Lists

CSC212 Lecture 8
D. Thiebaut, Fall 2014

Review

- **List =**
  - **Organization** of Data in a Linear Fashion, where **Order is Important**
  - **Set of actions** that can be carried out *efficiently* on the data.

Typical **Actions**

- **Append** at front, at end
- **Insert** in the middle
- **Remove** from front, from end, in middle
- Get the **length**
- Test if **empty**
- **Iterate** over all elements
- **Sort** the list

Can you think of a particular set of data (in every-day life) you would want to keep in a list?

Three Examples

- **Road race**: Keep track of arrival time for all runners (Number, Time). What operations can you imagine?

- **List of people** who…

- List of **email addresses** of…

Review
Vectors

Vector (Java Platform SE 7

docs.oracle.com/javase/7/docs/api/java/util/Vector.html

Apps Cal DTWiki2 DTWiki GDriveS 212 231 US Test 231-2012 track CS Google Drv

Overview Package Class Use Tree Deprecated Index Help

Prev Class Next Class Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.util

Class Vector<E>

java.lang.Object
  java.util.AbstractCollection<E>
    java.util.AbstractList<E>
      java.util.Vector<E>

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

Direct Known Subclasses:

Stack

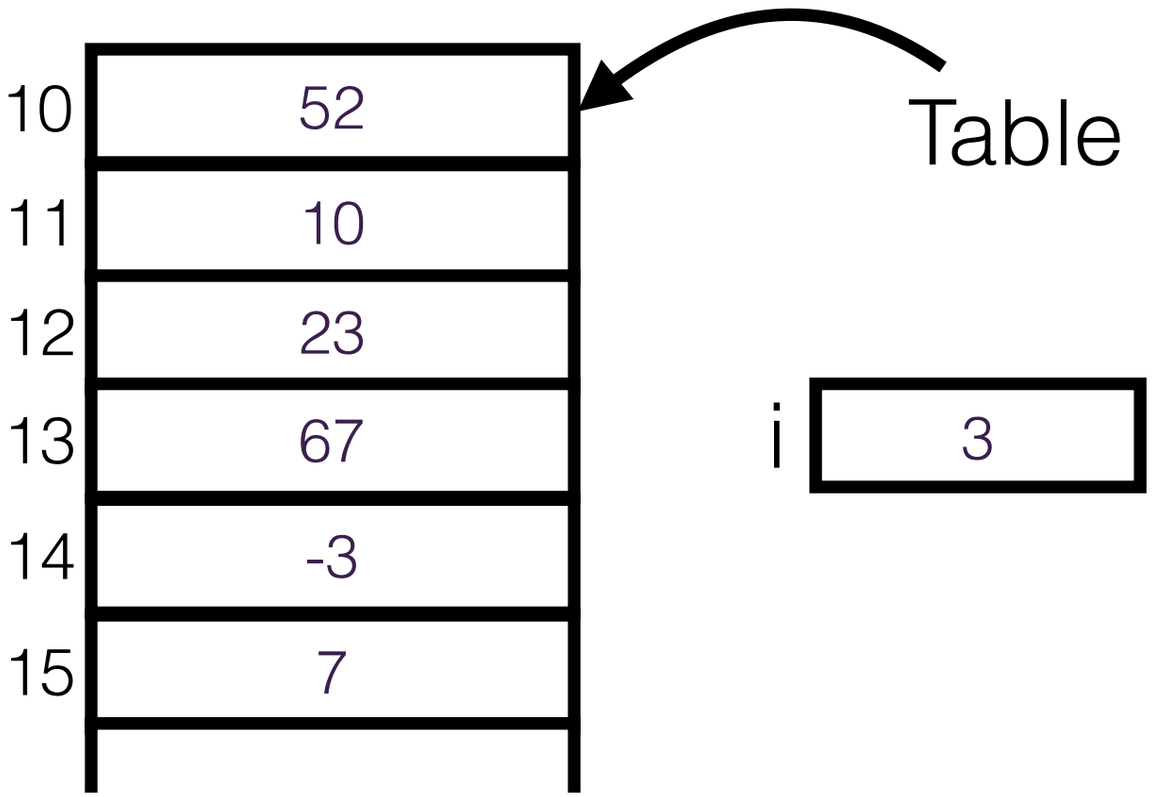
```
public class Vector<E>
extends AbstractList<E>
implements List<E>, RandomAccess, Cloneable, Serializable
```

