Introduction To Data Structures

This section introduces the concept of a data structure as well as providing the details of a specific example: a list.

James Tam

Tip For Success: Reminder

- Look through the examples and notes before class.
- This is especially important for this section because the execution of these programs will not be in sequential order.
- Instead execution will appear to 'jump around' so it will be harder to understand the concepts and follow the examples illustrating those concepts if you don't do a little preparatory work.

James Tam

Location Of The Online Examples

- Course website:
  - www.cpsc.ucalgary.ca/~tamj/233/examples/lists
- UNIX directory:
  - /home/233/examples/lists

James Tam

What Is A Data Structure

- A composite type that has a set of basic operations (e.g., display elements of a list) that may be performed on instances of that type.
  - It can be accessed as a whole (e.g., pass the entire list as a parameter to a function).
  - Individual elements can also be accessed (e.g., update the value for a single list element).
- The type may be a built-in part of the programming language
  - e.g., lists are included with the Python language and need not be defined before they can be used
- The type may also be defined by the programmer inside a program (for languages which don't include this composite type)

```
class List
{
    :   :
}
```

James Tam

What Is A Data Structure (2)

- In some cases the data structure may only be partially implemented as part of the language, some operations must be manually written by the programmer.
- Example: The ability to add an element to a list has been implemented as a pre-created Python function.

```
aGrid = []              # Creates an empty list
aGrid.append (12)       # Adds a number to the end of the list
```

- In a language such as 'C' a list is implemented as an array but the operation to add elements to the end of the list must be written by the programmer.
- Lesson: when choosing a programming language look for built-in support for key features.

James Tam

Lists

- Lists are a type of data structure (one of the simplest and most commonly used).
  - e.g., grades for a lecture can be stored in the form of a list
- List operations: creation, adding new elements, searching for elements, removing existing elements, modifying elements, displaying elements, sorting elements, deleting the entire list).
- Basic Java implementation of lists: array, linked list.

James Tam

Arrays

- An array of 'n' elements will have an index of zero for the first element up to index (n-1) for the last element.
- The array index is an integer and indicates which element to access (excluding the index and just providing the name of the list means that the program is operating on the entire list).
- Similar to objects, arrays employ dynamic memory allocation (the name of the array is actually a reference to the array).
- Many utility methods exist.
- Several error checking mechanisms are available.

James Tam

Arrays

- An array of 'n' elements will have an index of zero for the first element up to index (n-1) for the last element.
- The array index is an integer and indicates which element to access (excluding the index and just providing the name of the list means that the program is operating on the entire list).
- **Similar to objects, arrays employ dynamic memory allocation (the name of the array is actually a reference to the array).**
- Many utility methods exist.
- Several error checking mechanisms are available.

James Tam

Declaring Arrays

- Arrays in Java actually use a reference to the array so creating an array requires two steps:
  1) Declaring a reference to the array
  2) Allocating the memory for the array

James Tam

Declaring A Reference To An Array

•**Format:**

// The number of pairs of square brackets specifies the number of
// dimensions.
<*type*> [] <array *name*>;

•**Example:**

int [] arr;
int [][] arr;

James Tam

Allocating Memory For An Array

- **Format:**
  <*array name*> = new <*array type*> [<*no elements*>];

- **Example:**
  arr = new int [SIZE];
  arr = new int [ROW SIZE][COLUMN SIZE];

  (Both steps can be combined together):
  int [] arr = new int[SIZE];

James Tam

Arrays: An Example

- The name of the online example is can be found in the directory: simpleArrayExample

```
public class Driver
{
   public static void main (String [] args)
   {
      int i;
      int len;
      int [] arr;
```

James Tam

Arrays: An Example

```
        Scanner in = new Scanner (System.in);
        System.out.print("Enter the number of array elements: ");
        len = in.nextInt ();
        arr = new int [len];
        System.out.println("Array Arr has " + arr.length + " elements.");
        for (i = 0; i < arr.length; i++)
        {
            arr[i] = i;
            System.out.println("Element[" + i + "]=" + arr[i]);
        }
    }
}
```

James Tam

Arrays

- An array of 'n' elements will have an index of zero for the first element up to index (n-1) for the last element.
- The array index is an integer and indicates which element to access (excluding the index and just providing the name of the list means that the program is operating on the entire list).
- Similar to objects, arrays employ dynamic memory allocation (the name of the array is actually a reference to the array).
- Many utility methods exist.
- **Several error checking mechanisms are available.**
  - Null array references
  - Array bounds checking

James Tam

Using A Null Reference

```
int [] arr = null;
arr[0] = 1;
```

NullPointerException

James Tam

Exceeding The Array Bounds

```
int [] arr  = new int [4];
int i;
for (i = 0; i <= 4; i++)
  arr[i] = i;
```

ArrayIndexOutOfBoundsException

(when i = 4)

James Tam

Arrays Of Objects (References)

•Example:

```
public class Foo
{
   private int num;
   public void setNum (int aNum) { num = aNum; }s
}
```

•An array of objects is actually an array of references to objects

e.g., Foo [] arr = new Foo [4];

•The elements are initialized to null by default

arr[0].setNum(1); ← **NullPointerException**

James Tam

Arrays Of Objects (References)

•Since each list element is a reference (and references are set to null by default in Java), before elements can be accessed an object must be created for each element.

•For single references:

- Foo f; // No object exists yet
- f = new Foo (); // Creates an object and the reference 'f' refers to it.

