Topic 14
Iterators

"First things first, but not necessarily in that order "

-Dr. Who

A Question

```
public class WordList {
  private ArrayList<String> myList;

  // pre: none
  // post: all words that are exactly len
  // characters long have been removed from
  // this WordList with the order of the
  // remaining words unchanged
  public void removeWordsOfLength(int len){
     for(int i = 0; i < myList.size(); i++){
          if( myList.get(i).length() == len )
               myList.remove(i);
     }
```


Attendance Question 1

- When does method `removeWordsOfLength` work as intended?

A. Always

B. Sometimes

C. Never

```
// original list = [“dog”, “cat”, “hat”, “sat”]
// resulting list after removeWordsOfLength(3) ?
```


The Remove Question

- Answer?

```
public void removeWordsOfLength(int len) {
  Iterator<String> it = myList.iterator();
  while( it.hasNext() )
     if( it.next().length() == len )
          it.remove();


  }
}
// original list = [“dog”, “cat”, “hat”, “sat”]
// resulting list after removeWordsOfLength(3) ?
```

Iterators

- ArrayList is part of the Java Collections framework
- Collection is an interface that specifies the basic operations every collection (data structure) should have
- Some Collections don't have a definite order
  - Sets, Maps, Graphs
- How to access all the items in a Collection with no specified order?

Access All Elements - ArrayList

```
public void printAll(ArrayList list){
    for(int i = 0; i < list.size(); i++)
        System.out.println(list.get(i));
}
```

- How do I access all the elements of a Set? The elements don't have an index.
- *Iterator* objects provide a way to go through all the elements of a Collection, one at a time

Iterator Interface

- An iterator object is a "one shot" object
  - it is designed to go through all the elements of a Collection once
  - if you want to go through the elements of a Collection again you have to get another iterator object
- Iterators are obtained by calling a method from the Collection

Iterator Methods

- The Iterator interface specifies 3 methods:

```
boolean hasNext()
//returns true if this iteration has more elements

Object next()
//returns the next element in this iteration
//pre: hastNext()

void remove()
/*Removes from the underlying collection the last element
   returned by the iterator.
  pre: This method can be called only once per call to next.
   After calling, must call next again before calling remove
   again.
*/
```

Attendance Question 2

- Which of the following produces a syntax error?

```
ArrayList list = new ArrayList();
Iterator it1 = new Iterator(); // I
Iterator it2 = new Iterator(list); // II
Iterator it3 = list.iterator(); // III
```

A. I
B. II
C. III
D. I and II
E. II and III

Typical Iterator Pattern

```
public void printAll(ArrayList list){
    Iterator it = list.iterator();
    Object temp;
    while( it.hasNext() ) {
        temp = it.next();
        System.out.println( temp );
    }
}
```

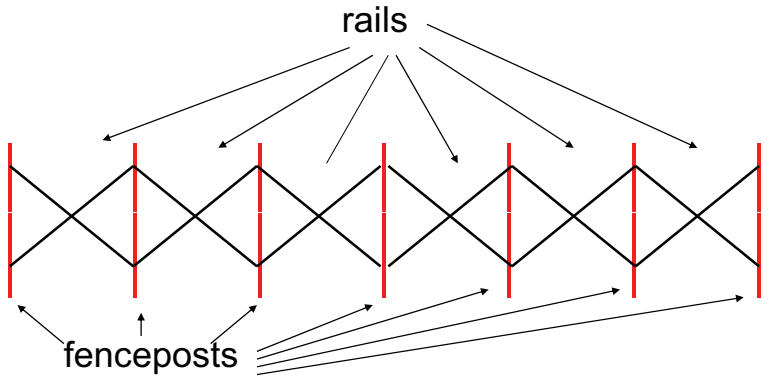

Typical Iterator Pattern 2

```
public void printAll(ArrayList list){
    Iterator it = list.iterator();
    while( it.hasNext() )
        System.out.println( it.next() );
}

// go through twice?
public void printAllTwice(ArrayList list){
    Iterator it = list.iterator();
    while( it.hasNext() )
        System.out.println( it.next() );
    it = list.iterator();
    while( it.hasNext() )
        System.out.println( it.next() );
}
```

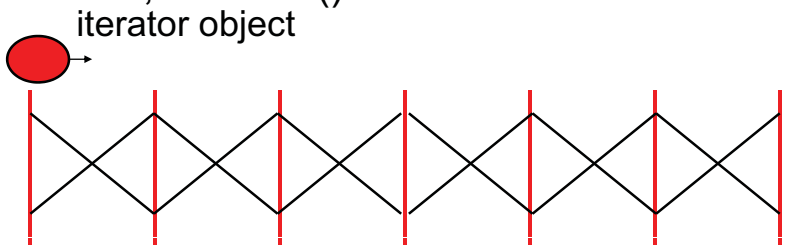

A Picture of an Iterator

- Imagine a fence made up of fence posts and rail sections



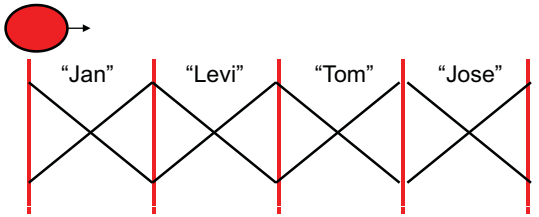
Fence Analogy

- The iterator lives on the fence posts
- The data in the collection are the rails
- Iterator created at the far left post
- As long as a rail exists to the right of the Iterator, hasNext() is true