A Note on Notation…

```
public class Generic1<T> {
    private T info;

    Generic1( T x ) {
        info = x;
    }

    public String toString() {
        return "info = "  + info;
    }

    static public void main( String[] args ) {
        Generic1<Integer> n = new Generic1( 100 );

        Generic1<String> s = new Generic1( "Hello there!" );

        System.out.println( n );
        System.out.println( s );
    }
}
```

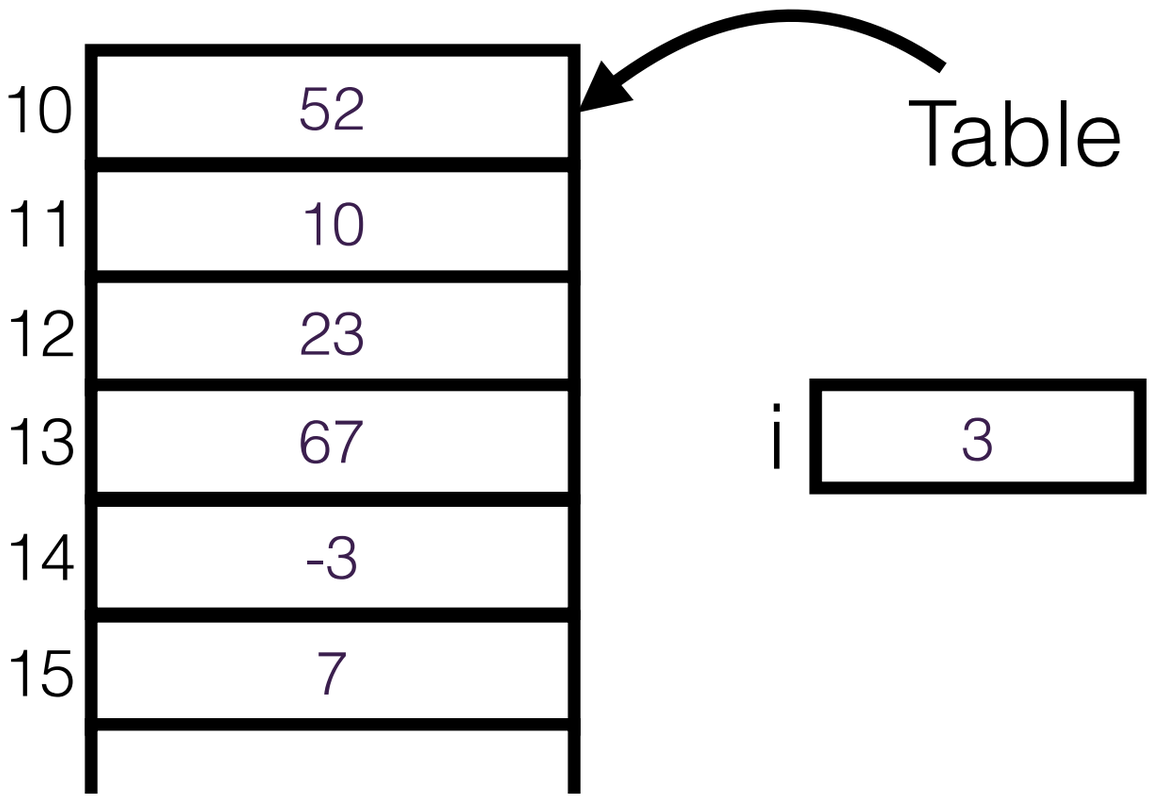
```
public class Generic2 {
    private Object info;

    Generic2( Object x ) {
        info = x;
    }

    public String toString() {
        return "info = "  + info;
    }

    static public void main( String[] args ) {
        Generic2 n = new Generic2( (Integer) 100 );

        Generic2 s = new Generic2( (String) "Hello there!" );

        System.out.println( n );
        System.out.println( s );
    }
}
```