•For arrays of references

```
Foo [] arr = new Foo [4];   // Creates array of references (each reference is
  currently null)
int i;
for (i = 0; I < 4; i++)
    arr[i] = new Foo();      // Each element will refer to a Foo object each time
                             // through the loop.
```

James Tam

A More Complex List Example

- This example will track a book collection.
- It will be implemented as an array and as a linked list.
- List operations implemented:
  - Creation of the list
  - Erasure of the entire list
  - Display of the list (iterative and recursive implementation)
  - Adding new elements
  - Removing elements
- There will two example implementations: array, linked list

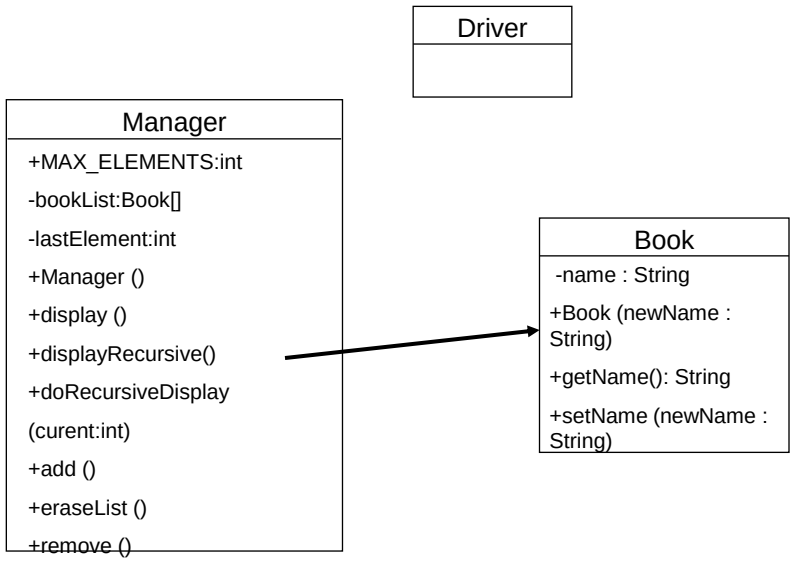
James Tam

List: Array Implementation

- The online example can be found in the directory: array
- Classes
  - Book: tracks all the information associated with a particular book
  - Manager: implements all the list operations
  - Driver: starting execution point, calls methods of the Manager class in order to change the list.

James Tam

Array Example: UML Diagram



James Tam

Class Book

```
public class Book
{
   private String name;

   public Book (String aName) { setName(aName); }

   public void setName (String aName) { name = aName; }

   public String getName ()  { return name; }
}
```

James Tam

Class Manager

```
public class Manager
{
   public final int MAX_ELEMENTS = 10;
   private Book [] bookList;
   private int lastElement;

   public Manager ()
   {
         // Code to be described later
    }
```

James Tam

Class Manager (2)

```
   public void display()
   {
        // Code to be described later
   }

   public void displayRecursive ()
   {
        // Code to be described later
   }

   private void doRecursiveDisplay (int current)
   {
        // Code to be described later
   }
```

James Tam

Class Manager (3)

```
    public void add ()
    {
        // Code to be described later
    }

    public void eraseList ()
    {
        // Code to be described later
    }

    public void remove ()
    {
        // Code to be described later
    }
}
```

James Tam

Driver Class

```
public class Driver
{
    public static void main (String [] args)
    {
        Manager aManager = new Manager();

        // Display: Empty listt
        System.out.println("Part I: display empty list");
        aManager.display();
        System.out.println();

       // Destroy list
       System.out.println("Part II: erasing the entire list and displaying the empty
                                list");
       aManager.eraseList();
       aManager.display();
       etc.
}
```

James Tam

List Operations: Arrays (Display)

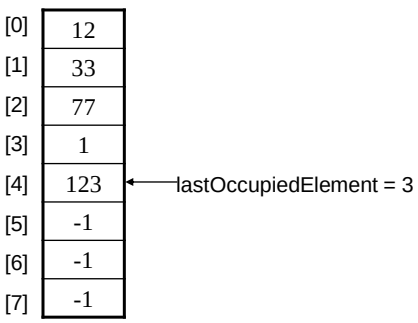
- Unless it can be guaranteed that the list will always be full (unlikely) then some mechanism for determining that the end of the list has been reached is needed.
- If list elements cannot take on certain values then unoccupied list elements can be 'marked' with an invalid value.
- Example: grades (simple array elements)

[0]	100
[1]	75
[2]	65
[3]	0
[4]	80
[5]	-1
[6]	-1
[7]	-1

James Tam

List Operations: Arrays (Display: 2)

- If list elements can't be marked then a special variable ("last" index) can be used to mark the last occupied element (works with an array of simple types or an array of more complex types like objects).