Fence Analogy

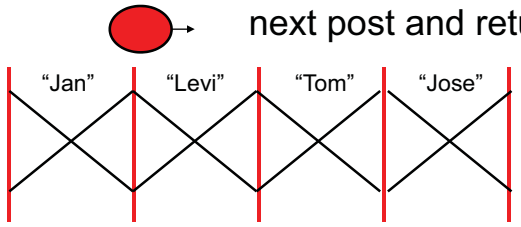
```
ArrayList<String> names =
                    new ArrayList<String>();
names.add("Jan");
names.add("Levi");
names.add("Tom");
names.add("Jose");
Iterator<String> it = names.iterator();
int i = 0;
```



Fence Analogy

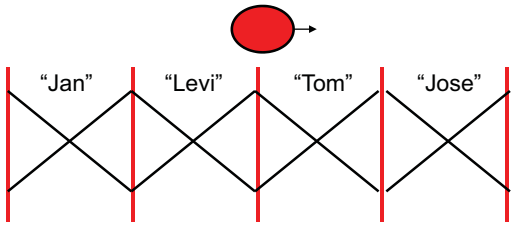
```
while( it.hasNext() ) {
    i++;
    System.out.println( it.next() );
}
// when i == 1, prints out Jan
```

first call to next moves iterator to next post and returns "Jan"



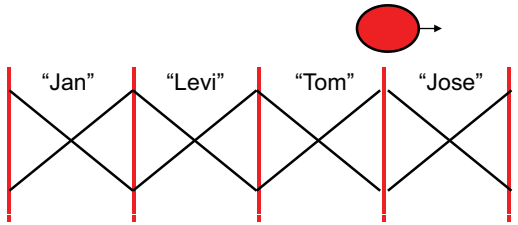
Fence Analogy

```
while( it.hasNext() ) {
     i++;
    System.out.println( it.next() );
}
// when i == 2, prints out Levi
```



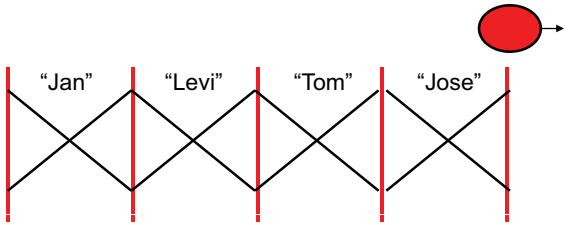
Fence Analogy

```
while( it.hasNext() ) {
     i++;
    System.out.println( it.next() );
}
// when i == 3, prints out Tom
```



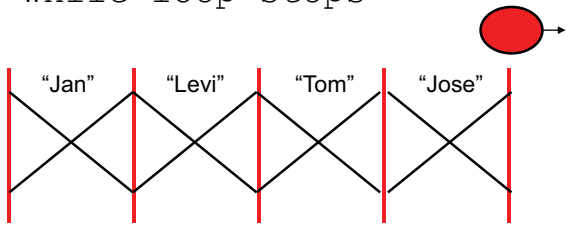
Fence Analogy

```
while( it.hasNext() ) {
     i++;
    System.out.println( it.next() );
}
// when i == 4, prints out Jose
```



Fence Analogy

```
while( it.hasNext() ) {
     i++;
    System.out.println( it.next() );
}
// call to hasNext returns false
// while loop stops
```



Attendance Question 3

- What is output by the following code?

```
ArrayList<Integer> list;
List = new ArrayList<Integer>();
list.add(3);
list.add(3);
list.add(5);
Iterator<Integer> it = list.iterator();
System.out.println(it.next());
System.out.println(it.next());
```

A. 3 B. 5 C. 3 3 5

D. 3 3 E. 3 5

Comodification

- If a `Collection` (`ArrayList`) is changed while an iteration via an iterator is in progress an Exception will be thrown the next time the `next()` or `remove()` methods are called via the iterator

```
ArrayList<String> names =
                    new ArrayList<String>();
names.add("Jan");
Iterator<String> it = names.iterator();
names.add("Andy");
it.next(); // exception will occur here
```


remove method

- Can use the `Iterator` to remove things from the `Collection`
- Can only be called once per call to `next()`

```
public void removeWordsOfLength(int len) {
  String temp;
  Iterator it = myList.iterator
  while( it.hasNext() ) {
     temp = (String)it.next();
      if( temp.length() == len )
          it.remove();
   }
}
// original list = ["dog", "cat", "hat", "sat"]
// resulting list after removeWordsOfLength(3) ?
```


Common Iterator Error

```
public void printAllOfLength(ArrayList<String> names,
                                        int len)
{   //pre: names != null, names only contains Strings
    //post: print out all elements of names equal in
    // length to len
    Iterator<String> it = names.iterator();
    while( it.hasNext() ){
        if( it.next().length() == len )
            System.out.println( it.next() );
    }
}
// given names = ["Jan", "Ivan", "Tom", "George"]
// and len = 3 what is output?
```


The Iterable Interface

- A related interface is `Iterable`
- One method in the interface:
  `public Iterator<T> iterator()`
- Why?
- Anything that implements the `Iterable` interface can be used in the for each loop.

```
ArrayList<Integer> list;
//code to create and fill list
int total = 0;
for( int x : list )
    total += x;
```

Iterable

- If you simply want to go through all the elements of a Collection (or Iterable thing) use the for each loop
  - hides creation of the Iterator

```
public void printAllOfLength(ArrayList<String> names,
                                              int len){
    //pre: names != null, names only contains Strings
    //post: print out all elements of names equal in
    // length to len
   for(String s : names){
      if( s.length() == len )
            System.out.println( s );
   }
}
```


Implementing an Iterator

- Implement an Iterator for our GenericList class
  - Nested Classes
  - Inner Classes
  - Example of encapsulation
  - checking precondition on remove
  - does our GenricList *need* an Iterator?