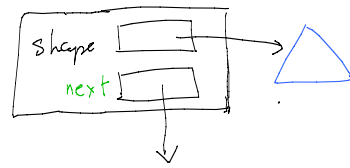


lecture 5

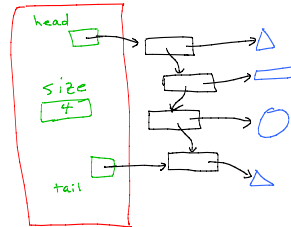
(more Java today - to be concrete)

- move on singly linked lists
- generic types in Java
- doubly linked lists

Last lecture



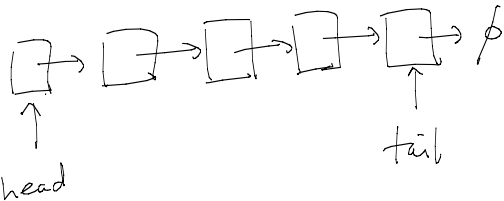
```
class SNode {
    Shape shape;
    SNode next;
    ...
}
```



```
class SLinkedList {
    SNode head;
    SNode tail;
    int size;
    ...
}
```

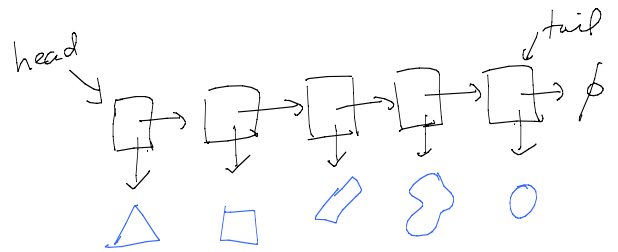
Singly Linked Lists

Last lecture we concentrated on how to add & remove nodes.



e.g. add First, addLast, removeFirst, removeLast

Recall that each node has an element.



How to manipulate nodes and elements?

```
class SNode {
```

```
    Shape shape;
    SNode next;
```

```
    SNode() {}
```

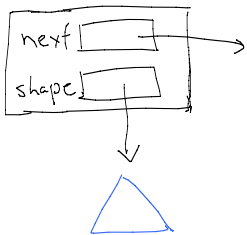
```
    Shape getElement() {}
```

```
    void setElement(Shape s) {}
```

```
    SNode getNext() {}
```

```
    void setNext(SNode n) {}
```

```
}
```

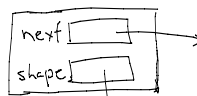


```
class SNode {
```

```
    Shape shape;
    SNode next;
```

// constructor

```
SNode() {}
```

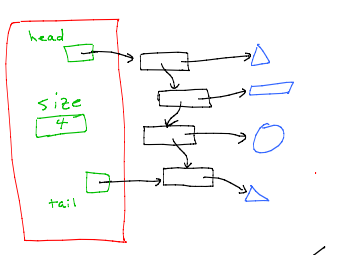


```
    Shape getElement()
        return shape;
}
```

```
void setElement(Shape s) {
    shape = s;
}
```

```
void setNext(SNode n) {
    next = n;
}
```

```
SNode getNext() {
    return next;
}
```



```

class SLinkedList {
    SNode head;
    SNode tail;
    int size;
    SLinkedList() {

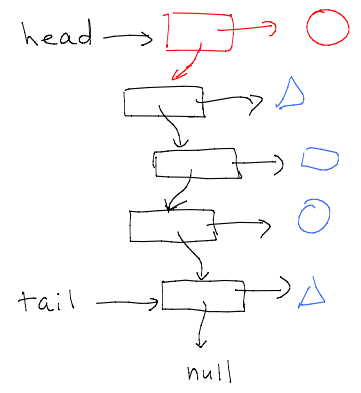
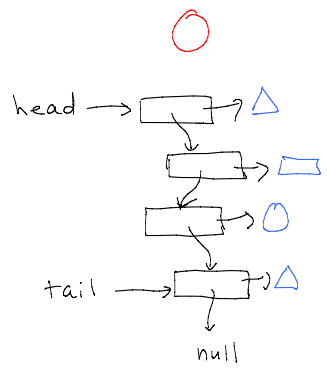
```

```

void addFirst(Shape s) {
    ...
}
void addLast(Shape s) {
    ...
}
Shape removeFirst() {
    ...
}
Shape removeLast() {
    ...
}
}

