Iterators

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

-Dr. Who

Iterator Interface

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

`Iterator` Interface Methods

- The `Iterator` interface specifies 3 methods:

```
boolean hasNext()
```

//returns true if this iteration has more elements

```
T next()
```

//returns the next element in this iteration

//pre: hasNext()

```
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.

*/

Question 1

- Which of the following produces a syntax error?

```
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`

Question 1

- Which of the following produces a syntax error?

```
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`

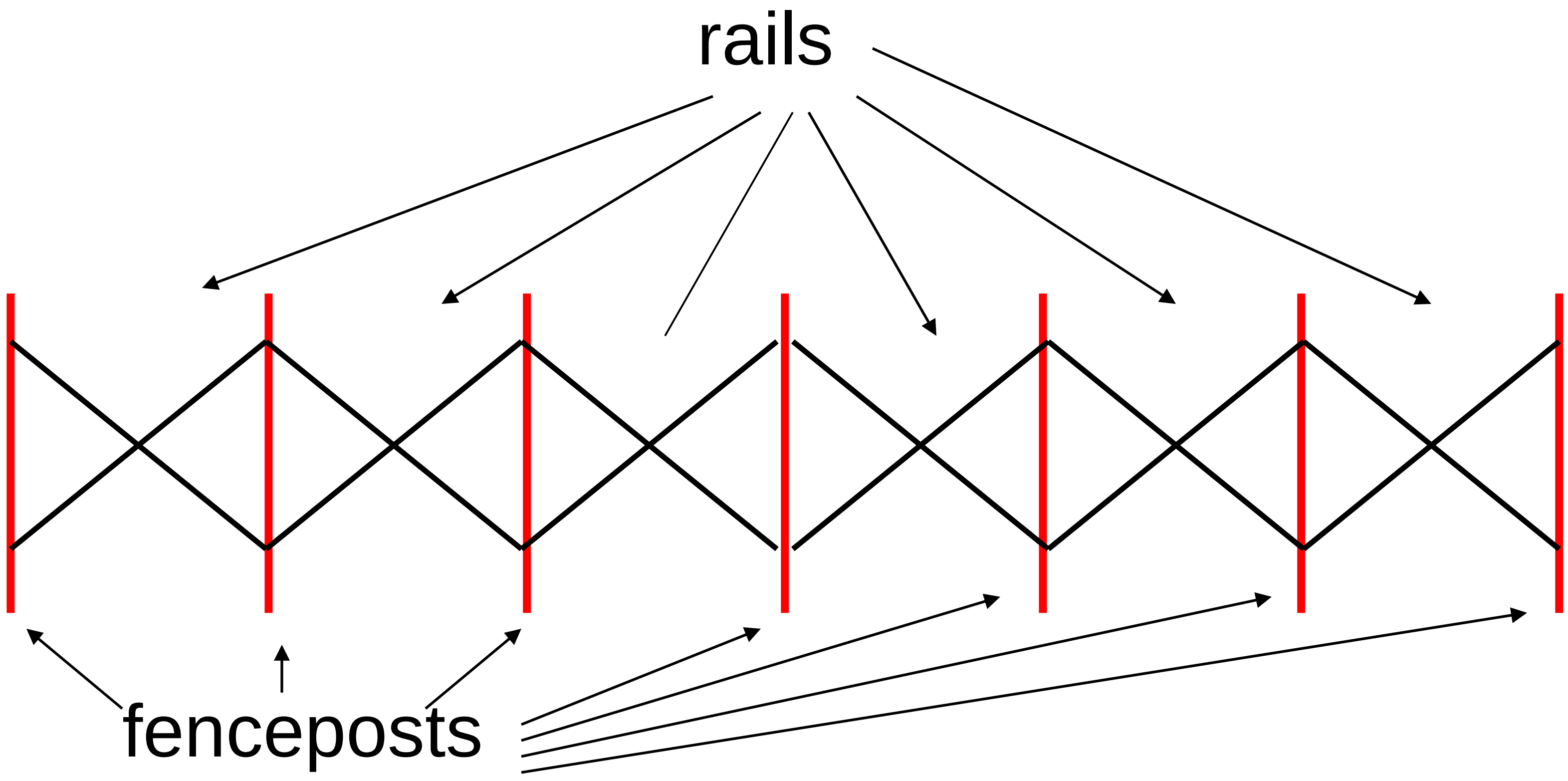
C. `III`

[✓ D.] `I and II`

E. `II and III`

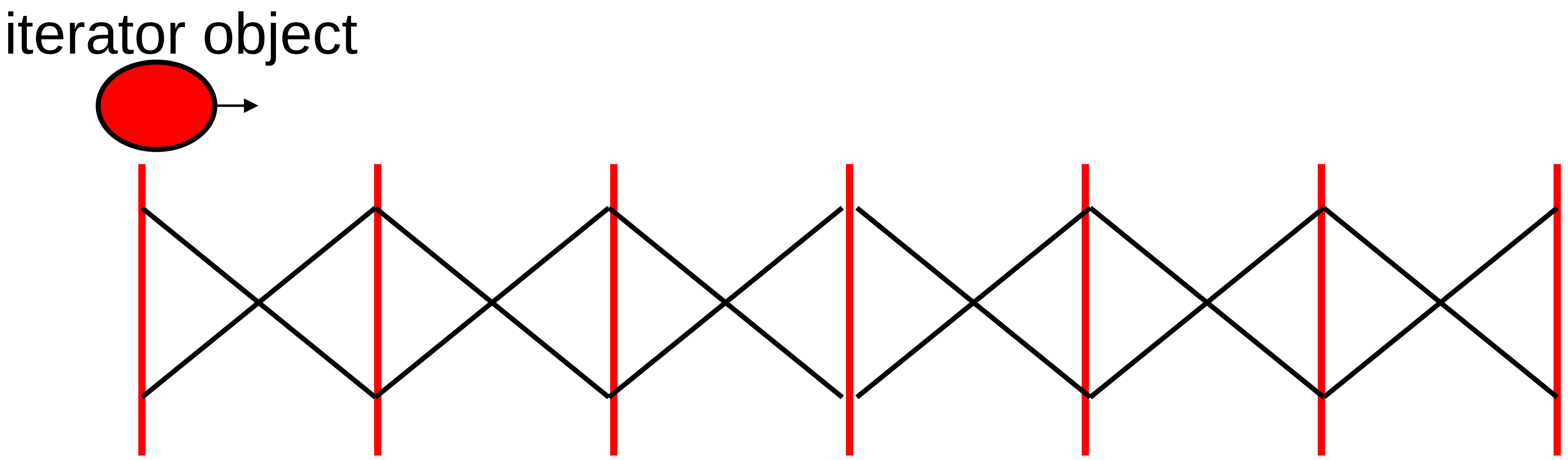
Iterator

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