James Tam

List Operations: Arrays (Creation)

•Simply declare an array variable

array name> = new <*array type*> [<*no elements*>];

•Constructor

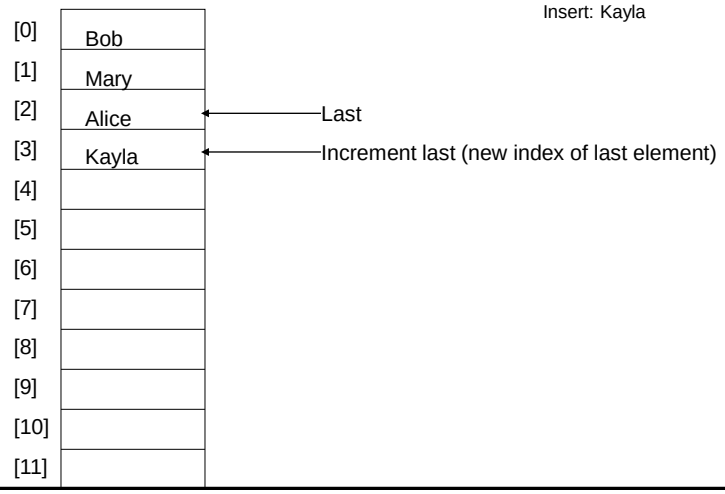
```
// Call in the Driver
Manager aManager = new Manager();

// In the Manager class
public Manager ()
{
    bookList = new Book[MAX_ELEMENTS];
    int i;
    for (i = 0; i < MAX_ELEMENTS; i++)
        bookList[i] = null;

    lastElement = -1;
}
```

James Tam

List Operations: Arrays (Insertion At End)

•Insertion at the end.

- Some mechanism is needed to either find or keep track of the last occupied element.



James Tam

List Operations: Arrays (Insertion At End: 2)

•Driver class

```
aManager.add();
aManager.add();
```

•Manager class

```
public void add ()
{
    String newName;
    Scanner in;
```

James Tam

List Operations: Arrays (Insertion At End: 3)

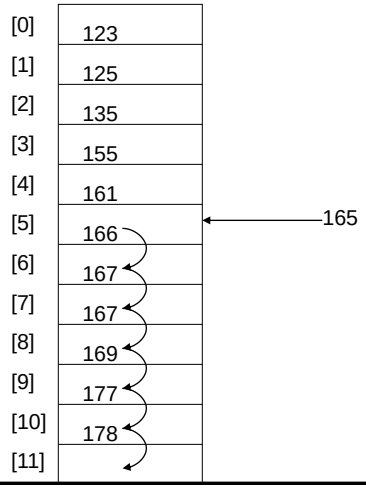
```
 if ((lastElement+1) < MAX_ELEMENTS)
 {
    System.out.print("Enter a title for the book: ");
    in = new Scanner (System.in);
    newName = in.nextLine ();
    lastElement++;
    bookList[lastElement] = new Book(newName);
 }

 else
 {
    System.out.print("Cannot add new element: ");
    System.out.println("List already has " + MAX_ELEMENTS + " elements.");
 }
}  // End of add
```

James Tam

List Operations: Arrays (In Order Insertion)

•In order insertion.

- Some mechanism is needed to find the insertion point (search).
- Elements may need to be shifted.



James Tam

List Operations: Display List

•Driver Class

```
aManager.display();
```

•Manager Class

```
public void display()
{
    int i;
    System.out.println("Displaying list");
    if (lastElement <= -1)
        System.out.println("\tList is empty");
    for (i = 0; i <= lastElement; i++)
    {
        System.out.println("\tTitle No. " + (i+1) + ": "+ bookList[i].getName());
    }
}
```

James Tam

List Operations: Alternative Display List

•Driver class

```
aManager.displayRecursive();
```

•Manager class

```
public void displayRecursive ()
{
    if (lastElement <= -1)
    {
        System.out.println("\tList is empty");
    }
    else
    {
        final int FIRST = 0;
        System.out.println("Displaying list");
        doRecursiveDisplay(FIRST);
    }
}
```

James Tam

List Operations: Alternative Display List (2)

```
private void doRecursiveDisplay (int current)
{
    if (current <= lastElement)
    {
        System.out.println("\tTitle No. " + (current+1) + ": "+
        bookList[current].getName());
        current++;
        doRecursiveDisplay(current);
    }
}
```

James Tam

List Operations: Erasure Of Entire List

•Driver Class

```
aManager.eraseList();
```

•Manager Class

```
public void eraseList ()
{
    // Assignment below not needed, nor is there any need in Java
    // to manually delete each element.
    // bookList = null;

    lastElement = -1;
}
```

James Tam

List Operations: Arrays (More On Destroying The Entire List)

- Recall that Java employs automatic garbage collection.
- Setting the reference to the array to null will eventually allow the array to be garbage collected.

  <*array name*> = null;
- Note: many languages do not employ automatic garbage collection and in those cases, either the entire array or each element must be manually de-allocated in memory.

James Tam

Memory Leak

- A technical term for programs that don't free up dynamically allocated memory.
- It can be serious problem because it may result in a drastic slowdown of a program.

James Tam

List Operations: Arrays (Removing Last Element)

- Driver:

```
aManager.remove();
```

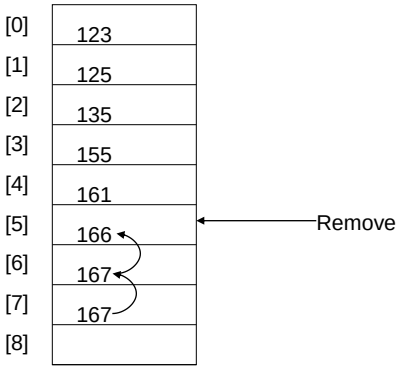
- Manager:

```
public void remove ()
{
    if (lastElement > -1)
    {
        lastElement--;
        System.out.println("Last element removed from list.");
    }
    else
        System.out.println("List is already empty: Nothing to remove");
}
```

James Tam

List Operations: Arrays (Search & Removing Elements)

- A search is needed to find the removal point.
- Depending upon the index of the element to be deleted, other elements may need to be shifted.