A Small Detour…
Assembly Language



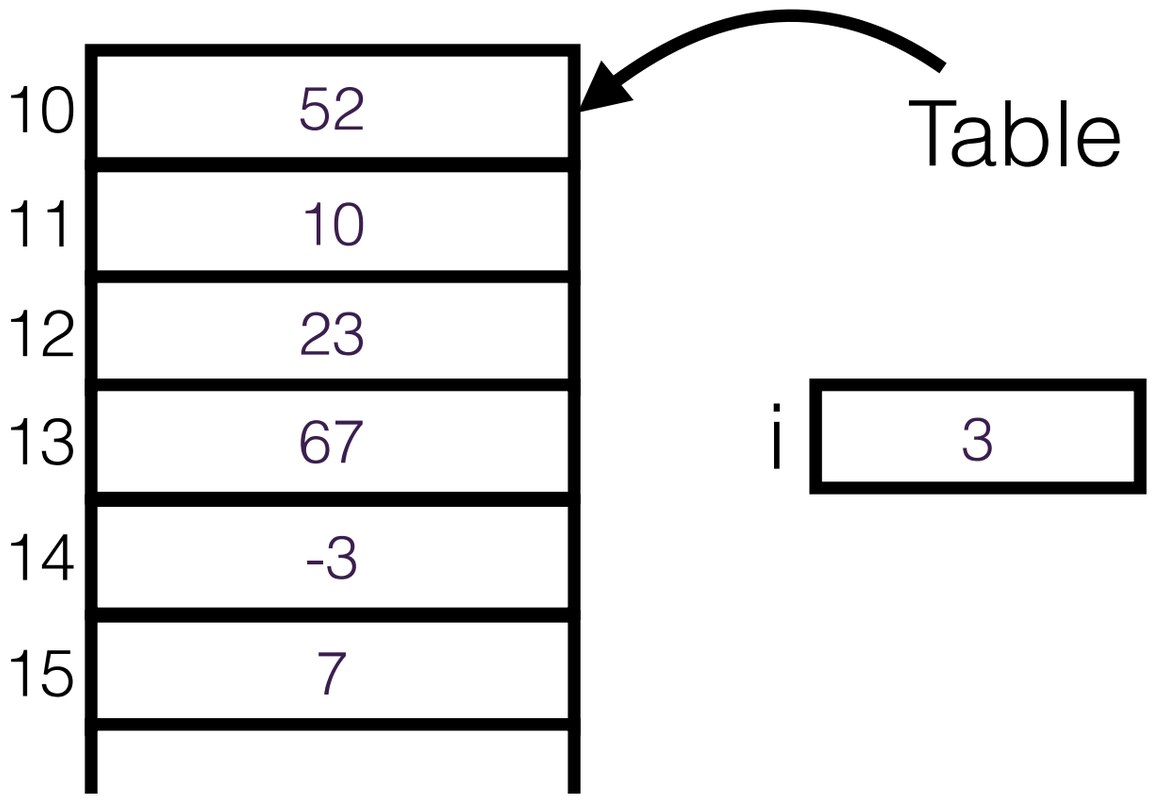

```
; Table[i] = 0
```

A Small Detour…
Assembly Language




```
; Table[i] = 0
  mov reg, Table
  add reg, mem(i)
  mov mem(reg), 0
```

A Small Detour…
Assembly Language




```
; Table[i] = 0
  mov reg, Table
  add reg, mem(i)
  mov mem(reg), 0
```

Accessing data in an array (vector) = CONSTANT TIME
~ 3 ns

Evaluating the Efficiency of Vectors:

- Adding item at **tail**
- Adding item in **front**
- Adding item in **middle**
- **Iterate** over all elements
- Look at **first** item in list
- Look at **last** item in list

