Linear Lists

Data structures

Spring 2017

Data Objects

data object:

set or collection of instances

integer = {0, +1, -1, +2, -2, +3, -3, …}

daysOfWeek = {S,M,T,W,Th,F,Sa}

Data Object

instances may or may not be related

myDataObject = {apple, chair, 2, 5.2, red, green, Jack}

Data Structure

Data object +

relationships that exist among instances

and elements that comprise an instance

Among instances of integer

$369 < 370$

$280 + 4 = 284$

Data Structure

Among elements that comprise an instance

369

3 is more significant than 6

3 is immediately to the left of 6

9 is immediately to the right of 6

Data Structure

The relationships are usually specified by specifying operations on one or more instances.

add, subtract, predecessor, multiply

Linear (or Ordered) Lists

instances are of the form

$(e_0, e_1, e_2, \ldots, e_{n-1})$

where e_i denotes a list element

$n >= 0$ is finite

list size is n

Linear Lists

$L = (e_0, e_1, e_2, e_3, \ldots, e_{n-1})$

relationships

e_0 is the zero’th (or front) element

e_{n-1} is the last element

e_i immediately precedes e_{i+1}

Linear List Examples/Instances

Days of Week = (S, M, T, W, Th, F, Sa)

Months = (Jan, Feb, Mar, Apr, …, Nov, Dec)

Linear List Operations—size()

determine list size

$$L = (a,b,c,d,e)$$

size = 5

Linear List Operations—get(theIndex)

get element with given index

$$L = (a,b,c,d,e)$$

$get(0) = a$

$get(2) = c$

$get(4) = e$

$get(-1) =$ error

$get(9) =$ error

Linear List Operations—indexOf(theElement)

determine the index of an element

L = (a,b,d,b,a)

indexOf(d) = 2

indexOf(a) = 0

indexOf(z) = -1

Linear List Operations—remove(theIndex)

remove and return element with given index

$L = (a,b,c,d,e,f,g)$

remove(2) returns *c*

and *L* becomes *(a,b,d,e,f,g)*

index of *d,e,f,* and *g* decrease by *1*

Linear List Operations—remove(theIndex)

remove and return element with given index

L = (a,b,c,d,e,f,g)

remove(-1) => error

remove(20) => error

Linear List Operations—add(theIndex, theElement)

add an element so that the new element has a specified index

L = (a,b,c,d,e,f,g)

add(0,h) => L = (h,a,b,c,d,e,f,g)

index of *a,b,c,d,e,f,* and *g* increase by *1*

Linear List Operations—add(theIndex, theElement)

L = (a,b,c,d,e,f,g)

add(2,h) => L = (a,b,h,c,d,e,f,g)

index of *c,d,e,f,* and *g* increase by *1*

add(10,h) => error

add(-6,h) => error

Data Structure Specification

- Language independent
  - Abstract Data Type
- Java
  - Interface
  - Abstract Class

Linear List Abstract Data Type

```
AbstractDataType LinearList
{
  instances
    ordered finite collections of zero or more elements
  operations
    isEmpty(): return  true iff the list is empty,   false otherwise
    size():  return the list size (i.e., number of elements in the list)
    get(index): return the indexth element of the list
    indexOf(x): return the index of the first occurrence of  x in
              the list, return -1 if x is not in the list
    remove(index):  remove and return the indexth element,
        elements with higher index  have their index reduced by 1
    add(theIndex, x): insert x as the indexth element, elements
        with theIndex >= index have their index increased by 1
     output(): output the list elements from left to right
}
```

Linear List as Java Interface

An interface may include constants and abstract methods (i.e., methods for which no implementation is provided).

Linear List as Java Interface

```
public interface LinearList
{
   public boolean isEmpty();
   public int size();
   public Object get(int index);
   public int indexOf(Object elem);
   public Object remove(int index);
   public void add(int index, Object obj);
   public String toString();
}
```

Implementing An Interface

```
public class ArrayLinearList implements LinearList
{
   // code for all LinearList methods must be provided here
}
```

Linear List As An Abstract Class

An abstract class may include constants, variables, abstract methods, and nonabstract methods.

