Revised 27/07/03

Self-Reference - Recursion

Cmput 115 - Lecture 7
Department of Computing Science
University of Alberta

Some code in this lecture is based on code from the book:
Java Structures by Duane A. Bailey or the companion structure package

About This Lecture

- **In this lecture we will learn about self-reference.**
- **This powerful idea allows us to:**
  - **Write self-referencing algorithms.**
  - **Construct self-referencing objects.**
  - **Prove properties about algorithms using mathematical induction.**
- **In this lecture we will focus on self-referencing Algorithms**

Outline

- **Self-reference**
- **Recursive methods**
- **Computing the sum 1 + 2 + … n recursively**
- **Inserting an element in a vector recursively**
- **Stack frames**
- **A trace of recursion**

Self-Reference

- **Self-reference occurs when object refers to itself or more generally to another object from the same class.**
- **Self-reference also occurs when a method (or algorithm) calls itself. Such a method is called a recursive method.**
- **Self-reference also occurs when the proof of a theorem relies on the application of the same theorem to a simpler case. This situation is called mathematical induction.**

Recursive Methods

- **Recursion occurs when a method calls itself, either directly or indirectly.**
- **For recursion to terminate, two conditions must be met:**
  - **there must be one or more simple cases that do not make recursive calls.**
  - **the recursive call must somehow be simpler than the original call so that they lead to the base case.**

To write a recursive function

- **One needs to transform the given problem into a same problem with a smaller size such that the solution can be obtained based on the solution to the smaller problem.**
- **One needs to identify a boundary problem with a simple solution (I.e., it can be solved without recursion.)**

7

Example

- **To write a function to compute factorial(N)**
  - **factorial (N) = factorial(N-1) * N**
  - **Factorial (1) = 1.**

```
public static  int factorial ( int N)  {
  if ( N == 1 )
    return 1;
  else
    return N * function(N-1)
}
```

8

Example

- **To search an object in a list**
  - **find (a, List)**
    - **List = L1 + L2**
    - **Find(a,List) = find(a, L1) or find(a, L2)**
  - **Termination**
    - **Find(a, [a]) = yes,**
    - **Find(a, [b]) = no**

9

Recursive Sum Example

- **Write a method that computes the sum of the integers from 1 to n.**
- **Note that:**

  1 + 2 + … + n = (1 + 2 + … + n-1) + n

```
public static int sum(int n) {
// post: return the sum of ints from 1 to the given value

    if (n < 1)
        return 0;
    else
        return sum(n - 1) + n;
}
```

code based on Bailey pg. 59

10

Vector Element Insertion

- **Recall the iterative implementation:**

```
public void insertElementAt(Object object, int index) {
//pre: 0 <= index <= size()
//post: inserts the given object at the given index,
// moving elements from index to size()-1 to the right

    int i;

    this.ensureCapacity(this.elementCount + 1);
    for (i = this.elementCount; i > index; i--)
        this.elementData[i] = this.elementData[i - 1];
    this.elementData[index] = object;
    this.elementCount++;
}
```

code based on Bailey pg. 39

11

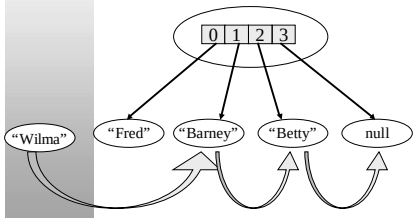
Recursive Element Insertion - 1

- **In a recursive solution:**
  - **The base case is if the insertion location is after the last element (index == size); just add the element using the addElement(Object) method.**
  - **Otherwise, take the element at the index and insert it at the next location (index + 1) using a recursive call; then put the new object at the index location.**

code based on Bailey pg. 60

12

Recursive Element Insertion - 2



code based on Bailey pg. 60

13

Recursive Element Insertion - 3

- **Here is a recursive implementation:**

```
public void insertElementAt(Object object, int index) {
//pre: 0 <= index <= size()
//post: inserts the given object at the given index,
// moving elements from index to size()-1 to the right

     Object previous;

     if (index >= this.size())
           this.addElement(object);
     else {
           previous = this.elementAt(index);
           this.insertElementAt(previous, index + 1);
           setElementAt(object, index);
     }
}
```

code based on Bailey pg. 60

14

Multiple Activations of a Method

- **When we invoke a recursive method on an object, the method becomes active.**
- **Before it is finished, it makes a recursive call to the same method.**
- **This means that when recursion is used, there is more than one copy of the same method active at once.**
- **Therefore, each active method has its own frame which contains independent copies of its direct references.**

