# csci 210: Data Structures

## Iterators

# Iterators

- An iterator abstracts the process of scanning through a collection of elements one at a time
- An iterator is a class with the following interface
  - `boolean hasNext()`
    - return true if there are elements left in the iterator
  - `Type  next()`
    - return the next element in the iterator

# Iterators in Java

- Java.util.Iterator interface
- All classes that implement collections of elements (Vectors, Lists, ArrayList, etc) have iterators
  - they have a method called "iterator()" which returns an iterator of the elements in the collection
- Example

```
ArrayList<Type> a;
//Vector<Type> a;
//Stack<Type> a;
//LinkedList<Type> a;

Iterator<Type> it = a.iterator();
while (it.hasNext()) {
    Type e = it.next();
    //process e
    //...
}
//or
for (Iterator<Type> it = a.iterator(); it.hasNext();) {
    Type e = it.next();
    //...
}
```

# Iterators in Java

- a Java specific for loop that uses iterators (under the hood)

```
Vector<Type> v;
for (Type x: v) {
    //x is the current element  in v and the loop iterates
    //through all elements of v
    System.out.print("the current element is " + x);
}
```

# Iterators

- Why use iterators?
  - They lead to more generic, high level code
  - They hide the details of the specific collection (linked list or array, or whatever else)
  - You can change the data structure, and the loop remains the same

# List iterators

- The preferred way to access a Java.util.LinkedList is through an iterator

| | |
|---|---|
| | **lastIndexOf**(Object o)<br>Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this elem |
| ListIterator | **listIterator**(int index)<br>Returns a list-iterator of the elements in this list (in proper sequence), starting at the specified position in the list. |
| Object | **remove**(int index) |

### listIterator

```
public ListIterator listIterator(int index)
```

Returns a list-iterator of the elements in this list (in proper sequence), starting at the specified position in the list. Obeys the general contract of `List.listIterator(int)`.

The list-iterator is *fail-fast*: if the list is structurally modified at any time after the Iterator is created, in any way except through the list-iterator's own `remove` or `add` methods, the list-iterator will throw a `ConcurrentModificationException`. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

**Specified by:**
`listIterator` in interface `List`
**Specified by:**
`listIterator` in class `AbstractSequentialList`

**Parameters:**
`index` - index of first element to be returned from the list-iterator (by a call to `next`).
**Returns:**
a ListIterator of the elements in this list (in proper sequence), starting at the specified position in the list.
**Throws:**
`IndexOutOfBoundsException` - if index is out of range (`index < 0 || index > size()`).
**See Also:**
`List.listIterator(int)`

- a ListIterator includes

| Method Summary | |
|---|---|
| void | **add**(Object o)<br>Inserts the specified element into the list (optional operation). |
| boolean | **hasNext**()<br>Returns `true` if this list iterator has more elements when traversing the list in the forward direction. |
| boolean | **hasPrevious**()<br>Returns `true` if this list iterator has more elements when traversing the list in the reverse direction. |
| Object | **next**()<br>Returns the next element in the list. |
| int | **nextIndex**()<br>Returns the index of the element that would be returned by a subsequent call to `next`. |
| Object | **previous**()<br>Returns the previous element in the list. |
| int | **previousIndex**()<br>Returns the index of the element that would be returned by a subsequent call to `previous`. |
| void | **remove**()<br>Removes from the list the last element that was returned by `next` or `previous` (optional operation). |
| void | **set**(Object o)<br>Replaces the last element returned by `next` or `previous` with the specified element (optional operation). |