Linear List As Java Abstract Class

```
public abstract class LinearListAsAbstractClass
{
  public abstract boolean isEmpty();
  public abstract int size();
  public abstract Object get(int index);
  public abstract int indexOf(Object theElement);
  public abstract Object remove(int index);
  public abstract void add(int index,
                           Object  theElement);
  public abstract String toString();
}
```

Extending A Java Class

```
public class ArrayLinearList
                extends LinearListAsAbstractClass
{
  // code for all abstract classes must come here
}
```

Implementing Many Interfaces

```
public class MyInteger implements Operable, Zero,
                                  CloneableObject
{
   // code for all methods of Operable, Zero,
   // and CloneableObject must be provided
}
```

Extending Many Classes

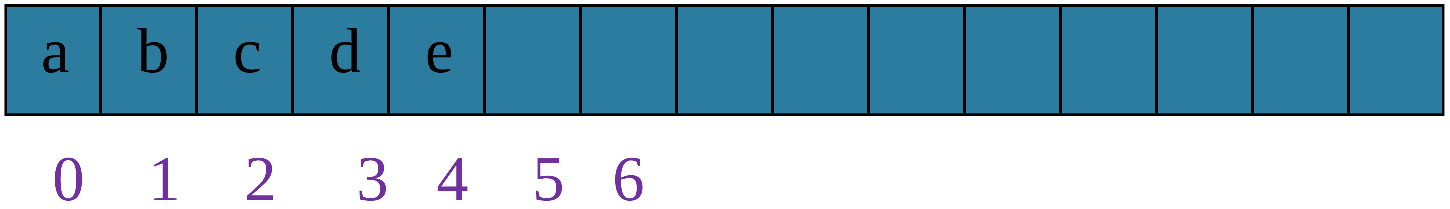
NOT PERMITTED IN JAVA

A Java class may implement as many interfaces as it wants but can extend at most one class.

Linear Lists –
Array Representation

Linear List Array Representation

use a one-dimensional array element[]



a b c d e

0 1 2 3 4 5 6

L = (a, b, c, d, e)

Store element i of list in element[i].