A JAVA EXAMPLE

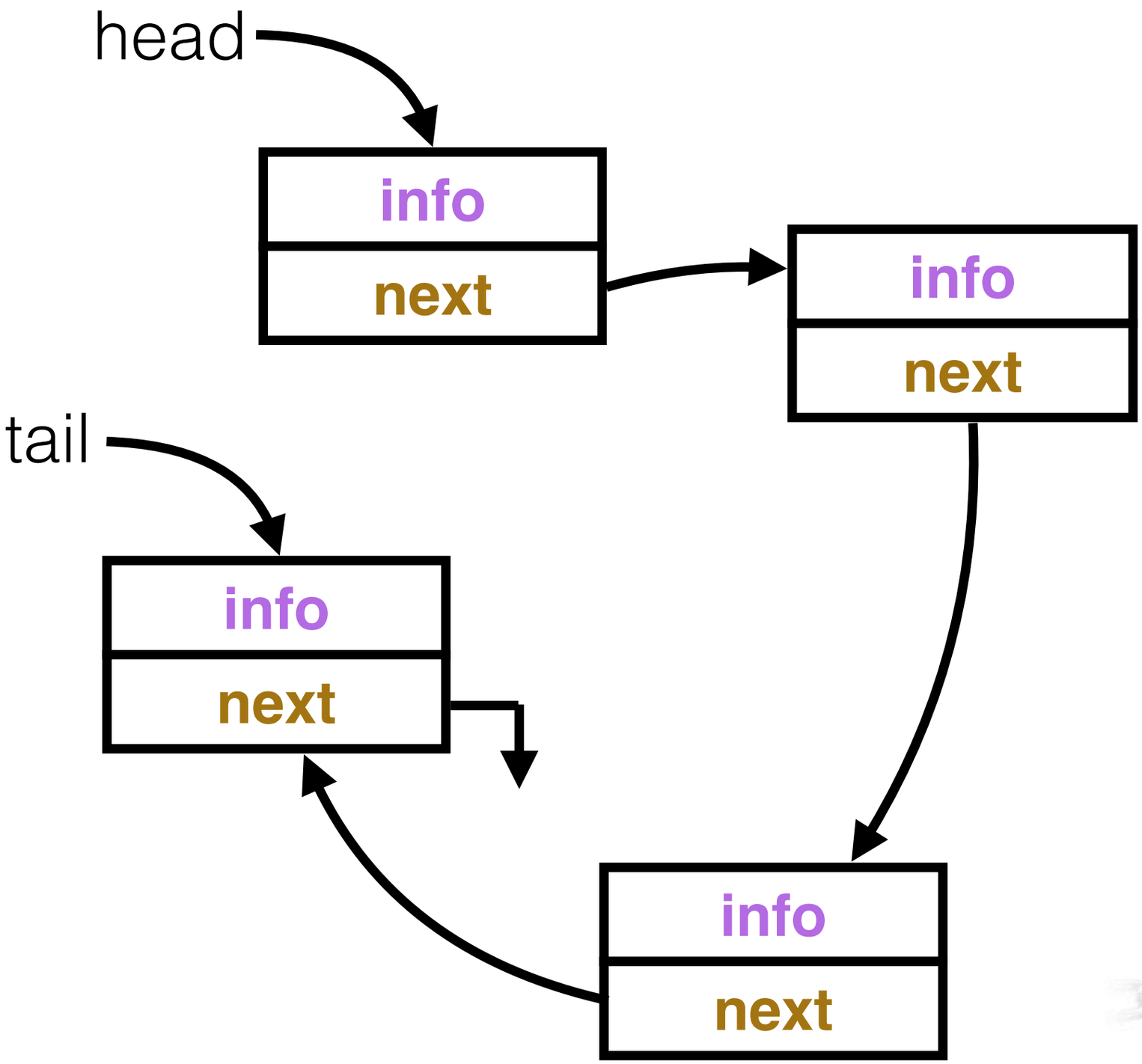
2 Different Syntaxes

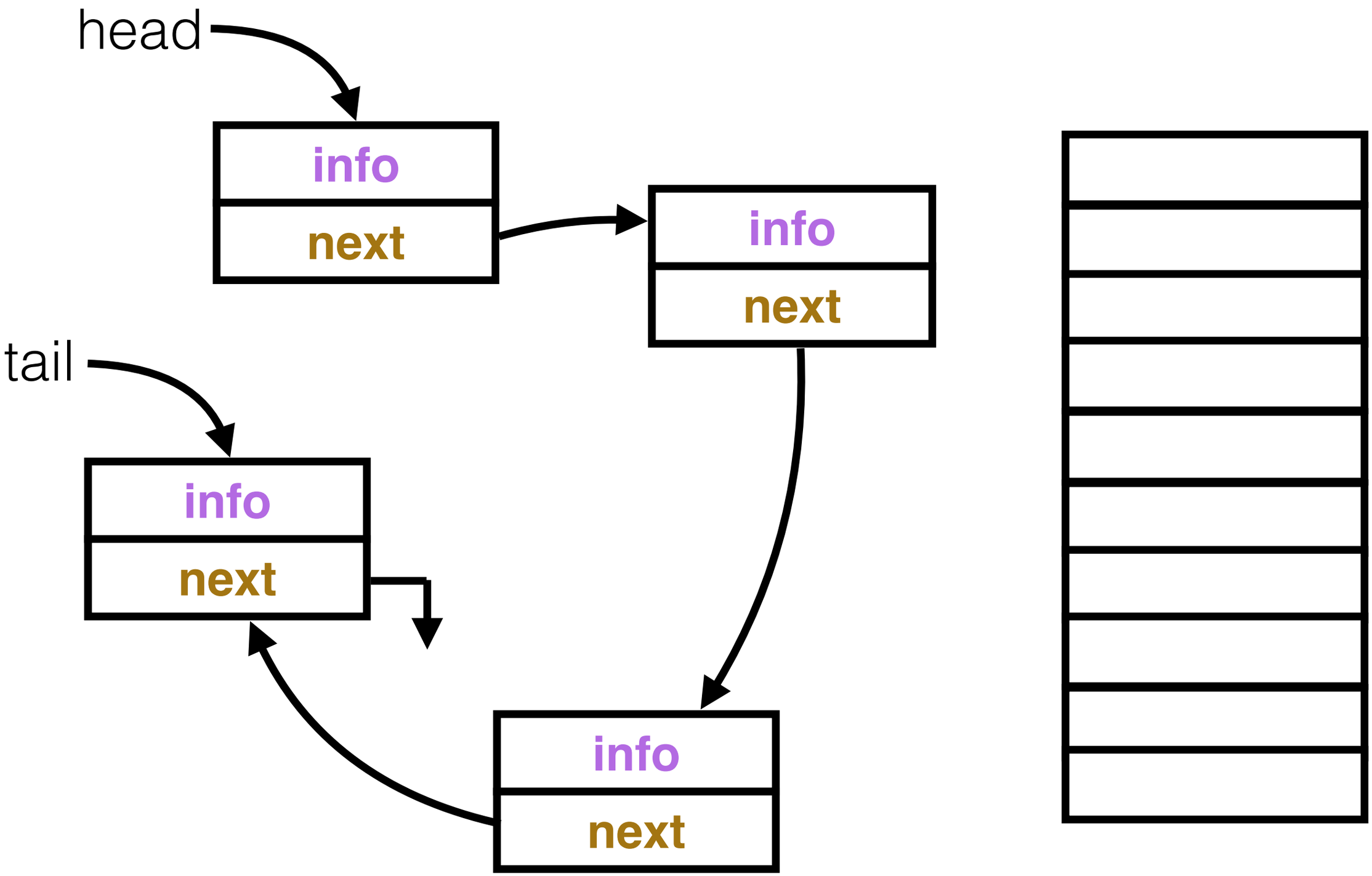
```
import java.util.Iterator;
import java.util.Vector;

public class Vector2 {
   static public void main( String[] args ) {
       Vector<Integer> list = new Vector<Integer>( );

       for ( int i=0; i<10; i++ )
           list.add( i * 3 );

       try {
           System.out.println( "Element at 4 = " + list.get( 4 ) );
       } catch (ArrayIndexOutOfBoundsException e ) {
           // do nothing
       }

      Iterator<Integer> it = list.iterator();
       while ( it.hasNext() ) {
           int x = it.next();
           System.out.println( x );
       }
   }
}
```