15

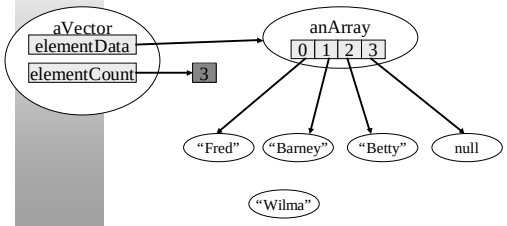
Stack Frames for InsertElementAt

- **Each frame has its own pseudo-variable, `this`, but since the recursive calls all have the same receiver, all "this" variables are bound to the same object.**
- **Each frame has two parameters, object and index.**
- **Each frame has its local variable, previous, bound to a different object.**
- **These frames all exist at the same time.**

16

Recursive Vector Insertion Example

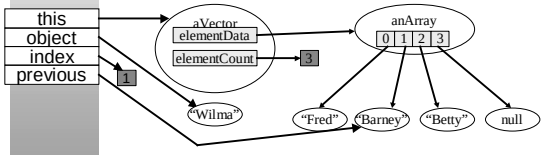
- **For example, insert "Wilma" at index 1.**



17

Calling insertElementAt("Wilma", 1)

```
if (index >= this.size())
      this.addElement(object);
else {
      previous = this.elementAt(index);
      this.insertElementAt(previous, index + 1);
```

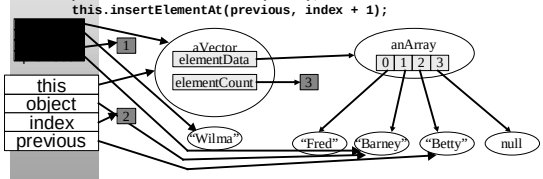


code based on Bailey pg. 60

18

Calling insertElementAt("Barney", 2)

```
if (index >= this.size())
      this.addElement(object);
else {
      previous = this.elementAt(index);
      this.insertElementAt(previous, index + 1);
```

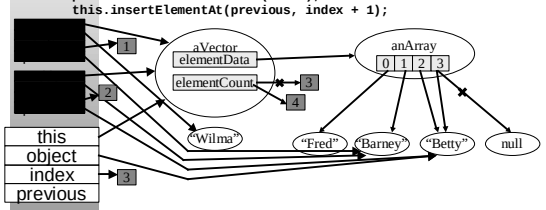


code based on Bailey pg. 60

19

Calling insertElementAt("Betty", 3)

```
if (index >= this.size())
        this.addElement(object);
else {
        previous = this.elementAt(index);
        this.insertElementAt(previous, index + 1);
```

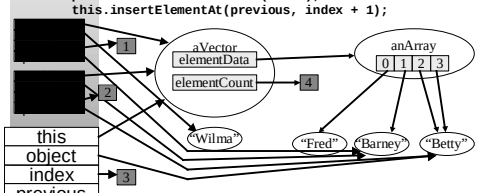


code based on Bailey pg. 60

20

Returning insertElementAt("Betty", 3)

```
if (index >= this.size())
        this.addElement(object);
else {
        previous = this.elementAt(index);
        this.insertElementAt(previous, index + 1);
```

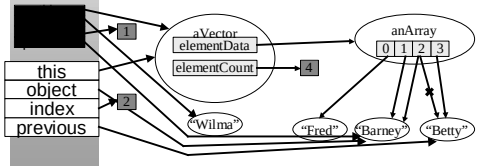


code based on Bailey pg. 60

21

Returning insertElementAt("Barney", 2)

```
if (index >= this.size())
        this.addElement(object);
else {
        previous = this.elementAt(index);
        this.insertElementAt(previous, index + 1);
        setElementAt(object, index);
}
```

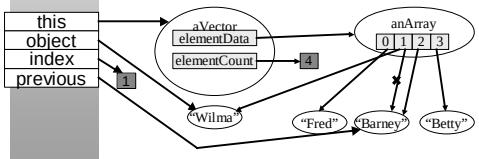


code based on Bailey pg. 60

22

Returning insertElementAt("Wilma", 1)

```
if (index >= this.size())
        this.addElement(object);
else {
        previous = this.elementAt(index);
        this.insertElementAt(previous, index + 1);
        setElementAt(object, index);
}
```



code based on Bailey pg. 60

23

Recursive Vector Insertion Done

- **After, inserting "Wilma" at index 1.**