Right To Left Mapping

										e	d	c	b	a

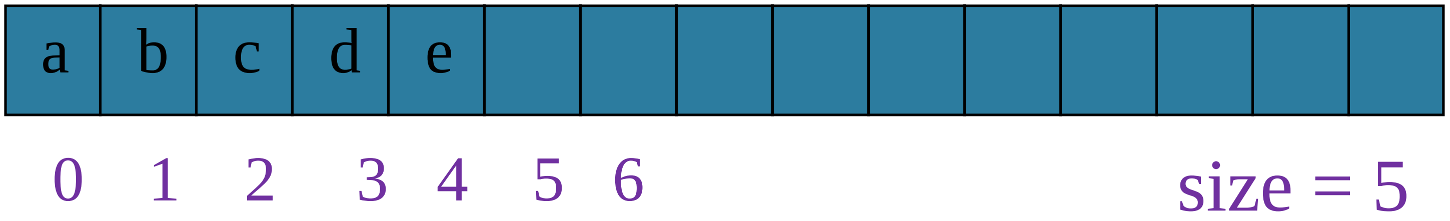
Mapping That Skips Every Other Position

a		b		c		d		e						

Wrap Around Mapping

d	e											a	b	c

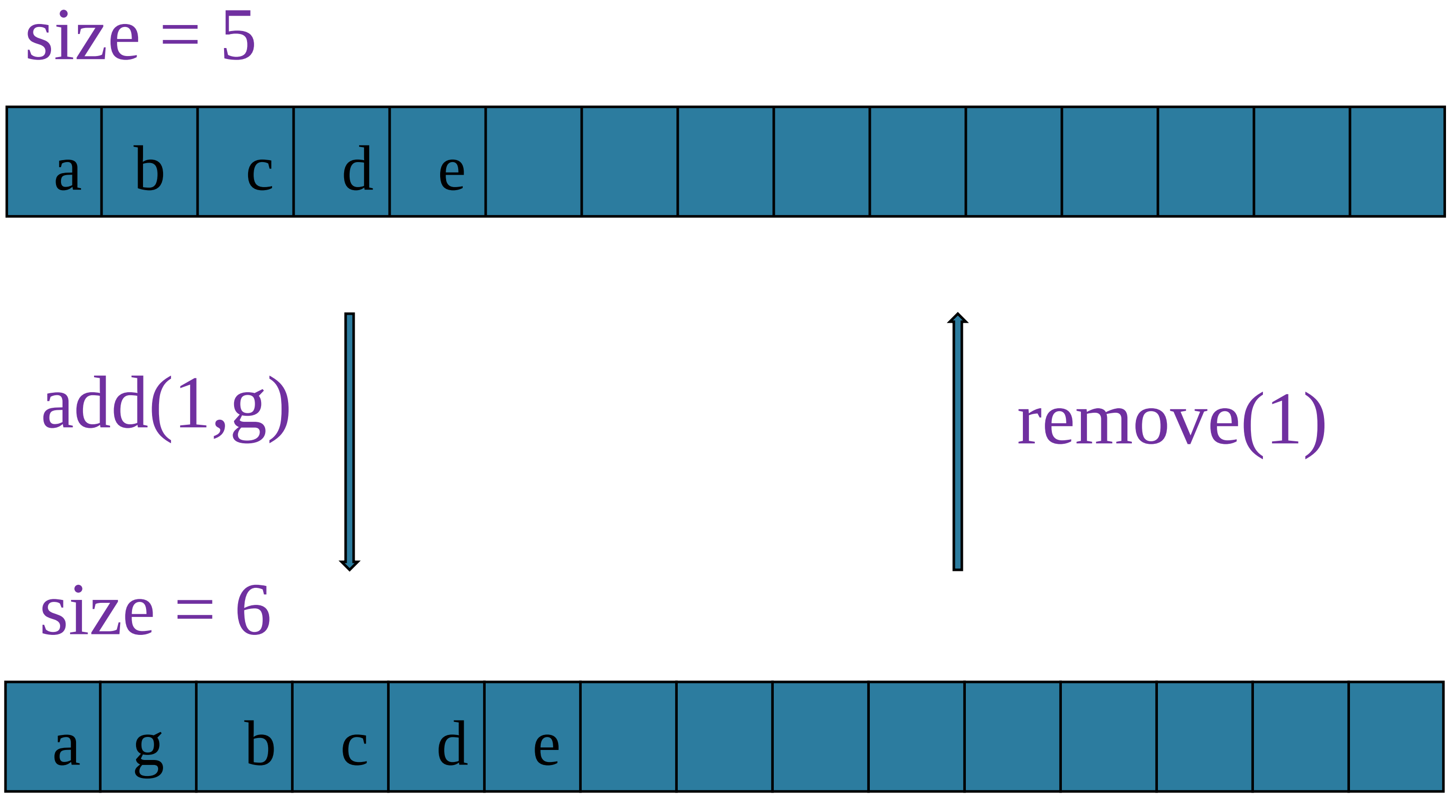
Representation Used In Text



put element i of list in element[i]

use a variable size to record current number of elements

Add/Remove An Element



size = 5

a	b	c	d	e										

add(1,g)

remove(1)

size = 6

a	g	b	c	d	e									

Data Type Of Array element[]

Data type of list elements is unknown.

Define element[] to be of data type Object.

Cannot put elements of primitive data types (int, float, double, char, etc.) into our linear lists.

Length of Array element[]

Don’t know how many elements will be in list.

Must pick an initial length and dynamically increase as needed.