James Tam

Lists: Array Implementation (Summary)

- Advantage:
  - Arrays are simple and easy to use
  - The array implementation of a list may be completed faster
- Disadvantage:
  - Unless the programming language has arrays that automatically resize (grow and shrink as needed) then using an array is often wasteful.
    - The number of elements created is often more than what's needed
  - Insertions and deletions of elements may be slow and inefficient: first element added/removed many shifts may be required, there are many elements that must be shifted, each element requires a great deal of resources to stores.

James Tam

Linked Lists

- An alternate implementation of a list.
  - As the name implies, unlike an array the linked list has explicit connections between elements
  - This connection is the only thing that holds the list together.
    - Removing a connection to an element makes the element inaccessible.
    - Adding a connection to an element makes the element a part of the list.
- The program code is somewhat more complex but some operations are more efficient (e.g., additions and deletions don't require shifting of elements).
  - Just change some connections.
- Also linked lists tend to be more memory efficient that arrays.
  - The typical approach with an array implementation is to make the array larger than needed. (Unused elements are allocated in memory and the space is wasted).
  - With a linked list implementation, elements only take up space in memory as they're needed.

James Tam

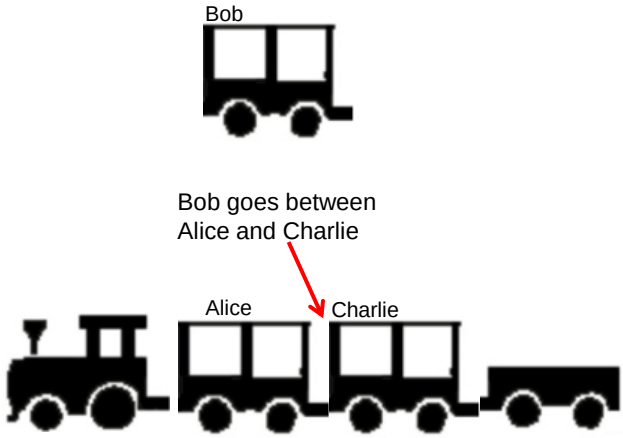
Linked Lists

- Insertions and removal of elements can be faster and more efficient because no shifting is required.
- Elements need only be linked into the proper place (insertions) or bypassed (deletions)

James Tam

Insertion

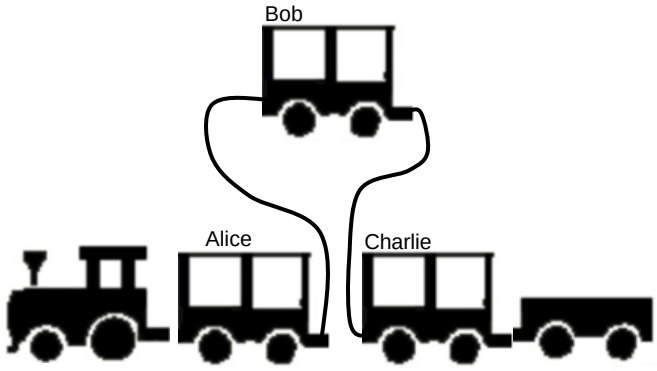
•Find the insertion point



James Tam

Insertion (2)

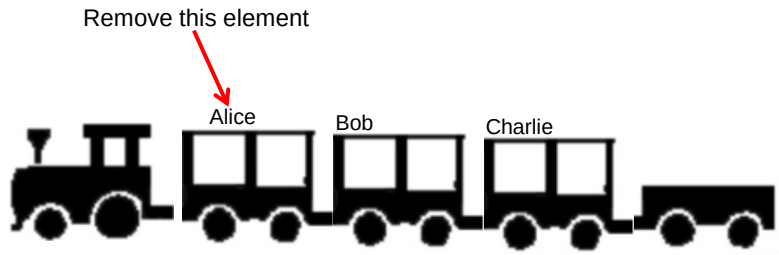
•Change the connections between list elements so the new element is inserted at the appropriate place in the list.



James Tam

Deletions

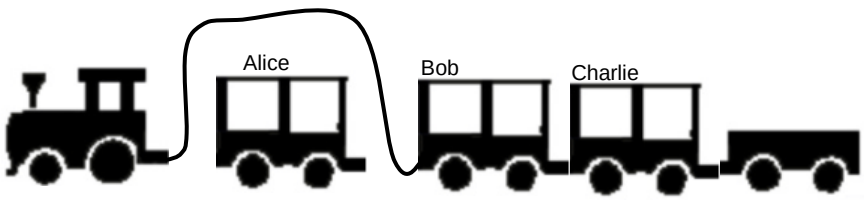
•Find location of the element to be deleted



James Tam

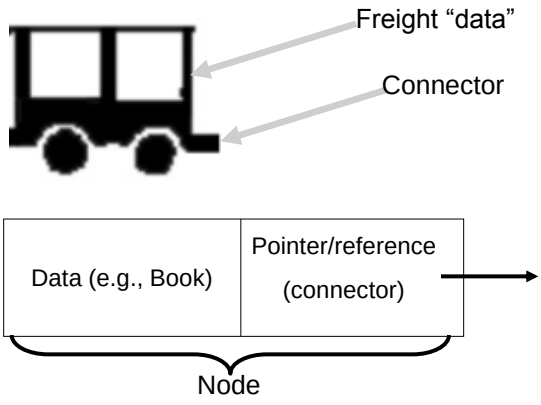
Deletions (2)

•Change the connections so that the element to be deleted is no longer a part of the list (by passed).