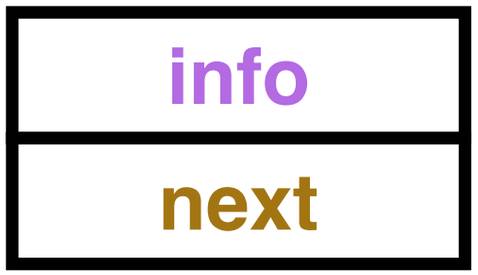
```
import java.util.Iterator;
import java.util.Vector;

public class Vector1 {
    static public void main( String[] args ) {
        Vector list = new Vector( );

        for ( int i=0; i<10; i++ )
            list.add( (Integer) (i * 3) );

        try {
            System.out.println( "Element at 4 = " + list.get( 4 ) );
        } catch (ArrayIndexOutOfBoundsException e ) {
            // do nothing
        }

        Iterator<Integer> it = list.iterator();
        while ( it.hasNext() ) {
            int x = it.next();
            System.out.println( x );
        }
    }
}
```

LINKED-LISTS

(Chapter 3 in Drozdek)


head
info
next
info
next
tail
info
next
info
next


head
info
next
info
next
tail
info
next
info
next




```
class intSSLNode {

    public int info;
    public intSSLNode next;

    public intSSLNode( int i ) {
        this( i, null );
    }
    public intSSLNode( int i, intSSLNode n ) {
        info = i;
        next = n;
    }
}
```

Questions:

- How do we implement a Linked List in Java, once we have intSSLNode?
- What is an empty list?
- How do we mark the end of a list?

