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;
```

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

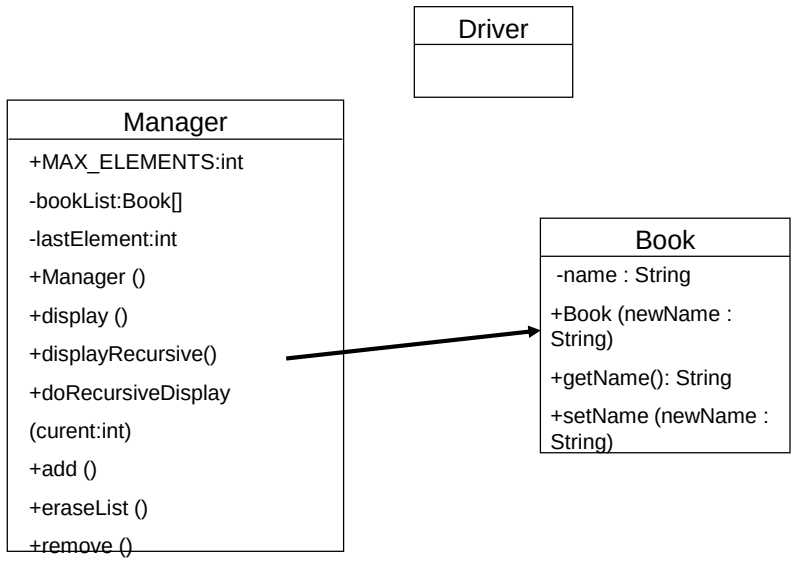
James Tam

List: Array Implementation

- The online example can be found in the directory (under the A2 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)

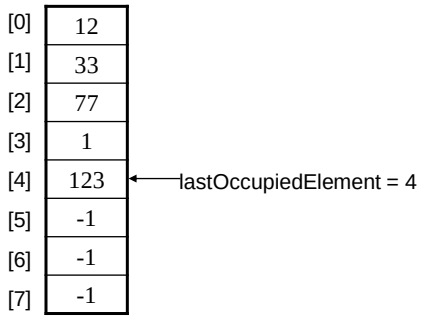
- Unless it can be guaranteed that the list will always be full (unlikely) then some mechanism for determining that the end of the list has been reached is needed.
- If list elements cannot take on certain values then unoccupied list elements can be 'marked' with an invalid value.
- Example: grades (simple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

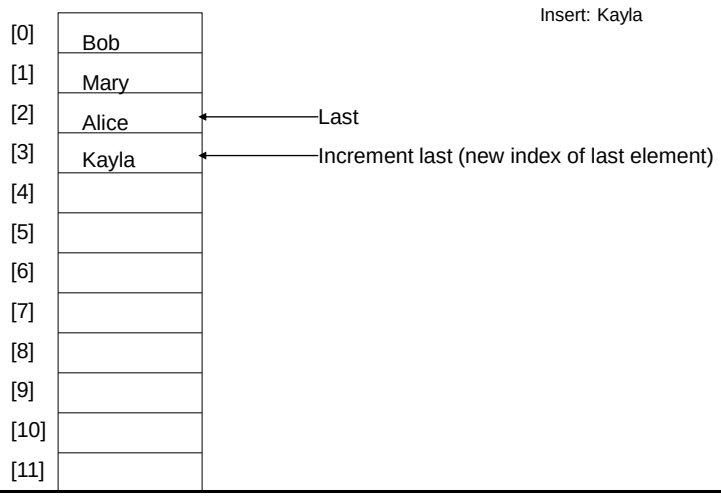
```
// Call in the Driver
Manager aManager = new Manager();

// 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

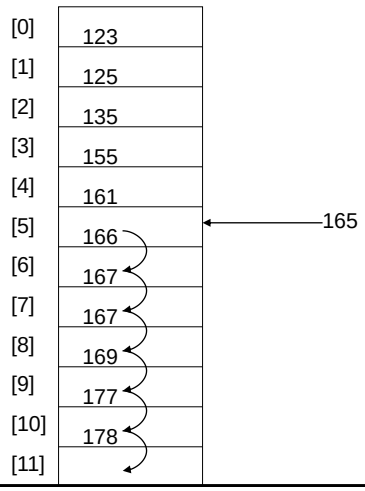
List Operations: Arrays (Insertion At End: 3)

```
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();
```

Not necessary for A2 because it involves recursion

•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:

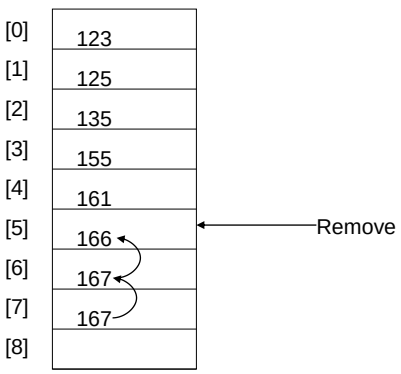
```
public void remove ()
{
    if (lastElement > -1)
    {
        lastElement--;
        System.out.println("Last element removed from list.");
    }
    else
        System.out.println("List is already empty: Nothing to remove");
}
```

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