Outline

- reminder: invariants
- Big-O Analysis of array list efficiency
- Linked Lists

Reminder: Invariants

- **a useful property of a program that the programmer knows and that is always true, is an *invariant***
  - in the ArrayList implementation one invariant is that the elements of data from 0..size-1 are all of type E
  - the invariant must be true whenever a public method is called and when a public method returns, but may not be true in the middle of a method body
    - every invariant must hold by the end of each constructor, that is, constructors must **establish** all the invariants
  - for example, the ArrayList class has the invariant that size <= data.length
  - in the ArrayList `add`, we may change size before we assign the new value: the invariant is temporarily broken when size is changed, then restored before the method completes

Reminder: Lists

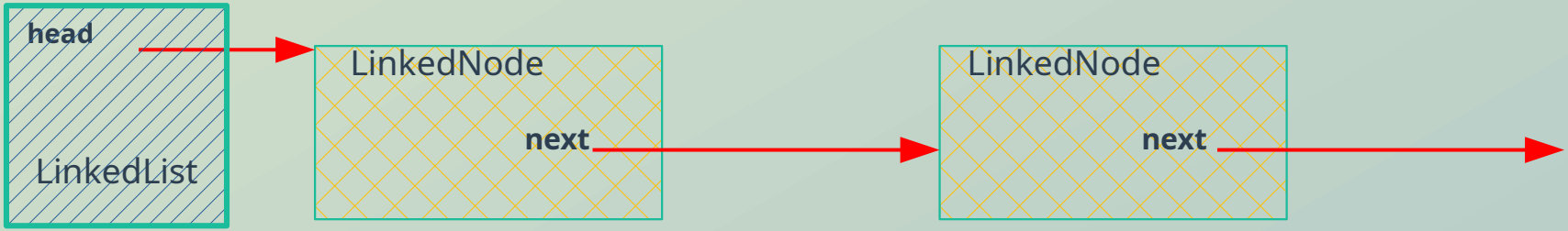
- **a list holds a sequence of values**
- **each value is at a specific index in the list**
- **we can insert items into a list and remove items from a list**
  - insert or remove at the beginning, at the end, or at a specific index

Linked Lists
and comparison with Array Lists

- in an array list, the elements of the list are stored in the array
- if more elements are needed, the array is replaced by a bigger array
- that means the elements are stored in a single object (an array) whose size may be different depending on the number of elements
- instead, consider storing the elements into a variable number of fixed-size objects
- where each object can only store one element
- there will be as many such objects as elements in the list

Linked List Nodes

- **objects used to store list elements are called Nodes**
- **a Linked List has zero or more nodes**
  - so there is a LinkedNode class and a LinkedList class
- **how to keep track of all these nodes?**
- **each of the nodes has a reference (*link*) to another node**



- **the LinkedList class has one variable to keep track of the first element, the *head* of the linked list**

Linked Lists Nodes in Java

```
private class LinkedNode<E> {
  private E item;                    // value in this node
  private LinkedNode<E> next;        // reference to the next node, or null

  private LinkedNode(E value) {      // constructor
    item = value;
    next = null;                     // last node in the linked list
  }
  private LinkedNode(E value, LinkedNode<E> reference) {    // constructor
    item = value;
    next = reference;
  }
}
```

Linked Lists Details

- **each node has two data fields: an element value, and a reference to the next node**
- **the reference to the next node is a pointer to the next node:**
  - the `next` reference of the last node in the linked list must be `null`
  - every other `next` reference in the linked list must refer to the next node

Linked Lists Implementation: class structure

- **LinkedNode is a private class inside LinkedList**
- **a private class is local to the enclosing (parent) class**
- **all data fields (meaning, all instance variables) in the private class are accessible to the code in the parent class**
  - even private instance variables are accessible to the code in the parent class!

LinkedList Implementation

- the linked list class only really needs one class variable: a reference to the start of the list, conventionally known as the head of the list
- the book also has a size class variable, which can be useful for error checking
- my implementation (and most other implementations) also keeps a reference to the last node in the linked list, the tail of the list

In-Class Exercises

- for a linked list:
- what is the run-time performance of add(E)?
- what is the run-time performance of add(int, E)?
- what is the run-time performance of remove(), which removes the head of the linked list?
- what is the run-time performance of remove(int), which removes an element at an arbitrary position?