Lect 08-09

Projects 2: partially-filled array: building, traversing and editing
Project 3: singly-linked list: building and traversing.
Project 4: add editing to Project 3.

Reading List.
Live Coding with principles illustrated.
Reading List (Arrays)

• Book Ch. 5: Inserting & deleting done to a sequence (adt) implemented by a partially-filled array (data str.)
  
  • Albany supplement: sdc's notes on arrays
    • Gaddis&Mugada?
Reading List (Linked List)

- Basics: Mad PhD w/ neighbors; thief lab.
- Book 7.2 building [manually! :( ] & traversing of a sequence (adt) implemented by a linked list (data str.)
  (introduces editing)
- 7.3 editing continues

- LECTURE NOTES! and LIVE CODE examples!
• Book gets better at Ch. 8: scene graphs as really used in graphics SW. both 2d and 3d
When the going gets slow, invest a little time to make tedious activity go FASTER!!! (less distracting..)

FileChooser.pickMediaPath();

Put that in your program to make the FileChooser “remember” a folder. (It's just a G&E java-source thing.)
Please invent solid principles for how you use some of the variables!
class LinearCollage
{
    private Picture myArray[];
    //...
    // the length is available
    // with myArray.length.

    private int nPictures;
    //nPictures will ALWAYS
    // equal the nr. of images
    //the customer thinks is
    //CURRENTLY in his collage.
class LinearCollage {
    private int nPictures;
    // nPictures will ALWAYS equal the nr. of images
    // the customer thinks is CURRENTLY in his collage.

    You can absolutely trust this...and base your code on it....
    because..
    you make sure EVERY LINE OF CODE keeps it absolutely TRUE!
Changing nPictures

• starts at 0
• is NEVER CHANGED except:
  add 1 when a Picture is added or pasted.
  subtract 1 when a Picture is cut.
Using \text{nPictures}:

- It indexes next available element of \text{myArray}.
- Tell if pasting is impossible because the array is filled to capacity: \text{myArray.length} == \text{nPictures}.
- Tell if cutting is impossible because the array is empty: \text{nPictures} == 0.
- Calculate middle position: \text{nPictures} / 2 \text{ roughly}.
Linked List
One of the many (excessively many!!) things Java classes are good for is breaking software into modules.
The job of my LinearCollage object is to build & edit one linear collage.

How?

That's the private business of the LinearCollage class.
class LinearCollage
    //Linked List Version!
{
    private class Node
    {
        Picture data;
        Node pnext;
    }
    //Node called, in Java, //an “inner class”
class LinearCollage {

    private class Node {
        Picture data;
        Node pnext;
    }

    private int nPictures;
    private Node pFirst;
    private Node pLast;
}
class LinearCollage {

  private class Node {
    Picture data;
    Node pnext;
  }

  private int nPictures;
  private Node pFirst;
  private Node pLast;

LinearCollage Linked List Version

Data Structure Diagram - Generic State

Purpose:
- pLast -- helps QUICKLY add a new element at the end.
- pFirst -- necessary for getting at all the Pictures.
- nPictures -- helps QUICKLY calculate middle pos'n.
LinearCollage Linked List Version
Data Structure Diagram - Initial State

myCollage

LinearCollage

nrPictures 0
pFirst null
pLast null

Purposes:

pLast -- helps QUICKLY add a new element at the end.
pFirst -- necessary for getting at all the Pictures.
nPictures -- helps QUICKLY calculate middle pos'n
LinearCollage Linked List Version

Data Structure Diagram - One Picture State

myCollage

LinearCollage

nrPictures 1

pFirst

pLast

Picture 1

tone

Node

data

pNext null

Purposes:
pLast--helps QUICKLY add a new element at the end.
pFirst--necessary for getting at all the Pictures.
nPictures--helps QUICKLY calculate middle pos'n
(there ain't any) Dumb Questions

Doesn't Node need accessors and mutators? NO—not by Java language rules.

– Shouldn't Node have them? Info. hiding? Node (name and class def) is ALREADY HIDDEN!

How LinearCollage implements its sequenc is private business of LinearCollage. Let it define a cool private class for linked list list nodes, COMPLETELY INTERNAL to LinearCollage. Nobody can see or touch Nodes because they're private insides of LinearCollage.
Do you really want accessors & mutators?? Chaiken thinks they're a waste..

class LinearCollage {

    private class Node {
        private Picture data;
        private Node pnext;
        public Node(Picture p)
            { data=p; pnext=null; }

        /*acc*/public Node getNext()
            { return pnext; }

        /*mut*/public void putNext(Node n)
            { pnext=n; }

    }
}
class LinearCollage {

    private class Node
    { Picture data; Node pnext; }
    private nPictures;
    private Node pFirst;
    private Node pLast;

    public LinearCollage()
    {
        nPictures = 0;
        pFirst = null;
        pLast = null;
    }
}
LinearCollage Linked List Version
Data Structure Diagram - Initial State

Purposes:
pLast--helps QUICKLY add a new element at the end.
pFirst--necessary for getting at all the Pictures.
nPictures--helps QUICKLY calculate middle pos'n
class LinearCollage {

    private class Node
    {
        Picture data; Node pnext;
    }

    private nPictures;
    private Node pFirst;
    private Node pLast;

    public LinearCollage() { ... }

    public void addPAtEnd(Picture p)
    {
        Node tmp = new Node();
        tmp.pnext = null;
        tmp.data = p;
    }
3 MORE THINGS TO DO.

DONE!