James Tam

List Elements: Nodes



James Tam

Example: Defining A Node

```
public class BookNode
{
   private Book data;
   private BookNode next;
      :          :          :
}
```

Information stored by each element

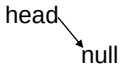
Connects list elements

James Tam

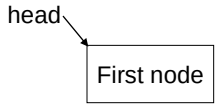
Example: Marking The Start Of The List

```
public class Manager
{
   private BookNode head;
}
```

Case 1:
Empty list



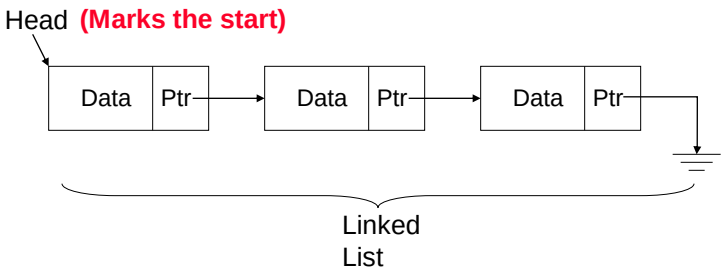
Case 2: Non-empty list



James Tam

Linked Lists: Important Details

- Unlike arrays, many details must be manually and explicitly specified by the programmer: start of the list, connections between elements, end of the list.



- Caution! Take care to ensure the reference to the first element is never lost.

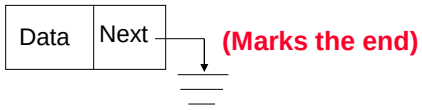
1 The approximate equivalent of a pointer ("ptr") in Java is a reference.

James Tam

More On Connections: The Next Pointer

- Because linked lists only create elements as needed a special marker is needed for the end of the list.
- The 'next' attribute of a node will either:
  - Contain a reference/address of the next node in the list.

  - Contain a null value.



- (That means there is a reference to the start of the list, the next pointer of each element can be used to traverse the list).

James Tam

List: Linked List Implementation

- The online example can be found in the directory:
  - /home/233/examples/lists/linked
- Classes

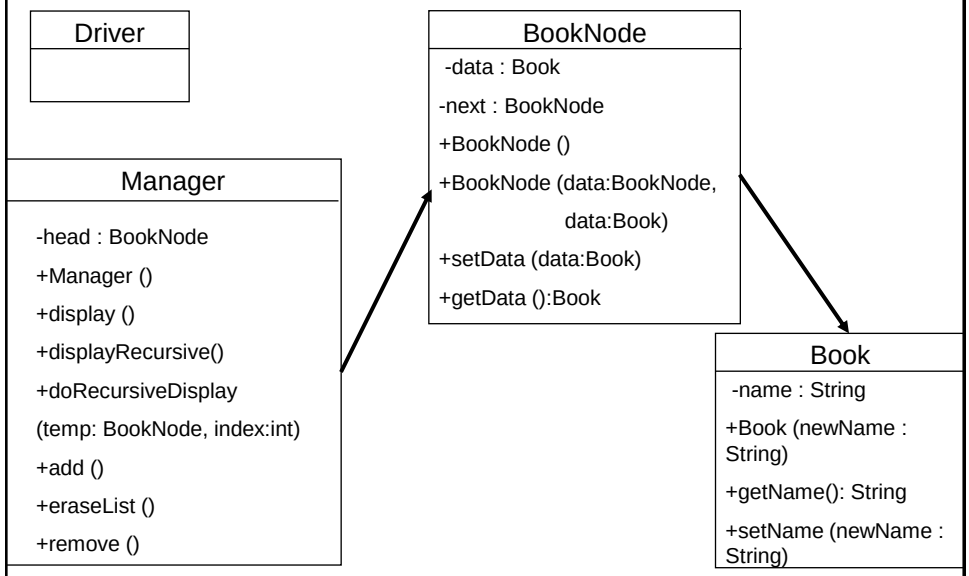
  (These classes are the same as the array version)
  - Book: tracks all the information associated with a particular book
  - Driver: starting execution point, calls methods of the Manager class in order to change the list.

  - (This class contains the same methods as the array version)
  - Manager: implements all the same list operations (the implementation differs from the array version)

  - (New class)
  - BookNode: needed so that list elements can be linked. Data for node is a book, but also it needs a 'next' attribute to link the other nodes in the list – it's these inter node connections that links all the elements of the list.

James Tam

Linked List Example: UML Diagram



James Tam

Class Book (Same As Array Implementation)

```
public class Book
{
   private String name;

   public Book (String aName) { setName(aName); }

   public void setName (String aName) { name = aName; }

   public String getName ()  { return name; }
}
```

James Tam

Class Driver (Same As The Array Implementation)

```
public class Driver
{
   public static void main (String [] args)
   {
       Manager aManager = new Manager();

       // Display: Empty list
      System.out.println("Part I: display empty list");
      aManager.display();

       // Destroy list
      System.out.println("Part II: erasing the entire list and displaying the empty
                                 list");
      aManager.eraseList();
      aManager.display();
      System.out.println();

     Etc.
```

James Tam

Class Manager

```
public class Manager
{
   private BookNode head;
   public Manager ()
   {
          // Code to be described later
    }
```