```

Example: addFirst(Shape s)
 BEFORE AFTER



Example

```

public void addFirst(Shape s) {
    SNode node = new SNode();
    node.shape = s;
    if (head == null) {
        head = node;
        tail = node;
    } else {
        node.next = head;
        head = node;
    }
}

```

Lists are useful.

But do we want to re-implement all these methods for every class we define?
 (Shape, Dog, BankAccount, ...)

NO!

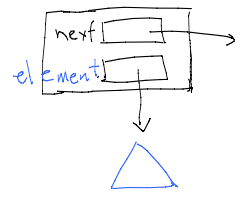
Java Generic Types

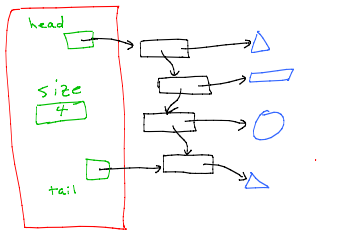
(allow us to work with any type.)

```

class SNode<E> {
    E element;
    SNode<E> next;
    E getElement() {
        return element;
    }
    void setElement(E s) {
        element = s;
    }
    void setNext(SNode<E> n) {
        next = n;
    }
    SNode<E> getNext() {
        return next;
    }
}

```





```
class SLinkedList<E> {
    SNode<E> head
    SNode<E> tail
    int size
}
SLinkedList()
```

```
void addFirst(E s) {
    ...
}
void addLast(E s) {
    ...
}
E removeFirst() {
    ...
}
E removeLast() {
    ...
}
```

```
SLinkedList<Dog>
    singlylist = new SLinkedList<Dog>();

SLinkedList<Shape>
    shapelists = new SLinkedList<Shape>();
```

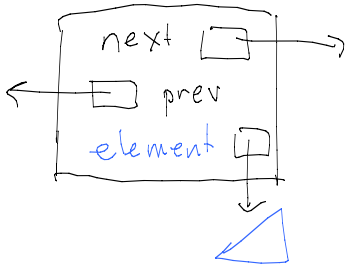
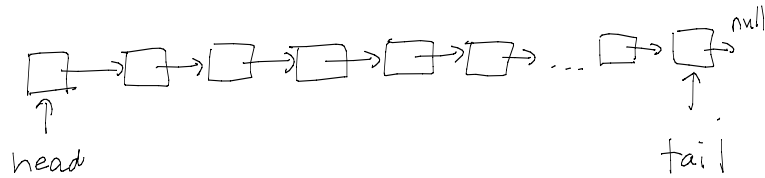
Generic type is basically a parameter passed to the SLinkedList constructor.

See online code.

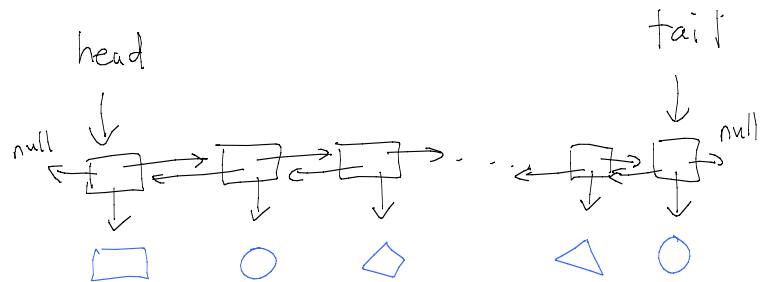
- Several methods implemented
 - addFirst ... removeLast
 - ...
- You are expected to read it and play with it (and report bugs :))