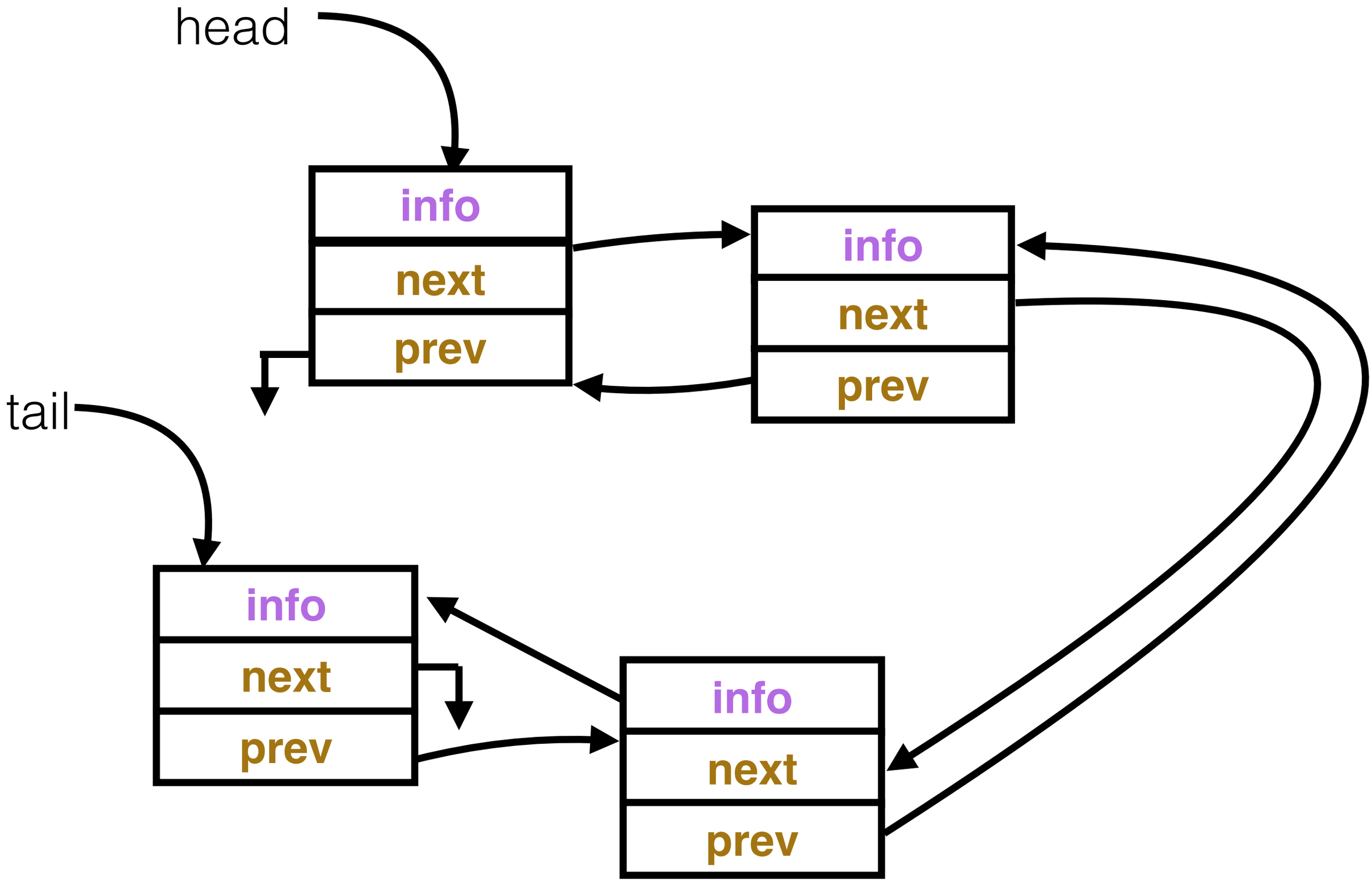
- How do we add an element at the front?
- How do we add an element at the end (tail)?
- How do we remove an element from the front? From the end?

```
public class IntSSList {
    protected IntSLLNode head, tail;
    public IntSSList () {
        head = tail = null;
    }
    public boolean isEmpty() {
        return (head == null);
    }
    public void addToHead( int x ) {
        head = new IntSSLNode( x, head );
        if ( tail==null )
            tail = head;
    }
    public int deleteFromHead() {
        int el = head.info();
        if ( head==tail )
            head = tail = null;
        else head = head.next;
        return el;
    }
    // continues on Page 84 in Drozdek
}
```

