Linked Lists

* We can conceal internal details of the linked list using inner classes
* In linked lists, data are packaged into “nodes”, and each node has a reference to the next node in the list
* It’s natural that a node should itself be a class e.g.

public class LinkedListNode {

 private String data;

 private LinkedListNode next;

 // …

}

* The issue is that nothing outside of the linked list needs a LinkedListNode – it’s only useful inside the LinkedList
* It might be confusing to the client to have a LinkedListNode alongside LinkedList – because the client will never use the LinkedListNode
* The solution is to make LinkedListNode a private class inside LinkedList

Inner Classes

* An inner class may be private or public
* If private, the inner class and all its features are completely hidden from the client … even if they are “public”!
* Moreover the outer (containing) class has full access to all features of the inner class, regardless of access specifier (private/public)
* And vice versa: inner has access to everything in the outer
* If I create an object of the outer class, an object of the inner class is NOT automatically created
* HOWEVER … in order to create an object of the inner class type, I must first create an outer class object
* A key advantage of inner classes is this:
* Say you want to implement an interface, and have access to everything in the LinkedList … BUT … you don’t want the LinkedList to directly implement the interface.
* For example, iterator
* Solution: Create a private inner class that implements the interface (here, iterator) – then you can return from the class an instance of the interface

What happens when inner classes are public?

* The client can create an object of inner class type
* Every inner class must be associated with an object of outer class type

To create an object of inner type:

Outer.Inner x = (new Outer()).new Inner();

To access a feature (e.g. x) of the outer from the inner:

Outer.this.x

However, this is only necessary if there is a name collision