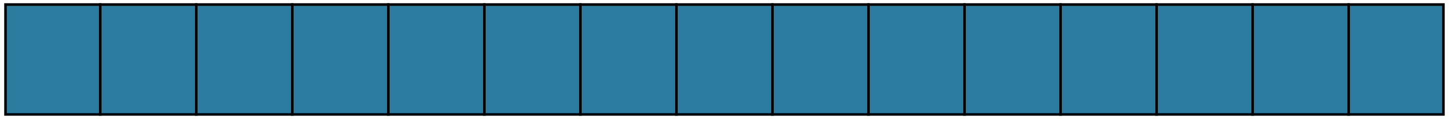
Increasing Array Length

Length of array element[] is 6.

a	b	c	d	e	f

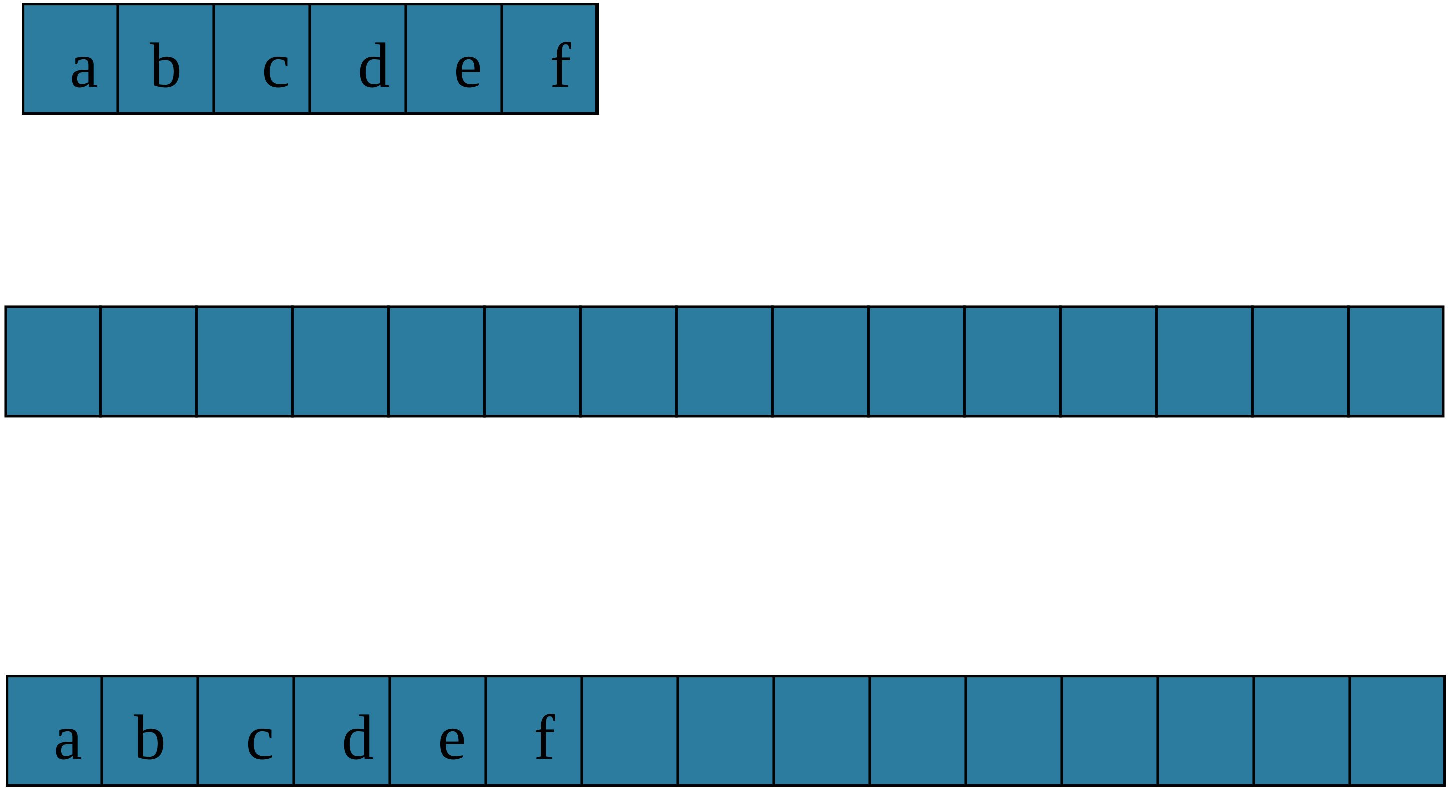
First create a new and larger array

newArray = new Object[15];



Increasing Array Length

Now copy elements from old array to new one.