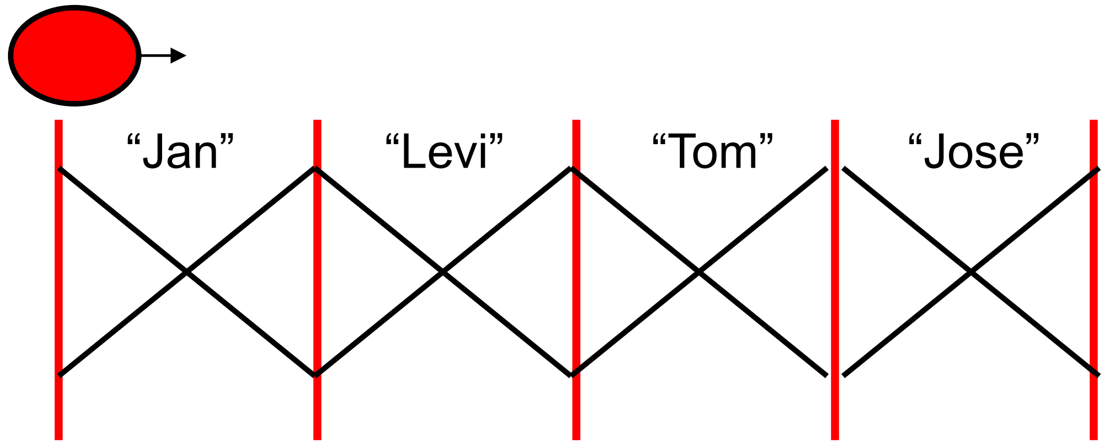
Fence Analogy

- 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

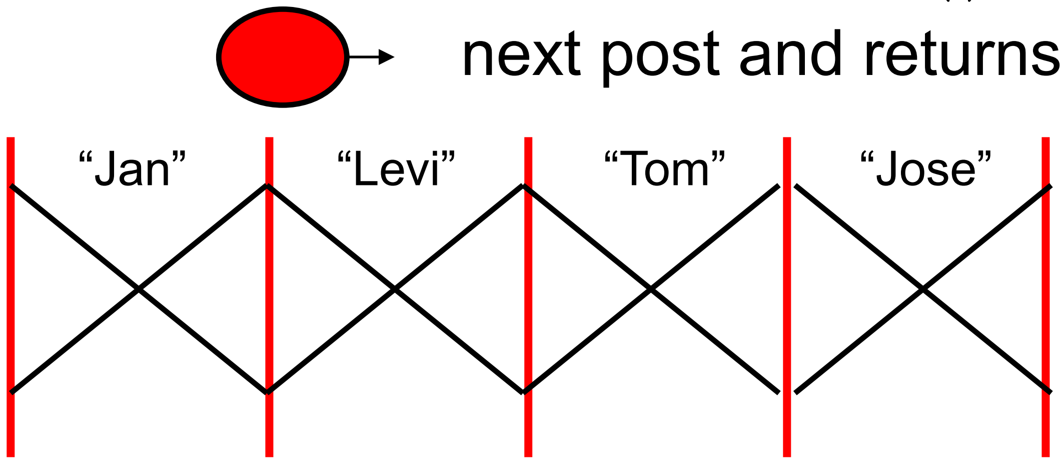
```
ArrayList<String> names = new ArrayList<String>();
names.add("Jan");
names.add("Levi");
names.add("Tom");
names.add("Jose");
Iterator<String> it = names.iterator();
int i = 0;
```



Fence Analogy

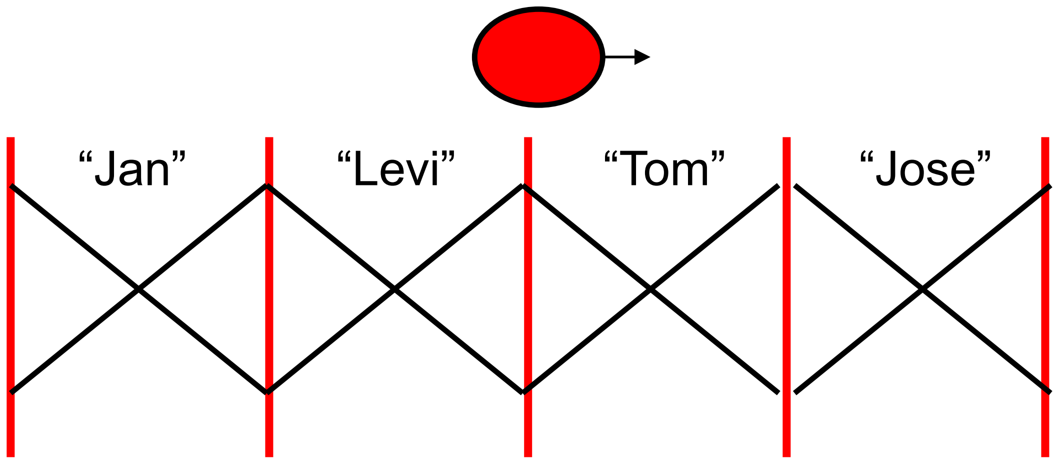
```
while( it.hasNext() )
{
    i++;
    System.out.println( it.next() );
}
// when i == 1, prints out Jan
```

first call to `next()` moves Iterator to next post and returns "Jan"



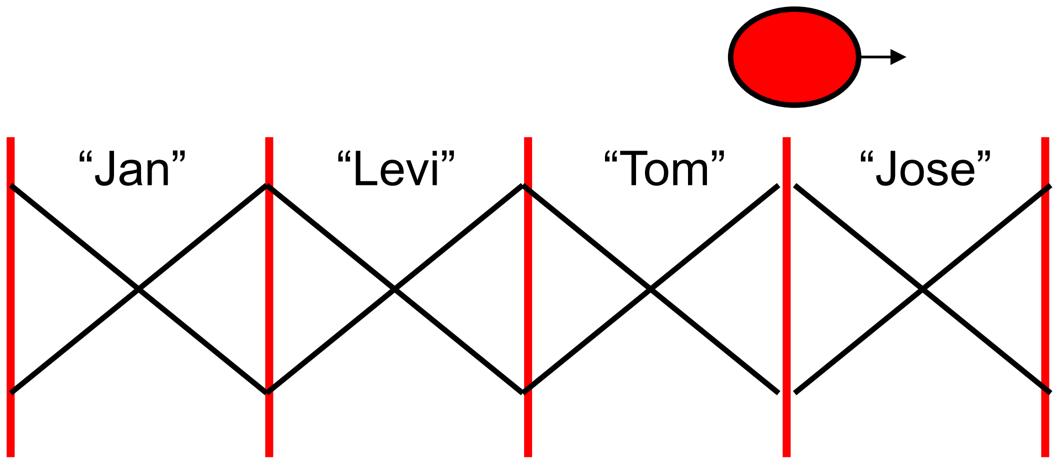
Fence Analogy

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



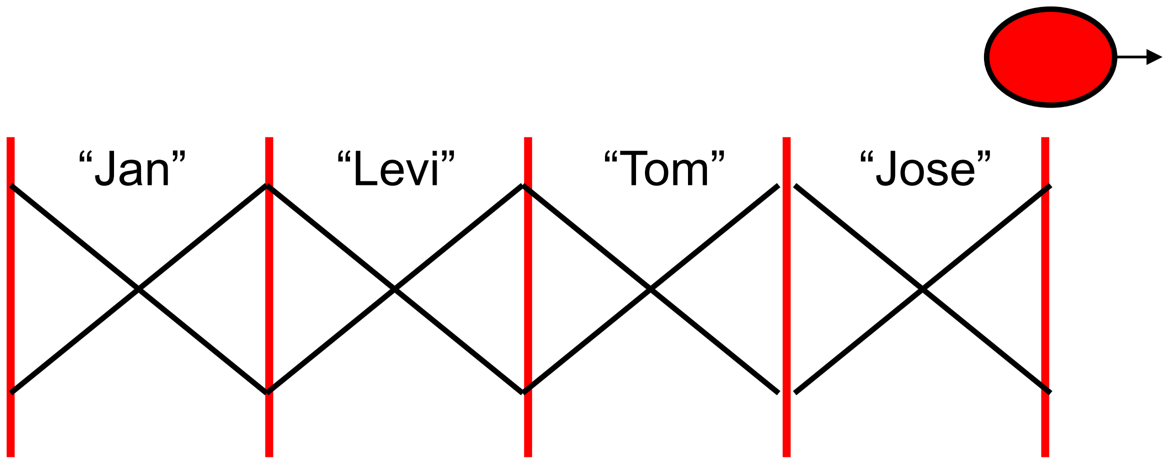
Fence Analogy

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



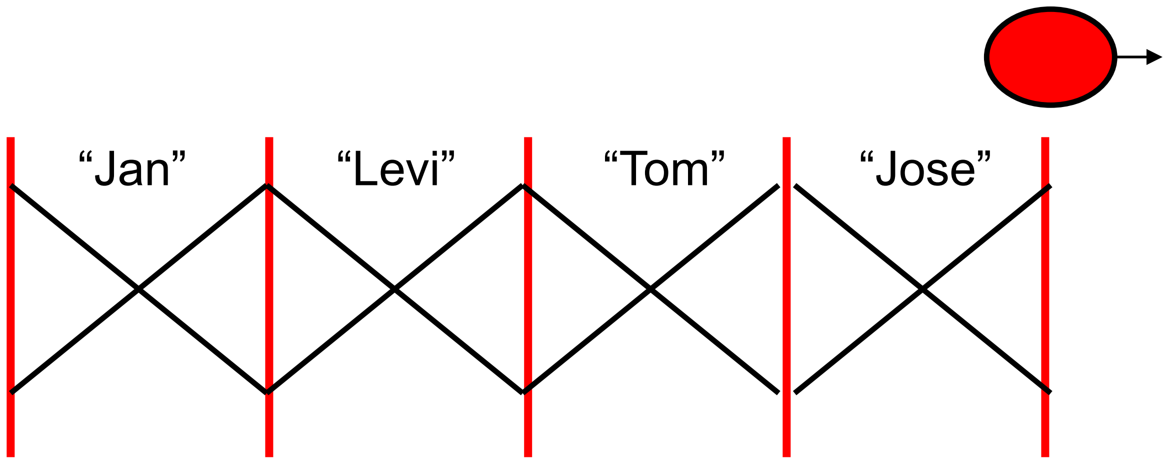
Fence Analogy

```
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
```



Typical Iterator Pattern

```
public void printAll(Collection<String> list)
{
    Iterator<String> it = list.iterator();
     while( it.hasNext() )
     {
          T temp = it.next();
          System.out.println( temp );
     }
}
```

Question 2

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

Question 2

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

remove method

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

```
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”, “fish”, “cat”, “gerbil”]
// resulting list after removeWordsOfLength(3) ?
```

Question 3

```
public void printTarget(ArrayList<String> names, int 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 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

Question 3

```
public void printTarget(ArrayList<String> names, int 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 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:

```
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 Collections

- If you simply want to go through all the elements of an ADT (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 );
    }
}
```

Iterable

- Can also use typical `for` loop

```
for (Iterator<T> itr = list.iterator();itr.hasNext(); )
{
      T element = iter.next();
      // can call methods of element
       ...
}
```

Implementing an Iterator

- Implement an `Iterator`
  - Nested / Inner Classes
  - Example of encapsulation
    - checking precondition on `remove` method
    - does our List class *need* an Iterator?

Comodification

- If a ADT with an Iterator is changed after the `Iterator` has been instantiated, an `ConcurrentModificationException` will be thrown the next time call `next()` or `remove()` methods

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