Ordered List ADT

ADT

- **abstract data type** (ADT) – set of data objects together with a set of operations
- No mention of how the data are stored or how the operations are implemented

- **physical data structure** – the underlying storage mechanism used for the data objects

Ordered List ADT: Add Op

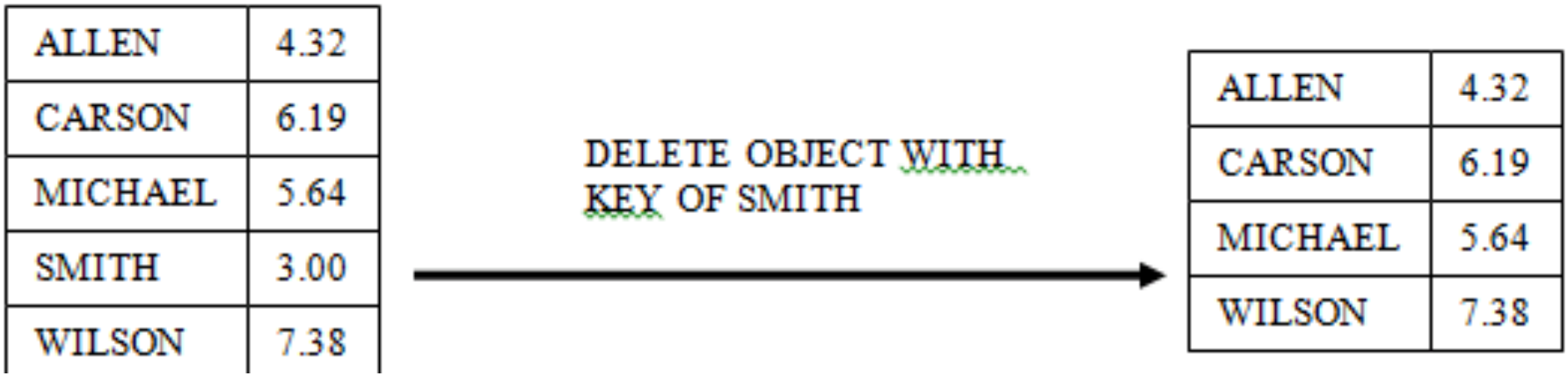
```
class Employee
{
  String name; // Order and retrieve elements by name.
  float   data;// Ordinary data field(s).
}
```

ALLEN	4.32
CARSON	6.19
SMITH	3.00
WILSON	7.38

ALLEN	4.32
CARSON	6.19
MICHAEL	5.64
SMITH	3.00
WILSON	7.38

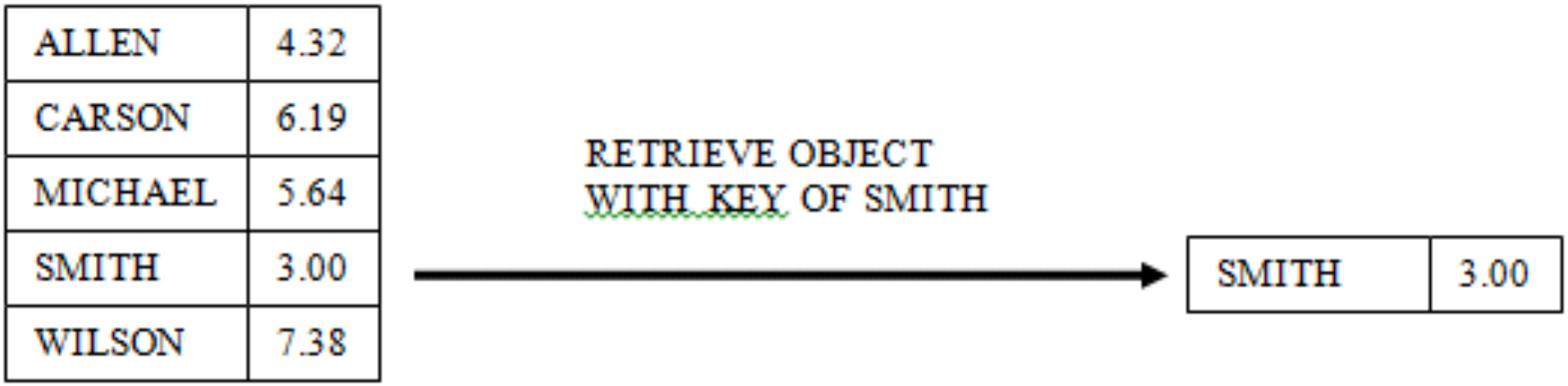
Ordered List ADT: Delete Op

```
class Employee
{
  String name; // Order and retrieve elements by name.
  float   data;// Ordinary data field(s).
}
```