Purposes:
- pLast--helps QUICKLY add a new element at the end.
- pFirst--necessary for getting at all the Pictures
- nPictures--helps QUICKLY calculate middle position
class LinearCollage {
    private class Node
    {
        Picture data; Node pnext;
    }
    private nPictures;
    private Node pFirst;
    private Node pLast;
    public LinearCollage() { ... }
    public void addPAtEnd(Picture p)
    {
        Node tmp = new Node();
        tmp.pnext = null;
        tmp.data = p;
        if( pFirst == null )
        {
            pFirst = tmp;
            pLast = tmp;
        }
        else
        {
        }
        nPictures++;
    }
LinearCollage  Linked List Version
Data Structure Diagram - One Picture State

myCollage

LinearCollage

nrPictures 1
pFirst
pLast

Picture

Node

data

pFirst

pLast

null

Purposes:
pLast--helps QUICKLY add a new element at the end.
pFirst--necessary for getting at all the Pictures.
nPictures--helps QUICKLY calculate middle pos'n
class LinearCollage {
    private class Node {
        Picture data; Node pnext;
    }
    private nPictures;
    private Node pFirst;
    private Node pLast;
    public LinearCollage() { ... }
    public void addPAtEnd(Picture p) {
        Node tmp = new Node();
        tmp.pnext = null;
        tmp.data = p;
        if( pFirst == null )
        { pFirst = tmp;
          pLast = tmp;
        }
        else { /* ??? */ }
        nPictures++;
    }
}
LinearCollage Linked List Version
Data Structure Diagram - One Picture State

Purpose:
PLast -- helps QUICKLY add a new element at the end.
PFirst -- necessary for getting at all the Pictures.
nPPictures -- helps QUICKLY calculate middle pos'n
LinearCollage Linked List Version

Data Structure Diagram - Generic State
class LinearCollage {
    private class Node {
        { Picture data; Node pnext; }
    }

    private nPictures;
    private Node pFirst;
    private Node pLast;

    public LinearCollage() { ... }

    public void addPAtEnd(Picture p) {
        Node tmp = new Node();
        tmp.pnext = null;
        tmp.data = p;
        if (pFirst == null) {
            pFirst = tmp;
            pLast = tmp;
        } else {
        }/* ??? */
        nPictures++;
    }
}
Lecture 09

- Adding the FIRST Node.
- Adding the SECOND, THIRD, ... and other Nodes.
Purposes:

pLast---helps QUICKLY add a new element at the end.
pFirst---necessary for getting at all the Pictures.
nPPictures---helps QUICKLY calculate middle pos'n
LinearCollage Linked List Version

Data Structure Diagram - One Picture State

**myCollage**
- **nrPictures**
  - 1
- **pFirst**
- **pLast**

**Picture 1**

**Node**
- **data**
- **pNext**
  - **null**

**Purposes:**

pLast -- helps QUICKLY add a new element at the end.
pFirst -- necessary for getting at all the Pictures.
nPictures -- helps QUICKLY calculate middle pos'n
class LinearCollage {

    private class Node
    {
        Picture data; Node pnext;
    }

    private nPictures;
    private Node pFirst;
    private Node pLast;
    public LinearCollage() { ... }

    public void addPAAtEnd(Picture p)
    {
        Node tmp = new Node();
        tmp.pnext = null;
        tmp.data = p;
    }
What was changed?
(A) LinearCollage
(B) Node (and tmp)
(C) Both
(D) Neither

```java
Node tmp = new Node();
tmp.pNext = null;
tmp.data = p;
```
Node (and tmp) changed!
Let us move on ... write code to change the LinearCollage object

```java
Node tmp = new Node();
tmp.pNext = null;
tmp.data = p;
```
Purposes:
pLast--helps QUICKLY add a new element at the end.
pFirst--necessary for getting at all the Pictures. nPictures--helps QUICKLY calculate middle pos'n
Purposes:
pLast--helps QUICKLY add a new element at the end.
pFirst--necessary for getting at all the Pictures.
nPictures--helps QUICKLY calculate middle pos'n
if (pFirst == null)
{
    pFirst = tmp;
    pLast = tmp;
} else {
}

nPictures++;

Purposes:
pLast--helps QUICKLY add
    a new element at the end.
pFirst--necessary for getting at all the Pictures.

nPictures--helps QUICKLY calculate middle posIn
The First Node has been Added.
class LinearCollage {
    private class Node
        { Picture data; Node pnext; }
    private nPictures;
    private Node pFirst;
    private Node pLast;
    public LinearCollage() { ... } 
    public void addPATEnd(Picture p)
    {
        Node tmp = new Node();
        tmp.pnext = null;
        tmp.data = p;
        if( pFirst == null )
            { pFirst = tmp;
            pLast = tmp; }
        else { /* Let's figure out how to add 2nd, 3rd, and other Nodes?? */ } 
        nPictures++;
    }
}
Just before we add Node 2---to the end of the list.
Just before Node 4 is added.
iClicker Quiz
Look on the paper,
ONLY LOOK AT
the
Diagram
NOT THE CODE (B)!
in {/* ??? */} which??

(A) { pLast.pNext = tmp;
    pLast = tmp; }
    nPictures++;

(B) { pLast = tmp;
    pLast.pNext = tmp; }
    nPictures++;

(C) Both A and B are OK code.
Paper Quiz:  Student's program (B) has a bug and he can't figure it out.

MARK the DIAGRAM to show EXACTLY what's in the computer AFTER the buggy code has been run.
(B) { pLast = tmp;
    pLast.pNext = tmp; }

nrPictures++;