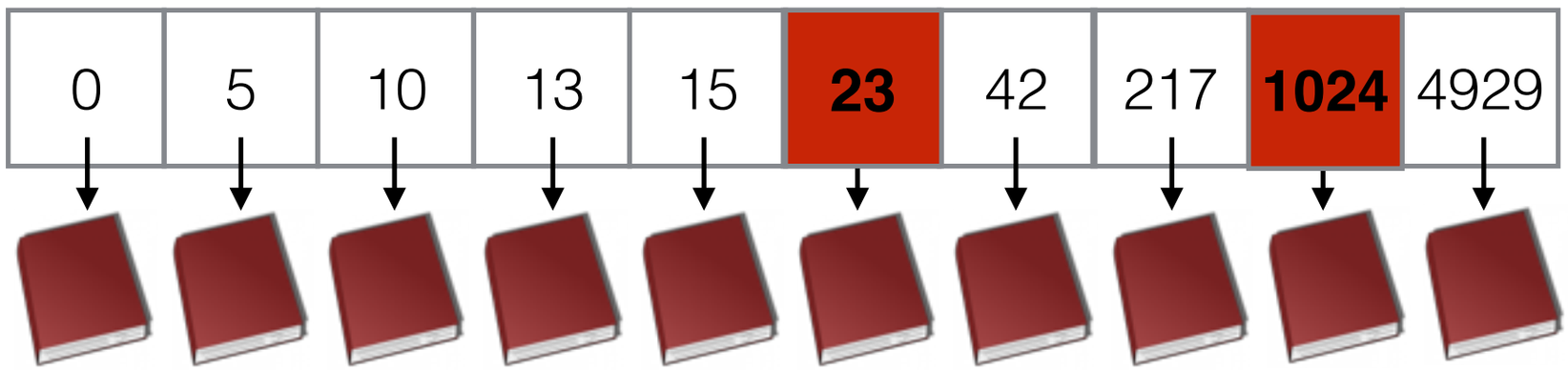
`compareTo` returns
negative int if this < o
positive int if this > o
0 if this == o

- comparison usually involves querying some member of `other`.
- The type parameter makes sure that all `other` objects have these fields.

http://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html

Binary Search

Given a sorted list, find the entry with a specific key.

A = 

x = 1024

- Find entry y in the middle if A: y = A[A.length/2]
  - if (y == x) we found the entry.
  - if (y < x) continue search on second half of A.
  - if (y > x) continue search on first half of A.

- In the worst case we need $\log_2$(length(A)) steps.

Binary Search with `Comparable`

```
public class BinarySearch<AnyType extends Comparable<AnyType>> {

        public int binarySearch( AnyType [ ] a, AnyType x ) {
                int low = 0, high = a.length - 1;

                while( low <= high ) {
                    int mid = ( low + high ) / 2;

                    if( a[ mid ].compareTo( x ) < 0 )
                        low = mid + 1;
                    else if( a[ mid ].compareTo( x ) > 0 )
                        high = mid - 1;
                    else
                        return mid;   // Found
                }
                return -1;
        }
}
```

Static Generic Methods

```
public static <AnyType extends Comparable<AnyType>>
    int binarySearch( AnyType [ ] a, AnyType x )
    {
        int low = 0, high = a.length - 1;

        while( low <= high )
        {
            int mid = ( low + high ) / 2;

            if( a[ mid ].compareTo( x ) < 0 )
                low = mid + 1;
            else if( a[ mid ].compareTo( x ) > 0 )
                high = mid - 1;
            else
                return mid;   // Found
        }
        return -1;
    }
```

A Comparable Person Class

```
public class Person implements Comparable<Person> {

    private String firstName;
    private String lastName;

    public Person(String last, String first) {
        lastName = last;
        firstName = first;
    }

    public int compareTo(Person other) {
        int lastNameComp = lastName.compareTo(other.lastName);
        if (lastNameComp == 0)
            return firstName.compareTo(other.firstName);
        else
            return lastNameComp;
    }
}
```