Ordered List ADT: Retrieve Op

```
class Employee
{
  String name; // Order and retrieve elements by name.
  float   data;// Ordinary data field(s).
}
```



Ordered List ADT

Assume **ItemType** is some user-defined object or class type in Java.

Assume **KeyType** is the type of one attribute whose value uniquely identifies an object.

Example: Ticket object, let its number be its key field. Most common policy is to require key field values to be unique.

Example

```
public class Ticket
{
   private int myNumber;
   public Ticket() { myNumber = 0; }
   public Ticket(int R) { myNumber = R; }
   public int getNum() { return myNumber; }
   public String toString() { return “Number “ +
                                      myNumber; }
}
```

Ordered List ADT

public void clear(): Resets the list to the empty state.

public boolean add(Ticket T): Inserts object into the list if its key is unique and the list is not full. Returns true if successful; otherwise, false.

public boolean delete(int keyValue): Deletes the object with the given key. Returns true if the object was found and deleted; else, false.

public Ticket retrieve(int keyValue): Returns the object having the specified key value; otherwise, return null if not found.

public boolean isEmpty(): Returns true if and only if the list is empty.

public boolean isFull(): Returns true if and only if the list is full.

public void print(): Prints all of the objects in the list in ascending order by the key field.

OrderedList Interface

```
public interface TicketOrderedList
{
public void clear();
public boolean add(Ticket C);
public boolean delete(int keyValue);
public Ticket retrieve(int keyValue);
public boolean isEmpty();

public boolean isFull();
public void print();
}
```

Implementation Option 1

- add method does work of inserting objects into sorted place in the array.

- Retrieve method uses _____ search?

Implementation Option 1

- add() method does work of inserting objects into sorted place in the array.

- retrieve() method uses binary search

FATicketOrdredList

```
public class FATicketOrderedList implements
TicketOrderedList
{
  private static final int MAX_SIZE = 100;
  private Ticket[] myList;
  private int myCount; // Number of Tickets stored so far.

  public FATicketOrderedList()
  {
  }
}
```

FATicketOrdredList

```
public class FATicketOrderedList implements TicketOrderedList
{
  private static final int MAX_SIZE = 100;
  private Ticket[] myList;
  private int myCount; // Number of Tickets stored so far.

  public FATicketOrderedList()
  {
    myList = new Ticket[MAX_SIZE];
    myCount = 0;
  }
}
```

FATicketOrdredList

```
/*Inserts Ticket into the list if its key is unique and the list is not full.
  * Returns true if successful; otherwise, false */
public boolean add(Ticket C)
{
  if(isFull()) return false;
  if(retrieve( C.getNum() ) != null) return false;
  if(isEmpty()) // handle insert into empty as special case
    myList[myCount] = C;
  else {
    int i = myCount - 1;
    while(  (i <= 0) && (C.getNum() < myList[i].getNum() )
    {  // shifts existing Ticket from slot i to i+1
     myList[ i + 1 ] = myList[ i ];
     i--;
    }
    myList[i+1] = C;
  }// end else loop to shift and insert.
  myCount++;
  return true;
}
```

