Doubly Linked Lists and Circular Lists

Weiss, section 6.9 (pages 274–276)
Doubly Linked Lists

- With singly linked lists, it is difficult to traverse back. You need to keep a reference to the previous node.
- You need to have more references if you access the previous elements.
- Maintain two links per node: one forward and one backward.

| prev | Reference to element | next |

- Usually, a doubly linked list has special nodes at both ends of the list.
- Header node before the head of the list and trailer node after the tail of the list.
- These nodes are called dummy (sentinel) nodes.
Doubly Linked Lists

- Dummy nodes eliminate null pointer cases. It ensures that next and prev references exist for every element.
- Instead of checking current.next == null, check if
  - current.next == trailer
  - current.prev == header
- For many short lists, sentinel nodes use up extra space. To avoid that, sentinels are reused to reference the first and last nodes.
Doubly Linked Lists

```java
public class DNode {
    private Integer element;
    private DNode next, prev;

    public DNode(Integer e, DNode p, DNode n) {
        element = e;
        prev = p;
        next = n;
    }

    public Integer getElement() { return element; }

    public DNode getPrev() { return prev; }

    public DNode getNext() { return next; }

    public void setElement(Integer newElement) { element = newElement; }

    public void setPrev(DNode newPrev) { prev = newPrev; }

    public void setNext(DNode newNext) { next = newNext; }
}
```
Doubly Linked Lists

Add a new node (z) after node v.

```
addAfter(v, z) {
    w = v.getNext();
    v.setNext(z);
    z.setPrev(v);
    z.setNext(w);
    w.setPrev(z);
}
```
Doubly Linked Lists

Remove node $z$

```
remove(z) {
    v = z.getPrev();
    w = z.getNext();
    w.setPrev(v);
    v.setNext(w);
    z.setPrev(null);
    z.setNext(null);
}
```
Circularly Linked List

- Circularly linked list is an extension of the singly or doubly linked list.
- The next pointer of last node points back to the first node. There is no first or last node.
- Traverse the list from any node using a cursor.
Circularly Linked List

```java
public class SNode {
    private Integer element;
    private SNode next;

    public SNode(Integer e, SNode n) {
        element = e;
        next = n;
    }

    public Integer getElement() { return element; }
    public SNode getNext() { return next; }
    public void setElement(int newElement) { element = newElement; }
    public void setNext(SNode newNext) { next = newNext; }
}
```
public class CircularList {
    private SNode cursor;
    private int size;

    public CircularList() {
        cursor = null;
        size = 0;
    }

    public void advance() { cursor = cursor.getNext(); }

    public void add( SNode newNode ) {
        if( cursor == null ) {
            newNode.setNext(newNode);
            cursor = newNode;
        } else {
            newNode.setNext(cursor.getNext());
            cursor.setNext(newNode);
        }
        size++;
    }
}
public SNode remove() {
    SNode oldNode = cursor.getNext();
    if (cursor == oldNode) {
        cursor = null;
    } else {
        cursor.setNext(oldNode.getNext());
        oldNode.setNext(null);
    }
    size--;
    return oldNode;
}
Nested Class

• Java allows to define a class inside another class.
• Nested classes are either static or non-static. Non-static classes are called *inner* classes.
• A nested class is a member of its enclosing class.
• Nested static class accesses the instance members of the enclosing class via the objects of the enclosing class.
• Inner classes have direct access to the members of the enclosing class.

• Nested classes are useful due to
  ▪ Encapsulation. A nested class is hidden inside the enclosing class.
  ▪ More readability. A nested class is put closer to where it is used.

```java
class OuterClass {
    static class StaticNestedClass{ ... }
    class InnerClass { ... }
}
```
Nested Class

- Static nested classes are called using the *name* of the *enclosing* class. `OuterClass.StaticNestedClass staticObject = new OuterClass.StaticNestedClass();`

- Inner classes are called using the *instance* of the *enclosing* class. `OuterClass.InnerClass innerObject = outerObject.new InnerClass();`
Nested Class

x is the parameter in methodInFirstLevel method.
x is an member variable in FirstLevel inner class.
x is a member variable in ShadowTest outer class.

```java
public class ShadowTest {
    private int x;

    public ShadowTest(int newx) {
        this.x = newx;
    }

    class FirstLevel {
        private int x;

        public FirstLevel(int newx) {
            this.x = newx;
        }

        public void methodInFirstLevel(int x) {
            System.out.println("x = " + x);
            System.out.println("this.x = " + this.x);
            System.out.println("ShadowTest.this.x = " + ShadowTest.this.x);
        }
    }
}
```
Nested Class

- Parameter in methodInFirstLevel **shadows** the member variable in the inner class.
- Use `this` for the inner class and `ShadowTest.this` for the outer class.

```java
public static void main(String[] args) {
    ShadowTest st = new ShadowTest(2);
    ShadowTest.FirstLevel fl = st.new FirstLevel(1);
    fl.methodInFirstLevel(23);
}
```

*Output:*

23
1
2