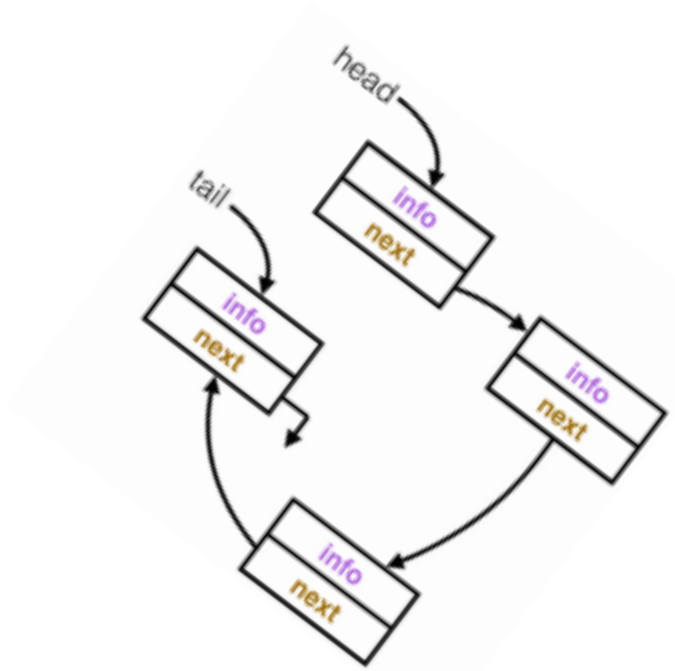
Evaluating the Efficiency of Linked Lists:

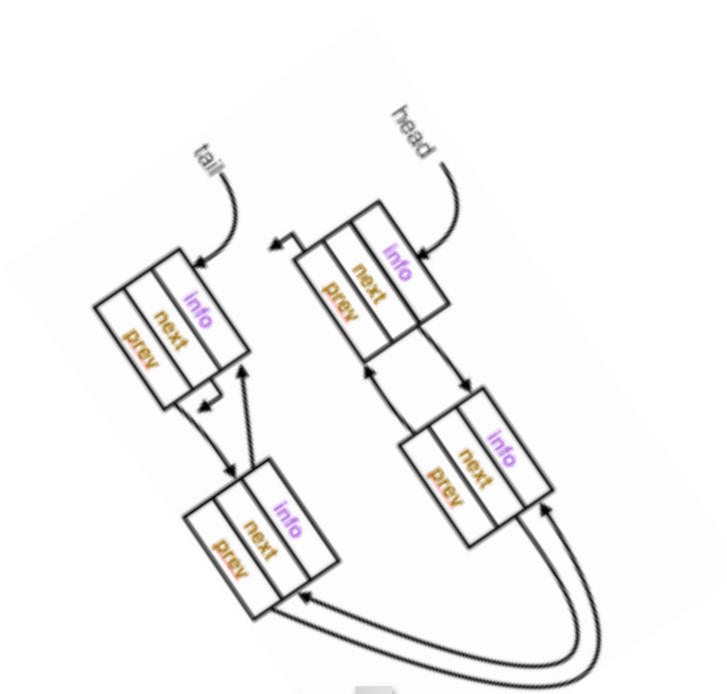
- Adding item at **tail**
- Adding item in **front**
- Adding item in **middle**
- **Iterate** over all
- Look at **first** item in list
- Look at **last** item in list