James Tam

Class Manager (2)

```
public void display()
{
      // Code to be described later
}

public void displayRecursive ()
{
      // Code to be described later
}

private void doRecursiveDisplay (int current)
{
      // Code to be described later
}
```

James Tam

Class Manager (3)

```
public void add ()
{
    // Code to be described later
}

public void eraseList ()
{
    // Code to be described later
}

public void remove ()
{
    // Code to be described later
}
}
```

James Tam

Class BookNode (New: Only In Linked List Version)

```
public class BookNode
{
   private Book data;
   private BookNode next;

   public BookNode ()
   {
        data = null;
        next = null;
   }

   public BookNode (Book data, BookNode next)
   {
        setData(data);
        setNext(next);
   }
```

James Tam

Class BookNode (New: Only In Linked List Version: 2)

```
   public void setData (Book data)  {   this.data = data;  }

   public Book getData ()  {  return data;  }

   public void setNext (BookNode next)  {   this.next = next;  }

   public BookNode getNext ()  {  return next;   }
}
```

James Tam

List Operations: Linked Lists (Creation)

- After a type for the list has been declared then creating a new list requires that an instance be created and initialized.
- Example:

```
BookNode head = null;
```

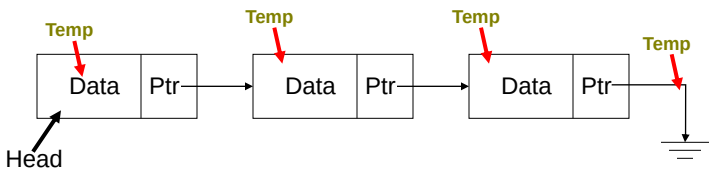
- Constructor (Manager):

```
public Manager ()
{
    head = null;
}
```

James Tam

List Operations: Linked Lists (Display)

- A temporary pointer/reference is used when successively displaying the elements of the list.
- When the temporary pointer is null, the end of the list has been reached.
- Graphical illustration of the algorithm:



- Pseudo code algorithm:

```
while (temp != null)
    display node
    temp = address of next node
```

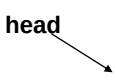
James Tam

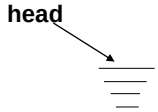
Display Method

```
public void display()
{
    int i = 1;
    BookNode temp = head;
    System.out.println("Displaying list");
    if (head == null)
       System.out.println("\tList is empty");
    while (temp != null)
    {
        System.out.println("\tTitle No. " + i + ": "+ temp.getData().getName());
        i = i + 1;
        temp = temp.getNext();
     }
}
```

James Tam

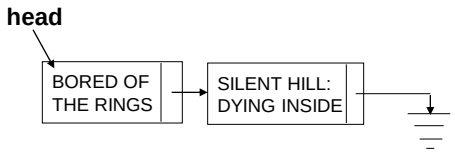
Displaying The List: Iterative Implementation (Loop)





James Tam

Displaying The List: Iterative Implementation (Loop: 2))



James Tam

Traversing The List: Display

- **Study guide:**
- Steps (traversing the list to *display* the data portion of each node onscreen)
  1. Start by initializing a temporary reference to the beginning of the list.
  2. If the reference is 'null' then display a message onscreen indicating that there are no nodes to display and stop otherwise proceed to next step.
  3. While the temporary reference is not null:
     a) Process the node (e.g., display the data onscreen).
     b) Move to the next node by following the current node's next reference (set the reference to refer to the next node).

James Tam

Displaying List: Recursive Implementation

•Driver class call:

```
aList.displayRecursive();
```

•Manager class:

```
public void displayRecursive()
{
  System.out.println("Displaying list");
   if (head == null)
        System.out.println("\tList is empty");
  else
  {
      final int FIRST = 0;
      doRecursiveDisplay (head,FIRST);
  }
}
```

James Tam

Displaying List: Recursive Implementation (2)

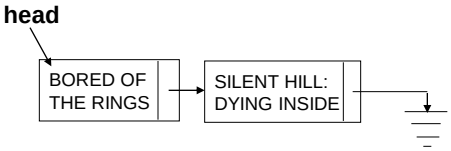
```
private void doRecursiveDisplay (BookNode temp, int index)
{
   if (temp == null)
       return;
   else
   {
      index++;
      System.out.println("\tTitle No. " + index + ": "+
                                 temp.getData().getName());
      temp = temp.getNext();
      doRecursiveDisplay(temp,index);
   }
}
```

James Tam

Displaying The List: Recursive Implementation

James Tam

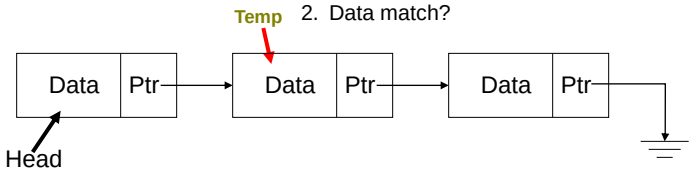
Displaying The List: Recursive Implementation (2)



James Tam

List Operations: Linked Lists (Search)

- The algorithm is similar to displaying list elements except that there must be an additional check to see if a match has occurred.
- Conditions that may stop the search:
  1. Temp = null (end)?
  2. Data match?