FATicketOrdredList

```
public class FATicketOrderedList implements TicketOrderedList
{
   /*Inserts Ticket into the list if its key is unique and the list is not full.
     * Returns true if successful; otherwise, false */
   public boolean add(Ticket C)
   {
     if(isFull() || retrieve(C.getNum()!= null) return false;
     int intoIndex = 0;
     if(myCount > 0)
     {  int x = myCount - 1;
        while(x >= 0 && myList[x].getNum() < C.getNum())
        {
          myList[x+1] = myList[x]; // Shift existing to right
          intoIndex = x;
          x = x - 1;
        }
     }
     myList[intoIndex] = C;
     myCount++;
     return true;
   }
}
```

```
Returns array index of found Ticket; else -1 if not found.
// Access to private Ticket[] myList, int myCount
private int binarySearch(int targetKey)
{
  int low = 0;
  int high = myCount - 1; // Stop at last existing Ticket stored.
  int middle;

  while(low <= high)
  {   // Array index of midway point of [ low-high ]
     middle = (low + high) / 2;
      // == < > work for primitive types. Int, char,
     if ( targetKey == myList[middle].getNum() )
        return middle;
     else if ( targetKey < myList[middle].getNum() )
        high = middle - 1; // Home-in on lower half
      else
        low = middle + 1; // Home-in on upper half
 } // end while loop.
 return -1;
}
```

FATicketOrderedList

```
public class FATicketOrderedList implements TicketOrderedList
{
   /* Deletes the object having the specified key value;
    * otherwise, return false if not found */
   public boolean delete(int key)
   {// Call on private binary search helper method
     int foundAtIndex = binarySearch(key, myList);

     // CASE 1 - Not found
     if(foundAtIndex == -1) return false;
     myList[foundAtIndex] = null;
     myCount--;
     for(int i = foundAtIndex; i < myCount; i++)
     {
       myList[ i ] = myList[ i + 1 ];
     }
     myList[myCount] = null; // Insures nullls after our data

     return true;
   }
}
```

FATicketOrderedList

```
public class FATicketOrderedList implements
TicketOrderedList
{
   /* Returns the object having the specified key value;
    * otherwise, return null if not found */
   public Ticket retrieve(int key)
   {
     int foundAtIndex = binarySearch(key, myList);
     if(foundAtIndex >= 0)
       return myList[foundAtIndex];
     else
       return null;
   }
}
```

FATicketOrderedList

```
public void print()
{
   // Only print those objects that exist in our list.
   // What would happen if we used x < MAX_SIZE?
   for(int x = 0; x < myCount; x++)
      System.out.println(myList[x].toString());
}
```

FATicketOrderedList

```
public boolean isEmpty()
{
   if(myCount == 0) return true;
   else return false;
}
```

Alternately, since myCount == 0 evaluates to a boolean, we can just say return myCount == 0

Comparing String Keys

int x = 5; int y = 5;

if(x == y) // evaluates to true

String x = “cougars”; String y = “cougars”;

if(x == y) // ????

Use Methods to Compare Objects

String x = “cougars”; String y = “cougars”;

if(x.equals(y)) // true, case sensitive compare

String x = “cougars”; String y = “COUGARS”;

if(x.equalsIgnoreCase(y)) // case insensitive

Ordering of Strings

String x = “cougar”; String y = “terrier”;

x.compareTo(y)

Return value is

-1 when x comes before y in dictionary

 0 when x and y are the same word

+1 when x comes after y in dictionary

Case Insensitive Version

String x = “cougar”; String y = “terrier”;

x.compareToIgnoreCase(y)

Return value is

-1 when x comes before y in dictionary

 0 when x and y are the same word

+1 when x comes after y in dictionary

Revise to Compare String Keys

```
Returns array index of found Item; else -1 if not found.
private int binarySearch(String targetKey, Item[] data)
{
  int low = 0;
  int high = data.length - 1;
  int middle;

  while(low <= high)
  {
     middle = (low + high) / 2;
     // Must compare key attributes using String compareTo method…
     if ( targetKey.compareTo( data[middle].getName() ) ????? )

        high = middle - 1; // avoid re-checking middle spot

     else if ( ?????? )

        low = middle + 1;

      else return middle; // found index of key

 } // end while loop.
return -1; // not found, failure return
```