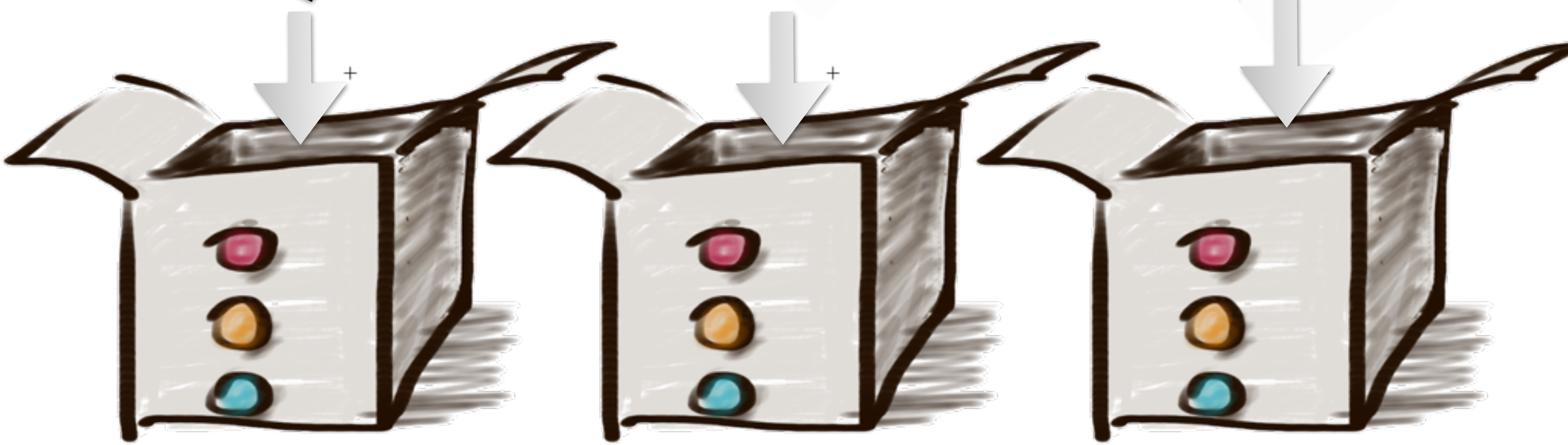
Doubly-Linked Lists

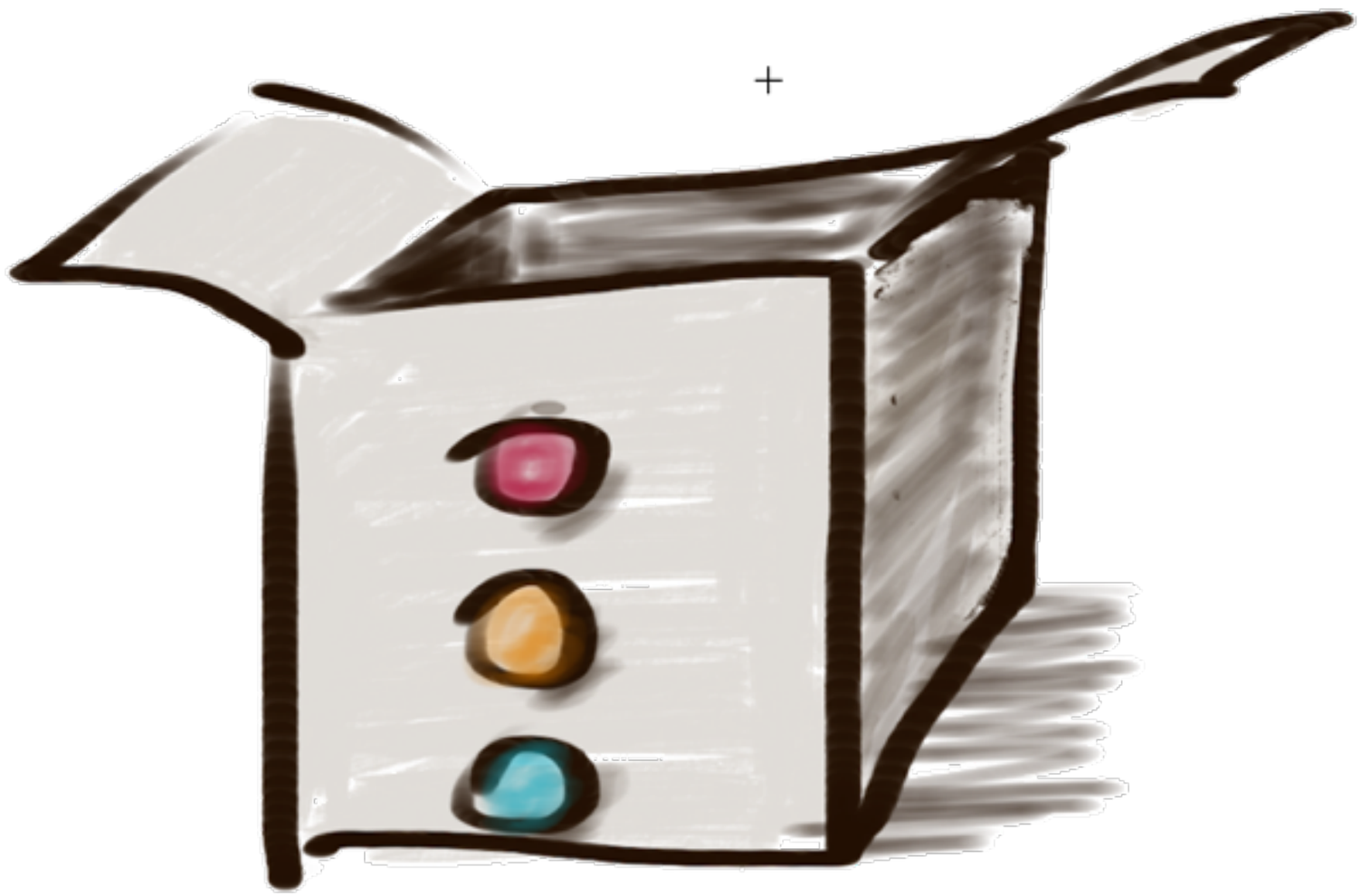

head
info
next
prev
info
next
prev
tail
info
next
prev
info
next
prev

Class Exercise


head
tail
info
next
info
next
info
next
info
next


tail
head
info
next
prev
info
next
prev
info
next
prev
info
next
prev





LIST

Modularization

OOP

Encapsulation