Searching for Presidents

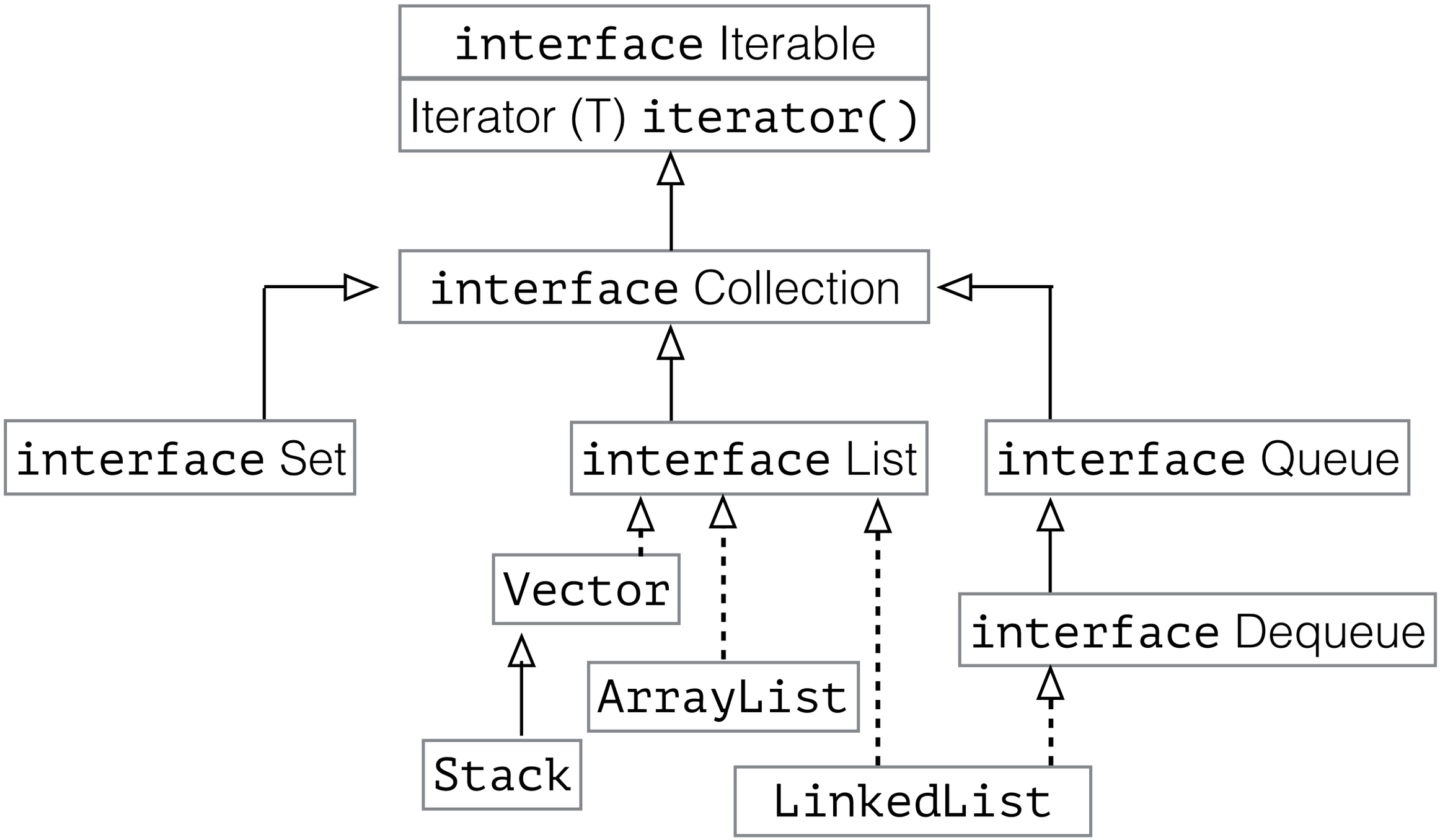
```
Person[] presidents = {
            new Person("Adams","John "),
            new Person("Adams","John Quincy "),
            new Person("Arthur","Chester Alan "),
            new Person("Buchanan","James "),
            new Person("Bush","George "),
            new Person("Bush","George W."),
            new Person("Carter","Jimmy "),
            new Person("Cleveland","Grover "),
            new Person("Clinton","Bill "),
            .    .   .
            new Person("Washington","George "),
            new Person("Wilson","Woodrow ")
        };
int index = BinarySearch.binarySearch(presidents,
                                      new Person("Obama","Barack"));
System.out.println(index);
```

The Java Collection API

```
package java.util;

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

http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html

Lists in the Java API



http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html

Java API List Interface

```
package java.util;

interface List<E> extends Collection<E> {
    E get(int index);
    int indexOf(Object o);
    int lastIndexOf(Object o);
    E remove(int index);
    E set(int index, E element);
    List<E> subList(int fromIndex, int toIndex)
}
```

http://docs.oracle.com/javase/7/docs/api/java/util/List.html

Using Java Collections

- Weiss exercise 3.1
  - You are given a list, `L`, and another list, `P`, containing integers sorted in ascending order.
  - The operation `printLots(L,P)` will print the elements in `L` that are in positions specified by `P`.
  - Implement this procedure using only methods of the Collections API.

L = [1 2 3 4 5 6]
P = [0 2 5]

printLots(L,P) →

1
3
6