James Tam

List Operations: Linked Lists (Search: 2)

- Pseudo code algorithm:

```
Temp refers to beginning of the list
If (temp is referring to empty list)
    display error message "Empty list cannot be searched"
While (not end of list AND match not found)
    if (match found)
        stop search or do something with the match
    else
        temp refers to next element
```

James Tam

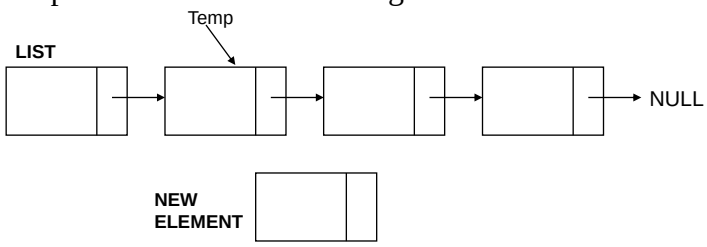
List Operations That Change List Membership

- These two operations (add/remove) change the number of elements in a list.
- The first step is to find the point in the list where the node is to be added or deleted (typically requires a search).
- Once the point in the list has been found, changing list membership is merely a reassignment of pointers/references.
  - Again: unlike the case with arrays, no shifting is needed.

James Tam

List Operations: Linked Lists (Insertion)

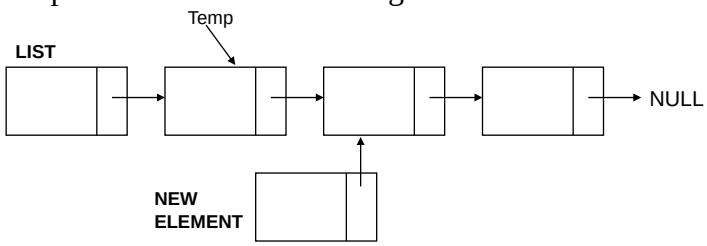
- Graphical illustration of the algorithm:



James Tam

List Operations: Linked Lists (Insertion: 2)

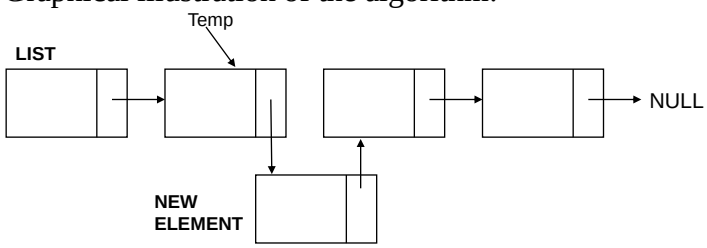
•Graphical illustration of the algorithm:



James Tam

List Operations: Linked Lists (Insertion: 3)

•Graphical illustration of the algorithm:



James Tam

List Operations: Linked Lists (Insertion: 4)

•Pseudo code algorithm:

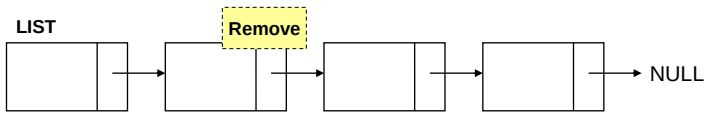
Node to be inserted refers to node after insertion point

Node at insertion point refers to the node to be inserted

James Tam

List Operations: Linked Lists (Removing Elements)

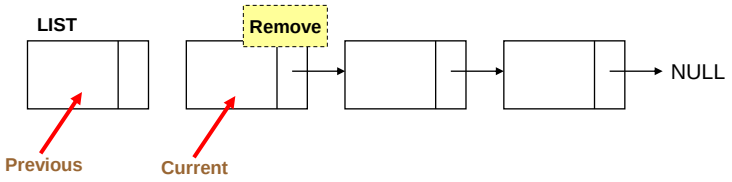
•Graphical illustration of the algorithm



James Tam

List Operations: Linked Lists (Removing Elements: 2)

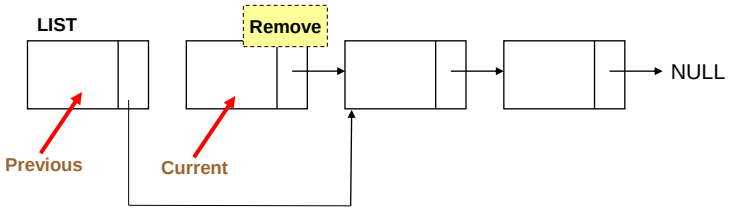
•Graphical illustration of the algorithm



James Tam

List Operations: Linked Lists (Removing Elements: 2)

•Graphical illustration of the algorithm



Node to be removed has been bypassed
(effectively deleted from the list)

James Tam

List Operations: Linked Lists (Removing Elements: 3)

•Pseudo code algorithm:

Previous node refers to the node referred by current node

James Tam

Removing A Node From The List (4)

```
public void remove ()
{
  // CASE 1: EMPTY LIST
  if (head == null)
      System.out.println("List is already empty: Nothing to remove");

  // CASE 2: NON-EMPTY LIST
  else
  {
     BookNode previous = null;
     BookNode current = head;
     String searchName = null;
     boolean isFound = false;
     String currentName;
     Scanner in = new Scanner(System.in);
     System.out.print("Enter name of book to remove: ");
     searchName = in.nextLine();
```

James Tam

Removing A Node From The List (5)

```
// Search for point of removal
 while ((current != null) &&
        (isFound == false))
 {
    currentName = current.getData().getName();
    if (searchName.compareToIgnoreCase(currentName) == 0)
      isFound = true;
    else
    {
       previous = current;
       current = current.getNext();
    }
 }
```

