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

-Dr. Who
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?
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 Interface Methods

- The Iterator interface specifies 3 methods:

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

  E 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.
   */
  ```
Clicker Question 1

Which of the following produces a syntax error?

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

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

```java
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

"Jan"  "Levi"  "Tom"  "Jose"
Typical Iterator Pattern

```java
public void printAll(Collection<String> list) {
    Iterator<String> it = list.iterator();
    while (it.hasNext()) {
        String temp = it.next();
        System.out.println( temp );
    }
}
```
What is output by the following code?

```java
ArrayList<Integer> list;
list = new ArrayList<Integer>();
list.add(3);
list.add(3);
list.add(3);
list.add(5);
Iterator<Integer> it = list.iterator();
System.out.print(it.next() + " ");
System.out.print(it.next() + " ");
System.out.print(it.next());
```

A. 3  
B. 3 5  
C. 3 3 5  
D. 3 3  
E. 3 3 then a runtime error
remove method

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

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

// original list = ["dog", "cat", "hat", "sat"]
// resulting list after `removeWordsOfLength(3)` ?
```
Clicker Question 3

Given names = [“Jan”, “Ivan”, “Tom”, “George”] and len = 3 what is output by the printTarget method?

A. Jan Ivan Tom George
B. Jan Tom
C. Ivan George
D. No output due to syntax error
E. No output due to runtime error
The Iterable Interface

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

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

- 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

```java
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 GenericList *need* an Iterator?
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

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