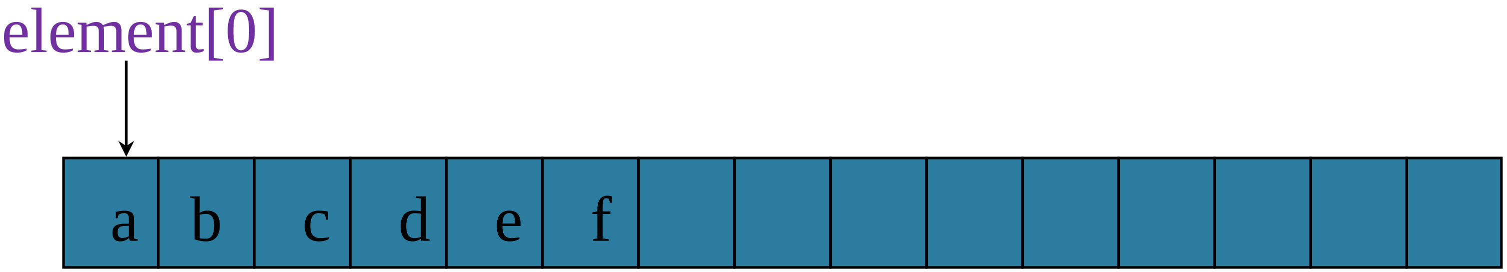
a b c d e f

a b c d e f

Increasing Array Length

Finally, rename new array.

element = newArray;



element[0]

a	b	c	d	e	f									

element.length = 15

Altogether Now

```
// create a new array of proper length and data type
Object [] newArray = new Object [newLength];


// copy all elements from old array into new one
System.arraycopy(element, 0, newArray,
                 0, element.length);

// rename array
element = newArray;
```

How Big Should The New Array Be?

At least 1 more than current array length.

Cost of increasing array length is

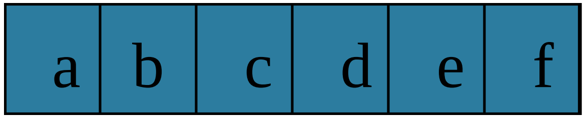
Θ(new length)

Cost of n add operations done on an initially empty linear list is

$\Theta(n^2)$

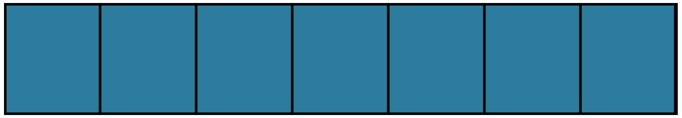
Space Complexity

element[6]



a	b	c	d	e	f

newArray = new char[7];



space needed = 2 * newLength – 1

= 2 * maxListSize – 1

Array Doubling

Double the array length.

a	b	c	d	e	f

newArray = new char[12];

a	b	c	d	e	f						

Time for n adds goes up by Θ(n).

Space needed = 1.5*newLength.

Space needed <= 3*maxListSize – 3

How Big Should The New Array Be?

Resizing by any constant factor

new length = c * old length

increases the cost of n adds by $\Theta(n)$.

Resizing by an additive constant increases the cost of n add operations by $\Theta(n^2)$

How Big Should The New Array Be?

Resizing by any constant factor

$\text{new length} = c * \text{old length}$

requires at most $(1+c) * (\text{maxListSize} - 1)$ space.

Resizing by an additive constant c requires

at most $(\text{maxListSize} - 1) + (\text{maxListSize} - 1 + c)$

$= 2 * (\text{maxListSize} - 1) + c$ space.

What Does Java Do?

java.util.Vector … array doubling

java.util.ArrayList … c = 1.5

dataStructures.ArrayLinearList of text … c = 2