Linked Lists (contd.)

List use examples

```
List<Employee> empList = new ArrayList<>();
Employee employee1 = new Employee(14567);
Employee employee2 = new Manager(45789, 345);
empList.add(employee1);
empList.add(employee2);
Manager mgr1 =  (Manager) empList.get(1);
empList.contains(employee1)  -- returns true
empList.remove(0);  // removes employee1
```

- Note that when comparing each element, it invokes each element's equals() method. For first element Employee class's equals() implementation (if it exists) will be called and for second element Manager class's equals() implementation (if it exists) will be called.

Iterator Design Pattern

- Allows a collection to control its own navigation order and hides the details from the consumer

```
public interface Iterator<E> {
    boolean hasNext();
    E next();
    …..
}
```

- Collection<E> has a method to get an iterator:

```
Iterator<E> iterator() { ..}
```

Collection Iterators

```
List<String> list = new ArrayList<>();
list.add("Hello");
list.add("!!");
list.add(1,"World");
```

- Old style code:

```
Iterator<String> lit = list.iterator();
while (lit.hasNext()) {
   System.out.println(lit.next());
} // prints " Hello","World","!!"
```

- New style code:

```
for (String str : list) {
    System.out.println(str);
}
```

LinkedList complexity

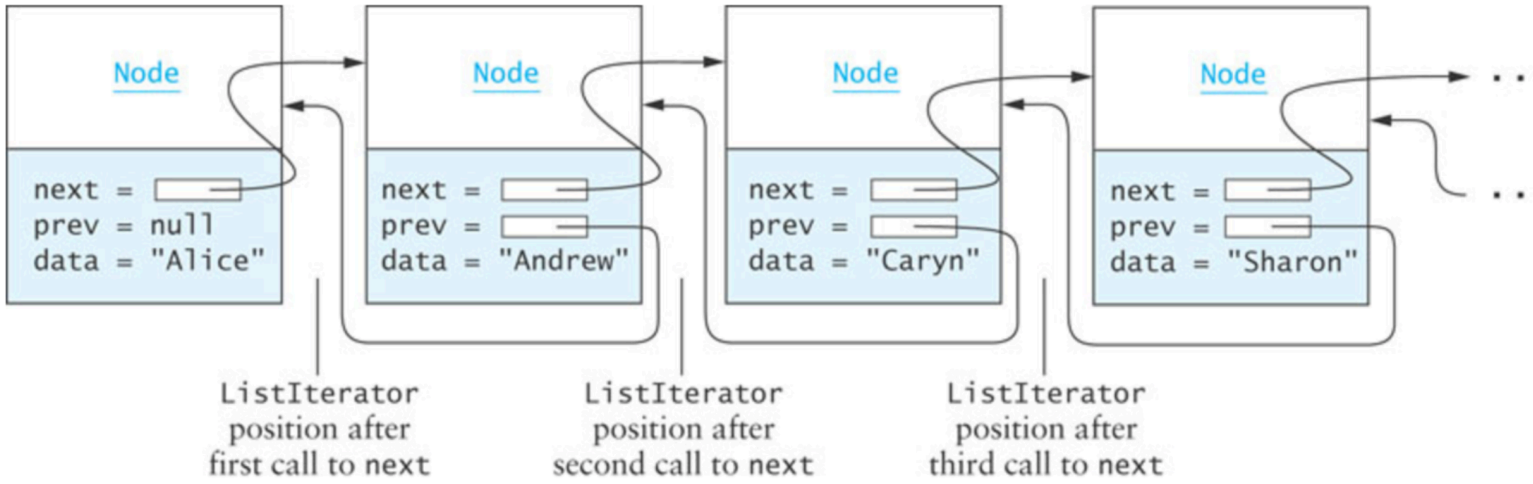
- get(index) – O(n) worst-case
- set(index() – O(n) worst-case
- add(e) – O(1) worst-case complexity
- add(index,e) – O(n) worst-case to find position + O(1) complexity for inserting new node; special case add(0,e) – O(1) worst-case
- remove(index) – O(n) worst-case to find position + O(1) complexity for removing node
- remove(0), remove(size()- 1) (special case) – O(1) worst complexity
- indexOf(elem) – O(n) worst-case

List Iterators

Java uses a the ListIterator ADT for linked lists. This iterator includes the following methods:

