# Data Structures

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

Students in COP3530 =
(Jack, Jill, Abe, Henry, Mary, …, Judy)

Exams in COP3530 =
(exam1, exam2, exam3)

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 *index*th element of the list
    *indexO f(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 *index*th element,
        elements with higher index  have their index reduced by 1
    *add(theIndex, x)*: insert x as the *index*th 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 1 class.

## Data Structures In Text

All but 1 of our data structures are specified as Java interfaces.

Exception is *Graph* in Chapter 17.

Java specifies all of its data structures as interfaces.

*java.util.List*