Doubly Linked Lists

Motivation: recall problem from last class that removeLast() requires N steps.



```
class DNode<E> {
    E element
    DNode<E> next
    DNode<E> prev
}
```

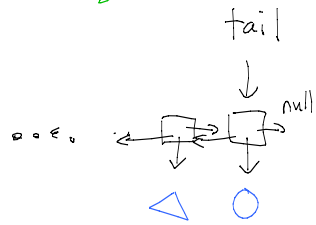


```

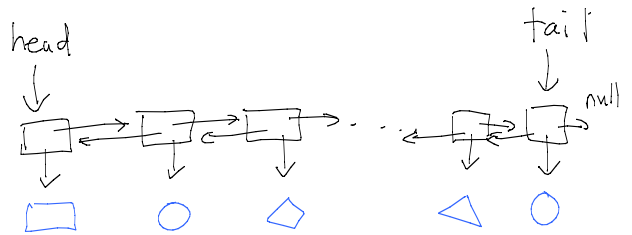
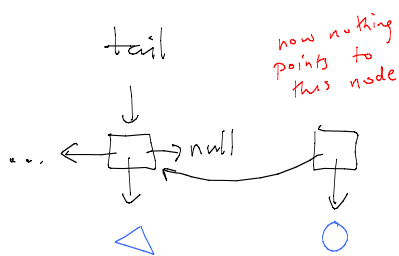
remove Last ( ) {
    tail = tail . prev
    tail . next = null
}

```

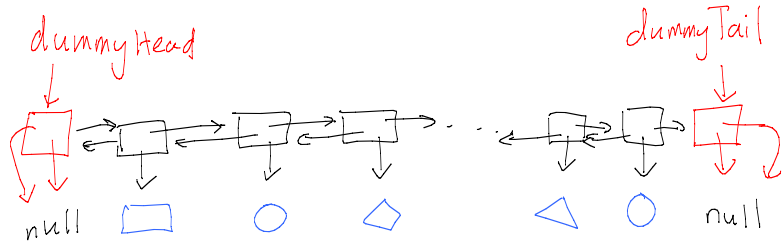
BEFORE



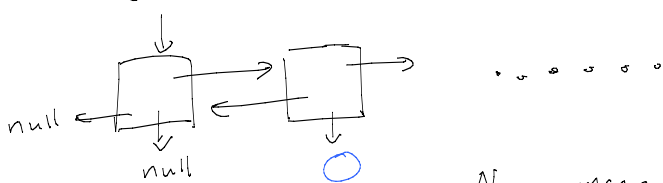
AFTER



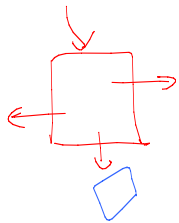
An alternative data structure



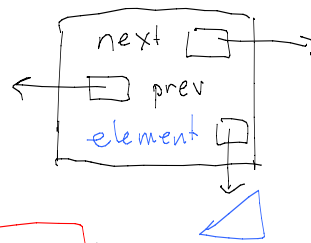
dummyHead



new Node



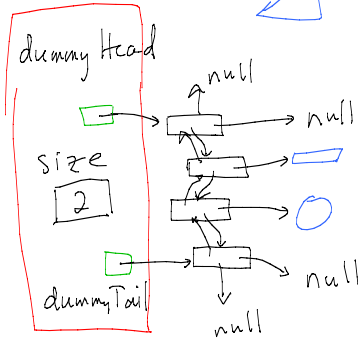
Now manipulating the front or back of the list does not require special treatment eg. Compare with addFirst from SLinkedList earlier this lecture.



```

class DNode<E> {
    E element
    DNode<E> next
    DNode<E> prev
}

```



```

class DLinkedList<E> {
    DNode<E> dummyHead
    DNode<E> dummyTail
    int size
}

```

Algorithms

| | array | singly linked list | doubly linked list |
|--------------|-------|--------------------|--------------------|
| add First | N | | |
| remove First | N | | |
| add Last | | | |
| remove Last | | N | |
| ... | ? | ? | ? |