James Tam

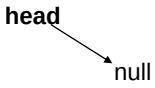
Removing A Node From The List (6)

```
// CASE 2A OR 2B: MATCH FOUND (REMOVE A NODE)
if (isFound == true)
{
  System.out.println("Removing book called " + searchName);
  // CASE 2A: REMOVE THE FIRST NODE
  if (previous == null)
    head = head.getNext();
  // CASE 2B: REMOVE ANY NODE EXCEPT FOR THE FIRST
  else
    previous.setNext(current.getNext());
}
// CASE 3: NO MATCHES FOUND (NOTHING TO REMOVE).
else
  System.out.println("No book called " + searchName + " in the
                      collection.");
    }
  }
}
```

James Tam

Removing A Node From The List (7)

•Case 1: Empty List



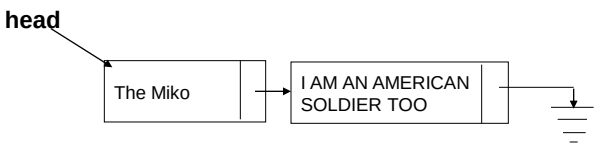
searchName:

isFound:

James Tam

Removing A Node From The List (8)

•Case 2A: Remove first element



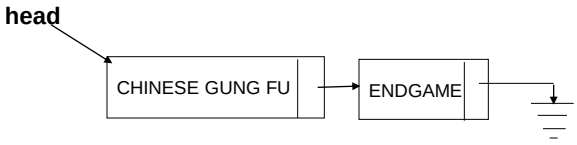
searchName: The Miko

isFound:

James Tam

Removing A Node From The List (9)

•Case 2B: Remove any node except for the first



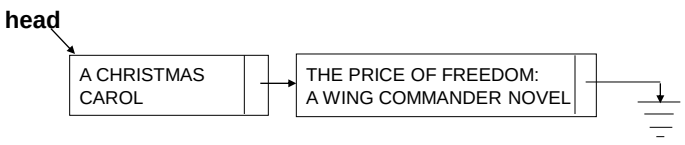
searchName: ENDGAME

isFound:

James Tam

Removing A Node From The List (10)

•Case 3: No match



searchName: MOBY DICK

isFound:

James Tam

Removing A Node From The List

Study guide:

- Main variables:
  1. A temporary reference: refers to the node to be deleted. It is needed so that the program can retain a reference to this node and free up the memory allocated for it after the node has been 'bypassed' (step 4A on the next slides).
  2. A previous reference: refer to the node just prior to the one to be deleted. The 'next' field of this reference will be set to skip over the node to be deleted and will instead point to the node that immediately follows the node to be deleted.
  3. The head reference: The actual reference (and not a copy) is needed if the first node is deleted.
  4. The search key – in this example it is a string but it could be any arbitrary type as long as a comparison can be performed.
  5. A boolean variable that stores that status of the search (the search flag). (Start the search by assuming that it's false and the flag is set to true when a successful match occurs.

James Tam

Removing A Node From The List (2)

- **Steps**

1. Initialize the main variables.
   a) The temporary reference starts at the front of the list.
   b) The boolean flag is set to false (no matches have been found yet).
   c) The previous reference is set to null (to signify that there is no element prior to the first element).
2. If the list is empty (temporary reference is null) display a status message to the user (e.g., "list is empty") and stop the removal process.
3. While the end of the list has not been reached (temporary reference is not null) AND no matches have been found yet (boolean flag is false) :
   a) Compare the search key with the appropriate field in the node referred to by the temporary reference.
   b) If there's a match then set the search flag to true (it's true that a match *has* been found now).
   c) If no match has been found set the previous reference to the node referred to by the temporary reference and move the temporary reference to the next node in the list.

James Tam

Removing A Node From The List (3)

4. (At this point either the whole list has been traversed or there has been successful match and the search has terminated early):
   a. If the search flag is set to true then a match has been found.
      i. If the first node is the one to be deleted (previous reference is null) then set the head reference to the second node in the list.
      ii. If any other node is to be deleted then bypass this node by setting the 'next' field of the node referred to by the previous reference to the node immediately following the node to be deleted.
      iii. In both cases the temporary reference still refers to the node to be deleted. (If applicable) free up the allocated memory using the temporary reference.
   b. If the search flag is set to false no matches have been found, display a status message to the user (e.g., "no matches found").

James Tam

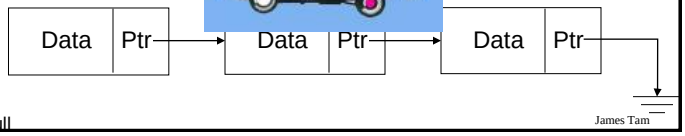
List Operations: Linked Lists (Destroying The Entire List)

- In Java: removing an entire list is similar to how it's done with the array implementation.

```
head = null;
```

- Important reminder: many languages to not employ automatic garbage collection and in those cases each node must be manually de-allocated in memory (step through each element in the list and free up the memory but take care not to lose the connection with the rest of the list).
- Linked list example:

```
public void eraseList ()
{
    head = null;
}
```



James Tam

After This Section You Should Now Know

- What is a data structure
- How a data structure may be defined in Java
- Common list operations
- How a Java array employs dynamic memory allocation
- What is a memory leak
- How the common list operations are implemented using linked lists
- What are the advantages and disadvantages of implementing a list as an array vs. as a linked list

James Tam