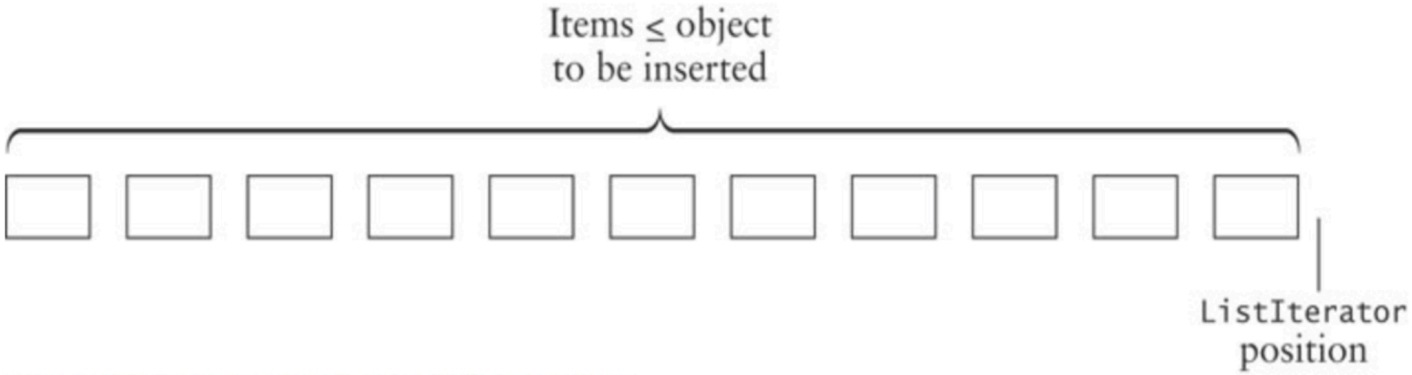
- add(e): add e at the current cursor position
- hasNext(): true if there is an element after the cursor
- hasPrevious: true if there is an element before the cursor
- previous(): return the element e before the cursor and move cursor to before e
- next(): return the element e after the cursor and move cursor to after e
- set(e): replace the element returned by last next or previous operation with e
- remove(): remove the element returned by the last next or previous method

Insertion into ordered linked list

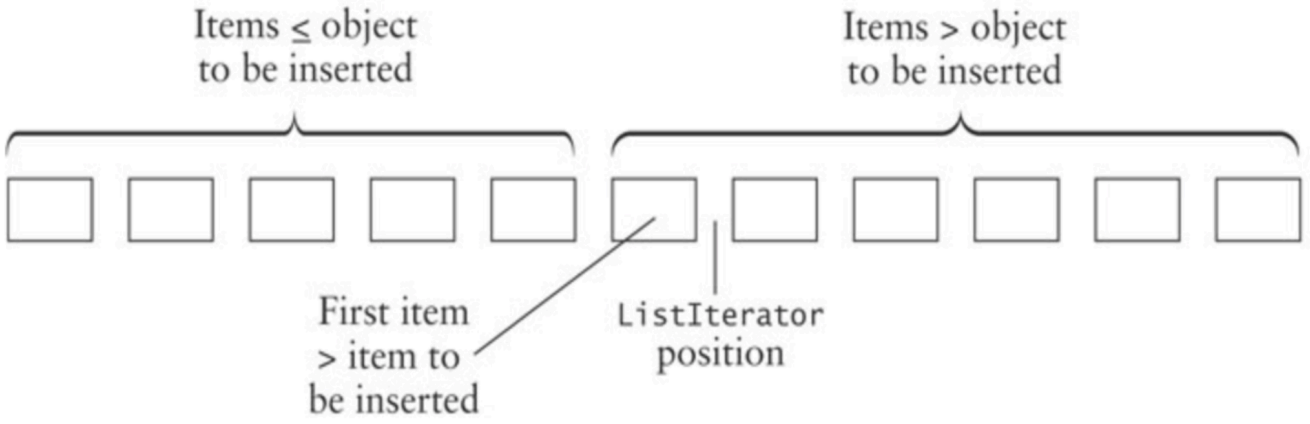


Insertion into ordered linked list

Case 1: Inserting at the end of a list



Case 2: Inserting in the middle of a list



Insertion into ordered linked list

```
public void add (E e) {
  ListIterator<E> iter = theList.listIterator();
  while (iter.hasNext()) {
    if (e.compareTo(iter.next()) < 0) {
      // found element > new one
      iter.previous();  // back up by one
      iter.add(e);      // add new one
      return;           // done
    }
  }
  iter.add(e);  // will add at end
}
```